



# Parallelization of the full-wave code REFMULX

Tiago Ribeiro



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

---

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



## Introduction

REFMULX model

## Original serial code: REFMULX

Serial code profiling

Serial code optimization

## Parallelization strategy

OpenMP thread parallelism

MPI task parallelism

Hybrid MPI/OpenMP parallelism

## Performance results

Intra-node strong scaling

Inter-node strong scaling

Inter-node weak scaling

## Conclusions



## Introduction

REFMULX model

### Original serial code: REFMULX

Serial code profiling

Serial code optimization

### Parallelization strategy

OpenMP thread parallelism

MPI task parallelism

Hybrid MPI/OpenMP parallelism

### Performance results

Intra-node strong scaling

Inter-node strong scaling

Inter-node weak scaling

## Conclusions

# REFMULX model approximations



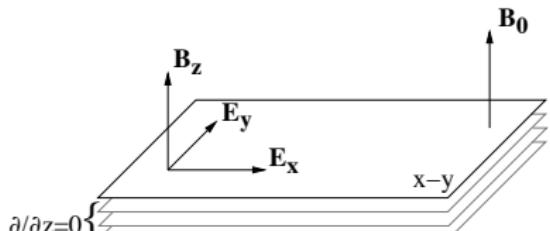
Solve for Maxwell's equations + constitutive relation (current)

$$\nabla \times \mathbf{H} = \epsilon_0 \frac{\partial \mathbf{E}}{\partial t} + \sigma \mathbf{E} + \mathbf{J}$$

$$\nabla \times \mathbf{E} = -\mu_0 \frac{\partial \mathbf{H}}{\partial t} - \sigma^* \mathbf{E}$$

$$\frac{\partial \mathbf{J}}{\partial t} = \epsilon_0 \omega_p^2 \mathbf{E} - \nu \mathbf{J} + \vec{\omega}_c \times \mathbf{J}$$

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



## REFMULX model: Cartesian coords.

---

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

Curl equations evolved with Yee scheme (1966)

Plasma-wave coupling solvers for  $\mathbf{J}$ :

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

## Introduction

REFMULX model

## Original serial code: REFMULX

Serial code profiling

Serial code optimization

## Parallelization strategy

OpenMP thread parallelism

MPI task parallelism

Hybrid MPI/OpenMP parallelism

## Performance results

Intra-node strong scaling

Inter-node strong scaling

Inter-node weak scaling

## Conclusions



# Serial code profiling

## Gprof: GNU Profiling software

% cumulative	self time	seconds	self seconds	calls	ms/call	ms/call	total 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	repaintJxyFields
0.00	89.40	0.00		700	0.00	0.00	setSinLn
(....)							

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

## Introduction

REFMULX model

## Original serial code: REFMULX

Serial code profiling

Serial code optimization

## Parallelization strategy

OpenMP thread parallelism

MPI task parallelism

Hybrid MPI/OpenMP parallelism

## Performance results

Intra-node strong scaling

Inter-node strong scaling

Inter-node weak scaling

## Conclusions



# 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



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

Serial code optimization

## Parallelization strategy

OpenMP thread parallelism

MPI task parallelism

Hybrid MPI/OpenMP parallelism

## Performance results

Intra-node strong scaling

Inter-node strong scaling

Inter-node weak scaling

## Conclusions

# OpenMP threads

---

Use OpenMP parallel loop directive on hot-spot functions

- C row-major order: i contiguous in memory (x-direction)
  - threads in slow varying index j: outer loop (y-direction)
  - properly set 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 threads

Use OpenMP parallel loop directive on hot-spot functions

- C row-major order: *i* contiguous in memory (*x*-direction)
- threads in slow varying index *j*: outer loop (*y*-direction)
- properly set 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	restricted	16 threads	speedup
Intel C compiler	14 965 s	2 634 s	299 s	50.1 (6.0)



## Introduction

REFMULX model

## Original serial code: REFMULX

Serial code profiling

Serial code optimization

## Parallelization strategy

OpenMP thread parallelism

**MPI task parallelism**

Hybrid MPI/OpenMP parallelism

## Performance results

Intra-node strong scaling

Inter-node strong scaling

Inter-node weak scaling

## Conclusions

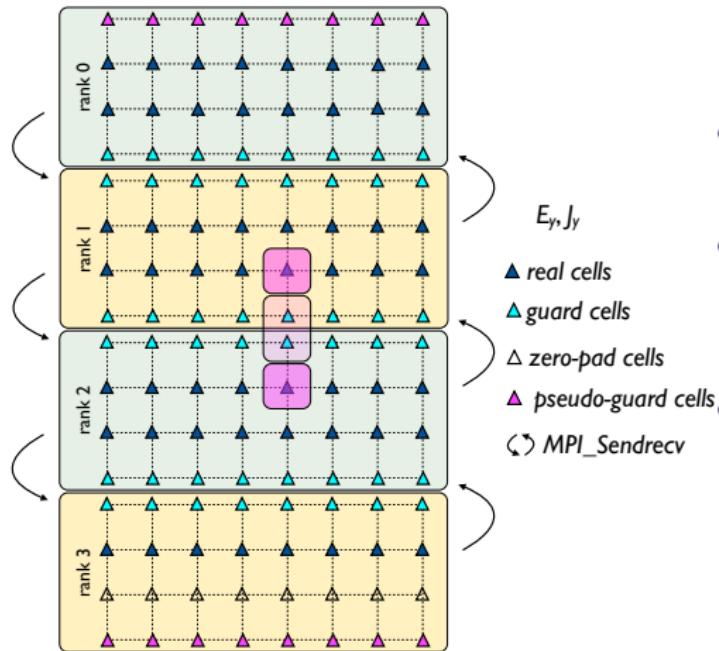
- MPI parallelization is more powerful/flexible
  - extends beyond compute node
  - typical good scaling properties
- BUT requires bigger implementation effort
  - explicit domain decomposition
  - explicit data communication

# 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



## Introduction

REFMULX model

## Original serial code: REFMULX

Serial code profiling

Serial code optimization

## Parallelization strategy

OpenMP thread parallelism

MPI task parallelism

Hybrid MPI/OpenMP parallelism

## Performance results

Intra-node strong scaling

Inter-node strong scaling

Inter-node weak scaling

## Conclusions

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 ( $N \leq 16$ )

# 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



## Introduction

REFMULX model

## Original serial code: REFMULX

Serial code profiling

Serial code optimization

## Parallelization strategy

OpenMP thread parallelism

MPI task parallelism

Hybrid MPI/OpenMP parallelism

## Performance results

**Intra-node strong scaling**

Inter-node strong scaling

Inter-node weak scaling

## Conclusions

# Process affinity & memory hierarchy



8 processes:



Compact process affinity



Equidistant process affinity



Distributed process affinity (“HELIOS place”)

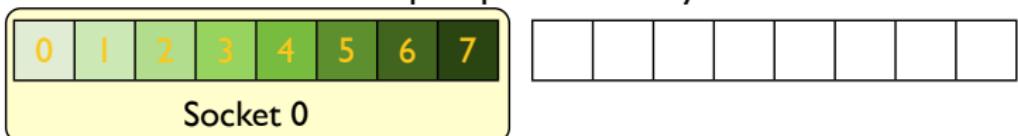


# Process affinity & memory hierarchy

8 processes:



Compact process affinity



Equidistant process affinity



Distributed process affinity (“HELIOS place”)

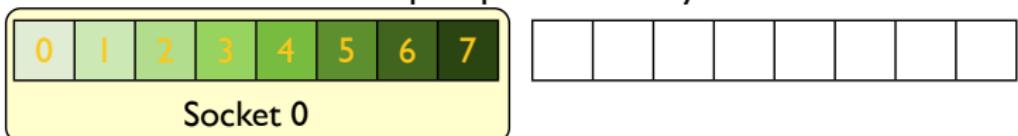


# Process affinity & memory hierarchy

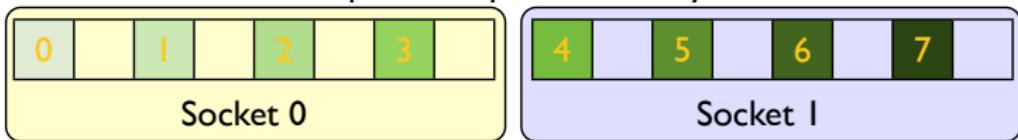
8 processes:



Compact process affinity



Equidistant process affinity



Distributed process affinity (“HELIOS place”)

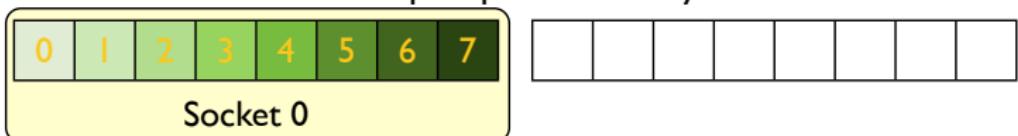


# Process affinity & memory hierarchy

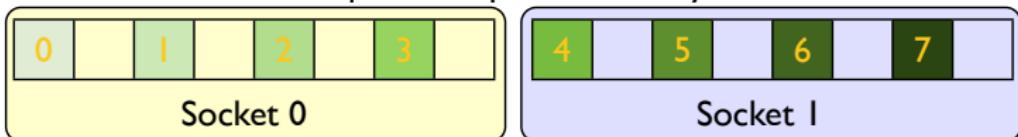
8 processes:



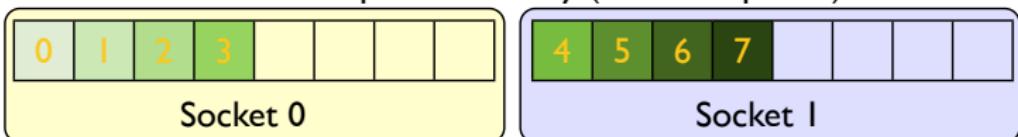
Compact process affinity



Equidistant process affinity

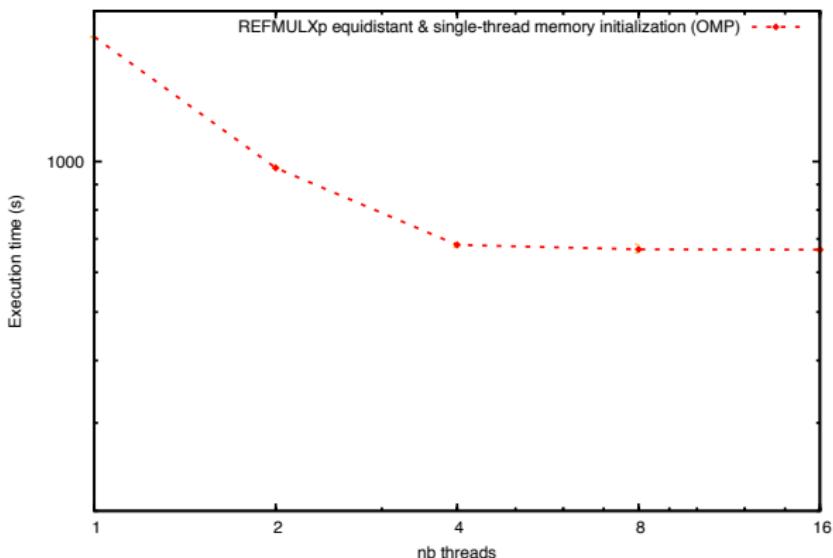


Distributed process affinity (“HELIOS place”)



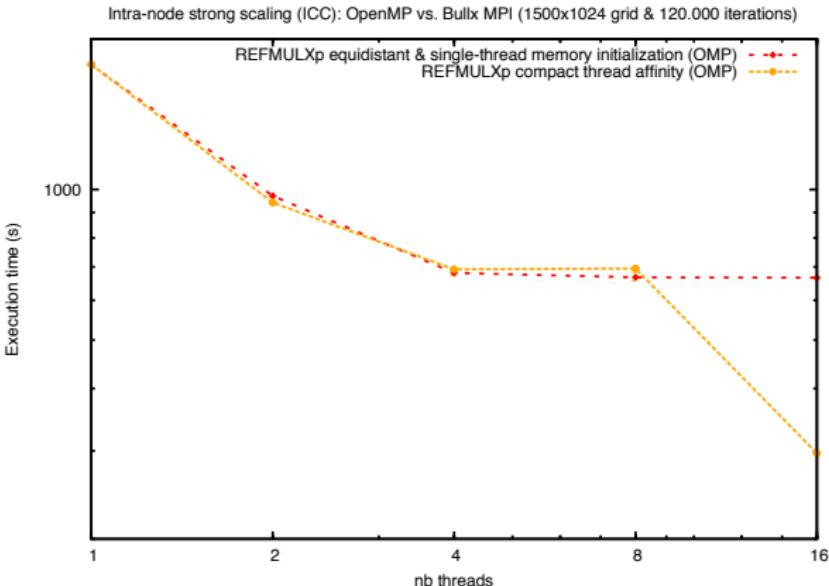
# Process affinity & memory hierarchy

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



- first touch page policy (OpenMP)

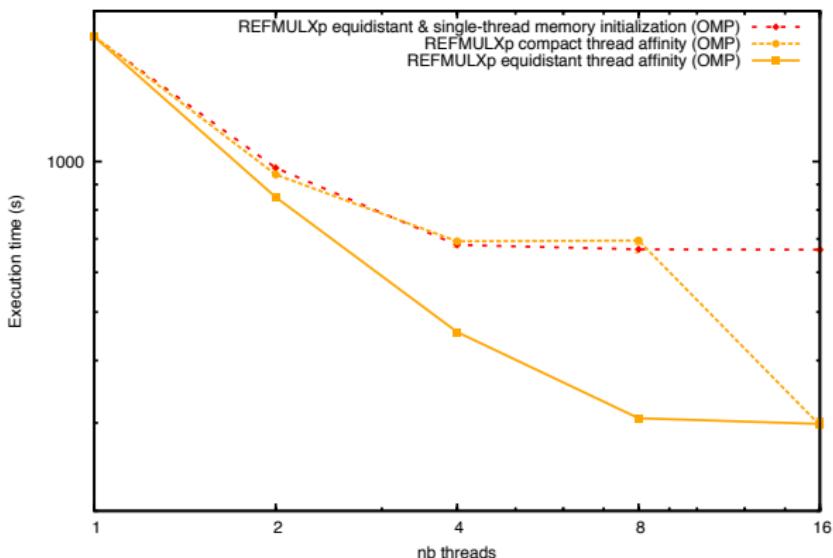
# Process affinity & memory hierarchy



- first touch page policy (OpenMP)

# Process affinity & memory hierarchy

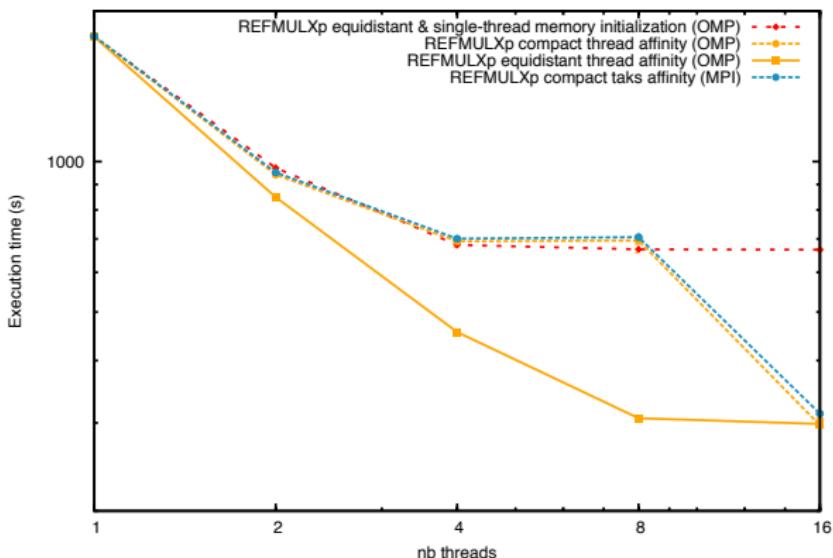
Intra-node strong scaling (ICC): OpenMP vs. Bullx MPI (1500x1024 grid & 120.000 iterations)



- first touch page policy (OpenMP)
- process-to-core assignment (OpenMP/MPI)

# Process affinity & memory hierarchy

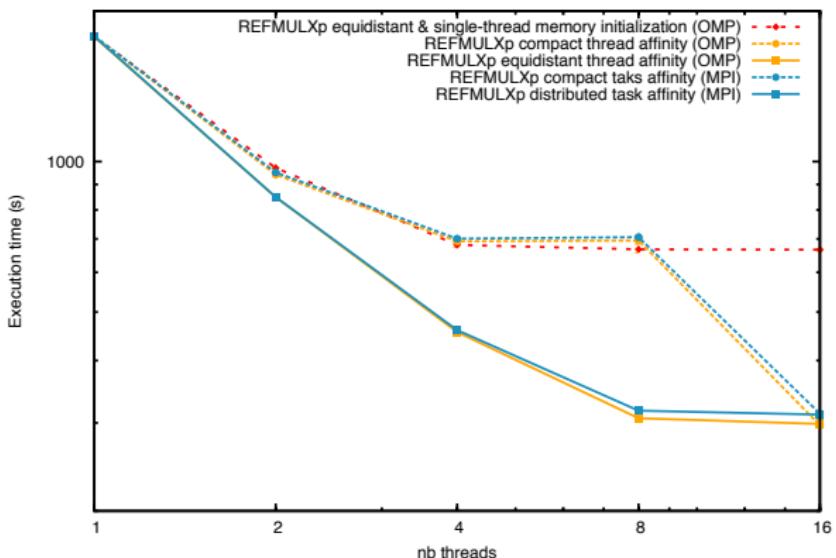
Intra-node strong scaling (ICC): OpenMP vs. Bullx MPI (1500x1024 grid & 120.000 iterations)



- first touch page policy (OpenMP)
- process-to-core assignment (OpenMP/MPI)

# Process affinity & memory hierarchy

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



- first touch page policy (OpenMP)
- process-to-core assignment (OpenMP/MPI)



## Introduction

REFMULX model

## Original serial code: REFMULX

Serial code profiling

Serial code optimization

## Parallelization strategy

OpenMP thread parallelism

MPI task parallelism

Hybrid MPI/OpenMP parallelism

## Performance results

Intra-node strong scaling

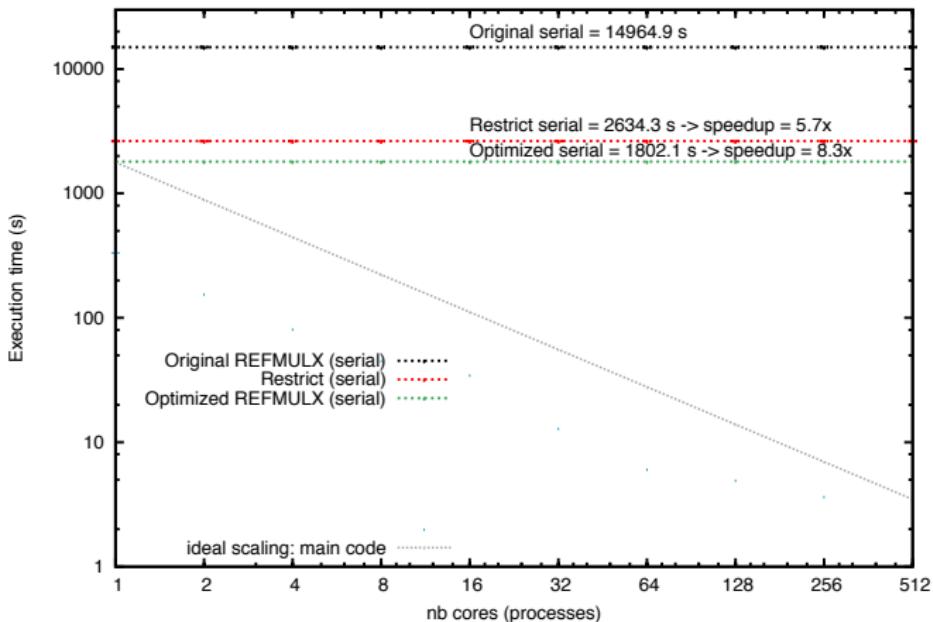
**Inter-node strong scaling**

Inter-node weak scaling

## Conclusions

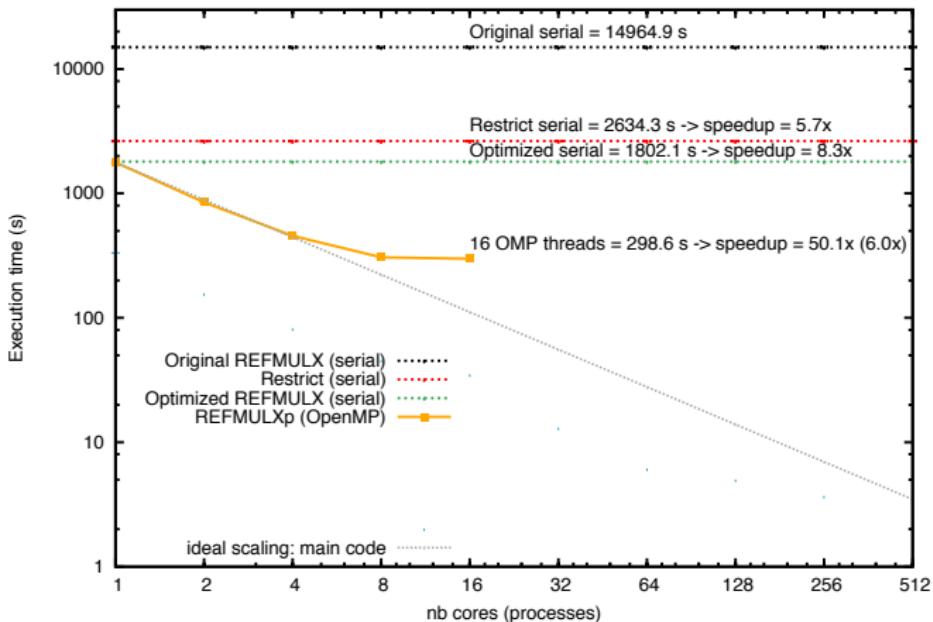
# Inter-node scaling: MPI

Strong scaling (Intel CC): OpenMP vs. Bullx MPI, 1500x1024 grid & 120.000 iterations

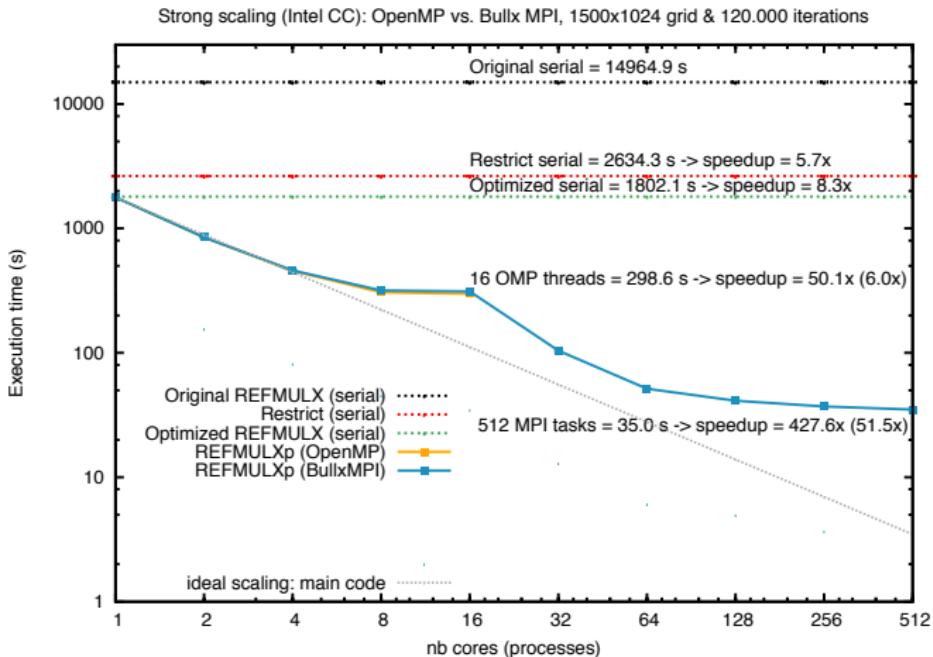


# Inter-node scaling: MPI

Strong scaling (Intel CC): OpenMP vs. Bullx MPI, 1500x1024 grid & 120.000 iterations



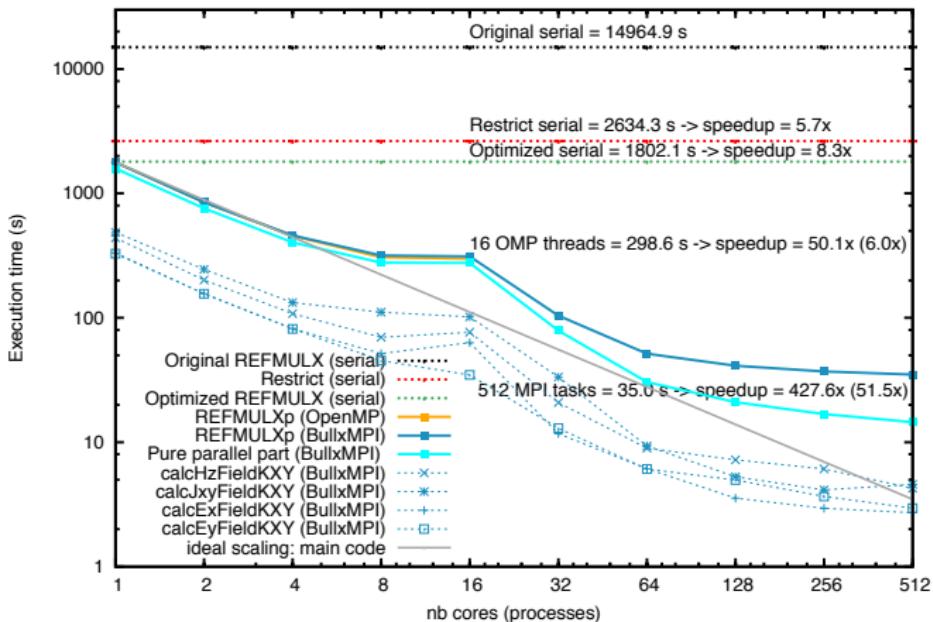
# Inter-node scaling: MPI



# Inter-node scaling: MPI



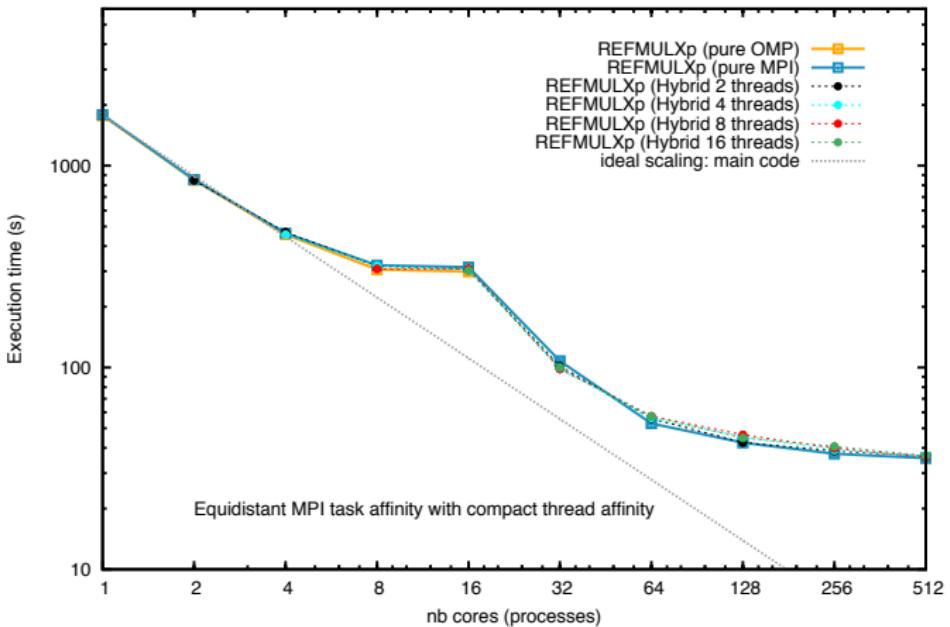
Strong scaling (Intel CC): OpenMP vs. Bullx MPI, 1500x1024 grid & 120.000 iterations



Scaling saturation: Amidahl's law and communication overhead

# Inter-node scaling: MPI vs. hybrid

Strong scaling (Intel CC & Bullx MPI). Pure vs. Hybrid, 1500x1024 grid & 120.000 iterations



Hybrid: same performance, less memory footprint



## Introduction

REFMULX model

## Original serial code: REFMULX

Serial code profiling

Serial code optimization

## Parallelization strategy

OpenMP thread parallelism

MPI task parallelism

Hybrid MPI/OpenMP parallelism

## Performance results

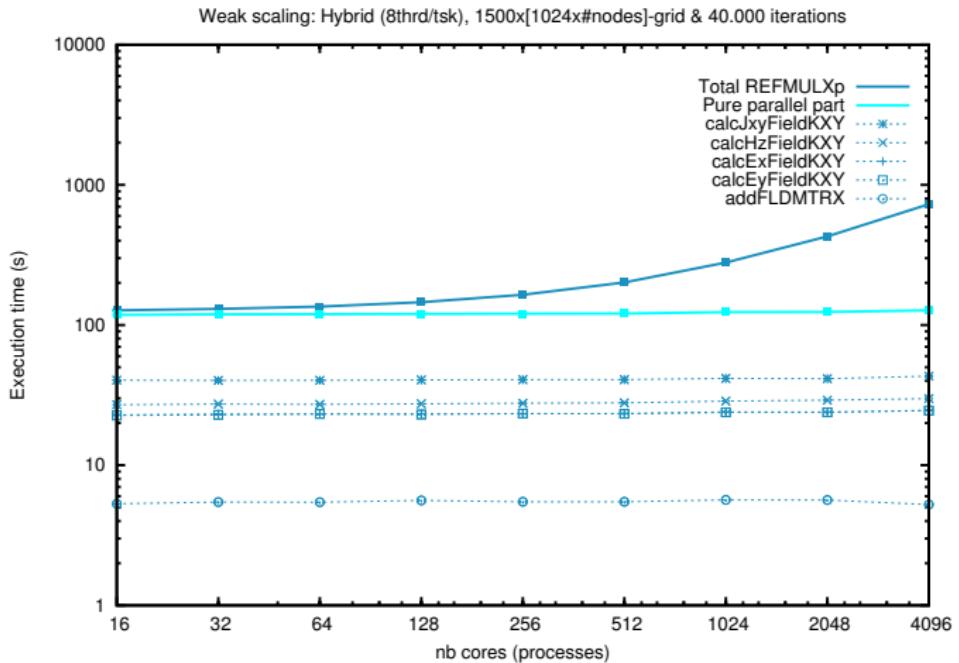
Intra-node strong scaling

Inter-node strong scaling

**Inter-node weak scaling**

## Conclusions

# Inter-node weak scaling: MPI



Perfect scaling of parallel part, I/O becomes bottleneck



## Introduction

REFMULX model

## Original serial code: REFMULX

Serial code profiling

Serial code optimization

## Parallelization strategy

OpenMP thread parallelism

MPI task parallelism

Hybrid MPI/OpenMP parallelism

## Performance results

Intra-node strong scaling

Inter-node strong scaling

Inter-node weak scaling

## Conclusions

# Conclusions

- Project REFMUL2P'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	$64 \times 8$
time	14 965 s	1 802 s	298.6 s	35.0 s	35.9 s
speedup	1	8.3	50.1 (6.0)*	<b>427.6</b> (51.5)*	<b>416.8×</b> (50.2)*

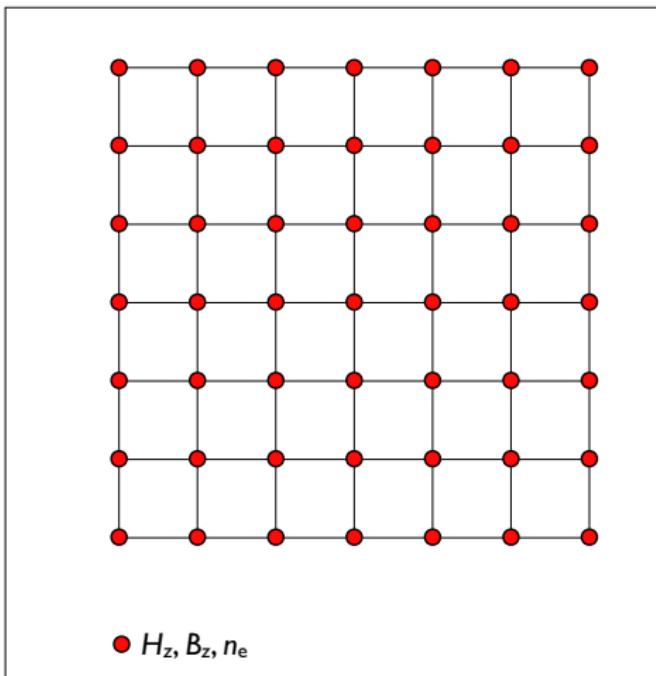
(\*) speedup factors relative to optimized serial code

Backup slides

# REFMULX's grids



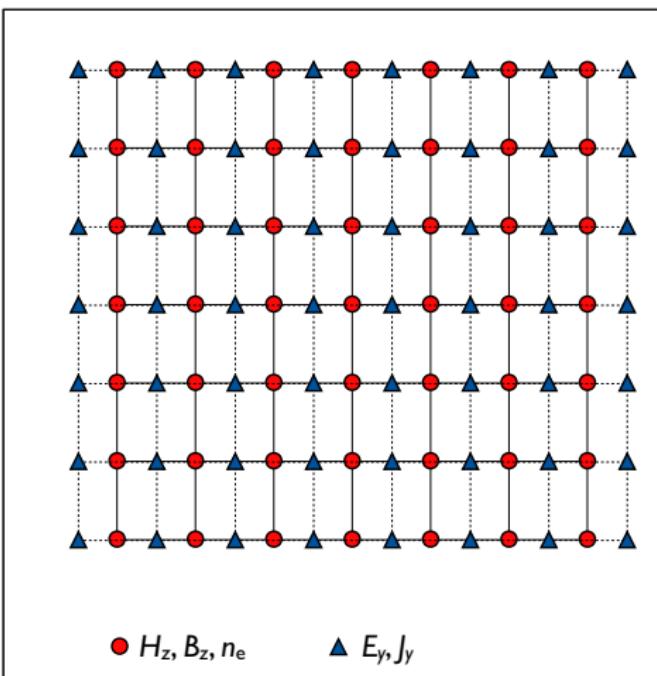
Staggered grids & localized envelopes/boundary layers



# REFMULX's grids



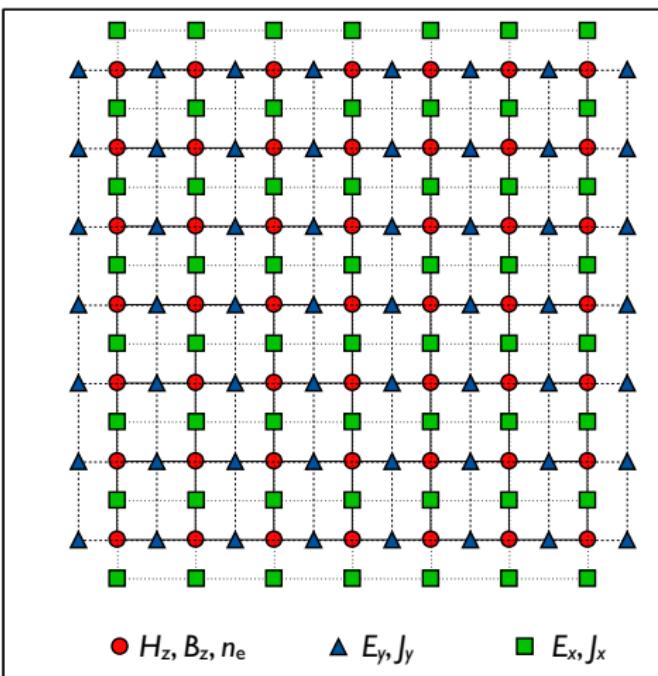
Staggered grids & localized envelopes/boundary layers



# REFMULX's grids



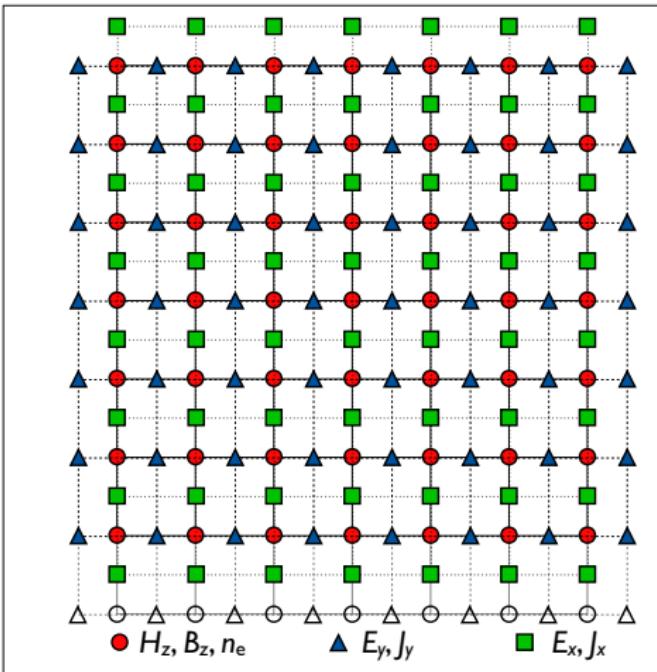
Staggered grids & localized envelopes/boundary layers



# REFMULX's grids



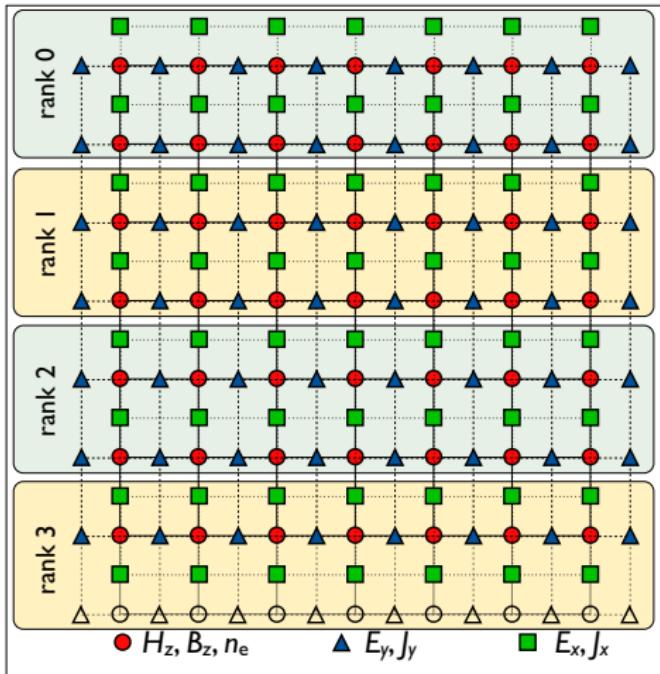
Staggered grids & localized envelopes/boundary layers



# REFMULX's grids

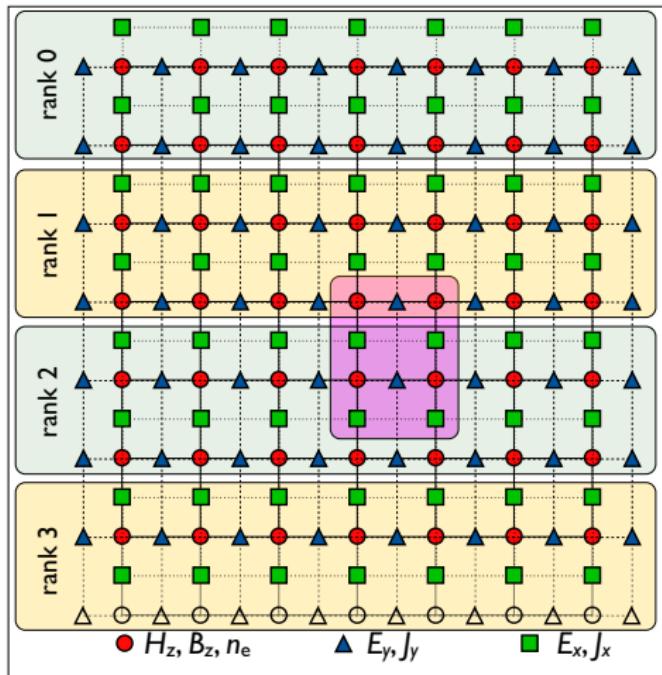


Staggered grids & localized envelopes/boundary layers

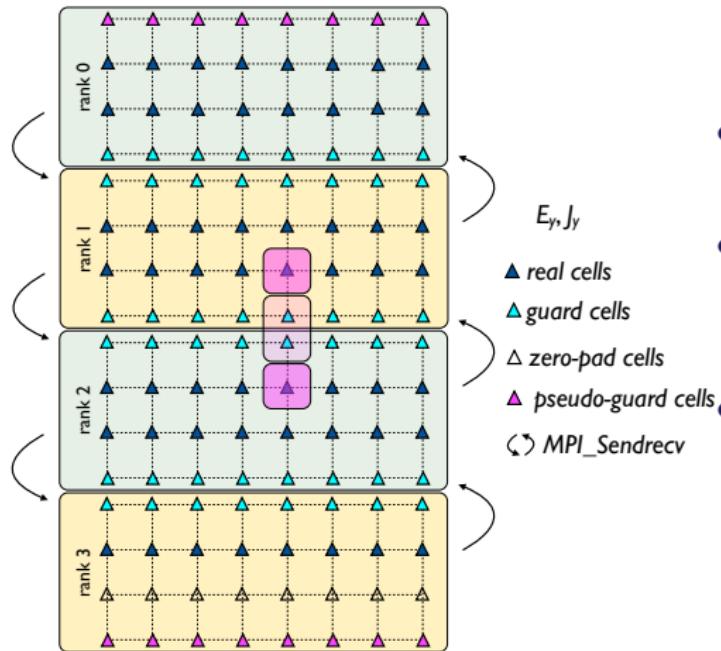


# REFMULX's grids

Staggered grids & localized envelopes/boundary layers



# MPI domain decomposition



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