Parallelization of the full-wave code REFMULX

Tiago Ribeiro[1], Filipe da Silva[2] and Stéphane Heuraux[3]

[1] Max-Planck-Institut für Plasmaphysik 85748 Garching, Germany
[2] Instituto de Plasmas e Fusão Nuclear, Instituto Superior Técnico, Universidade de Lisboa 1049-001 Lisboa, Portugal
[3] Institut Jean Lamour, UMR 7198 CNRS-University of Lorraine, Vandoeuvre, France

This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

Motivation

Simulate microwave reflectometry diagnostics for large magnetic fusion devices like ITER.

- Standard experimental diagnostic
  - n_e and E_r profiles, turbulence, plasma position...
- Often measurements require simulation aided interpretation

Motivation

HLST

Simulate microwave reflectometry diagnostics for large magnetic fusion devices like ITER.

- Standard experimental diagnostic
  - n_e and E_r profiles, turbulence, plasma position...
- Often measurements require simulation aided interpretation
- Finite differences time-domain (FDTD) full-wave simulations
  - time-dependent solutions
  - easy incorporation of plasma phenomena
  - fine space and time resolution required: CFL condition
  - both CPU and memory cost-intensive, especially for 2D/3D
- REFMUL family of codes, developed at IPFN-Lisbon

Introdution
REFMULX model

Original serial code: REFMULX
Serial code profiling and optimization

Parallelization strategy
OpenMP, MPI and hybrid parallelism

Performance results
Intra-node strong scaling
Inter-node strong scaling
Inter-node weak scaling

Conclusions & outlook

Introdution
REFMULX model

Original serial code: REFMULX
Serial code profiling and optimization

Parallelization strategy
OpenMP, MPI and hybrid parallelism

Performance results
Intra-node strong scaling
Inter-node strong scaling
Inter-node weak scaling

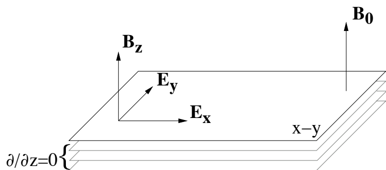
Conclusions & outlook

REFMULX model approximations

Solve for Maxwell's equations + constitutive relation (current)

$$\begin{aligned}
\nabla \times \mathbf{H} &= \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t} + \sigma \mathbf{E} + \mathbf{J} \\
\nabla \times \mathbf{E} &= -\mu_0 \frac{\partial \mathbf{H}}{\partial t} - \sigma^\star \mathbf{E} \\
\frac{\partial \mathbf{J}}{\partial t} &= \varepsilon_0 \omega_p^2 \mathbf{E} - \nu \mathbf{J} + \vec{\omega}_c \times \mathbf{J}
\end{aligned}$$

- propagation in $x - y$ plane
- static background $\mathbf{B}_0$
- no azimut. gradients $\partial/\partial z = 0$
- X-mode propagation $\mathbf{H} \parallel \mathbf{B}_0$



REFMULX model: Cartesian coords.

$$
\begin{aligned}
\varepsilon_0 \frac{\partial E_x}{\partial t} + \sigma E_x &= \frac{\partial H_z}{\partial y} - J_x \\
\varepsilon_0 \frac{\partial E_y}{\partial t} + \sigma E_y &= -\frac{\partial H_z}{\partial x} - J_y \\
\mu_0 \frac{\partial H_z}{\partial t} + \sigma^\star H_z &= \frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} \\
\frac{\partial J_x}{\partial t} &= \varepsilon_0 \omega_p^2 E_x - \nu J_x - \omega_c J_y \\
\frac{\partial J_y}{\partial t} &= \varepsilon_0 \omega_p^2 E_y - \nu J_y + \omega_c J_x
\end{aligned}
$$

Curl equations evolved with Yee scheme (1966)
Plasma-wave coupling solvers for **J**:

- Xu & Yuan (XY) 2006
- Déspres & Campos Pinto (DP) 2013

Introdution
REFMULX model

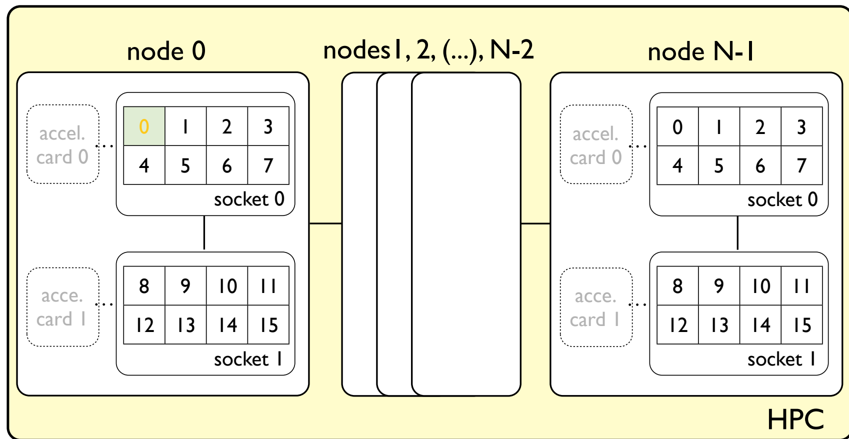
Original serial code: REFMULX
Serial code profiling and optimization

Parallelization strategy
OpenMP, MPI and hybrid parallelism

Performance results
Intra-node strong scaling
Inter-node strong scaling
Inter-node weak scaling

Conclusions & outlook

Serial code profiling and optimization



Serial code profiling

Gprof: GNU Profiling software

```
    % cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
34.61     30.94    30.94      700    44.19    44.19  calcHzFieldKXY
23.19     51.66    20.72      700    29.61    29.61  calcEyFieldKXY
21.90     71.23    19.57      700    27.96    27.96  calcJxyFieldKXY
18.93     88.16    16.92      700    24.18    24.18  calcExFieldKXY
 1.30     89.32     1.16      700     1.66     1.66  addFLDMTRX
 0.06     89.37     0.05       22     2.27     2.27  initFLDMTRX
 0.02     89.39     0.02        4     5.00     5.00  escrvField
 0.01     89.40     0.01        1    10.00    10.00  setNe
 0.00     89.40     0.00     1400     0.00     0.00  setTCut
 0.00     89.40     0.00      700     0.00     0.00  repointJxyFields
 0.00     89.40     0.00      700     0.00     0.00  setSinLn
(....)
```

Solver functions (XY) represent 99% of total cost: hot-spot

Serial code optimization

Restriction of pointer's scope within a given context

```
float *__restrict pHzx, *__restrict pEy;
pHzx=Hzx.m;
pEy=Ey.m;

for(i=0;j<M;i++) pHzx[i]=pHzx[i]+pEy[i+1];
```

- writes through pHzx don't affect values read through pEy

Serial code optimization

Restriction of pointer's scope within a given context

```
float *__restrict pHzx, *__restrict pEy;
pHzx=Hzx.m;
pEy=Ey.m;

for(i=0;j<M;i++) pHzx[i]=pHzx[i]+pEy[i+1];
```

- writes through pHzx don't affect values read through pEy

	original code	restricted	speedup
GNU C compiler	14 633 s	13 275 s	1.1
Intel C compiler	14 965 s	2 634 s	5.7

potentially less data reloading to hardware registers

Serial code optimization

Minimize FLOP inside loops, especially divisions (mock-up example)

```
for(i=0;i<Hzx.imx;i++)
{
  ax=dt*pMgCx[i]/(2*mu0);
  bx=dt/(mu0*dx*(1+ax));

  pHzx[i]=
    bx*pHzx[i]+
    (1-ax)/(1+ax)*pEy[i];
}
```

```
cax=dt/(2*mu0);
cbx=dt/(mu0*dx);
for(i=0;i<Hzx.imx;i++)
{
  ax=cax*pMgCx[i];
  bx=1/(1+ax);

  pHzx[i]=bx*(
      cbx*pHzx[i]+
      (1-ax)*pEy[i]);
}
```

Serial code optimization

HLST

Minimize FLOP inside loops, especially divisions (mock-up example)

```
for(i=0;i<Hzx.imx;i++)
{
  ax=dt*pMgCx[i]/(2*mu0);
  bx=dt/(mu0*dx*(1+ax));

  pHzx[i]=
    bx*pHzx[i]+
    (1-ax)/(1+ax)*pEy[i];
}
```

```
cax=dt/(2*mu0);
cbx=dt/(mu0*dx);
for(i=0;i<Hzx.imx;i++)
{
  ax=cax*pMgCx[i];
  bx=1/(1+ax);

  pHzx[i]=bx*(
      cbx*pHzx[i]+
      (1-ax)*pEy[i]);
}
```

	original code	restricted	min. FLOP	speedup
GNU C compiler	14 633 s	13 275 s	5 670 s	2.5
Intel C compiler	14 965 s	2 634 s	1 802 s	8.3

Introduction
REFMULX model

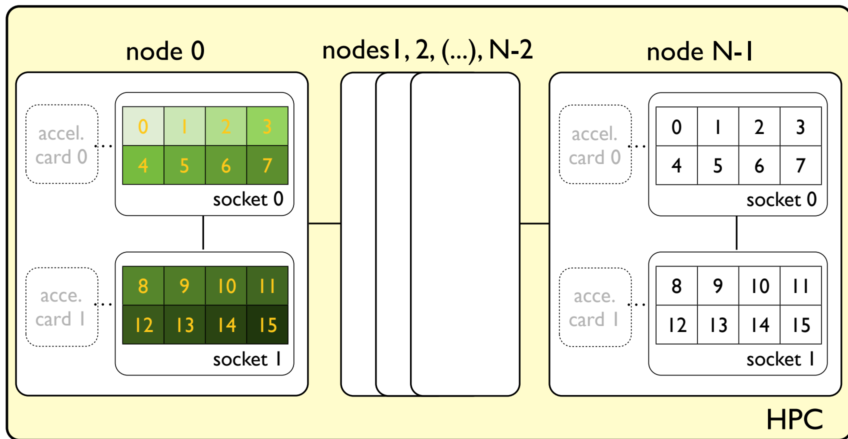
Original serial code: REFMULX
Serial code profiling and optimization

Parallelization strategy
OpenMP, MPI and hybrid parallelism

Performance results
Intra-node strong scaling
Inter-node strong scaling
Inter-node weak scaling

Conclusions & outlook

OpenMP thread parallelism



OpenMP thread parallelism

Use OpenMP parallel loop directive on hot-spot functions

- threads in slow index (C row-major order)
- data scoping inside parallel regions (shared/private)

```
#pragma omp parallel for default(none)                    \
  firstprivate(cax, cbx, cay, cby, ...)                   \
  shared(Hzx, Hzy, Ex, Ey, ...)                           \
  private(i, j, ax, bx, pHzx, pHzy, pEx, pEy, ... )
  for(j=0;j<Hzx.jmx;j++)
    { (...)
      for(i=0;i<Hzx.imx;i++)
        { (...)
```

OpenMP thread parallelism

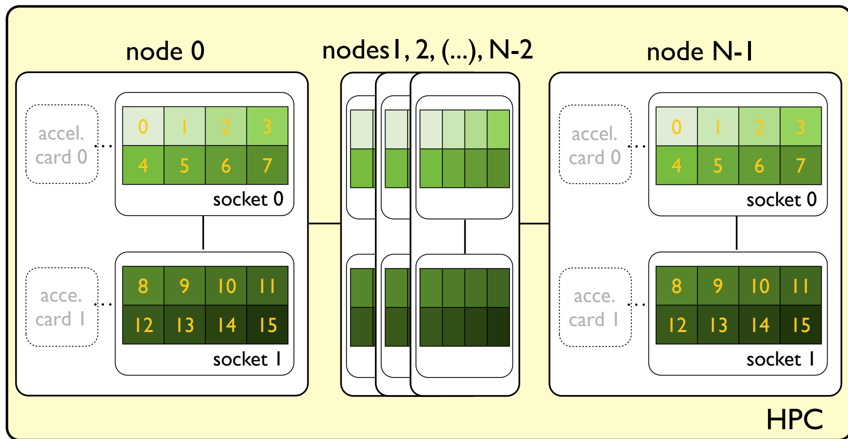
Use OpenMP parallel loop directive on hot-spot functions

- threads in slow index (C row-major order)
- data scoping inside parallel regions (shared/private)

```
#pragma omp parallel for default(none)                    \
  firstprivate(cax, cbx, cay, cby, ...)                   \
  shared(Hzx, Hzy, Ex, Ey, ...)                           \
  private(i, j, ax, bx, pHzx, pHzy, pEx, pEy, ... )
  for(j=0;j<Hzx.jmx;j++)
    { (...)
      for(i=0;i<Hzx.imx;i++)
        { (...)
```

	original code	optimized	16 threads	speedup
Intel C compiler	14 965 s	1 802 s	299 s	50.1 (6.0)

MPI task parallelism

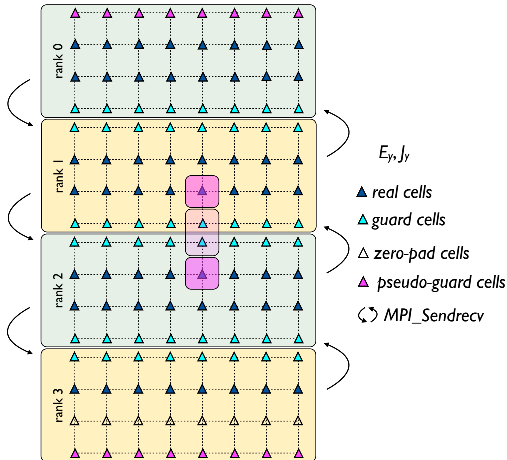


MPI task parallelism

- MPI parallelization is more powerful/flexible
  - extends beyond compute node
  - typical good scaling properties
- BUT requires bigger implementation effort
  - explicit domain decomposition
  - explicit data communication

- decision
  - decomposition over slow-varying index (as in OpenMP)
- challenges
  - staggered grids
  - localized boundary/envelope functions

MPI domain decomposition



- two guard-cells per sub-domain
- (some) extrema sub-domains with zero-pad cells
- localized boundary/envelope functions across sub-domains

Hybrid MPI/OpenMP parallelism

Ideally suited for present-day/future hierarchical architectures

- less guard-cells
  - less data communication
  - less memory (guard cells and MPI communication buffers)
- HELIOS: exploit intra-node shared memory
  - N threads + $16/N$ MPI tasks/node ($N \leq 16$)

- parallelization paradigm
  - MPI tasks on outer loop
  - OMP threads on outer loop

- Already in place: no inter-task communication at thread level

Introdution
REFMULX model

Original serial code: REFMULX
Serial code profiling and optimization

Parallelization strategy
OpenMP, MPI and hybrid parallelism

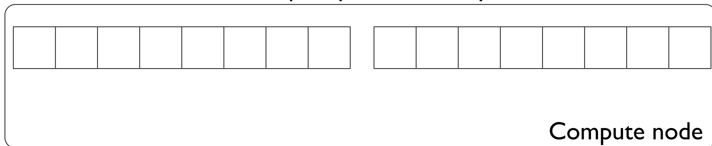
Performance results
Intra-node strong scaling
Inter-node strong scaling
Inter-node weak scaling

Conclusions & outlook

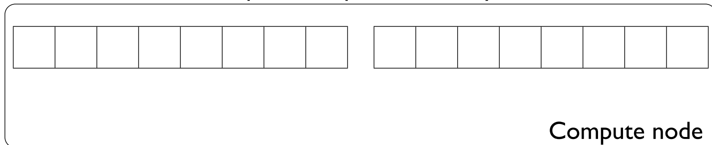
Process affinity & memory hierarchy

8 processes:

Compact process affinity



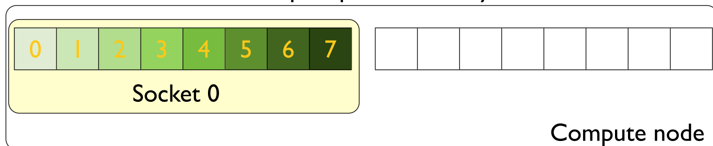
Equidistant process affinity



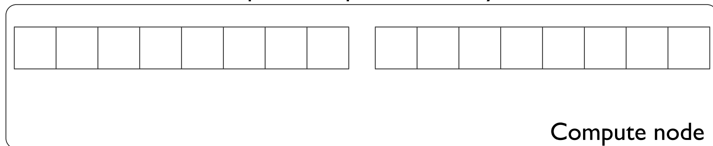
Process affinity & memory hierarchy

8 processes:

Compact process affinity



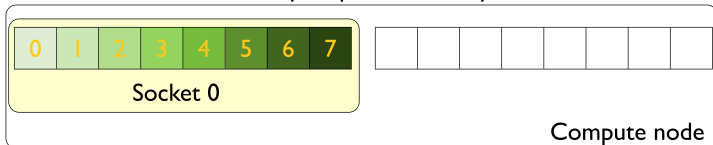
Equidistant process affinity



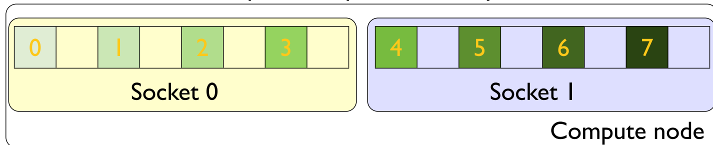
Process affinity & memory hierarchy

8 processes:

Compact process affinity



Equidistant process affinity



Process affinity & memory hierarchy

Intra-node strong scaling (ICC): OpenMP vs. Bullx MPI (1500x1024 grid 120.000 iterations)

REFMULXp equidistant single-thread memory initialization (OMP)

speedup = 22.5x (2.7x)

Execution time (s)

1000

nb threads

1 2 4 8 16

Process affinity & memory hierarchy

Intra-node strong scaling (ICC): OpenMP vs. Bullx MPI (1500x1024 grid 120.000 iterations)

REFMULXp equidistant single-thread memory initialization (OMP)
REFMULXp compact thread affinity (OMP)

Execution time (s)

1000

speedup = 22.5x (2.7x)

speedup = 50.1x (6.0x)

1 2 4 8 16

nb threads

Process affinity & memory hierarchy

Intra-node strong scaling (ICC): OpenMP vs. Bullx MPI (1500x1024 grid 120.000 iterations)

REFMULXp equidistant single-thread memory initialization (OMP)
REFMULXp compact thread affinity (OMP)
REFMULXp equidistant thread affinity (OMP)

speedup = 22.5x (2.7x)

speedup = 50.1x (6.0x)

Execution time (s)

1000

1 2 4 8 16

nb threads

Process affinity & memory hierarchy

Intra-node strong scaling (ICC): OpenMP vs. Bullx MPI (1500x1024 grid 120.000 iterations)

REFMULXp equidistant single-thread memory initialization (OMP)
REFMULXp compact thread affinity (OMP)
REFMULXp equidistant thread affinity (OMP)
REFMULXp compact taks affinity (MPI)
REFMULXp distributed task affinity (MPI)

Execution time (s)

1000

speedup = 22.5x (2.7x)

speedup = 50.1x (6.0x)

1 2 4 8 16

nb threads

- first touch page policy (OMP) & process affinity are key (OMP/MPI)

Introdution
REFMULX model

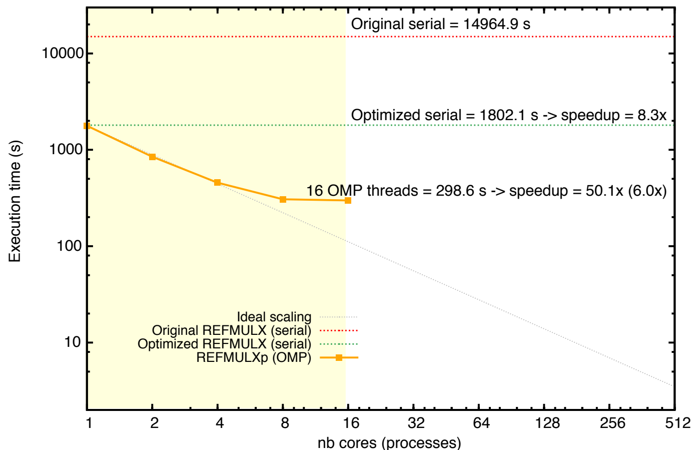
Original serial code: REFMULX
Serial code profiling and optimization

Parallelization strategy
OpenMP, MPI and hybrid parallelism

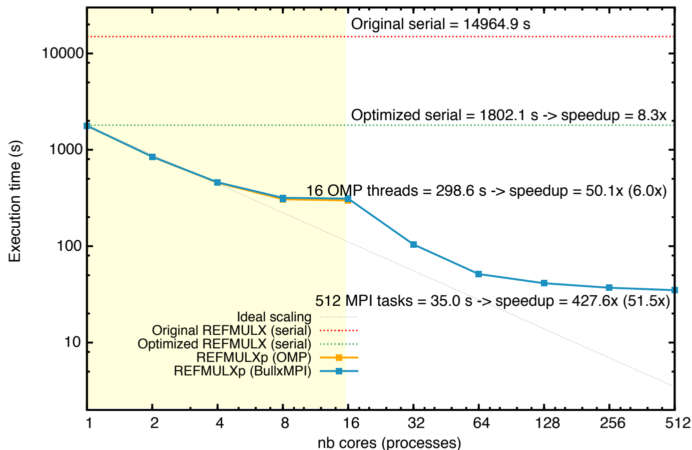
Performance results
Intra-node strong scaling
Inter-node strong scaling
Inter-node weak scaling

Conclusions & outlook

Inter-node scaling: MPI

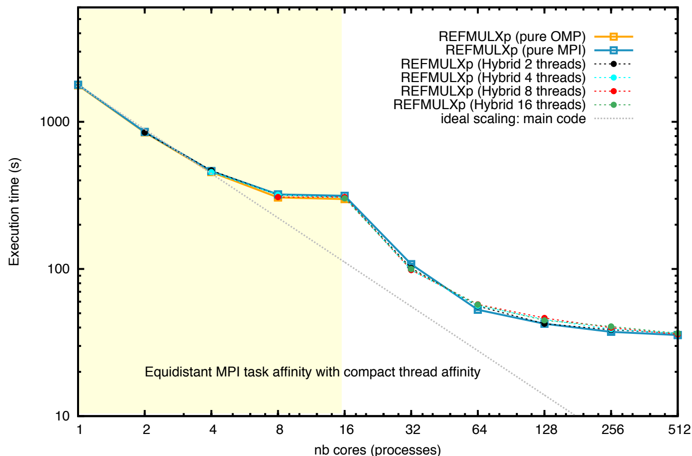


Inter-node scaling: MPI



Scaling saturation: Amidahl's law and communication overhead

Inter-node scaling: MPI vs. hybrid



Hybrid: same performance, less memory footprint

Introdution

REFMULX model

Original serial code: REFMULX

Serial code profiling and optimization

Parallelization strategy

OpenMP, MPI and hybrid parallelism

Performance results

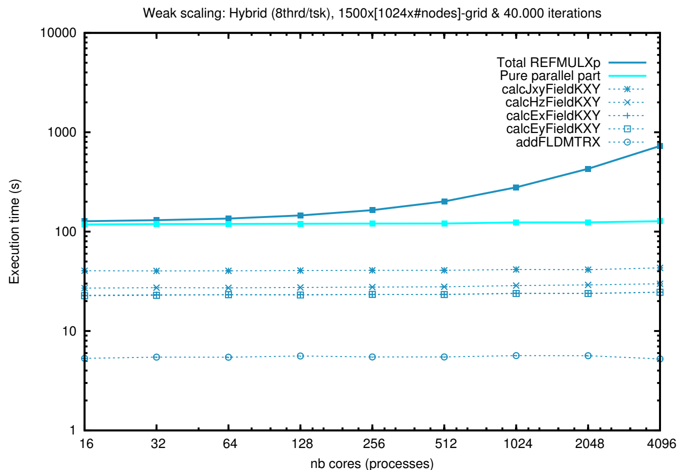
Intra-node strong scaling

Inter-node strong scaling

Inter-node weak scaling

Conclusions & outlook

Inter-node weak scaling: MPI



Perfect scaling of parallel part, I/O becomes bottleneck

Introdution
REFMULX model

Original serial code: REFMULX
Serial code profiling and optimization

Parallelization strategy
OpenMP, MPI and hybrid parallelism

Performance results
Intra-node strong scaling
Inter-node strong scaling
Inter-node weak scaling

Conclusions & outlook

Conclusions & outlook

- Project's main results
  - serial optimization and hybrid parallelization of REFMULX code
  - over two orders of magnitude speedup achieved on standard test-case
  - bigger cases possible, although parallel I/O becomes necessary

	serial		parallel		
	original	optim.	OpenMP	MPI	hybrid
# procs.	1	1	16	512	64×8
time	14 965 s	1 802 s	298.6 s	35.0 s	35.9 s
speedup	1	8.3	50.1	427.6	416.8
			(6.0)*	(51.5)*	(50.2)*

(*) speedup factors relative to optimized serial code

- Work on 3D version ongoing: OpenMP parallelisation done

Backup slides

The team and its goals

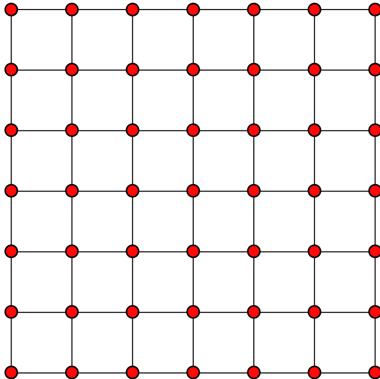
Complementary fields of expertise within the team elements:

- computer science
- applied mathematics
- plasma physics

all with HPC experience

- NOT focused on own research
- driven by proposals submitted at European level
- 78 projects approved since 2009, major EU codes involved
  (SOLPS, JOREK, NEMORB, GENE, EUTERPE, GYGLES, GEM, ...)

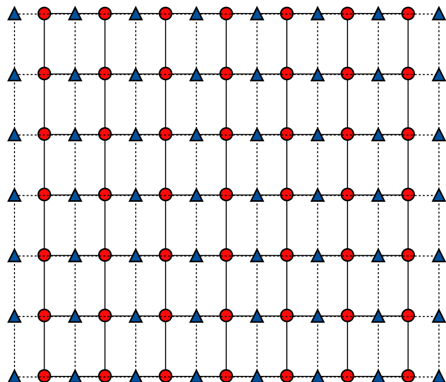
MPI domain decomposition



- staggered grids

● H_z, B_z, n_e

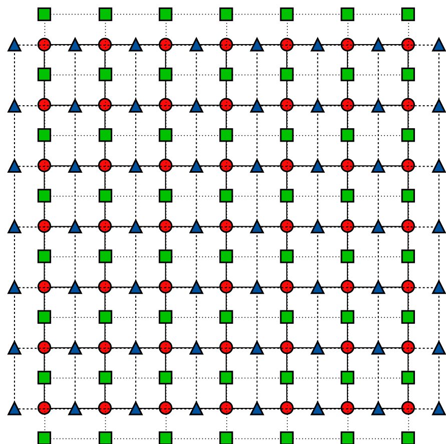
MPI domain decomposition



H_z, B_z, n_e E_y, J_y

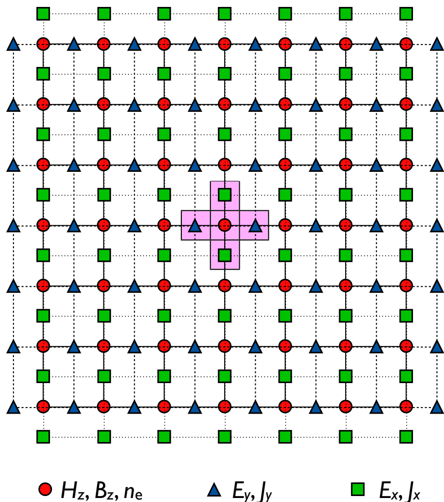
- staggered grids

MPI domain decomposition



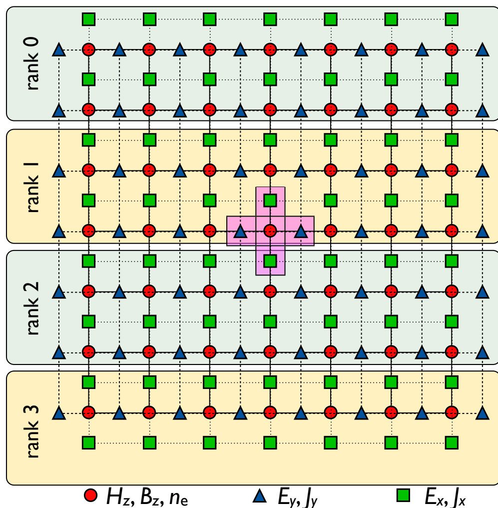
- staggered grids

MPI domain decomposition



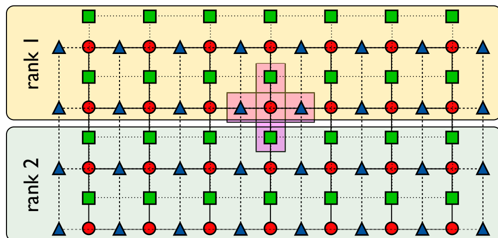
- staggered grids
- stencil: 1st order method

MPI domain decomposition



- staggered grids
- stencil: 1st order method

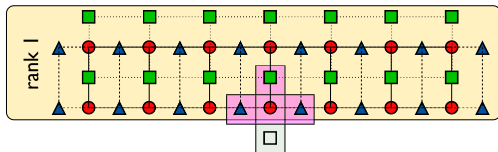
MPI domain decomposition

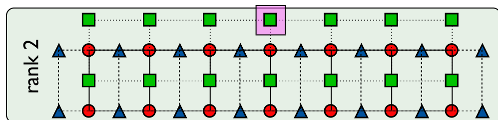


- staggered grids
- stencil: 1st order method

● H_z, B_z, n_e ▲ E_y, J_y ■ E_x, J_x

MPI domain decomposition

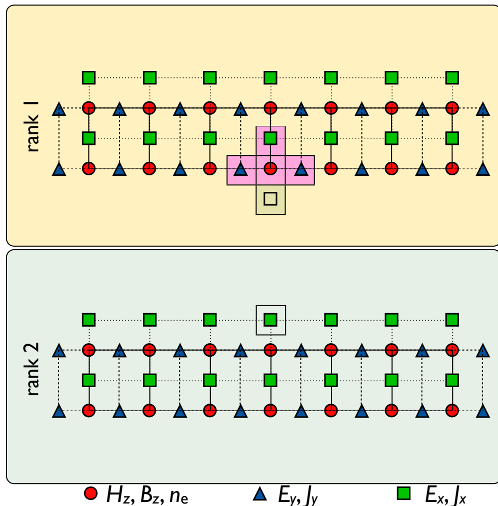




- staggered grids
- stencil: 1st order method

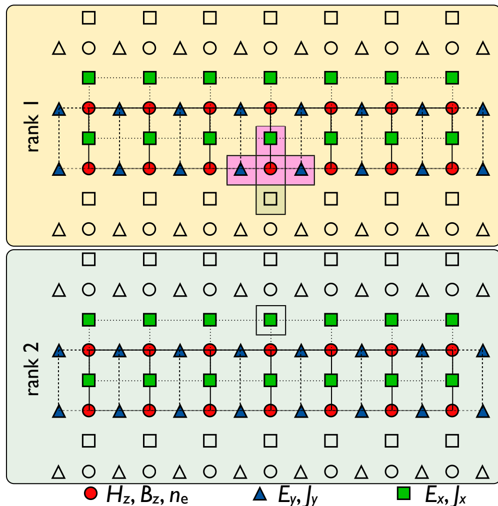
● H_z, B_z, n_e ▲ E_y, J_y ■ E_x, J_x

MPI domain decomposition



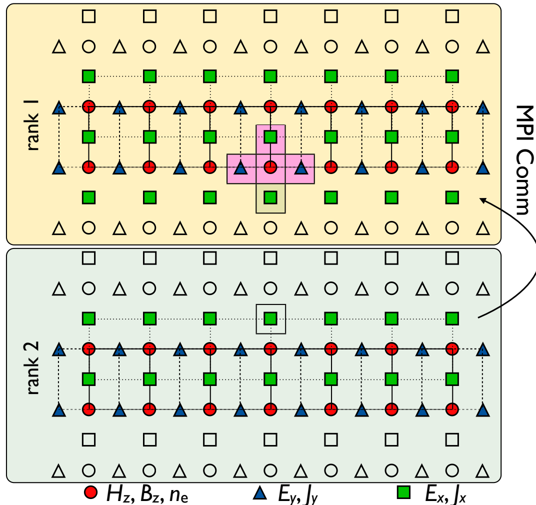
- staggered grids
- stencil: 1st order method

MPI domain decomposition



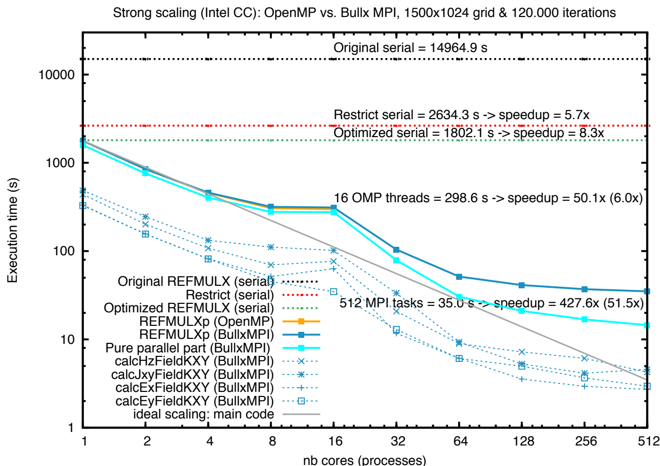
- staggered grids
- stencil: 1st order method
- two guard-cells per grid per sub-domain

MPI domain decomposition



- staggered grids
- stencil: 1st order method
- two guard-cells per grid per sub-domain
- MPI communication required

Inter-node scaling: MPI



Scaling saturation: Amidahl's law and communication overhead