Computational methods for plasma fluid equations

Guillaume Fuhr [1]

[1] P.I.I.M. Lab./CNRS
Aix-Marseille University

email: *guillaume.fuhr@univ-amu.fr*

ITER International School, 2014

Outline

Fluid Equations for plasma

Braginskii Eq.

$$\text{Continuity Eq. :} \quad \frac{\partial}{\partial t} n_s + \nabla \cdot (n_s \mathbf{V}_s) = 0 \tag{1}$$

$$\text{Eq. of motion :} \quad n_s m_s \frac{d}{dt} \mathbf{V}_s + q_s n_s (\mathbf{E} + \mathbf{V}_s \times \mathbf{B}) + \nabla p_s + \nabla \cdot \Pi_s^d = \mathbf{R}_{ei} \tag{2}$$

$$\text{Energy Eq. :} \quad \frac{3}{2} n_s \frac{d}{dt} T_s + p_s \nabla \cdot (V_s) = -\nabla \cdot \mathbf{q}_s - \Pi_s^d : \nabla \mathbf{V}_s + Q_s \tag{3}$$

with

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \mathbf{V} \cdot \nabla$$

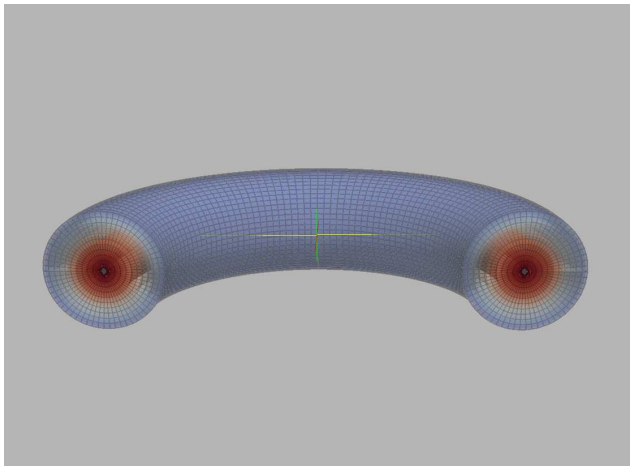
Physical ingredients

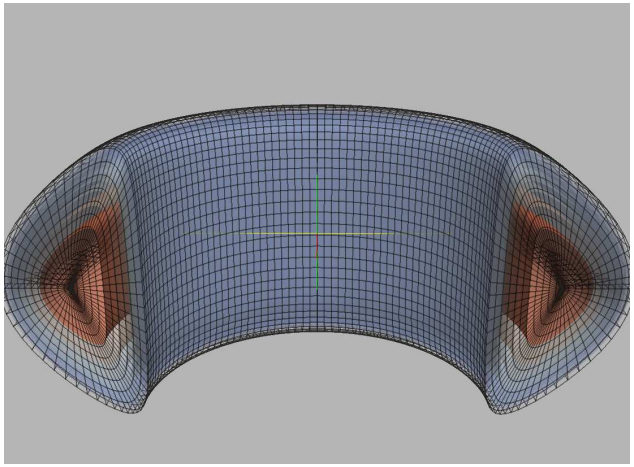
- Pressure : $p, p_e, p_i, p_e + p_i$
- Density : $n, n_e, n_i, n_e + n_i$
- Temperature : $T_e, T_i, T_e + T_i$
- Parallel velocity : $V_{\|}$
- Electrostatic potential : ϕ
- Electromagnetic potential : ψ

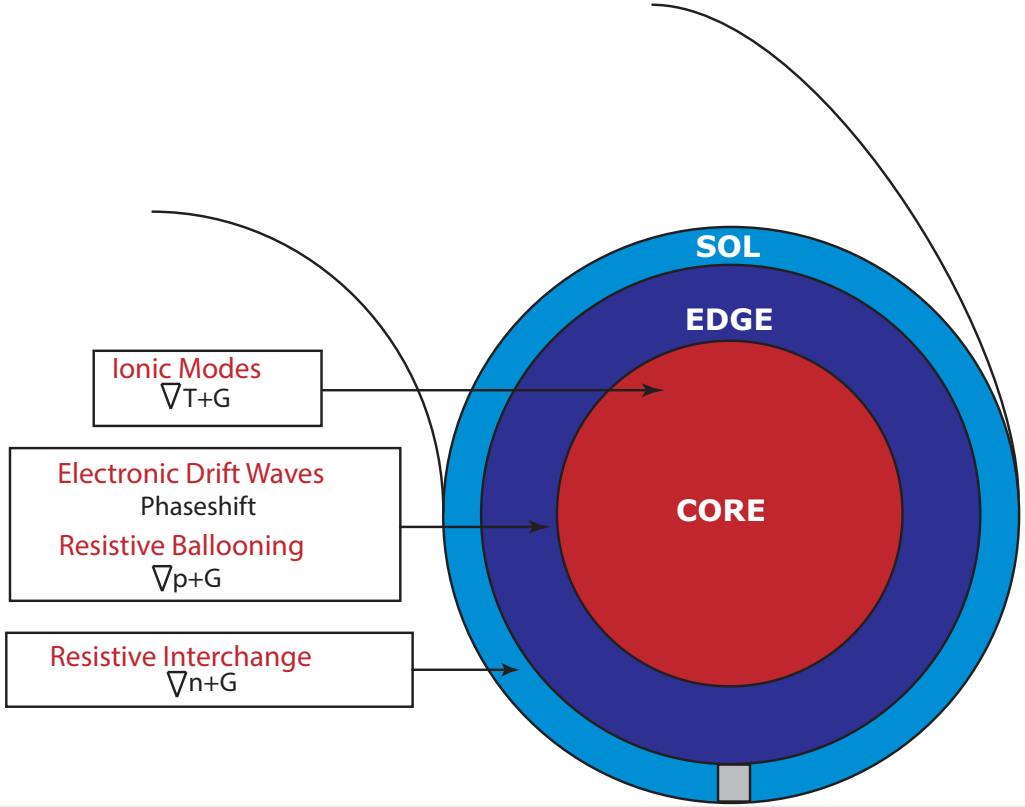
- Curvature : $K(\cdot)$
- $\mathbf{E} \times \mathbf{B}$ drift : $V_E \cdot \nabla$
- gradient $\| \mathbf{B}$: $\nabla_{\|}$
- diffusion like process : $\nabla_{\perp} [D \cdot \nabla_{\perp}]$
- diamagnetic effects
- ...

- Fluid model : from 2 to 6 fields with 2 to a lot of operators/couplings
- (too) wide range of possibilities/classifications

Physical Assumptions







- Fluid model key ingredients linked to modeled region
- Geometry determines coordinates and symmetries

Fluid equations for plasma

from a numerical point of view

$$\frac{\partial}{\partial t}\begin{bmatrix}\Omega\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix} = L\left(\begin{bmatrix}\phi\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix}\right) + NL\left(\begin{bmatrix}\phi\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix}\right) \tag{4}$$

$$\Omega = \nabla_{\perp}^2 \phi \tag{5}$$

$L(\cdots)$: advection+diffusion, linear

$$: \frac{\partial}{\partial x_i}\left[A(x_j)\cdot\right] + \frac{\partial^2}{\partial x_i^2}\left[D(x_j)\cdot\right] \text{ with } i \neq j$$

$NL(\cdots)$: Poisson brackets, non-linear

$$[f, g] = \frac{1}{r}\left(\partial_r f \partial_\theta g - \partial_r g \partial_\theta f\right)$$

Numerical discretization : finite differences

Finite difference : approximate a function through a Taylor's expansion

$$f(x_{i+1}) = f(x_i + \Delta x) \simeq f(x_i) + \Delta x \frac{\partial f(x_i)}{\partial x} + \frac{\Delta x^2}{2} \frac{\partial^2 f(x_i)}{\partial x^2} \quad (6)$$

$$f(x_{i-1}) = f(x_i - \Delta x) \simeq f(x_i) - \Delta x \frac{\partial f(x_i)}{\partial x} + \frac{\Delta x^2}{2} \frac{\partial^2 f(x_i)}{\partial x^2} \quad (7)$$

2^{nd} order differencing

$$\frac{\partial f(x_i)}{\partial x} \simeq \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x} \quad (8)$$

$$\frac{\partial^2 f(x_i)}{\partial x^2} \simeq \frac{f(x_{i+1}) + f(x_{i-1}) - 2f(x_i)}{\Delta x^2} \quad (9)$$

Fluid Equations for plasma

$$\frac{\partial}{\partial t}\begin{bmatrix}\Omega\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix} = L\left(\begin{bmatrix}\phi\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix}\right) + NL\left(\begin{bmatrix}\phi\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix}\right)$$

$$\Omega = \nabla_{\perp}^2 \phi$$

temporal discretization

Ref. : S. Jardin, *Comp. Methods in Plasma Physics*, CRC Press
J.H.Ferziger and M. Peric„ *Comp. Methods for Fluid Dynamics*, Springer

$$\frac{\partial F}{\partial t} = Rhs \Rightarrow F^{t+1} - \sum_{i=-M}^{0} \alpha_i F^{t+i} = \sum_{i=-M}^{1} \beta_i Rhs^{t+i}$$

Explicit :

$$F^{t+1} = F^t + \beta_0 Rhs^t$$

- + : implementation and cost
- + : parallel scalability
- - : C.F.L. condition $\leftrightarrow$ time step restriction
- + : advection

Implicit :

$$F^{t+1} + \beta_1 Rhs^{t+1} = F^t$$

- - : expensive
- - : parallel scalability
- + : unconditionally stable
- + : diffusion

temporal discretization

focus on explicit schemes

Runge-Kutta 4 schemes:

$$F^{t+1} = F^t + \Delta t \sum_{p=0}^{4} \beta_p k_p$$

$$k_j = Rhs(t_n + c_j \Delta t, y_i + \Delta t \sum_{q=0}^{4} \alpha_{iq} k_q)$$

- 5^{th} order scheme
- 4 evaluations of RHS per time step

Karniadakis scheme :

$$\frac{F^{t+1} - \sum_{p=0}^{2} \alpha_p F^{t-p}}{\Delta t} = \sum_{p=0}^{2} \beta_p Rhs^{t-p}$$

$$\alpha_p = 3, -3/2, 1/3$$

$$\beta_p = 3, -3, 1$$

- a variant of a third order Adams-Bashforth scheme
- increased stability and accuracy with respect to AB3 scheme
- need another method for the initial time steps

Fluid Equations for plasma

linear part

$$\frac{\partial}{\partial t}\begin{bmatrix}\Omega\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix} = L\left(\begin{bmatrix}\phi\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix}\right) + NL\left(\begin{bmatrix}\phi\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix}\right)$$

$$\Omega = \nabla_{\perp}^2 \phi$$

Numerical method for diffusion

- diffusion equation :

$$\partial_t F(r,t) = \nabla_\perp \left[D(r) \cdot \nabla_\perp \right] F(r,t)$$

- finite difference discretization in 1D

$$\begin{aligned}\left[D(r) \cdot \nabla_\perp \right] F(r,t) &\simeq \partial_r D_i \frac{F_{i+1} - F_{i-1}}{2\Delta r} + D_i \frac{F_{i+1} + F_{i-1} - 2F_i}{\Delta r^2} \\ &\simeq \left(\frac{D_i}{\Delta r^2} + \frac{\partial_r D_i}{2\Delta r}\right) F_{i+1} - 2\frac{D_i}{\Delta r^2} F_i + \left(\frac{D_i}{\Delta r^2} - \frac{\partial_r D_i}{2\Delta r}\right) F_{i-1}\end{aligned}$$

- stability condition (C.F.L. condition) : $2max(D(r))\Delta t \leq \Delta r^2$

$$n_{CFL} = max(D(r))\frac{\Delta t}{\Delta r^2} \leq \frac{1}{2}$$

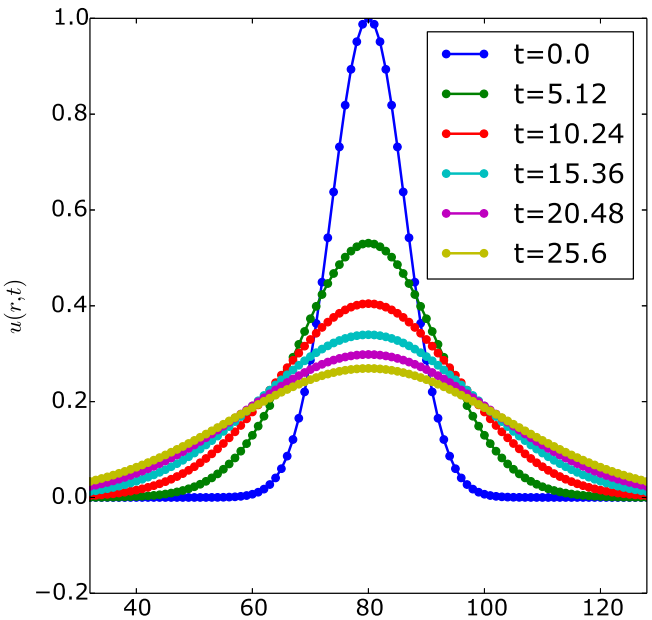
- stability criteria slightly modified with more advanced explicit time schemes

Diffusion and C.F.L.

Test Case

$$\partial_t u(r,t) = D\partial_r^2 u(r,t), \qquad u(r,t=0) = e^{-(r-r_0)^2/\sigma^2} \tag{10}$$

$$n_{CFL} = 0.1$$

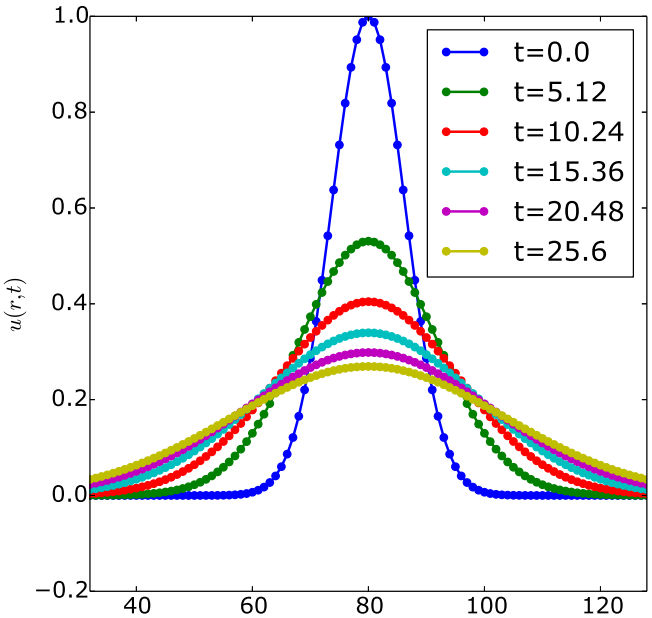


Diffusion and C.F.L.

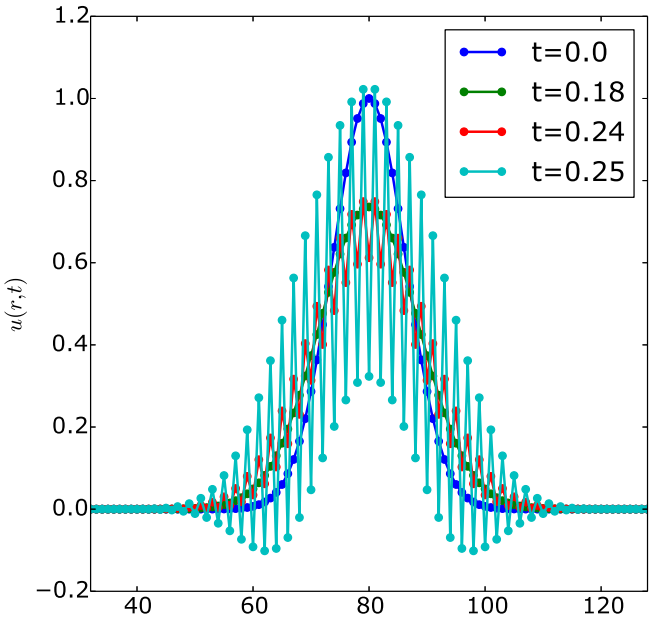
Test Case

$$\partial_t u(r,t) = D\partial_r^2 u(r,t), \qquad u(r,t=0) = e^{-(r-r_0)^2/\sigma^2} \tag{10}$$

$n_{CFL} = 0.1$



$n_{CFL} = 1.0$



Amdhal's Law

Relation between numerical resources and kind of problems/algorithms used

$$S(n) = \frac{1}{B + \frac{1}{n}(1 - B)}$$

n : cores, B : serial part, S : Speed Up

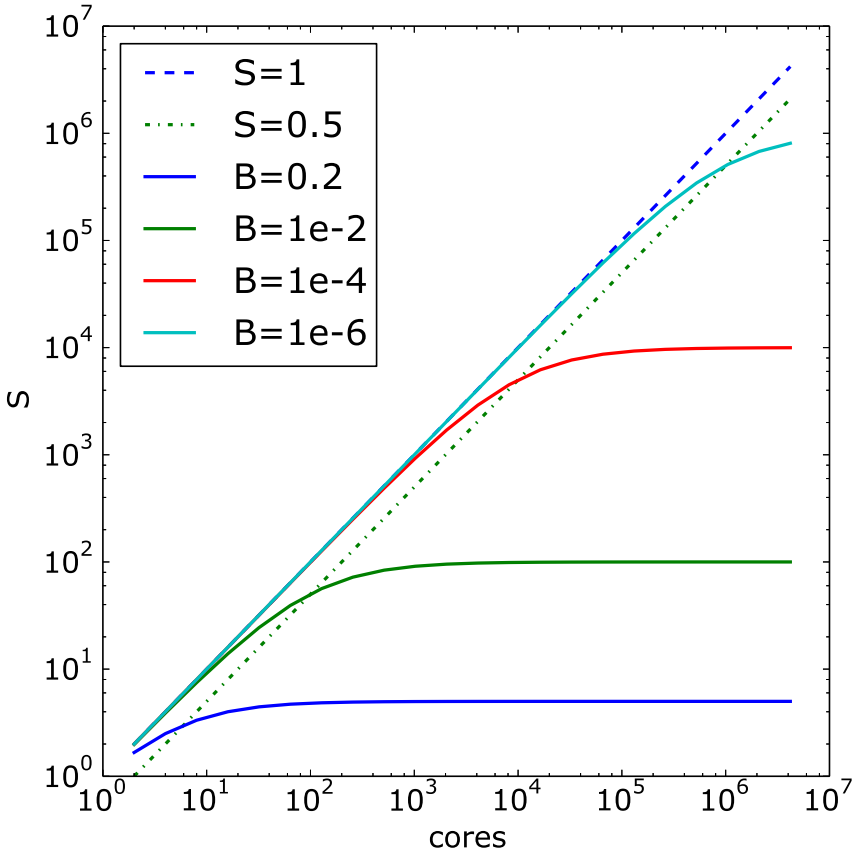
Serial part better defined as "part cannot be run fully in parallel" and include :

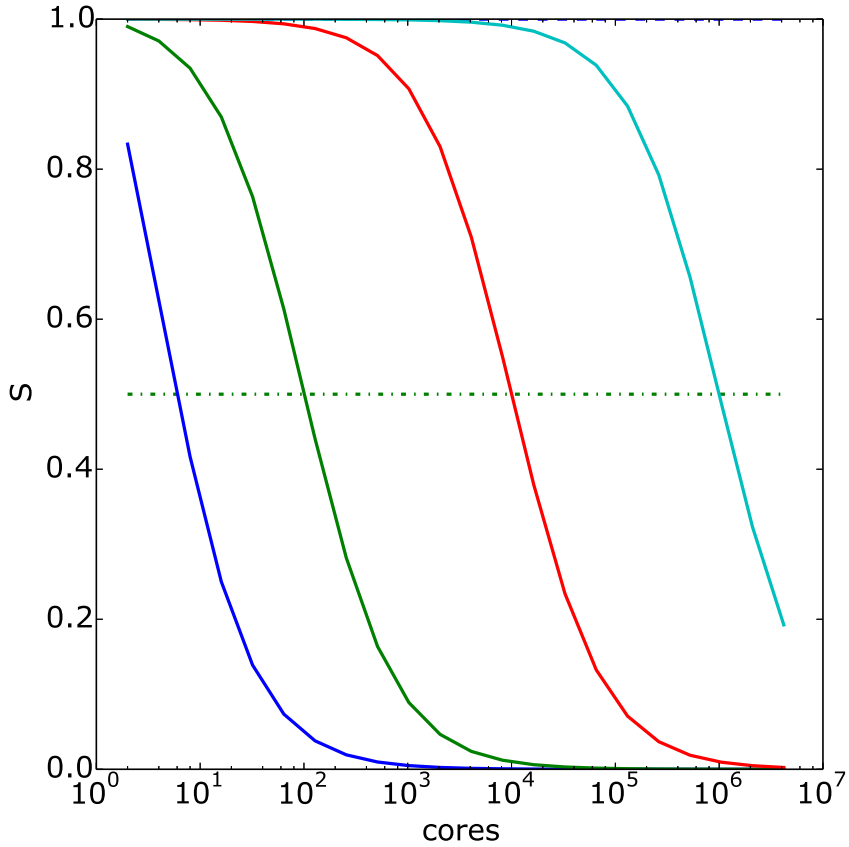
- communications between cores (how much and how often)
- dependances in algorithms
  Ex: $u_r = L_1(u_{r+1})$

Amdhal's Law, influence of B

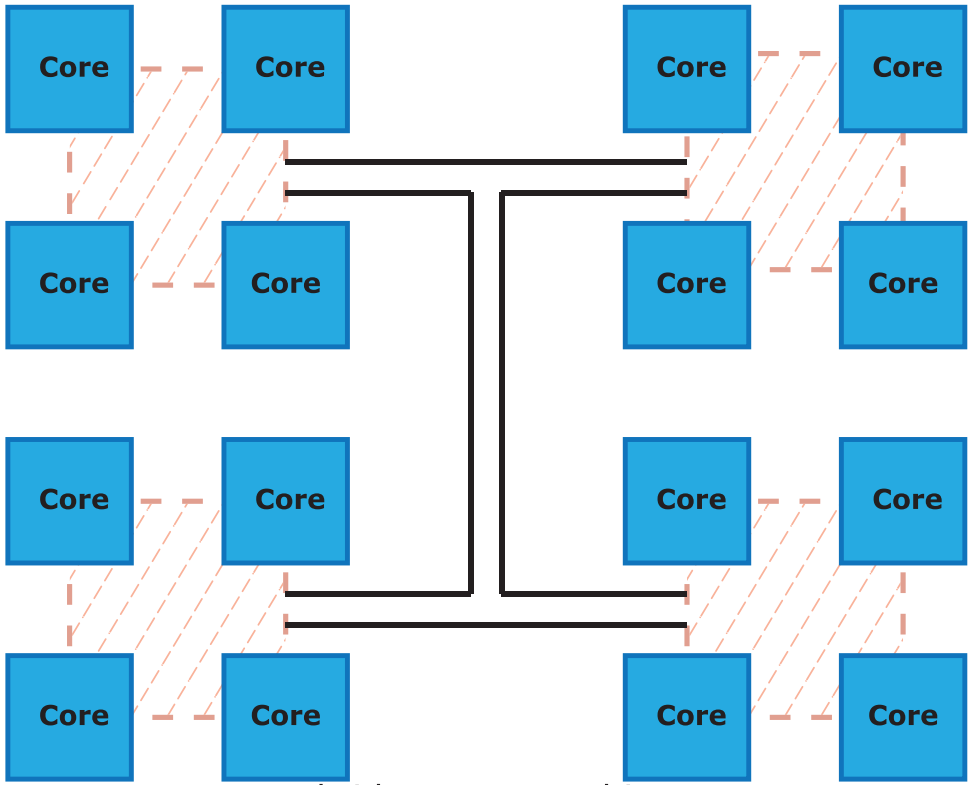
Scaling of $S(n)$ for different values of B

$$S(n) = \frac{1}{B + \frac{1}{n}(1 - B)}$$





Hybrid Architectures



Hybrid Memory Architecture

- Shared memory (S.M.A.) inside nodes and distributed memory between nodes
- Necessary to use advantages of S.M.A. for HPC developments

Graining
Fine or Coarse

- Parallelism : doing tasks in parallel :)
- Algorithms/Models can be classified according to how often subtasks need to synchronize/communicate

- Coarse graining : domain decomposition
- Fine graining : idea of microtasking
  Ex : divide iterations of a loop

Diffusion, from serial to parallel

- Time advance scheme : R.K.4
- 2^{nd} order F.D.
- How to parallelize this algorithm efficiently?
- Two approach : fine and coarse graining with OpenMP

Serial code

```
Coef = D/(dr*dr);
for(i=1; i<Nx-1; i=i+1)
   laplacian[i] = Coef*(un[i+1]+un[i-1]-2*un[i])
```

Fine Graining

OpenMP fine graining

- add *#pragma omp for* in C/C++ or !$*OMP DO* in Fortran before loops

Implementation

```
Coef = D/(dr*dr);
#pragma omp parallel for
for(i=1; i<Nx-1; i= i+1)
   laplacian[i] = Coef*(un[i+1]+un[i-1]-2*un[i])
```

Fine Graining

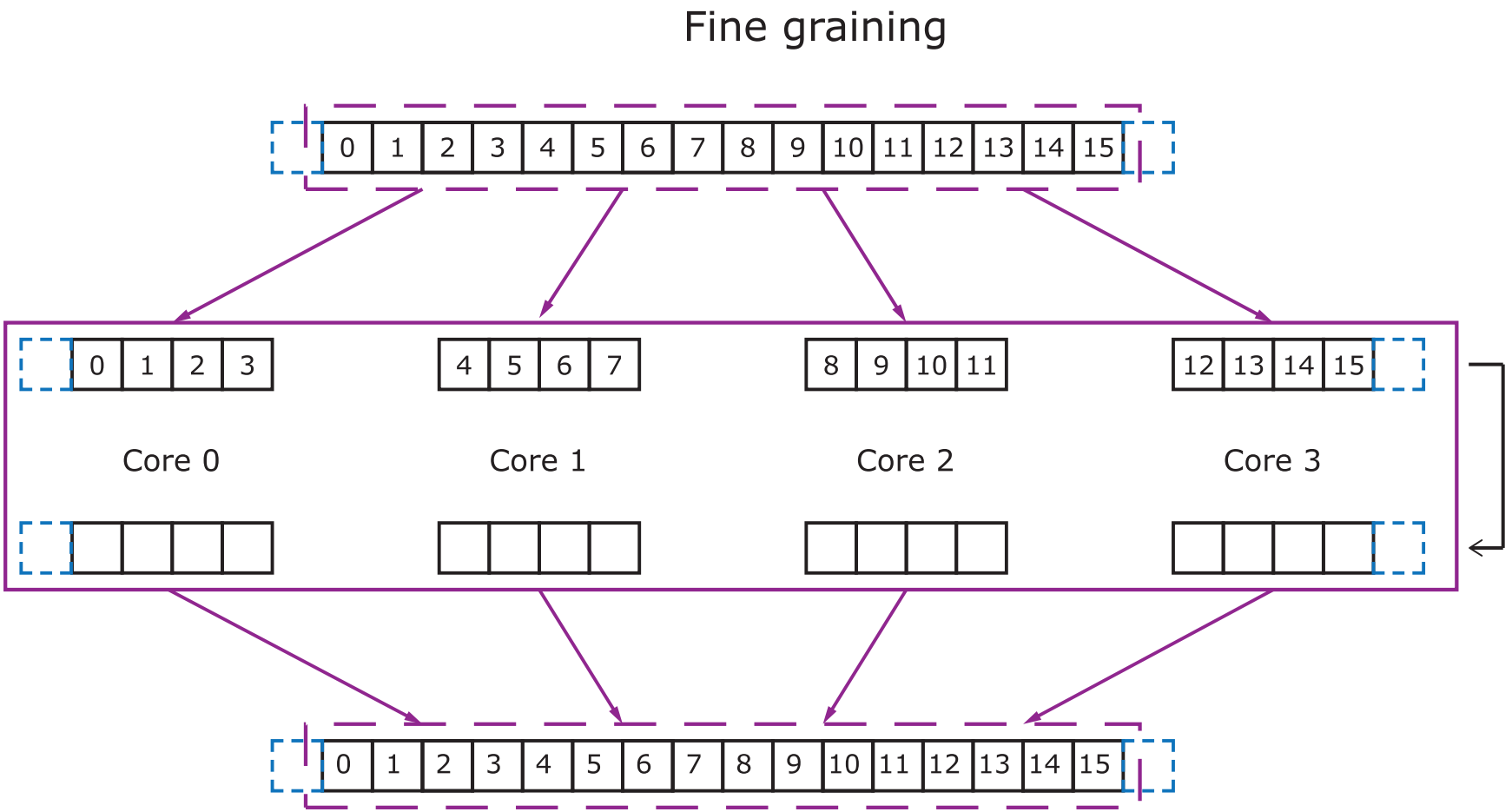
OpenMP fine graining

- add *#pragma omp for* in C/C++ or !$*OMP DO* in Fortran before loops

Implementation

```
Coef = D/(dr*dr);
!$OMP PARALLEL DO
do i=2,Nx-1
   laplacian(i) = Coef*(un(i+1)+un(i-1)-2*un(i))
end do
!$ OMP END PARALLEL DO
```

Fine Graining



Diffusion : Fine Graining acceleration

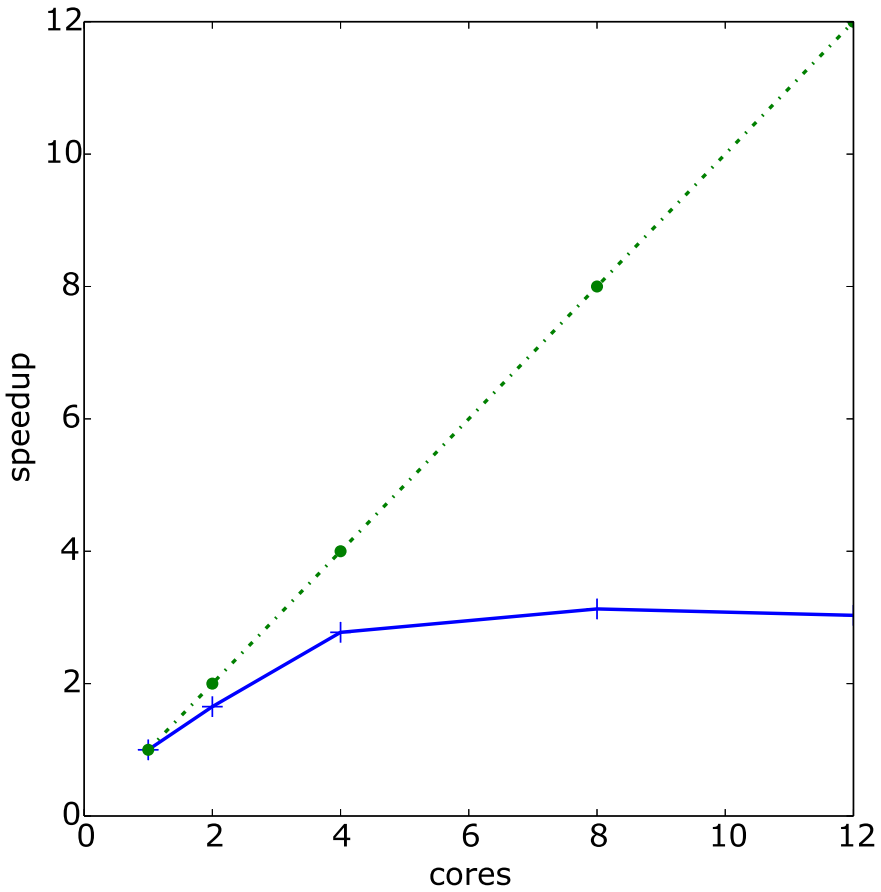
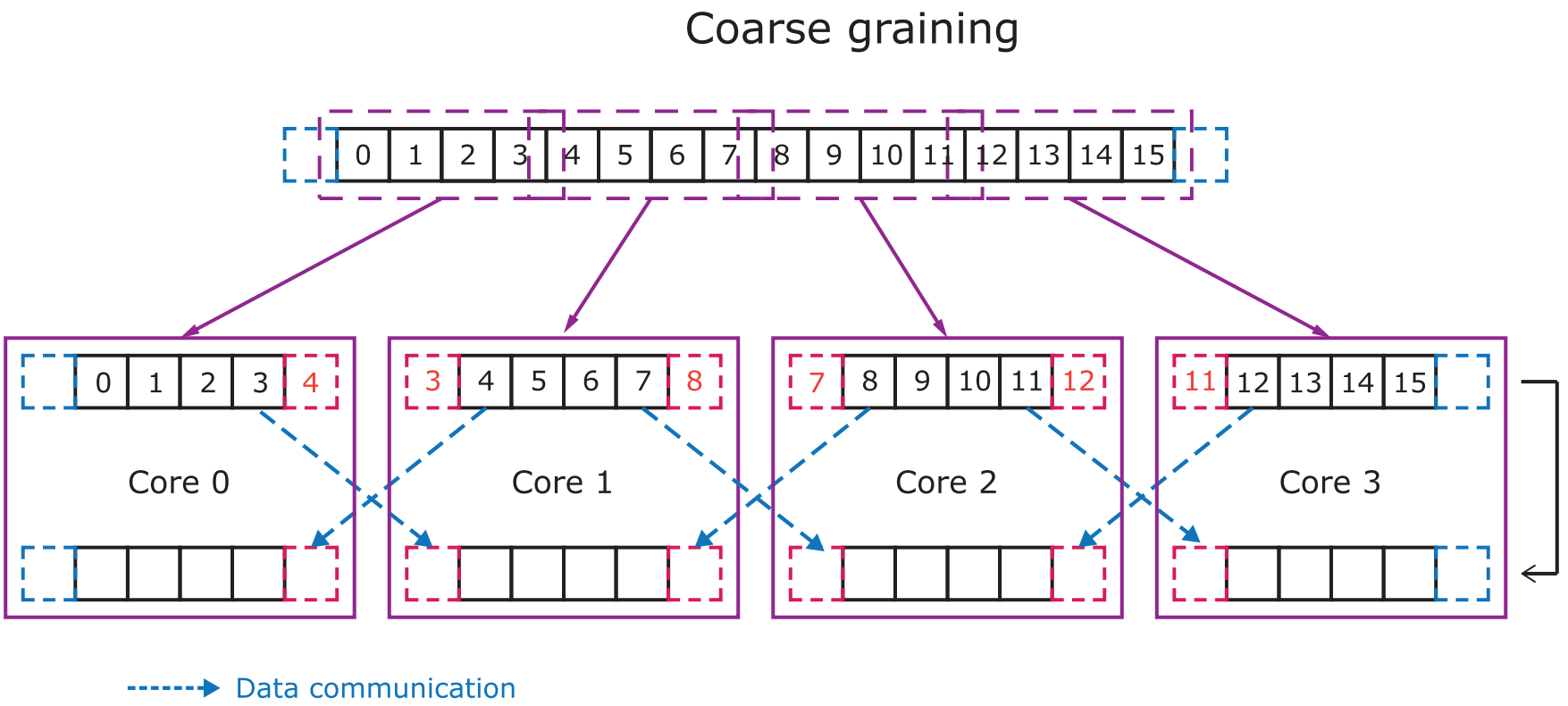


Figure: Acceleration, case $Nx = 8096$

- Not very effective
- Reasons : not enough work/task
- synchronization and threads creations/destructions too expensive in comparison

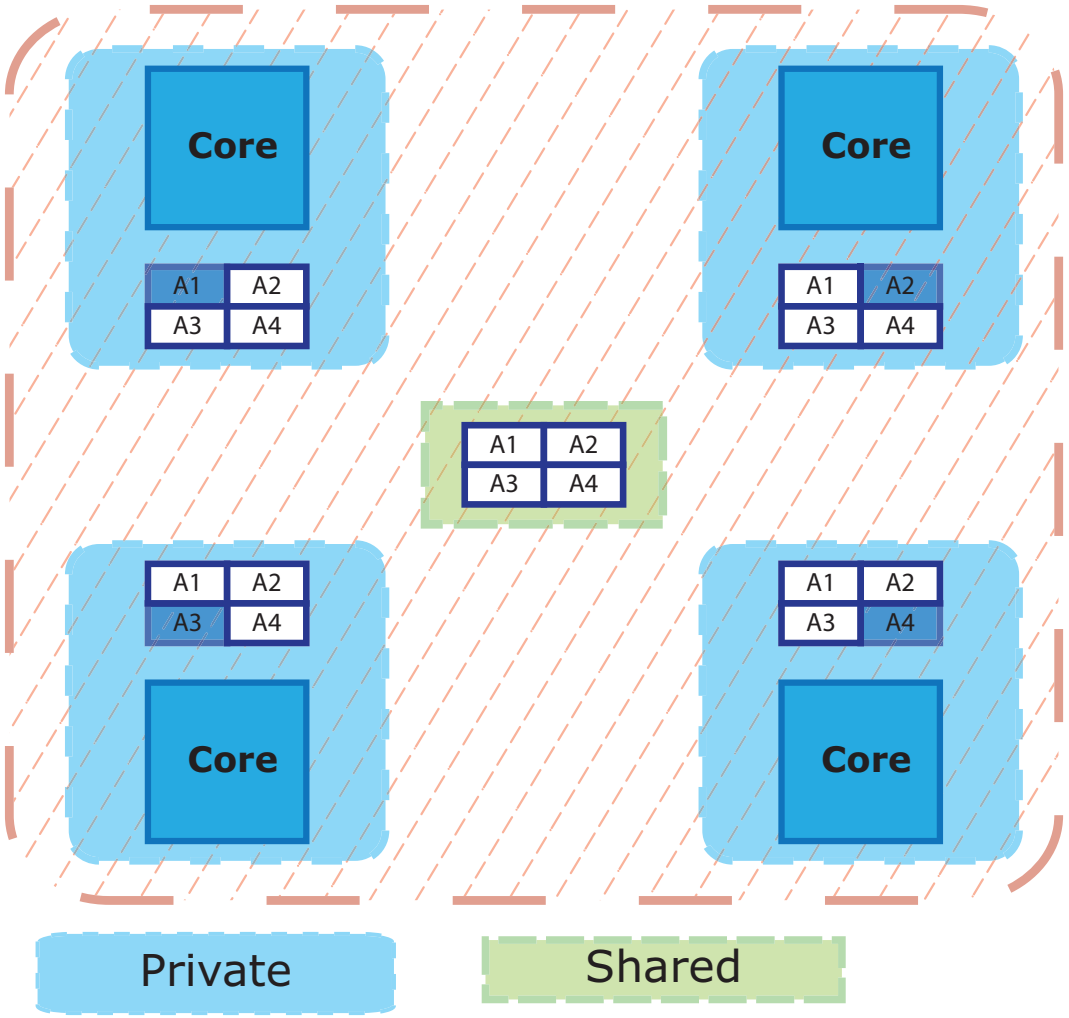
Coarse Graining



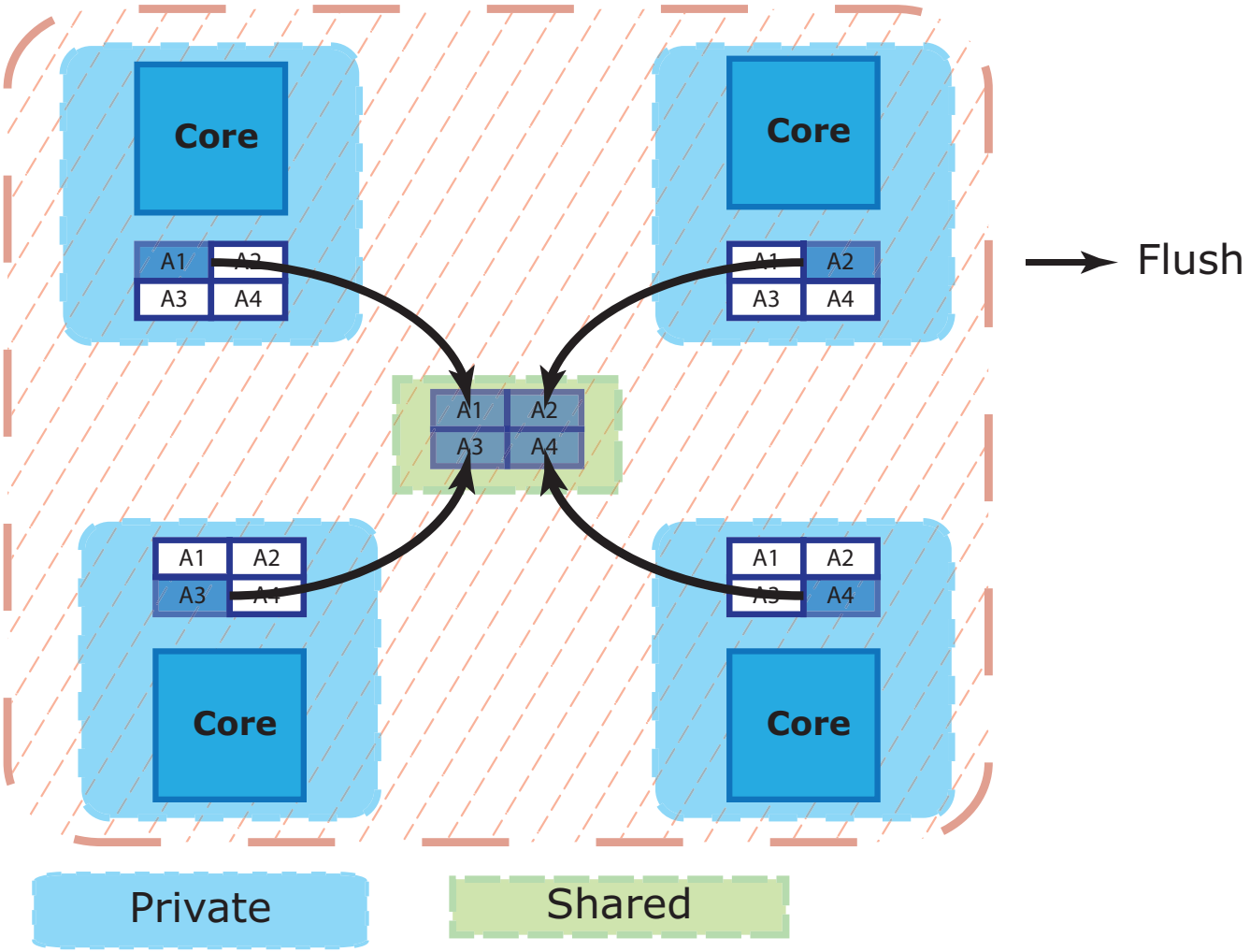
Coarse Graining in OpenMP

- How to implement Coarse Graining in OpenMP
- OpenMP : each variable can be defined as shared between threads or private to a thread
- like in MPI, each thread contains only a part of the domain (private arrays)
- Problem : no send/receive directives in OpenMP
- Solution : use an intermediate shared array between threads to communicate data : update values and synchronize threads (through flush and barrier directives)

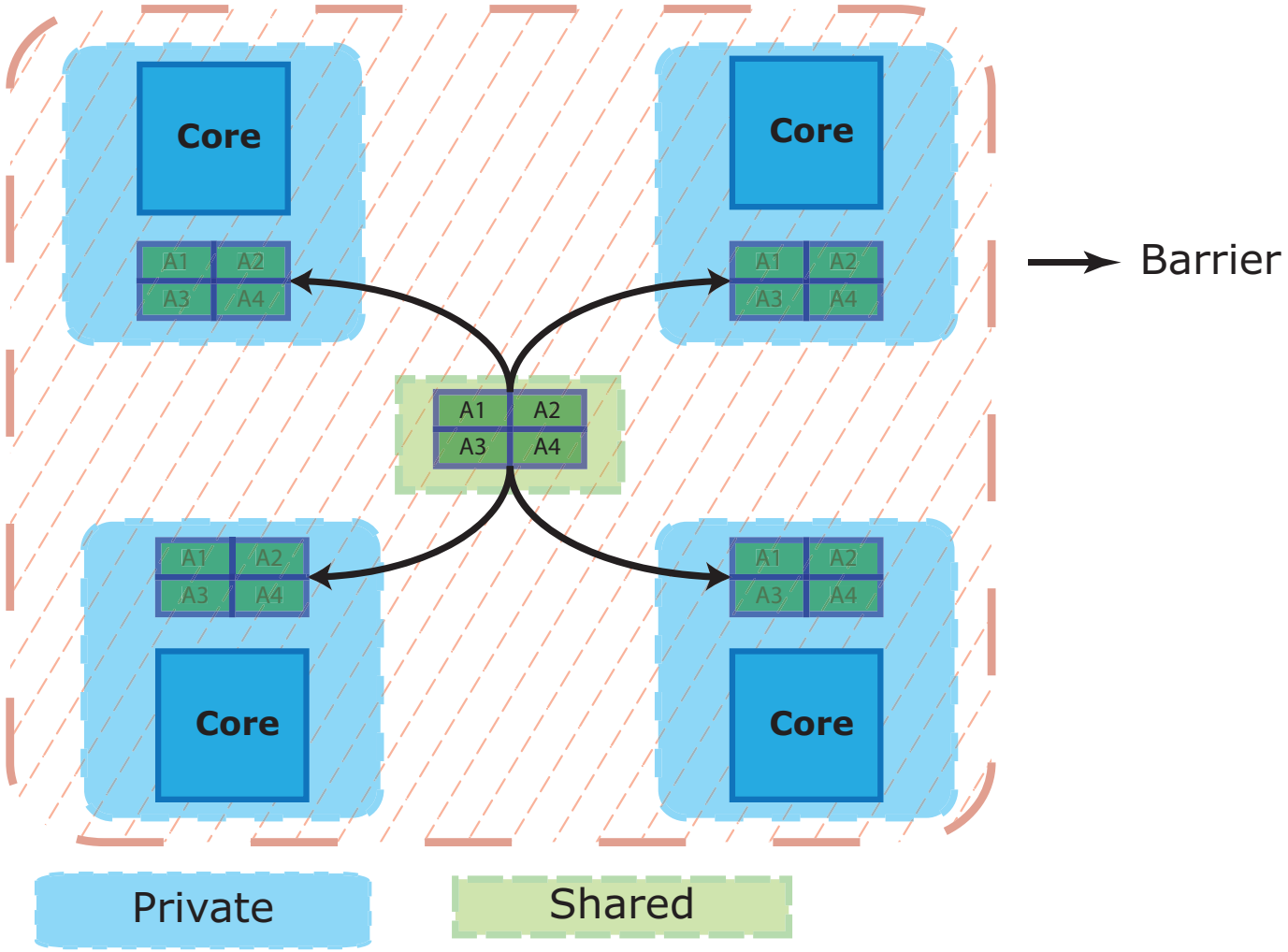
Coarse Graining in OpenMP



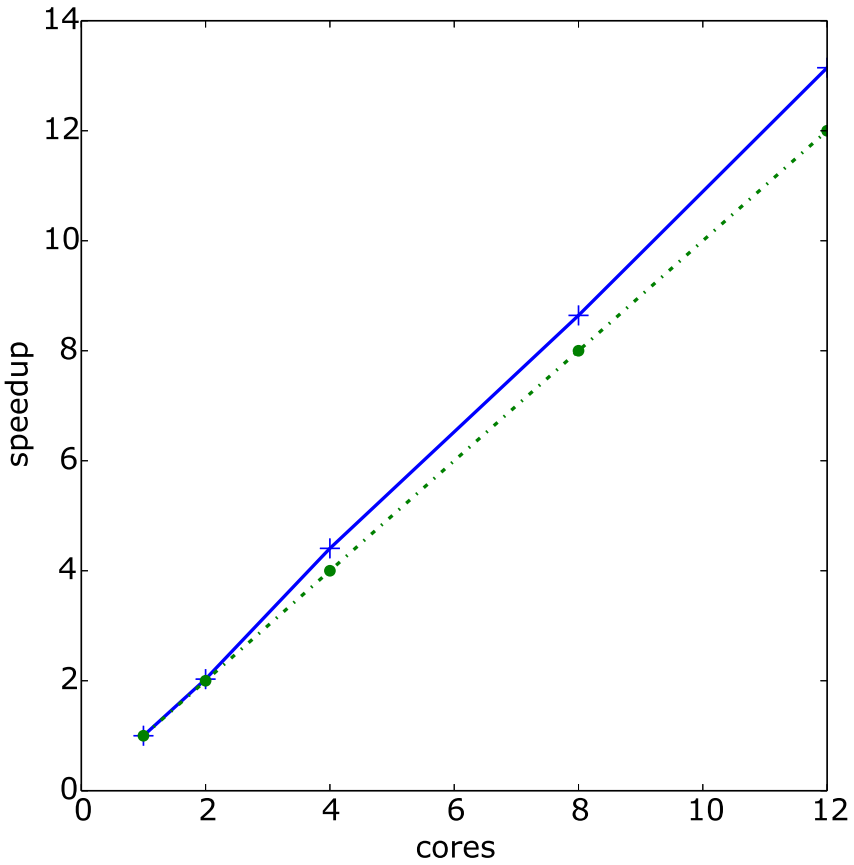
Coarse Graining in OpenMP



Coarse Graining in OpenMP



Diffusion : numerical resolution / graining



- same resolution as fine graining case (Nx=8096)
- As in previous case, grid size will influence scaling
- "Over speed up" linked to low level optimization by compiler
- Cons : implementation more difficult

Advection

- $1D$ advection equation :

$$\partial_t u(r,t) + \partial_r [f(u)] = 0$$

- If $f(u) = V * u$, analytical solution expressed as:

$$u(r,t) = u_0(r - V * t)$$

- using same discretization as in diffusive case

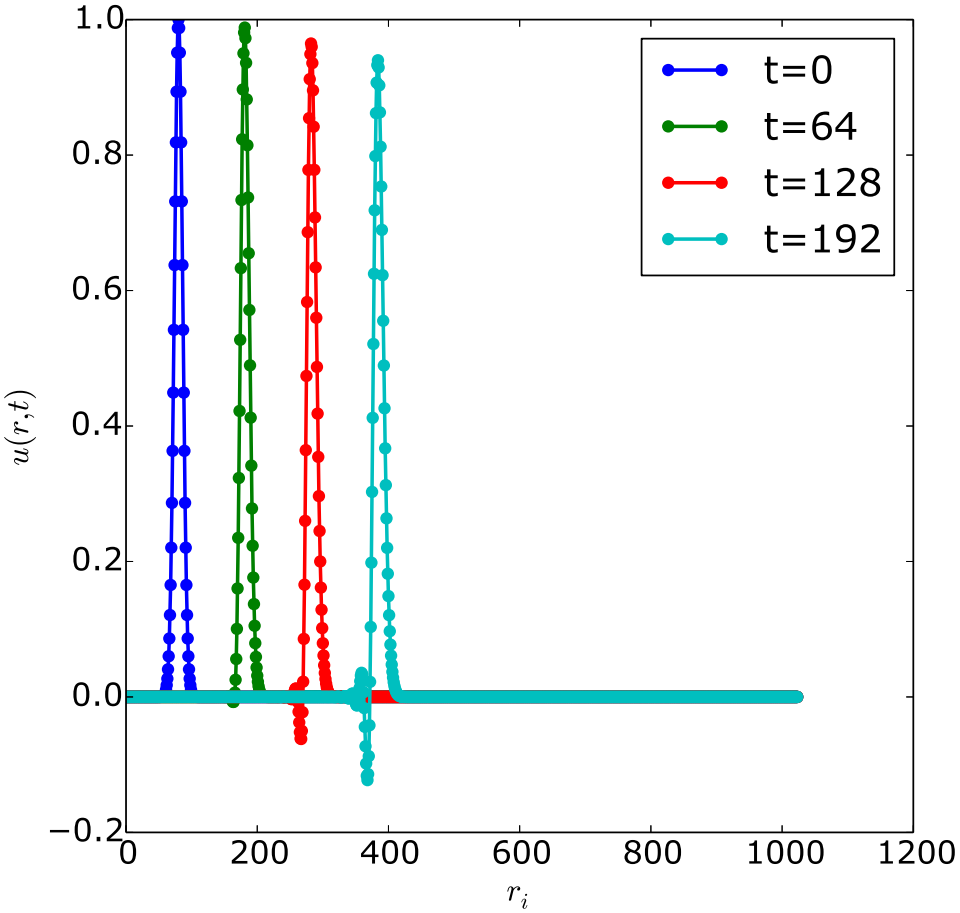
$$\partial_t u(r,t) = -V \frac{u_{i+1} - u_{i-1}}{2\Delta r} \tag{11}$$

- associated C.F.L. number :

$$\begin{cases} \text{explicit Euler :} & \text{unconditionally unstable} \\ \text{R.K.4 :} & n_{CFL} = V\Delta t/\Delta r \leq 1 \end{cases} \tag{12}$$

Advection simulation

- Numerical results using $u_0(r) = exp(-r^2/\sigma^2)$
- C.F.L : $n_{CFL} = 0.4$

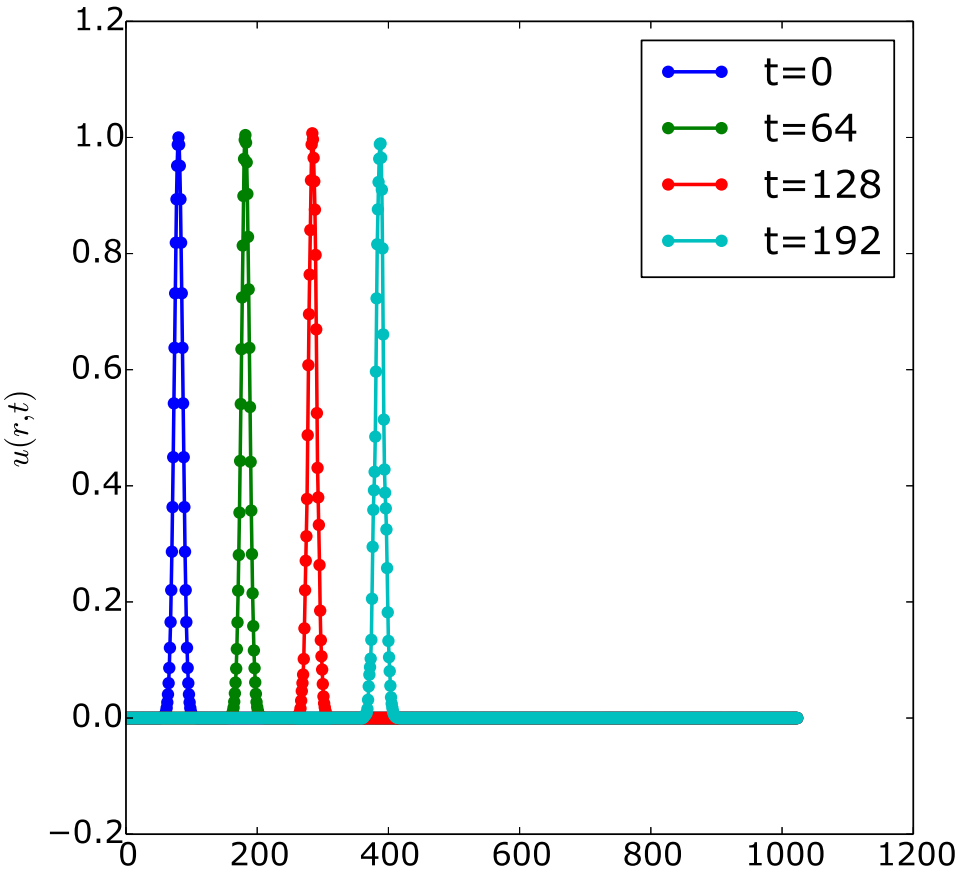


- system not conservative (diffusion)
- numerical oscillations generated
- phenomena disappears when $\Delta r \to 0$

Advection simulation

Ref. : C. S. Shu

- Numerical results using $u_0(r) = exp(-r^2/\sigma^2)$
- C.F.L : $n_{CFL} = 0.4$



- 5^{th} order WENO scheme used here
- smooth interpolation locally around discontinuities

$$\partial_t u(r,t) = -\frac{1}{\Delta r}\left(\hat{f}_{r+1/2} - \hat{f}_{r-1/2}\right)$$

$$\hat{f}_{r+1/2} = \hat{f}(u_{r-2}, \cdots, u_{r+2})$$

- Pro: parallel implementation

Fluid Equations for plasma

non-linear part

$$\frac{\partial}{\partial t}\begin{bmatrix}\Omega\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix} = L\left(\begin{bmatrix}\phi\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix}\right) + NL\left(\begin{bmatrix}\phi\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix}\right)$$

$$\Omega = \nabla_{\perp}^2 \phi$$

Numerical Methods for Poisson brackets

Ref. : A. Arakawa, *J. Comp. Phys.* 1, 119 (1966), repr. vol 135 (1997) 103

- Non-linear terms represented by Jacobian operator :

$$[f, g]_{x,y} = \partial_x f \partial_y g - \partial_y f \partial_x g$$

- using fin. diff. scheme,

$$\partial_x f \to \frac{f_{i+1,j} - f_{i-1,j}}{2\Delta x}, \quad \partial_y f \to \frac{f_{i,j+1} - f_{i,j-1}}{2\Delta y}$$

$$[f, g]_{x,y} = \frac{(f_{i+1,j} - f_{i-1,j})(g_{i,j+1} - g_{i,j-1})}{4\Delta x \Delta y} - \frac{(f_{i,j+1} - f_{i,j-1})(g_{i+1,j} - g_{i-1,j})}{4\Delta x \Delta y} \tag{13}$$

- energy conservation not verified,

$$(13) \simeq \partial_x f \partial_y g - \partial_y f \partial_x g + \frac{1}{2\Delta x \Delta y} \frac{\partial^2 g}{\partial x \partial y} \left[\left(\frac{\partial f}{\partial x} \right)^2 - \left(\frac{\partial f}{\partial y} \right)^2 \right]$$

Arakawa scheme

Ref. : A. Arakawa, *J. Comp. Phys.* 1, 119 (1966), repr. vol 135 (1997) 103
M. Kuhn *et al.*, *SYNASC*, 15th Int. S.Y.N.A.S.C, (2013)

- energy conservative scheme, simplified version :

$$
\begin{aligned}
J_{i,j}(\xi,\psi) = -\frac{1}{4\Delta x\Delta y}\Big[&\left(\psi_{i,j-1}+\psi_{i+1,j-1}-\psi_{i,j+1}-\psi_{i+1,j+1}\right)\xi_{i+1,j}\\
&-\left(\psi_{i-1,j-1}+\psi_{i,j-1}-\psi_{i-1,j+1}-\psi_{i,j+1}\right)\xi_{i-1,j}\\
&+\left(\psi_{i+1,j}+\psi_{i+1,j+1}-\psi_{i-1,j}-\psi_{i-1,j+1}\right)\xi_{i,j+1}\\
&-\left(\psi_{i+1,j-1}+\psi_{i+1,j}-\psi_{i-1,j-1}-\psi_{i-1,j}\right)\xi_{i,j-1}\\
&+\left(\psi_{i+1,j}-\psi_{i,j+1}\right)\xi_{i+1,j+1}-\left(\psi_{i,j-1}-\psi_{i-1,j}\right)\xi_{i-1,j-1}\\
&+\left(\psi_{i,j+1}-\psi_{i-1,j}\right)\xi_{i-1,j+1}-\left(\psi_{i+1,j}-\psi_{i,j-1}\right)\xi_{i+1,j-1}\Big]
\end{aligned}
$$

- speedup of this version $\simeq$ 15%
- optimized memory management, speedup $\simeq$ 75%

Fluid Equations for plasma

Poisson equation

$$\frac{\partial}{\partial t}\begin{bmatrix}\Omega\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix} = L\left(\begin{bmatrix}\phi\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix}\right) + NL\left(\begin{bmatrix}\phi\\ p_e\\ \psi\\ v_{\parallel}\\ \cdots\end{bmatrix}\right) \tag{14}$$

$$\Omega = \nabla_{\perp}^2 \phi \tag{15}$$

The last but not the least...

Poisson equation

reference : Y. Zhang and R. W. Hockney

- Obtain electrostatic potential from vorticity

$$\Omega = \nabla_{\perp}^{2} \phi$$

- 2^{nd} order C.D., laplacian expressed as a tridiagonal system :

$$\begin{bmatrix} b_0 & c_0 & 0 & & \cdots & 0 \\ a_1 & b_1 & c_1 & 0 & \cdots & 0 \\ 0 & a_2 & b_2 & c_2 & \cdots & 0 \\ & & \ddots & \ddots & \ddots & \\ 0 & & & \ddots & \ddots & 0 \\ 0 & \cdots & & a_{N-1} & b_{N-1} & c_{N-1} \\ 0 & \cdots & & 0 & a_N & b_N \end{bmatrix} \begin{bmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \\ \vdots \\ \vdots \\ \phi_{N-1} \\ \phi_N \end{bmatrix} = \begin{bmatrix} \Omega_0 \\ \Omega_1 \\ \Omega_2 \\ \vdots \\ \vdots \\ \Omega_{N-1} \\ \Omega_N \end{bmatrix}$$

- Typical algorithm, Thomas algorithm
  - simplified Gauss elimination
  - algorithm in 2 steps, backward and forward substitution

Thomas algorithm

Thomas algorithm

- Forward step

$$c_1^* = c_1/b_1, \quad \Omega_1^* = \Omega_1/b_1$$

$$c_i^* = \frac{c_i}{b_i - a_i c_{i-1}^*}, \quad \Omega_i^* = \frac{\Omega_i - a_i \Omega_{i-1}^*}{b_i - a_i c_{i-1}^*}$$

- backward step

$$\phi_i = \Omega_i^* - c_i^* \phi_{i+1}$$

Pro :

- fast algorithm
- low complexity : $O(N)$
- only $8n$ operations

Cons :
dependencies $\rightarrow$ no parallelization possible

Parallel cyclic reduction

- reduces a problem of N equations with N unknowns to N/2 systems of 2 equations with 2 unknowns

$$E_i^{(s)} : a_{i-2^{s-1}}^{(s)} \phi_{i-2^{s-1}} + b_i^{(s)} \phi_i + c_{i+2^{s-1}}^{(s)} \phi_{i+2^{s-1}} = \Omega_i^{(s)} \qquad (16)$$

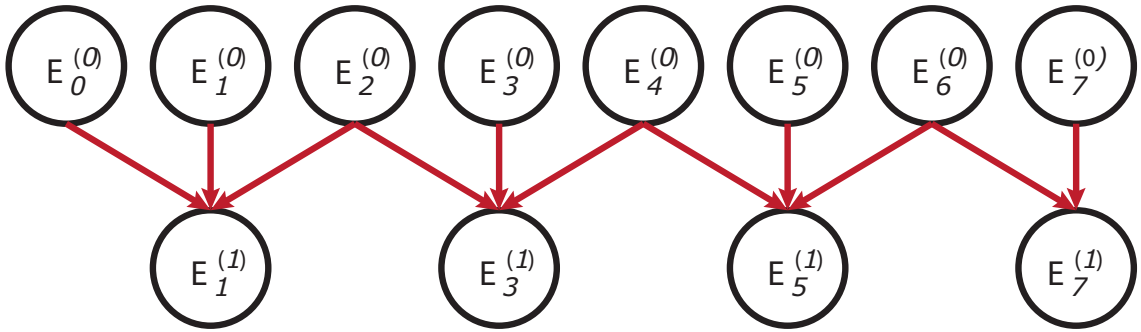
$$k_1 = a_{i-2^{s-1}}^{(s-1)} / b_{i-2^{s-1}}^{(s-1)}, \quad k_2 = c_{i+2^{s-1}}^{(s-1)} / b_{i+2^{s-1}}^{(s-1)}$$

$$a_i^{(s)} = -a_i^{(s-1)} * k_1, \quad b_i^{(s)} = b_i^{(s-1)} - k_1 c_{i-2^{s-1}}^{(s-1)} - k_2 a_{i+2^{s-1}}^{(s-1)}$$

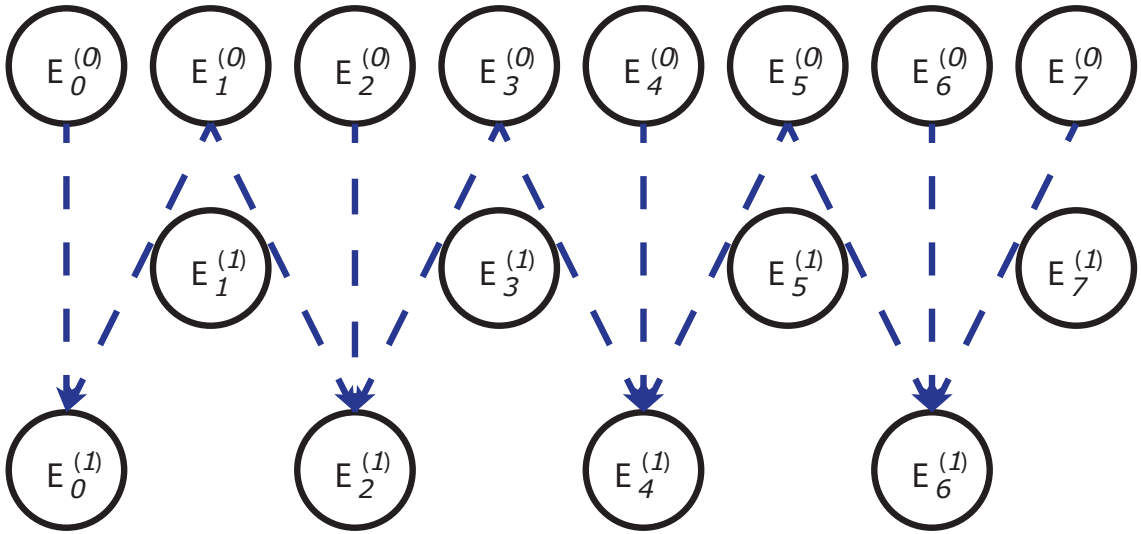
$$c_i^{(s)} = -c_i^{(s-1)} * k_2, \quad \Omega_i^{(s)} = \Omega_i^{(s-1)} - k_1 \Omega_{i-2^{s-1}}^{(s-1)} - k_2 \Omega_{i+2^{s-1}}^{(s-1)}$$

- $12 N log_2(N)$ operations
- cons : N must be a power of 2

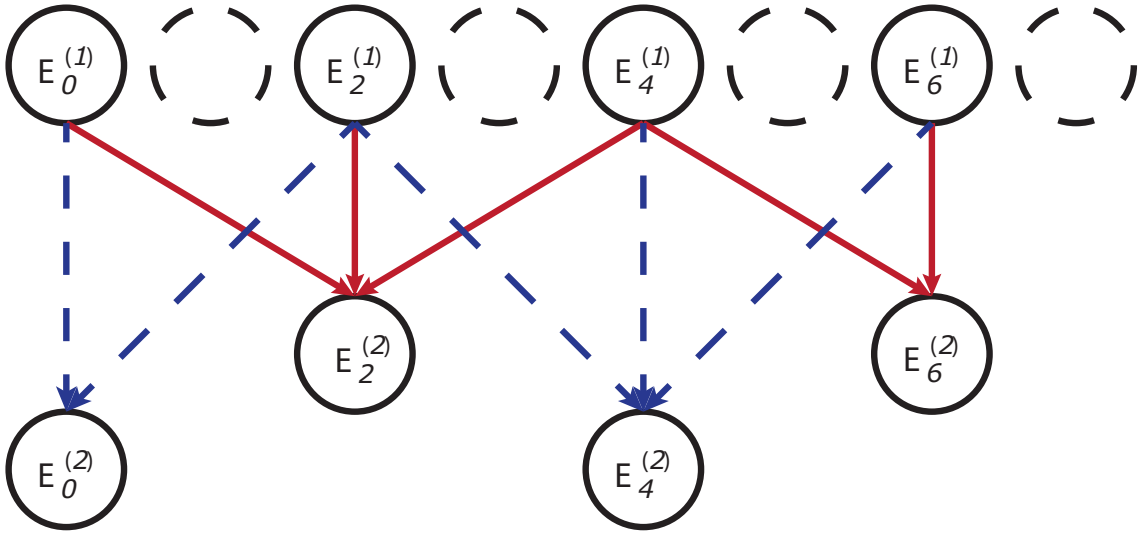
Parallel cyclic reduction



Parallel cyclic reduction



Parallel cyclic reduction



4 systems to solve
($i = 0, 1, 2, 3$)

$$b_i^{s_F}\phi_i + c_i^{s_F}\phi_{i+4} = \Omega_i^{s_F}$$
$$a_{i+4}^{s_F}\phi_i + b_{i+4}^{s_F}\phi_{i+4} = \Omega_{i+4}^{s_F}$$

advantage : final 2×2 system (and all the reductions) can be solved in parallel

Conclusion

- Physical goal will determine mainly the schemes used (stability analysis, turbulence,...)
- Choose wisely plasma model used for the physics but also the associated algorithms.
- Don't forget that parallels algorithms efficiency/scalability depend strongly on data size and load balancing
- Do it yourself, using specialized libraries (PETSC (with SLEPc), FFTW, LAPACK, MKL, SuperLU,...)
- Avoid 1^{st} order schemes

References

- S. Jardin, *Computational Methods in Plasma Physics*, CRC Press
- J.H.Ferziger and M. Peric„ *Computational Methods for Fluid Dynamics*, Springer
- G. E Karniadakis *et al.*, *J. Comp. Phys.* 97, 414 (1991)
- C.-W. Shu, , *Essentially Non-Oscillatory and Weighted Essentially Non-Oscillatory Schemes for Hyperbolic Conservation Laws*, NASA/CR-97-206253, ICASE Report No. 97-65, http://library-dspace.larc.nasa.gov/dspace/jsp/handle/2002/14650
- A. Arakawa, *J. Comp. Phys.* 1, 119 (1966), repr. vol 135 (1997) 103.
- M. Kuhn *et al.*, *SYNASC*, 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 503 - 510 (2013).

More References

- R. W. Hockney and C. R. Jesshope. *Parallel Computers*. Adam Hilger, Bristol, 1981
- Y. Zhang *et al.*, *ACM Sigplan Not.* **45**, 127 (2010)
- G.S. Patterson and S.A. Orszag, *Phys. Fluids* **14**, 2538 (1971)
- W.D. D'haeseleer,*Flux coordinates and magnetic field structure*, Springer
- W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes 3rd Ed.* (Cambridge University Press, Cambridge, 2007).
- Open MP specifications, http://openmp.org/wp/openmp-specifications/
- tutorials on HPC

  `https://computing.llnl.gov/?set=training&page=index#training_materials`