Proceedings 2010

JSC Guest Student Programme on Scientific Computing

Editors
Robert Speck
Mathias Winkel

December 2010

FZJ-JSC-IB-2010-04

Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH
D-52425 Jülich,
Tel. +49 2461 61-6402
http://www.fz-juelich.de/jsc/

FZJ-JSC-IB-2010-04

JSC Guest Student Programme on Scientific Computing
Proceedings 2010

Editorial

One of the main missions of JSC besides providing and operating supercomputer resources, IT tools, methods and know-how for the Forschungszentrum Jülich and for European users through the John von Neumann Institute for Computing is the education and encouragement of young, promising scientists in the broad field of scientific computing. For young academics JSC hosted its traditional ten week guest student programme again during summer 2010. Within this programme students from the natural sciences, engineering, computer science and mathematics had the opportunity to familiarise themselves with different aspects of scientific computing. Mainly supported by staff members of JSC, the participants worked on various topics in computational science including mathematics, physics, biology, hardware and software development tools.

The guest students and their advisers were:

Valentina Banciu (Bucharest)	Oliver Bücker, Ivo Kabadshow
Konstantin Boyanov (Zeuthen)	Willi Homberg
Andreas Breslau (Cologne)	Paul Gibbon
Axel Hübl (Dresden)	Anupam Karmakar
Timo Hülsmann (Wuppertal)	Ulrich Kemloh, Mohcine Chraibi
Alina Georgiana Istrate (Wuppertal)	Godehard Sutmann
Julie Krainau (Berlin)	Sandipan Mohanty, Jan H. Meinke
Martin Licht (Bonn)	Natalie Schröder, Bernd Körfgen
Markus Mayr (Vienna)	Brian Wylie, Zoltan Szebenyi
Yosef Meller (Tel Aviv)	Bernhard Steffen
Marco Müller (Leipzig)	Thomas Neuhaus, Michael Bachmann (IFF)
Elin Solberg (Gothenburg)	Inge Gutheil

The programme started with a training course in parallel programming and an introduction to the usage of the supercomputers in Jülich. In a concluding colloquium the guest students presented and discussed their results, encountered problems and solutions with other students and scientists. This JSC publication containing their individual scientific reports underlines their ability of self-contained, focused and cooperative work as young scientists on up-to-date topics.

We would like to thank the guest students for their work and dedication, contributing to challenging and exciting scientific topics. We would also like to thank the advisers for their cooperation and patient help, not only regarding the student's work. Thanks go to Marc-Andrè Hermanns and Christian Rössel, who organised and held the training course and Ria Schmitz for her tireless work on and against bureaucracy. Our special thanks go to the Verein der Freunde und Förderer des Forschungszentrum Jülich and IBM for providing reliable financial support.

Further information, reviews, former results and the recent announcement of the next programme in 2011 can be found on http://www.fz-juelich.de/jsc/gsp.

Jülich, December 2010 Robert Speck and Mathias Winkel

Contents

Implementation and Evaluation of Integrators for the Fast Multipole Method

Valentina Banciu

University of Bucharest
405 Atomistilor Str.
7000 Bucharest, Romania

E-mail: vbanciu@live.com

Abstract:
Among alternative linear scaling and/or low cost methods, the Fast Multipole Method (FMM) has become the method of choice in applications where one is interested in accurate potentials. In this report, we revisit the method operators and steps, and analyze different particle integrators with respect to accuracy, stability, effectiveness or memory footprint.

---

1 Fundamentals of the Fast Multipole Method

In many problem domains and applications, simulations of physics-based interaction models (e.g. pairwise interactions) result in substantial computational overhead. The evaluation of Coulombic and gravitational interactions in large-scale ensembles of particles is an integral part of the numerical simulation of a large number of physical processes. The N-body problem of electrostatics, for example, requires the evaluation of the Coulomb energy and forces

$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{q_i q_j}{r_{ij}} \quad (i \neq j) \tag{1}$$

$$\mathbf{F}(\mathbf{r}_j) = q_j \sum_{i=1}^{N}\frac{q_i}{r_{ij}^3}\mathbf{r}_{ij} \quad (i \neq j)\,, \tag{2}$$

where q_j are the corresponding charges of particles centered at $\mathbf{r}_j$, and r_{ij} denotes the Euclidean distance between two particles. Equation (1) sums up the Coulomb energy for the N particles; the forces in Equation (2) are evaluated at an equally large number of target locations $\mathbf{r}_j$. For a large ensemble, these evaluations may prove problematic from the computational cost point of view, which scales $\mathcal{O}(N^2)$. Also, because the Coulomb potential is long-range, cut-offs are not possible.

Fast summation schemes can be used in order to overcome the $\mathcal{O}(N^2)$ scaling, e.g. the FMM, which systematically organizes multipole representations of local charge distributions so that each particle interacts with local expansions of the potential due to all distant particles.
The FMM was introduced in 1987 by L. Greengard and V. Rokhlin and has been declared one of the most significant algorithms of the 20^{th} century. It is applicable to any problem involving an $\mathbf{r}^{-n}$ pairwise potential, it conforms to an error bound and it shows linear scaling $\mathcal{O}(N)$.

1.1 Fundamentals

Consider two charges located at positions $\mathbf{r} = (r, \theta, \phi)$ and $\mathbf{a} = (a, \alpha, \beta)$. It turns out convenient to work with spherical coordinates essentially because the operators of the FMM, which are discussed in the next section, are separable in such a frame. The inverse distance between these two charges can also be written as an expansion of the associated Legendre polynomials P_{lm} under the condition $r > a$. We have

$$\frac{1}{d} = \frac{1}{|\mathbf{r} - \mathbf{a}|} = \sum_{l=0}^{\infty} P_l(\cos\gamma)\frac{a^l}{r^{l+1}} \tag{3}$$

$$= \sum_{l=0}^{\infty}\sum_{m=-l}^{l} \frac{(l+|m|)!}{(l-|m|)!}\frac{a^l}{r^{l+1}} P_{lm}(\cos\alpha)P_{lm}(\cos\theta)e^{-im(\beta-\phi)}\,, \tag{4}$$

where γ is the angle subtended between $\mathbf{a}$ and $\mathbf{r}$.
We can then define the moments of a multipole expansion and the coefficients of a Taylor-like expansion, which will respectively define O_{lm} and M_{lm}

$$\omega_{lm} = q \cdot O_{lm} = q a^l \tilde{\tilde{P}}_{lm}(\cos\alpha) \cdot e^{-im\beta} \tag{5}$$

$$\mu_{lm} = q \cdot M_{lm} = q\frac{\tilde{P}_{lm}(\cos\theta)}{r^{l+1}} e^{im\phi}\,, \tag{6}$$

where

$$\tilde{P}_{lm} = \frac{1}{(l+|m|)!}P_{lm} \quad \text{and} \quad \tilde{\tilde{P}}_{lm} = (l-|m|)!P_{lm}\,. \tag{7}$$

We can therefore write the inverse distance as

$$\frac{1}{d} = \sum_{l=0}^{\infty}\sum_{m=-l}^{l} O_{lm} \cdot M_{lm} \tag{8}$$

and can approximate the sum to any given precision by a finite sum. The initial step of the FMM is to represent each particle by its charge q_i and its coordinates (x_i, y_i, z_i). The coordinates are first scaled to fit into a box with coordinate range $[0\ldots1; 0\ldots1; 0\ldots1]$. This simulation box is divided into a set of 8 equal child boxes, each child box being subsequently subdivided into smaller boxes so that a tree is built. The depth of the tree, respectively the number of divisions, is chosen to keep the number of particles in the lowest level box approximately independent from the total number of particles in the simulation box. Figure 1 shows such an hierarchical division of space for one, two and three dimensions.

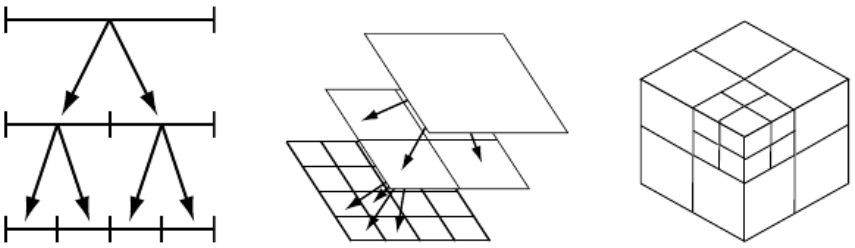


Figure 1: Parent and child boxes in 1D, 2D and 3D representing a binary, quad and oct tree.

1.2 FMM Operators

The FMM needs three operators to perform transformations on the multipole and Taylor-like expansions.
The *Multipole2Multipole* operator shifts multipole expansions from level L to $L-1$ up the tree by

$$\omega_{lm}(a+b) = \sum_{j=0}^{l} \sum_{k=-j}^{j} O_{l-j,m-k}(b)\omega_{jk}(a)\,. \tag{9}$$

The *Multipole2Local* operator transforms remote multipole expansions into local Taylor-like expansions on level L by

$$\mu_{lm}(R-a) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} M_{l+j,m+k}(R)\omega_{jk}(a)\,. \tag{10}$$

Finally, the *Local2Local* operator shifts Taylor-like expansions from level $L-1$ to L down the tree by

$$\mu_{lm}(r-b) = \sum_{j=l}^{p} \sum_{k=-j}^{j} O_{j-l,k-m}(b)\mu_{jk}(r)\,. \tag{11}$$

1.3 FMM Steps

Pass 1: *Form and shift multipole expansions*
After sorting all particle coordinates into the lowest level child boxes, we set up a multipole expansion at the center in each lowest level box. We shift the multipole expansion to the center of the associated parent box using the Multipole2Multipole operator. The translated moments are summed and stored in the parent box. The procedure is continued up to level 3, because on the next level no far field interactions exist. After this step, all children multipole expansions are available in their parent boxes.

Pass 2: *Transform distant multipole expansions*
Interaction between particles via multipole expansions is possible only if the considered boxes are "well separated". All boxes can be enclosed in spheres of radius $\sqrt{3}d$, d being the length of one side of the box, which have the same center as the box. Obviously, such spheres overlap with neighboring boxes, rendering the interaction via multipoles impossible. We convert the distant multipole expansions into Taylor-like expansions about the center of the box B_1, and thus create a local expansion representing all distant boxes; we only transform multipole expansions from boxes at the same level which are well separated from B_1, but are not well separated from B_1's parent box. The remaining child boxes interact via their parent boxes.

Pass 3: *Shift Taylor-like expansions*
The parent's Taylor-like expansion are translated to the center of all child boxes. This top-down shifting is repeated until the lowest level is reached. Now all lowest-level boxes contain the Taylor-like expansion from all "well-separated" boxes.

Pass 4: *Calculate far field energy, forces and potentials*
All Taylor-like epansions are now present at the lowest level. Each lowest level box contains a Taylor-like expansion representing all "well-separated" particles, and by summing up the box energies we get the far field energy.

Pass 5: *Calculate near field energy, forces and potentials*
All neglected neighboring particles which did not contribute to the current box interact directly. The total potential can be calculated by summing up the far- and near- field parts.
The parameters which the FMM depends on are the length of the multipole expansion, the "well-separatedness" criterion and the depth of the FMM tree.

2 Integrators

In molecular dynamics, the atoms and molecules of the simulated system are allowed to interact by approximations of known physics for a given period of time which is divided into time steps. Given the initial set of positions and velocities, we can compute the forces between the particles. Based on the computed values, we calculate the new velocities and the new positions after a time step h. We use this data to recompute the new forces and so on. Figure 2 schematizes the steps of the loop.

The propagation of such a particle system can be described by the temporal evolution of the phase space variables (y, y'). For a nonlinear N-body system, the equations of motion cannot be integrated exactly and one has to rely on numerical integration.
We want to approximate the solution of a differential equation of the type

$$y'(t) = f(t, y(t)) \qquad\qquad y(t_0) = y_0\,, \tag{12}$$

where f is a function that maps $[t_0, \infty) \times \mathbb{R}^d$ to $\mathbb{R}^d$, and the initial condition $y_0 \in \mathbb{R}^d$ is a given vector. The above formulation is called an initial value problem (IVP). Note that a higher-order equation can easily be converted to a system of first-order equations by introducing extra-variables. For example, the second-order equation $y'' = a$ can be rewritten as two first-order equations: $y' = v$ and $v' = a$.
There are different schemes of different orders to advance the coordinates over a step size h, but a good integrator must successfully meet the following requirements [3]:

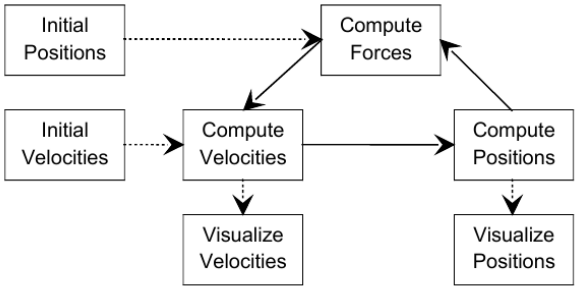


Figure 2: The molecular dynamics simulation loop: Given the initial set of positions and velocities, we compute the forces between the particles. Based on the obtained values, we calculate the new velocities and the new positions after a time step h. We use this data to recompute the new forces and so on.

- *Accuracy*, i.e. the solution of an analytically solvable test problem should be as close as possible to the numerical one,
- *Stability*, i.e. very long simulation runs should produce physically relevant trajectories, independent of numerical artifacts,
- *Conservativity*, there should be no drift or divergence in conserved quantities, like energy or momentum,
- *Reversibility*, i.e. it should have the same temporal structure as the underlying equations,
- *Effectiveness*, i.e. it should allow for large time steps without entering instability and should require a minimum of force evaluations, which usually the most CPU time per time step,
- *Symplecticity*, i.e. the geometrical structure of the phase space should be conserved.

In the following, the mentioned points will be discussed for a number of different integrators.

2.1 Euler Integration

The Euler method is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value. It is the most basic kind of explicit method.
We want to approximate the solution of the given initial system by using the first two terms of the Taylor-like expansion of y, which represents the linear approximation around the point $(t_0, y(t_0))$. One step of the Euler method from t_n to $t_{n+1} = t_n + \Delta t$ is

$$y_{n+1} = y_n + h \cdot f(t_n, y_n)\,. \tag{13}$$

The Euler method is explicit, i.e. the solution y_{n+1} is an explicit function of y_i for $i \leq n$.
The magnitude of the errors arising from the Euler method can be demonstrated by comparison with a Taylor-like expansion of y. If we assume that $f(t)$ and $y(t)$ are known exactly at a time t_0, then the Euler method gives the approximate solution at time $t_0 + h$ as

$$y(t_0 + h) = y(t_0) + h \cdot f(t_0, y(t_0)) = y(t_0) + h \cdot y'(t_0)\,. \tag{14}$$

In comparison, the Taylor-like expansion about t_0 gives

$$y(t_0 + h) = y(t_0) + h \cdot y'(t_0) + \frac{1}{2}h^2 \cdot y''(t_0) + \mathcal{O}(h^3)\,. \tag{15}$$

The error introduced by the Euler method is given by the difference between these equations, and reduces to

$$\frac{1}{2}h^2 y''(t_0) + \mathcal{O}(h^3)\,. \tag{16}$$

For small h, the dominant error per step is proportional to h^2. To solve the problem over a given range of t, the number of steps needed is proportional to $1/h$ so it is to be expected that the total error at the end of the fixed time will be proportional to h. For this reason, the Euler method is said to be first order. This makes the Euler method less accurate for small h than other higher-order techniques such as Runge-Kutta methods.
The Euler method can also be numerically unstable. This limitation - along with its slow convergence of error with h - means that the Euler method is not often used, except as a simple example of numerical integration.

2.2 Verlet Integration

This method was popularized in molecular dynamics by French physicist Loup Verlet in 1967, and is frequently used to calculate trajectories of particles. The Verlet integrator offers greater stability than the simpler Euler method, and also provides time-reversibility and symplecticity. The stability of the technique depends upon either an uniform update rate, or the ability to accurately identify positions at a small time step into the past.
Where Euler's method uses the forward difference approximation to obtain the first derivative in differential equations, Verlet integration can be seen as using the central difference approximation.
First, we develop the Taylor series around $y(t + h)$ and $y(t - h)$

$$y(t + h) = y(t) + h \cdot y'(t) + \frac{h^2}{2}y''(t) + ... \tag{17}$$

$$y(t - h) = y(t) - h \cdot y'(t) + \frac{h^2}{2}y''(t) - ...\,, \tag{18}$$

and adding the equations we have

$$y(t + h) + y(t - h) = 2 \cdot y(t) + h^2 \cdot y''(t) + ... \tag{19}$$

$$y(t + h) = 2 \cdot y(t) - y(t - h) + h^2 \cdot y''(t) + ... \tag{20}$$

At any timestep, we can compute $y'(t)$ by

$$y'(t) = \frac{y(t - h) + y(t + h)}{2h}\,. \tag{21}$$

Figure 3 schematizes the steps of the algorithm. This method also comes in another flavour, *velocity-Verlet*, which is more commonly used.

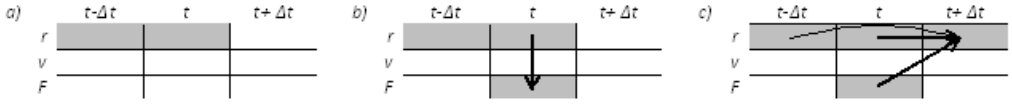


Figure 3: The steps of the Verlet integration: a) as a starting point we need the position of the current and previous time step; b) we compute the forces of the current time step from the current positions; c) we determine the new positions from the current and previous positions and the current forces.

2.3 Velocity Verlet Integration

This uses a similar approach but explicitly incorporates velocities, solving the first-timestep problem with the basic Verlet algorithm

$$y(t+h) = y(t) + h \cdot y'(t) + \frac{h^2}{2m} \cdot F(t) \tag{22}$$

$$y'(t+h) = y'(t) + \frac{h}{2m} \cdot [F(t) + F(t+h)]\,. \tag{23}$$

The force at time t is computed using Equation (2). It can be shown that the error on the velocity Verlet is of the same order as the basic Verlet. The velocity algorithm is not necessarily more memory consuming, because it is not necessary to keep track of the velocity at every timestep during the simulation. The standard implementation scheme of this algorithm is as follows

1. Calculate $y'(t + \frac{1}{2}h) = y'(t) + \frac{1}{2}h \cdot y''(t)$
2. Calculate $y(t + h) = y(t) + h \cdot y'(t + \frac{1}{2}h)$
3. Derive $y''(t + h)$ using $y(t + h)$
4. Calculate $y'(t + h) = y'(t + \frac{1}{2}h) + \frac{1}{2}h \cdot y''(t + h)$

This algorithm assumes that $y''(t + h)$ only depends on $y(t + h)$, and does not depend on $y'(t + h)$. The global error of this method is of order two. Additionally, if the acceleration indeed results from the forces in a conservative mechanical or Hamiltonian system, the energy of the approximation essentially oscillates around the constant energy of the exactly solved system, with a global error bound again of order two. The same holds for all other conserved quantities of the system like linear or angular momentum. Figure 4 schematizes the steps of the algorithm.

2.4 Leapfrog Integration

Leapfrog integration is a simple integration method, very similar to the velocity Verlet scheme. It is equivalent to calculating positions and velocities at time points which are interleaved. For example, the position is known at time step i and the velocity is known at time step $i + 1/2$.
The scheme is a second order method and hence usually works better than Euler integration, which is

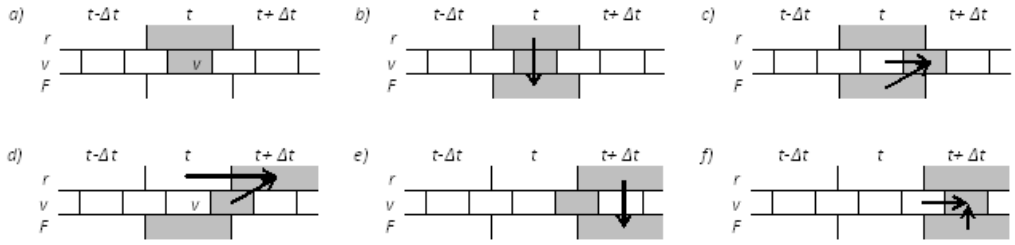


Figure 4: The steps of velocity Verlet integration: a) we know the current positions and velocities; b) from the current positions we can calculate the forces at the current time step; c) from the forces and velocities at the current time step, we can determine the rates for the next half time step; d) we calculate the new positions with the help of the current positions; e) we calculate the forces at the new positions; f) we determine the rates for full time step.

only first order. Unlike Euler integration, it is stable for oscillatory motion, as long as $h < \frac{1}{\omega}$.
The equations for leapfrog integration can be written as

$$y_{i+1} = y_i + y'_{i+1/2} \cdot h \tag{24}$$

$$y'_{i+1/2} = y'_{i-1/2} + y''_i \cdot h\,. \tag{25}$$

Of course, initial conditions are rarely specified at the staggered times required by the leapfrog scheme. Typically, we must use a so-called "self-starting" scheme (e.g. Euler) to take the first half step and establish the value of $y'_{i+1/2}$. On the other hand, the equations can be manipulated into a form which writes velocity at integer steps as

$$y_{i+1} = y_i + y'_i \cdot h + y''_i \frac{h^2}{2} \tag{26}$$

$$y'_{i+1} = y'_i + \frac{y''_i + y''_{i+1}}{2} h\,. \tag{27}$$

This second form usually requires solving the second equation implicitly, because y'' could depend on y'.
The Leapfrog scheme is time reversible. The Euler scheme does not have this property, and we will also see in the next section that neither do Runge-Kutta schemes. Time reversibility is important because it guarantees conservation of energy in many cases. Time-reversible integrators such as the Leapfrog scheme are essential also in situations where we are interested in long-term small changes in the properties of a nearly periodic orbit, and where even small systematic errors would mask the true solution. Figure 5 schematizes the steps of the algorithm.

2.5 Runge-Kutta Integration

The Runge-Kutta methods are an important family of implicit and explicit iterative methods. These techniques were developed around 1900 by the German mathematicians C. Runge and M.W. Kutta.

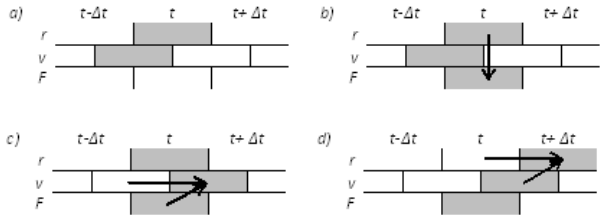


Figure 5: The steps of Leapfrog integration: a) we know the positions and velocities for the last half-time step; b) we compute the forces of the current time step from the current positions; c) we determine the speeds to the next step under the current forces; d) we calculate the new positions for the next half time step and the current positions.

The n-th order explicit Runge-Kutta scheme to advance a set of differential equations $y'(t) = f(t, y(t))$ over a step h can be expressed as

$$y(h) = y_0 + \sum_{j=1}^{n} w_j k_j \tag{28}$$

$$k_j = h \cdot f\left(t_i, y_0 + \sum_{i=1}^{j-1} \beta_{ji} k_i\right) \tag{29}$$

$$\alpha_j = \sum_{i=1}^{j-1} \beta_{ji} \tag{30}$$

$$\sum_{j=1}^{n} w_j = 1\,. \tag{31}$$

2.6 Low-Storage Runge-Kutta Integration

Classical Runge-Kutta integration schemes take up $4N$ storage locations, which turns problematic for sufficiently large particle numbers, while on the other hand low-storage versions of the method only require $2N$ storage locations [4, 5]. The principle is to leave useful information on the register which will receive the contribution $f(y_j)$ instead of starting with an empty one. Thus, the algorithm can be formulated as

$$q_j = a_j q_{j-1} + h \cdot f(t_{j-1}, y_{j-1}) \tag{32}$$

$$y_j = y_{j-1} + b_j q_j\,, \tag{33}$$

with $a_1 = 0$. Successive values of q_j and y_j overwrite the previous ones, so that at any stage the desired storage is achieved.

The parameters a_j and b_j can be expressed in terms of the coefficients in Equations (30), (31) only if these coefficients bear certain ratios, such as

$$b_j = \beta_{j+1,j} \qquad (j \neq n) \tag{34}$$

$$b_n = w_n \tag{35}$$

$$a_j = \frac{w_{j-1} - b_{j-1}}{w_j} \qquad (j \neq 1, w_j \neq 0) \tag{36}$$

$$a_j = \frac{\beta_{j+1,j-1} - \alpha_j}{b_j} \qquad (j \neq 1, w_j = 0)\,. \tag{37}$$

The algorithm adds contributions arising from the earlier velocities into the position vector x, but does not contaminate the velocity registers with contributions from the positions. This feature can be exploited to reduce the effect of round-off errors arising in the x registers.
The round-off error inevitably introduced after j timesteps is

$$e_j = (x_j - x_{j-1}) - b_j q_j\,. \tag{38}$$

This is scaled and added into the q register at the end of each stage so that the algorithm yields

$$q_j = a_j q_{j-1} - \frac{e_{j-1}}{w_j} + h \cdot f(t_{j-1}, y_{j-1})\,, \tag{39}$$

which ensures that the positions at the end of the complete step x_n are independent of e_j , $j < n$, and are only affected by the current round-off error e_n If we note $y'(t) = \lambda \cdot y(t)$, we obtain the stability polynomial of the nondegenerate scheme of order $p \leq 4$ as

$$P(h\lambda) = \sum_{j=0}^{p} \frac{h^j \lambda^j}{j!}\,. \tag{40}$$

It should be noted that all second and third order, and only some fourth order Runge-Kutta algorithms can be implemented as low storage schemes, because of the imposed constraints on parameters a_i and b_i.

3 Conclusion

In this report we have briefly revisited the FMM algorithm, its passes, parameters and operators. The FMM computes e.g. Coulomb forces and the total energy of the system with complexity that scales only $\mathcal{O}(N)$. However, when the charged particles are moving due to the forces that act upon them and their initial conditions we need good integrators in order to keep the computational costs and memory requirements under control. We have implemented and evaluated different integration algorithms. We have discussed a low storage Runge-Kutta scheme which only requires $2N$ storage size and has controlable errors. This scheme accepts second, third and fourth order implementations due to constraints which have been discussed.

Acknowledgements

This work has been carried out during the JSC Guest Student Programme. I would like to thank my advisors, Ivo Kabadshow and Oliver Bücker for their time and help, the organizers, Robert Speck and Mathias Winkel for their reliability and sheer awsomeness, and also my colleagues for many wonderful moments spent both inside and outside the office. I dedicate my effort to my father.

References

1. C. A. White, M. Head-Gordon, J. Chem. Phys. 101 (8), 1994
2. J. Carrier, L. Greengard, V. Rokhlin, SIAM J. Sci. Stat. Comput. 9 (4), 1988
3. G. Sutmann, Multiscale Simulation Methods in Molecular Sciences NIC Series 42 (1-49), 2009
4. J. H. Williamson, Journal of Computational Physics 35 (48-56), 1980
5. M. H. Carpenter, C. A. Kennedy, NASA Technical Memorandum 109112, 2010

Hardware and Software Routing on the QPACE Parallel Computer

Konstantin Boyanov

DESY Zeuthen
Platanenallee 6
D-15738 Zeuthen, Germany

E-mail: konstantin.boyanov@desy.de

Abstract:
The torus network of QPACE, the currently most energy efficient parallel computer in the world, allows up to now only for nearest-neighbor communication. This communication pattern is sufficient for numerical simulations in the field of Quantum Choromodynamics, the initial target application of QPACE. Nevertheless, expanding the torus network functionality to allow any-to-any communication is very important for broadening the spectrum of applications, which can take advantage of QPACE's high-performance, low-energy parallel architecture. Possible extensions to the custom designed Torus Network (TNW) are considered and a simple and low-overhead routing algorithm is proposed. Furthermore, the proposed algorithm is implemented and tested in the OMNeT++ event-based simulation environment. We show that the simulation model implemented is a good representation of the real hardware, and we also test and verify the algorithm implementation using communication patterns as they occur during matrix transposition.

1 Introduction

The QPACE (Quantum Chromodynamics Parallel Computing on the Cell) [1] tightly coupled parallel computer was designed and developed to provide a new generation of a massively parallel and scalable computer architecture, optimized for the use in the field of numerical simulations, especially for lattice Quantum Chromodynamics. Quantum Chromodynamics is the theory of strong interactions, i.e. the force that acts on the quarks and gluons, which form the building blocks of matter – the protons and neutrons. Studying these interactions requires a very large amount of compute-intensive operations. Therefore a high utilization of the underlying hardware is very important. This is the main reason why the QCD community is one of the few research communities that develop and operate custom-build parallel machines. Previous machines include the QCDOC [2] and apeNEXT [3].

As the computational power requirements stemming from QCD applications demand almost full utilization of the processor, and further development of custom ASICs becomes more complicated and costly, a different approach was taken in QPACE - a high-performance commodity processor, the IBM PowerXCell 8i [4] was used. The processors on each node card are interconnected with each other

through a high-speed, low-latency custom 3D torus network implemented on an FPGA circuit. It is autonomous enough to handle communication with no great intervention from the CPU and thus not hinder computation. Further, QPACE incorporates a novel liquid-cooling system [5], which allows for high mounting density and high energy efficiency. Together, the high-performance Cell processor, the custom network and the novel cooling subsystem, contribute to the fact that the QPACE parallel computer is at place one in the Green500 list [6], with an energy efficiency of 773 MFlops per Watt.

The direct torus network however, allows only for nearest-neighbor communication among nodes in QPACE. Although this is not a limiting factor for QCD applications, it would be very useful to extend this functionality, so other applications which require any-to-any communication patterns, could also profit from the performance and high-energy efficiency of the QPACE architecture. That is why we examine options to extend the communication functionality of the torus network, to allow routing of messages between not directly connected nodes. First we describe in Section 2 the architecture of the QPACE parallel computer in more detail, particularly the Network Processor (NWP) on which the torus network (TNW) is implemented. Then, in Section 3 we examine general routing techniques common to communication networks in high performance computing. After considering their advantages and disadvantages, we propose a routing algorithm with minimal hardware and software overhead, suitable for the QPACE architecture in Section 4. In order to implement and test the proposed routing algorithm, we use an event-based simulation environment, which is described together with the QPACE simulation model in Section 5. There, we present how the simulation model was verified and also introduce some initial results on bandwidth measurements at single node for a test application. At the end, in Section 6 we give a conclusion and an outlook for further developments.

2 Architecture of the QPACE parallel computer

State-of-the-art IBM PowerXCell 8i commodity multicore processors with one main and 8 assisting cores are used as processing units in the QPACE parallel machine. A custom designed network interconnect is directly attached to the Cell processor and implements the networking logic.

The QPACE parallel computer consists of backplanes which can hold up to 32 node cards. On each of the node cards there is a single PowerXCell 8i Processor and a Network Processor realized on Field Programmable Gate Array (FPGA) for interconnect between the node cards. QPACE backplanes are mounted in racks, whereas each rack can house up to eight backplanes, giving a total maximum of 256 node cards (2048 SPU cores) per rack.

Figure 1 shows a QPACE node card and its components. The components of interest here are the ones on the right hand-side of the node card, namely the network processor and the six PHY devices, one for each direction in the 3D-torus, surrounding the FPGA chip of the Network Processor (NWP). In QPACE there are several different networks that provide different vital communication facilities for the operation of the parallel machine. These include the following:

- High-bandwidth, low-latency nearest neighbor torus network, providing the communication between neighboring nodes in a 3D-torus communication topology.

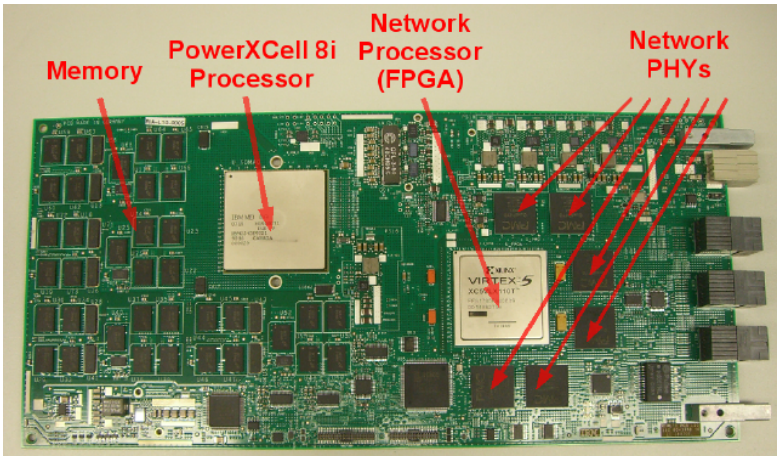


Figure 1: QPACE node card housing the PowerXCell 8i processor and memory modules (4GB) on the left-hand side and the network processor and PHY devices for connectivity in the six spatial dimensions of the 3D-torus network topology.

- Ethernet network, interconnecting all the nodes and the root cards to the front-end systems, used for user I/O as well as common login services, administrative tasks like booting, updates etc.
- A fast two-wire global network with a tree topology, which serves for transmitting global signals and interrupts and also is being utilized for synchronization purposes.

For the topic of routing only the implementation of the torus network is of interest. This is considered in detail in the next section.

2.1 Torus Network Processor

The Network Processor was implemented for this purpose on an FPGA (Virtex-5 LX110T by Xilinx) chip. It acts as an I/O fabric similar to the interconnect chips between processor, peripherals and network interfaces (also known as south bridges), found on off-the-shelf PC products. The only difference in the design here is that the I/O fabric is directly coupled to the processor and not through a PCIe bus. With the help of the NWP a 3D-torus nearest-neighbor communication network is realized, which allows for direct memory access-like communication between the Synergetic Processing Unit (SPE) cores on neighboring processor nodes. This implementation facet has the advantage that data does not have to be transported through the main memory and thus avoids the overhead associated with accessing the memory interface.

The internal structure of the NWP is schematically represented in Figure 2. The parts of the application logic are as follows: first a proprietary IBM implementation of the interface from the NWP towards the PowerXCell 8i can be seen in the top of the picture. This incorporates two 8-bit wide bidirectional (full-duplex) high-speed links with a bandwidth of up to 4 Gbytes/sec per direction between the Cell processor and the NWP. Below this logic block the Master and Slave interface logic blocks are located. They serve to merely interface the IBM proprietary logic implementation and the Torus Network logic through the Inbound Write Controller (IWC) and the Outbound-Write Controller (OWC). The IWC deals with (inbound-)data, that comes from the Cell processor, while the OWC deals with (outbound-

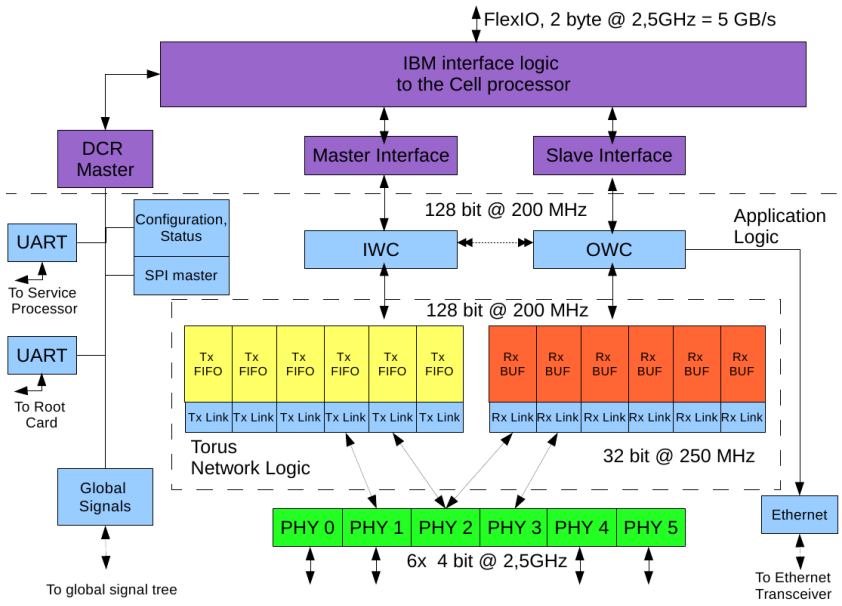


Figure 2: Schematic overview of the Torus Network Processor.

)data, coming from remote nodes. The OWC comprises an arbiter to give multiple incoming links access to the link to the processor. The logic implementing the actual Torus Links are attached to the IWC and OWC. Downwards the torus links interface with six full-duplex bidirectional 10-Gigabit Medium Independent Interfaces (XGMII) Ethernet PHY devices.

For each physical link there exists a Transmit FIFO and a Transmit link (Tx link) logic for data leaving the node, and a Receive buffer and a Receive link (Rx link) logic for data coming from a neighboring node. The transmit FIFOs of virtual channels on one physical link share a 16KB buffer space. In order to allow simultaneous communication between different remote cores along the same physical link, there are 8 virtual channels per physical link implemented. Each virtual channel has at its disposal a 2KB FIFO for sending packets. On each Tx link packets are immediately transmitted in the order in which they have arrived in the transmit FIFO.

The Receive buffer (Rx buffer) does not work like a FIFO buffer however. The outbound data packets (packets that come from an neighboring node and are designated to some other SPEs local storage) are being saved on addresses, which are calculated from three distinct parts, or values. One part of the address information is incorporated in the incoming data packet (remote offset) another is the default configuration of a base address in memory, and the last is a credit for particular memory address for the particular node (local offset) where the packet should land. In order to prevent data corruption by overwriting the incoming packet has to be kept into the Rx buffer, until an authorization, or credit, for a particular destination address and message size arrives at the Receive Buffer logic.

2.2 TNW Communication Protocol

A two-sided communication model is realized between two nodes in the QPACE torus topology. One reason for this is to minimize overhead caused by handshaking mechanisms between sender and receiver. If for example node A sends data to node B, than it has to rely on the fact that node B is issuing a receive operation.

Messages in the QPACE torus network must have a length which is a multiple of 128 bytes. Source and destination addresses must also be aligned to 128 bytes in order to avoid overhead caused by segmentation of data in DMA transactions between the SPE and the NWP. On the physical layer the messages are split into packets with 128 byte payload, 4 byte header and 4 byte CRC checksum.

The path of a message consisting of multiple packets from a source SPU to a destination SPU involves multiple transactions, where the sender pushes data towards the receiver. For every one of this transactions the integrity of the data must be assured. In order to guarantee that, first no data should be lost. If for example no buffer space is available at the receiver side, then simply all incoming packets are not being acknowledged until there is enough buffer space available. In this manner data loss is prevented by simply not allowing more write operations to occur.

3 General Routing techniques

In this section some general terminology regarding routing in direct networks is presented. Four common routing algorithms are considered for appliance in the QPACE architecture and their advantages and disadvantages are examined.

3.1 Static vs. Dynamic Routing

One of the first properties that divide routing techniques in two broad groups is how the information used for making the routing decisions is created and maintained. In this regard we distinguish between static routing techniques, where the description of routes through the network is created before the communication system becomes functional and is kept constant during operation. Static routing thus describes only a set of all possible routes between nodes and packets traveling between two nodes always take the same way.

This is however not the case with dynamic routing techniques, where the routing information is dynamically changed throughout the operation of the communication system. Such changes reflect the availability of connections or intermediate routing devices, as well as contention information. Thus messages traveling in a dynamically routed network can take different routes between the same two nodes.

3.2 Avoiding Deadlock and Livelock

When routed through a particular network, packets can block each other in circular fashion on intermediate routing nodes by occupying buffer space or other resources (like credits, tags, etc.). Many approaches exist which help avoid the conditions leading to deadlocks. In the sense of network routing, these are first making sure that enough resources (buffers, connections, etc) are available at any point in the operation of the communication network. If such abundant resources are not available, or their introduction into the architecture of the communication system is not feasible, another approach is to reduce the message size passing through intermediate nodes on its way to the destination node. One can also always offload the routing decisions from the hardware and introduce a routing fully implemented in software, which will have the needed deadlock prevention techniques implemented.

Another issue regarding routing of messages in networks is the so-called livelock. This refers to the situation where a packet or a set of packets run forever in circular motion around the destination node and never reaching it. This can be the case in highly loaded dynamically routed networks, where messages get constantly misrouted due to resource contention on intermediate nodes and unavailability of connection links towards the destination.

It is thus one of the main design task for routing algorithms to make sure that they are not allowing for situations where either deadlock or livelock can occur.

3.3 Common Routing Algorithms

In this section we take a look at several routing algorithms, common for direct networks in the high-performance computing domains.

In the first one, Store-and-Forward Routing [7], each router on the path between two distant nodes receives the whole message (multiple packets) and examines it. Based on this examination the router then makes a decision to which port to forward the message. This routing algorithm is not very well suited for on-chip networks because with increased message size or a larger number of messages, the router will need more on-chip buffers. Another disadvantage is the increase of point to point transmission delay, as every packet has to be completely stored on every routing layer (or hop/node).

Virtual cut-through [9, 15] routing tries to eliminate the disadvantages of store-and-forward routing by allowing the packet to be examined in chunks. In this way only small portions of the packet need to be stored and can be forwarded without the whole packet being present in the router's buffers. Major disadvantage of this approach is however, that if the next router in the path is not available (due to contention or failure) the whole packet must still be held in the internal buffers so it can be thoroughly examined and a new route can be defined.

In wormhole routing [10], each packet is segmented in so-called flits (short for flow control units). The header flit reserves the routing channel for upcoming flits of the same packet (the body) and the tale flit frees the reservation. Major advantage of wormhole routing is that it does not require for the whole packet to be stored in router's internal buffers while waiting for the header flit to be routed to the next stage in the route. This solves the problems present in the two previously mentioned techniques while

reducing buffer space requirements and point-to-point delays. Although a major disadvantage here is that during one message transmission the whole chain of communication links is reserved for only one communication. Other messages or packets must wait and even compete for the use of newly freed communication links. This introduces additional complexity and increases end-to-end transmission times, especially for larger messages (more packets).

In the last considered routing algorithm, the so-called Hot-Potato Routing, or deflection routing [11], a completely different approach is taken. Here, there is no buffer space foreseen on intermediate nodes for to-be-routed packets whatsoever. When a packet arrives at an input port of the routing node, it is instantly forwarded to one of the output ports of the node. The packets of a message have encoded in their header which ports along the possible paths towards their destination are most preferred to them, and the intermediate routing nodes try to forward them according to this information. If the wanted output port is not available however, or many packets compete for one and the same output port, then only one gets it while the other is misrouted to the next available direction. If the network is highly loaded, then it can be the case that packets are misrouted a lot, and even a livelock situation may occur. Thus the algorithm does not assure that the packets travel along the shortest between the source and destination nodes.

4 Proposed Routing Algorithm

There are several things that should be taken into account when considering a routing algorithm for the QPACE architecture. As we saw in the previous section, popular routing techniques are not so convenient for this architecture for various reasons. These include larger buffer spaces, packets not always following the shortest paths between two remote nodes, communication channels can be blocked for the use of only one message transmission and thus hinder the sending of other messages, etc.

Further, we want to keep the hardware changes required for the routing algorithm to function, as small as possible, because of limited availability of logic blocks on the FPGA chip.

Another important issue is that packets of two or more messages do not get mixed. In the present hardware design packet ordering rules are insured only within the boundaries of a single message traveling along a single physical link and single virtual channel. This ordering cannot be insured by dynamic routing algorithms for multiple messages using the same link and channel, without adding a lot of overhead information to the packets and thus decreasing bandwidth and increasing latency. Thus we consider only static routing algorithms which will decrease routing information overhead by using only fixed paths among nodes. In the next sections the proposed routing algorithm is presented in detail.

4.1 Addressing

The addresses of each of the nodes in the 3D torus topology configuration are represented by their coordinates in a Cartesian 3D space. As already mentioned, messages in QPACE are composed of equally sized packets of 128 byte size. Each message has a header of 4 bytes attached to it, which contains information used for calculating the address to which the packet/message should be written

to in the destination SPE local storage. For the proposed routing algorithm we will extend this packet header to contain information regarding the offset of the source (sender) and destination (receiver) in the Cartesian coordinates of the 3D torus topology.

For example communication between node A (source) and node B(destination), which have the following coordinates, or addresses $(A_x, A_y, A_z) = (1, 2, 1), (B_x, B_y, B_z) = (0, 2, 2)$. The information contained in the header will be then the difference of the destination address and the source address: $(\delta_x, \delta_y, \delta_z) = (B_x, B_y, B_z) - (A_x, A_y, A_z) = (-1, 0, 1)$. The so-called delta vector then describes how many hops there should be made in each spatial direction. The absolute value of the vector entries describes how many hops should be made along the corresponding axis, while the sign of the entries gives the direction along the axis. In the example above the packet should travel one hop along the -X axis, and one hop along the +Z axis.

At each hop the corresponding value in the header is decremented or incremented, according to where the packet is routed to. When arriving at the destination, all the entries of the delta vector must be zero. The rules behind that are described in the following section.

4.2 Routing Rules

Depending on the direction (Rx buffer) the packet has been received from, and also according to the values of the delta vector, a decision is made, to which of the transmit FIFOs it must be redirected for further transmission towards its destination. The rules are described in Table 1 and are schematically presented in Figure 3.

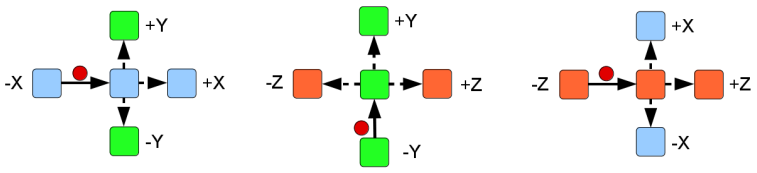


Figure 3: If a packet arrives at the receive link corresponding to -X direction it can be forwarded further along the X axis (being sent to the transmit FIFO corresponding to +X direction), or make a turn on in Y-direction.

The table shows that there are only 3 options available for packets to leave an intermediate node, depending on which link they arrived. We choose to only allow routing in one plane, because if we otherwise allow a packet to leave the intermediate node on all 5 available links, then we will be creating excessive connections between the receive and transmit buffers in the actual hardware implementation.

Another important issue here is that at the source node the sending direction must be chosen carefully in order for the message not to get stuck on some intermediate node without further forward routes. This could be the case if a packet only has to be routed along X and Z axes. If it travels first on the X axis, at the Rx buffer of some intermediate node it will not have any possibility to continue on the Z axis according to Table 1.

+X	-X, +Y, -Y
-X	+X, +Y, -Y
+Y	-Y, +Z, -Z
-Y	+Y, +Z, -Z
+Z	-Z, +X, -X
-Z	+Z, +X, -X

Table 1: The table describing the possible forward directions for a packet (right column), according to that, from where the packet came from (left column).

Besides a reduced number of additional connections between Tx and Rx buffers/FIFOs, the algorithm does not require any additional buffers on both send and receive links. We believe that with the simple logic and the absence of complex routing tables will allow for minimal and more importantly not complex hardware changes. The changes related to the proposed routing algorithm are presented in Figure 4.

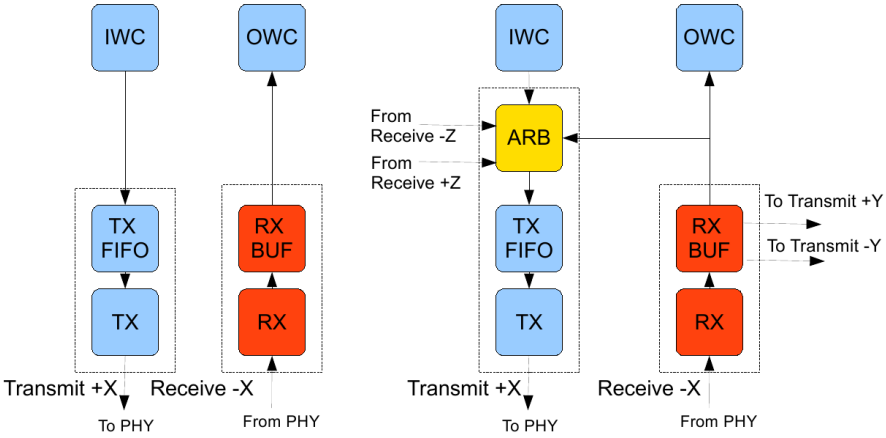


Figure 4: Changes to the hardware architecture of the torus network processor in regard to routing support. On the left-hand side is a scheme how a send/receive pair looks like now. On the right hand-side the proposed changes are depicted.

First, the main difference is the introduction of an arbiter on top of the transmit FIFO, which will arbitrate writing to its input port. This is necessary, because in the new architecture, there will be three additional links towards the transmit FIFO of each link coming from the corresponding receive buffers according to Table 1. Further, in the receive buffer the logic for examining the packet header routing information should be implemented, as well as the decision where to forward the packet to. Other parts of the network processor remain unchanged.

4.3 Deadlock Freedom

When considering this important property of the routing algorithm, it comes out that the routing algorithm is not deadlock free. Let us consider the following situation: The receive buffer of some node A in the QPACE network is filled with packets destined to another node waiting to be further forwarded.

In this case no further packets can travel along this link, even packets with node A as final destination might be blocked. If this condition occurs on all nodes simultaneously, the communication system deadlocks and no further communication can take place, because all send and receive buffers will wait on each other in a circular fashion.

As discussed in Section 3.2, there are strategies to prevent deadlock situations. In the context of the proposed routing algorithm, one of them is giving higher priority to packets within the network than packets injected into the network. This means that packets belonging to messages already traveling through the network will be preferred at the arbiter in Figure 4 before packets coming from the CPU to leave the node through this particular transmit link. Although sounding reasonable, this approach will not bring the communication network out of a deadlock situation if it has already occurred (consider the example above) – even with higher priority, the packets coming from the receive buffer through the transmit FIFO cannot proceed because on the remote receive buffer will be blocked. Another solution is a limited message size thus no receive buffer gets full. This will require that one knows all the possible paths going through a particular receive/transmit buffer. This number grows very large with increasing network size, and if we were to decrease the message size accordingly, then inconveniently small message sizes could occur, which will make the network practically unusable. To delegate the routing decision to software running on the CPU will be yet another option. Here we have the advantage that the deadlock freedom of existing hardware is preserved, but increased waiting times on intermediate nodes for the processor to make the routing decision will affect the performance of the network negatively.

Although the routing algorithm is not deadlock-free, we will consider it further, because of its simplicity and advantages, and also because we believe it is a very good starting point for further investigations and insights. The way we model the real hardware, implement the algorithm, test and verify it, is described in the next sections.

5 Implementation

We used the OMNeT++ (Objective Modular Network Testbed in C++) [13] event-based simulation environment for a test bed for the implementation of a model of the QPACE network processor and torus network, as well as the implementation and testing of the proposed routing algorithm.

In an event-based simulation the system is represented as chronological sequence of events, which are processed by the simulation environment. Every event marks a change in the state of the system. This kind of simulation is very suitable for testing new functionality of complex system without physically altering it, which is also very important for trying out new features of the QPACE network.

As the exact one-to-one modeling of the real hardware in the simulation environment is impossible and not necessary, we have omitted some features of the network processor which we consider not relevant to routing. First, there is the absence of virtual channels allowing messages from different SPE core to travel along the same physical link between two nodes. In real hardware there are eight virtual channels per link or direction. We consider this not relevant to routing, as it will not affect sending and forwarding packets substantially. Message transmission over a link without further decomposing it in virtual channels is sufficient for the considered communication patterns

There is also no feedback between receive and transmit link for acknowledgment of received packets or informing the transmit link that certain packets were lost or damaged. We also included contention information and flow control among neighboring nodes, so that transmit links are blocked if the receive buffer on the other side is currently full or otherwise busy and cannot accept further packets. Nevertheless, we implemented a backpressure mechanism between the Cell processor and the slave interface (sender side) for not allowing the slave interface to become overloaded.

5.1 Verification of the QPACE Model

For verifying that the simulation model reproduces the functionality of real hardware in a satisfactory way, a comparison of bandwidth measurements obtained with micro benchmarks on real hardware was made. In the micro benchmark the bandwidth is measured in a point-to-point communication over a single physical link. The sender sends one message at a time, that is, the sender posts a send operation waits for the acknowledgment from the receive side and then posts another send operation. The bandwidth is obtained as a function of the message size varying from 128 to 2048 bytes. For each message size thousand consecutive messages are sent. The functionality of the micro benchmark was reproduced in the simulation environment by choosing the simulation parameters to match the micro-benchmark behavior on real hardware. The two curves of both measurements are presented in Figure 5.

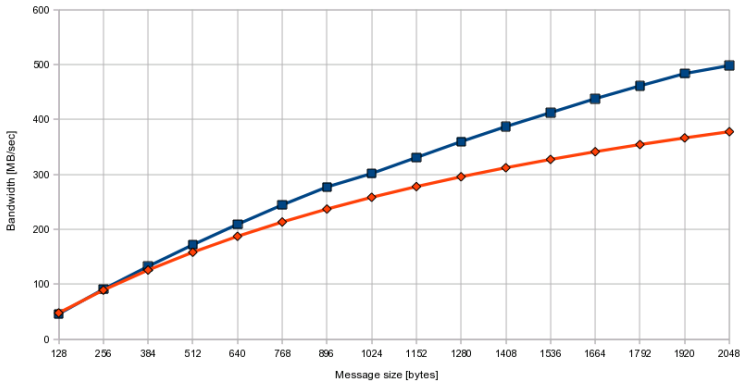


Figure 5: Bandwidth comparison for peer-to-peer communication. The rhomboids represent the bandwidth measured in the model, the squares – the one measured in real hardware. Along the X-axis are the different messages sizes used from 128 bytes to 2KBytes (in 128 byte step).

For testing the functionality of the routing algorithm we used a communication pattern from an example application, namely transposing large quadratic matrices. This communication pattern is common when a two dimensional data set should be processed first along one of the dimensions, then along the other. Examples of such algorithms are row-column Fast Fourier Transform algorithms [14]. The naive algorithm describing the transposition of the matrices has the following steps:

1. Every node i is assigned a set of rows in the matrix
2. Every node i sends parts of its row to every other node j, i $\neq$ j
3. Every node i receives parts of all other nodes' rows

In the end every node sends messages to any other node, and receives from any other node. Tests of the model were successfully passed using up to 1024 nodes - all nodes received the right amount of packets (according to matrix rows allocation to nodes), all packets came at the right destination and no packets were lost or got into a livelock.

6 Conclusion and Outlook

We have considered an important extension to the functionality of the QPACE parallel computer's Network Processor to support routing and thus any-to-any communication patterns. Common routing algorithms were examined and their advantages and disadvantages were taken into account. Based on the knowledge gained and also on goals set by limited hardware resources and complexity requirements we came up whit a low-overhead routing algorithm, which uses simple routing logic and requires minimal changes to the existing hardware architecture. To test and verify the algorithm we implemented a simulation model describing the real hardware in consistent fashion. We verified the model by comparing bandwidth measures with such on real hardware and gained sufficient results and good overlapping. Furthermore we implemented the new routing algorithm in the simulation environment and tested it successfully with the help of a common application and its communication pattern.

We have observed some interesting results in regard to bandwidth utilization when using the transpose matrix communication pattern - it does not fully utilize link bandwidth and thus more precise and comprehensive measurements must be made. In the future further measures interesting to the field of routing, like end-to-end packet transmission delay and bandwidth measures at a reference receiver node as function of torus size and packet count, are to be made.

References

1. http://hpc.desy.de/qpace
2. http://en.wikipedia.org/wiki/QCDOC
3. http://hpc.desy.de/ape/
4. https://www-01.ibm.com/chips/techlib/techlib.nsf/products/PowerXCell_8i
5. G. Goldrian et al. - "QPACE: power-efficient parallel architecture based on IBM PowerXCell 8i", 2010
6. http://www.green500.org/lists/2010/06/top/list.php
7. "Introduction to Store-and-forward routing" - http://www.springerlink.com/content/932322hx9j134585/
8. http://en.wikipedia.org/wiki/Cut_through
9. Terry Tao Ye, Luca Benini, Giovanni De Micheli - "Packetization and Routing Analysis of On-Chip Multiprocessor Networks"
10. http://en.wikipedia.org/wiki/Wormhole_switching
11. Erik Demaine, Sampalli Srinivas - "Routing Algorithms on static interconnection networks: A classification scheme"
12. http://en.wikipedia.org/wiki/Deflection_routing
13. http://www.omnetpp.org/
14. http://en.wikipedia.org/wiki/Fast_Fourier_transform#Multidimensional_FFTs
15. M. Blumrich, D. Chen, P. Coteus - "Design and Analysis of the BlueGene/L Torus Interconnection Network"

Development of a Parallel, Tree-based Neighbour-search Algorithm

Andreas Breslau

Forschungszentrum Jülich
Institute for Advanced Simulation
Jülich Supercomputing Centre
Wilhelm-Johnen-Straße
52428 Jülich

E-mail: a.breslau@fz-juelich.de

Abstract:
In Astrophysics it is quite common to use a combination of an N-body code and an SPH code for the computation of self-gravitating matter. SPH is a Lagrangian method where the particles are used as the discrete elements within a fluid description. Thermodynamic properties are computed at the simulation points from averages over neighbouring particles. To do this it is necessary to know the next neighbours of each particle.
In this article the implementation of a neighbour search algorithm using the tree-code PEPC is described. The algorithm is based on the existing routine for the force summation, adapted to return neighbour lists instead of multipoles. The correctness of the parallel neighbour search is verified using both visual and quantitative tests. The scaling of the algorithm with particle and process number is shown to be $\mathcal{O}(N \log N)$ or better.

1 Introduction

Many problems in Astrophysics require knowledge of the dynamics of huge amounts of self-gravitating matter. But for the simulation of self-gravitating gas it is not sufficient to just fill the simulation-box with N particles each with $1/N$ of the total mass and simulate it as an N-body problem. In this case only the attracting forces would be taken into account. But for gas also the repulsing forces resulting from the thermal movements of the molecules are relevant. While the gravitational part can be computed with a tree-code like PEPC [1] the thermodynamic forces have to be computed with a fluid-code. A natural and well tested technique to combine these is to use Smoothed Particle Hydrodynamics (SPH) [2] code for the fluid-computation [3, 4]. SPH computes the thermodynamic forces from averages over the neighbouring particles, which must first be determined via a search algorithm. For this one could apply a mesh to the simulation-area but for highly clustered matter this would be very inefficient. Since tree-codes are inherently adaptive, it is natural to try to exploit their built-in data structure to find these

neighbours. This article will describe the implementation of a next-neighbour search algorithm using the parallel tree-code PEPC.

2 Short introduction to SPH

Smoothed Particle Hydrodynamics (SPH) is a Lagrangian method for fluid-computation. It was first described by Monaghan and Gingold (1977) and Lucy (1977) [5, 6].

It is based on the fact that using the delta-distribution one can write

$$\int f(x')\delta(x_0 - x')dx' = f(x_0). \tag{1}$$

Now for a function W with the property

$$\lim_{h\to 0} W(x,h) = \delta(x), \tag{2}$$

one can write

$$A(\vec{r}) = \lim_{h\to 0} \int A(\vec{r}\,')W(\vec{r} - \vec{r}\,', h)d\vec{r}\,'. \tag{3}$$

For a small enough h one can simply write

$$A(\vec{r}) \approx \int A(\vec{r}\,')W(\vec{r} - \vec{r}\,', h)d\vec{r}\,', \tag{4}$$

or with $\rho(\vec{r}) = dm/d\vec{r}$

$$A(\vec{r}) = \int \frac{A(\vec{r}\,')}{\rho(\vec{r}\,')}W(\vec{r} - \vec{r}\,', h)dm'. \tag{5}$$

Discretisation finally leads to

$$A_a = \sum_b \frac{m_b}{\rho_b} A_b W(\vec{r_a} - \vec{r_b}, h). \tag{6}$$

From Eq. (6) one can see that a given property A_a of a particle a can be computed by summing up the properties A_b of the particles b weighted with their masses m_b and densities ρ_b and the function W.

For example the total force F on particle a can be computed as

$$F_a = F_g - m_a \sum_b m_b(\frac{P_b}{\rho_b^2} + \frac{P_a}{\rho_a^2})\nabla W(\vec{r_a} - \vec{r_b}, h), \tag{7}$$

where F_g is the gravitational force and P the pressure obtained from the ideal gas law $P \propto \rho\, T$.

3 About PEPC

PEPC is a so called hashed-oct-tree-based tree-code following the implementation described by Warren and Salmon in 1995 [7]. A tree-code is usually used for simulating N-body problems. For the simulation of N bodies interacting for example through gravity for each body $N - 1$ forces have to be computed, which leads to an $\mathcal{O}(N^2)$ run-time. This limits the manageable total number of particles on today's fastest supercomputers to a few $100\,000$ due to limited computation resources.

To simulate more particles a better-scaling method like a tree-code is needed. This code uses a tree as a data-structure to store and access information about the particles. To build the tree the whole simulation-box is identified with the root of the tree and then the box is consecutively subdivided into eight smaller child boxes, which are linked with the parent box. If one box contains only one particle the subdivision is stopped and the particle is linked to this box. Then for each box multipole moments of the potential are computed.

Now for the force computation particle-multipole interactions are taken into account rather than particle-particle interactions. To decide which multipole to use for the interaction a criterion is constructed based on the distance between particle and multipole and the size of the box associated with the multipole. So only for close interactions highly resolved tree-nodes are used, for distant interactions a few big tree-nodes are used.

This limits the computation effort for the force-summation to an $\mathcal{O}(N \log N)$ run-time.

4 Implementation of the next neighbour search algorithm

The implemented algorithm follows the idea described by Warren and Salmon in 1995 [7].

The first step was the implementation of a function to find neighbours within a given radius around the particle i.

Like in the function creating the interaction lists for the force summation, the neighbour search algorithm walks through the tree starting at the root, descending level by level. At each node it tests for an overlap between the search-sphere with radius r around the particle i and the box represented by the tested node. In case of an overlap the node is resolved, otherwise ignored. When reaching the leaf-level in case of an overlap also the separation between the particle attached to this particular leaf-node and the particle i is tested for being smaller than r. If so, the particle is added on the next neighbour list of particle i.

The parallel version of this algorithm has also to check whether the child-nodes of the tested tree-nodes are locally present, and if not, fetch them from their corresponding owner process.

```
search_neighbours_of_particle_i(r) {

  walk through tree from root to leaves
    1) particles within r put on next neighbour list
    2) ignore nodes/particles outside r
    3) for nodes with overlap with r
       if children locally present, resolve
       else get children from remote process
  end
}
```

Having verified this algorithm, a second version for obtaining *exactly* N_{nn} next neighbours of particle i was implemented. This was done using a modified version of the first one searching neighbour particles inside a fixed individual search radius r_i. After all particles in this sphere are found, it is tested whether at least the desired number of neighbours N_{nn} is found. If not, the individual search radius is increased for this particle and the search is started again.

When for all particles a minimum number of neighbours N_{nn} are found, a bubble-sort loop is used to successively move the furthest neighbour to the end of the list, which is then shortened by one, until exactly N_{nn} neighbours are left. Then, this bubble-sort loop is executed one more time to again move the furthest neighbour to the end of the list. The separation between this furthest neighbour and particle i is used as the new individual search radius for particle i during the next iteration.

```
while there are particles with less than N_nn found next neighbours
  search_neighbours_of_particle_i(r_i) {

    walk through tree from root to leaves
      1) particles within r_i put on next neighbour list
      2) ignore nodes/particles outside r_i
      3) for nodes with overlap with r_i
         if children locally present, resolve
         else get children from remote process
    end
  }

  if found next neighbours < N_nn
    increase r_i for this particle
    put particle on list to search neighbours again
  end
end

for all particles
  while n_found > N_nn
    move furthest particle to the end of the list
    shorten list by one
  end

  move furthest particle to the end of the list
  new search radius r_i is 1.1 * distance to this particle
end
```

5 Validation

The first validation approach was to visualize the list of neighbours with a appropriate tool such as ParaView for 3D setups or GLE for 2D setups and check the results by hand. But this was only possible for a random sample and so not sufficient to verify that the algorithm works correct in all cases. An example plot can be seen in Fig. 1.

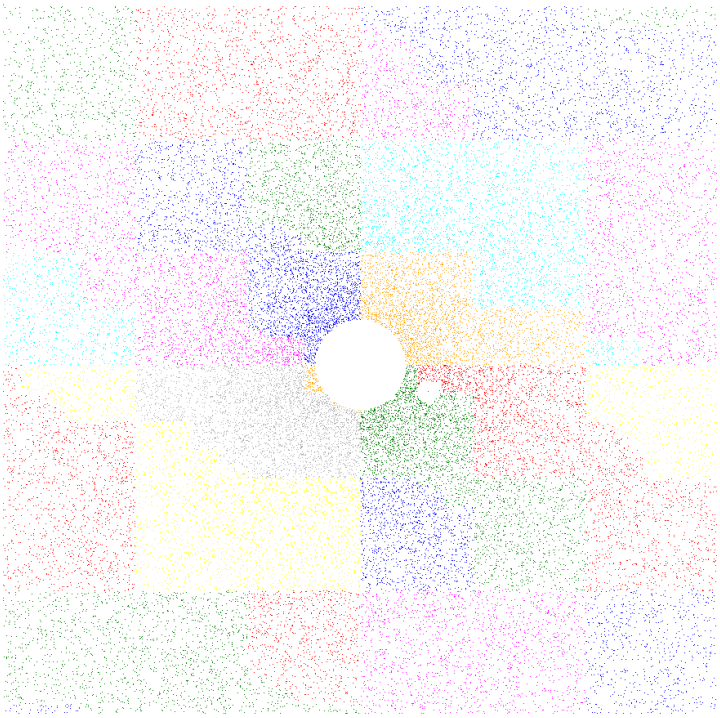


Figure 1: Visual validation of the results of the next neighbour search with 100 000 particles distributed in a two dimensional disc with a r^{-2} density run with 32 processes. This is a zoom on the inner part. The different colours represent the process domains. In the big white circle in the center are no particles because of the singularity of the r^{-2} density. In the other white circle the next neighbours of the particle in the center of this circle are over-plotted in white to have a better contrast.

A more rigorous validation tool was implemented in Perl to compute all $N - 1$ distances between one particle and all others. After sorting this list the N_{nn} next neighbours can be obtained from this list as the first N_{nn}. Since it was not possible to obtain a list of the labels of the next neighbours for one particle the validation tool outputs the coordinates of the neighbours, which are compared to the coordinates of the next neighbours written out by the neighbour search routine.

Because the coordinates are written out with a limited precision it can happen in case of high particle densities that among the furthest particles there are discrepancies when computed by the Perl-script and the neighbour search routine. This has to be borne in mind when checking to high densities with the validation tool.

6 Scaling

6.1 An analytical estimate

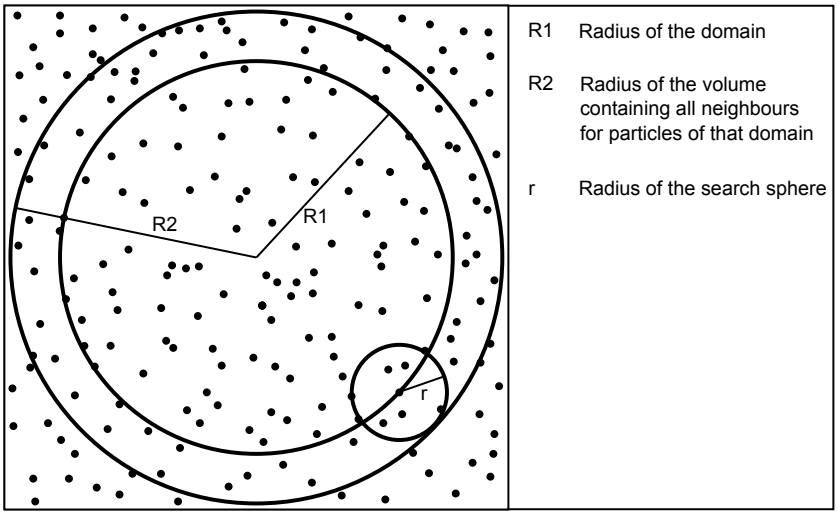


Figure 2: Sketch for the estimation. The inner big circle with the radius $R1$ represents one process domain. If there are many particles near the domain border the search spheres with radius r of these particles overlap and fill the area between the two bigger circles. All particles in this area have to be fetched from other processes.

Assuming that the density is constant within the whole simulation volume, one can estimate the number of particles to be fetched by one process. A total number of particles N in the volume V result in a mean-density of $\rho = N/V$. If simulated on p processes the average volume of one domain is $V_p = V/p$. Assuming that the domain is spherical its radius can be expressed as

$$R_{domain} = \left(\frac{3\pi V_p}{4}\right)^{\frac{1}{3}}. \tag{8}$$

With the mean-density the radius of the sphere containing all N_{nn} next neighbours of a particle can be written as

$$r_{search} = \left(\frac{3N_{nn}}{4\pi\rho}\right)^{\frac{1}{3}}. \tag{9}$$

As shown in Fig. 2 the search spheres of all particles close to the domain border form a shell around the domain, in which next neighbours of particles from the domain can be found. All these particles

have to be fetched during the next neighbour search and stored in the local memory. Their number can be calculated as

$$N_{fetch} = \frac{4}{3}\pi\rho\left[(R+r)^3 - R^3\right] \tag{10}$$

$$= \left(27\frac{N_{nn}N^2}{p^2} + 27\frac{N_{nn}^2 N}{p} + N_{nn}^3\right)^{\frac{1}{3}}. \tag{11}$$

It is easy to see that this leads to the asymptotic complexities

$$\mathcal{O}(N_{nn}), \quad \mathcal{O}(N^{2/3}), \quad \mathcal{O}(p^{-2/3}). \tag{12}$$

In case of a fixed number of particles per process N_p one finds

$$N_{fetch} = \left(27N_{nn}N_p^2 + 27N_{nn}^2N_p + N_{nn}^3\right)^{\frac{1}{3}}, \tag{13}$$

which leads to the complexities

$$\mathcal{O}(N_{nn}), \quad \mathcal{O}(N_p^{2/3}). \tag{14}$$

But this is only for the communication effort and the memory space needed locally to store the fetched particles. Additionally requested memory can lead to problems when simulating with too few particles per process, because the size of the locally allocated memory is initially calculated depending on the number of local particles. As shown in Tab. 1 the relative additionally needed memory increases with decreasing particles per process. This can lead to out of memory problems even though the memory is almost unused.

N_{nn}	N_p	N_{fetch}	$N_{fetch}[\%]$
50	10000	6000	61
50	50000	17000	33
50	200000	40000	20

Table 1: Estimated number of particles to fetch from remote processes depending on the number of particles per process.

6.2 Benchmarks

For benchmarking the new algorithm several sets of time measurements have been performed each with one configuration parameter varying. In each case the time needed by the neighbour search algorithm for the local walk ('nn walk'), the time to fetch data from other processes ('nn fetch') and the time needed by the rest of the program ('rest') was measured.

- For the weak scaling the number of particles per process was fixed to $150\,000$ and the number of neighbours required set to 50. The number of processes was varied from 80 to 800, which results in 1.2×10^7 to 1.2×10^8 particles in total. For the log-log plot of the absolute run-time

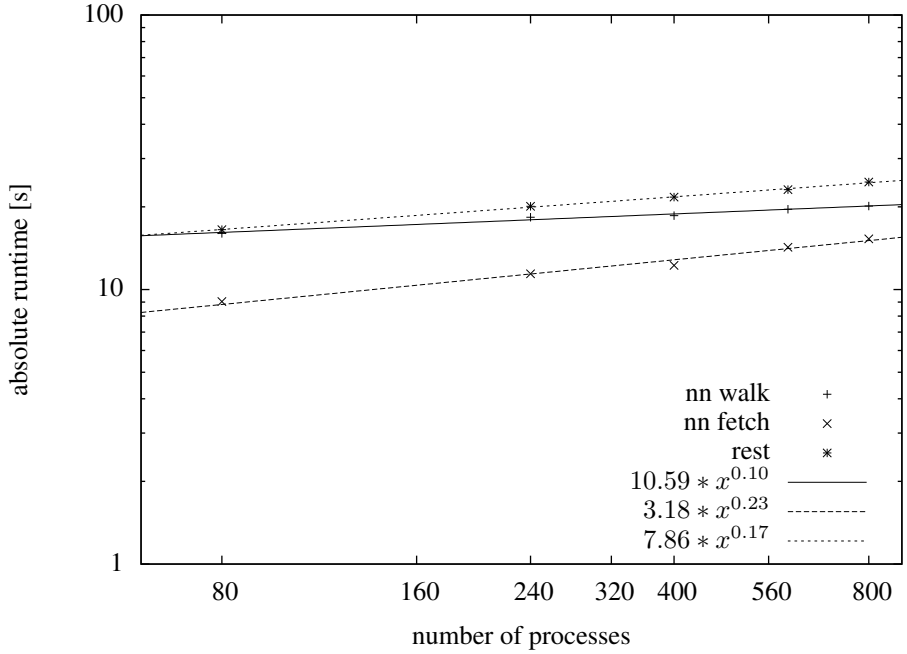


Figure 3: Weak scaling on Juropa using $150\,000$ particles per process and 50 next neighbours to find.

see Fig. 3. Since the computational effort per process is fixed the weak scaling should be almost constant. The slight increase in run-time of the 'nn walk' seen in the figure can be caused by a fuller hash-table due to the higher total number of particles.

- For the strong scaling the total number of particles was fixed to 1.2×10^7 and the number of neighbours set to 50. The number of processes was varied from 80 to 400, which results in $30\,000$ to $150\,000$ particles per process. For the log-log plot of the absolute run-time see Fig. 4. Ideal strong scaling means that if the number of processes is multiplied by a factor a the run-time is divided by a, which results in a p^{-1} dependence of the run-time, where p is the number of processes. As can be seen in the figure the local next neighbour search ('nn walk') scales almost ideal with a $p^{-0.96}$ dependence. The 'nn fetch', which is the time for the next neighbour communication, scales like $p^{-0.79}$. This is a close to the estimated $p^{-2/3}$, see Eq. (12).

- For the N scaling the number of processes was fixed to 240 and the number of neighbours set to 50. The total number of particles was varied from 6×10^6 to 3.6×10^7, which results in $25\,000$ to $150\,000$ particles per process. For the log-log plot of the absolute run-time see Fig. 5. The fitted functions show that all three measured program parts scale like $N \log N$, which is in excellent agreement with the expectations.

- For the N_{nn} scaling the number of particles per process was fixed to $50\,000$ and the number of processes set to 80. The number of next neighbours to find was varied from 25 to 150. For the log-log plot of the absolute run-time see Fig. 6. The increase in run-time for the rest of the program is probably again due to a fuller hash-table.

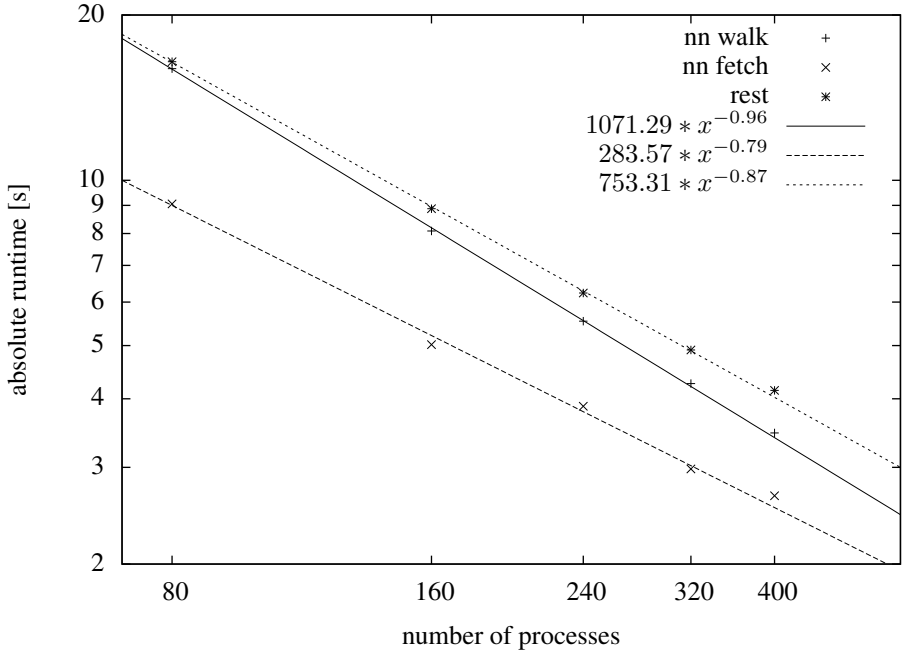


Figure 4: Strong scaling on Juropa using 12×10^6 particles and 50 next neighbours to find.

7 Summary and Outlook

A parallel tree-based neighbour search algorithm was successfully implemented and validated. The validation tool will be useful to demonstrate the correctness of improved algorithms implemented in the future.
The scaling is at least as good as the scaling of rest of PEPC, which is dominated by the $\mathcal{O}(N \log N)$ scaling of the force summation. Currently the absolute time consumption is relatively high (around 50% of the total iteration time). This is probably due to inefficiencies in the local part of the search algorithm but it should be possible to improve this. Further it would be interesting to measure the imbalance between the domains and implement a domain decomposition taking the next neighbour search work load into account.
If this is still not fast enough one could compute the next neighbour lists less frequently to reduce the search overhead.
And finally for a better energy conservation in SPH, a modified neighbour search algorithm with a more complex distance criterion has to be implemented.

Acknowledgments

The work presented here was done during the Guest Student Programme 2010 organized by the JSC, Forschungszentrum Jülich. I want to thank all the people behind this programme especially the organizers Robert Speck and Mathias Winkel for making it possible. Furthermore I want to express my gratitude to my adviser Dr. Paul Gibbon and the members of his group for their advise and sup-

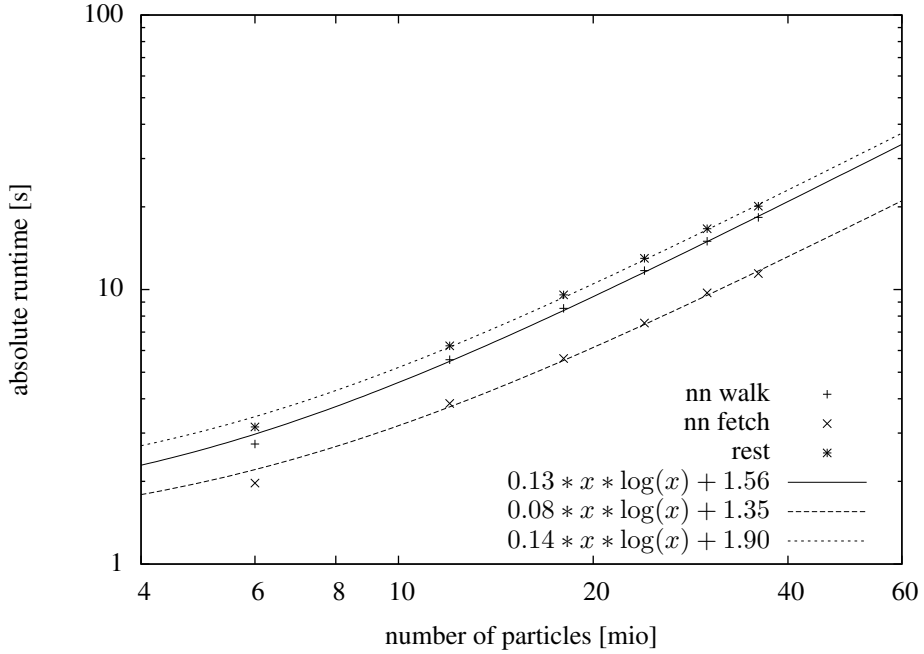


Figure 5: N scaling on Juropa using 240 processes and 50 next neighbours to find.

port and Prof. Dr. Susanne Pfalzner for giving me the opportunity to apply for this Programme. And last but not least I want to thank my GSP2010 colleagues for the nice time and for many interesting discussions.

References

1. Gibbon P, Speck R, Karmakar A, Arnold L, Frings W, Berberich B, et al. Progress in Mesh-Free Plasma Simulation With Parallel Tree Codes. Plasma Science, IEEE Transactions on. 2010 Sep;38(9):2367–2376.
2. Monaghan JJ. An introduction to SPH. Computer Physics Communications. 1988 Jan;48:89–96.
3. Hernquist L, Katz N. TREESPH - A unification of SPH with the hierarchical tree method. ApJS. 1989 Jun;70:419–446.
4. Springel V. The cosmological simulation code GADGET-2. MNRAS. 2005 Dec;364:1105–1134.
5. Gingold RA, Monaghan JJ. Smoothed particle hydrodynamics - Theory and application to non-spherical stars. MNRAS. 1977 Nov;181:375–389.
6. Lucy LB. A numerical approach to the testing of the fission hypothesis. AJ. 1977 Dec;82:1013–1024.
7. Warren MS, Salmon JK. A portable parallel particle program. Computer Physics Communications. 1995 May;87:266–290.

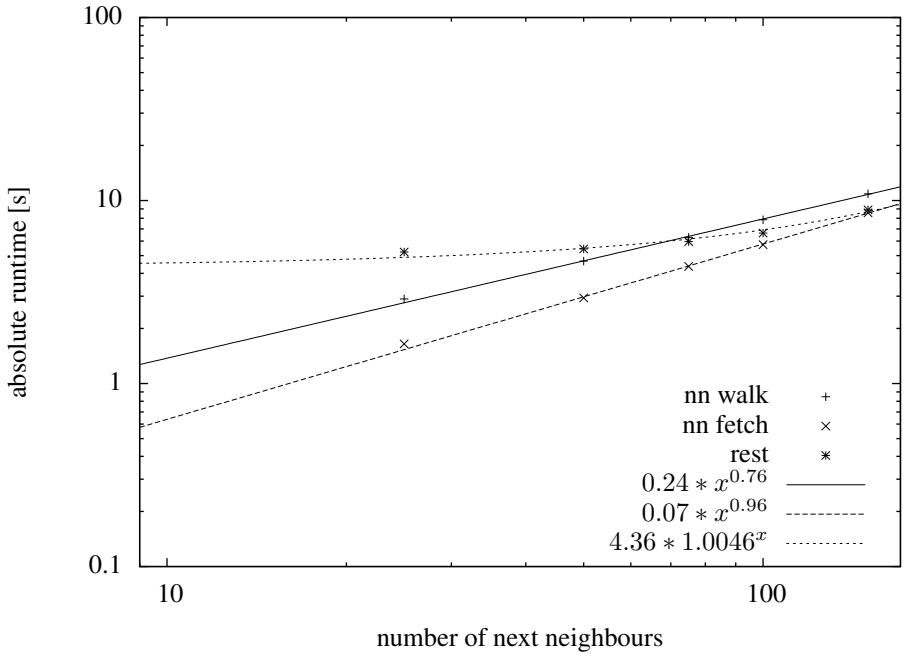


Figure 6: N_{nn} scaling on Juropa using $50\,000$ particles per process and 80 processes.

Implementation of a Parallel I/O Module for the Particle-in-Cell Code PSC

Axel Hübl

TU Dresden
Fakultät Mathematik und Naturwissenschaften
Fachrichtung Physik
01062 Dresden

E-mail: axel.huebl@mailbox.tu-dresden.de

Abstract:
An efficient parallel I/O module for the particle-in-cell code PSC has been developed using the highly scalable library SIONlib, harnessing a one-file-for-all-tasks strategy. This module enables efficient production runs on large-scale HPC systems. The performance has been extensively tested and compared with the existing one-file-per-task I/O module. The new implementation largely reduces the resource requirement for data dumping as well as for post-processing for the code PSC.

1 Introduction

Particle-in-cell (PIC) codes are efficient tools for kinetic plasma simulations, which are widely used to study laser plasma interaction experiments. During recent years the need for much higher resolutions and longer simulations made it essential to run these codes on massively parallel systems. Naturally, efficient handling of the huge amount of data produced by these massively parallel runs is absolutely necessary. One of the well accepted solutions is to use parallel one-file writing mechanisms to reduce the large data handling overheads. This motivates PIC code developers to implement such methods to cope with the need for running on several thousands of CPUs in parallel.

PSC [1] is a well-known open source 3D-PIC code, which has a one-file-per-task output strategy. Presently this code has an optimal production usage up to maximum 1k cores of the machine Juropa [2] at JSC with approximately 15 million numerical particles. It is obvious, that this limit is significantly driven by the data output method. Moreover, these high number of data files created per run can hardly be post-processed, analysed or archived even with the most efficient hardware available in the market. Additionally, the file system in use slows down because of a serialisation at the metadata servers, e.g. while creating these files.

A standard realisation of this issue is shown in Figure 1: the time needed to remove a constant amount of data of a physical system distributed over several tasks increases significantly by the number of involved tasks. The PSC output modules created one file per time-step per task, which resulted in a

task-dependent number of files. In the Figure, we compared this with a one-file-for-all-tasks operation. As a consequence, the built-in post-processor and various IDL [3] scripts were used to visualise these data, which turns the whole process out to be unmanageable with the out-of-the-box output of PSC.

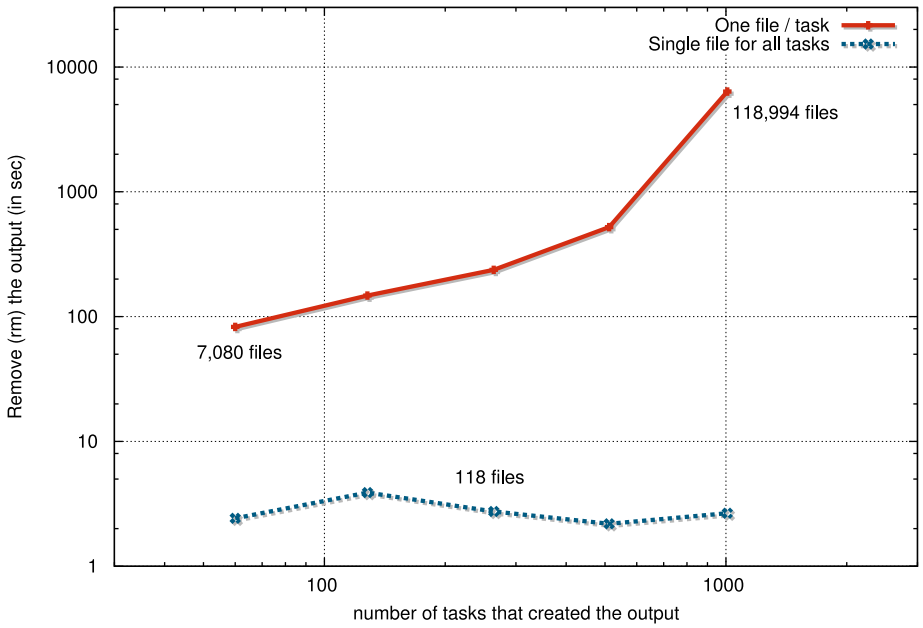


Figure 1: Time required to remove a computed series of output steps with each 41 million particles on Juropa. Having used the one-file-per-task strategy, the total number of created files was 118 times the number of involved tasks, whereas in the single-file-per-saving-instance the number of files are limited to the number of saving instances, which stays constant.

In this report, we are discussing a method of implementing a new parallel I/O for the PSC code and its testing, optimisation, scaling etc. Section 2 introduces the concept of I/O on massively parallel high performance machines. Our new implementation is discussed in Section 3. Results of some of the test post-processing and benchmarking runs are presented in Section 4. Finally, in Section 5, we summarise the current developments and have a look at future prospects.

2 Output in Massive Parallel Environments

Several approaches exist to create output efficiently on massively parallel computers. One strategy could be to collect the distributed data on a master task and write one file per output step. This obviously causes complete serialisation and global data-passing as it does not harness the parallel infrastructure of the underlying file system at all. Hence, considering parallel efficiency, this mechanism may not be suitable for the present scenario.

One of the more acknowledged solutions is to create one or only a few files for all tasks while writing in parallel in protected task-local areas of the same physical file. We are discussing the possi-

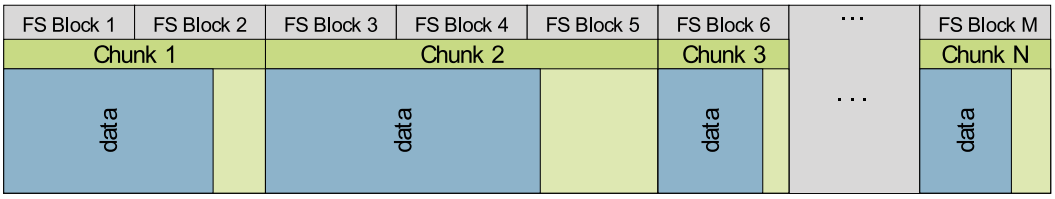


Figure 2: A simplified schematic of the file system block alignment used in SIONlib [6].

bility of realising this method using the MPI standard and the in-house developed library SIONlib below.

2.1 MPI File I/O

The MPI_File API [4] was first published in 1997. Since MPI 2.0, the I/O is a widely supported, platform independent part of the MPI standard. One of the reasons for the comprehensive support of MPI I/O is the open implementation ROMIO [5]. The interface itself is very similar to the message-passing part of MPI, whereupon several output masks and on-the-fly reordering schemes between datasets of different tasks can be applied. The so-called *external32* data representation can be used to create IEEE standardised platform and vendor independent binaries.

2.2 SIONlib

SIONlib [6] is a parallel I/O library, which maps a large number of task-local output chunks into one or a few physical files, and has been developed by W. Frings, JSC. The source code is openly available via the JSC homepage [7]. Currently, it supports the programming languages C, C++ and Fortran with MPI and OpenMP and scales on the full range of BlueGene/P [8] at JSC.

One main feature of SIONlib is that it takes care of aligning the data to the file system block size. SIONlib can be implemented easily in an existing single-file-per-task output scheme by creating an abstraction layer, that hides the internal task-local parts of a file from the file system without penalising the read and write performance. Moreover, this library adds meta information to each file, which, among others, allows the user to create endianness independent binary output.

The structure of a SIONlib generated file is demonstrated in Figure 2. Every single task writes in a protected part of the file, which is filled up with spaces up to the end of the last local file system block.

2.3 Other Possible Parallel I/O Libraries

It is also possible to use various other parallel I/O libraries, mostly MPI I/O based, that provide an on-the-fly converting to a popular portable data format, like parallel netCDF [9] and parallel HDF5 [10]. As an approach, we tried to keep the structure of the PSC internal datasets. Therefore we created our own binary middle-ware format during the parallel run of the simulation. Afterwards a converter is used to create widespread data formats for visualisation and archiving purposes.

3 The New I/O Implementation in PSC

Our basic idea of grasping the maximum possible performance during output was to hold the output datasets as long and as contiguous as possible. Therefore, the particle and field datasets are not sorted or rearranged during the output at all. Eventually, the created output has been reduced to one file per output step for fields and for particles.

At the moment we cannot use MPI I/O with the LUSTRE [11] file system on Juropa, because of file locking problems. Nevertheless, we could work out the MPI output and converting implementation with the GPFS [12] file system on Jugene. Unfortunately, we are not able to show a meaningful scaling with MPI I/O here, because of scaling limitations of the kernel of PSC.

The main differences in the binary output created by SIONlib are the strictly task-local parts of each file and the above mentioned aligning, which is not the same as in MPI I/O. For both methods we added a binary header with structural and physical information, before writing the raw data to disk. The increased complexity of the SIONlib file is balanced by the API of SIONlib that provides addressing information for the task-local parts of the file.

A comparison of the functions of MPI I/O and SIONlib is shown in table 1.

SIONlib	MPI I/O
call fsion_paropen_mpi(...)	call MPI_FILE_OPEN(...) call MPI_FILE_SET_VIEW(...)
call fsion_write(particles(1), *real8size,* numpart, fh, bwrote)	call MPI_FILE_WRITE_ORDERED(fh, particles(1), numpart, *MPI_REAL8,* status, err)
call fsion_parclose_mpi(...)	call MPI_FILE_CLOSE(...)

Table 1: Examples of the used output commands for SIONlib and MPI I/O.

3.1 Total Output Time Comparison

Figure 3 shows the total amount of time needed to output all field and particle attributes 64 times, compared to the serial output module during a production run. The size on disk of a particle file is about 4.1 GB and a field file is about 0.1 GB for one output step. The overhead for the collective file creation with SIONlib amortises with this configuration at 200 tasks and more. The parallel infrastructure of a file system with several Object Storage Targets (OSTs) can now be used efficiently because of the increased file size per output step.

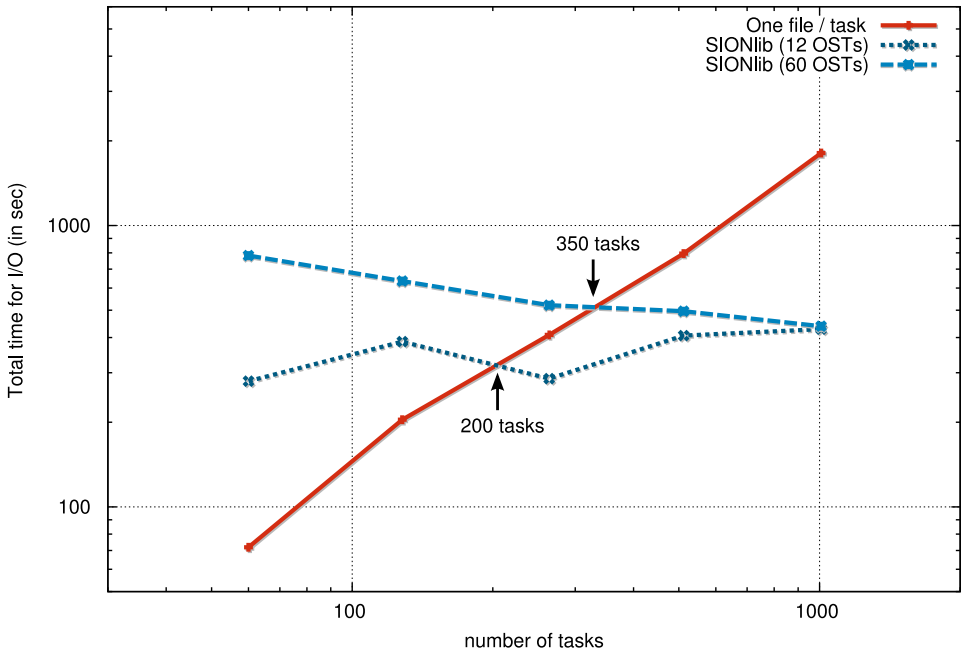


Figure 3: The total time needed for output during a strong scaling of a physical system with 41 million particles on Juropa. The serial I/O module is compared to the SIONlib module for different OSTs (see text for details.).

4 Post-Processing

Post-processing of the dumped data outputs in large parallel runs is a major issue for most of the PIC codes. We would like to use both, the old IDL visualisation scripts and a new flexible implementation for particle data analysis. Therefore, it is necessary not only to stick to the compatibility to the out-of-the-box IDL scripts, but also to introduce additional compatibilities with plotting software like Gnuplot [13], Matlab [14], Paraview [15] and VisIt [16].

Keeping in mind the easy maintainability and extendibility of the post-processing environment, a modular converter, that separates in- and output into independent layers, was used. The conversion of the particle data, e.g. from binary to a column ordered ASCII, was done via concatenation of the task-local output. On the other hand, the field attributes in the task-local parts of the output files are distributed in a domain decomposition scheme for two or three dimensional matrices. It is not wise to reorganise these data in a way a *virtual global task* would access it (as shown in Figure 4), because this would cause excessive random data access as well as buffering of the unused parts of the input file while converting. In this way the performance during post-processing can be kept up even if the output files are much bigger than the RAM of the converting machine. To allow this choice, we selected a visualisation format that understands decomposed grids without harming the performance during rendering.

A suitable file format for grid-in-grid structures like distributed fields is the XML syntax of the VTI-file format [17], which is part of VTK [18]. The XML VTI-file structure offers the possibility to create any

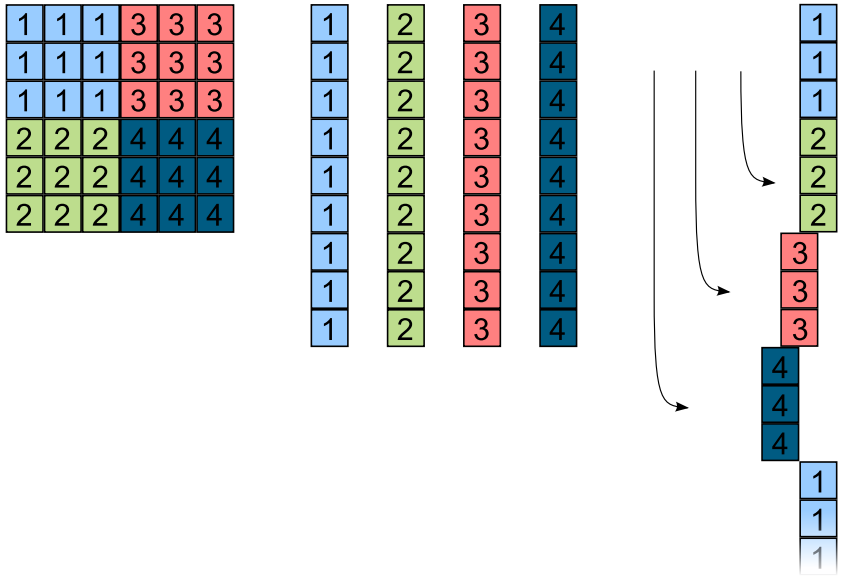


Figure 4: Scheme of a distributed matrix set. (Left) How a global task would see the distributed field. (Middle) The structure in the RAM of the separate tasks. (Right) The one dimensional reordering scheme that should be avoided therein.

decomposition of neighbouring parts of an equal spaced global mesh and can be split or joined into a few number of files, e.g. for a parallel visualisation software.

The benefit of holding the data long and contiguous during the post-processing can be seen in Figure 5, which shows the post-processing speed in MB/sec. The whole converting is mostly limited by the CPU clock rate (binary to ASCII translation) and has been speeded up by a factor of four later on by simply converting different time-steps in parallel. It is remarkable that the post-processing speed of the SIONlib files, with a constant amount of global data, is independent of the number of tasks that created these data. The post-processing speed has been increased by a factor of more than two. The chunk sizes per task seem to have no negative influence on the converting speed, even if they get below the file system block size of currently 1 MB on Lustre.

5 Conclusion & Future Prospects

We investigated and implemented the two parallel output libraries MPI I/O and SIONlib to create a single file per output step. Now the output time itself and the post-processing time are independent of the number of tasks. It is possible to overcome the production run limitations of PSC without the I/O bottleneck.

We expanded the output formats of PSC for particles on CSV, a column ordered ASCII format called PSCTXT and VTK, without loosing backwards compatibility to the old IDL scripts. The fields are now generated in the portable VTI format. The direct binary output itself can be converted platform independent on different machines and with different compilers.

The new output formats allow the use of parallel visualisation software like Paraview and VisIt, which

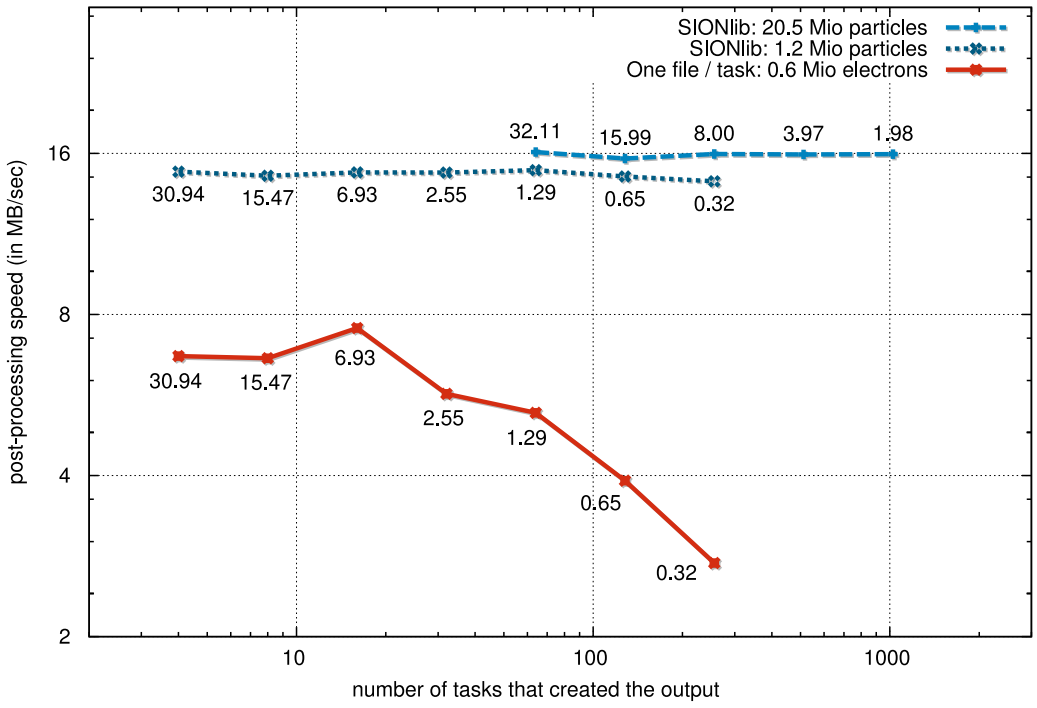


Figure 5: The amount of data in megabytes per second that can be converted during post-processing on a single CPU of Juropa with a 2.93GHz Intel Nehalem core. The labels show the task local data size (chunk size) in MB per output step.

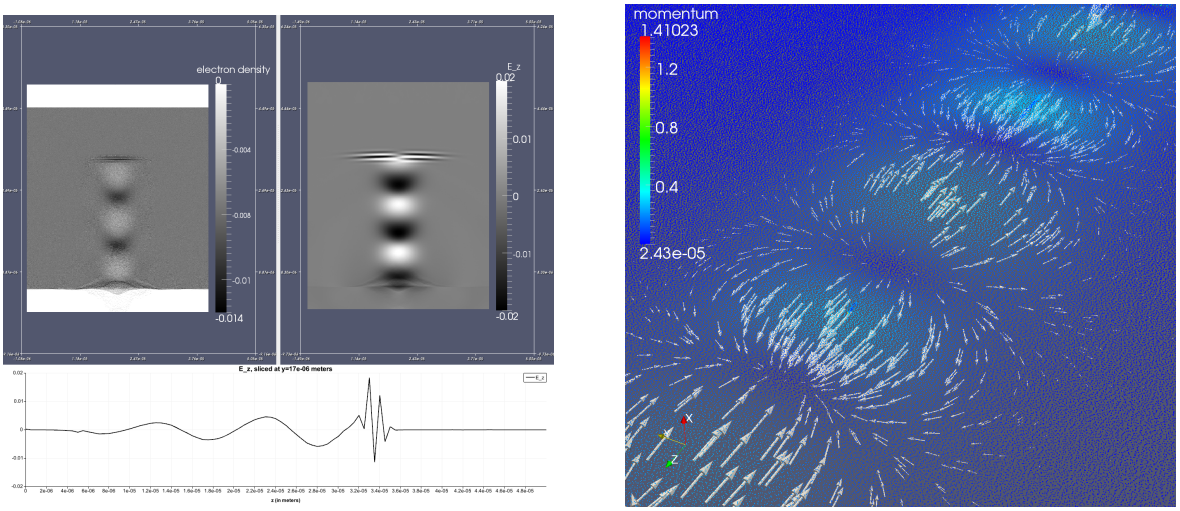


Figure 6: Visualisation example: a laser pulse propagates through a plasma and generates a so called *wake field acceleration*. (Left) The plasma electron density and the longitudinal electric field E_z. (Below) A slice along the vertical axis of E_z. (Right) The electron phase space long time after the laser pulse propagated.

in turn allow to use specialised rendering clusters like Juvis [19]. For a quantitative analysis, these software also provide analytical tools and filters. An example of the visualisation created in Paraview, using the new implementation, is shown in Figure 6.

For future developments, we presume that some easy improvements in the output formats can be made to handle large amounts of data (eg. terabytes order). The VTI-files can be created with binary data content and should be split up in sub-files to deploy the underlying file system of the visualisation cluster efficiently.

The I/O part of PSC supports more domain decomposition schemes, like adaptive grids, and the converters can be used for similar projects. Moreover, it is possible to extend the file format output, e.g. with HDF5 or netCDF, to interact with other simulation software.

A SIONlib based check pointing module is in progress, which can be used to restore the state of a complete parallel simulation from previous runs.

6 Acknowledgements

The author would like to thank his advisor Anupam Karmakar and other members of the group of Paul Gibbon for their open and productive support during his entire stay at the guest student programme. Special thanks go to Michael Bussmann from the FZ Dresden-Rossendorf for mentoring the author and supporting his application. Furthermore, the author expresses his gratitude to Wolfgang Frings and Marc-André Hermanns for the excellent technical introductions and the patient discussions about parallel I/O. Finally, sincere thanks to all the other guest students and the organisers of the programme for all the inspiring conversations and for a really nice time.

References

1. H. Ruhl. Classical Particle Simulations with the PSC code
2. Jülich Supercomputing Centre. JUROPA, http://www.fz-juelich.de/jsc/juropa/
3. ITT Visual Information Solutions. IDL (Interactive Data Language), http://www.ittvis.com/ProductServices/IDL.aspx
4. MPI Forum. MPI: A Message-Passing Interface Standard, Chapter 13, Version 2.2 (2009)
5. University of Chicago. ROMIO: A High-Performance, Portable MPI-IO Implementation, http://www.mcs.anl.gov/research/projects/romio/
6. W. Frings, F. Wolf, V. Petkov. Scalable massively parallel I/O to task-local files, SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Article No. 17, S. 1-11
7. W. Frings. SIONlib: Scalable parallel I/O for task-local files, http://www.fz-juelich.de/jsc/sionlib/
8. Jülich Supercomputing Centre. JUGENE, http://www.fz-juelich.de/jsc/jugene/
9. netCDF, http://www.unidata.ucar.edu/software/netcdf/
10. HDF5, http://www.hdfgroup.org
11. SUN Microsystems. The Lustre file system, http://www.lustre.org
12. IBM. GPFS (General Parallel File System), http://www-03.ibm.com/systems/software/gpfs/
13. Gnuplot, http://www.gnuplot.info
14. The MathWorks Inc. Matlab, http://www.mathworks.de
15. Kitware Inc. Paraview, http://www.paraview.org
16. Lawrence Livermore National Laboratory. VisIt, https://wci.llnl.gov/codes/visit/
17. Kitware Inc. The VTK User's Guide for VTK 4.2, Chapter 14.3, VTK File Formats
18. Kitware Inc. VTK (Visualization Toolkit), http://www.vtk.org
19. Jülich Supercomputing Centre. JUVIS. http://www.fz-juelich.de/jsc/cv/vislab/juvis/

Pedestrian Dynamics: Implementation and Analysis of ODE-solvers

Timo Hülsmann

Bergische Universität Wuppertal
Fachbereich E
Elektrotechnik, Informationstechnik, Medientechnik
Rainer-Gruenter-Str. 21
42119 Wuppertal

E-mail: Timo.Huelsmann@uni-wuppertal.de

Abstract:
In this report two approaches for run-time optimization of the General Centrifugal Force Model (GCFM) are analysed. First we give a short introduction in modeling pedestrian dynamics and introduce the GCFM. The first approach consists of ordering pedestrians' data in the local memory to preserve data locality. The purpose is to reduce cache misses and is done using space-filling curves. This is presented in the second part of this report. A small decrease of computation-time was achieved. In the third part different ODE-solvers are investigated, with fixed and with varying step-size. The solvers are the Velocity Verlet method and different orders of the Runge-Kutta-Fehlberg method. Here an increase of the average step-size was achieved.

1 Introduction

In recent years pedestrian dynamics has gained more importance and has become an interesting field of research due to continuously growing urban population and cities. Simulations are already conducted to enhance the safety and security of pedestrians in complex buildings and mass events. In this context and as part of the German Government's high-tech strategy, the Federal Ministry of Education and Research (BMBF) has funded the Hermes project [1] for addressing this demand. The aim of Hermes is the development of an evacuation assistant which will be installed and tested in the ESPRIT arena in Düsseldorf. One of the challenges is to simulate the evacuation of 50000 persons for the next 15 minutes faster than real-time. First approaches to optimize the run-time have been implemented and include the Linked-Cell method [2]. In this work different ODE-solvers are investigated with fixed and varying step-size.

2 Modeling pedestrian dynamics

The approaches of modeling pedestrians' motion fall into two main groups: microscopic and macroscopic models. In macroscopic models the system is described by its mean values and therefore conservation laws for density, flow, etc. are used. Microscopic models can be further categorized in spatially discrete (e.g. Cellular Automata) and spatially continuous models (e.g. Social Force Model, Generalized Centrifugal Force Model). A detailed description of these modeling approaches can be found in [3].

2.1 General Centrifugal Force Model

The GCFM [4] is based on Newton's Second Law, i.e. $\vec{F} = m \cdot \vec{a}$. Thus the trajectories (position in time) of the pedestrians are determined by the forces acting on them. Each pedestrian has a driving force which is directed to the desired direction. This direction can be an exit or an intermediate destination point. The driving force has the form

$$\vec{F}_{iB}^{driv} = m_i \frac{\vec{V}_i^0 - \vec{V}_i}{\tau} \tag{1}$$

where $\vec{V}_i^0$ is the desired velocity of pedestrian i, $\vec{V}_i$ is the current velocity and τ is a relaxation term.

There are two types of repulsive forces: repulsive forces from walls and repulsive forces from other pedestrians. These forces ensure a correct interaction with the environment and with other pedestrians.

$$\vec{F}_{ij}^{rep} = -m_i K_{ij} \frac{(\eta V_i^0 + V_{ij})^2}{\text{dist}_{ij}} \vec{e}_{ij} \tag{2}$$

$$\vec{R}_{ij} = \vec{R}_j - \vec{R}_i\,, \quad \vec{e}_{ij} = \frac{\vec{R}_{ij}}{\|\vec{R}_{ij}\|} \tag{3}$$

$$V_{ij} = \frac{1}{2}((\vec{V}_i - \vec{V}_j) \cdot \vec{e}_{ij} + |(\vec{V}_i - \vec{V}_j) \cdot \vec{e}_{ij}|) \tag{4}$$

$$K_{ij} = \frac{1}{2} \frac{\vec{V}_i \cdot \vec{e}_{ij} + |\vec{V}_i \cdot \vec{e}_{ij}|}{\|\vec{V}_i\|} \tag{5}$$

Equation (2) describes the repulsive force. The formulas are the same for forces from walls and other pedestrians. The relative velocity V_{ij} defined by Equation (4) ensures that pedestrians with higher velocity in front of pedestrian i do not affect him/her. The coefficient K_{ij} which is defined in Equation (5) becomes zero if pedestrian j is not in the field of vision of pedestrian i, i.e. pedestrian j has no effect on pedestrian i in this case. Each pedestrian is modelled by an ellipse whose major semi-axis depends on

the actual velocity of the pedestrian. Therefore dist_{ij} is the distance between the ellipses of pedestrian i and j.

Hence summing all forces up

$$\vec{F}_i = \sum_{i \neq j}^{N} \vec{F}_{ij}^{rep} + \sum_{B} \vec{F}_{iB}^{rep} + \vec{F}_i^{driv} \tag{6}$$

results in the force acting on pedestrian i. After defining the forces we can set up the differential equations

$$\frac{d}{dt}\vec{R}_i = \vec{V}_i\,, \quad m_i \frac{d}{dt}\vec{V}_i = \vec{F}_i\,, \quad i = 1, \ldots, N \tag{7}$$

for our model, where N denotes the number of pedestrians.

Some problems exist with the used implementation of the GCFM. Not all pedestrians reach the exit in the simulation. This is due to unrealistic high velocities which occur during the simulation. Therefore the pedestrians have a new position which is outside of the used geometry.

3 Preserving data locality with space-filling curves

In this part an older implementation, which goes back to [2], is used. This implementation uses the Linked-Cell method to reduce the complexity of the force-calculation to $\mathcal{O}(N)$. The simulation area is divided into quadratic cells with edge length L_c, where L_c equals the cutoff-radius. Pedestrians are assigned to cells at each simulation step. For force calculation only pedestrians from the own cell and the eight neighboring cells are considered.

We want to reduce cache-misses in the calculation of the forces. Since only the data of pedestrians in the neighborhood is needed to calculate the repulsive forces, it would be useful if the data of these pedestrians is also nearby in memory. The pedestrians' data consists of the current position, current velocity, desired velocity, angle of the major semi-axis of the ellipse describing the pedestrian towards the x-axis, current room, exit and an index.

3.1 Space-filling curves

A space-filling curve is a mapping, $f : [0,1] \to [0,1]^2$, from the unit interval to the unit square s.t. it fills the whole space. It is continuous and surjective. For details on space-filling curves we refer to [5].

We use the Hilbert curve (H_n) to arrange the pedestrians' data in memory since it has a good locality-preserving behavior. Its construction process is recursive and can be seen in figure 1. In the upper left corner the figure shows the first approximation H_1. The Hilbert curve in the mathematical sense is the limit for $n \to \infty$. Other space-filling curves are e.g. the Peano curve or the Z-curve.

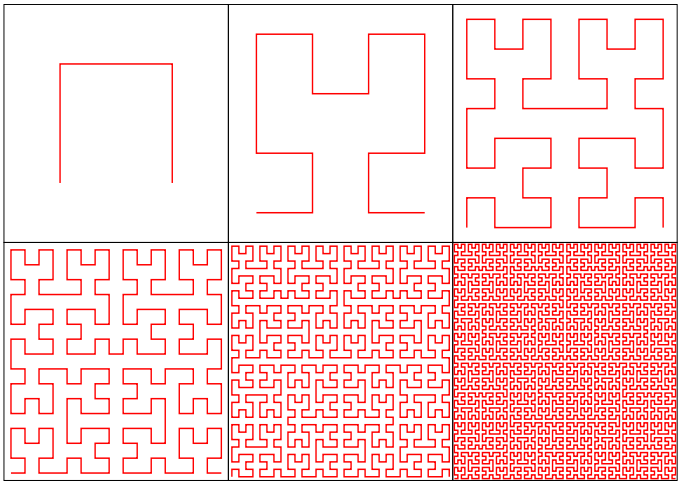


Figure 1: Construction process of the Hilbert curve, source:
`http://en.wikipedia.org/wiki/File:Hilbert_curve.svg`.

3.2 Test-cases

To test the effect of ordering the pedestrians' data in a specific way the pedestrians were distributed in four different ways: column-wise, row-wise, along a Hilbert curve and random. For the ordering along a Hilbert curve a function which evaluates the Hilbert curve at a specific argument was used. The approach to calculate the coordinates was taken from [6].

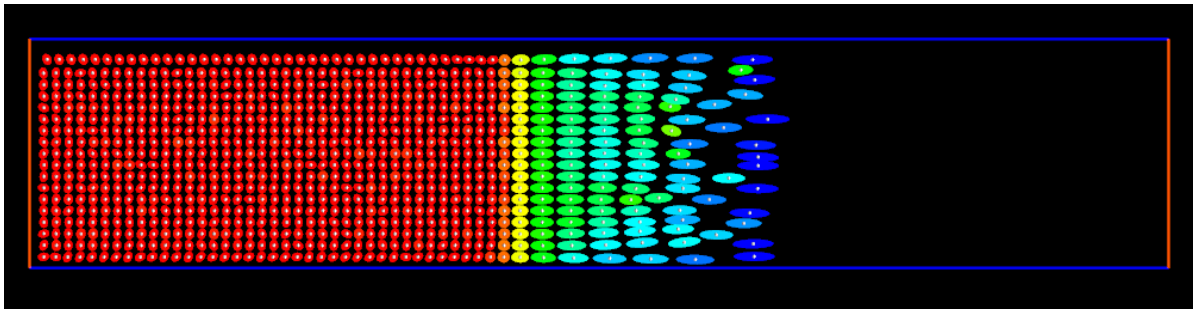
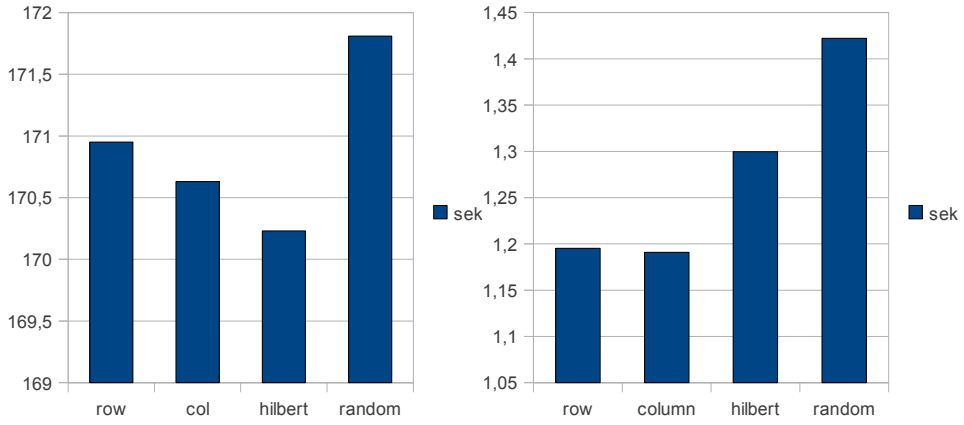


Figure 2: Corridor with pedestrians moving to the right

The used geometry was the corridor shown in figure 2. 1152 pedestrians were placed on the left side of the corridor and sent to the right.



(a) Run-time for complete simulation (b) Run-time for the first time-step

Figure 3: Results.

3.3 Conclusion and analysis

Figure 3a shows the run-time of the simulation until every pedestrian has left the corridor. These measurements were done on a single node of JUROPA [7]. We can see that the case of pedestrians' data ordered along a Hilbert curve had the fewest complete run-time (Figure 3a). The difference between ordering along a Hilbert curve and random ordering is $1.58s$, i.e. ordering along a Hilbert curve is only about 1% faster.

The time needed to compute the first step, as shown in figure 3b, shows a different picture. Here row-wise and column-wise ordering are the fastest. Further investigations showed that most of the computation-time was spent in parts of the program where the ordering of pedestrians is not important (e.g. the calculation of the ellipses). Hence in a more optimized implementation the effect of ordering along a Hilbert curve might become more visible.

4 Implementation and analysis of ODE-solvers

Integrating the system (7) has high computational effort. Hence increasing the step-sizes of the integration is desirable. So far the explicit Euler-method was used for integrating the system (7). In this part of the work we want to analyse if we can increase the step-sizes with the Runge-Kutta-Fehlberg method and with the Velocity-Verlet method. The influence of the new methods on the flux in front of a bottleneck is investigated. Here we use the integral of the flux, i.e. the N/t-diagram. Hence this diagram shows how many pedestrians have entered the bottleneck up to a specific time t.

4.1 Transformation of the GCFM into standard form

The system (7) can easily be transformed into a standard system of autonomous ODEs $\frac{d}{dt}\vec{x} = \vec{f}(\vec{x})$ by using

$$\vec{x} = (R_{1,x}, R_{1,y}, V_{1,x}, V_{1,y}, \ldots, R_{N,x}, R_{N,y}, V_{N,x}, V_{N,y})^T \tag{8}$$

$$\vec{f}(\vec{x}) = (V_{1,x}, V_{1,y}, F_{1,x}/m_1, F_{1,y}/m_1, \ldots, V_{N,x}, V_{N,y}, F_{N,x}/m_N, F_{N,y}/m_N)^T \tag{9}$$

where the subscripts i, x and i, y denote the x- resp. y-component of the position, velocity and force of the pedestrian i. Now we have a system of ODEs in standard form where we can apply standard numerical integration techniques.

4.2 Runge-Kutta-Fehlberg 2(3)

The Runge-Kutta-Fehlberg method [8] can integrate a system of ODEs $\frac{d}{dt}\vec{x} = \vec{f}(t, \vec{x})$. Since the GCFM describes an autonomous system, i.e. $\frac{d}{dt}\vec{x} = \vec{f}(\vec{x})$, we do not need to calculate values for t to evaluate the right-hand side $\vec{f}$. The Runge-Kutta-Fehlberg 2(3) method uses two integration schemes of order 2 and order 3 respectively. Here $\vec{x}_0$ denotes the current state of the system and $\vec{x}_1$ respectively $\hat{\vec{x}}_1$ denote the state of the system at time $t + h$.

$$\vec{x}_1 = \vec{x}_0 + h\sum_{i=1}^{3} b_i\vec{k}_i\,, \quad \hat{\vec{x}}_1 = \vec{x}_0 + h\sum_{i=1}^{4} \hat{b}_i\vec{k}_i \tag{10}$$

The increments read as:

$$\vec{k}_i = \vec{f}(t + hc_j, \vec{x}_0 + h\sum_{j=1}^{i-1} a_{ij}\vec{k}_j)\,, \quad i = 1, \ldots, 4. \tag{11}$$

The coefficients of Runge-Kutta methods are given in so called Butcher tableaus [9]. The Runge-Kutta-Fehlberg methods use embedded schemes where the coefficients b_i and $\hat{b}_i$ for the two methods of different order are given as shown in Table 1a. In Table 1b the Butcher tableau for the RKF 2(3) method is shown.

The difference $\vec{e} = \vec{x}_1 - \hat{\vec{x}}_1 = \mathcal{O}(h^3)$ between the result of the order 2 and the order 3 methods is used as an error estimation. This error estimation is used to calculate a new optimal step size so that the error lies within given absolute and relative tolerances (atol and rtol):

c_1	a_{11}	a_{12}	$\cdots$	a_{1s}
c_2	a_{21}	a_{22}	$\cdots$	a_{2s}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_s	a_{s1}	a_{s2}	$\cdots$	a_{ss}
	b_1	b_1	$\cdots$	b_{ss}
	$\hat{b_1}$	$\hat{b_1}$	$\cdots$	$\hat{b_s}$

(a) Butcher tableau with embedded scheme

0				
1/4	1/4			
27/40	-189/800	729/800		
1	214/891	1/33	650/891	
	214/891	1/33	650/891	
	533/2106	0	800/1053	-1/78

(b) Butcher tableau for RKF 2(3) method

Table 1: A general Butcher tableau and the tableau for the RFK 2(3) method.

$$h_{opt} = h_{used} \cdot \sqrt[3]{\frac{1}{\text{ERR}}}, \quad \text{ERR} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(\frac{e_j}{\text{atol} + \max\{|x_{0,j}|, |x_{1,j}|\} \cdot \text{rtol}}\right)^2}. \tag{12}$$

The new step size h_{opt} is scaled by a safety factor $\rho = 0.9$. Also it is ensured that the new step size lies within defined limits ($\sigma \cdot h_{used} < h_{new} < \theta \cdot h_{used}$). Here $\sigma = 0.2$ and $\theta = 5$ are used. These limits are used to avoid oscillating step-sizes.

4.3 Runge-Kutta-Fehlberg 4(5)

Another used method is the RKF 4(5) which reads:

$$\vec{x}_1 = \vec{x}_0 + h\sum_{i=1}^{5} b_i\vec{k}_i\,, \quad \vec{\hat{x}}_1 = \vec{x}_0 + h\sum_{i=1}^{6} \hat{b}_i\vec{k}_i. \tag{13}$$

$$\vec{k}_i = \vec{f}(t + hc_j, \vec{x}_0 + h\sum_{j=1}^{i-1} a_{ij}\vec{k}_j)\,, \quad i = 1, \ldots, 6. \tag{14}$$

The only difference between RKF 2(3) and RKF 4(5) is that we now use one method of order 4 and one method of order 5 to estimate the error. The coefficients are given in Table 2.

4.4 Velocity-Verlet method

The Velocity-Verlet method [10] is widely used for dissipative systems in molecular dynamics. This method has a fixed step-size.

0					
1/4	1/4				
3/8	3/32	9/32			
12/13	1932/2197	-7200/2197	7296/2197		
1	439/216	-8	3680/513	-845/4104	
1/2	-8/27	2	-3544/4104	1859/4104	-11/40
25/216	0	1408/2565	2197/4104	-1/5	
16/135	0	6656/12825	28561/56430	-9/5	2/55

Table 2: Butcher tableau for RKF 4(5) method

$$\vec{R}_i(t+h) = \vec{R}_i(t) + \vec{V}_i(t)h + \frac{1}{2}\vec{F}_i(t)h^2 \tag{15}$$

$$\vec{V}_i(t+h) = \vec{V}(t) + \frac{1}{2}\left[\vec{F}_i(t) + \vec{F}_i(t+h)\right]h \tag{16}$$

The steps for the integration are as follows:

1. Compute $\vec{R}_i(t+h)$ using Equation (15).
2. Estimate $\vec{V}_i(t+h)$ with $\vec{V}(t) + \frac{1}{2}\vec{F}_i(t)h$.
3. Compute $\vec{F}_i(t+h)$.
4. Compute $\vec{V}_i(t+h)$ using Equation (16).

4.5 Test-cases

As test-cases a bottleneck with 90 cm width (Figure 4) and a bottleneck with 160 cm width (Figure 5) were chosen. The pedestrians started in the left room, went through the bottleneck and left through the right. In the first test-case $N = 20$ was chosen and in the second test-case $N = 50$. In earlier tests a step-size of 0.01 was used for the explicit Euler method, but now we use a step-size of 0.001 as "reference" in order to minimize the problems mentioned in Subsection 2.1. The RKF 2(3) and RKF 4(5) methods were tested with different absolute and relative tolerances. Also different step-sizes were tested for the Velocity-Verlet method. To compare the accuracy of the methods it was measured how many pedestrians have entered the bottleneck at any time. For the RKF methods the used step-sizes were measured.

4.6 Results

Figures 6 and 7 compare the explicit Euler method and the Velocity-Verlet method in the two test-cases. In Figure 6a we can see that the N/t-diagram for the explicit Euler method and the Velocity-Verlet method with step-sizes from 0.001 up to 0.004 fit. For step-size 0.005 the N/t-diagram of the Velocity-Verlet method is below the N/t-diagram of the explicit Euler method with step-size 0.001.

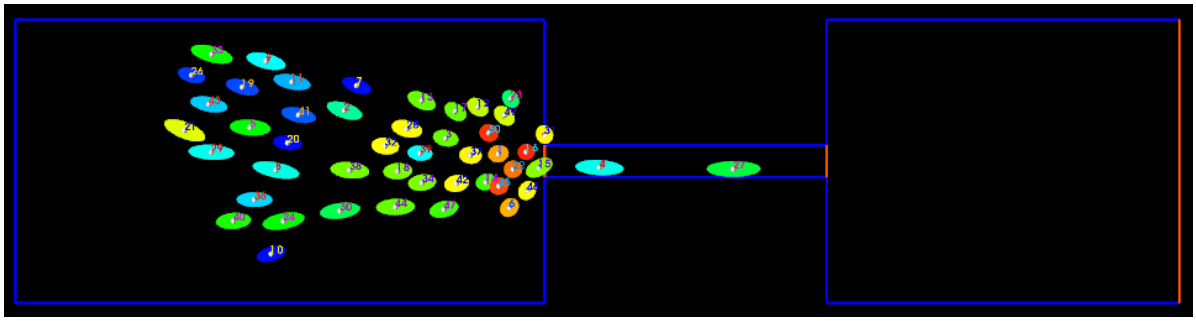


Figure 4: Bottleneck with 90 cm width

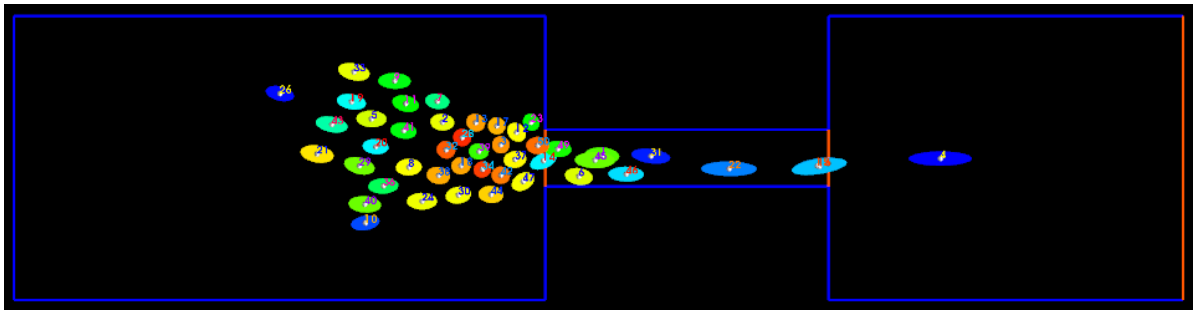


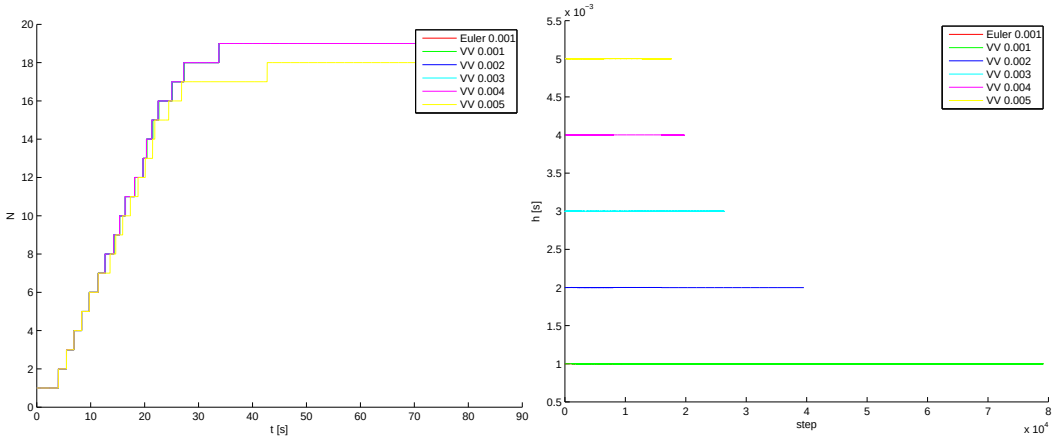
Figure 5: Bottleneck with 160 cm width

Again we compare the explicit Euler method and the Velocity-Verlet method, but now use the second test-case. The results are shown in figure (7). Here until about $25s$ the N/t-diagram is quite similar, but then starts to differ.

Now we come to the results for the RKF 2(3) method. In Figure 8 the Euler method is compared with the RKF 2(3) method in the case of an bottleneck of 90 cm width and $N = 20$. We can see in Figure 8a that the slope of the plot gets higher when the settings for the tolerances get lower. But even for the highest tolerances the slope of the RKF 2(3) N/t-diagram is much higher than of the explicit Euler N/t-diagram. When we look at the step-sizes in Figure 8b we can see that they get really big at the end. This comes due that at the end nearly all pedestrians have already left the geometry. Further investigations showed that the step-sizes get small when many pedestrians are in front of the bottleneck. So this is the behavior of an adaptive method one might expect. Fewest steps are needed for tolerances atol = 10^{-6} and rtol = 10^{-7}.

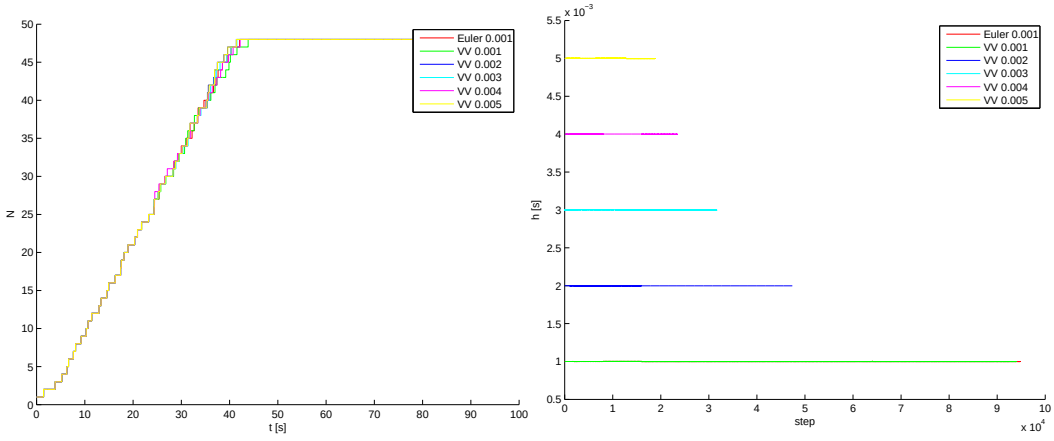
The results for the second test-case shown in Figure 9 are similar. Here also the slope of the N/t-diagram gets higher if the tolerances get lower. In comparison to the latter test-case much more steps are needed. Also the setting atol = 10^{-6} and rtol = 10^{-7} results in fewest steps.

Figure 10 shows the results for the RKF 4(5) method in the case of the 90 cm bottleneck. We can see a much better fit for the N/t-diagram in Figure 9a. Only at about 35 s there is a bigger difference between the plots from the RKF 4(5) method and the explicit Euler method. In Figure 9b we see the



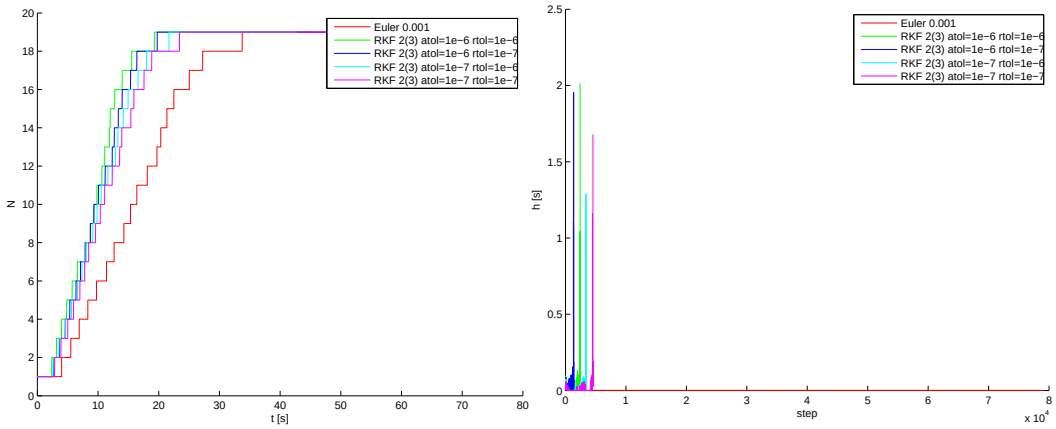
(a) Pedestrians entered the bottleneck at time t (b) Step-sizes

Figure 6: Comparison of explicit Euler method with Velocity-Verlet method, 90 cm bottleneck, $N = 20$



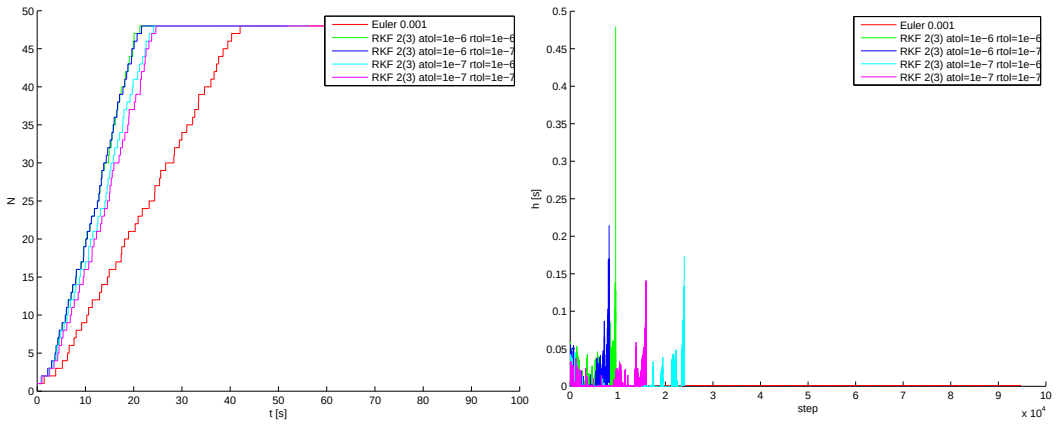
(a) Pedestrians entered the bottleneck at time t (b) Step-sizes

Figure 7: Comparison of explicit Euler method with Velocity-Verlet method, 160 cm bottleneck, $N = 50$



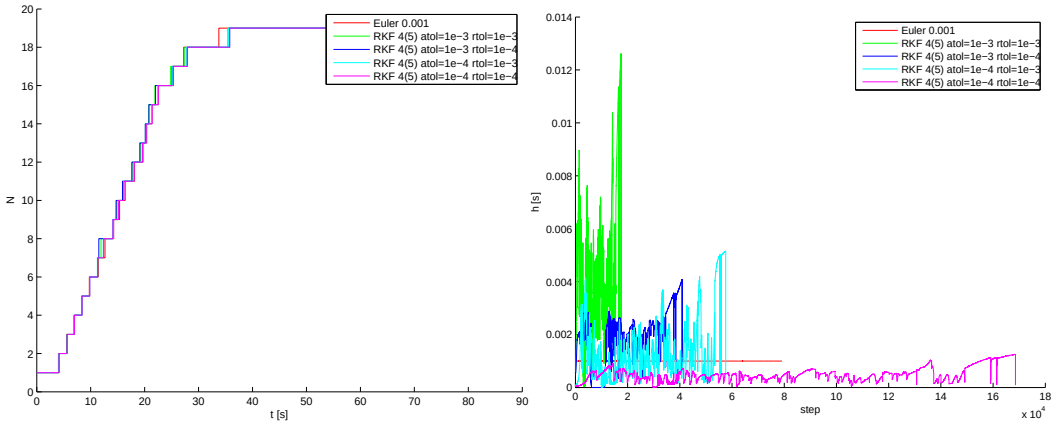
(a) Pedestrians entered the bottleneck at time t (b) Step-sizes

Figure 8: Comparison of explicit Euler method with RKF 2(3) method, 90 cm bottleneck, $N = 20$



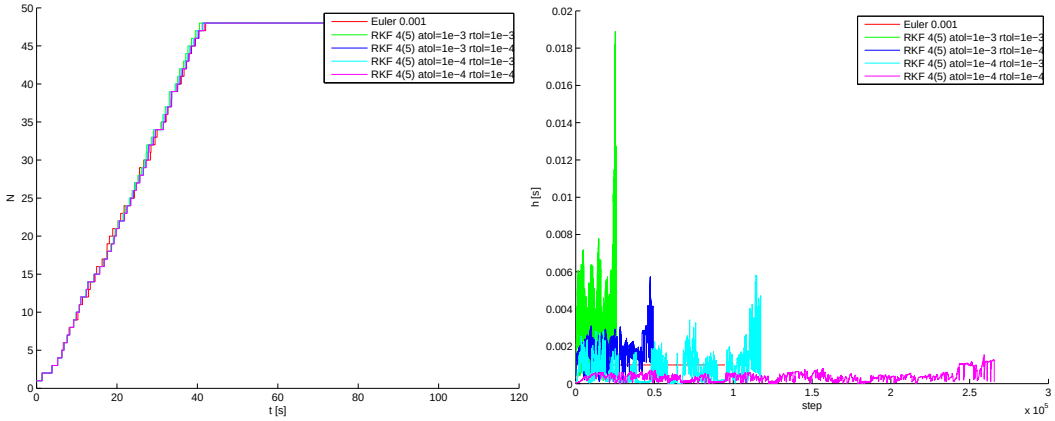
(a) Pedestrians entered the bottleneck at time t (b) Step-sizes

Figure 9: Comparison of explicit Euler method with RKF 2(3) method, 160 cm bottleneck, $N = 50$



(a) Pedestrians entered the bottleneck at time t (b) Step-sizes

Figure 10: Comparison of explicit Euler method with RKF 4(5) method, 90 cm bottleneck, $N = 20$



(a) Pedestrians entered the bottleneck at time t (b) Step-sizes

Figure 11: Comparison of explicit Euler method with RKF 4(5) method, 160 cm bottleneck, $N = 50$

used step-sizes. For atol = 10^{-4} and rtol = 10^{-4} more steps are needed than with the explicit Euler method. Fewest steps are needed for atol = 10^{-3} and rtol = 10^{-3}.

The results for the 160 cm bottleneck shown in Figure 11 are similar. We see again a better fit between the RKF 4(5) and the Euler method in Figure 11a. For atol = 10^{-3} and rtol = 10^{-3} again fewest steps are needed. For atol = 10^{-4} and rtol = 10^{-3} and for atol = 10^{-4} and rtol = 10^{-4} more steps than for the Euler method are needed.

5 Conclusion and Outlook

The Velocity-Verlet method allows to use a step-size of 0.004 instead of 0.001 for the explicit Euler method. RKF 2(3) uses much higher step-sizes but shows a big discrepancy in the N/t-diagram, i.e. the slope is higher than for the explicit Euler method. Hence the simulation with this method shows

another characteristic behavior. This might be improved by using higher tolerances and therefore also using double precision arithmetic. For the RKF 4(5) method we get better results in the N/t-diagram but the computational effort is much higher than for the RKF 2(3) method, i.e. much more function evaluations are needed.

Thus the Velocity-Verlet method and the RKF methods allow higher step-sizes. But it has to be investigated if we can reduce the whole simulation-time. Also other characteristics of pedestrian simulation (e.g. the density in front of a bottleneck) might be investigated.

Acknowledgments

This work has been performed within the program “Research for Civil Security” in the field “Protecting and Saving Human Life" funded by the German Government, Federal Ministry of Education and Research (BMBF).

References

1. Hermes – Investigation of an evacuation assistant for use in emergencies during large-scale public events. Jülich: Forschungszentrum Jülich; [updated 6 May 2009; cited 25 October 2010]. Available from: http://www.fz-juelich.de/jsc/hermes
2. Mehlich J. Laufzeitoptimierung von Simulationen raumkontinuierlicher Modelle der Fußgängerdynamik mithilfe von Nachbarschaftslisten. Berichte des Forschungszentrums Jülich; 2009.
3. Schadschneider A, Klingsch W, Klüpfel H, Kretz T, Rogsch C, Seyfried A. Evacuation dynamics: Empirical results, modeling and applications. In: R. A. Meyers (Ed). Encyclopedia of Complexity and System Science. Springer. 2009;31-42.
4. Chraibi M, Seyfried A, Schadschneider A. The Generalized Centrifugal Force Model. arXiv:1008.4297v1 [physics.soc-ph]; 2010. Available from: http://arxiv.org/abs/1008.4297
5. Sagan H. Space-Filling Curves. New York: Springer-Verlag; 1994.
6. Salomon D. Data Compression: The Complete Reference. 4th ed. London: Springer; 2006.
7. Clustercomputer JUROPA. Jülich: Forschungszentrum Jülich; [updated 25 August 2009; cited 25 October 2010]. Available from: http://www.fz-juelich.de/portal/forschung/information/supercomputer/juropa
8. Fehlberg E. Klassische Runge-Kutta-Formeln vierter und niedrigerer Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf Wärmeleitungsprobleme. Computing. 1970;6(1):61-71.
9. Stoer J, Bulirsch R. Numerische Mathematik 2. Berlin: Springer; 2005.
10. Sadus RJ. Molecular Simulation of Fluids: Theory, Algorithms and Object-Orientation. New York: Elsevier; 1999.

Integration of a High Order Compact Scheme into Multigrid

Alina Georgiana Istrate

Bergische Universität Wuppertal
Gaußstraße 20
42119 Wuppertal
E-mail: alina.istrate@uni-wuppertal.de

Abstract:
A 6th-order compact finite difference scheme was implemented into a geometric multigrid method, which is used to solve the 3-dimensional Poisson equation. The method is the basic component of a particle-particle particle-mesh method which is used to calculate long range Coulomb interactions between charged point like particles in molecular simulations.

1 Introduction

Coulomb interactions play a crucial role for static and dynamical proprieties in a variety of complex systems characterized by polar or charged system components, e.g: polar liquids, proteins, DNA or plasmas, to name a few.

Due to the long range nature of these interactions the calculation of forces is computationally very demanding. Many methods exist which try to reduce the complexity of the problem from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ or $\mathcal{O}(N \log(N))$. One of these methods is the so called particle-particle particle-mesh method (P^3M) which solves the field equation with a fast Fourier method by using a modified Green's function which adjusts the solution closely to the continuum solution [1].

Numerical approximation of three-dimensional partial differential equations (in particular Poisson equation) are computationally challenging with respect to memory and compute time, since often systems of equations on large 3-dimensional grids, which guarantee a necessary spatial resolution, are to be solved.

Traditional numerical schemes often have only 2nd-order accuracy and therefore require fine discretizations, i.e. demanding for large grids and therefore result in very large systems of equations. One approach to eliminate this is to use higher order or spectral methods which usually give comparable accuracy with much coarser discretizations.

The higher-order compact (HOC) schemes used in the following extend the idea of the standard central differencing schemes in which the lowest-order terms of the truncation error are locally approximated using the differential equation. The resulting scheme is "compact" which means that for a

given node point on the grid, the finite difference operators are evaluated on grid points with cartesian components $|\alpha| \leq 1, \alpha = i, j, k$, where i, j, k are relative indices with respect to the considered node. The result is that the nodal accuracy of the scheme is increased at the expense of only a slight increase in the density of the sparse matrix structure compared to the minimal $\mathcal{O}(h^2)$ stencil.

2 Theory

2.1 Physical problem

In classical mechanics, starting from a set $\mathcal{P}$ of particles in an initial state given by $S_0 = [\vec{x_1}, ..., \vec{v}_1, ...]$ a system can be described completely in terms of coordinates and velocities of the particles. The time evolution of the system is given by Newton's equations of motion:

$$\vec{v_i} = \frac{d}{dt}\vec{x_i} \qquad \vec{F_i} = \frac{d}{dt} m_i \vec{v_i}$$

for a particle with index $i \in [1, N]$, where N is the total number of particles in $\mathcal{P}$. The force acting on particle i is given by the sum of the forces due to all other particles in the system:

$$\vec{F_i} = \sum_{j \in \mathcal{P}/\{i\}} \vec{F}_{i,j}$$

The forces are given by the gradient of the potentials:

$$\vec{F_i} = -\nabla \Phi_i \qquad \vec{F}_{i,j} = -\nabla \Phi_{i,j} \qquad \Phi_i = \sum_{j \in \mathcal{P}/\{i\}} \Phi_{i,j}$$

Therefore, the evolution of a system is a consequence of the effective potential. In the following we consider just the electrostatic potential.

The Green's function of the Poisson equation in $\mathbb{R}^3$ is given by:

$$U(x) = \frac{1}{4\pi \|\vec{x}\|_2}$$

Taking into account the similarity of the given Green's function and the Coulomb potential

$$\Phi_i = \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{1}{4\pi\epsilon_0} \frac{q_j}{\|\vec{x}_i - \vec{x}_j\|_2}$$

one can state that the Coulomb potential is a solution of the following Poisson equation:

$$\Delta \Phi_i(\vec{x}) \;=\; \rho(\vec{x} \,|\, \vec{x}_i) \;=\; \frac{1}{\epsilon_0} \sum_{\substack{j=1 \\ j \neq i}}^{N} q_i \delta(\|\vec{x}_i - \vec{x}_j\|_2) \tag{1}$$

where $\rho(\vec{x}\,|\,\vec{x}_i)$ denotes all spatial coordinates $\vec{x}$ but the evaluation point $\vec{x}_i$, i.e. not considering self-interactions. The connection with the Poisson equation allows us to define numerical schemes to calculate the electrostatic quantities of the system that are based on the solution of the Poisson equation on a mesh.

2.2 Multigrid solution of Poisson equation

The idea of multigrid [2] is to solve the Poisson equation on a hierarchy of fine and coarse grids with finite difference schemes. On the finest grid the operator works on the field $\hat{\Phi}_h$, giving an approximation $\hat{\Phi}_h{}'$. It can be shown [2] that the resulting error $\hat{\epsilon}_h$ with respect to the exact solution itself obeys a Poisson equation $D_h\hat{\epsilon}_h = -\hat{r}_h$, where D_h is a finite difference approximation to the Δ-operator and the source term $\hat{r}_h$ consists of the residuum left in the relaxation step. By solving the Poisson equation for the error we obtain a correction to the first approximation. The error correction is done on the next coarser grid with mesh size $H = 2h$, where the residuum is restricted from $h \rightarrow H$ via a restriction operator, $\hat{r}_H = I_h^H \hat{r}_h$. It is found that the high frequency components of the error are rapidly removed on a given grid. Coarsening the grid results in the fact that low frequency components from the fine grid are transformed into high frequency components on the coarse grid, which will be efficiently removed. The hierarchy of grids is refined until only one mesh point is relaxed, which is done exactly. The calculated error is then transfered back down the hierarchy of grids, via a prolongation operator, as correction term to the field solution. This procedure, called *V-cycle*, is repeated until a threshold value for the remaining residuum is reached.

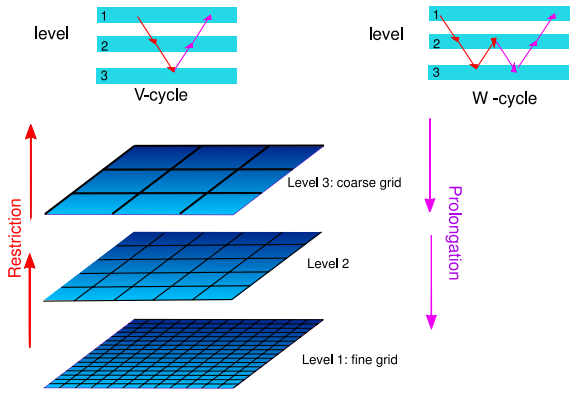


Figure 1: Steps in multigrid cycle with corresponding operators.

2.3 Discretization of the Laplacian

The Poisson equation is given by:

$$\nabla^2 u(x,y,z) = f(x,y,z) \tag{2}$$

where $u(x, y, z)$ is a field defined in 3D domain Ω with appropriate boundary conditions on $\partial\Omega$ and $f(x, y, z)$ is known as source function. One way for finding a numerical solution to the given Poisson equation is to approximate the solution to high-order by finite differences on a structured grid of size h.

The discrete form of Eq. (2) is

$$(u_{xx})_{ijk} + (u_{yy})_{ijk} + (u_{zz})_{ijk} = f_{ijk} \tag{3}$$

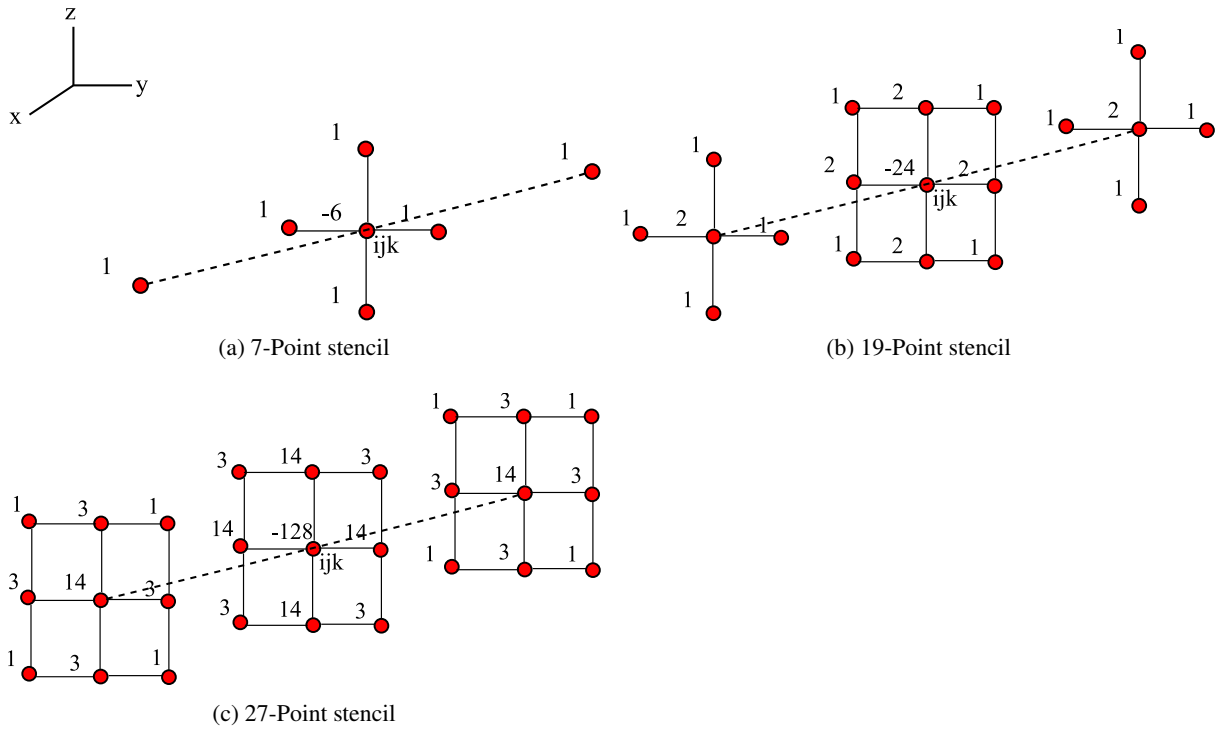
where (i, j, k) denote three-dimensional lattice indices and $(u_{\alpha\alpha})_{ijk}$ is the approximation to the second partial derivative with respect to the coordinate direction α. The simplest approximation is obtained by the following formula:

$$(u_{\alpha\alpha})_{ijk} = \frac{1}{h_\alpha^2}\delta_\alpha^2 u_{ijk} \tag{4}$$

where h_α is the grid spacing in direction α and δ_α^2 is the 2nd-order difference operator:

$$\delta_x^2 u_{i,j,k} = u_{i-1,j,k} - 2u_{i,j,k} + u_{i+1,j,k} \tag{5}$$

2.3.1 Stencil Notation



(a) 7-Point stencil

(b) 19-Point stencil

(c) 27-Point stencil

Figure 2: Stencil representation of 2nd-, 4th- and 6th-order compact schemes with corresponding matrix coefficients

For a simple usage of finite differences we introduce the stencil notation which can be a matrix-like

or a figure representation. In the matrix-like form, the coefficients belonging to the neighbouring grid points are written in squared brackets where the coefficient in the center is belonging to the actual grid itself. The figure is self-explanatory and in Fig. 2 the 7-point, 19-point and 27-point stencils are represented. For example, in 3D an approximation of $\mathcal{O}(h^2)$ of the Laplacian is given by:

$$D_h^{(2)}u(\vec{x}) = \frac{1}{h^2}[u(\vec{x}-h\vec{e}_1)+u(\vec{x}-h\vec{e}_2)+u(\vec{x}-h\vec{e}_3)-6u(\vec{x})+ \\ u(\vec{x}+h\vec{e}_1)+u(\vec{x}+h\vec{e}_2)+u(\vec{x}+h\vec{e}_2)]+\mathcal{O}(h^2) \tag{6}$$

where $D_h^{(2)}$ denotes the finite difference operator of 2nd-order.

This can be written in stencil matrix form as:

$$\frac{1}{h^2}\begin{bmatrix} & 1 & \\ & & \end{bmatrix} \quad \frac{1}{h^2}\begin{bmatrix} & 1 & \\ 1 & -6 & 1 \\ & 1 & \end{bmatrix} \quad \frac{1}{h^2}\begin{bmatrix} & 1 & \\ & & \end{bmatrix} \tag{7}$$

and it can be represented as shown in Fig. 2 (a)

2.3.2 Higher order finite difference operators

Higher-order finite difference operators are derived from the expansion

$$\left.\frac{\partial^2 u}{\partial \alpha^2}\right|_{\alpha=ih_\alpha} = \frac{4}{h_\alpha^2}[\sinh^{-1}(\frac{\delta_\alpha}{2})]^2 \\ = \frac{1}{h_\alpha^2}\delta_\alpha^2\{1-\frac{1}{12}\delta_\alpha^2+\frac{1}{90}\delta_\alpha^4-\frac{1}{560}\delta_\alpha^6\pm ...\}u_{i,j,k} \tag{8}$$

For example, the 4th-order accurate scheme can be derived from Eq. (8) if we just take into account the first two terms in the expansion:

$$(u_{\alpha\alpha})_{ijk} = \frac{1}{h_\alpha^2}\delta_\alpha^2(1-\frac{1}{12}\delta_\alpha^2) \tag{9}$$

which can be written explicitly for the x- component as:

$$(u_{xx})_{ijk} = -\frac{1}{h^2}\frac{1}{12}(u_{i-2,j,k}-16u_{i-1,j,k}+30u_{i,j,k}-16u_{i+1,j,k}+u_{i+2,j,k}) \tag{10}$$

In the end we just want to present the matrix form notation for the 13-point stencil for the 4th-order scheme:

$$\Delta_{i,0,k} = -\frac{1}{12h^2}\begin{bmatrix} 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{-16} & 0 & 0 \\ \mathbf{1} & \mathbf{-16} & \mathbf{90} & \mathbf{-16} & \mathbf{1} \\ 0 & 0 & \mathbf{-16} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 \end{bmatrix} \tag{11}$$

$$\Delta_{i,\pm1,k} = \frac{1}{12h^2}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{16} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \Delta_{i,\pm1,k} = \frac{1}{12h^2}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{12}$$

Increasing the order of the finite difference schemes implies the inclusion of neighbor grid points of increasing extent. For example, in the case of the previous 4th-order scheme one has to operate on the next-nearest gridpoint in the solver. This approach can give rise to a problem especially at those points close to the boundary. In this case it is of considerable interest to construct compact higher-order schemes, which need less informaton from neighbor grid points in space than those schemes derived from Eq. (8). In the ideal case compact schemes only need information from the next nearest grid points. Next, the fourth and 6th-order compact schemes are introduced for which the derivation can be found in Ref. [4].

2.4 HOC formulation

High order compact (HOC) solvers for the Poisson equation are based in essence on the finite difference approximation, Eq. (5), to the 2nd-order differential operator. By applying 2nd-order differential operators to both sides of the Poisson equation, introducing the finite difference operators and rearranging terms in the equation, it is possible to come up with schemes which are local on the left-hand side of the finite difference Poisson equation, Eq. (3). As mentioned before, compactness means that the operators only need information from their neighbor grid points. The construction of the HOC schemes leads on the one hand to schemes of larger locality, on the other hand the number of necessary neighbor grid points is increased, i.e. not only grid points in the cartesian directions are considered but also in diagonal directions from the central mesh point, thereby increasing the computational work. However, strong advantages of these schemes are (i) data locality which is essential for optimal use of cache and (ii) the remaining error in the approximation to the differential operator has larger isotropy.

In the following the formulation for the 4th- and 6th-order HOC schemes, as proposed in Ref. [4], is summarized.

2.4.1 4th order compact scheme

A 4th-order scheme is given by

$$[\partial_{x_1}^2 + \partial_{x_2}^2 + \partial_{x_3}^2 + \frac{h^2}{6}(\partial_{x_1}^2\partial_{x_2}^2 + \partial_{x_1}^2\partial_{x_3}^2 + \partial_{x_2}^2\partial_{x_3}^2)]u_{i,j,k} \tag{13}$$
$$= f_{i,j,k} + \frac{h^2}{12}\left[\delta_{x_1}^2 + \delta_{x_2}^2 + \delta_{x_3}^2\right] f_{i,j,k} + \mathcal{O}(h^4)$$

which corresponds to a 19-point stencil encompassing all the nodes of the mesh located on the three grid planes that intersect node ijk, but not the corner points of the surrounding cube. The stencil and matrix coefficients are illustrated in Fig. 2(b)

2.4.2 6th-order compact scheme

The $\mathcal{O}(h^6)$ compact approximation is given by

$$[\partial_{x_1}^2 + \partial_{x_2}^2 + \partial_{x_3}^2 + \frac{h^2}{6}(\partial_{x_1}^2\partial_{x_2}^2 + \partial_{x_1}^2\partial_{x_3}^2 + \partial_{x_2}^2\partial_{x_3}^2) + \frac{h^4}{30}\partial_{x_1}^2\partial_{x_2}^2\partial_{x_3}^2]\Phi_{i,j,k} = \tag{14}$$
$$f_{i,j,k} + \frac{h^2}{12}\nabla^2 f_{i,j,k} + \frac{h^4}{360}\nabla^4 f_{i,j,k} + \frac{h^4}{90}[\frac{\partial^4 f}{\partial x_1^2\partial x_2^2} + \frac{\partial^4 f}{\partial x_2^2\partial x_3^2} + \frac{\partial^4 f}{\partial x_1^2\partial x_3^2}] + \mathcal{O}(h^6)$$

where ∇^4 is the biharmonic operator. The resulting stencil involves all 27 grid points including the 8 corner nodes as can be seen in Fig. 2(c). One should note that in the case of the 6th-order compact scheme the analytical derivatives of the source term are used.

In Ref. [4] the mixed 4th-order derivatives were derived with a prefactor of $1/180$ while our calculations revealed a coefficient of $1/90$, cmp. Eq. (15). In this work both factors were investigated for the case of eigenfunctions of Laplace operator and the result proves that the correct factor is indeed $1/90$. The equation with the factor $1/180$ will be referred further as 6th-order case 1, while the one in which the $1/90$ factor was used will be referred as 6th-order case 2.

2.5 Error analysis

The numerical schemes described before generate a class of linear systems which in the exact form are described by:

$$\mathbf{A}_m\mathbf{\Phi} = \mathbf{b}_m + \sum_{p=m}^{\infty}\mathbf{c}_p\mathbf{h}_p \tag{15}$$

where $\mathbf{m}$ is the order of accuracy of the approximation scheme, $\mathbf{A_m}$ is the coefficient matrix, $\mathbf{\Phi}$ is the vector of nodal unknowns, $\mathbf{c_p}$ are the truncation error coefficient vectors and $\mathbf{b_m}$ is the right-hand-side vector. From the computational point of view we are just able to solve the equation

$$\mathbf{A_m \Phi_h = b_m} \tag{16}$$

which means that the error satisfies:

$$\mathbf{e} = \Phi - \Phi_h = \sum_{p=m}^{\infty} \mathbf{A}_m^{-1} \mathbf{c}_p \mathbf{h}_p \tag{17}$$

By dropping the higher-order terms one expect the asymptotic behavior of the error at interior node ijk to be $e_{ijk} \approx \mathcal{O}(h^m)$. That means that in a log-log plot of the error at a given node versus h would be approximately a straight line with slope m when h is sufficiently small.

2.6 Meshed Continuum Method

The idea is to replace the point charges by a charge distribution which is sampled on the mesh, meaning that the values of the charge distribution are not interpolated (translated) to grid points. This avoids introduction of errors due to interpolation but on the other hand there is an effective discretization error, which is related to the finite resolution of smoothness properties on a grid. This is related to a sampling problem, where wavelengths smaller than two grid cells cannot be resolved. As will be seen later, smoothness of charge distributions on the grid is essential for an accurate solution of Poisson's equation and therefore a minimum number of grid points under the charge distribution are required. The derivation of the method can be found in the PhD thesis of Mattias Bolten [3].

2.6.1 Point symmetric densities described by B-splines

A point symmetric density function ρ_g defined by

$$\rho_g = g(\|x\|_2) \tag{18}$$

is considered with the following properties:

- g:$\mathbb{R}_0^+ \to \mathbb{R}_0^+$
- g is sufficiently smooth
- $\int_{\mathbb{R}^3} \rho_g(x) = 1$
- solution Φ_g of $-\Delta\Phi_g(x) = \frac{1}{\epsilon_0}\rho_g(x)$ is known analytically
- g has limited support, i.e.

$$g(x) = 0 \quad \text{for} \quad x > R \tag{19}$$

In Ref. [3] a B-spline function was chosen as a symmetric point density. The B-spline function of 0th-order is the box-function, which can be considered as the composition of two Heavyside functions

$$B_0(x) \;=\; \theta(x-1/2) - \theta(x+1/2) \;=\; \begin{cases} 1 & ,-\frac{1}{2} \le x \le \frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

Higher order B-spline functions can then be constructed via the convolution properties

$$B_{i+1}(x) = 2B_{\lfloor i/2 \rfloor}(2x) * 2B_{\lceil i/2 \rceil}(2x), \qquad \text{for } i = 1, 2... \tag{21}$$

The 4th-order B-spline density is given by:

$$\rho_{B_4}(r) = \begin{cases} \frac{27\cdot(81\cdot r^4 - 54\cdot r^2\cdot R^2 + 11\cdot R^4)}{32\cdot R^7} & r \le \frac{R}{3} \\ \frac{27\cdot(-9\cdot r^2 + 6\cdot r\cdot R + R^2)(27\cdot r^2 - 42\cdot r\cdot R + 17\cdot R^2)}{64\cdot\pi\cdot R^7} & r \le \frac{2R}{3} \\ \frac{2187\cdot(r-R)^4}{64\cdot\pi R^7} & r \le R \end{cases} \tag{22}$$

which produces the potential:

$$\Phi_{B_4}(r) = \begin{cases} \frac{3645r^6 + 5103r^4 - 3465r^2R^4 + 1673R^6}{560R^7} & r \le \frac{R}{3} \\ \frac{(32805r^7 - 102060r^6R + 107163r^5R^2 - 28350r^4R^3 - 16065r^3R^4 + 9933rR^6 + 10R^7)}{3360rR^7} & r \le \frac{2R}{3} \\ -\frac{3645r^7 - 20412r^6R + 45927r^5R^2 - 51030r^4R^3 + 25515r^3R^4 - 5103rR^6 + 338R^7}{1120rR^7} & r \le R \end{cases} \tag{23}$$

3 Results

The 6th-order compact scheme was implemented into the Fortran version of the existing code PP3MG. A comparison between the 2nd-, 4th- and 6th-order schemes was performed. The implementation was tested for the two cases: (i) an eigenfunction of the Laplace operator and (ii) the 4th-order B-spline point density. The error norm for grid spacings $h = \{1/8, 1/16, 1/32, 1/64, 1/128, 1/256, 1/512\}$ was computed for the three given schemes. For the analysis the following error norm was used

$$E = \frac{\sqrt{\sum (u_{ijk} - u'_{ijk})^2}}{\sqrt{\sum (u'_{ijk})^2}} \tag{24}$$

where u'_{ijk} is the known analytical solution of the potential.

3.1 Eigenfunctions of the Laplace Operator

The first test case was implemented for solving the Poisson equation with the source term distribution being an eigenfunction of the laplacian with periodic boundary conditions:

$$f_{i,j,k} = 12\pi \sin(2\pi i h_x) \sin(2\pi j h_y) \sin(2\pi k h_z) \tag{25}$$

for which the analytical solution is given by:

$$u_{i,j,k} = \sin(2\pi i h_x) \sin(2\pi j h_y) \sin(2\pi k h_z) \tag{26}$$

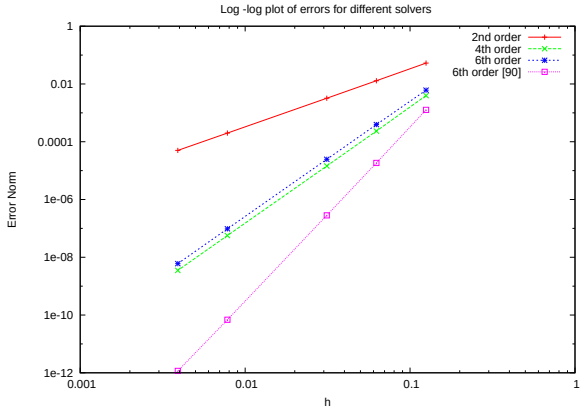


Figure 3: Convergence plot for an eigenfunction of Laplace operator

The result is shown in Fig. 3. As expected, the log-log-plot of the error versus the grid spacing shows the correct behaviour described by the theory. The slope of the functions are 2.00, 4.01, 3.99 and 6.00 for the 2nd-order, 4th-order, 6th-order case 1 and respectively 6th-order case 2. This result proves that in Ref. [4] the factor of $1/180$ should be replaced by $1/90$ and furthermore, that the implementation of 6th-order compact scheme is correct.

3.2 Charges described by 4th-order B-spline density

For the 6th-order compact scheme the 2nd-order, 4th-order and the mixed 4th-order derivatives of the source term must be computed analytically; in this work the calculations were done using Mathematica.

Three cases were tested for the given source term which are different by the number of neighbours taken into account for the width of the charge distribution. The test cases are presented in Table 1.

Fig. 3.2 shows the behaviour of the relative error of Madelungs's constant versus grid spacing h for the three before mentioned compact schemes in all three test cases described in Table 1. The behaviour of the error in the case of the new implemented 6th-order HOC, which is in all three cases lower than the 4th-order scheme, suggests that a 4th-order B-spline function is not smooth enough after applying the 4th-order differential operators. This is understandable from the fact that the B-spline functions are

Table 1: Test cases

h	test 1	test 2	test3
	[Neighbours]		
1/8	-	2	4
1/16	2	4	8
1/32	4	8	16
1/64	8	16	32
1/128	16	32	64
1/256	32	64	128
1/512	64	128	256

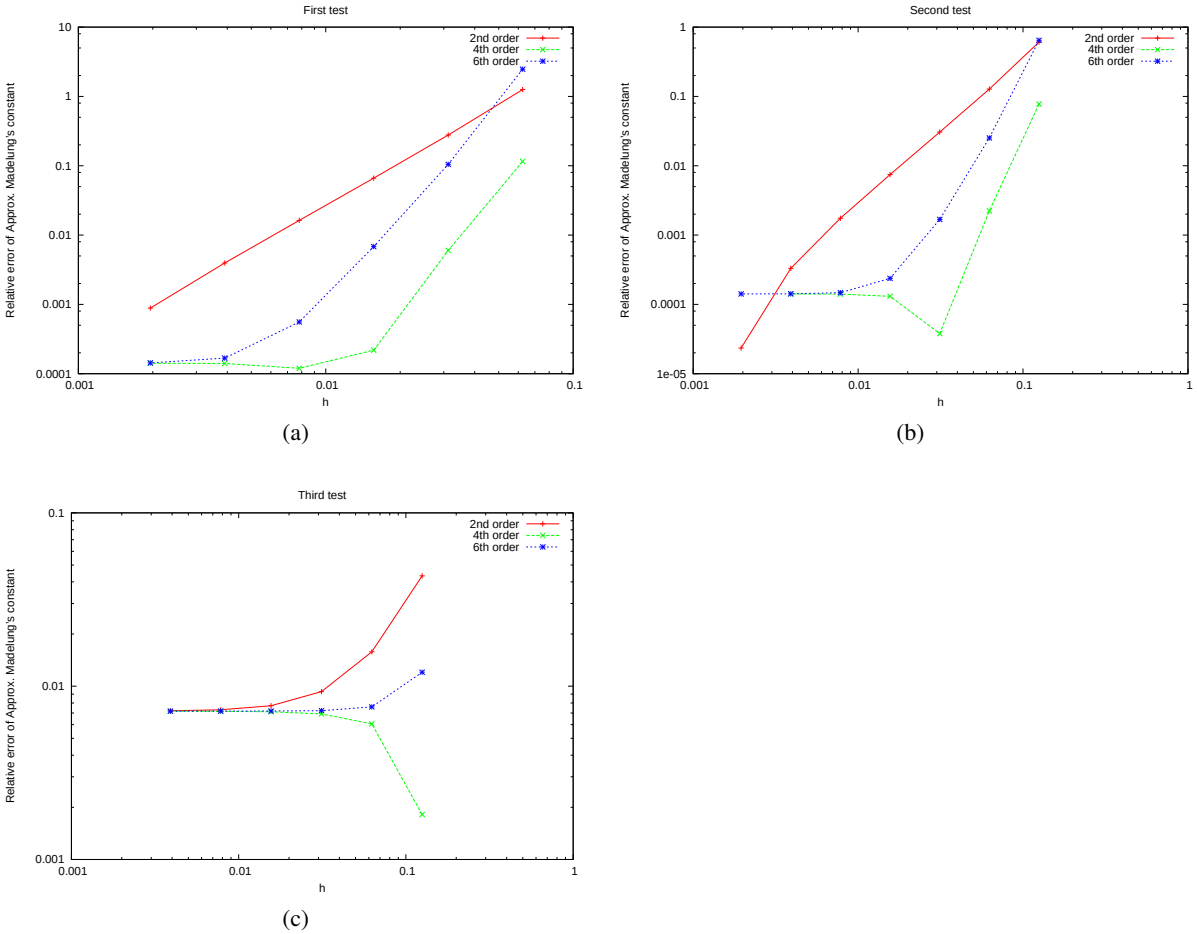


Figure 4: Convergence plot for the 4th-order B-spline symmetric point density

piecewise polynomial functions of 4th-order, where the pieces are defined in intervals, cmp. Eq. (22). Since a 4th-derivative is applied, different constant values in each partial interval appear on the right-hand side of the Poisson equation, thereby introducing step functions, i.e. non-smooth behavior and

therefore destroys the prescribed order of the solution. This observation leads to the strong request of using B-splines of order ≥ 6 for the 6th-order compact solver.

4 Conclusions and Outlook

In the present work, the implementation of a 6th-order compact scheme in the PP3MG code was accomplished. Test runs were performed for the solution of eigenfunctions of the Laplace operator and for a 4th-order B-spline charge density. The results suggest that the 4th-order B-spline is not a sufficiently smooth function for the operators of the 6th-order compact scheme. Furthermore the results obtained on the eigenfunctions for which the analytical solution is known demonstrate that the given task of an implementation of the 6th-order HOC into the existing code was successfully fulfilled. Further work should concentrate on the following objectives:

- implementation of higher order B-spline functions for the charge density
- performance gain by combining different HOC schemes hierarchically
- implementation of different 6th-order compact schemes [6]
- parallel performance measurements

5 Acknowledgments

I would like to thank my adviser Dr. Godehard Sutmann for accepting me to work in his group and for being patient with my numerous questions. Thank you to Robert and Mathias for everything they have done in the last two months and especially for having that nice smile whenever we entered their office with some scientific or less scientific problems. They were always there ready to help us. I would like to express my gratitude to my colleagues from the ”Aquarium “ first of all for tolerating me for the entire two months and for providing the wonderful nerdy/geeky athmosphere:). Especially I would like to thank Andreas, ”the google guy“ for having the infinite patience to answer at every one of my hundred questions and also Elin "the mathematics geek” sitting in front of me. Thank you again to all my colleagues for succeding of creating a pleasant social working enviroment.

References

1. G.Sutmann, B. Steffen, A particle-particle particle-multigrid method for long-range interactions in molecular simulations, Computer Physics Communications 169 (2005) 343-346.
2. U. Trottenberg, C.W. Oosterlee,A.Schüller, Multigrid, Academic Press, San Diego, 2001.
3. Multigrid methods for structured grids and their application in particle simulation, Mattias Bolten NIC Series Vol. 41, John von Neumann-Institute for Computing, 2008.
4. W.F. Spotz, G.F. Carey, A high-order compact formulation for the 3d Poisson equation, Numer. Methods Partial Differential Equations 12 (1996) 235-243.
5. Yaw Kyei, John Paul Roop, and Guoqing Tang, “A Family of Sixth-Order Compact Finite-Difference Schemes for the Three-Dimensional Poisson Equation”, Advances in Numerical Analysis, vol. 2010.
6. G. Sutmann and B. Steffen, High-order compact solvers for the three dimensional Poisson equation, J. Comp. Appl. Math., 187 (2006) 142-170.

Statistical Modelling of Protein Folding

Julie Krainau

Humboldt Unversität zu Berlin
Institut für Physik
Newtonstr. 15
12489 Berlin

E-mail: krainau@physik.hu-berlin.de

Abstract:
The open source protein folding and aggregation software ProFASi implements a physics based approach for studying protein folding and thermodynamics using Monte Carlo simulations. In this report, the main ideas of this approach are outlined, along with a qualitative presentation of various terms of ProFASi's force field, and a newly developed method for simulations with constraints. Simulation results for a simple helical peptide, a 73 residue 3-helix bundle protein and a 76 residue α/β protein will be presented and compared.

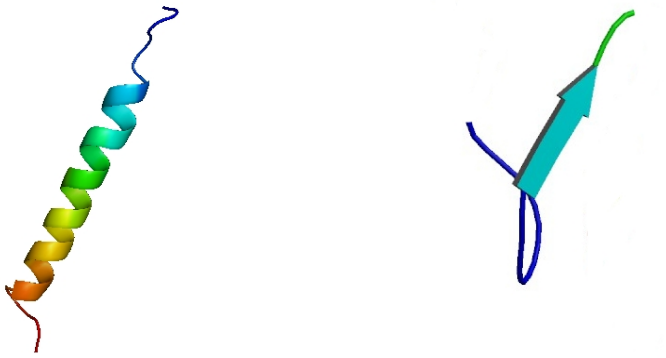
---

1 Introduction

Proteins are biological macromolecules. They perform a range of vital functions for the living cell such as catalysing biochemical reactions, regulating gene expressions or mediating cell signals. Their covalent structure is based on a chain of amino acids. Each protein is uniquely defined by its amino acid sequence, but proteins are more complex than random chains of amino acids. They exhibit the interesting property of folding into well-defined 3d-strutures called their native structures. Each instance of a protein molecule with a certain amino acid sequence folds to the same 3d-structure in the protein's natural environment.

While discussing proteins and their structures, the terms "primary structure", "secondary structure", etc. are often used. The primary structure is just the sequence of amino acids. "Secondary structure" refers to a set of universal structural motifs which occur as parts of the structure of most proteins. They are defined by specific arrangements of hydrogen bonds, and the appearance of two important secondary structure elements, the α-helices and β-sheets is illustrated in Fig. 1a and 1b respectively.

Even if one assumes that an amino acid has only 10 alternative conformations, a small protein chain of 20-30 amino acids would have an astronomical number of possible structures. In 1969, the biologist Levinthal calculated the time it would take a protein to fold if protein folding were a trial and error process. It turned out that this kind of folding would take 10^{87}s for a 100 residue protein, whereas,



(a) α-helix (b) β-strand

Figure 1: Secondary structure elements

the age of the universe is estimated to be about 10^{17}s. This gargantuan mismatch is named Levinthals paradox and led to the insight that a random conformational search does not occur. In nature proteins fold in timescales ranging from a few μs to seconds.

There are different reasons why the understanding of protein folding is an important research area. First of all, it is known that the folded shape of a protein is closely related to its function. Hence, a misfolded protein can not fulfill its primary function. Misfolding and aggregation of proteins is currently thought to be related to a number neurodegenerative diseases such as Alzheimer disease, Parkinson disease and bovine spongiform encephalopathy. Besides the scientific interest in understanding the phenomenon of protein folding, there is an economic interest, since treatment of these diseases has become a major economic burden of our times.

2 Simulation of Protein Folding

One key concept for the computer simulations of protein folding is that for proteins which fold reversibly, the folding process can be modelled as a thermodynamic process at the relevant temperatures. As the most probable state of such a system at a given temperature is the minimum of free energy, the folded structure is associated with this state at the relevant temperatures.

The first step in the usual strategy for physics based computer simulations of proteins is to formulate a model for the proteins and their interactions with an appropriate amount of detail for the intended domain of applications. For instance, to understand protein folding, the interaction between protein and its surrounding water environment is important, but the dynamics of quarks and gluons constituting the protons and neutrons inside the atoms is not. The properties of the system are then studied using Molecular Dynamics or Monte Carlo simulations – a choice which again depends on the question at hand. In this study, a protein interaction model called the "Lund force field" was used, along with Monte Carlo

simulations. The Lund force field is known to be well suited for study of large scale conformational transitions in proteins, such as folding and aggregation.

2.1 Formulation of the Interaction Potential

The interaction potential should provide a good approximation for all physical effects which might be relevant for the folding process. The Lund force field is formulated as a sum of four main terms: the excluded volume term, the hydrophobicity term, the hydrogen bond term and electrostatic interactions. In the following, we present a brief qualitative overview of these terms.

2.1.1 Excluded Volume Term

The excluded volume term ensures that atoms do not overlap spatially. This is usually modelled with the van der Waals interaction between the atoms. But in the Lund force field, a purely repulsive term is used. Chain collapse and structure formation are driven by other terms.

2.1.2 Hydrophobicity Term

The hydrophobic effect is a phenomenon familiar from everyday experiences. For instance, when a drop of oil is placed in water, instead of mixing with water, the oil stays segregated in a form minimising contact with water. Certain amino acids exhibit this sort of "fear" of water, and prefer arrangements where they are not exposed to water. The hydrophobicity term of Lund force field models this effect, and is the main contribution to the collapsing of the protein chain and the minimisation of its surface.

Introducing a non-polar material in water, reduces the number of hydrogen bonds. This leads to a reduction of available configurations of water molecules. The system is more ordered at the interface. The resulting decrease of entropy means an increase of the free energy. In effect, if the introduced molecule has hydrophobic (non-polar) atoms, it tries to hide its hydrophobic parts from water molecules, so that it can lower the reduction of configurations and avoid an high increase of free energy. Therefore, introducing a molecule with non-polar atoms in water results in a more compact shape of the molecule.
The main feature of the hydrophobicity is reducing the contact area between water and hydrophobic atoms. There are different ways to model this effect without introducing water molecules: A very costly way to get the hydrophobicty is to calculate the effective surface in contact with water. This can be done by rolling a water molecule, approximated as a sphere, over the whole molecule. It is also assumed that the atoms of the molecule have a spherical shape. Each atom is assigned to a certain value which indicates its magnitude of hydrophobicity. The effective contact area is proportional to the hydrophobic energy term. But, this is a very expensive way with rough assumptions and computationally more economic methods are desirable.

Another method for modelling the hydrophobicity term is done by giving each amino acid a certain value proportional to its hydrophobic strength. These values can be derived by comparison with experimental data. The hydrophobicity term in the Lund force field is a pairwise additive approximation

which is computationally very efficient and has been shown to work well for small proteins. The hydrophobicity term is negative, respecting that the energy of the protein is lowed when hydrophobic parts are hidden.

2.1.3 Hydrogen Bond Term

Hydrogen bonds are a result of electrostatic interaction between a hydrogen atom bound to an electronegative atom and another electronegative atom. In the Lund force field, it is modelled as an interaction between two electric dipoles, one of which contains a hydrogen. The dipole with the hydrogen atom is called the donor and the other dipole is called the acceptor. The current version of the Lund force field only models hydrogen bonds between NH and CO dipoles.

The distance dependence of the hydrogen bond term is modelled with a 12-10 Lennard-Jones term in the distance between the hydrogen and the negatively charged atom of the acceptor. The lowest energy state of a dipole-dipole system corresponds to the conformation where the dipoles are aligned. For taking the orientation dependence into account, a factor $(\cos\alpha\cos\beta)^2$ is multiplied with the Lennard-Jones potential. The angles α and β are the N–H–O and the H–O–C angles respectively.

2.2 Metropolis Monte Carlo Method

The interaction potential determines the so called "energy landscape" for a given protein. The dimension of this energy landscape is the number of degrees of freedom in the system. The average properties of the system for the interaction potential can be calculated if one generates a large number of states such that the probability of occurrence of a state in the sample is related in a simple way to its correct thermodynamic probability. For this purpose, we have used methods based on the classic Metropolis Monte Carlo method. The algorithm is composed of the following simple steps:

1. Algorithm starts off with an initial conformation, associated with energy E_{old}
2. One degree of freedom is changed randomly and a new conformation is created with energy E_{new}
3. Calculate $\Delta E = E_{old} - E_{new}$
4. New state is accepted with a probability of $\min\left(1, e^{-\frac{\Delta E}{kT}}\right)$
   - If the new state is accepted, set $E_{old} = E_{new}$
   - If the new state is rejected, reverse E_{new}
5. Continue with step 2

For an infinite number of steps the global minimum will be found.

2.3 Parallel Tempering

As the acceptance probability contains the temperature as an argument, the temperature is related to the size of steps which can be made while exploring the energy landscape. This would not cause problems if the the energy landscape were smooth. However, as the energy landscape of biological macromolecules like proteins is very rough, the global minimum cannot be found in a acceptably small number of steps.
The problem is that a low temperature implies a small step size, hence it can happen that system gets trapped in a local minimum. If a higher temperature is chosen to make big steps within in the energy landscape, it is very likely that the simulation leaps over interesting minima.
The parallel tempering is one of the popular modern methods to improve sampling on rough energy landscapes. Its main idea is to have N replica of the target protein and start independent MC runs for each replica at different temperatures T_i. After a predefined number of iterations the conformation of a pair of different of adjacent runs is exchanged with the following probability:

$$p = e^{\Delta\beta\Delta E} \tag{1}$$

This ensures that the system remains in equilibrium. In effect, each conformation performs a random walk in the temperature space and the number of conformations at each temperature remains fixed.

3 ProFASi

ProFASi [1] stands for Protein Folding and Aggregation Simulator and is an open source C++ package for Monte Carlo simulations. Originally, it was developed for small peptides of 20-30 residues. But it has been shown that the latest version of ProFASi's force field described the folding and thermodynamic properties of 20 proteins of sizes between 10-67 residues with different secondary structure elements [2].
It uses an intermediate resolution model: all atoms of the protein are explicitly represented, bond lengths and bond angles are kept fixed. The solvent is represented by an implicit water interaction potential.
ProFASi uses different kinds of tricks for improving the performance. For instance, the cut-off and cell list method are used for the calculation of the most expensive of the energy terms. Another distinguishing feature is that at each MC step, only the energy of the changed parts is calculated. This is allowed, because MC simulation needs only the energy difference between two states.

4 Results

My task during the Guest Student Programmme was to test ProFASi with proteins of different lengths in order to find out how the existing force field can be improved. The superior aim is to formulate a force field that is able to simulate folding of more proteins. All simulations ran on JUROPA at JSC. The proteins that were chosen for testing consist of 29 to 76 residues and have α-helices as well as β-sheets. For finding them and getting data of their native conformation, the protein data bank www.rcsb.org was

used.
Analysing protein folding simulations needs a measure of how close a simulated structure and experimentally measured reference structure are. A commonly used measure for comparing two structures is the minimised root-mean-square-deviation (RMSD) over all translations (T) and all rotations (R)

$$RMSD = \min_{\{T\},\{R\}} \left(\sqrt{\frac{1}{N} \sum_{i=1}^{N} |\vec{r}_i - \vec{r}_i^{\,ref}|^2} \right) \tag{2}$$

The parameter r_i stands for the coordinates of the i^{th} atom and $\vec{r}_i^{\,ref}$ is the position of the i^{th} atom in the reference structure.

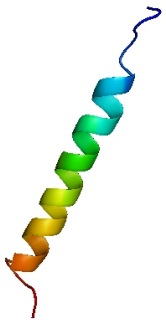
4.1 1GCN

The protein 1GCN consists of 29 residues, 471 atoms and has 135 degrees of freedom. Its simulation used parallel tempering with 16 temperatures in a range of 270 to 370 Kelvin. The simulation ran on 16 processors for two days and performed 10.000.000 MC sweeps. One MC sweep consists of as many elementary MC steps as there are degrees of freedom in the system.

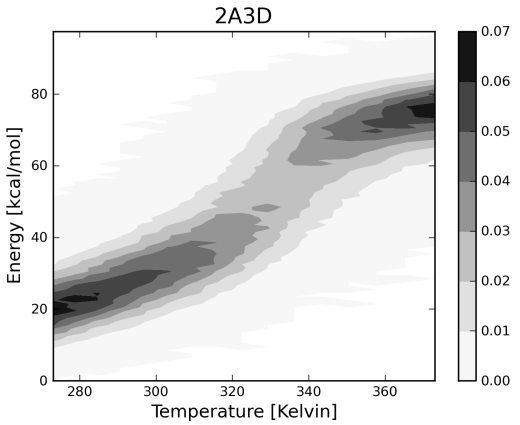
As it can be seen in Figure 2a, 1GCN has one α-helix. The plot in Figure 2b shows the population during the simulation dependent on energy and temperature: There are two distinct peaks observable. One at a lower energy of 20 kcal/mol and a second one at higher energy of 73 kcal/mol. Therefore, it can be said that the protein occupies two different states during the simulation. This indicates that the protein has undergone a transition from a high to a low energy state.
None of the replica got stuck in a low energy state. As example for this, for one replica the energy is plotted for up to 2.000.000 MC sweeps in Figure 2c. The plot in Figure 2d shows a peak in the heat capacity which indicates a transition.
It can be seen that most of simulated structures have an RMSD in the range of 9-12 Å (Fig. 2e). At the lowest temperature, the native population was around 23 %.

4.2 2A3D

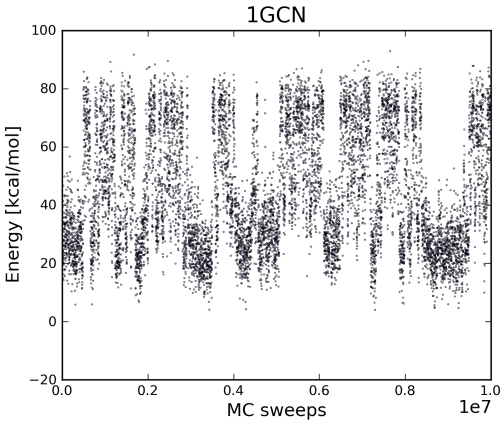
The protein 2A3D (Fig. 3a) is a designed 3-helix bundle protein, consisting of 73 residues. It contains 1140 atoms and has 333 degrees of freedom. Its folding has been simulated with 32 different temperatures between 273 to 373 Kelvin, using 1024 processors. The simulation ran one day and performed 1.300.00 MC sweeps.
As it is shown in figure 3b, there were broadly two energy states seen in the simulation. By looking at the plot in figure 3c, one can observe that there is maximum of folded structures at a low RMSD of 4 Å. Thus, most of the replica have folded into the correct structure. A better understanding of the folding process can be gained by analysing the helix content plot in figure 3d: It is significant that the central helix is less stable than the other two helices. Moreover, there is evidence that the C-terminal helix folds at lower temperatures, compared to the other two.
The lowest energy structure has a RMSD of 3.7 Å, this can be noted as a good result for a protein of



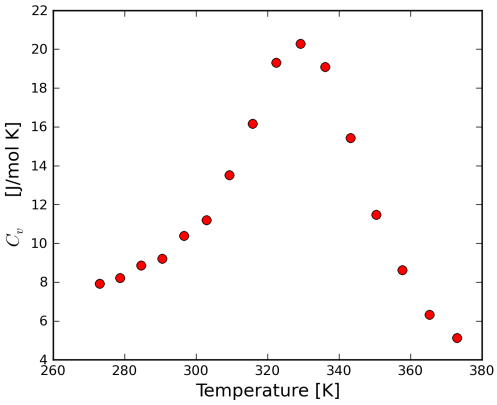
(a) 1GCN: Native structure



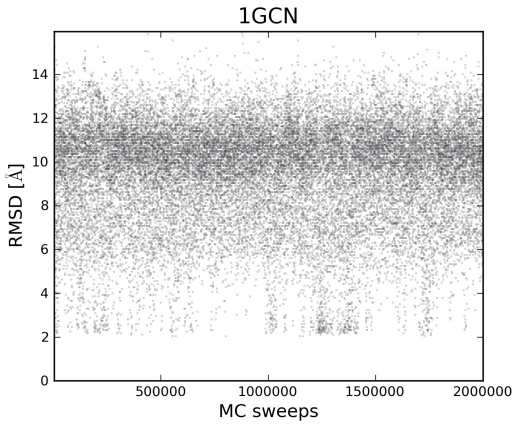
(b) 1GCN: The energy as a function of temperature, grey scale coded with population



(c) 1GCN: Run time history of energy for one replica

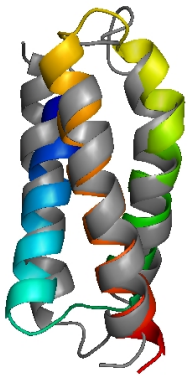


(d) 1GCN: Specific heat as a function of temperature

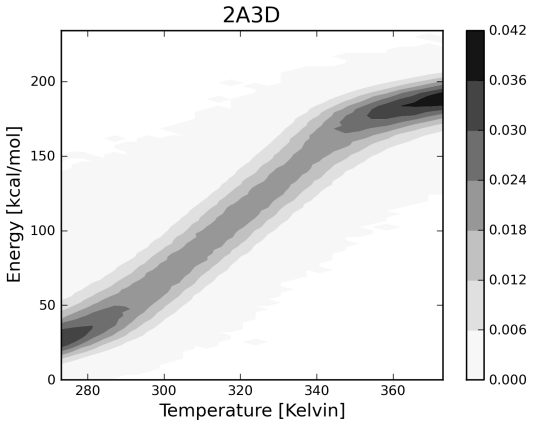


(e) 1GCN: Run time history of RMSD

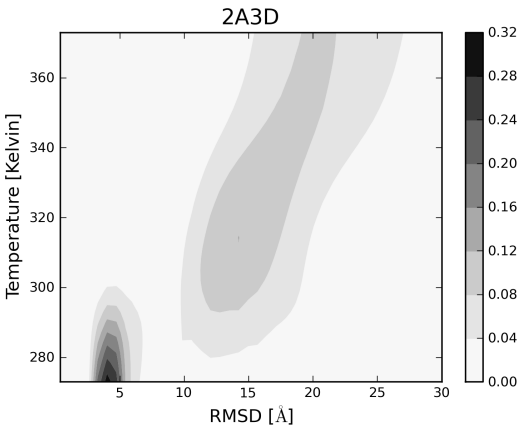
Figure 2: 1GCN



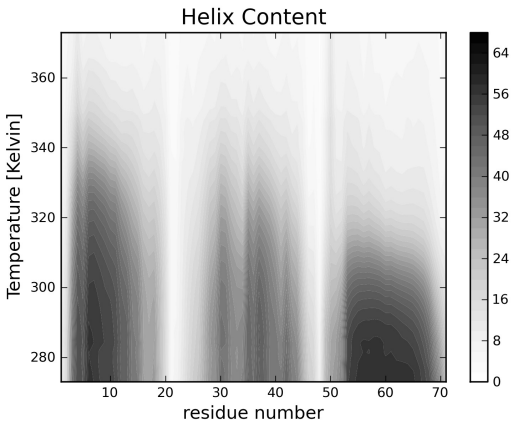
(a) 2A3D: Overlap of native (grey) and simulated structure (coloured)



(b) 2A3D: The energy as a function of time, grey scale coded with population



(c) 2A3D: RMSD as a function of temperature, grey scale coded with population



(d) 2A3D: Helix content for each residue in per cent coded by grey scale, dependent on temperature

Figure 3: 2A3D

this size. Since, the lowest energy structure is very similar to the experimental native state, it is shown that the force field worked very well for this protein.

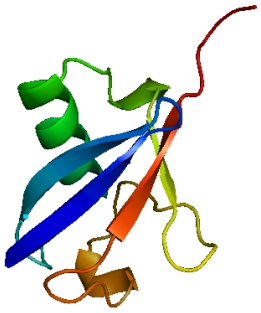
4.3 1UBQ

The protein ubiquitin, 1UBQ, is a natural protein of 76 residues. It has 1231 atoms and 368 degrees of freedom. As secondary structure elements, it has both helices and β-strands. The two N-terminal β-strands form a hairpin. Replica-exchange Monte Carlo simulations of ubiquitin were performed using 32 temperatures ranging from 273 to 400 Kelvin on 64 processors running for one day. Each generated trajectory had 1.679.000 MC sweeps.
In figure 4b, it can clearly be seen that the proteins have folded during the simulation. However, looking the RMSD-temperature plot in figure 4b shows that at low temperatures only large RMSD values were populated. Besides, analysing individual runs revealed that, so far, none of the replica folded into the correct structure. Comparing the low energy structure 4c with the native structure indicates that the protein has folded an α-helix at the C-terminus, instead of a β-strand.
A comparison between the helix content plot (Fig. 4d) and the β-strand content plot (Fig. 4e) shows that the α-helix is more stable then the hairpin at lower temperatures.
It is possible to understand the observed misfolded structures from an energetic point of view. The simulation suggests that the hairpin and the α-helix being thermodynamically more stable are more likely to form first, followed by the sequence adjacent β-strand. Due to development of hydrogen bonds, the β-strands are attached to each other. Therefore, the C-terminus does not have a place to arrange between them. In this case, it is energetically favourable to fold into an α-helix.

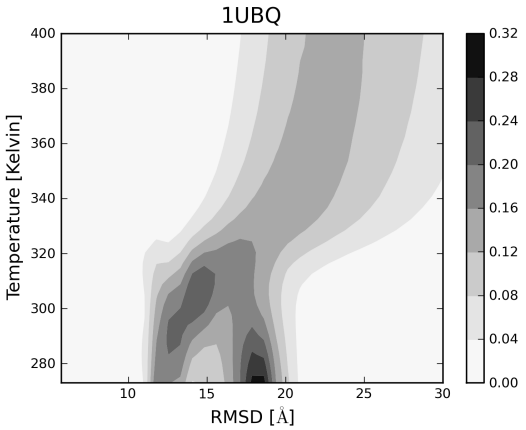
Clearly, in nature ubiquitin avoids such an event. It seems that the order of formation of secondary structure elements must be different. A possible hypothesis is that the blocking β-strand is hidden until the C-terminus β-strand has formed and is attached to the hairpin. This could happen if a long α-helix is formed at the C-terminus including the residues of the third strand. The next step would be breaking the end part of the α-helix, so that it folds into the missed β-strand. When it gets attached to the hairpin, the rest of the α-helix can rearrange into the remaining β-strand. Precisely, this sequence of events was observed in an earlier simulation with the same model [PNAS, 105(23) 8004, 2008]. In that article the authors proposed temporary caching of β-strands in adjoining helices as a mechanism involved in the formation of complex β-strand arrangements.

4.4 Simulation with Constraints

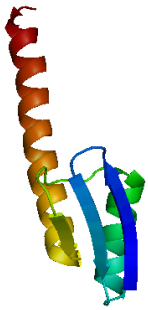
In the case that some features of the molecule are already known, this previous knowledge can be used as constraints. This is useful for making the simulation more efficient and save computing time. Another situation where constraints can be used is given if the simulation did not find the native structure. Then, using constraints on some angles can help find out what went wrong in the simulation. At best, this may give hints about possible improvements of the force field. There are different ways to use constraints: Dihedral and distance constraints.



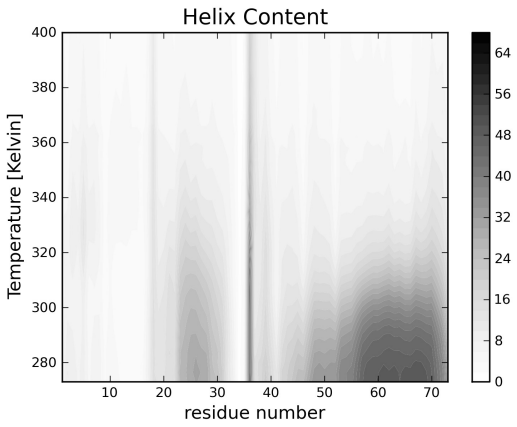
(a) 1UBQ: Native structure



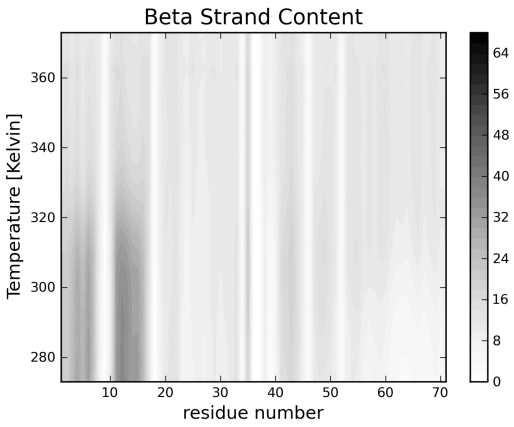
(b) 1UBQ: RMSD as a function of temperature, grey scale coded with population



(c) 1UBQ: Lowest energy structure obtained by simulation



(d) 1UBQ: Helix content for each residue in per cent coded by grey scale, dependent on temperature



(e) 1UBQ: Beta strand content for each residue in per cent coded by grey scale, dependent on temperature

Figure 4: 1UBQ

4.4.1 Dihedral Constraints

Dihedral constraints restrict one or more dihedral angles to desired ranges. This can be achieved in two ways in ProFASi: by using an interaction distribution or by choosing angles directly from a modified potential. The second method is faster and is preferred for such constraints. One possible distribution is the Von Mises distribution, it is a circular normal distribution:

$$P(\Phi) = \frac{\kappa \cos(\Phi - \Phi_C)}{2\pi I_0(\kappa)} \tag{3}$$

The Von Mises distribution is the circular analogue to the gaussian distribution. Φ_C is analogue to the mean value μ and $1/(\kappa)$ is the analogue to the variance. If κ is zero the distribution is uniform. If κ is large the distribution becomes very concentrated around the angle Φ_C. By choosing the value for κ the strictness of the applied constraints can be adjusted. The $I_0(\kappa)$ is the Bessel function of order 0.

Additionally, it is possible to give the degrees of freedom weights, in the sense that the ones with a higher weight are changed more often.

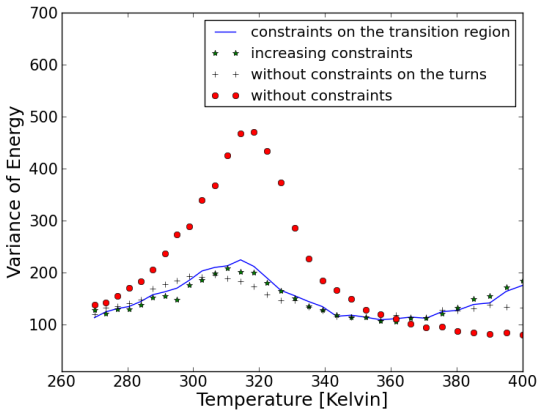
4.4.2 Distance Constraints

Distance restraints are used to favour predefined separations between a set of atom pairs. In the case that the secondary structure elements are well formed, but the tertiary structure is incorrect, it can be helpful to set distance constraints. This can be done by adding an additional term to the interaction potential that has its minimum at the favoured distance. If this trick helps to find the correct conformation of the protein, it hints at possible improvements of the interaction potential.

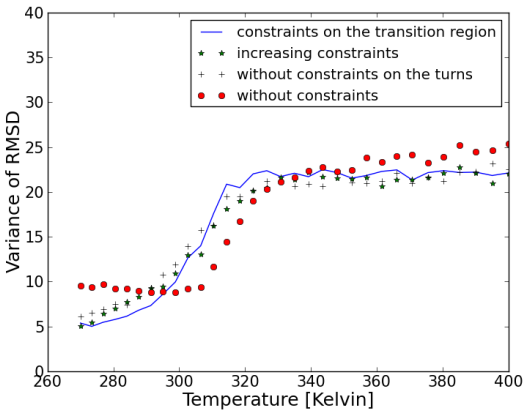
4.4.3 Testing Different Constraints for the Simulation of 1UBQ

To see if our hypothesis might lead to the folding of ubiquitin, despite limitations of our model, we experimented with simulations with constraints. There are different approaches to guide 1UBQ into its native structure by using constraints. The part of the protein that works well does not need to be simulated again. So, it is reasonable to set constraints on this part, so that more computing time is spent in the difficult part of the protein. For this protein, the hairpin and the long α-helix are restrained. The constraints on the turns are less strict than the ones on secondary structure elements.
In a second approach, two kinds of temperature dependent constraints are tested for 1UBQ: One option is to set temperature dependent constraints, so that with increasing temperature the strictness of the constraints decreases. The other option is to set stronger constraints at the transition temperature. The aim of these approaches is to find the minimum set of constraints which are sufficient for getting the protein folded. That could help to test the reliability of our hypothesis as well as giving us a clue to an improved force field. When using constraints, one has to keep in mind that constraints make it more difficult for the simulation to escape from a local minimum. This behaviour must be compensated by

increasing the maximum temperature.



(a) 1UBQ: Variance of energy as a function of temperature



(b) 1UBQ: Variance of RMSD as a function of temperature

Figure 5: 1UBQ

All the constraints simulations used the same settings as the unconstrained one and each ran for one day. The comparison between the different runs has been made by plotting the variance of RMSD as well as the variance of energy as a function of temperature (Fig. 5b, Fig. 5a). The resulting plots led to the insight that the different constraints guide the folding process into the same direction. Regarding the variance in energy it can be seen that the constraints suppress the variance in the transition region. It appears that in our simulations, constraints inhibit rather than help correct folding of ubiquitin. Applying constraints on all secondary structure elements makes it difficult for β-strands to arrange themselves in a native like fashion. It is possible that the conformation change of the residues of a β-strand to the corresponding region of the Ramachandran plot proceeds concomitantly with formation of β-sheet hydrogen bonds. Our hitherto inconclusive results speak against the use of dihedral restraints on β-strands.

5 Conclusion

We got excellent results for the folding of the 73 residue designed 3-helix bundle protein, 2A3D, making it the largest protein folded with this model. The protein folds to the correct native state at physiologically relevant temperatures with almost 100 % native population below 300 Kelvin. Whereas, for the smaller 1GCN peptide, the native population was around 20 per cent at 280 Kelvin. Natural proteins, like ubiquitin, constitute a much greater challenge. In order to test what it would take for a molecule like 1UBQ to fold, we tested a newly developed method to do simulations with temperature dependent constraints. Comparing the different constraints methods revealed no significant differences among them for ubiquitin. This result speaks against the use of such constraints in simulations intended for structure prediction, especially if complex β-sheets may be present. It also supports the idea that perhaps complex β-sheets can not form by assembling pre-formed β-strands.

Acknowledgement

First of all, I would like to thank my advisers Sandipan Mohanty and Jan Meinke for their outstanding supervision. Furthermore, I wish to thank Robert Speck and Mathias Winkel for the great organisation of the JSC Guest Student Programmme. Finally, I thank my fellow guest students for the good time we had.

References

1. A. Irbäck, S. Mohanty, PROFASI: A Monte Carlo simulation package for protein folding and aggregation, J. Comput. Chem.27, 1548-1555, 2006
2. A. Irbäck, S. Mitternacht, S. Mohanty, An Effective All Atom Potential for Proteins, PMC Biophysics 2, 2009
3. D. Voet, J. Voet, C. Pratt, Lehrbuch der Biochemie, WILEY-VCH Verlag GmbH, 2010
4. S. Wallin, Physical Modeling of Protein Folding, Lund University, Department of Theoretical Physics, 2003
5. G. Favrin, Statistical Modeling of Protein Folding and Aggregation, Lund University, Department of Theoretical Physics, 2004

Domain Distribution for Parallel Modeling of Root Water Uptake

Martin Licht

Universität Bonn
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Numerische Simulation
Wegelerstr. 6
53115 Bonn

E-mail: mlicht@uni-bonn.de

Abstract:
Towards a parallel simulation of water transport in coupled soil-root systems, we analyze several strategies for soil domain distributions aligned to root geometries. Our results tentatively point out potential for a well-scaling simulation, when combined with adaptive mesh refinement and multithreaded root simulation. We recognize technical and conceptual questions that might emerge along this direction. For our investigations we enhanced the MPI-program parSWMS by a basic root model.

1 Introduction

Within the eukaryotic domain, plantae are a regnum of great diversity. Although they are ubiquitous and of tremendous importance for ecosystems and human culture, there are still many fundamental questions which are left to investigate.

One particular biological process common to most plants and outstandingly vital for their biology is the interaction of the plant organism with the soil. This takes place at the roots, and the uptake of water is the most important process there. Understanding it is a matter of ongoing research and also has numerous industrial applications, e.g. to develop more efficient ways of watering agricultural fields, or avoid unnecessary contamination by pesticides.

While mathematical models within biology and agricultural sciences have matured during the recent decades, the extensive employment of computer simulations has been introduced to those areas of research. At the Research Center Jülich, a cooperation between the Jülich Supercomputing Center (JSC) and the Institute of Chemistry and Dynamics of the Geosphere IV (ICG IV) exists, where researchers combine experiments, modeling and simulation, to deepen the knowledge of this fundamental aspect of plant biology.

2 Numerical models

The process of root water uptake encompasses several coupled dynamics from physics and biology. Because plant roots reside within the soil, the water transport there has to be understood sufficiently well. The water transport within the root network is treated separately from the soil system.

The following explanation outlines mathematical and numerical models of both the soil system and the root system, and shows how these are coupled on a numerical level. These models can be extended to model several further physical processes – e.g., the transport of solute (like nutrites) within the water. As much as it is important in order to understand the plant biology, its influence on the water dynamics can be neglected in numerical simulations.[1]

2.1 Water transport within soil

The water content throughout the soil domain is denoted by θ, and is in bijective relation to the pressure head, h. The model of Van Genuchten [5] states their relationship as

$$\theta(h) = \theta_a + (\theta_m - \theta_a)(1 + |\alpha h|)^{-m}. \tag{1}$$

Here, θ_a, θ_m, α, and m denote parameters. The water transport within the soil domain is then described by the Richards equation [8]

$$\partial_t \theta = \nabla(K(h)\nabla h). \tag{2}$$

The term K represents the conductivity tensor and is defined via

$$K(h) = K_S(1 + |\alpha h|)^{-ml}(1 + (1 - (1 + |\alpha h|)^{\frac{mn}{n-1}})^{\frac{n-1}{n}})^2, \tag{3}$$

where K_S, n and l denote further material parameters.[2] We introduce the uptake by an additional sink term S. The extended Richards equation reads

$$\partial_t \theta = \nabla(K(h)\nabla h) + S(h). \tag{4}$$

The sink term S is basically a reaction term, and is of the form

$$S(h) = \alpha(h)\beta R. \tag{5}$$

Here, R denotes the flux at the root collar. $\alpha(h) \in [0, 1]$ denotes a water stress response function [2] - it describes how much of its uptake capacity a root segment is able to employ, and we will inspect its form closer below. $\beta \in [0, 1]$ describes to how much a segment can contribute to the over-all uptake, and, in fact, corresponds to the fraction of the segments lengths to the summed length of the whole network.

[1] Our overview follows [17], [2] and [10].

[2] For porous medium equations in a more general setting, see [9]

Let P_0, P_1, P_2 and P_3 be material parameters. The term α is then described [6] by

$$\alpha = \begin{cases} \frac{h-P_3}{P_2-P_3} & \text{for } h \in [P_3, P_2] \\ 1 & \text{for } h \in [P_2, P_1] \\ \frac{h-P_0}{P_1-P_0} & \text{for } h \in [P_1, P_0] \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

2.2 Numerical solutions for soil water transport

The numerical solution method, as implemented both in R-SWMS as in parSWMS (see below), is a modified Finite Element approach [12]. The soil domain, in case of a rectangular box, is parted into cuboids – each of these is then, in a second step, parted into tetrahedrons.[3] [4] We employ a Finite Element Scheme on this domain, with a spatially discretized version of the relevant functions. A backward Euler Scheme is used to perform time steps.

If we deviate from the ordinary FEM-approach slightly, we can handle the non-linearity of the Richards equation and obtain a stable numerical method.

First, the chosen numerical method utilizes a Picard iterative scheme [10]. At each iteration step, the coefficients of the equation are evaluated with the water content of the previous iteration (for the first iteration, this is extrapolated from previous time steps), and then the equation is solved. This iterates until a good solution is found, and a time step can finally be performed.[5]

Second, the method implements modification by Celia et al. [10]. This is the reason why the left-hand side of (1) is expressed in terms of θ.[6]

2.3 Water transport within roots

Doussan et. al. have developed a model for water transport within root networks [1]. The root segments are modelled by 1-dimensional line segments, whose widths are material parameters. This makes sense, as we expect radial homogeneity for the flow dynamics.

Let ψ_x be the water potential within the roots, and ψ_s be the soil water potential along the roots. With z being the vertical coordinate, the time evolution of the water potential within the setting is reduced to the description of the flux within the roots, J_h, and the radial flux between root and soil, J_r:

$$J_h(z) = -K_h \partial_z \psi_x, \tag{7}$$

$$J_r(z) = L_r(\psi_s - \psi_r). \tag{8}$$

[3] The chosen root water uptake model requires the initial partition into rectangular boxes.

[4] R-SWMS and parSWMS part the cubes according to slightly different schemes. Furthermore they are able to handle more general geometries than simulation boxes. For these, an uptake model has yet to be developed and implemented.

[5] Consult the conclusions for a development perspective to replace the Picard-Iteration by a better scheme.

[6] For further details, we refer to the SWMS_3D-Manual.

In (7), K_h is the axial conductance of the in-xylem flow, and L_r forms the radial conductivity in (8).

2.4 Numerical solutions for root water transport

For the numerical simulation, we employ a discretized version of the root network and the surrounding soil. Doussan et al. model the network as a combinatorial graph with a tree structure. Each root node is furthermore hydrodynamically coupled with the soil node that represents the soil portion it is located in. We describe it here, since our investigations have been directed towards this mid-term goal. Its parallel implementation has been out of this works scope.

The time evolution is given in terms of a set of difference equations, with J_h, J_r, K_h, L_r and ψ_x replaced by functions on this discrete model. The equations read

$$J_h(z) = -K_h \frac{\psi_x}{\triangle z}, \tag{9}$$

$$J_r(z) = L_r(\psi_s - \psi_x). \tag{10}$$

Therewith we can compute the evolution if we solve linear systems of equations, as we now show. Let N_c be the number of nodes within the tree (s.t. the number of links between the nodes is $N_c - 1$). Let the incidence matrix of the root network be given by $N \in \{-1, 0, 1\}^{N_c(N_c-1)}$,

$$N_{ij} = \begin{cases} 1 & \text{if there is a link from } i \text{ to } j \\ 0 & \text{if there is no such edge} \\ -1 & \text{if there is a link from } j \text{ to } i. \end{cases} \tag{11}$$

With $\psi \in \mathbb{R}^{N_c}$ being the water potential vector, we define the difference vector $d\psi = N\psi \in \mathbb{R}^{N_c-1}$. If $D \in \mathbb{R}^{N_c-1}$ describes the flux vector (see below), then we obtain (as there is no loss or creation of water between nodes)

$$N^t D = 0 \in \mathbb{R}^{N_c}. \tag{12}$$

The vector D itself obeys $D = Kd\psi$, where

$$K_{ij} = \begin{cases} \frac{-K_h}{\nabla z} & \text{for } i = j \\ 0 & \text{otherwise} \end{cases}. \tag{13}$$

On the other hand, we see from $J_r(z) = L_r(\psi_s - \psi_r)$, with L and J_r being diagonal coefficient matrices,

$$L\psi_x = J_r - L\psi_s. \tag{14}$$

Combining these, we obtain the linear system of equations $C\psi = Q$, where

$$C = NKN^t + L, \tag{15}$$
$$Q = (Jr + L\psi_z). \tag{16}$$

For humidity-independent material parameters, it can be solved by any common LSE-solver. For a performant numerical approximation, it is appropriate to use an algorithm which takes into account the sparse structure of the matrix. Doussan et al. have employed a preconditioned conjugate gradient method.

2.5 Coupling of both systems

Whereas the presence of roots in effect causes the sink term S in the Richards equation to be non-zero, the water content in the soil that surrounds the root segments serves as a numerical source for the boundary conditions. For simulation experiments, we can employ the full interaction pair, or dismiss either of them. R-SWMS solves the two system in turn, until a solution with acceptable error is found [11].

To understand the uptake model, we recall that each node of the root graph is contained within a geometric cuboid, and interacts numerically with the cuboid's corner nodes. For this the β-field on the mesh nodes has to be set appropriately.

For each Cuboid C we are given its volume $Vol(C)$, and we identify it with the set of its corner nodes. Let $\mathfrak{R}$ be the set of all root segments, and for each $r \in \mathfrak{R}$, l_r denotes the length of a root segment, and $C(r)$ the cuboid r is located in. For each mesh node x, let $\mathcal{R}(x)$ be set of all root segments that reside within a cuboid which has x as one of its corners. $d(x, r)$ will denote the geometric distance between mesh node x and root node r. First we define the field β' on the soil mesh by

$$\beta'_x = \sum_{r \in \mathfrak{R}(x)} \frac{d(x,r)}{\sum\limits_{c \in C(r)} d(c,r)} \cdot \frac{l_r}{\sum\limits_{s \in \mathfrak{R}(x)} l_s} \cdot R. \tag{17}$$

In order to obtain a β-field whose integral is R again, we finally set $\beta = \dfrac{\beta'}{S}$, where

$$S = \sum_{C \text{ Cuboid}} Vol(C) \sum_{x \in C} \beta'_x. \tag{18}$$

3 Software developments

We briefly review the software on which the developments have been based, and render the historical relations between these.

- The SWMS_3D code has been developed by J.Šimůnek, K.Huang and M.Th.van Genuchten in

1992 [2], written in FORTRAN 77. It simulates water and solute transport within soil geometries. It provides no root model, but the sink terms may be set to fixed values according to an input file.

- R-SWMS, as written by Prof. Mathieu Javaux et al. at the ICG IV again in 2008, is an enhancement to SWMS_3D, which can simulate complete soil-root systems. It comprises the Doussan model and implements root growth.
- parSWMS is a development of Dr. Horst Hardelauf e.a. [7] at the ICG IV from 2006. It covers the same the functionality as SWMS_3D. The programming language changed from FORTRAN to C++, and it computes in parallel. It has been shown to provide near optimal scaling up to 32 processes.

parSWMS is the program we have mainly worked with. It utilizes two notable libraries. On the one hand, the parallel numerics are completely shielded by the library PETSc [14]. On the other hand, and relevant for this project, it uses parMETIS [13], to find an optimal distribution of the elements and nodes amongst the processes.

4 Extensions of parSWMS

The joint-project at the JSC and ICG IV aims at developing a parallelized simulation of the whole soil-root network. During the JSC-GSP, the objectives have been to inspect different approaches towards this medium-term goal, and derive conclusions which of these might be suitable. The particular objectives have been:

1. Implement the datastructures to model the root network in parSWMS.[7]
2. Implement a root water uptake model.
3. Benchmark the performance of several strategies to influence the soil distribution with respect to a given root geometry.

4.1 Implementation of the root data structures

Within the Doussan model, the root networks are modeled as directed trees (see Figure 1). Each node of the tree is assigned several parameters (e.g. position, age, surface) and does reference, if possible, its parent node (to describe the topology of the root network). As this graph is a tree, we can identify each node (except the few top nodes) with the edge which points to it, which is the concept underlying the R-SWMS input files.

R-SWMS distinguishes three kinds of nodes that constitute the root network. These have slightly different semantics, and are given by different lists in the input files. Since they share many common properties, they have been implemented in parSWMS by an unified concept. We briefly review them:

1. seed nodes: The seeds, from which the root segments initially originate. These nodes do not have parent nodes.

[7] Each process manages its own complete copy of the root network, because we did not investigate how to parallelize the root system model.

2. segment nodes: The mediate segments of the root network. Each has a parent node, and at least one child node.
3. branching tips nodes: The tips of the root network, where growth takes place. These have no children.

To obtain the topological description of the root network, it is sufficient to know each node's parent node (or whether it is a top node). If we maintain redudant data, we can employ asymptotically better algorithms, as outlined below. The code has been structured in a way such that it allows easy migration between these implementations, adapted to the respective models.

Let $N \in \mathbb{N}_0$ be the number of root nodes.

- Memory-minimalist approach: If each segment refers to its parent segment, the network topology can be described by the least possible overhead of N indices. This is easy to implement, but many operations require complexity quadratic in N, because one typically runs a nested pair of loops.
- index-oriented approach: Each segment refers to its parent. Furthermore, it is an element within a doubly-linked list which contains the child nodes of its parent. It refers to its predecessor and successor in that list, and to the first element of its own child list. References may be null, to indicate absence of the respective root segments. The total overhead is $4N$ indices. If this structure is maintained, all common operations can be performed with asymptotically optimal runtime.

Both these approaches have been implemented. The index-oriented approach is more complex, but a performance increase has been found in comparison to the memory-minimalist implementation.[8]

Details on the data structures and routines can be found in the Technical Report, together with a review of the reasons on which the design decisions were made.

4.2 Implementation of root water uptake

parSWMS can utilize a fixed β-field and calculate the sink terms accordingly. But towards an implementation of the Doussan model, it is necessary to calculate the β-field in accordance to a given root geometry.

The algorithm developed is based on the R-SWMS routines and implements the coupled model by Schröder [3] (see Figure 2). The implementation developed works for regular box-shaped grid geometries, which suffice for many simulation settings.[9] The β-field is assembled and normalized at each time step.[10] Segment parameters other than length have not been relevant for our investigations.

In case the distance between neighboring soil nodes is markedly shorter than the typical segment length, as for fine or refined grids, the segments will pass through numerically unaffected intermediate cuboids.

[8] It seems recommendable to separate the root segment's material properties from the topological information, and to maintain a code structure which allows both these to be changed and developed independently from one-another, maybe even to be changed dynamically during run-time. This eases migrating from one implementation to the other, if necessary.

[9] The more general case of arbitrary geometries and triangulations have not been considered. Coupled soil-root models for these have yet to be developed.

[10] As long as the root geometry does not change, it is sufficient to perform this operation only once and keep the distribution.

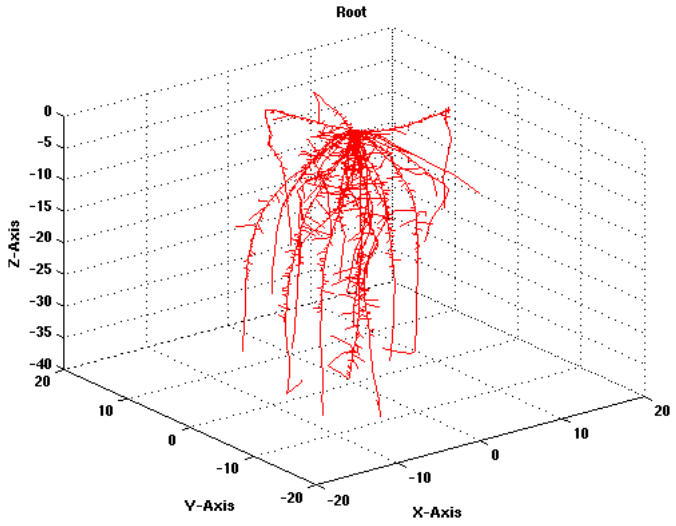


Figure 1: Matlab-Illustration of a one-plant root network within a simulation box.

This causes the β-field to vanish except for a few isolated grid points. We regard this undesirable. We have developed a routine that inserts 'auxiliary' segments into the root network, until its resolution matches the soil grid's one.

4.3 Distribution strategies for the soil domain

The main part of our work consisted in (i) implementing the routines that apply different distribution strategies and (ii) measuring the run-time performance of these strategies. After we reviewed the strategies that have been considered and recalled their respective motivation, we examine the results of the benchmarkings.

1. Keep the distribution as it is suggested by parMETIS (henceforth refered to as "normal" or "first" strategy).
2. Keep the soil elements that numerically interact with root segments on only one process: If the root system has less degrees of freedom than the soil system, it might be feasible to solve it on only one process, whereas the communication overhead for the system coupling may be too high. An OpenMP-based shared-memory parallel framework for the root system has been implemented for R-SWMS [4] (henceforth "second" strategy).
3. A multiple plant scenario: for each plant, we leave parts of the soil, which is affected by only one plant's root system on one single process each. For large fields of plants, this is a reasonable compromise between the above, as the part of the soil penetrated by multiple roots is relatively small (heneforth "third" strategy).

The original idea has been to utilize features of the already employed parMETIS library. Having modified the decisive function calls, the resulting distributions turned not be convincing for our purposes:

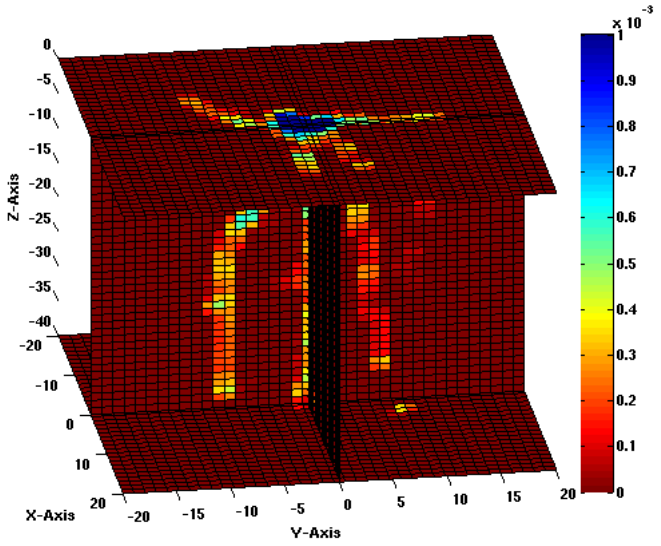


Figure 2: Illustration of the β-field with the sample root network.

Either the weights have not been high enough, to obtain a distribution with desired properties - or, when the weights had been set higher in order to amplify the effect, the program crashed.

Because of that, another approach has been implemented for the second and third strategy. Being returned a distribution by parMETIS, we alter the distribution by our own routines. This approach has been implemented and used during benchmarking, in order to obtain more meaningful results about the strategies impact (see Figure 3).

The benchmarks have been taken out on JUROPA parallel computer. The tests for each scenario or feature have been taken with 1 to 4 nodes on JUROPA, which corresponds to 8, 16, 24 and 32 processes. A one-process run has been evaluated for comparison with serial execution. parSWMS experiences a performance penalty for more than 32 processes, due to communication overhead.[11] The scenario comprised an 20cm $\times$ 20cm $\times$ 40cm box with no boundary conditions, root of age 45d and an homogenous initial water content of 20%. The simulated period of time has been 365d.

4.4 Comparison of first and second strategy

We expected an effective serialization of the program in case of the second strategy, since for all number of processes, a non-negligible part of the soil geometry remains managed by only one process. We used a $40 \times 20 \times 40$ grid, and the results correspond to the above expectation (see Figure 4).

We employed a finer grid, with a resolution of $200 \times 100 \times 200$. We recall that the roots are essentially one-dimensional, so a finer resolution decreases the volume of the root-affected soil, but increases its surface (see this as a non-rigorous illustration). This might decrease elements and nodes on the master

[11] A finer inspection with regards to number of processors has not been done. The evaluation of the measurements would have been complicated by causes specific to the architecture of JUROPA (see below).

process, but entail larger communication overhead. Of these two antagonistic possibilities, the first has been shown to apply in our scenario - despite a performance decrease, the program still scaled well (see Figure 5).

The time measurements have been disturbed due to a less performant routine at initialization. For the coarse scenario, more than 80s, and for the fine-resolution scenario, about 1h should be drawn off.

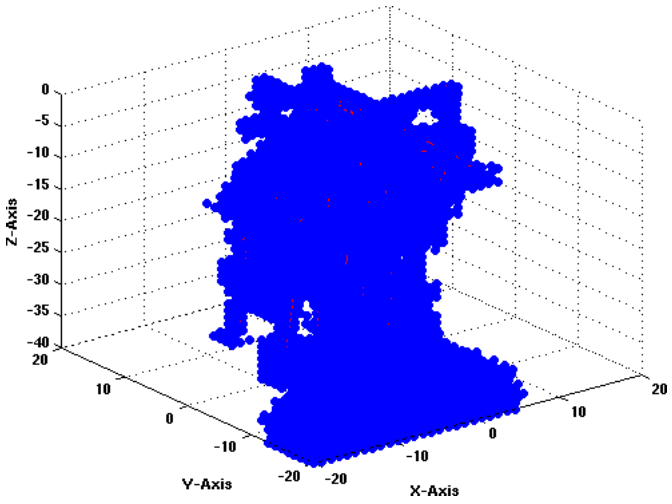


Figure 3: Depiction of the soil part that is managed by process 0 when using the modified approach with our sample scenario. Not only can we see that all mesh nodes within the reach of the root are one process only, but also the remaining parts of the soil, that have been assigned to process 0 in the normal way, can be seen at the bottom.

N	T_n	T_a
1	1107	1107
2	437	558
3	332	463
4	242	449
5	204	444

Figure 4: Running-times of normal (T_n, [s]) and aligned (T_a, [s]) strategy of distribution with a coarse grid scenario, depending on number of JUROPA nodes (N, [-]). Averages of 5 runs each.

4.5 Comparison of first and third strategy

For the case of two plants, we used again a $40 \times 20 \times 40$ grid. The root system consisted of two offset copies of the single-root system. We recognized a performance decrease with the third strategy, as compared to the normal distribution (see Figure 6). The scaling seems to be poor, but we attribute this to the overhead at initialization. It is visible, as the unnecessary slow-down (see above) has been removed for these benchmarks.

N	T_n	T_a
1	33246	33246
2	21330	22638
3	10636	13123
4	7339	8692
5	5801	7076

Figure 5: Running-times of normal (T_n, [s]) and aligned (T_a, [s]) strategy of distribution with a fine grid scenario, depending on number of JUROPA nodes (N, [-]). Averages of 5 runs each.

N	T_n	T_a
1	609	609
2	234	307
3	217	246
4	136	219
5	126	212

Figure 6: Running-times of first (T_n, [s]) and third (T_a, [s]) strategy of distribution on a two-plants scenario, depending on number of JUROPA nodes (N, [-]). Averages of 5 runs each.

5 Results

We have extended the feature set of parSWMS and obtained results on the performance of different implementations. Though we have to regard these always with respect to parSWMS, they allow for more general conclusions about possible algorithmic decisions for the modeling of root water uptake. This condition also holds for our measurements on different distribution strategies.

The root model has been successfully implemented. The code is well-structured and provides routines for debugging and benchmarking. We suspect, that for larger scale simulations with more root networks, we have to consider the specific properties of the model and adapt the internal data structures. The code has been designed to alleviate future changes. The root water uptake according to root geometries has been implemented successfully and produces results as expected by prior simulations.

Although we must not disregard that our benchmarking results are specific to parSWMS, we can draw general conclusions on the performance of our distribution strategies, and point out prospective issues of concern. For the modified distributions, we have observed a measurable slow-down as expected. The program has been effectively serialized, but for finer geometries the code scaled significantly better. In case the root affected soil is assigned to one only process, an a-priori refinement around the root segments might be a promising direction. This approach has already been shown by Schröder to give accurate results with good performance [3].

Nevertheless, we expect the performance influence of our modified distribution to depend strongly on the range and concentration of our root network, and this should be inspected further for more conclusive results. We have found in addition, that the built-in features of parMETIS do not suffice for our intentions. An effective work-around has been developed, but an elegant and more precise solution has yet to be found.

6 Outlook

A promising perspective seems to be using parSWMS with a modified distribution strategy, combined with a-priori refinement and a multithreaded solver for the root system. The optimal distribution strategy presumably depends on properties of the root network (e.g. range, density) – importance and impact of these details have yet to be worked out.

The root model of parSWMS is very basic and can be extended easily, e.g. by root growth or solute uptake. parSWMS' simulation of soil water transport – its original purpose – can probably be improved on a basic level by (i) adding a Newton-based solver and use the current routines for preiterating (ii) add grid redistribution and adaptiveness with respect to work load.

parSWMS does not scale well beyond 32 processors, and its performance strongly depends on the architecture. It could be worth to inspect these aspects in order to allow for benchmarking results – and simulations – on larger scales.

Acknowledgments

I would like to thank my advisors Natalie Schröder and Bernhard Körfgen, and I wish them the very best for their professional future and scientific life. Furthermore my thanks go to Prof. Mathieu Javaux, for this support and advice, Dr. Horst Hardelauf for his explanations, and to the friendly IT support of the JSC. I would like to express my very best thanks to Robert Speck and Mathias Winkel, who organized the programme and performed an excellent job. Last but not least, I would like to express my gratitude to my fellow guest students - it has been a pleasant experience to get to know with them, and each single one of them is an interesting and pleasant person. I hope you achieve the goals you set for your life and will keep the time we spent together in fond memories.

References

1. C. Doussan, L. Pàges, G. Vercambre: Modelling of the Hydraulic Architecture of Root Systems: An Integrated Approach to Water Absorption – Model Description. Annals of Botany 81 (1998) 213 - 223.
2. Šimůnek, J., Huang, K., and Van Genuchten, M.Th. 1995. The SWMS_3D code for simulating water flow and solute transport in three-dimensional variably-saturated media. Version 1.0. Research Report 139, Riverside, U. S.
3. Thomas Schröder: Three-dimensional modelling of soil-plant interactions: Consistent coupling of soil and plant root systems. PhD-Thesis. Jülich Research Centre 2010.
4. David Garre: Weiterentwicklung eines Programms für die Simulation der Wasseraufnahme durch viele Pflanzenwurzeln in ungesättigten Böden. Master-Thesis. Instute for Chemistry and Dynamics of the Geosphere. Jülich Research Centre 2010.
5. M.T. van Genuchten and D.R. Nielsen, On describing and predicting the hydraulic properties of unsaturated soils, Ann Geophys 3 (5) (1985), pp. 615-628.
6. Feddes, R.A., Raats, P.A.C., 2004. Parameterizing the soil-water-plant root system. In: Feddes, R.A., de Rooij, G.H., van Dam, J.C. (Eds.), Unsaturated-zone Modeling: Progress, Challenges, Applications, vol. 6, Wageningen UR Frontis Series, pp. 95-141.
7. Hardelauf, H., M. Javaux, M. Herbst, S. Gottschalk, R. Kasteel, J. Vanderborght, and H. Vereecken, PARSWMS: a parallelized model for simulating 3-D water flow and solute transport in variably saturated soils. Vadose Zone Journal, 6(2), 255-259, 2007.
8. L.A.Richards. Capillary conduction of Liquids through porous Mediums. Physics 1, 318. 1931.
9. J.L.Vazquez. The Porous Medium Equation. Mathematical Theory. Oxford Univ. Press, 2006

10. Celia, M. A., E. T. Bouloutas, and R. L. Zarba. 1990. A general mass-conservative numerical solution for the unsaturated flow equation, Water Resour. Rex, 26(7), 1483-1496.
11. Mathieu Javaux, Valentin Couvreur, Tom Schroeder, Jan Vanderborght. R-SWMS: three-dimensional, simultaneous modelling of root growth, transient soil water flow, and solute transport and uptake. Release 3.4.0, February 2010
12. Dietrich Braess. Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics - Paperback. Cambridge University Press; 3 edition (April 30, 2007)
13. ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering [homepage on the internet]. 2010 [cited 2 October 2010] Available from: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview.
14. PETSc: Home Page [homepage on the Internet]; 2010 [cited 2 October 2010]. Available from http://www.mcs.anl.gov/petsc/petsc-as/.
15. PETSc and Threads [homepage on the Internet]; 2010 [cited 2 October 2010]. Available from http://www.mcs.anl.gov/petsc/petsc-as/miscellaneous/petscthreads.html.
16. R.B.Jackson, John S.Sperry and T.E.Dawson. Root water uptake and transport: using physiological processes in global predictions. Trends in Plant Science. Volume 5, Issue 11, 1 November 2000, Pages 482-488
17. Mathieu Javaux, Xavier Draye: Modeling 3-D root water uptake with R-SWMS. Unpublished slides of the talk held on September 14-15, 2009 at Louvain-la-Neuve, Belgium.

Analysis Tools for the Results of Scalasca

Markus Mayr

Vienna University of Technology
Institute for Analysis and Scientific Computing
Wiedener Hauptstraße 8–10
1040 Wien, Austria

E-mail: markus.mayr@tuwien.ac.at

Abstract:
Scalasca is a tool set for performance analysis of parallel applications. To detect errors in Scalasca, the software is tested as is customary. One of the main steps of the testing procedure is to try to find errors in Scalasca's analysis reports. These errors can be of two different kinds. First, the output can be ill-formed and second, the measurement or analysis data can be wrong. Both kinds of errors can be detected automatically. We provide tools that analyze Scalasca's output and report errors. This report serves as an overview for this set of testing tools and the library these tools are built upon.

1 Introduction

Worldwide efforts at building parallel machines with high performance levels also put a burden on application developers that face difficulties in exploiting the full potential of such machines. That is why the HPC community needs powerful and robust performance-analysis tools that make the optimization of parallel applications both more effective and more efficient [1].

In order to satisfy their growing demand for computing power, supercomputer applications are required to harness unprecedented degrees of parallelism. With an exponentially rising number of cores, the often substantial gap between peak performance and that actually sustained by production codes is expected to widen even further. However, increased concurrency levels place higher scalability demands not only on applications but also on parallel programming tools needed for their development. Scalasca is a tool set specifically designed for use on large-scale systems. This Section gives an overview of Scalasca and its analysis reports.

1.1 Scalasca

The current version of Scalasca [1] supports measurement and analysis of the MPI, OpenMP and hybrid programming constructs most widely used in highly-scalable HPC applications written in C, C++ or

Fortran on a wide range of HPC platforms. From a high level point of view, Scalasca usage proceeds through three clearly separated steps.

1. Preparation of an instrumented executable: During this step Scalasca adds code to the executable to obtain the actual measurement data later on.
2. Measurement collection and analysis: The instrumented executable is run on a computer system and measurement data is collected. The data is analyzed, either using runtime summarization or by creating a trace of all relevant events. Independently of the measurement mode, Scalasca provides one report that contains all the obtained analyses.
3. Analysis report examination: Finally, Scalasca contains a tool to examine the generated analysis report. It is called CUBE and discussed in Section 1.2.

Our testing tools operate at the same stage as the CUBE report examination tool. Hence, it is crucial to understand the interface between step 2 and step 3 in order to understand how the testing tools work. In the subsequent sections 1.2 and 1.3 we explore this interface by considering Scalasca's report examination tool CUBE and its file format.

1.2 CUBE

The CUBE GUI [2] is a presentation component suitable for displaying a wide variety of performance data for parallel programs including MPI and OpenMP applications and allows interactive exploration of that data. CUBE has been designed around a high-level data model of program behavior called the *CUBE performance space* and consists of three dimensions:

- Metric dimension: A set of metrics that describe *what* was measured, for example *execution time*.
- Program dimension: Contains the program's call tree which includes all call paths onto which metric values can be mapped.
- System dimension: Contains all system resources allocated and associated program entities executing in parallel, i.e. processes and/or threads depending on the programming model.

Measurement data then corresponds to a map

$$\{\text{Metrics}\} \times \{\text{Call Tree Nodes}\} \times \{\text{System Resources}\} \rightarrow \mathbb{R}$$

and is called *severity* mapping. Each dimension of the performance space can be organized in a *hierarchy*. An example of this hierarchy that is displayed by the CUBE GUI is shown in Figure 1.

The CUBE package not only consists of the report examination GUI but also provides a rich set of *algebra tools* [3] that can manipulate reports and also serve as a solid base to build further report analysis tools on.

1.3 CUBE file format & API

The CUBE data format is an XML instance that is usually stored compressed. The CUBE library provides an interface to create and read CUBE files. CUBE files basically contain all metadata, i.e. the

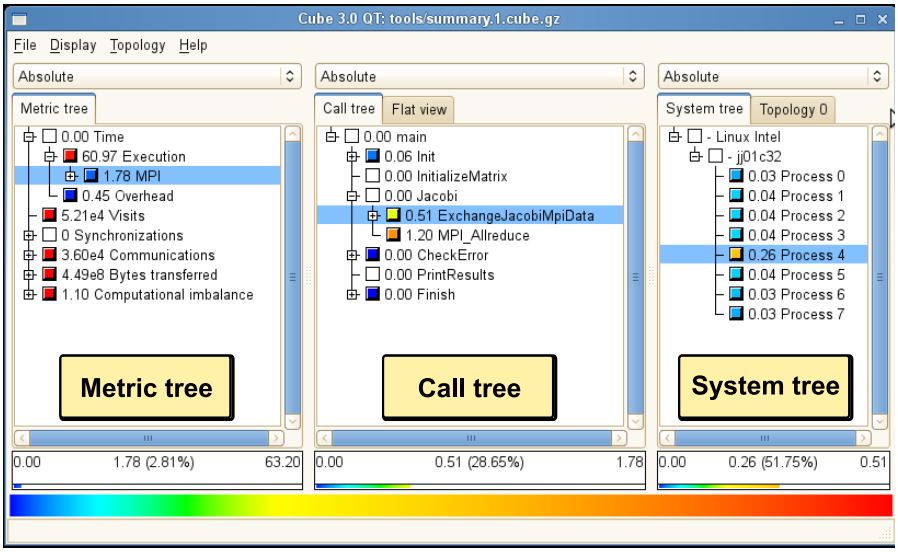


Figure 1: The CUBE GUI displaying all three dimensions of the *CUBE measurement space*.

information about the metric dimension, the program dimension and the system dimension, as well as the analysis data.

Each dimension is represented by a number of C++ classes. The relation between the displayed tree in Figure 1 and the representation within the file and the CUBE library is straight-forward for the metric and system dimension. Since the testing tool interacts most closely with the program dimension, it will be discussed in more detail.

The program dimension consists of a data structure that involves two basic entities. First, there is the `Region` data structure. A `Region` corresponds to a single block in the program that has been instrumented, such as a program routine or parallel loop. A CUBE contains a plain set of `Regions` and each of them contains information about the associated instrumented block. Second, there is a call tree structure, whose nodes are called `Cnodes`. Each `Cnode` corresponds to a certain `Region` in a specific calling context.

2 Testing tools

Software testing is a common way in software development to detect errors. For Scalasca, we test whether the whole tool chain behaves as expected, by analyzing the final output, i.e., the analysis report (in CUBE format). One can distinguish between two kinds of test cases for Scalasca's analysis reports.

First, there are a number of tests that can be applied on a single CUBE file in order to look for errors in the metadata or to perform basic sanity checks on the measurement data. This kind of tests is implemented as part of the `cube3_sanity` tool that is documented in section 2.3.

Second, a number of tests can be performed on multiple CUBE files in order to compare the measurement data of these files. Almost all test cases that compare two different CUBE files basically act on

either the set of Regions or the call tree that consists of Cnodes. A command-line tool to specify Cnode-based tests is provided by the cube3_test tool that is documented in section 2.4.

Furthermore, we provide extensions to the cube3_cut tool that are documented within recent versions of the CUBE User Guide [2]. We also provide the cube3_info tool which is a command-line tool to show extracts of multiple CUBE files side by side, which is documented in section 2.2 and the cube3_canonicalize tool which is a pre-processing tool that helps to compare and potentially merge CUBE files later on. cube3_canonicalize is documented in section 2.1.

Currently, our testing strategy consists of three steps. First, we run sanity checks on all CUBE files using cube3_sanity. Then we canonicalize the files and use the cube3_cut tool to remove certain nodes in a way that it is possible to more readily compare all relevant CUBE files. Finally we use the cube3_test tool to apply comparative tests on all involved files.

2.1 Canonicalize

Iterates over all regions of a given CUBE file and removes excess information from the region names and attributes, reducing them to a canonical form, e.g., without parameters or return values, without trailing underscores, and all in lower case. This tool is usually used as a preprocessing step in order to apply tools that later on work with multiple CUBE files.

Usage: ./cube3_canonicalize [flags] <input>

<input> denotes the name of the CUBE file that is going to be processed. The output CUBE file is almost the same as the input CUBE file, but the regions within it contain *less* information. [flags] is any combination of the following flags:

-h, --help	Help; Output a brief help message.
-o, --output out.cube	Specify the output CUBE file name.
-p, --pdt	Remove annotations provided by PDToolkit.
-m, --max-length length	Truncate region names to length characters.
-c, --lower-case	Convert all function names to lower case.
-f, --remove-file-names	Remove all regions' file names.
-l, --remove-line-numbers	Remove all regions' line numbers.

Let us consider a few examples.

- The function name "int main(int, char **) C" is usually provided by the PDToolkit. Applying cube3_canonicalize -p produces the function name 'main".

- The GNU Fortran compiler stores the name of the entry function as "MAIN__". By running the command cube3_canonicalize -c we would end up with "main" which is the same name that the CCE Fortran compiler provides for the entry function, for example.

2.2 Info

Prints out specified information extracted from one or more CUBE files. Prints out any number of metrics or aggregated metrics. It is capable of removing callpaths from the tree according to a given set of rules in order to show only the most relevant parts of the tree with respect to some aspect. When callpaths are removed they are replaced with a synthetic callpath which accumulates aggregated metric values.

An additional `visitors` metric is derived by counting the number of processes (or threads) that execute the callpath at least once: callpaths not executed by all processes may indicate conditional execution or missing/misplaced measurements.

An abbreviated example of the output is below. The width of the columns was reduced and the lines were cut off after a fixed number of characters (denoted with ellipsis).

```
user@host% ./cube3_info -m time -r time,0.25  summary.1.cube.gz summary.2.cube.gz
|    Time |    Time | Diff-Calltree
| 63.2041 | 63.5443 |  * main
|  0.5866 |  0.8209 |  |  * ***** Aggregated 5 siblings (+8 children) within main
| 62.6150 | 62.7221 |  |  * Jacobi
|  1.2047 |  1.1936 |  |  |  * ***** Aggregated 1 siblings (+0 children) withi...
| 21.9683 | 22.0265 |  |  |  * ExchangeJacobiMpiData
|  0.5090 |  0.5294 |  |  |  |  * ***** Aggregated 3 siblings (+0 children) wi...
```

Usage: `./cube3_info [ options ] <CUBE file>+`

`<CUBE file>+` denotes an arbitrary number of CUBE files separated by spaces. `[ options ]` is any combination of the following options. Each option also has a long name which is listed at the end of the options description and must be prefixed with two dashes (`--`) when used.

`-h`	Help; Output a brief help message. (Long name: `help`)
`-w`	Prints out the visitors metric for each node. (Long name: `visitors`)
`-m metric`	Prints out the metric described by the string `metric` next to each call tree node. (Long name: `metric`)
`-l`	Prints out a list of all available metrics within the given CUBE files. (Long name: `list`)
`-b`	Prints out some basic information about the first CUBE file in the list. (Long name: `basics`)
`-a metric,thrh`	Removes all nodes from the call tree whose absolute aggregated values are below `thrh` for the metric `metric`. (Long name: `absolute`)
`-r metric,thrh`	Removes all nodes from the call tree whose values are below `thrh` fraction of the sum of all roots' values. `thrh` should be a value between 0 and 1. (Long name: `relative`)
`-x metric,thrh`	For each node of the call tree we consider all children $\mathcal{N}$ of that node. Let v_i be the value for the given metric and the node $i \in \mathcal{N}$. Then we select a maximal subset $\mathcal{M} \subseteq \mathcal{N}$ with the property $\sum_{i \in \mathcal{M}} v_i \leq \texttt{thrh} \sum_{i \in \mathcal{N}} v_i$ and remove all nodes from $\mathcal{M}$. (Long name: `cum-sum`)

The `metric` string contains all the information to map a value to each node of the call tree. For a very general description of how this string can look like, please refer to Section 3.2. Here, we only cover the most frequent use-case.

The string `metric` can have the following form:

```
uniq_metric_name[:[[INCL | EXCL]:[Process[.Thread]]]]
```

`uniq_metric_name` is the name of the metric as reported by the list option. `INCL` means that you want to print out the inclusive value and `EXCL` represents the exclusive value. Finally `Process` is the process you want to aggregate this value over. `*` means that the value is aggregated over all processors. If `Process` is a non-`*` value then `Thread` may be the `Process`-local rank of the thread that the value is printed out for. The square brackets indicate that most parts of the string are optional.

2.3 Sanity

Runs a set of sanity checks on a single CUBE file. Some of the tests can be switched off and one can also provide an output file for more detailed error output.

Some of the sanity checks are only run on parts of the call tree other checks were successful on. To represent these dependencies, the single tests are organized in trees, shown using indentation.

An example of the output is below.

```
user@host% ./cube3_sanity -nl -o details.log summary.cube.gz
Name not empty or UNKNOWN ... 19 / 19 OK
  No ANONYMOUS functions ... 19 / 19 OK
  No TRUNCATED functions ... 19 / 19 OK
  File name not empty ... 19 / 19 OK
  No TRACING outside Init and Finalize ... 19 / 20 OK
```

One function failed the last test. We can examine the `details.log` file to find out more.

```
user@host% cat details.log
        ...
        In call node with id 19 (Name: TRACING)
        Call path
                   TRACING (File: EPIK, Line: -1)
         called by BadGuy (File: bad_file.c, Line: -1)
         called by Finish (File: main.c, Line: -1)
         called by main (File: main.c, Line: -1)
        Found TRACING outside MPI_Init and MPI_Finalize.
        Parent's name is BadGuy
        ...
```

Usage: `./cube3_sanity [ options ] <CUBE file>`

`<CUBE file>` is the file that sanity checks are applied to and `[ options ]` is any combination of the following options. All options also have a long name that is listed at the end of the options description and must be prefixed with two dashes (`--`) when used.

`-h`	Help; Output a brief help message. (Long name: `help`)
`-n`	Disables the (very time consuming) check for negative metric values. (Long name: `no-negative-values`)
`-l`	Disables the check for proper line numbers. (Long name: `no-line-numbers`)
`-f file.filt`	Adds an additional test that checks whether a node's name is matched by any pattern in the filter file file.filt. (Long name: `filter`)
`-o output_file`	Path to the output file. If no output file is given, detailed output will be suppressed. A summary will always be printed out to `stdout`. (Long name: `output`)

2.4 Test

Compares two or more experiments according to some specified criteria. Prints out a summary of the outcome of the tests and optionally also prints out more detailed information to a file.

The key idea behind the `cube3_test` tool is to create new call trees out of the default call tree `ALL` that contains the whole call tree, to remove certain nodes from that tree and to apply checks on these sub-trees.

We provide two examples below. The first example simply checks whether the `visits` metric matches exactly and if the `time` metric matches *approximately*. For a detailed explanation of the comparison behaviour, see below. The file `details.log` contains more details about the two cases where the `time` metric did not match sufficiently closely. We can find more details in the file `details.log` we specified using the `-o` option.

```
user@host% ./cube3_test -e visits -s time,REL,0.33 \
           -o details.log summary.1.cube.gz summary.2.cube.gz
Equality basic@visits:incl:*.* ... 20 / 20 OK
Similarity basic@time:incl:*.* (relative, 0.33) ... 18 / 20 OK
```

Let us examine the errors detailed in the `details.log` file.

```
user@host% cat details.log
...
In call node with id 1 (Name: Init)
   for cnode metric Time (basic@time:incl:*.*)
|    Time |    Time | Call path
|  0.5193 |  0.7538 |            Init (File: main.c, Line: -1)
| 63.2041 | 63.5443 |  called by main (File: main.c, Line: -1)
Absolute difference between the file with id 1 and a file with
a lower id is 0.234487 but should be lower than 0.210071.
...
```

The second example creates the tree `time_rel` as a copy of the `ALL` tree and removes all nodes from that tree where the `time` metric is below 1 second (aggregated over all processes). Then it runs the same test as above.

```
user@host% ./cube3_test -c time_rel -a time,1,time_rel -s time,REL,0.33 \
           summary.1.cube.gz summary.2.cube.gz
Similarity basic@time:incl:*.* (relative, 0.33) ... 4 / 4 OK
```

We only considered 4 out of the 20 callpaths from the original call tree. In all other functions, we spent less than 1 second (in *each of the* experiments). So for all time-consuming call tree nodes the `time` metric matches *approximately*. For this simple case, the creation of an additional working copy was not necessary and only served the purpose of showing the usage of this option.

Usage: `cube3_test [ options ] <CUBE_file> <CUBE_file>+`

`[ options ]` is any combination of the following. The order of the options matters. All options also have a long name that is listed at the end of the options description and must be prefixed with two dashes (`--`) when used.

`-h`	Help; Output a brief help message. (Long name: `help`)
`-c tree1[,tree2]`	Create a new copy of the tree `tree2` that is called `tree1`. If `tree2` is omitted, `tree1` will be a copy of the `all` tree. (Long name: `create-tree`)
`-e metric[,tree]`	Checks whether the metric described by the string `metric` is equal for all provided CUBE files and for all nodes of the tree `tree`. If `tree` is omitted, all nodes are checked. (Long name: `equals`)
`-s metric,type,tol[,tree]`	Checks whether the metric described by the string `metric` is sufficiently similar for all CUBE files, according to the qualifiers `type` and `tol`. `tol` specifies the tolerance which is given as a real number. The meaning of the tolerance value depends on the `type`. `type` is either `ABS` or `REL`. If `type` is `ABS` than the nodes are similar if the difference between all values for that metric is below `tol`. If `type` is `REL` than the nodes are similar if the difference between all values for that metric is below $\sum_{i=1}^{n} \texttt{tol} \frac{\texttt{value of metric for cnode of } i\texttt{-th file}}{n}$ where n is the number of CUBE files. If `tree` is omitted, all nodes are checked. (Long name: `similar`)
`-a metric,thrh[,tree]`	Removes all nodes from the tree `tree` whose values are below `thrh` for the metric `metric`. If `tree` is omitted, nodes are removed from the default tree `ALL`. (Long name: `absolute`)
`-r metric,thrh[,tree]`	Removes all nodes from the tree `tree` whose values are below `thrh` percent of the sum of all roots' values. `thrh` is supposed to be a value between 0 and 1. If `tree` is omitted, nodes are removed from the default tree `ALL`. (Long name: `relative`)

`-x metric,thrh[,tree]`

For each node of the tree `tree` we consider all children $\mathcal{N}$ of that node. Let v_i be the value for the given metric and the node $i \in \mathcal{N}$. Then we select a maximal subset $\mathcal{M} \subseteq \mathcal{N}$ with the property

$$\sum_{i \in \mathcal{M}} v_i \leq \texttt{thrh} \sum_{i \in \mathcal{N}} v_i$$

and remove all nodes from $\mathcal{M}$. If `tree` is omitted, nodes are removed from the default tree `ALL`. (Long name: `cum-sum`)

For information on how the `metric` string should look like, refer to Sections 2.2 and 3.2.

3 High-level overview of testing library

The testing library uses slightly different classes for reading in CUBE files or managing them. These classes are shortly discussed in section 3.1. Then it creates one or more working copies of the call tree. We provide classes that remove certain nodes from these copies and help to select relevant parts of the call tree with respect to some aspect by using the `AbridgeTraversal` class which is discussed in Section 3.3.

Finally, we check whether these parts of the call tree fulfil certain conditions. A short example of such a constraint is given in section 3.4.

Many tests only check a single value for each call node. Therefore, we provide a number of classes that map each call tree node onto a real value. These `CnodeMetric` is shortly discussed in section 3.2.

3.1 Reading in one or more CUBE files

The library extends CUBE's base classes for file input slightly and provides the new classes `MdAggrCube` and `MultiMdAggrCube` that read in one or multiple CUBE files respectively. `MdAggrCube` is constructed using its copy constructor, `MultiMdAggrCube` is constructed by providing a vector of `AggrCube` instances. Examples are given in Listing 1 and Listing 2 respectively.

```
1 istream stream = ...; // An input stream instance
2 AggrCube* cube = new AggrCube();
3 stream >> *cube; // Read in the CUBE file
4 MdAggrCube* mdcube = new MdAggrCube(cube);
5 delete cube;
```

Listing 1: Reading in a CUBE file

```
1  vector<istream> streams = ...; // Some input stream instances
2  vector<AggrCube*> cubes;
3  for (vector<istream>::iterator it = streams.begin();
4        it != streams.end(); ++it)
5  {
6    AggrCube* cube = new AggrCube();
7    *it >> *cube;
8    cubes.push_back(cube);
9  }
10 MultiMdAggrCube* mdcube = new MultiMdAggrCube(cubes);
```

Listing 2: Reading in multiple CUBE files

3.2 Call tree node metrics

When considering the measurement data for a certain `Cnode`, it is often possible to formulate tests based on a single number to verify if the measurement data for that certain `Cnode` is sane with respect to some aspect. The class `CnodeMetric` and its sub-classes basically encapsulate all information to compute a certain number and present a simple interface to the testing library.

`CnodeMetric` itself is an abstract class that is implemented by the two classes `VisitorsMetric` and `AggregatedMetric` at the moment. Each `CnodeMetric` must be able to serialize to and deserialize from an ordinary string that contains all relevant information.

The serialized string generally has the form `prefix@data`. The prefix uniquely identifies the class that implements the metric. The format of `data` depends on the type. For a documentation of the format for the prefix `basic`, please refer to Section 2.2. For the prefix `visitors`, the string `data` is simply expected to be empty.

3.3 Removing parts of the call tree

We usually work on working copies of the call tree, which are called `CnodeSubForest`. A copy of the whole call tree can be obtained by applying the `get_forest` method on a `MdAggrCube` instance. `CnodeSubForest` instances can also be created by copying another instance.

Removing nodes from a `CnodeSubForest` is done using the `AbridgeTraversal` class. There are multiple `threshold types` for removing nodes: in our example in Listing 3 we consider the type `THRTYPE_ABSOLUTE` which removes nodes based on an absolute value.

```
1  MdAggrCube* cube = ...; // A cube instance
2  CnodeSubForest* forest = cube->get_forest(); // A CnodeSubForest instance
3  AbridgeTraversal("basic@time:incl:*.*", 1,
4                   THRTYPE_ROOT_ABSOLUTE).run(forest);
```

Listing 3: Removing nodes with a `time` value of less than 1.

3.4 Constraint implementation example

Constraints are always written by extending the class `AbstractConstraint` which provides basic output routines and other features. There are a few classes that extend `AbstractConstraint` and provide further methods. In this example, we choose the super-class `CMetricCnodeConstraint`.

Listing 4 implements a constraint that checks whether the difference for the metric `time` is below $1e{-}5$ for each pair of corresponding call tree nodes, and Listing 5 shows how to use it.

```
1  class SimpleConstraint : public CMetricCnodeConstraint {
2    public:
3      SimpleConstraint(CnodeSubForest* forest)
4        : CMetricCnodeConstraint(forest, "basic@time:excl:*.*")
5      {}
6
7      virtual string get_name() {
8        return string("SimpleConstraint");
9      }
10
11     virtual string get_description() {
12       return string("Checks␣whether␣for␣each␣pair␣of␣corresponding␣"
13             "Cnode␣instances␣the␣absolute␣difference␣for␣the␣metric␣")
14             + get_metric()->to_string() + "␣is␣below␣1e-5.");
15     }
16
17     virtual void check() {
18       if (get_forest()->get_reference_cube()->get_number_of_cubes() != 2)
19         throw new Error("I␣need␣exactly␣two␣CUBE␣files␣for␣this!");
20       CMetricCnodeConstraint::check();
21     }
22
23     virtual void cnode_handler(Cnode* node) {
24       double v1 = get_metric()->compute(node, (unsigned int) 0);
25       double v2 = get_metric()->compute(node, (unsigned int) 1);
26       if (fabs(v1-v2) > 1e-5)
27         fail("Difference␣is␣too␣big!", node);
28       else
29         ok();
30     }
31 };
```

Listing 4: Implementation of a simple constraint class

The documentation for this piece of code now reads:

- In line 3–5 we simply construct our `SimpleConstraint` class by calling the constructor of `CMetricCnodeConstraint`.
- In line 7–9 we provide a string that is shown as the name of this test. It is reasonable to include information about the parameters for this constraint as well.
- In line 11–15 we provide more detailed information about this constraint.
- In line 17–21 we have the possibility to check preconditions this constraint depends on. In this case, we check that there are exactly two CUBE files available.

- In line 23–30 we finally implement the code that actually verifies whether the constraint is not broken for the CUBE files. In line 24–25 we use the `CnodeMetric` to compute the values for both CUBEs. Line 27 contains the `fail` method which is used to indicate that the test failed but also to provide useful information to the user. In line 29 we call the `ok` method. For each test either the `fail`, the `skip` or the `ok` method has to be called exactly once.

```
1 CnodeSubForest* forest = ...; // A CnodeSubForest instance
2 SimpleConstraint(forest).check();
```

Listing 5: Checking a constraint

4 Conclusion

During the guest student programme, we developed a set of tools to run a small number of automatic tests on CUBE files. Furthermore, we developed and improved tools to compare two or more CUBE files automatically. We developed a common library of functions used by all testing functions, that we hope will be flexible enough to adapt to new use cases.

At the moment, we have a number of tools available, each exposing only a part of the functionality of the underlying library. They can be easily put together to automate many steps of the testing procedure, although some parts still require manual involvement. In particular the `cube3_canonicalize` tool misses features to make the information that is removed during canonicalization available later.

Especially for large CUBE files, the set of many one-purpose tools results in loading and saving the CUBE files many times which results in a large overhead. A single tool that combines all functionality may address this problem.

At the moment, only relatively few tests are implemented. Further tests have to be written. The format in which these tests should be specified, however, is open for discussion. The current approach is to either specify tests within a C++ program or on the command line as a sequence of parameters to a certain program.

Acknowledgement

I would like to thank my advisers Brian Wylie and Zoltan Szebenyi, who were always available when I came up with questions and provided valuable feedback throughout the whole programme. Furthermore, I would like to thank Dirk Praetorius and Samuel Ferraz-Leite from Vienna University of Technology, whose support and help made the participation in this programme possible.

References

1. M. Geimer, F. Wolf, B. Wylie, E. Ábrahám, D. Becker, B. Mohr The Scalasca performance toolset architecture. Concurrency and Computation: Practice and Experience, 22(6):702-719, April 2010
2. The Scalasca Developement Team CUBE 3.3 - User Guide
3. F. Song, F. Wolf, N. Bhatia, J. Dongarra, S. Moore An Algebra for Cross-Experiment Performance Analysis. In Proc. International Conference on Parallel Processing (ICPP), pages 63-72, Montreal, Canada, IEEE Society, August 2004.

Modeling of Doubly-connected Fields of CPV/T Solar Collectors

Yosef Meller

Tel Aviv University,
Levanon street,
Tel Aviv, Israel

E-mail: yosefmel@post.tau.ac.il

1 Introduction

In Tel Aviv University (TAU), at the Solar Energy Lab, we are developing a solar collector that is designed to be integrated into commercial and industrial buildings. In this way it is hoped that excess energy that was not converted to electricity by the collector's photovoltaic (PV) module can be used in-place as heat for air-conditioning or other heat-consuming applications, thereby considerably increasing the overall system efficiency. In addition, the collector uses light-concentration technology in order to reduce the area of expensive PV cells and replace it by relatively cheaper mirrors. The combination of these technologies is called CPV/T: Concentrating, PhotoVoltaic, Thermal. A first experimental solar field using this collector type (shown in Figure 1) was operational over the last 2 years in Yokne'am, Israel; the design, construction and operation were conducted in cooperation with the UPP-Sol consortium funded by the European 6th Framework Program.

Side by side with designing the collector itself, designing and optimizing the layout and component selection for a field of such collectors is another task of the project. The target of the optimization may be

Figure 1: Experimental field of concentrating PV/thermal collectors.

maximization of annual energy or of collector efficiency, or minimization of the capital cost (in terms of $\$/\mathrm{W}$) or the cost of energy (in terms of the levelized energy cost in $\$/\mathrm{kWh}$).

The optimization procedure requires the ability to predict the performance – the electric and thermal energy output and efficiency – of a complete field composed of such collectors and accompanying equipment. The annual performance is sought, and the calculation thereof is by integration of the instantaneous performance over the year; this makes the calculation of the instantaneous field performance the most basic building block of optimization. The task is complicated by two main field-level features:

1. In order to achieve high voltage and thereby reduce wire losses, PV cells and collectors are connected in series. The relationship between current, voltage, insolation and temperature of the cells is non-linear and involves all those variables at the same time.

2. The utilization of excess heat becomes more efficient and applicable to more consumer types as the temperature at which that heat can be provided increases. Heat is collected by a cooling fluid that runs under the PV cells; cooling the cells raises the coolant's temperature, but only by a few degrees. For this reason a series connection of the cooling layer is also employed. Thermal and electrical connections are independent of each-other and may define different networks.

The goal of this project is to create a simulation program capable of accounting for these effects, which gives a reasonably accurate representation of the physical system, and can be used on a practicing engineer's workstation, which given today's state-of-the-art could be no larger than a "mini-cluster of four cores", as the relatively modern family of Intel processors was described by Prof. J. Grotendorst of the Juelich Supercomputing Centre.

In this report I present the simulation software that was developed for this end. First, the physical models that are used are presented. This is followed by a presentation of technical issues that arose during implementation and the solutions given to them. Finally, results of simulations performed using the software are presented and discussed.

2 Collector-level model

The collectors shown in Figure 1 are known in the field as *parabolic dishes*. They concentrate light into a focal area, in which a *receiver* component converst the incident light energy into usable electricity and heat. The collectors are kept pointing at the sun at all times of the day, and therefore at times when the sun is relatively low some of the collectors may shade other collectors that are standing "behind" them when looking from the sun's point of view. This effect and other collector-related optical losses are handled separately, and are outside the scope of this project, but it is important to remember that on different times, different receivers in the same field may be subject to different incident light, and different areas of the same receiver may also be illuminated differently.

The receiver is composed of two integrated power-conversion systems (Figure 2): The first surface encountered by incoming light (henceforth "the top") is a layer of photovoltaic (PV) cells which convert up to 36% of the energy directly into electricity; the rest of the energy is converted to heat, some of it is lost due to different loss mechanisms on the outer surfaces, and the rest is collected by the second

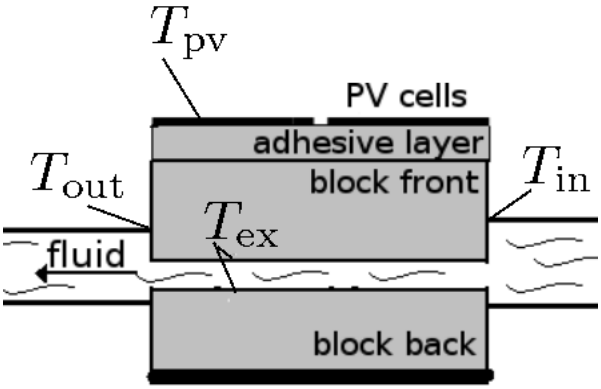


Figure 2: Receiver structure.

energy collection system, a heat-exchange block through which a coolant (water or a water/ethylene-glycol solution) is pumped.

2.1 Temperatures, power flows and losses

The receiver state can be characterized by the four temperatures noted in Figure 2: the PV cells temperature (T_{pv}); the temperature at the exchanger's contact with the fluid (T_{ex}), which is assumed to be uniform along the flow-path due to the high thermal conductivity of the exchanger block; and the coolant inlet and outlet temperatures (T_{in} and T_{out} respectively). The heat power lost to the environment can be calculated from these temperatures and the ambient temperature (T_{am}) using the following relations:

- Heat lost by convection from the top is

$$q_{\mathrm{conv,pv}} = \frac{T_{\mathrm{pv}} - T_{\mathrm{am}}}{R_{\mathrm{cpv}}},$$

  where R_{cpv} is the thermal resistance to convection (in $\mathrm{K/W}$) that is calculated using the appropriate empirical heat-transfer correlations.

- Heat loss by radiation emission from the top is

$$q_{\mathrm{em}} = \varepsilon \sigma A T_{\mathrm{pv}}^4,$$

  where ε is the emissivity factor, σ is the Stefan-Boltzmann constant, and A is the top surface's area.

- Heat loss by convection from the bottom is

$$q_{\mathrm{conv,ex}} = \frac{T_{\mathrm{ex}} - T_{\mathrm{am}}}{R_{\mathrm{cex}}},$$

where R_{cex} is the thermal resistance to convection (in $\mathrm{K/W}$) that is calculated similarly to R_{cpv}, and T_{ex} is dependent on the heat not lost by the top layer or converted to electricity:

$$T_{\mathrm{ex}} = T_{\mathrm{pv}} - \left(q_{\mathrm{in}} - q_{\mathrm{el}} - q_{\mathrm{em}} - q_{\mathrm{conv,pv}}\right) R_{\mathrm{int}}$$

for an internal thermal resistance R_{int} between the cells and the exchanger block. q_{el} is the energy that was converted to electricity, a quantity whose calculation is detailed in the following Section.

Together these losses are represented by

$$q_{\mathrm{loss}} \equiv q_{\mathrm{em}} + q_{\mathrm{conv,pv}} + q_{\mathrm{conv,ex}}.$$

The heat entering the heat exchanger can be calculated in two methods. Firstly, it can be calculated by standard heat-exchanger equations [1]

$$q_{\mathrm{o}} = \dot{m} C_{\mathrm{p}} \left(1 - e^{-UA/\dot{m}C_{\mathrm{p}}}\right) \left(T_{\mathrm{ex}} - T_{\mathrm{in}}\right), \tag{1}$$

where $\dot{m}$ is the coolant flow rate, C_{p} the coolant specific heat, and UA is the overall exchanger's heat conductivity, which is a function of both T_{in} and $\dot{m}$, whose parameters are fitted from empirical measurements of the receiver.

Secondly, it can be derived from the energy conservation law, as

$$q_{\mathrm{o}}^{*} = q_{\mathrm{in}} - q_{\mathrm{loss}} - q_{\mathrm{el}}, \tag{2}$$

where q_{in} is the light energy incident on the receiver. The need for two different calculation methods will become clear when the field-level model is explained in Section 3.

The outlet temperature may be calculated from the inlet temperature and coolant-entering heat by

$$T_{\mathrm{out}} = T_{\mathrm{in}} + \frac{q_{\mathrm{o}}}{\dot{m} C_{\mathrm{p}}}. \tag{3}$$

2.2 Electric state and power calculation

The electric state of the PV cells is usually represented as a relationship between the current and voltage in the cell for a given insolation and temperature, using one or two exponential terms (respectively known as the *one-diode* and *two-diode* model [2]). A real current-voltage (I-V) curve for a silicon solar cell is shown in Figure 3.

The I-V curve changes with the level of insolation, as shown in Figure 3. However, since the cells in a receiver are connected in series, and several receivers may also be connected in series, they must have the same current. So, if the string current is higher than the maximum current achievable by any cell on a given illumination level, that cell instead becomes *reverse-biased*, with negative voltage and the string current. In this mode the cell consumes power instead of producing it. To mitigate this effect, each cell has a *bypass diode* connected in parallel to it, that allows the current to go as high as needed with only minimal negative voltage.

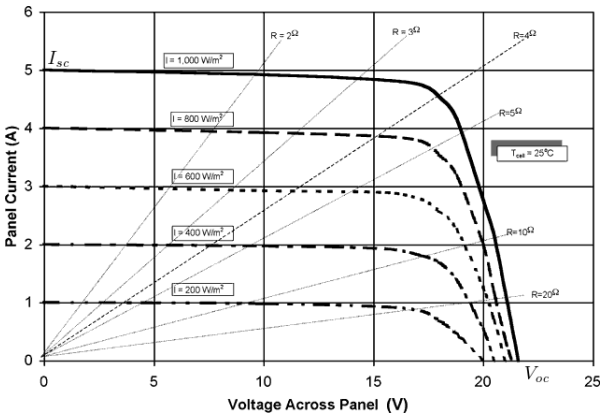


Figure 3: An I-V curve for a silicon solar cell.

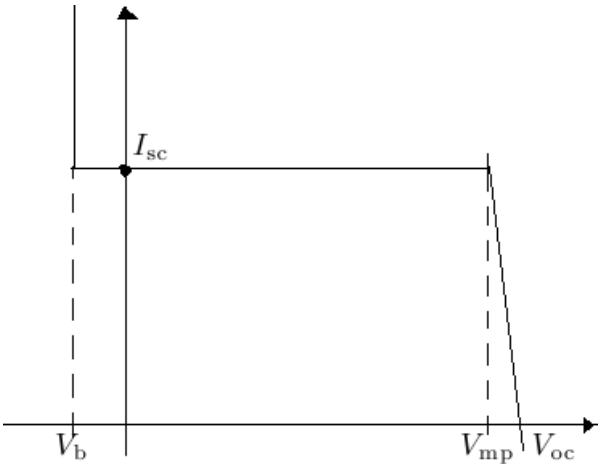


Figure 4: Piecewise-linear I-V curve.

The model for the cell and diode can be solved directly by use of the Lambert-W function [3], but this presents two problems. First, it requires an internal iteration which slows down the computation. More importantly, the curve contains Sections in which either the current or the voltage are nearly constant. Non-linear-equation solvers, such as the one that will be needed to solve the state of a full field, require the problem to be relatively *well-conditioned*, i.e. they do not allow a small change in the decision variables to translate into a very large change in the function's value, which will happen if a nearly-constant-voltage Section is encountered when stepping over voltage, or a nearly-constant-current Section is encountered when stepping over current.

To overcome this problem, it was chosen to represent the both the current and voltage of each solar cell as a piecewise-linear function of an expression composed of both the current and voltage. The electric state variable is defined as

$$b = V - R_{\mathrm{eq}} I, \tag{4}$$

where V is the cell voltage, I is the cell current, and $R_{\mathrm{eq}} = 1\ \Omega$ is here to make the units agree. A schematic of the piecewise-linear model is shown in Figure 4.

Phrased in terms of b, the current function is

$$I = \begin{cases} \frac{V_{\mathrm{cut}}-b}{R_{\mathrm{eq}}}, & b < b_{\mathrm{bypass}} \\ \frac{R_{\mathrm{sh}}I_{\mathrm{sc}}-b}{R_{\mathrm{eq}}+R_{\mathrm{sh}}}, & b_{\mathrm{bypass}} < b < b_{\mathrm{mp}} \\ \frac{V_{\mathrm{oc}}-b}{R_{\mathrm{eq}}+R_{\mathrm{s}}}, & b > b_{\mathrm{mp}} \end{cases}, \tag{5}$$

where R_{sh} is the shunt resistance of the cell, R_{s} is the series resistance, I_{sc} is the short-circuit current and V_{cut} is the voltage at which the bypass diode is activated; all these properties are obtainable from cell data-sheets. The open-circuit voltage, V_{oc}, is a function of temperature and other cell properties, calculated by the method used by Kribus and Mittelmann [4].

2.3 Pressure drop

Another modeled feature of the receiver which is needed for the full-field simulation is the pressure drop. The pressure-drop in each receiver is represented as a quadratic function of the flow rate, whose coefficients depend on the inlet temperature linearly:

$$\Delta P = (a_0 + T_{\mathrm{in}}a_1)\,\dot{m}^2 + (b_0 + T_{\mathrm{in}}b_1)\,\dot{m} + (c_0 + T_{\mathrm{in}}c_1)\,.$$

All constants in this expression are obtained by fitting of empirical measurements.

3 Field-level model

On the field level, the collectors may be connected to each-other thermally or electrically, which imposes constraints on the possible states of the field, so that for a short list of input variables that will be detailed below, it is possible to use the models detailed in the previous Section to solve for all the other variables.

First, out of the four temperatures defining the receiver's thermal state, three may be solved by a knowledge of the fourth. So, since T_{in} for the first receiver in a thermal string is dictated by the application, we may use it to solve for the other temperatures, and use Equation (3) to find the operating temperature of the next receiver in the string. This method is problematic, however, as some of the temperatures depend on q_{el}, which in turn depends on T_{pv}, which depends on the other temperatures.

To disentangle this feedback link, T_{pv} is used as an independent variable; its final value is determined by a non-linear equations solver such that it meets the constraint that both methods of calculating outlet power, i.e. Equations (1) and (2) agree,

$$q_{\mathrm{o}} = q_{\mathrm{o}}^{*}.$$

Similarly, the electric state must be found and meet the constraint of series connection – that is, all currents in a series must equal each other. Using these constraints, we have for N cells only $N-1$ equations. To complete the state, we take into consideration the ratio of total string voltage to string current, which is externally set by a device called the *inverter*, whose purpose is to find the optimal I-V

point, i.e the I-V point that produces the highest q_{el}. Modeling the inverter by this ratio, R_{inv}, we can rephrase the electric constraint for PV cell i as

$$I_i = \sum_{i=1}^{N} V_i / R_{\mathrm{inv}}.$$

As the calculation of I_i, V_i also depends on T_{pv}, these constraints must be solved together with the thermal constraints, as part of the same equation system.

Finally, the thermal state of a string depends on the flow rate, but the flow-rate cannot be set for each thermal string separately. For a field with several thermal strings, a global flow-rate is set, and the individual string flow rates must satisfy the condition that the pressure-drops in the thermal lines must equal each-other, and that the flow rates sum up to the global flow rate. The thermal string pressure drop is a sum of the pressure drop in each of the receivers in the string. Since the pressure-drops depend on T_{in}, these constraints must also be solved together with the other constraints in the same equation system.

4 Selection of solver

The solution of the non-linear equation-system defining the field (as introduced in the previous Section) is obtained using the Levenberg-Marquardt (LM) algorithm, which is designed to minimize the sum of squared errors of the constraint equations. An implementation of this algorithm that is available in the SciPy package [5] was used.

To determine its performance, the SciPy LM solver was compared to a different SciPy solver, based on the L-BFGS-B algorithm [6]. Both solvers were tasked with performing an annual simulation of a reduced version of the model in which the electric efficiency model is simplified and only the pressure-drop constraints must be solved. The LM algorithm solved this problem in 38 minutes, compared to 84 minutes with L-BFGS-B. The long times are a result of the inefficient implementation of the reduced problem, but the same reduction ratio can be seen in the number of calls to the underlying constraints calculation, which is independent on the efficiency of implementation.

Due to the clear advantage of the LM solver, it was chosen for solving the full problem.

5 Initial solutions

The LM solver arrives at a final solution by starting from an initial solution and changing the problem's state variables (T_{pv} for each receiver, electric state for each cell and flow-rate for each thermal string) in a way that reduces the sum of squared errors in each iteration, until no further significant reduction is possible. This method encounters problems when the initial solution lies close to some local minimum of the error, because the iteration will find that local minimum and will not be able to find the global minimum.

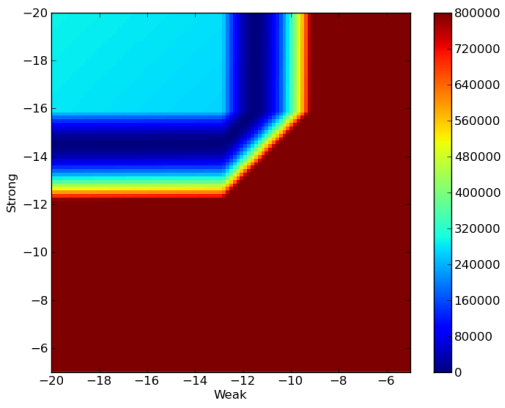


Figure 5: The error landscape for two receivers connected in electric series.

The problem is demonstrated in Figure 5. The sum of square errors of the electric constraints only is shown for a case of two receivers in electric series connection, over the electric state of each receiver. In the case shown, one receiver (labeled "strong") is illuminated with $1\ \mathrm{kW}$, and the other one (labeled "weak") is illuminated with $0.8\ \mathrm{kW}$. the inverter parameter is set to $R_{\mathrm{inv}} = 0.1\ \Omega$, a selection for which the weak receiver should be in the bypass mode (the left constant-voltage part of the I-V curve, where the bypass diode is active) and the strong receiver should be in the nearly-constant-current section of the I-V curve.

The error landscape clearly shows a "ditch" – a narrow band in which the error is very small compared to the rest of the landscape. If the solver reaches any point in that ditch, it will not move on to a different position readily. The ditch has one place which is its lowest point, at the correct solution, but this point will not be reached in reasonable time if the solver encounters the ditch at a farther point first.

The cause for the appearance of the ditch is the fact that when changing the electric state of a receiver on the flat-current part of his curve, the local current does not change – but the target current, which is computed from the sum of series voltages, is changes for *all receivers in the same series*, increasing the error for everyone.

To avoid this effect, an initial solution close enough to the true solution must be chosen. In general electric-circuit solvers such as SPICE [7], and in other similar fields, this is done by use of a continuation method: the solution of a slightly different problem which is known to be solvable is used as an initial guess for a problem closer to the one we target; its solution is used as initial solution for the next one, and so on until we have an initial solution good enough for our target problem. In this case we may solve the constraints for $R_{\mathrm{inv}} = 100, 10, 1, 0.5, 0.2, \ldots, 0.1$.

Clearly, this method multiplies the solution time several-fold; therefore, it cannot be used. Instead, we must find an initial solution based on the specific problem's features. In this problem, we use the fact that while the I-V curve depends on temperature, that dependence is not large. Hence, we can solve the piecewise-linear electric model for some temperature in the general neighbourhood of the final solution (a neighbourhood that is tens of degrees-Celsius wide). Using the exactly-solvable linear

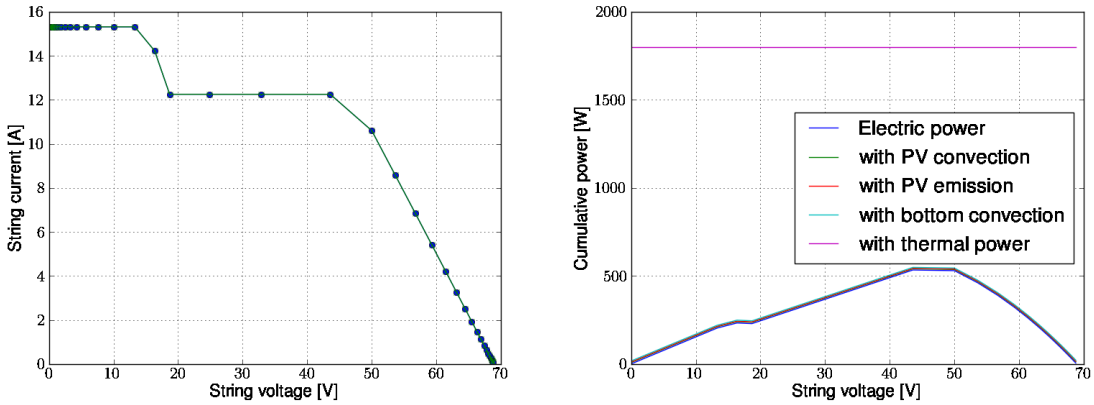


Figure 6: Validation run results: current-voltage (left) and power-voltage (right).

model for this temperature, we get an initial solution for the electric state that is close to the final solution and is therefore solvable.

Other than the electric variables, there is no similar problem with the temperature and flow-rate variables, so there is no need to find an initial solution close to the final one; such a solution, however, would still be advantageous, as the time it takes to reach the final solution gets longer as the initial solution gets farther from the final one. The method for finding such a solution could be the use of already-solved problems.

In an experiment based on the simplified problem described in Section 4, the method of using the previous time-interval's solution as the current time-interval's initial solution was tested. The number of inner-function calls for using a default initial solution in which all flow-rates are equal was compared to the number of calls using the previous solution: the default initial solution resulted in 1.417 times the calls of the previous-time-point initial solution. This already shows a lot of improvement, but it may be tuned even further, by finding solved points whose conditions are closer to the currently-solved problem by a better heuristic than taking the previous hour; it is speculated that the same hour of the previous day would be closer, but it was not tested.

6 Results

The simulation software was validated by simulating the case of two electrically series-connected receivers, one illuminated with $1\ \mathrm{kW}$, and the other being illuminated with $0.8\ \mathrm{kW}$. 50 different values are used for R_{inv}, equally distributed on the log scale from 10^{-3} to 10^{3}. The simulation produces an I-V curve for the electric series (Figure 6, left) which reproduces the form of two steps that is expected from literature and experiments; the power-voltage curve (Figure 6, right) also shows the expected form, of a double-peak curve. the smaller peak, on the left, is barely visible due to the low difference in illumination between the receivers, and gets more pronounced if that difference is increased. Furthermore, the cumulative power including electric output, thermal output and losses, always sums up to the input power, which shows that the model maintains the energy-conservation law.

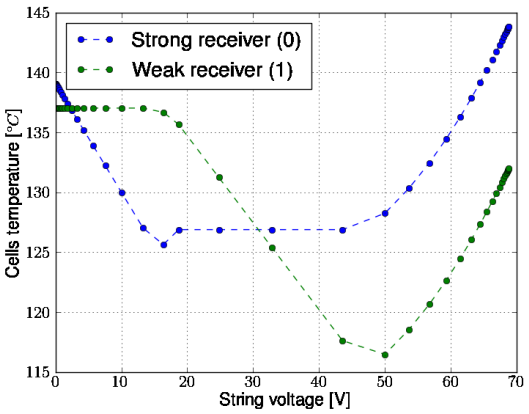


Figure 7: Temperatures of electrically series-connected receivers over string voltage.

Using the results of the same case, we may examine also that the temperature is predicted correctly. The temperature graphs for the weak and strong receivers is shown in Figure 7. Two features are noticeable in that graph: minimum-temperature points for both receivers; and inequality of the short-circuit (zero voltage) temperature and the open-circuit (max. voltage) temperature.

The minimum points in the temperature curves correspond each to the peak-electric-power point of the respective receiver. The strong receiver achieves its peak efficiency at a string voltage of about 18 V; and at that point the heat that passes on to the thermal block is the lowest, therefore the temperature increases the least. The weak receiver only reaches his peak electric efficiency at about 50 V string voltage, and at that point its temperature is lowest.

As the electric efficiency of both receivers is exactly zero at both ends of the curve, disconnected receivers should have the same temperature on both ends; in Figure 7, however, there is a slight inequality: the strong receiver is hotter on the open-circuit point than on the short-circuit point, for the weak receivers the effect is reversed. The reason for this is that close to the short-circuit point of the string, the weak receiver is in bypass mode, and his bypass diode consumes some of the power produced by the strong receiver - unloading power from the strong receiver to the weak one.

7 Conclusion

A simulation program was developed for predicting the performance electrically and thermally connected CPV/T collectors. The main challenge in creating such a simulation is the need to account for the interdependence of a large number of variables, including temperatures, electric state variables, and flow rates. To account for all the interdependent effect, the system was modeled as a non-linear equation system; a solver algorithm was selected and various aspects of the solution method were tuned for the particular problem.

Results of validation runs show that the model reproduces expected results, and also provides insight into the thermal/electric behaviour of the system and the dependence within it.

The running time is reasonable, and will allow usage of the program on a regular workstation. There is still much room for code optimization and various other improvements to the selection of initial solutions which will allow the model to scale up to larger simulated systems. The goal of the project, therefore, is achieved.

References

1. Holman JP. Heat Transfer. McGraw/Hill; 2002.
2. Quaschning V, Hanitsch R. Numerical simulation of current-voltage characteristics of photovoltaic systems with shaded solar cells. Sol. Energy. 1996; 56(6).
3. Petrone G, Spagnuojo G, Vitelli M. Analytical model of mismatched photovoltaic fields by means of Lambert W-function. Sol. Energy Mater. Sol. Cells. 2007; 91:1652-1657.
4. Mittelman G. Cogeneration With Concentrating Photovoltaic Systems. Tel Aviv University; 2006.
5. Jones E, Oliphant T, Peterson P, others. SciPy: Open source scientific tools for Python. 2001–. www.scipy.org
6. Zhu C, Byrd RH, Lu P, Nocedal J. A Limited Memory Algorithm for Bound Constrained Optimization. Northwestern University; 1994.
7. Quarles, Thomas L. Analysis of Performance and Convergence Issues for Circuit Simulation. EECS Department, University of California, Berkeley; 1989. Available online: http://www.eecs.berkeley.edu/Pubs/TechRpts/1989/1216.html

Towards Optimized Parallel Tempering Monte Carlo

Marco Müller

Universität Leipzig
Institut für Theoretische Physik
Vor dem Hospitaltore 1
04103 Leipzig

E-mail: mueller@itp.uni-leipzig.de

Abstract:
Parallel tempering Monte Carlo methods are an important tool for numerical studies of models with large complexity, since interesting questions, like the origin of phase transitions and structure formation, can only be tackled by means of statistical analysis. This work introduces the method and discusses ways of improving performance when using many-core architectures.

---

1 Introduction

Statistical mechanics treats systems with sizes of the order of 10^{23} constituents. These complex systems often cannot be solved analytically in principle or in practice. Monte Carlo simulations have been proven a useful tool for estimating quantities in complex systems. Striving for a better understanding, high precision results have to be achieved. Among others, three major generalized ensemble methods were developed: Multicanonical simulations [1], Wang-Landau sampling [2] and Parallel Tempering [3, 4, 5].
Computer technology releases have changed in the last few years. The speed of central processing units (CPUs) got stuck at about 3 GHz due to the difficulties involved in producing smaller electronics at the edge of the physically possible. A shift towards parallel computing can be observed, as nowadays computers get faster by using multi-core technology. As the parallel tempering Monte Carlo method is by design easy to parallelize, it seems to be the natural choice for the parallel treatment of complex systems. This work focuses on the implementation of a parallel tempering simulation that allows for optimizations to be tested for systems exhibiting free-energy barriers associated with phase transitions.

For the sake of simplicity, the implementation was tested first for the well-known Ising-model [6]. It describes a set of spins $s_i, i = 1, \ldots, N$, each being in one of two states, nearest neighbours interacting

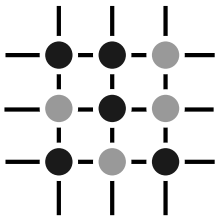


Figure 1: Schematic excerpt from an Ising magnet with $N = 3 \times 3 = 9$ spins being in one of two states, either bright or dark, on a regular twodimensional grid.

with a constant coupling strength J. The Hamiltonian of a system of N spins reads in Potts-model notation

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} \delta_{s_i s_j}, \qquad s_i \in \{0, 1\}, \tag{1}$$

with $\langle \cdot, \cdot \rangle$ denoting pairs of nearest neighbours and δ being the Kronecker symbol. It is a model for magnetism exhibiting a temperature-driven second-order phase transition between ferro-and paramagnetism. On regular grids in two dimensions, the Ising model was exactly solved by Onsager in the thermodynamic limit $N \to \infty$ [7]. For Ising magnets with a *finite* number of spins located on a regular 2D-grid with periodic boundary conditions, Beale was able to provide an exact solution [8]. Therefore the model is perfectly suited to test new algorithms.

2 Monte Carlo Methods

2.1 Importance Sampling

For a classical system, the probability of finding a microstate μ with energy E in a heat bath at temperature T with an associated inverse thermal energy $\beta = 1/k_B T$ is given by

$$\mathcal{P}^B(\mu) = \frac{1}{\mathcal{Z}} e^{-\beta E(\mu)}. \tag{2}$$

The Boltzmann constant $k_B \approx 1.38 \cdot 10^{23} J/K$ was set to 1 throughout this work, therefore β is referred to as an inverse temperature. The quantity $\mathcal{Z}$ denotes the canonical partition function

$$\mathcal{Z} = \sum_{\mu} e^{-\beta E(\mu)} = \sum_{E} \Omega(E) e^{-\beta E}, \tag{3}$$

where $\Omega(E)$ is the density (or better number) of states for given energy E. Simulations should generate microstates with the same probability (2). In principle, the information about the underlying system is not sufficient to draw microstates directly according to the probability $\mathcal{P}^B$. In practice they are constructed as members of the equilibrium distribution of a *Markov process*. Let $S = \{\mu_1, \mu_2, \ldots, \mu_n\}$ be the set of possible states of the system. A Markov process is a stochastic process that changes the microstate μ_i of the system to another state μ_j with a transition probability p_{ij}

that is independent of all preceding states. The transition probabilities have to fulfil the following properties:

$$p_{ij} \geq 0 \quad \forall i,j; \qquad \sum_j p_{ij} = 1 \quad \forall i; \qquad \sum_i p_{ij}\mathcal{P}_i^B = \mathcal{P}_j^B \quad \forall j. \tag{4}$$

The first condition (4) ensures *ergodicity* of the Markov process, i.e. every state $\mu_j \in S, j = 1, \ldots, n$ can be reached starting from any state $\mu_i \in S, i = 1 \ldots n$ (not necessarily within a single step). The other properties are for normalization and *balance*. Balance means the Boltzmann distribution $\mathcal{P}^B$ is a fixed point of the transition probability.

An algorithm for the construction of a Markov chain, satisfying the conditions (4) was first proposed by Metropolis [9] for systems that allow the calculation of the internal energy $E_i := E(\mu_i)$. The transition probability reads

$$p_{ij}^{metr} = \min\left\{1, e^{-\beta(E_j - E_i)}\right\}. \tag{5}$$

An update of the system is accepted every time the system gets to a configuration with a lower energy than the current one, but is only accepted with probability p_{ij}^{metr} when the internal energy increases. Algorithm 1 describes the Metropolis method in more detail for applications on spin lattices.

2.2 Parallel tempering

For systems exhibiting a rather complex transition behaviour, such as spin glasses or proteins, the Metropolis algorithm can become extremely inefficient near or at transition points. In the parallel tempering Monte Carlo algorithm, N_r non-interacting replicas of the system are simulated at a set of inverse temperatures $\{\beta_1, \beta_2, \ldots, \beta_{N_r}\}$. After a number of canonical Monte Carlo updates on each system, a replica swap is attempted. Applying the Metropolis criterion (5), the transition probability for swapping systems i and j can be calculated

$$p_{ij}^{pt} = \min\left\{1, e^{\Delta}\right\} \text{ for } \Delta = (\beta_j - \beta_i)\left[E_j - E_i\right]. \tag{6}$$

Algorithm 2 shows the implementation of the parallel tempering Monte Carlo method in detail. The partition function of the coupled replicas is now the product of the canonical partition functions

$$\mathcal{Z}_{pt} = \prod_{i=1}^{N_r} \mathcal{Z}(\beta_i) = \prod_{i=1}^{N_r} \sum_j \Omega(E_j) e^{-\beta_i E_j}, \tag{7}$$

as the non-interacting systems are statistically independent. The right hand side of equation (7) again employs the density of states Ω.

By introducing a global swap, a fixed replica can now perform a random walk in temperature space. Complex energy landscapes can be traversed faster, as replicas are now capable of getting to high temperatures, where autocorrelation times are much shorter than at low temperatures or near critical points. To collect as many statistically independent samples at low temperatures as possible,

one should try to minimize the time for each replica to travel across temperature space. This can be achieved by choosing the set of inverse temperatures carefully. Several approaches have been proposed, ranging from diffusion based methods [10] to autocorrelation dependent spacing [11]. However, it is not obvious how to apply these optimizations efficiently for the usage on many-core architectures.

Algorithm 1: Metropolis update

```
choose a spin
choose a new value for that spin
draw a random number r ∈ (0, 1]
if r < p_ij^metr then
    accept new state
else
    reject new state
end
```

Algorithm 2: Parallel tempering Monte Carlo sweep

```
for every replica
do
    for a number of sweeps do
        metropolis_update(replica)
    end
    choose a replica S_i randomly
    choose an adjacent replica S_j out of the neighbours of S_i
    draw a random number r ∈ (0, 1]
    if r < p_ij^pt then
        swap replicas
    end
end
```

3 Results

3.1 Implementation

The parallel tempering Monte Carlo simulation for the more general Edwards-Anderson-Potts spin glass Hamiltonian,

$$\mathcal{H} = -\sum_{\langle i,j \rangle} J_{ij}\delta_{s_i s_j},\ s_i \in \{1 \cdots q\}\,, \tag{8}$$

was implemented from scratch in C++, using the Message-Passing-Interface (MPI) for parallelization. For $J_{ij} = J\ \forall i, j$, the Hamiltonian reduces to the Potts model. In this work, only the case of $q = 2$ was treated, which corresponds to the Ising model. The inverse temperatures β_i are exchanged instead of swapping replicas to reduce the communication overhead introduced by replica swaps. This approach

leads to arbitrary permutations of the inverse temperatures on the processes that run in parallel. To assign measured observables to the correct temperature, a proper bookkeeping is needed. The swapping attempts can be implemented in one of two ways:

- Every process communicates with a current neighbouring (i.e. neighbour in the inverse temperature space) process. If an exchange is to be performed, all neighbours of the two processes involved have to be informed about the swap.
- A master process handles the exchange of inverse temperatures between processes, by gathering information from all processes and sending back to them the new inverse temperatures.

Here, the latter approach was implemented, leading to a simpler bookkeeping. Every process has to wait for the master process to be finished, so a serial bottleneck is employed. This is called the synchronisation barrier. Collective communication can be used when implementing the master-slave scheme, which can be faster (at least not slower) than an all-point-to-point communication.

3.2 Verification

The implementation was tested using the exact solution of the finite two dimensional Ising model on regular periodic lattices near the critical point, in Potts model notation at $\beta_c = \log(1 + \sqrt{2}) \approx 0.881$. The specific heat is defined as

$$c_V(\beta) = \frac{1}{N}\left(\frac{\partial \langle E \rangle}{\partial T}\right)_{V,N} = \beta^2 \frac{\langle E^2 \rangle - \langle E \rangle^2}{N} \tag{9}$$

with V being the volume of the system (it corresponds to the number of particles for spin lattices). As the specific heat is the first-order derivation of the internal energy E with respect to temperature, the quantity is very sensible to phase transitions (Figure 2).

3.3 Inverse temperature distribution

The distribution of the inverse temperatures is important, as the acceptance rate decreases exponentially with $\Delta\beta$. For the transition probability (6) to be reasonable large, a sufficient overlap in the energy histograms is needed. Let $H_\beta(E) := \Omega(E)e^{-\beta E}/\mathcal{Z}$ be the normalized histogram at inverse temperature β. The overlap between two histograms H_{β_i} and H_{β_j} is defined by the area that is enclosed by both histograms, for discrete energies reading

$$q_{ij} = \sum_E \min\left\{H_{\beta_i}(E), H_{\beta_j}(E)\right\}. \tag{10}$$

Since $\sum_E H_\beta(E) = 1$ and $H_\beta \geq 0\,\forall E$, the overlap q_{ij} is a quantity in the unit interval. The temperature distribution for the two-dimensional Ising model can be chosen so that the overlap between all adjacent heat baths is nearly constant. Therefore, an approximation of the density of states was calculated using histograms that were obtained in previous runs at different temperatures. The density of states is iteratively reweighted [12, 13] to a new set of inverse temperatures, until the overlap is constant. The acceptance rate, the ratio of accepted parallel tempering steps to the total number of update attempts,

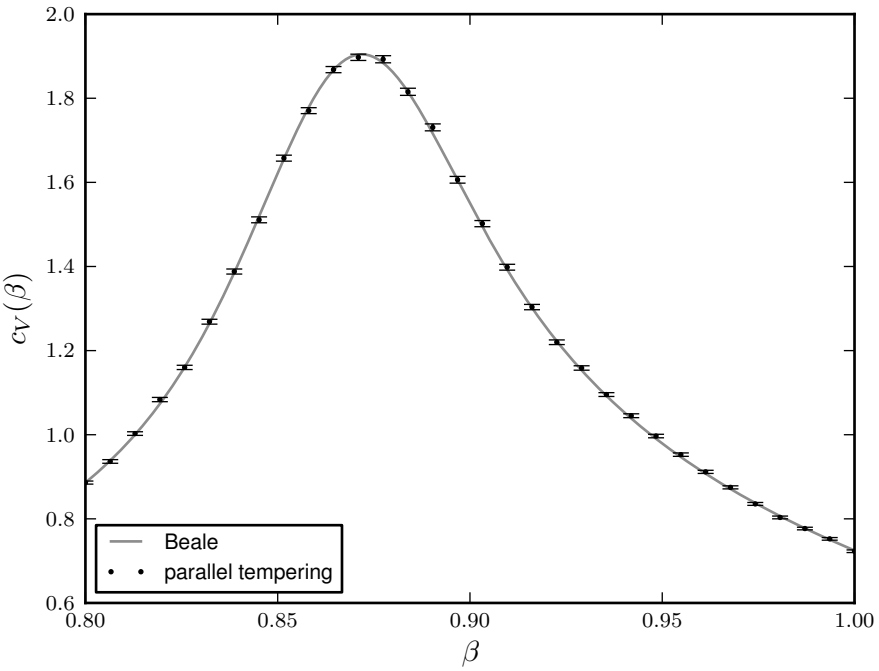


Figure 2: Specific heat near the transition point of the 2D-Potts model using parallel tempering Monte Carlo with the following parameters: $q = 2$, grid dimensions = 32×32, number of energies = 2^{17}, number of jackknife blocks = 2^9 . The peak at the inverse energy $\beta_t \approx 0.872$ is associated with a transition of the internal energy.

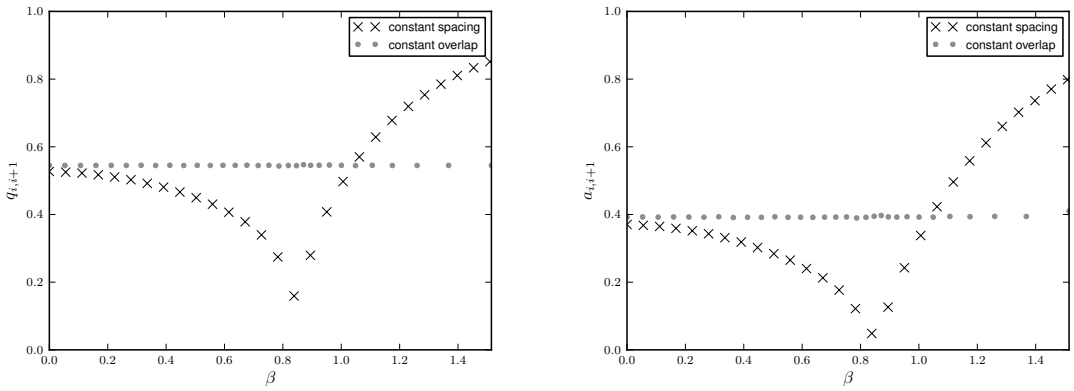


Figure 3: Overlap q_{ij} of different distributions of 32 inverse temperatures (left) and the acceptance rate a_{ij} (right). A pronounced peak of the distribution using constant spacing can be seen near the transition inverse temperature of the system at $\beta_t \approx 0.872$. Also, the acceptance rate shows the same behaviour as the overlap. Parameters of the simulation: $q = 2$, grid dimensions = 32×32, parallel tempering sweeps = 2^{20}, using 20 local Metropolis sweeps per replica exchange attempt.

is a quantity that measures how fast a fixed replica can get from one heat bath to another. Overlap and acceptance rate for the 32×32 Ising lattice are shown in Figure 3.

3.4 Refinement and replicated temperature partition simulations

When using many cores in a parallel tempering simulation, it is not so clear how the heat baths should be chosen. Having a fixed temperature interval, there are at least two possible ways of improving the simulation by using more replicas:

- Increasing the number of inverse temperatures, thus refining the interval. The number of replicas N_r coincides with the number of heat baths $N_r = N_\beta$.
- Keeping the number of heat baths fixed and using more than one replica at each temperature. Every heat bath gets n_β replicas, hence $N_r = n_\beta \cdot N_\beta$.

For the latter approach the partition function becomes

$$\mathcal{Z}_{pt} = \prod_{i=1}^{N_r} \mathcal{Z}(\beta_i) = \left(\prod_{j=1}^{N_\beta} \mathcal{Z}(\beta_j) \right)^{n_\beta}, \tag{11}$$

because a number of n_β factors of the left hand side are equal.

When a fixed replica got from the lowest temperature β_{max} to the highest temperature β_{min} and back again, it is said that a *tunnel event* occurred, a term borrowed from first-order barrier models. The average number of tunnel events of all replicas is denoted by N_t. The tunnel time per replica τ is then defined by

$$\tau = \frac{1}{N^2} \frac{N_{mc}}{N_t}, \tag{12}$$

where N_{mc} is the total number of Monte Carlo updates that are conducted. The coefficient N^{-2} compensates for the random walk in the energy space. As the energy of spin lattices with N spins is of order $\mathcal{O}(N)$, the time to traverse an interval ΔE is of order $\mathcal{O}(N^2)$. The tunnel time for both a refinement scheme and a replicated temperature partition simulation for the 2D-Ising model is shown in Figure 4.

4 Conclusion and Outlook

An introduction to parallel tempering Monte Carlo methods has been given. An implementation, allowing for optimizations to be tested, was verified using the exact results of the Ising model. Open questions about how to apply the method to architectures with many cores were discussed and two approaches, the refinement and replicated temperature partition scheme were given. First results of both schemes for the Ising model have been shown. However, the Ising model has no complicated free energy landscape, therefore it is hard to measure the efficiency of either scheme. Parallel tempering

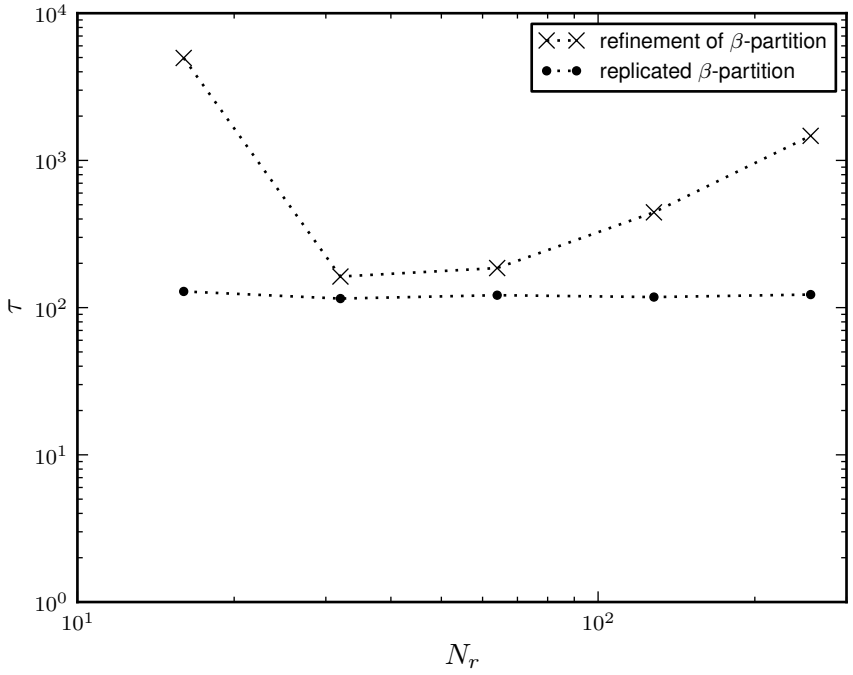


Figure 4: Tunnel time per replica in the 32×32 2D-Ising model over the total number of replicas $N_r \in \{16, 32, 64, 128, 256\}$. The simulation was carried out with 20 Monte Carlo sweeps per parallel tempering sweep, 2^{18} parallel tempering sweeps, a total number of $20 \cdot 2^{18} \cdot 32^2 \approx 10^{10}$ Monte Carlo updates. For the replicated β-partition simulation, a fixed partition of 16 different β-values was used. The partition was constructed to have a constant overlap in the histograms in the fixed interval $\beta \in [0, 1.73]$, hence $n_\beta \in \{1, 2, 4, 8, 16\}$. The tunnel time per replica is nearly constant when increasing the number of replicated partitions, therefore the total number of tunnel events increases linearly with the number of replicas (processes). Having only 16 inverse temperatures with constant spacing creates only a little overlap in the histograms, therefore the tunnel time is big for the β-refinement curve. However, increasing the number of temperatures also increases the overlap and the tunnel time decreases. Using many temperatures when attempting parallel tempering updates on adjacent replicas increases the tunnel time, because the improvement in the overlap between next-nearest neighbours cannot be used by the algorithm that only allows for one step in the inverse temperature space, thus leading to a factor $\mathcal{O}(N_r^2)$ that accounts for a directed random walk in the β-space.

methods are designed to be more efficient applied on models exhibiting more complex free energy landscapes.

In the future, more data has to be collected on complex models to evaluate the important question of how to use many-core architectures in an optimal way. Also, it would be very interesting to implement parallel tempering algorithms on shared memory systems like cost-efficient general purpose graphical processing units.

Acknowledgement

I wish to sincerely thank Dr. Thomas Neuhaus and Dr. Michael Bachmann for their support and advice and the vast amount of time they could spare for this project, and everybody who was involved in planning the Guest Student Programme, especially Robert Speck and Mathias Winkel. I want to express my gratitude for my fellow guest students for making this summer a rich experience and for being open for so many discussions. Finally, I would like to thank Professor Wolfhard Janke in Leipzig for caring so much about his students and supporting my application.

References

1. Berg BA, Neuhaus T. Multicanonical algorithms for first order phase transitions. Physics Letters B. 1991;267:249 – 253.
2. Wang F, Landau DP. Efficient, Multiple-Range Random Walk Algorithm to Calculate the Density of States. Physical Review Letters. 2001;86:2050–2053.
3. Swendsen RH, Wang JS. Replica Monte Carlo Simulation of Spin-Glasses. Physical Review Letters. 1986;57:2607–2609.
4. Geyer CJ. Markov Chain Monte Carlo Maximum Likelihood. Computing Science and Statistics, Proceedings of the 23rd Symposium on the Interface. 1991;156–163.
5. Hukushima K, Nemoto K. Exchange Monte Carlo Method and Application to Spin Glass Simulations. Journal of the Physical Society of Japan. 1996;65:1604–1608.
6. Ising E. Beitrag zur Theorie des Ferromagnetismus. Zeitschrift fur Physik. 1925;31:253–258.
7. Onsager L. Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition. Physical Review. 1944;65:117–149.
8. Beale PD. Exact Distribution of Energies in the Two-Dimensional Ising Model. Physical Review Letters. 1996;76:78–81.
9. Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E. Equation of State Calculations by Fast Computing Machines. The Journal of Chemical Physics. 1953;21:1087–1092.
10. Katzgraber HG, Trebst S, Huse DA, Troyer M. Feedback-optimized parallel tempering Monte Carlo. Journal of Statistical Mechanics: Theory and Experiment. 2006;03:P03018.
11. Bittner E, Nußbaumer A, Janke W. Make Life Simple: Unleash the Full Power of the Parallel Tempering Algorithm. Physical Review Letters. 2008;101:130603.
12. Ferrenberg AM, Swendsen RH. New Monte Carlo technique for studying phase transitions. Physical Review Letters. 1988;61:2635–2638.
13. Ferrenberg AM, Swendsen RH. Optimized Monte Carlo data analysis. Physical Review Letters. 1989;63:1195–1198.

Scaling of Linear Algebra Routines on the IBM BlueGene/P System JUGENE

Elin Solberg

University of Gothenburg
Department of Mathematical Sciences
Chalmers Tvärgata 3
41258 Gothenburg
Sweden

E-mail: solberg@student.gu.se

Abstract:
Three different solvers of the dense real symmetric eigenproblem are available in the parallel linear algebra library ScaLAPACK – a fourth one, building on the algorithm MR^3, is planned to be included in a future, not yet announced, version of the library. This report presents the results of benchmarking the three library solvers as well as a still experimental version of the MR^3 solver.
The benchmarking was performed on the IBM BlueGene/P system JUGENE, using up to 8192 cores to solve problems with a maximum matrix size of 122880×122880.
Two main cases were investigated: (1) eigenvalues randomly spread in a given interval and (2) massively clustered eigenvalues. In the first case the scalability and accuracy of the new MR^3 solver did not quite answer expectations, whereas in the second case, the new solver proved to be a promising alternative to the existing routines.

1 Introduction

The problem of computing eigenvalues and eigenvectors, together referred to as eigenpairs, of a matrix appears in various applications of computational sciences. One example of such an application is Density Functional Theory (DFT), which is in [1] described as an "important, and most successful approach for studying electronic structure phenomena in materials". In DFT computations, typically at least 20 – 25% of the eigenpairs of large, dense matrices need to be computed.

To meet the needs of DFT, as well as other applications, making efficient use of the growing computing power of modern parallel systems, fast and highly scalable routines for solving the eigenproblem are needed. Several different routines are available, and the purpose of this project has been to compare the performance and scalability of four parallel solvers of the dense symmetric eigenproblem on the IBM BlueGene/P system JUGENE.

The performance of the routines has mainly been tested on matrices varying in size from 1000×1000 to 12288×12288, using between 64 and 1024 processors. Some first results using up to 8192 processors for problems of size up to 122880×122880 are also presented.

A main question throughout this project has been how the performance is influenced by the matrix spectrum. Attempting to give a first, far from general, answer to that question, two types of eigenvalue distributions have been tested:

1. eigenvalues randomly generated in a given interval, and
2. all eigenvalues clustered around ε, except one which has the value 1.

In the first case, the interval in which eigenvalues were generated was varied in size from [0,1] to [-100,20000], where a smaller interval is likely to cause more and larger clusters of eigenvalues, possibly affecting performance and/or accuracy of the computations. These two eigenvalue distributions are also considered in [2], p. 58.

With two of the solvers, optionally only a part of the eigenpairs, defined by the user, may be computed. This option has been tested under the same conditions as described above, specifying ten percent of the eigenpairs to be computed.

2 The Solvers Examined

In this section first a brief introduction is given to the four examined eigensolvers; PDSYEV, PDSYEVD, PDSYEVX, and PDSYEVR. After that a special, and for this project important, feature of PDSYEVX is discussed in some detail. Finally the storage of matrices when using ScaLAPACK [3] routines is outlined, and motivations are given for the block sizes chosen for each of the solvers.

Throughout this and subsequent sections the number of processors will be denoted by p and the matrix size will be denoted by n, meaning that e.g. a 1000×1000 matrix has the matrix size $n = 1000$.

The four eigensolvers examined in this project all proceed in the following three phases:

1. reduction of the full matrix to a tridiagonal matrix with the same eigenvalues as the original one,
2. solution of the tridiagonal eigenproblem,
3. back-transformation of the eigenvectors of the tridiagonal matrix into the eigenvectors of the full matrix.

The first and third phases are performed in a similar way by all solvers, and both have a serial complexity of $\mathcal{O}(n^3)$ and require $\mathcal{O}(n^2)$ memory, so that with perfect scalability the parallel complexity would be $\mathcal{O}(n^3/p)$ and $\mathcal{O}(n^2/p)$ memory would be needed by each processor [4].

The main difference between the four eigensolvers is what algorithm is used for the second phase, i.e. for computing the eigenpairs of the tridiagonal matrix. In the routine PDSYEV, the tridiagonal eigenproblem is solved using a parallel QR algorithm, which has a complexity of $\mathcal{O}(n^3/p)$ for computing all eigenpairs of a tridiagonal matrix [2]. The solver PDSYEVD instead uses the divide and conquer method for the second phase computations, which in the worst case has the same complexity as the QR

algorithm, but in practice often performs better than that, which can also be observed in the results presented in this report, see Section 5 [5]. A drawback of PDSYEVD is that it requires $\mathcal{O}(n^2)$ extra memory, so that it cannot be used for as large matrices as the other solvers.

As mentioned in the introduction, there are applications where the possibility to compute only a part of the eigenpairs would be desirable. Whereas neither PDSYEV nor PDSYEVD have this option, both PDSYEVX and PDSYEVR do have it. In PDSYEVX, bisection is used to compute k eigenvalues of the tridiagonal matrix at a cost of $\mathcal{O}(kn/p)$ flops and inverse iteration is then used to compute the corresponding eigenvectors. When eigenvalues are well separated, the complexity of inverse iteration is also $\mathcal{O}(kn/p)$, but in the worst case, when eigenvalues are heavily clustered, the complexity grows to $\mathcal{O}(k^2n)$ – i.e. the routine becomes slow, and what is more, it does not scale at all. The reason for this will be discussed in some more detail in the next section.

The three solvers presented above, PDSYEV, PDSYEVD and PDSYEVX, are all routines from the library ScaLAPACK. The fourth solver examined in this project is a still experimental version of PDSYEVR, written by Christof Vömel[1], which uses a recently developed algorithm, the parallel multiple relatively robust representations (MR^3) algorithm, to solve the tridiagonal eigenproblem [4, 6]. PDSYEVR is planned to be included in the next version of ScaLAPACK. As mentioned, this solver can be used to optionally compute only a specified part of the eigenpairs. Computing k eigenpairs, the parallel complexity of MR^3 is $\mathcal{O}(kn/p)$, independent of the matrix spectrum.

2.1 PDSYEVX and the Parameter ORFAC

In theory, eigenvectors of a symmetric matrix, corresponding to distinct eigenvalues, are orthogonal. However, the approximate eigenvectors computed by the inverse iteration used in PDSYEVX are not always orthogonal. This is especially the case for eigenvectors corresponding to eigenvalues which are close to each other. To handle this problem, each computed eigenvector is orthogonalized, using a modified Gram-Schmidt method, against the already computed eigenvectors corresponding to eigenvalues that are close. Here 'close' is defined by the user; eigenvectors are orthogonalized if the distance between their corresponding eigenvalues is less than ORFAC $\cdot \|A\|$, where A is the matrix whose eigenpairs are being computed, and ORFAC is a parameter to be set by the user. Orthogonalization of the eigenvectors of clustered eigenvalues is what reduces the performance of serial inverse iteration from complexity $\mathcal{O}(kn)$ to $\mathcal{O}(k^2n)$ when computing k eigenpairs.

In order to avoid extensive communication, PDSYEVX is implemented in such a way that orthogonalization is done on one processor per cluster. This is the reason for the – in the worst case – complete loss of scalability mentioned in the previous section. This also leads to memory imbalance, since all orthogonalized eigenvectors have to be stored in the memory of one processor.

As expected, setting ORFAC to 0 leads in general to nonorthogonal computed eigenvectors. The default value of ORFAC, 10^{-3}, on the other hand produces orthogonal eigenvectors but is generally rather slow – perhaps slower than necessary in many cases. In this project ORFAC $= 10^{-4}$ has been used unless otherwise stated, in some cases leading to better performance than with the default value, without altering the orthogonality of the eigenvectors.

[1] School of Engineering, Züricher Hochschule für Angewandte Wissenschaften, Zürich.

2.2 Matrix Storage Using ScaLAPACK

Whenever ScaLAPACK routines involving dense matrices are used, the matrices are expected to be distributed over the processes in a certain way: the (two-dimensional) block-cyclic way. This section first deals with explaining the block-cyclic distribution, and then values of the block size parameter used in this project are motivated.

Two main objectives need to be considered when distributing matrices over a grid of processes: maximizing workload balance and minimizing communication between the processes. These two objectives often contradict each other, and thus a good tradeoff between them is relevant to achieve high performance.

Using the block-cyclic distribution, the matrix is divided into blocks of size $mb \times nb$. The blocks are distributed over the process grid cyclicly in both dimensions, as illustrated by the following example: Assume we have a 9×11 matrix and chose the row block size mb to be 2, and the column block size nb to be 3, see the left part of Figure 1. Assume also that we have a 2×2 process grid, with processes numbered from left to right starting with the first row, and counting from 0. The blocks will then be assigned to the processes as shown in the right part of Figure 1. Each process stores its parts of the

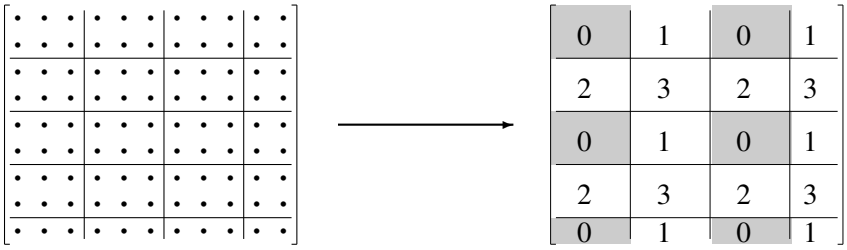


Figure 1: Two-dimensional block-cyclic distribution of a 9×11 matrix over a 2×2 grid of processes.

entire matrix as one smaller matrix in memory. In the example above, the grey blocks in the figure are thus stored as one matrix in the zeroth process.

In order to achieve a good tradeoff between workload balancing and communication minimization, the block sizes should be chosen with some care. Small blocks lead generally to much communication since neighboring elements are more often stored in different processes, whereas with large blocks the workload balancing might be bad. For a more detailed explanation on this topic see [3].

In the context of eigenproblems, the concerned matrices are square, and in all tests performed in this project the grid of processes was either square or $m \times 2m$, where m is an integer. Considering this setup, the same block size was used in both dimensions in all test runs, i.e. $mb = nb$ in all tests. Next, motivations are given for the choice of block size used for the individual eigensolvers. Note especially the unexpected way in which block sizes are used in PDSYEV.

To decide what block sizes to use with the solvers PDSYEVD, PDSYEVX and PDSYEVR, a few preliminary test runs were performed on 256 and 1024 processors with $nb = 20$, 32, 40 and 64. No major effects were detected when changing the block size, yet the small differences that did occur led to choosing $nb = 64$ for PDSYEVX and PDSYEVR with up to 256 processors, and in all other cases $nb = 32$.

With PDSYEV the situation regarding block sizes is more complicated than with most other ScaLAPACK routines, which is however not obvious at first glance; using a too large block size may lead to an apparently inexplicable loss of scalability. The reason is to be found in a subroutine, DSTEQR2, called by PDSYEV, which solves the tridiagonal eigenproblem, computing on each process entire eigenvectors, so that entire columns of the eigenvector matrix need to be accessed and updated by each process. Therefore the matrix is redistributed before the call to DSTEQR2 over a new grid of processes, consisting of only one row, but keeping *the same* block size as before. With all processes aligned in one row, each row of blocks is distributed over the same processes, thus fulfilling the requirement that each process should access entire columns. In the previous example, this leads to a matrix distribution as depicted in Figure 2, which has only a slight load imbalance; the number of columns assigned to processor three is smaller than the number of columns assigned to the other processors.

0	1	2	3
0	1	2	3
0	1	2	3
0	1	2	3
0	1	2	3

Figure 2: Two-dimensional block-cyclic distribution of a 9×11 matrix over a 1×4 grid of processes.

This is of course only a toy example. We now move on to an example closer to "real life" computations, which shows more clearly the effect of choosing block size without considering the above discussion. Assume we want to compute the eigenpairs of a matrix of size $n = 12288$, using PDSYEV with 256 processors, originally structured in a 16×16 grid. Considering the choice of block size for the other solvers it does not seem unreasonable to choose $nb = 64$. This choice would lead to a total of $n/nb = 192$ blocks in each dimension, giving with the original processor grid $192/16 = 12$ blocks per processor in each dimension, or totally $12^2 = 144$ blocks per processor. The load balance on the original processor grid is thus perfect. Reordering the processors to a 1×256 grid however, the first 192 processors will receive 64 columns each, while the last 64 processors will idle during the entire QR computation. Hence using $nb = 64$ for a matrix of size $n = 12288$, DSTEQR2 does not scale at all to more than 192 processors.

In the tests of PDSYEV performed in this project nb was chosen with the above discussion in mind, ranging from 2 to 32 and depending on matrix size as well as number of processes.

3 Hardware and Software

All tests in this project were performed on the IBM BlueGene/P system JUGENE [7]. Its base component is the compute node of type 4-way SMP processor, with four 32-bit PowerPC 450 cores, each with 850 MHz. On one nodecard 32 compute nodes are mounted, and 32 nodecards in turn make up one rack. The complete JUGENE system consists of 72 racks, leading to a total of 73728

compute nodes, or 294912 cores. The overall peak performance is 1 Petaflop. Each node has a main memory of 2 Gbytes, which has been a limiting factor when performing tests with large matrices.

For all tests a testing program provided by my advisor Inge Gutheil was used, in which a set of eigenvalues are generated and used as the diagonal of a diagonal matrix, which is then transformed into a full matrix by Householder transformations with a random Householder vector. This setup allows for subsequent accuracy tests of the computed eigenvalues and eigenvectors. The testing program was compiled and linked using the script mpixlf77_r, which uses bgxlf_r version 11.1 for compiling, and links to the MPI supplied by IBM for JUGENE. The BLAS included in ESSL version 4.3 were used, together with public domain versions of BLACS 1.1 and ScaLAPACK 1.8.

4 The Tests Performed

This section provides an overview of the tests performed, before the results are presented in the next section.

The main part of the tests in this project were performed with 64, 128, 256, 512 and 1024 processors (cores), using square grids where possible and otherwise $2 \times 2m$ grids, where m is an integer. The tests were performed on matrices of size 1000 + $500k$ and 1024 + $512k$, where $k = 0, 1, \ldots, 22$. The eigenvalues of the test matrices were generated in two ways:

1. random, drawn from a uniform distribution on a given interval, and
2. all clustered around ε except one, which has the value 1[2].

In the first of the above two cases, the following five intervals were tested: [0,1], [-10,20], [-100,200], [-100,2000] and [-100,20000].

The above setup was tested computing all eigenpairs with each of the solvers as well as computing ten percent of the eigenpairs with PDSYEVX and PDSYEVR, exceptions being made in some cases for very slow computations, in order not to waste computing resources. Additionally a few preliminary tests with 1024, 2048, 4096 and 8192 processors were performed on matrices of size $n = 40960$, 81920 and 122880.

5 Results

In this section, the results of the tests described above are presented. Apart from run times, the orthogonality of computed eigenvectors will be discussed. To that end let Z be the matrix whose columns are the computed eigenvectors. Then the eigenvectors are considered numerically orthogonal if $\|Z^T Z - I\| = \mathcal{O}(n\varepsilon)$, and we hence would like $\|Z^T Z - I\|/(n\varepsilon)$ to be of moderate size. In the tests performed, the values were around or less than one, unless otherwise stated.

[2] ε = relative machine precision $\cdot$ base of the machine. On JUGENE $\varepsilon \approx 2.2 \cdot 10^{-16}$.

The residuals of the eigenpairs were also computed, and in all cases fulfilled $\|(A - \lambda I)z\| = \mathcal{O}(\varepsilon\|A\|)$, where λ is an eigenvalue, z its corresponding eigenvector and A is the matrix whose eigenpairs are being computed.

Before the results are presented, a note on the different intervals in which random eigenvalues were generated: The reason for testing different cases was that in a smaller interval (keeping n fixed), more and larger clusters are more likely to appear, which might influence performance and/or the orthogonality of computed eigenvectors. In general however, no such differences were observed wherefore in the following only the results for eigenvalues generated in the interval [0,1] will be presented.

5.1 All Eigenpairs, 64 - 1024 Processors

The run times of all four solvers, computing all eigenpairs using between 64 and 1024 processors can bee seen in Figure 3, in each case plotted against the matrix size. Note that in this and all subsequent plots a log-scale is used for both the x- and the y-axis. Noting the different y-axis scales, we see that PDSYEV (top left) is, as expected, much slower than the other solvers. PDSYEVD (top right) and PDSYEVX (bottom left) have approximately the same performance, although the latter is generally somewhat faster than the former. All three solvers scale well at least with large matrices, and the tendency is that the gaps between the different numbers of processors keep growing as n grows, so that it may be assumed that even better scalability would be reached with larger matrices.

With PDSYEVR, the performance using 64 - 256 processors is the same as with PDSYEVD and PDSYEVX. Using more processors however, the times *grow* for small matrices compared to using less processors, and even with the largest matrices tested, no scaling is achieved when increasing the number of processors from 512 to 1024. As we will see in Section 5.3, this problem with scalability seems to be present also when using more processors for larger matrices. With PDSYEVR the values of $\|Z^T Z - I\|/(n\varepsilon)$ were generally between 10 and 100, which is acceptable, but shows that the orthogonality of the eigenvectors was not quite as good as with the other solvers.

Whereas altering the size of the interval in which random eigenvalues were generated did not appreciably alter the results, letting all eigenvalues except one be clustered around ε indeed influenced the performance of PDSYEVX and PDSYEVR dramatically. The results are displayed in Figure 4, where the results of PDSYEV (top left) and PDSYEVD (top right) are included for the sake of completeness, although the run times are nearly identical to the corresponding ones in Figure 3.

As expected, with clustered eigenvalues the performance of PDSYEVX is greatly reduced (see the bottom left plot in Figure 4), due to the orthogonalization discussed in Section 2.1. In order not to waste computing resources, only two tests were performed with PDSYEVX in this case, using 64 and 1024 processors respectively. In both cases the maximal matrix size was $n = 3500$. In addition to the very bad performance we also note that the routine does not scale at all. The reason for this is that all but one of the eigenvalues belong to one and the same cluster, so that all orthogonalization is performed on only one processor. In spite of the orthogonalization, the value of $\|Z^T Z - I\|/(n\varepsilon)$ grew to 2000 in the worst case.

Somewhat surprisingly, PDSYEVR (bottom right plot in Figure 4) does not have a problem with

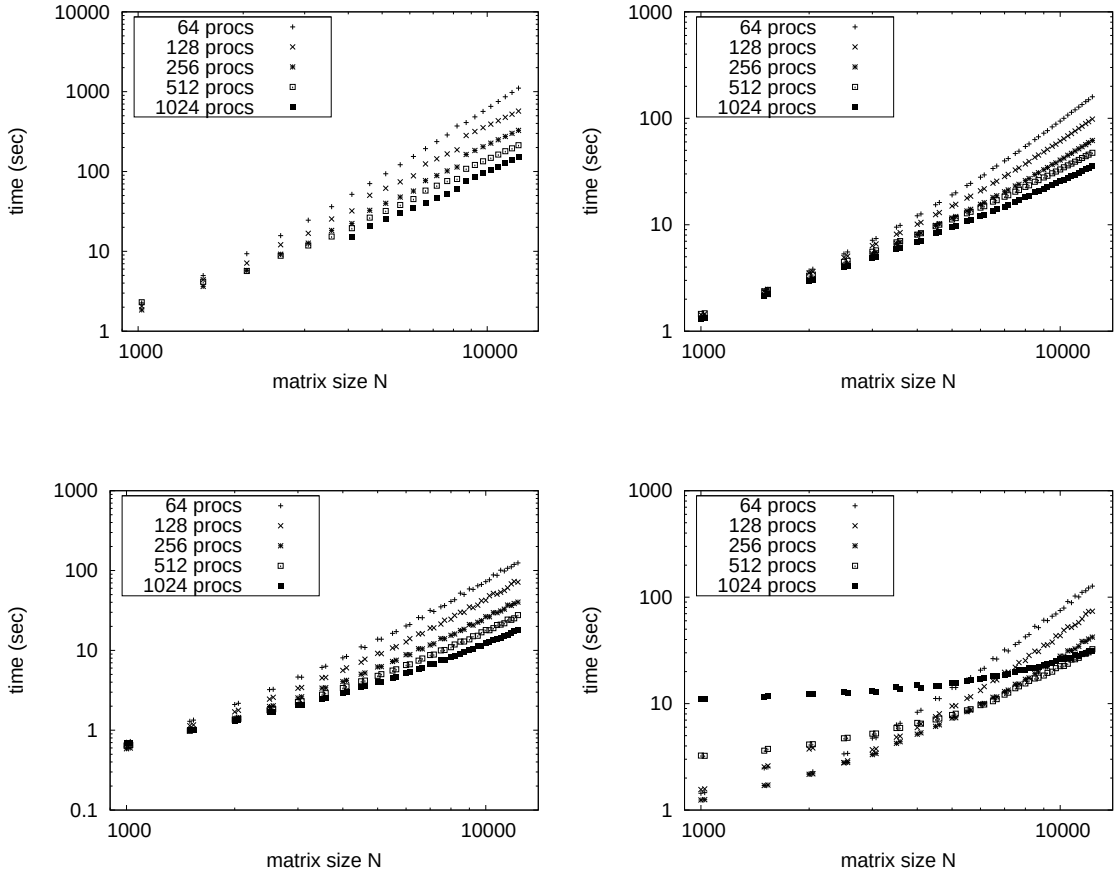


Figure 3: Run times computing all eigenpairs of matrices with eigenvalues randomly generated in [0,1], using PDSYEV (top left), PDSYEVD (top right), PDSYEVX (bottom left) and PDSYEVR (bottom right).

scalability when eigenvalues are clustered, as it did when eigenvalues were randomly spread (cf. Figure 3). Moreover, with clustered eigenvalues, the orthogonality of the computed eigenvectors is as good as with PDSYEV and PDSYEVD, again unlike with randomly generated eigenvalues.

5.2 Ten Percent of the Eigenpairs, 64 - 1024 Processors

The overall results when computing ten percent of the eigenpairs with PDSYEVX and PDSYEVR are the same as the corresponding results when computing all eigenpairs, except of course, the run times are shorter. The discussion of the next results will therefore be kept very brief, in order not to bore the reader with too much repetition.

The run times of computing ten percent of the eigenpairs, with eigenvalues randomly spread in [0,1]

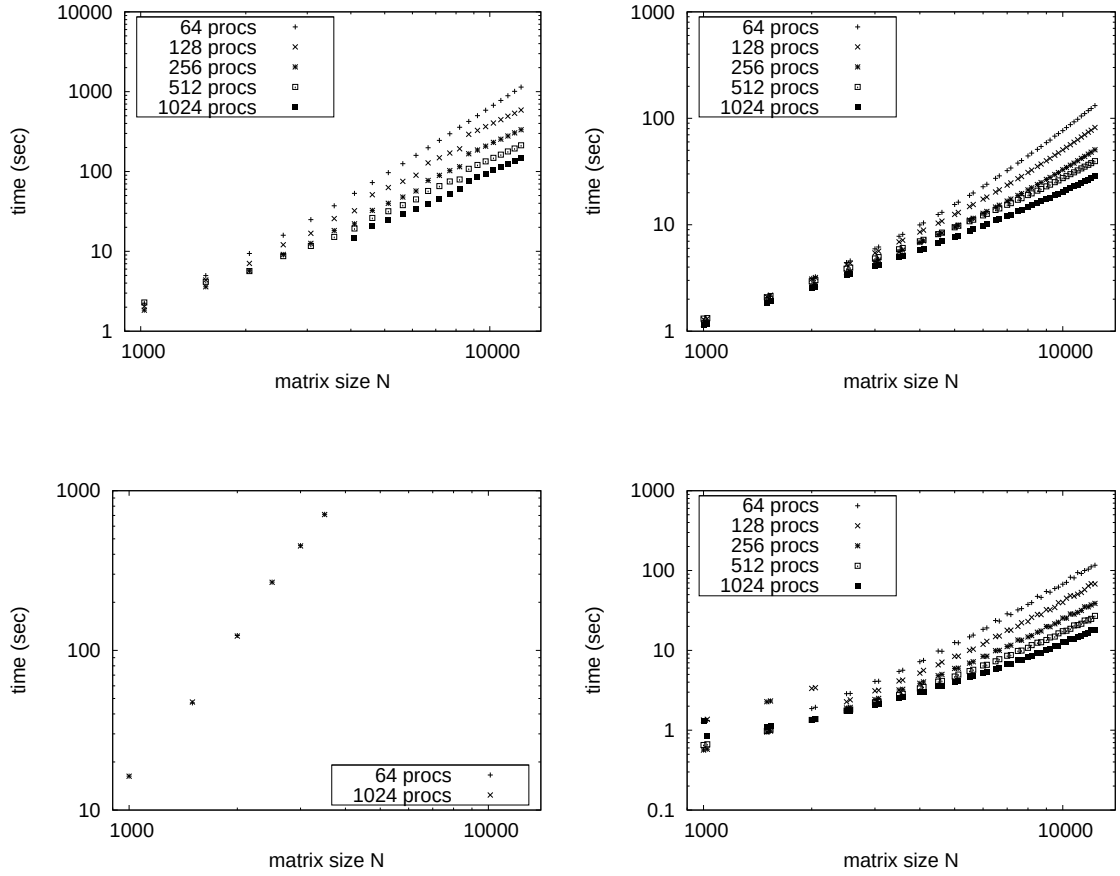


Figure 4: Run times computing all eigenpairs of matrices with eigenvalues clustered around ε, using PDSYEV (top left), PDSYEVD (top right), PDSYEVX (bottom left) and PDSYEVR (bottom right).

are displayed in Figure 5. PDSYEVX (left) has good performance and scaling, whereas PDSYEVR performs well with small numbers of processors, but does not scale to 512 processors or more. Unlike when computing all eigenpairs though, the scaling problem appears not for small matrices but only for larger ones.

Run times for computing ten percent of the eigenpairs when eigenvalues are clustered are shown in Figure 6. Note the difference in y-axis scale between the two plots, showing that PDSYEVX (left) is many times slower than PDSYEVR (right). Again PDSYEVX hardly scales at all, whereas for PDSYEVR with clustered eigenvalues there is no longer a problem with scalability using many processors.

5.3 Chosen Tests with 1024 - 8192 Processors

Some first tests with between 1024 and 8192 processors and matrix sizes up to $n = 122880$ were performed, and the results are presented in this section.

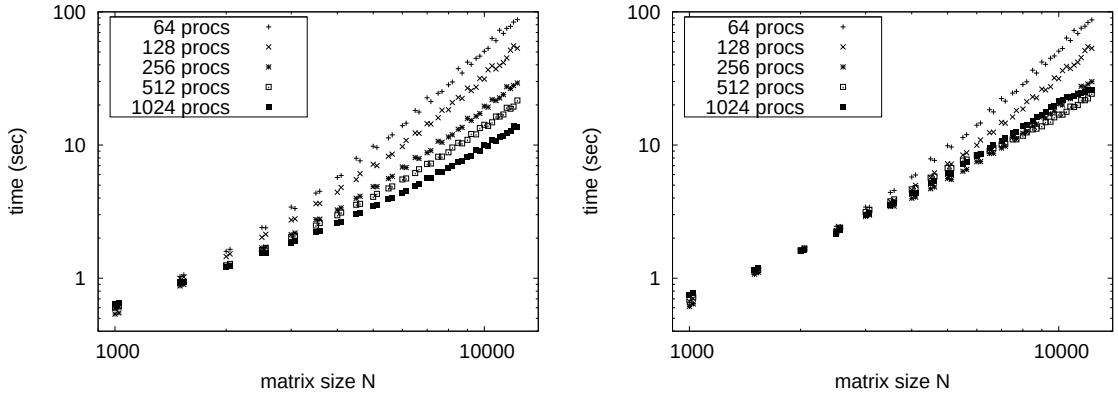


Figure 5: Run times computing ten percent of the eigenpairs of matrices with eigenvalues randomly generated in [0,1], using PDSYEVX (left) and PDSYEVR (right).

Attempts were done to benchmark PDSYEVD on large matrices, but even with 4096 processors (using only one core per node to access the maximum amount of memory per process), the memory did not suffice to perform the test with $n = 122880$. It should be mentioned that the testing program has a considerable memory overhead, for testing purposes. Still this result demonstrates PDSYEVD's need for $\mathcal{O}(n^2)$ extra memory, mentioned in Section 2, and shows that there is a need for an algorithm without this extra memory requirement.

With PDSYEVX, only the case with randomly spread eigenvalues was tested with a larger number of processors; testing the case with clustered eigenvalues would be nonsense considering the corresponding results presented above. To save computing resources only ten percent of the eigenpairs were computed, which also means less memory was needed than if all eigenpairs had been computed. Still, for the memory to suffice, the parameter ORFAC, discussed in Section 2.1, had to be reduced to 10^{-6} for the largest matrix when using 8192 processors, and to 10^{-5} in all other cases. This did however not alter the orthogonality of the computed eigenvectors. The run times can be seen in Figure 7, and it may be concluded that PDSYEVX scales well even with up to 8192 processors, if the matrices are large enough.

With PDSYEVR both clustered and randomly spread eigenvalues were tested, although the latter case was tested only with $n = 122880$. Both tests were performed computing all eigenpairs, so the run times should not be compared to those of PDSYEVX, but should rather be considered as mere test of scalability. The results with clustered eigenvalues are plotted in Figure 7, and we see that the routine scales well with large matrices. This is satisfying, but unfortunately, with $n \geq 81920$ the orthogonality of the computed eigenvectors is completely lost – the value of $\|Z^T Z - I\|/(n\varepsilon)$ is of order 10^{10}.

Also with randomly spread eigenvalues, the orthogonality of the eigenvectors computed by PDSYEVR is completely lost. The run times, shown in Table 1, reveal that also with large processor numbers and large matrices there seems to be a problem with scalability – the time with 8192 processors is longer than with any smaller number of processors.

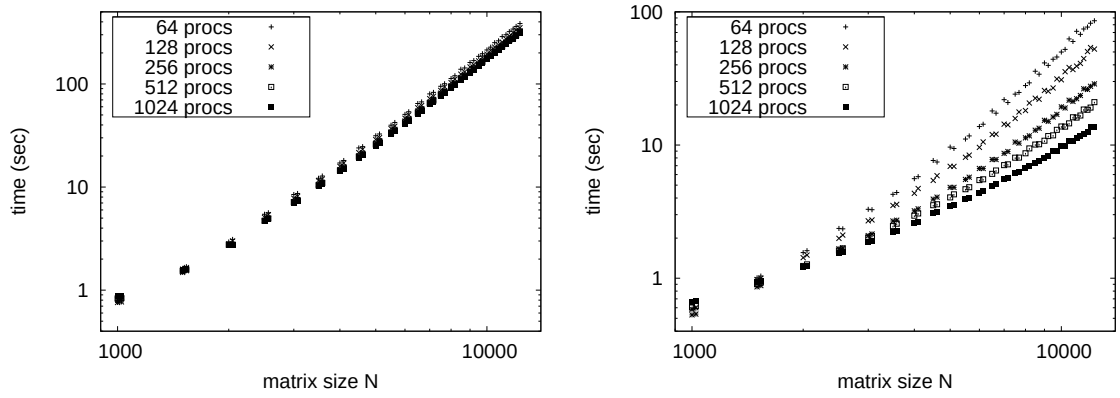


Figure 6: Run times computing ten percent of the eigenpairs of matrices with eigenvalues clustered around ε, using PDSYEVX (left) and PDSYEVR (right).

# processors	1024	2048	4096	8192
time (sec)	7018	4607	3553	7862

Table 1: Run times computing all eigenpairs of matrices of size $n = 122880$, with eigenvalues randomly generated in [0,1], using PDSYEVR.

6 Conclusion and Outlook

In this report, the results of benchmarking four parallel solvers of the dense symmetric eigenproblem have been reviewed. Of special interest were the new routine PDSYEVR, based on the parallel MR^3 algorithm, and the routine PDSYEVX with the ORFAC parameter set to less than the default value of 10^{-3}.

For matrices with randomly spread eigenvalues PDSYEVX with ORFAC $\leq 10^{-4}$ performed well regarding run times as well as orthogonality of the computed eigenvalues. With clustered eigenvalues the performance of PDSYEVX was as expected very poor. PDSYEVR on the other hand performed well on moderate sized matrices with clustered eigenvalues, whereas with randomly spread eigenvalues there seemed to be a problem with scalability. For the largest matrices tested, the orthogonality of eigenvectors computed by PDSYEVR was lost independent of the matrix spectrum, and again a problem with scalability appeared when eigenvalues were randomly spread. With clustered eigenvalues however, good scaling was attained also for the largest matrices and numbers of processors.

It would be interesting to do more benchmarking with a large number of processors. Also, alternative implementations of the MR^3 algorithm are being developed, and should be tested and compared to the implementation benchmarked in this project. It should be noted though, that the most time consuming parts of solving the eigenproblem for a dense matrix are the reduction to tridiagonal form and the back-transformation of eigenvectors. It would thus be of interest to test alternative algorithms for these two phases as well, although presently no such implementations are available. Addition-

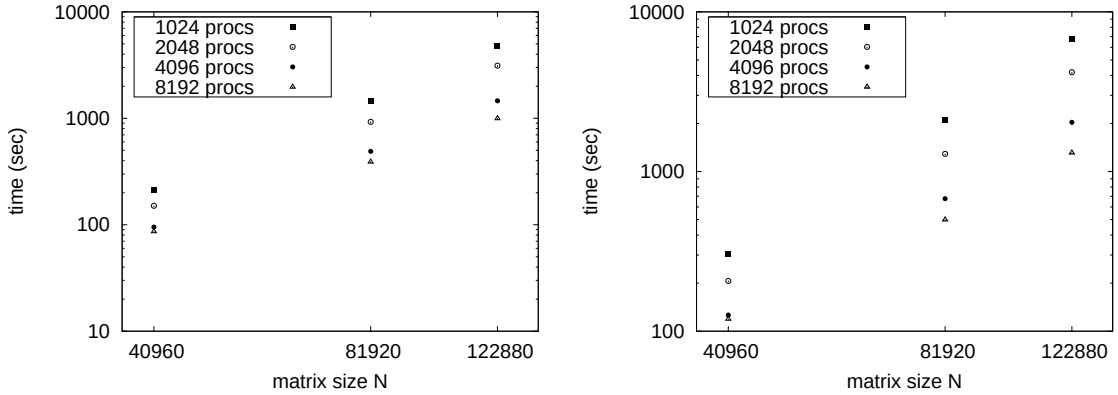


Figure 7: Run times computing ten percent of the eigenpairs of matrices with eigenvalues randomly generated in [0,1], using PDSYEVX (left).
Run times computing all eigenpairs of matrices with eigenvalues clustered around ε, using PDSYEVR (right).

ally, it would be interesting to test other matrix spectra than the two considered in this report, to have a more general view of the way in which performance is influenced by the distribution of eigenvalues.

Acknowledgments

This project has been made within the framework of the Guest Student Programme on Scientific Computing 2010, organized by JSC. I wish to thank my advisor Inge Gutheil for giving me guidance during the project and insight into her current work. The thanks also goes to Robert Speck and Mathias Winkel for organizing the programme and to my fellow students for helping me in many practical aspects of the project and for the nice atmosphere during the programme.

References

1. A. J. Freeman and E. Wimmer. Density Functional Theory as a Major Tool in Computational Materials Science. *Annual Review of Materials Science* 1995;25: 7–36.
2. P. Bientinesi, I. S. Dhillon and R. A. van de Geijn. A Parallel Eigensolver for Dense Symmetric Matrices Based on Multiple Relatively Robust Representations. *SIAM Journal on Scientific Computing* 2005;27(1): 43–66.
3. L. S. Blackford et al. *ScaLAPACK Users' Guide*. Philadelphia: SIAM; 1997. Also available under: Netlib. *ScaLAPACK Users' Guide*. http://netlib.org/scalapack/slug/index.html (accessed 7 October 2010).
4. D. Antonelli and C. Vömel. LAPACK working note 168: PDSYEVR. ScaLAPACK's Parallel MRRR Algorithm for the Symmetric Eigenvalue Problem. Technical Report USB//CSD-05-1399, University of California, Berkeley, 2005.
5. J. W. Demmel. *Applied Numerical Linear Algebra.* Philadelphia: SIAM; 1997.
6. B. N. Parlett and I. S. Dhillon. Relatively Robust Representations of Symmetric Tridiagonals. *Linear Algebra and Appl.* 2000;309(1-3): 121–151.
7. Jülich Supercomputing Center. *IBM BlueGene/P*. http://www.fz-juelich.de/jsc/service/sco_ibmBGP (accessed 7 October 2010).