Overview on HPC software (compilers, libraries, tools)

July 5, 2018 | Slavko Brdar\(^1,3\), Klaus Goergen\(^2,3\), Inge Gutheil\(^1\), Damian Alvarez\(^1\), Michael Knobloch\(^1\)

\(^1\) Jülich Supercomputing Centre, Research Centre Jülich
\(^2\) Institute of Bio- and Geosciences, Agrosphere (IBG-3), Research Centre Jülich
\(^3\) Centre for High-Performance Scientific Computing in Terrestrial Systems (Geoverbund ABC/J)
General purpose libraries

Libraries for geoscience

Debuggers, performance tools
The Module Setup

- Tools are available through “modules”
  - Allows to **easily manage** different versions of programs
  - Works by **dynamic** modification of a user's environment

- Module setup based on **EasyBuild** and **lmod**
  - Staged, hierarchical setup
  - Automatically manages dependencies via toolchains

- Consistent setup on JURECA (cluster & booster) and JEWELS
Figure: Current toolchain tree in JURECA
Most Important Module Commands

module

- spider  # show all products
- spider product  # show product details
- avail  # show all available products
- list  # list loaded products

- load product(s)  # setup access to product
- unload product(s)  # release access
- swap product1 product2  # replace v1 of product with v2

- whatis product(s)  # print short description
- help product(s)  # print longer description
- show product(s)  # show what “settings” are performed
Compiler and MPI libraries

- **Compiler**
  - Intel C/C++ and Fortran compiler
  - GNU C/C++ and Fortran compiler
  - PGI C/C++ and Fortran compiler
  - Clang C/C++ compiler
  - NVIDIA CUDA compiler

- **MPI libraries**
  - Intel MPI
  - Parastation MPI
  - MVAPICH MPI (CUDA aware)
Hierarchical modules

- GCCcore/.5.5.0 is preloaded, which enables a lot of base software
- For HPC software you have to load a compiler, to expand the module tree
  - ml Intel
- Then you load an MPI version
  - ml ParaStationMPI
- Then you can load any other packages
  - ml QuantumESPRESSO/6.2
ml spider *name* shows whether a library is available in the current stage and in which versions

ml spider *name/version* shows which environment you have to load before you can load that version

Many packages are hidden. To see them use ml spider --show-hidden *name*
For R, Python and Perl we use bundles
  - You might be looking for a software package that is part of a bigger module

Use `ml key software`
  - `ml key numpy will suggest SciPy-Stack`
    - You might be looking for a software package that is part of a bigger module

You can use then `ml spider` to find out how to load the module
Modules environment

Stages
- The whole software stack in JURECA and the Booster is updated every 6 months
  - Right when there is an allocation for new projects
- Old stages are still accessible
- To check availability in other stages first type
  `ml use /usr/local/software/jureca[booster]/OtherStages`
Vendor specific Libraries

- MKL Intel® Math Kernel Library
  versions as mentioned in general informations,
  2018.2.199 on JURECA and Booster
Sequential Libraries and Packages (II)

Public domain Libraries

- LAPACK (Linear Algebra PACKage)
- ARPACK (Arnoldi PACKage)
- GSL (Gnu Scientific Library)
- GMP (Gnu Multiple Precision Arithmetic Library)

Commercial library

NAG Fortran Library: JURECA only
Parallel Libraries

MPI Parallelism

- ScaLAPACK (Scalable Linear Algebra PACKage)
- ELPA (Eigenvalue Solvers for Petaflop-Applications)
- Elemental, C++ framework for parallel dense linear algebra
- FFTW (Fastest Fourier Transform of the West)
- MUMPS (MUltifrontal Massively Parallel sparse direct Solver)
  not yet on Booster
- ParMETIS (Parallel Graph Partitioning)
- Hypre (high performance preconditioners)
MPI Parallelism (II)

- PARPACK (Parallel ARPACK), Eigensolver
- SPRNG (Scalable Parallel Random Number Generator)
- SUNDIALS (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers)
- All three not yet on Booster

Parallel Systems, MPI Parallelism

- PETSc, toolkit for partial differential equations
Usage on JURECA

Linking a program name.f calling routines from ScaLAPACK, default version, Intel compiler:

```
mpif77 name.f -lmlk_lscalapack_lp64
-lmlk_lblacs_intelmpi_lp64 -lmlk_lintel_lp64
-lmlk_lintel_thread[-lmlk_sequential]
-lmlk_lcore -liomp5 -lpthread
```
MUMPS
MUltifrontal Massively Parallel sparse direct Solver

- Solution of linear systems with symmetric positive definite matrices, general symmetric matrices, general unsymmetric matrices
- Real or Complex
- Parallel factorization and solve phase, iterative refinement and backward error analysis
- F90 with MPI and OpenMP since 5.1.1
- current version 5.1.2
- http://graal.ens-lyon.fr/MUMPS/
ParMETIS

Parallel Graph Partitioning and Fill-reducing Matrix Ordering
developed in Karypis Lab at the University of Minnesota
4.0.3 on JURECA
http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview

Hypre

High performance preconditioners
Version 2.14.0 on JURECA, also version with bigint,
http://www.llnl.gov/CASC/hypre/software.html
SPRNG

The Scalable Parallel Random Number not yet on Booster Generators Library for ASCI Monte Carlo Computations version 5.0:
various random number generators in one library
Version 1.0 separate library for each random number generator
http://sprng.cs.fsu.edu/

Sundials (CVODE)

Package for the solution of ordinary differential equations, Version 3.1.0
not yet on Booster
https://computation.llnl.gov/casc/sundials/main.html
PETSc

- Portable, Extensible Toolkit for Scientific Computation
- Numerical solution of partial differential equations
- version 3.9.0
- with several other packages included
- complex version and version with 8-Byte integer
- debug versions in Devel Stages only
- http://www.mcs.anl.gov/petsc/
- ml spider petsc
Software for Computational Engineering

- JURECA only
- CFD Package **OpenFOAM**
  - Version 4.1 in Stages/2017b and some older versions in older stages
  - and OpenFOAM-Extend 3.1 and 3.2, only in older stages
- Commercial **FEM Software**
  - **ANSYS**, **LS-DYNA**, **COMSOL** are technically maintained on JURECA
  - **Licenses** must be provided by the **User**!
Further information and JSC-people

http://www.fz-juelich.de/ias/jsc/jureca
http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/_node.html

Mailto
Supercomputer support:
sc@fz-juelich.de
I. Gutheil: Parallel mathematical Libraries
i.gutheil@fz-juelich.de
D. Alvarez: Software Manager
d.alvarez@fz-juelich.de
B. Körfggen: Physics and Engineering software
b.koerfggen@fz-juelich.de
General purpose libraries

Libraries for geoscience

Debuggers, performance tools
NCVIEW
visual browser for netCDF files
simple to use, quick and easy

on Jureca:  ml ncview

NCO
operators for manipulating netCDF or HDF files
  ncatted  attribute editor
  ncks     kitscher sink
  ncrcat   concatentor

on Jureca:  ml NCO
more info: www.ncl.ucar.edu

NCL
interpreted language for scientific data analysis
and visualization
netCDF 3/4, GRIB 1/2, HDF 4/5, ASCII, binary

on Jureca:  ml NCL
more info: www.ncl.ucar.edu
basemap / Python

command line operators for climate and nwp model data.
GRIB 1/2, netCDF 3/4 supported

on Jureca: ml basemap/1.0.7-Python-2.7.14

CDO

command line operators for climate and nwp model data.
GRIB 1/2, netCDF 3/4 supported

on Jureca: ml CDO
more info: www.mpimet.mpg.de/cdo

GRIB API

API from C, Fortran, Python for GRIB 1/2 maintained by ECMWF discontinued. Replaced by ecCodes.

on Jureca: ml grib-api
more info: software.ecmwf.int/wiki/display/GRIB/Home
Compiled environment file

Publicly available

/homea/slts/slts00/local/jureca/env_ini.JURECA.Stage2018a

Once logged in do

source env_ini.JURECA.Stage2018a
General purpose libraries

Libraries for geoscience

Debuggers, performance tools
Outline

Make it work, make it right, make it fast.

Kent Beck

Debugger:
- TotalView / DDT
- MUST
- Intel Inspector

Performance Tools:
- Score-P
- Scalasca
- Vampir
- Intel Vtune Amplifier
- Intel Advisor
- Performance Reports
- TAU
- NVIDIA Visual Profiler
- Darshan
- PAPI

• Local module setup
• Compilers
• Libraries
Debugging Tools (status: May 2018)

- TotalView
- DDT
- MUST
- Intel Inspector
DDT Parallel Debugger

- UNIX Graphical Debugger for C, C++, F77, F90 programs
- Modern, easy-to-use debugger
- Special, non-traditional features
  - Multi-process and multi-threaded
  - 1D + 2D array data visualization
  - Support for MPI parallel debugging (automatic attach, message queues)
  - Support for OpenMP (Version 2.x and later)
  - Support for CUDA and OpenACC
  - Job submission from within debugger

- http://www.allinea.com
- **NOTE:** License limited to 64 processes (shared between all users)
DDT: Main Window

- Process controls
- Process groups
- Source code
- Variables
- Expression evaluator
- Stack trace
Typical Performance Analysis Procedure

- Do I have a performance problem at all?
  - Time / speedup / scalability measurements
- What is the key bottleneck (computation / communication)?
  - MPI / OpenMP / flat profiling
- Where is the key bottleneck?
  - Call-path profiling, detailed basic block profiling
- Why is it there?
  - Hardware counter analysis
  - Trace selected parts (to keep trace size manageable)
- Does the code have scalability problems?
  - Load imbalance analysis, compare profiles at various sizes function-by-function, performance modeling
Performance Tools (status: May 2018)

- Score-P
- Scalasca 2
- Vampir[Server]
- HPCToolkit
- Allinea Performance Reports
- Darshan
- NVIDIA Visual Profiler
- TAU
- Intel VTune Amplifier XE
- Intel Advisor
- mpiP*
- Extrae/Paraver*
- PAPI*
Score-P

- Community instrumentation and measurement infrastructure
  - Developed by a consortium of performance tool groups
    - Next generation measurement system of
      - Scalasca 2.x
      - Vampir
      - TAU
      - Periscope
    - Common data formats improve tool interoperability
    - http://www.score-p.org
Collection of trace-based performance analysis tools

- Specifically designed for large-scale systems

Unique features:

- Scalable, automated search for event patterns representing inefficient behavior
- Scalable identification of the critical execution path
- Delay / root-cause analysis

Based on Score-P for instrumentation and measurement

- Includes convenience / post-processing commands providing added value

http://www.scalasca.org
Flat MPI Profile: Example

% module load Toolchain Score-P Scalasca
% mpif90 -O3 -c foo.f90
% mpif90 -O3 -c bar.f90
% scorep --nocompiler \  
    mpif90 -O3 -o myprog foo.o bar.o

##########################
##  In the job script:  ##
##########################

module load Toolchain Scalasca
scalasca -analyze \  
  srun --tasks-per-node P --ntasks n [...] --exe ./myprog

##########################
## After job finished:  ##
##########################

% scalasca -examine scorep_myprog_Ppnt_xt_sum
Flat MPI Profile: Example (cont.)

- **Aggregate execution time on master threads**
- **Time spent in a particular MPI call**
- **Time spent in selected call as percentage of total time**
Call-path Profile: Example

```bash
% module load Toolchain Score-P Scalasca
% scorep mpif90 -O3 -c foo.f90
% scorep mpif90 -O3 -c bar.f90
% scorep \
  mpif90 -O3 -o myprog foo.o bar.o

#############################
##  In the job script:  ##
#############################

module load Toolchain Scalasca
scalasca -analyze \
  srun --tasks-per-node $P --ntasks $n [...] --exe ./myprog
```
Call-path Profile: Example (cont.)

- Estimates trace buffer requirements
- Allows to identify candidate functions for filtering
  - Computational routines with high visit count and low time-per-visit ratio
- Region/call-path classification
  - MPI (pure MPI library functions)
  - OMP (pure OpenMP functions/regions)
  - USR (user-level source local computation)
  - COM ("combined" USR + OpeMP/MPI)
  - ANY/ALL (aggregate of all region types)

% scalasca -examine -s epik_myprog_Ppnxt_sum
scorep-score -r ./epik_myprog_Ppnxt_sum/profile.cubex
INFO: Score report written to ./scorep_myprog_Ppnxt_sum/scorep.score
% less scorep_myprog_Ppmxt_sum/scorep.score

Estimated aggregate size of event trace: 162GB
Estimated requirements for largest trace buffer (max_buf): 2758MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY): 2822MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=2822MB to avoid intermediate flushes or reduce requirements using USR regions filters.)

<table>
<thead>
<tr>
<th>flt type</th>
<th>max_buf[B]</th>
<th>visits</th>
<th>time[s]</th>
<th>time[%]</th>
<th>time/visit[us]</th>
<th>region</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>2,891,417,902</td>
<td>6,662,521,083</td>
<td>36581.51</td>
<td>100.0</td>
<td>5.49</td>
<td>ALL</td>
</tr>
<tr>
<td>USR</td>
<td>2,858,189,854</td>
<td>6,574,882,113</td>
<td>13618.14</td>
<td>37.2</td>
<td>2.07</td>
<td>USR</td>
</tr>
<tr>
<td>OMP</td>
<td>54,327,600</td>
<td>86,353,920</td>
<td>22719.78</td>
<td>62.1</td>
<td>263.10</td>
<td>OMP</td>
</tr>
<tr>
<td>MPI</td>
<td>676,342</td>
<td>550,010</td>
<td>208.98</td>
<td>0.6</td>
<td>379.96</td>
<td>MPI</td>
</tr>
<tr>
<td>COM</td>
<td>371,930</td>
<td>735,040</td>
<td>34.61</td>
<td>0.1</td>
<td>47.09</td>
<td>COM</td>
</tr>
</tbody>
</table>

USR 921,918,660 2,110,313,472 3290.11 9.0 1.56 matmul_sub
USR 921,918,660 2,110,313,472 5914.98 16.2 2.80 binvcrhs
USR 921,918,660 2,110,313,472 3822.64 10.4 1.81 matvec_sub
USR 41,071,134 87,475,200 358.56 1.0 4.10 lhsinit
USR 41,071,134 87,475,200 145.42 0.4 1.66 binvrhs
USR 29,194,256 68,892,672 86.15 0.2 1.25 exact_solution
OMP 3,280,320 3,293,184 15.81 0.0 4.80 !$omp parallel

[...]
Call-path Profile: Filtering

- In this example, the 6 most frequently called routines are of type USR
- These routines contribute around 35% of total time
  - However, much of that is most likely measurement overhead
    - Frequently executed
    - Time-per-visit ratio in the order of a few microseconds

- Avoid measurements to reduce the overhead
- List routines to be filtered in simple text file
Filtering: Example

% cat filter.txt
SCOREP_REGION_NAMES_BEGIN
  EXCLUDE
    binvcrhs
    matmul_sub
    matvec_sub
    binvrhs
    lhsinit
    exact_solution
SCOREP_REGION_NAMES_END

- Score-P filtering files support
  - Wildcards (shell globs)
  - Blacklisting
  - Whitelisting
  - Filtering based on filenames
Allinea Performance Reports

- **Single page** report provides quick overview of performance issues
- Works on unmodified, optimized executables
- Shows CPU, memory, network and I/O utilization
- Supports MPI, multi-threading and accelerators
- Saves data in HTML, CVS or text form


**Note:** License limited to 512 processes (with unlimited number of threads)
Example Performance Reports

Summary: cp2k.popt is CPU-bound in this configuration

The total wallclock time was spent as follows:

- **CPU**: 56.5% 
  - Time spent running application code. High values are usually good.
  - This is **average**, check the CPU performance section for optimization advice.

- **MPI**: 43.5% 
  - Time spent in MPI calls. High values are usually bad.
  - This is **average**, check the MPI breakdown for advice on reducing it.

- **I/O**: 0.0% 
  - Time spent in filesystem I/O. High values are usually bad.
  - This is **negligible**, there's no need to investigate I/O performance.

This application run was **CPU-bound**. A breakdown of this time and advice for investigating further is in the CPU section below.

### CPU

A breakdown of how the 56.5% total CPU time was spent:

- **Scalar numeric ops**: 27.7% 
  - The per-core performance is **memory-bound**. Use a profiler to identify time-consuming loops and check their cache performance.
  - Little time is spent in **vectorized instructions**. Check the compiler's vectorization advice to see why key loops could not be vectorized.

- **Vector numeric ops**: 11.3% 

- **Memory accesses**: 60.9%

- **Other**: 0.0%

### MPI

Of the 43.5% total time spent in MPI calls:

- **Time in collective calls**: 8.2%
- **Time in point-to-point calls**: 91.8%

<table>
<thead>
<tr>
<th>Estimated collective rate</th>
<th>169 Mb/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated point-to-point rate</td>
<td>50.6 Mb/s</td>
</tr>
</tbody>
</table>

The point-to-point transfer rate is low. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait. Use an MPI profiler to identify the problematic calls and ranks.

### I/O

A breakdown of how the 0.0% total I/O time was spent:

- **Time in reads**: 0.0%
- **Time in writes**: 0.0%
- **Estimated read rate**: 0 bytes/s
- **Estimated write rate**: 0 bytes/s

No time is spent in I/O operations. There's nothing to optimize here!

### Memory

Per-process memory usage may also affect scaling:

<table>
<thead>
<tr>
<th>Mean process memory usage</th>
<th>82.5 Mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak process memory usage</td>
<td>89.3 Mb</td>
</tr>
<tr>
<td>Peak node memory usage</td>
<td>7.4%</td>
</tr>
</tbody>
</table>

The peak node memory usage is low. You may be able to reduce the total number of CPU hours used by running with fewer MPI processes and more data on each process.
% module load AllineaPerformanceReports

###################################
## In the job script:  ##
###################################

perf-report --mpi="slurm"
    srun --procs-per-node=P --nodes=n [...] ./myprog [args]

###################################
## After job finished:  ##
###################################

% less myprog_<NP>p_<DATE>.txt
% firefox myprog_<NP>p_<DATE>.html
Darshan

- I/O characterization tool logging parallel application file access
- Summary report provides quick overview of performance issues
- Works on unmodified, optimized executables
- Shows counts of file access operations, times for key operations, histograms of accesses, etc.

- Supports POSIX, MPI-IO, HDF5, PnetCDF, …
  - Doesn’t support mpif90 on BlueGene systems (use mpif77)
  - Binary log file written at exit post-processed into PDF report

- Open Source: installed on many HPC systems
Example Darshan report extract

| jobid: | uid: | nprocs: 4096 | runtime: 175 seconds |

![Average I/O cost per process chart](chart1)

![I/O Operation Counts chart](chart2)

![I/O Sizes chart](chart3)

![I/O Pattern chart](chart4)
% load compiler and MPI module
% module load darshan-runtime darshan-util

##########################################################################
## In the job script: ##
##########################################################################
export LD_PRELOAD=$EBROOTDARSHANMINRUNTIME/lib/libdarshan.so
export DARSHAN_LOG_PATH=$PWD
export DARSHAN_LOGFILE=darshan.log
srun --tasks-per-node P --ntasks n [...] ./myprog [args]

##########################################################################
## After job finished: ##
##########################################################################
% darshan-job-summary.pl darshan.log
% gv darshan.pdf
Support

- For general support: sc@fz-juelich.de
- Tool-specific support via corresponding mailing lists
  - Score-P: support@score-p.org
  - Scalasca: scalasca@fz-juelich.de

- Workshops and Trainings:
  - Regular VI-HPS Tuning Workshops
    - Several days
    - Multiple tools, e.g. Score-P, Scalasca, Vampir, TAU, ...
    - Bring-your-own-code
    - http://www.vi-hps.org/training/tws/
  - JSC Porting and Tuning Workshop Series
  - WS’s and trainings at several HPC conferences