Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

Efficient parallel I/O

Developing a module for the particle-in-cell code PSC

September 27, 2010 | Axel Hübl

Outline

Motivation and issues with the present configuration

- Part 1: The particle-in-cell code `PSC`

Outline

Motivation and issues with the present configuration

- Part 1: The particle-in-cell code `PSC`
- Part 2: Output in massiv parallel environments

Outline

Motivation and issues with the present configuration

- Part 1: The particle-in-cell code `PSC`
- Part 2: Output in massiv parallel environments
- Part 3: New implementation

Outline

Motivation and issues with the present configuration

- Part 1: The particle-in-cell code `PSC`
- Part 2: Output in massiv parallel environments
- Part 3: New implementation
- Part 4: Post-Processing

Outline

Motivation and issues with the present configuration

- Part 1: The particle-in-cell code `PSC`
- Part 2: Output in massiv parallel environments
- Part 3: New implementation
- Part 4: Post-Processing
- Part 5: Results

Outline

Motivation and issues with the present configuration

- Part 1: The particle-in-cell code `PSC`
- Part 2: Output in massiv parallel environments
- Part 3: New implementation
- Part 4: Post-Processing
- Part 5: Results
- Part 6: Conclusion

Mitglied der Helmholtz-Gemeinschaft

JÜLICH FORSCHUNGSZENTRUM

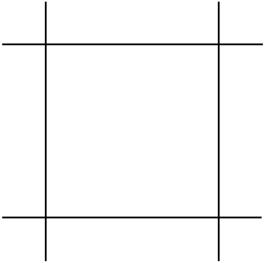
Efficient parallel I/O

Part I: The particle-in-cell code PSC

September 27, 2010 | Axel Hübl

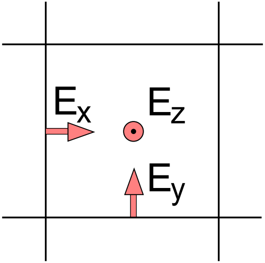
Particle-in-cell codes in general

A short introduction



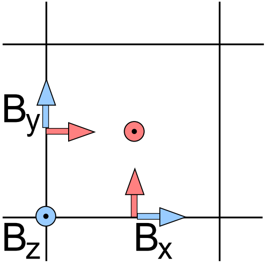
Particle-in-cell codes in general

A short introduction



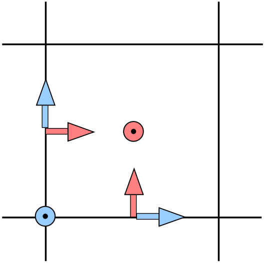
Particle-in-cell codes in general

A short introduction



Particle-in-cell codes in general

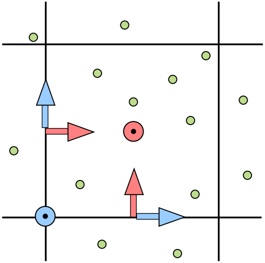
A short introduction



Particle-in-cell codes in general

A short introduction

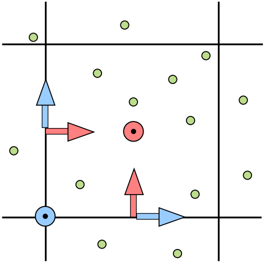
- Particles are smeared on the mesh grids randomly



Particle-in-cell codes in general

A short introduction

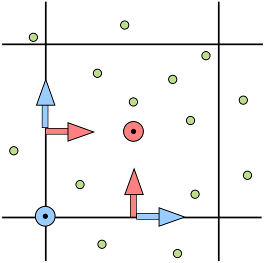
- Particles are smeared on the mesh grids randomly
- Using grid based field and particle solving methods



Particle-in-cell codes in general

A short introduction

- Particles are smeared on the mesh grids randomly
- Using grid based field and particle solving methods

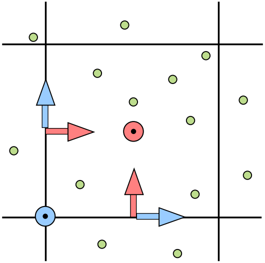


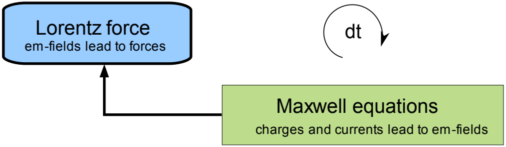
Maxwell equations
charges and currents lead to em-fields

Particle-in-cell codes in general

A short introduction

- Particles are smeared on the mesh grids randomly
- Using grid based field and particle solving methods





Lorentz force

em-fields lead to forces

dt

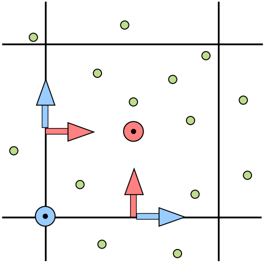
Maxwell equations

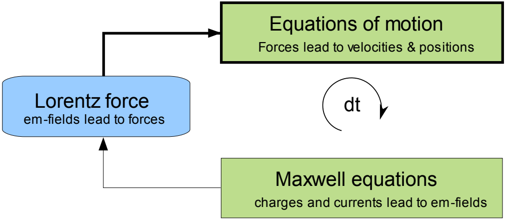
charges and currents lead to em-fields

Particle-in-cell codes in general

A short introduction

- Particles are smeared on the mesh grids randomly
- Using grid based field and particle solving methods





Equations of motion
Forces lead to velocities & positions

dt

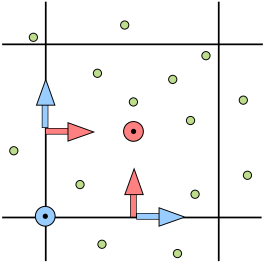
Lorentz force
em-fields lead to forces

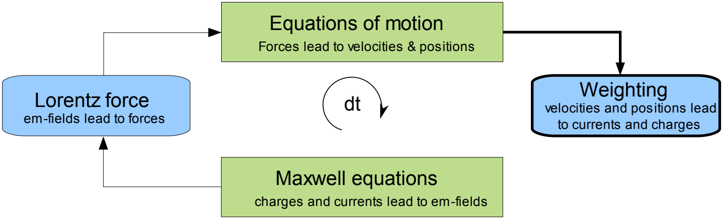
Maxwell equations
charges and currents lead to em-fields

Particle-in-cell codes in general

A short introduction

- Particles are smeared on the mesh grids randomly
- Using grid based field and particle solving methods





Equations of motion
Forces lead to velocities & positions

Weighting
velocities and positions lead to currents and charges

dt

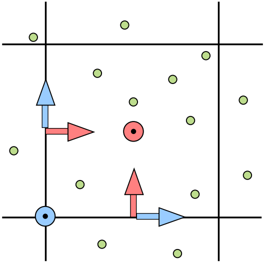
Lorentz force
em-fields lead to forces

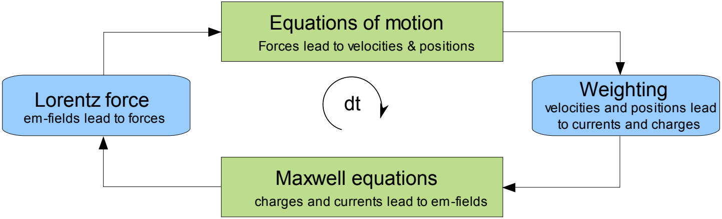
Maxwell equations
charges and currents lead to em-fields

Particle-in-cell codes in general

A short introduction

- Particles are smeared on the mesh grids randomly
- Using grid based field and particle solving methods





Equations of motion

Forces lead to velocities & positions

Weighting

velocities and positions lead to currents and charges

Maxwell equations

charges and currents lead to em-fields

Lorentz force

em-fields lead to forces

The particle-in-cell code PSC

A mesh-based plasma code

- Major SimLab Plasma physics (SLPP) code for laser-plasma interaction simulations

The particle-in-cell code `PSC`

A mesh-based plasma code

- Major SimLab Plasma physics (SLPP) code for laser-plasma interaction simulations
- Originally created by Hartmut Ruhl, LMU, Munich

The particle-in-cell code `PSC`

A mesh-based plasma code

- Major SimLab Plasma physics (SLPP) code for laser-plasma interaction simulations
- Originally created by Hartmut Ruhl, LMU, Munich
- Full 3D cartesian mesh based, relativistic em-field solver and particle pusher

The particle-in-cell code PSC

A mesh-based plasma code

- Major SimLab Plasma physics (SLPP) code for laser-plasma interaction simulations
- Originally created by Hartmut Ruhl, LMU, Munich
- Full 3D cartesian mesh based, relativistic em-field solver and particle pusher
- Present optimal usage: up to 1k tasks, with around 15 Million particles (kernel)

The particle-in-cell code `PSC`

Issues with the present I/O

Old I/O

- Puts down the production runs to a great extent

The particle-in-cell code PSC

Issues with the present I/O

Old I/O

- Puts down the production runs to a great extent
- One file per task and timestep

The particle-in-cell code `PSC`

Issues with the present I/O

Old I/O

- Puts down the production runs to a great extent
- One file per task and timestep
- Slows down file system

The particle-in-cell code `PSC`

Issues with the present I/O

Old I/O

- Puts down the production runs to a great extent
- One file per task and timestep
- Slows down file system
- Files can not be handled any more

The particle-in-cell code `PSC`

Issues with the present I/O

Old I/O

- Puts down the production runs to a great extent
- One file per task and timestep
- Slows down file system
- Files can not be handled any more
- Huge post-processing time

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

Efficient parallel I/O

Part II: Output in massiv parallel environments

September 27, 2010 | Axel Hübl

Naive implementations

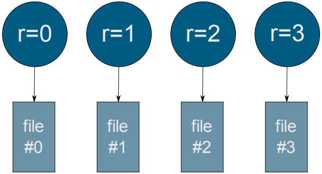
In I/O is ***parallel*.NE.*parallel* !**

- Distributed *global* data set

Naive implementations

In I/O is *parallel*.NE.*parallel* !

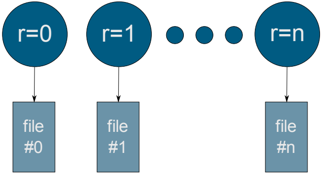
- Distributed *global* data set
- One file per task



Naive implementations

In I/O is *parallel*.NE.*parallel* !

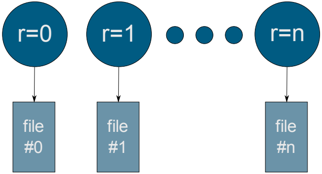
- Distributed *global* data set
- One file per task



Naive implementations

In I/O is ***parallel*.NE.*parallel* !**

- Distributed *global* data set
- One file per task



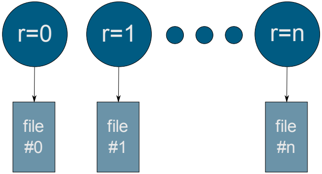
Problems:

- inode limits and post-processing

Naive implementations

In I/O is *parallel*.NE.*parallel* !

- Distributed *global* data set
- One file per task



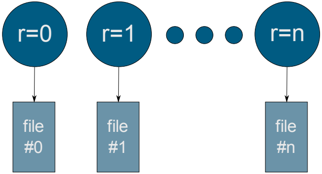
Problems:

- inode limits and post-processing
- file create and delete

Naive implementations

In I/O is *parallel*.NE.*parallel* !

- Distributed *global* data set
- One file per task



Problems:

- inode limits and post-processing
- file create and delete
- small files and file system block size

Naive implementations

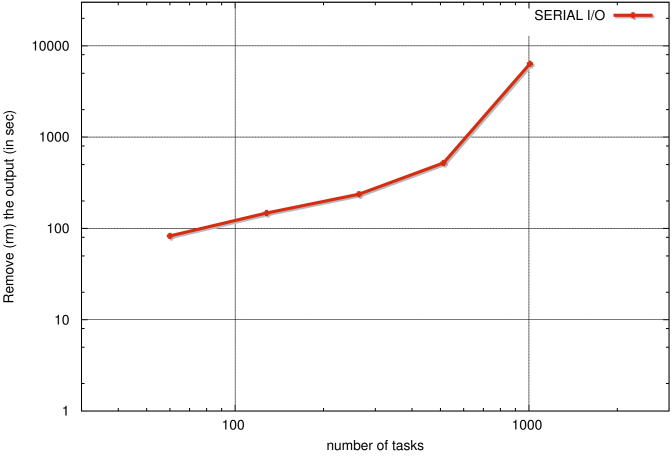
How to handle the output?

- Remove the files of 41 Mio particles

Naive implementations

How to handle the output?

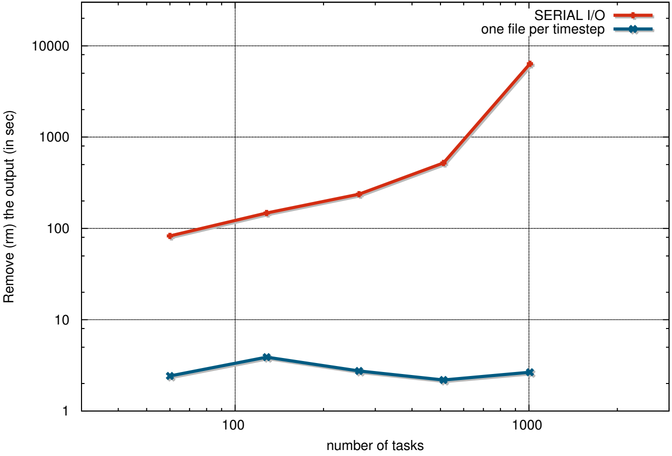
- Remove the files of 41 Mio particles



Naive implementations

How to handle the output?

- Remove the files of 41 Mio particles



SIONlib

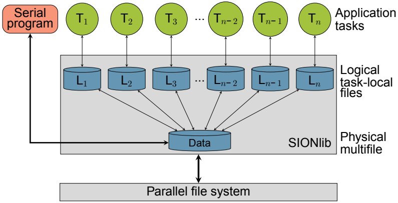
Scalable parallel I/O for task-local files

- Inhouse developed by W. Frings

SIONlib

Scalable parallel I/O for task-local files

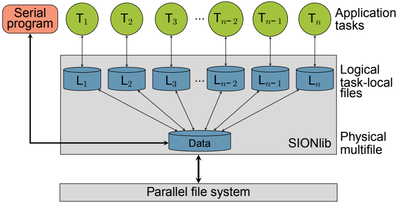
- Inhouse developed by W. Frings



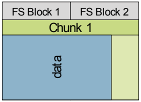
SIONlib

Scalable parallel I/O for task-local files

- Inhouse developed by W. Frings



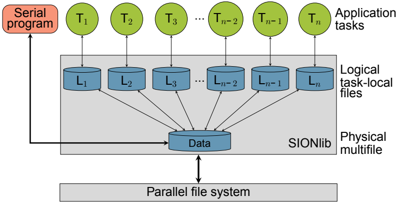
- Block alignment:



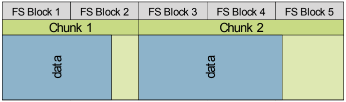
SIONlib

Scalable parallel I/O for task-local files

- Inhouse developed by W. Frings



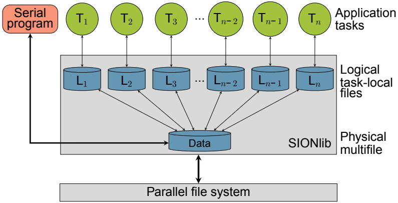
- Block alignment:



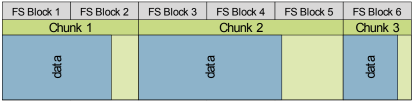
SIONlib

Scalable parallel I/O for task-local files

- Inhouse developed by W. Frings



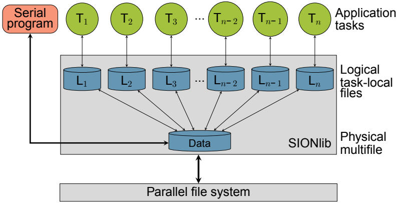
- Block alignment:



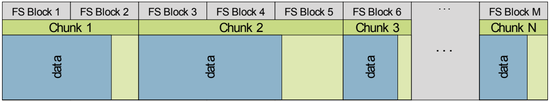
SIONlib

Scalable parallel I/O for task-local files

- Inhouse developed by W. Frings



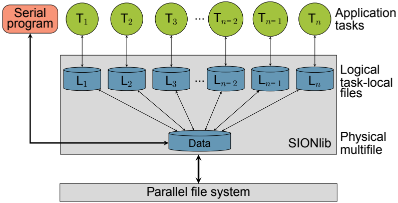
- Block alignment:



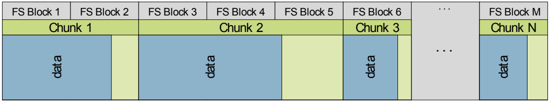
SIONlib

Scalable parallel I/O for task-local files

- Inhouse developed by W. Frings



- Block alignment:

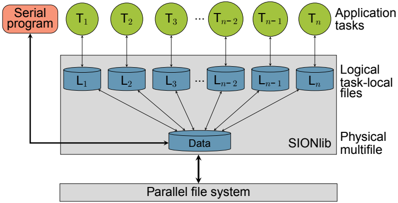


- Support: C, C++, FORTRAN with MPI and OpenMP

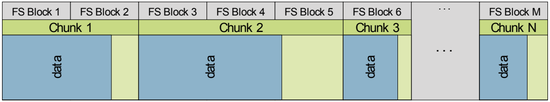
SIONlib

Scalable parallel I/O for task-local files

- Inhouse developed by W. Frings



- Block alignment:



- Support: C, C++, FORTRAN with MPI and OpenMP

W. Frings et al.:
Scalable Massively Parallel
I/O to Task-Local Files

www.fz-juelich.de/jsc/sionlib/

MPI-I/O

MPI File extension

- API 1997, part of the MPI standard since 2.0

MPI-I/O

MPI File extension

- API 1997, part of the MPI standard since 2.0
- On-the-fly converts possible

MPI-I/O

MPI File extension

- API 1997, part of the MPI standard since 2.0
- On-the-fly converts possible
- Very similar to the message-passing part of MPI

MPI-I/O

MPI File extension

- API 1997, part of the MPI standard since 2.0
- On-the-fly converts possible
- Very similar to the message-passing part of MPI
- Presently not useable with LUSTRE file system on Juropa

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

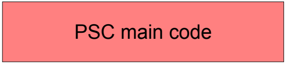
Efficient parallel I/O

Part III: New implementation

September 27, 2010 | Axel Hübl

New implementation

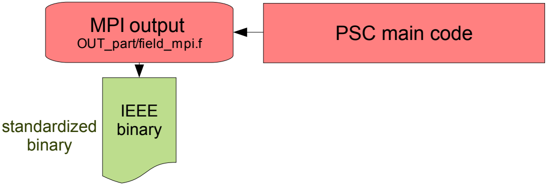
Overview



PSC main code

New implementation

Overview



MPI output
OUT_part/field_mpi.f

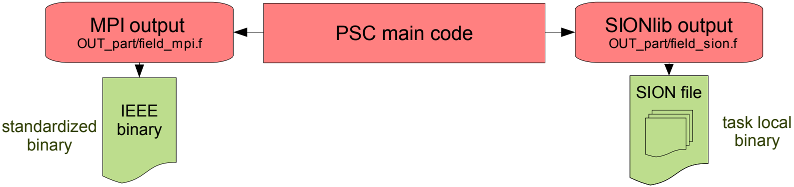
PSC main code

IEEE binary

standardized binary

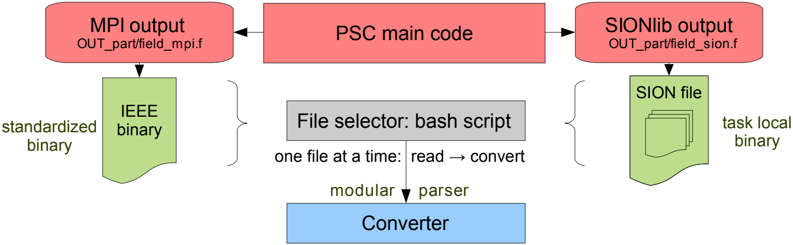
New implementation

Overview



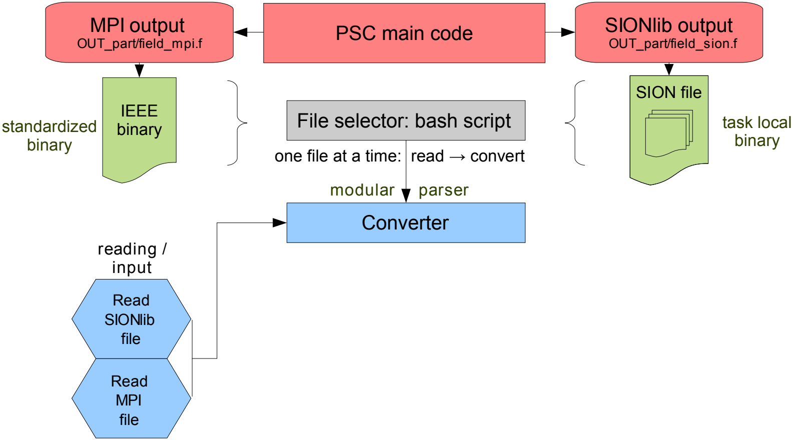
New implementation

Overview



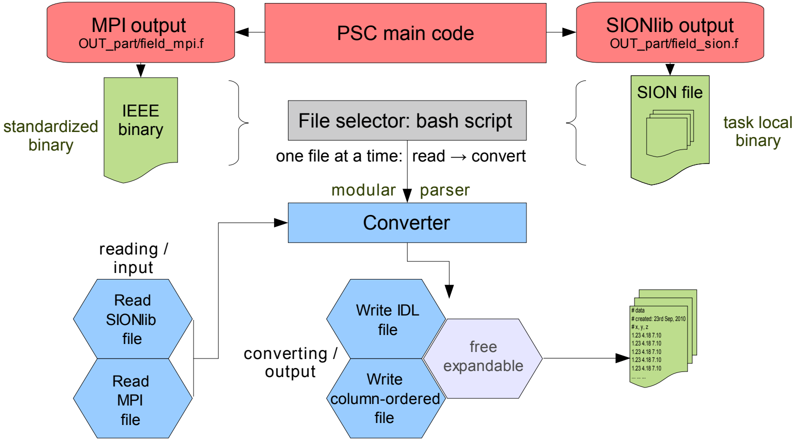
New implementation

Overview



New implementation

Overview



MPI output
OUT_part/field_mpi.f

PSC main code

SIONlib output
OUT_part/field_sion.f

standardized binary

IEEE binary

File selector: bash script

one file at a time: read → convert

SION file

task local binary

modular parser

Converter

reading / input

Read SIONlib file

Read MPI file

converting / output

Write IDL file

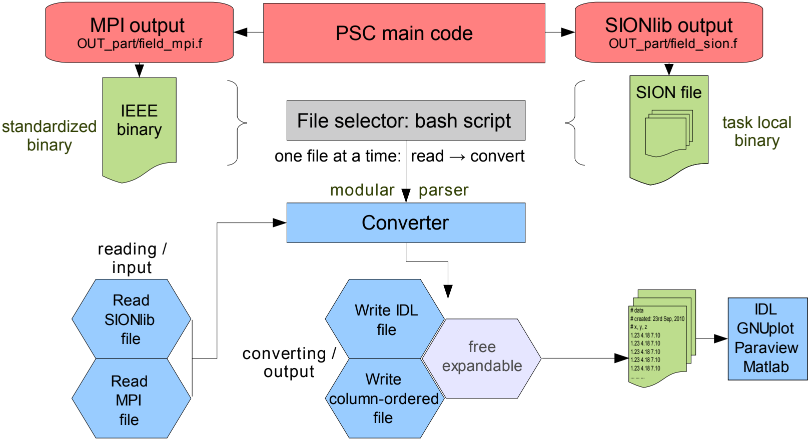
Write column-ordered file

free expandable

```
# data
# created: 23rd Sep, 2010
# x, y, z
1.23 4.18 7.10
1.23 4.18 7.10
1.23 4.18 7.10
1.23 4.18 7.10
1.23 4.18 7.10
... ... ...
```

New implementation

Overview



New implementation

Example of used output commands

SIONlib vs. MPI-I/O

```
call
fsion_paropen_mpi(...)
```

```
call MPI_FILE_OPEN(...)
call MPI_FILE_SET_VIEW(...)
```

New implementation

Example of used output commands

SIONlib vs. MPI-I/O

```
call
fsion_paropen_mpi(...)

call fsion_write(
       particles(1),
       real8size,
       numpart,
       fh, bwrote)
```

```
call MPI_FILE_OPEN(...)
call MPI_FILE_SET_VIEW(...)
```

New implementation

Example of used output commands

SIONlib vs. MPI-I/O

```
call
fsion_paropen_mpi(...)

call fsion_write(
       particles(1),
       real8size,
       numpart,
       fh, bwrote)
```

```
call MPI_FILE_OPEN(...)
call MPI_FILE_SET_VIEW(...)

call MPI_FILE_WRITE_ORDERED(
       fh, particles(1),
       numpart,
       MPI_REAL8,
       status, err)
```

New implementation

Example of used output commands

SIONlib vs. MPI-I/O

```
call
fsion_paropen_mpi(...)

call fsion_write(
       particles(1),
       real8size,
       numpart,
       fh, bwrote)

call
fsion_parclose_mpi(...)
```

```
call MPI_FILE_OPEN(...)
call MPI_FILE_SET_VIEW(...)

call MPI_FILE_WRITE_ORDERED(
       fh, particles(1),
       numpart,
       MPI_REAL8,
       status, err)

call MPI_FILE_CLOSE(...)
```

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

Efficient parallel I/O

Part IV: Post-Processing

September 27, 2010 | Axel Hübl

Post-processing

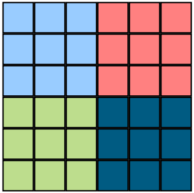
What to do with distributed output?

- Domain decomposition for a field:

Post-processing

What to do with distributed output?

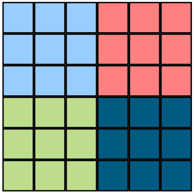
- Domain decomposition for a field:

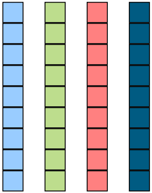


Post-processing

What to do with distributed output?

- Domain decomposition for a field: RAM (Fortran order)

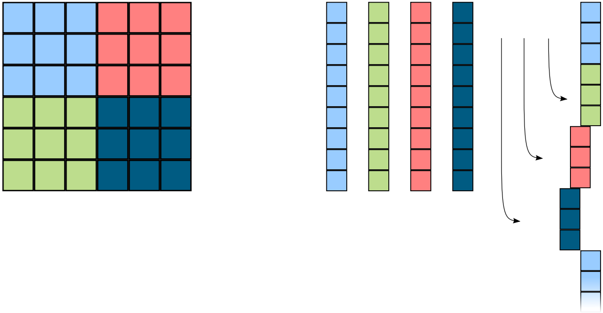




Post-processing

What to do with distributed output?

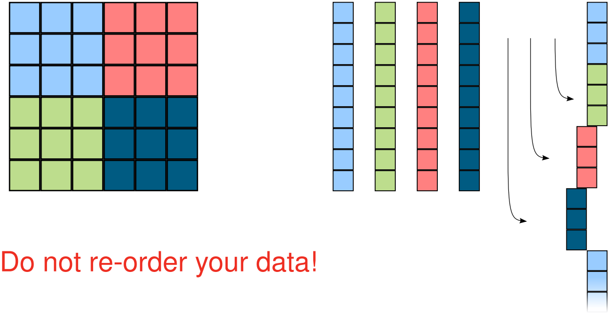
- Domain decomposition for a field: RAM (Fortran order)



Post-processing

What to do with distributed output?

- Domain decomposition for a field: RAM (Fortran order)

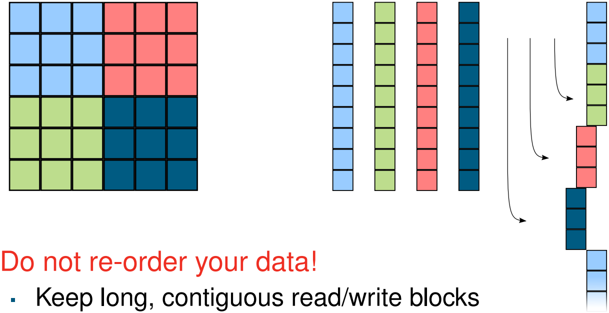


- Do not re-order your data!

Post-processing

What to do with distributed output?

- Domain decomposition for a field: RAM (Fortran order)

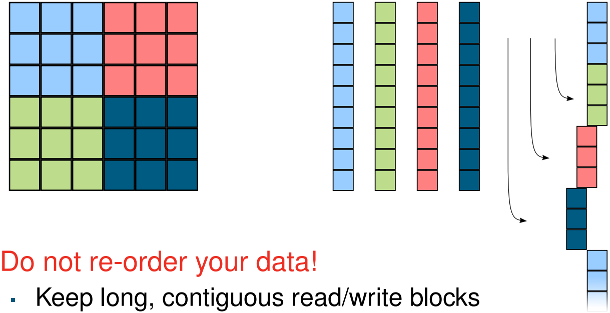


- Do not re-order your data!
  - Keep long, contiguous read/write blocks

Post-processing

What to do with distributed output?

- Domain decomposition for a field: RAM (Fortran order)

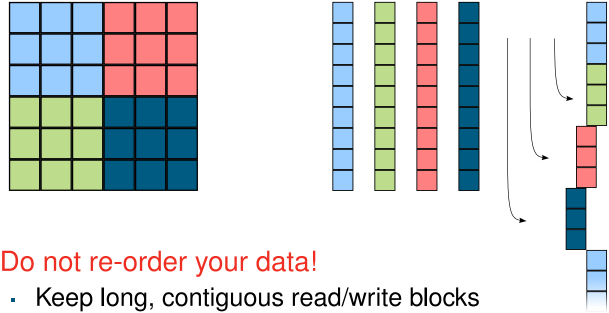


- Do not re-order your data!
  - Keep long, contiguous read/write blocks
  - Result: no temp data, less seeking, no double reading

Post-processing

What to do with distributed output?

- Domain decomposition for a field: RAM (Fortran order)



- Do not re-order your data!
  - Keep long, contiguous read/write blocks
  - Result: no temp data, less seeking, no double reading
- Choose a visualization software, that understands decomposition schemes.

Mitglied der Helmholtz-Gemeinschaft

Efficient parallel I/O

Part V: Results

September 27, 2010 | Axel Hübl

Scaling

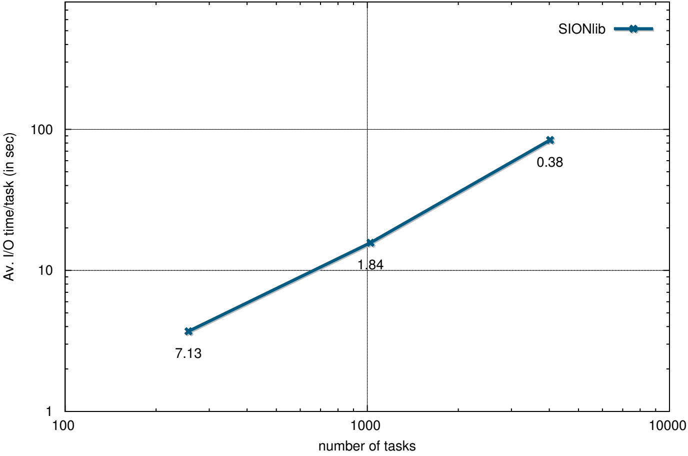
SIONlib vs MPI-I/O

- Jugene, 19 Mio particles: Data per tasks drastic below 2 MB

Scaling

SIONlib vs MPI-I/O

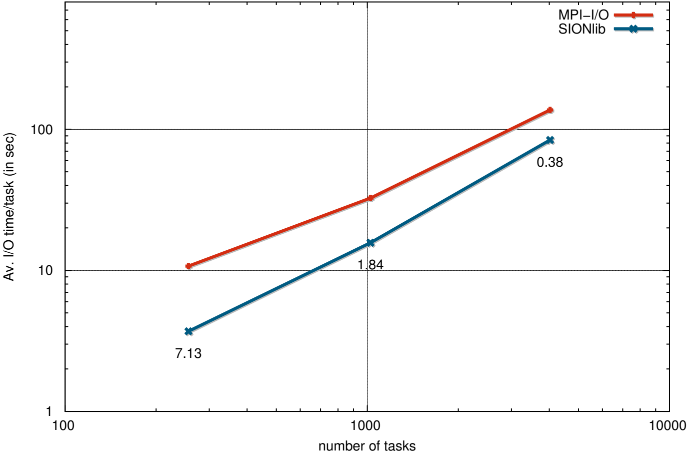
- Jugene, 19 Mio particles: Data per tasks drastic below 2 MB



Scaling

SIONlib vs MPI-I/O

- Jugene, 19 Mio particles: Data per tasks drastic below 2 MB



Scaling

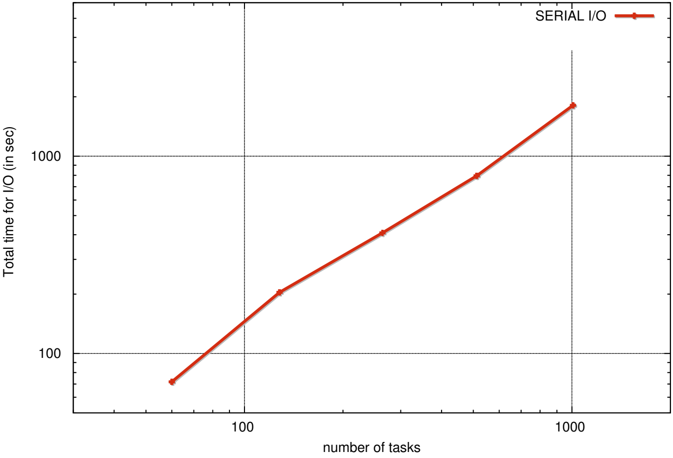
Sion vs. Task-local files

- 41 Mio particles: JuROPA

Scaling

Sion vs. Task-local files

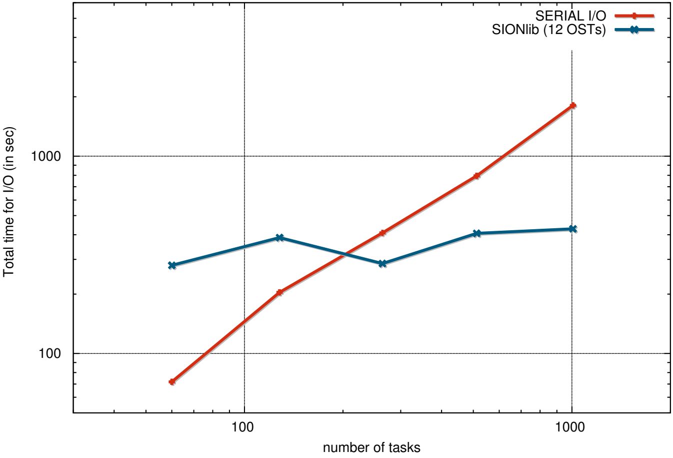
- 41 Mio particles: JuROPA



Scaling

Sion vs. Task-local files

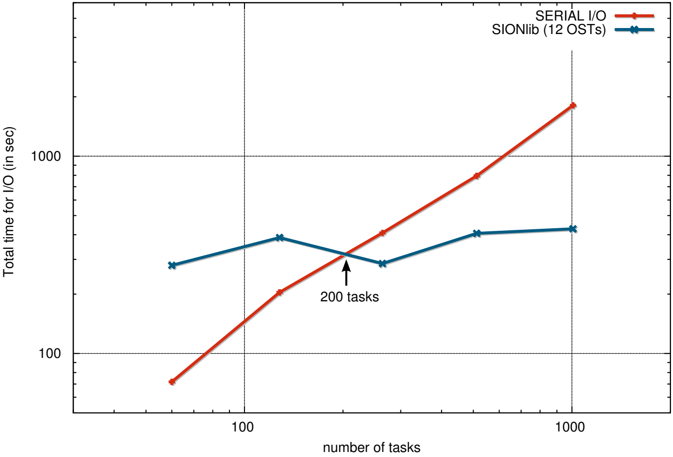
- 41 Mio particles: JuROPA



Scaling

Sion vs. Task-local files

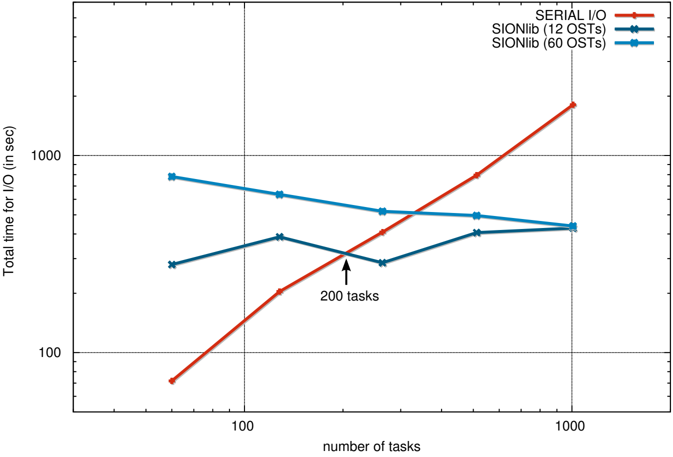
- 41 Mio particles: JuROPA



Scaling

Sion vs. Task-local files

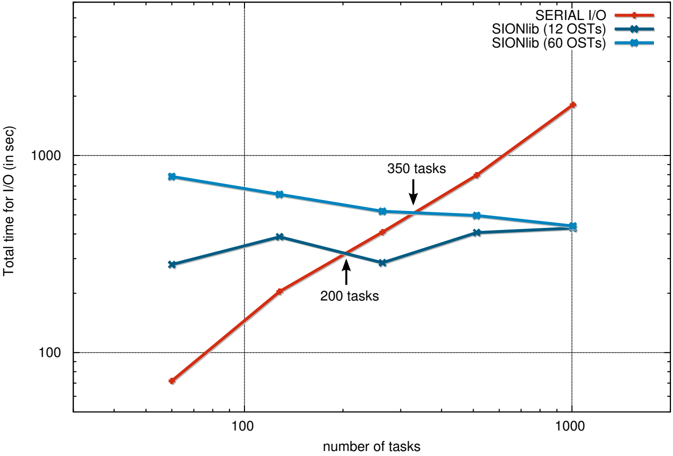
- 41 Mio particles: JuROPA



Scaling

Sion vs. Task-local files

- 41 Mio particles: JuROPA



Scaling

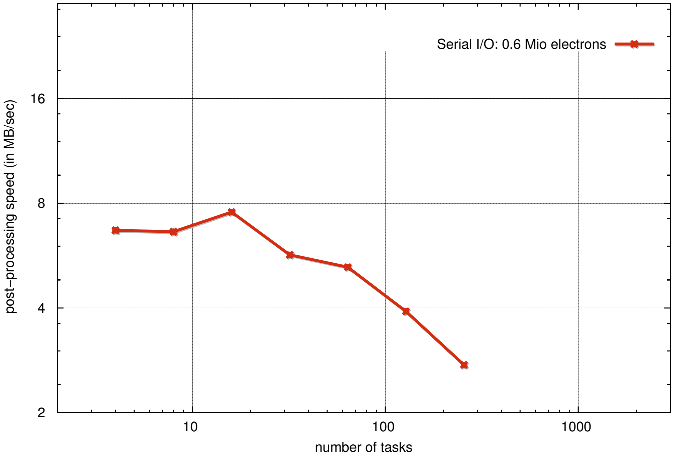
Post-processing time

- Converting all timesteps of a simulation (1 core, 2.93GHz):

Scaling

Post-processing time

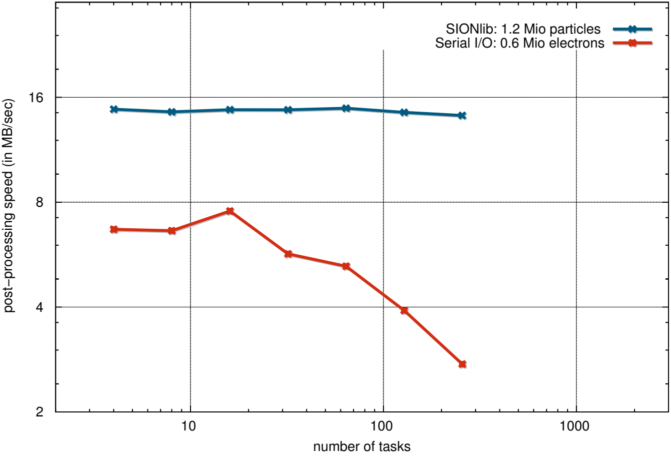
- Converting all timesteps of a simulation (1 core, 2.93GHz):



Scaling

Post-processing time

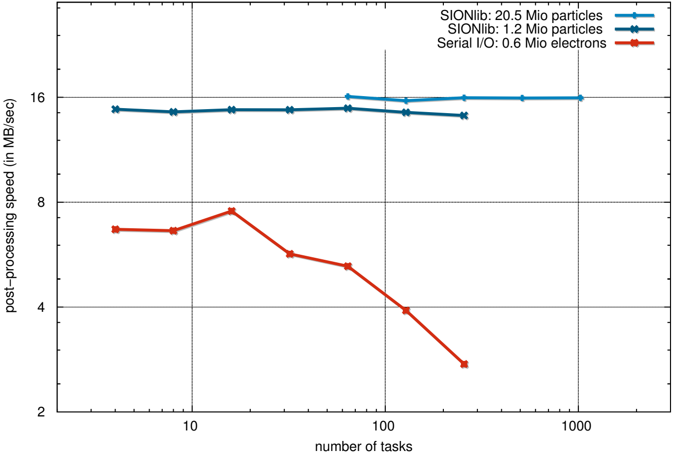
- Converting all timesteps of a simulation (1 core, 2.93GHz):



Scaling

Post-processing time

- Converting all timesteps of a simulation (1 core, 2.93GHz):



Visualization

Wake field acceleration

- A laser pulse propagates through a plasma

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

Efficient parallel I/O

Part VI: Conclusion

September 27, 2010 | Axel Hübl

Conclusion

What is possible now

Achievements

- single file per time step for fields and particles

Conclusion

What is possible now

Achievements

- single file per time step for fields and particles
- portable & fast:

Conclusion

What is possible now

Achievements

- single file per time step for fields and particles
- portable & fast:
  - litte/big endian independent

Conclusion

What is possible now

Achievements

- single file per time step for fields and particles
- portable & fast:
  - litte/big endian independent
  - fast post-processing

Conclusion

What is possible now

Achievements

- single file per time step for fields and particles
- portable & fast:
  - litte/big endian independent
  - fast post-processing
  - widely supported output formats

Conclusion

What is possible now

Achievements

- single file per time step for fields and particles
- portable & fast:
  - litte/big endian independent
  - fast post-processing
  - widely supported output formats
- possible parallel visualization through Paraview for large scale/ultra-high resolution productions

Conclusion

What is possible now

Achievements

- single file per time step for fields and particles
- portable & fast:
  - litte/big endian independent
  - fast post-processing
  - widely supported output formats
- possible parallel visualization through Paraview for large scale/ultra-high resolution productions
- I/O is now ready for future improvements of PSC

Conclusion

What is possible now

Achievements

- single file per time step for fields and particles
- portable & fast:
  - litte/big endian independent
  - fast post-processing
  - widely supported output formats
- possible parallel visualization through Paraview for large scale/ultra-high resolution productions
- I/O is now ready for future improvements of PSC
- Converters and output modules could also be used for similar projects

Conclusion

What is possible now

Future prospects

- Checkpointing with SIONlib

Conclusion

What is possible now

Future prospects

- Checkpointing with SIONlib
- More converters: hdf5, netcdf, ...

Conclusion

What is possible now

Future prospects

- Checkpointing with SIONlib
- More converters: hdf5, netcdf, ...
- visualization scripting for publication quality

Conclusion

What is possible now

Future prospects

- Checkpointing with SIONlib
- More converters: hdf5, netcdf, ...
- visualization scripting for publication quality
- satisfactory preliminary comparison for the physical output, but microscopical comparison for a set of physical problems needed

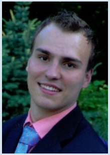
Thank you for your attention!

Contact

Author, Disclaimer

Author

- Axel Hübl, TU Dresden
- Physics student (Diploma)
- axel.huebl _at_ web.de



Disclaimer

- Some images used in this talk are intellectual property of other authors and may not be distributed or reused without their explicit approval.