Fusione termonucleare controllata e High Performance Computing

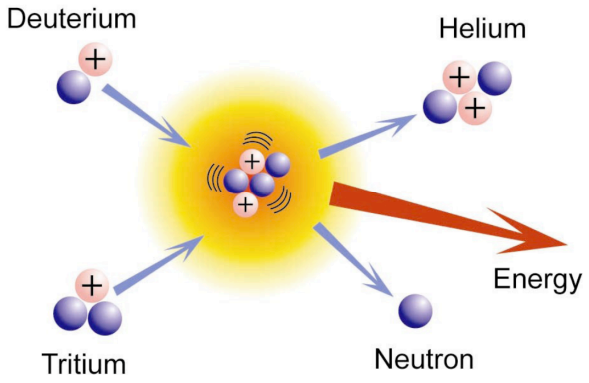
S. Briguglio, G. Fogaccia e G. Vlad
ENEA Frascati

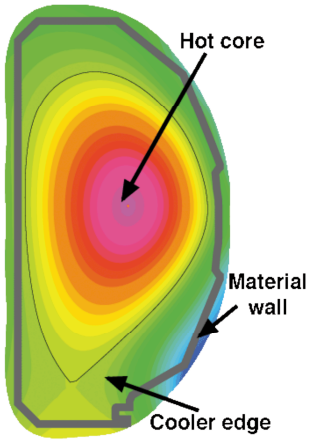
Sommario

- La fusione nucleare
- La simulazione *particle in cell* (PIC)
- Il porting di un codice PIC su architetture parallele
- Il dimensionamento dell'architettura di calcolo

La fusione nucleare

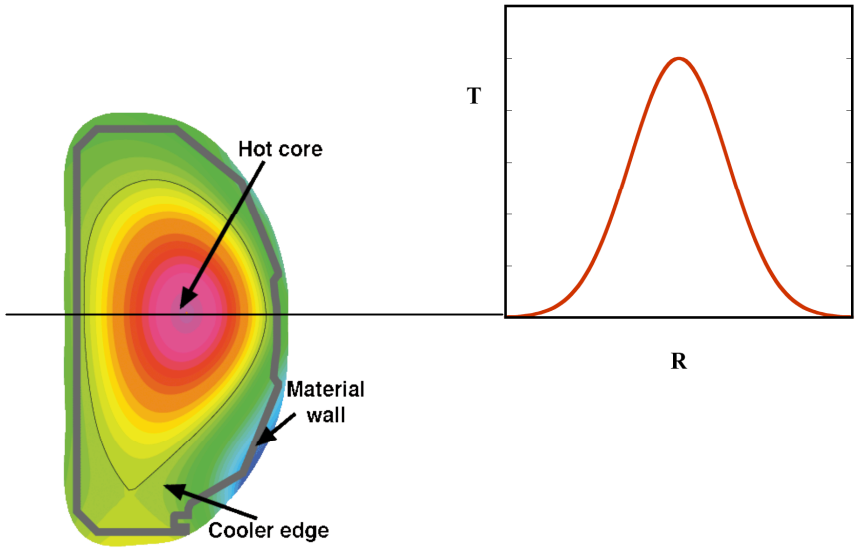
- Tappe verso il reattore:
  - riscaldare un gas (plasma) di deuterio-trizio (corrente, microonde, fasci di particelle)
  - garantire che i nuclei di elio siano ben confinati e trasferiscano la loro energia al plasma
  - spegnere il riscaldamento dall'esterno e lasciare che siano i nuclei di elio a tener caldo il plasma (ignizione)





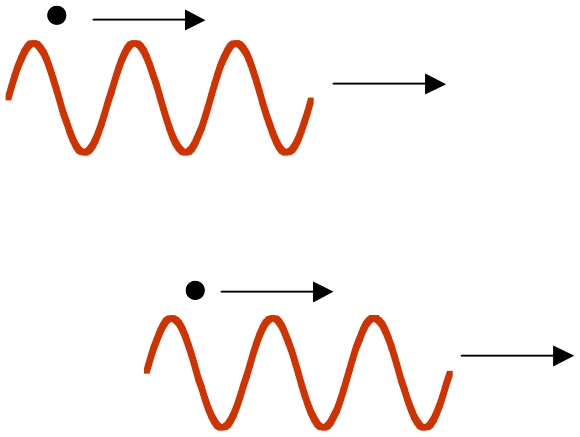
La fusione nucleare (segue…)

- Per tenere caldo il centro del plasma è necessario un trasporto di energia e di particelle ridotto



- Onde elettromagnetiche che si sviluppano nel plasma, però, incrementano il trasporto

- Risonanze tra onde elettromagnetiche e particelle cariche che viaggiano in fase con l'onda:
  - l'onda cresce in ampiezza a spese dell'energia della particella
  - il campo elettromagnetico che così si produce trascina la particella verso il bordo del plasma



La fusione nucleare (segue…)

- Una **comprensione piena** del comportamento del sistema richiede una trattazione
  - **auto-consistente** (non solo moto delle particelle determinato dai campi elettromagnetici, ma anche campi determinati dalla distribuzione di particelle)
  - **cinetica** (particelle con velocità diverse interagiscono con i campi in modo diverso: risonanze)
  - **non lineare** (il confinamento delle particelle dipende da quanto i campi modificano le loro orbite: rilevante l'ampiezza di saturazione)

$\Rightarrow$ approccio **computazionale**

La simulazione “a particelle”

Idea ovvia: far calcolare al computer, ad ogni istante, posizioni e velocità di tutte le particelle, e i campi elettromagnetici corrispondenti

Equazioni del moto:

$$\frac{d\mathbf{x}_i}{dt} = \mathbf{v}_i(t)$$

$$\frac{d\mathbf{v}_i}{dt} = \frac{e_i}{m_i}\left[\mathbf{E} + \frac{\mathbf{v}}{c} \times \mathbf{B}\right]_i$$

Equazioni dei campi:

$$\nabla \cdot \mathbf{E} \approx 4\pi \sum_{i=1}^{N_{part}} e_i \delta(\mathbf{x} - \mathbf{x}_i(t))$$

$$\nabla \times \mathbf{B} \approx \frac{4\pi}{c} \sum_{i=1}^{N_{part}} e_i \mathbf{v}_i(t) \delta(\mathbf{x} - \mathbf{x}_i(t))$$

La soluzione più immediata: simulazione N-corpi

- Calcolo dei campi nel punto in cui si trova la particella come sovrapposizione dei campi prodotti da ciascuna delle altre particelle:

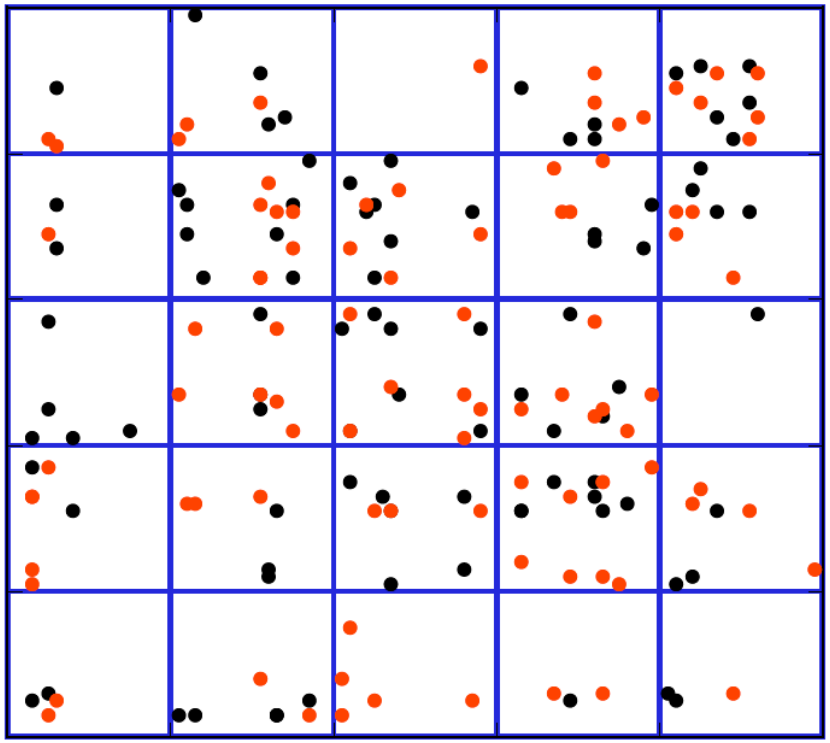
$$\mathbf{E}(\mathbf{x}_i, t) \approx \sum_{\substack{1 \\ j \neq i}}^{N_{part}} e_j \, \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|^3}$$

$$\mathbf{B}(\mathbf{x}_i, t) \approx \frac{1}{c} \sum_{\substack{1 \\ j \neq i}}^{N_{part}} e_j \, \mathbf{v}_j \times \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|^3}$$

- il **numero di operazioni** per l'evoluzione delle coordinate delle particelle scala con $\mathbf{N_{part}^2}$

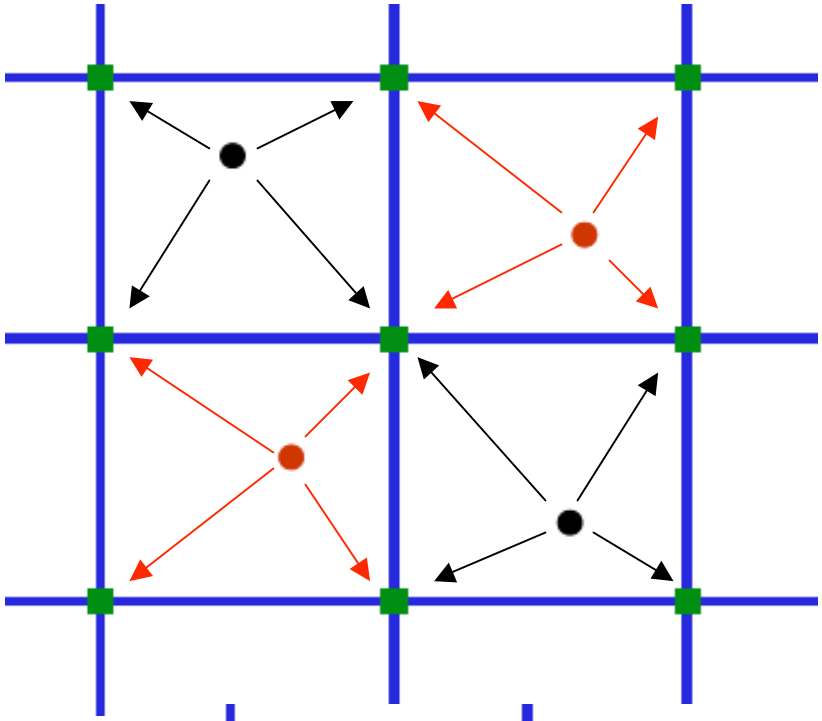
Alternativa: *particle in cell*

- Benché le particelle si muovano nel continuo, il dominio spaziale viene suddiviso in celle (griglia)

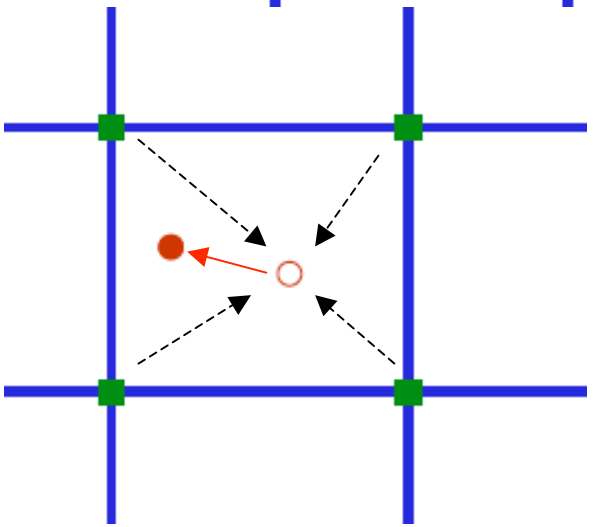


- Ciascuna particella “distribuisce” la propria densità di carica e di corrente ai vertici vicini: *density computing* ($\propto \mathbf{N_{part}}$)

- I campi vengono calcolati solo sui vertici delle celle, in base alle densità di carica e di corrente: *field solving* ($\propto \mathbf{N_{celle}}\ \mathbf{Log}\ \mathbf{N_{celle}}$)



- Posizione e velocità di ciascuna particella sono ricalcolate in base ai campi nella posizione della particella, ottenuti da un’interpolazione dei valori calcolati sui vertici vicini: *particle pushing* ($\propto \mathbf{N_{part}}$)



- Il **numero delle operazioni** scala con N_{part} ($N_{part} \gg N_{celle}$), anziché con N_{part}^2

Plasma numerico: accuratezza

- **Discretizzazione** dello **spazio fisico** e dello **spazio delle velocità**: ciascuna "particella" rappresenta un gran numero di particelle fisiche con coordinate **x**,**v** in un intorno di $\mathbf{x}_i,\mathbf{v}_i$ (**macroparticella**):

  - celle di volume $\propto \lambda_\perp^2 \lambda_{||}$ ($\lambda_\perp$ e $\lambda_{||}$ lunghezze d'onda tipiche)

  - molte macroparticelle in ogni cella (descrizione accurata dello spazio delle velocita')

  - ITER: $N_{cell} \sim 10^7 \div 10^8$
    $N_{part} \sim 10^2 \div 10^3 N_{cell}$

Parallelizzazione di un codice PIC su architettura *distributed memory*

- $N_{part} >> N_{celle}$

  $\Rightarrow$ prioritaria la distribuzione dei carichi (memoria e calcolo)

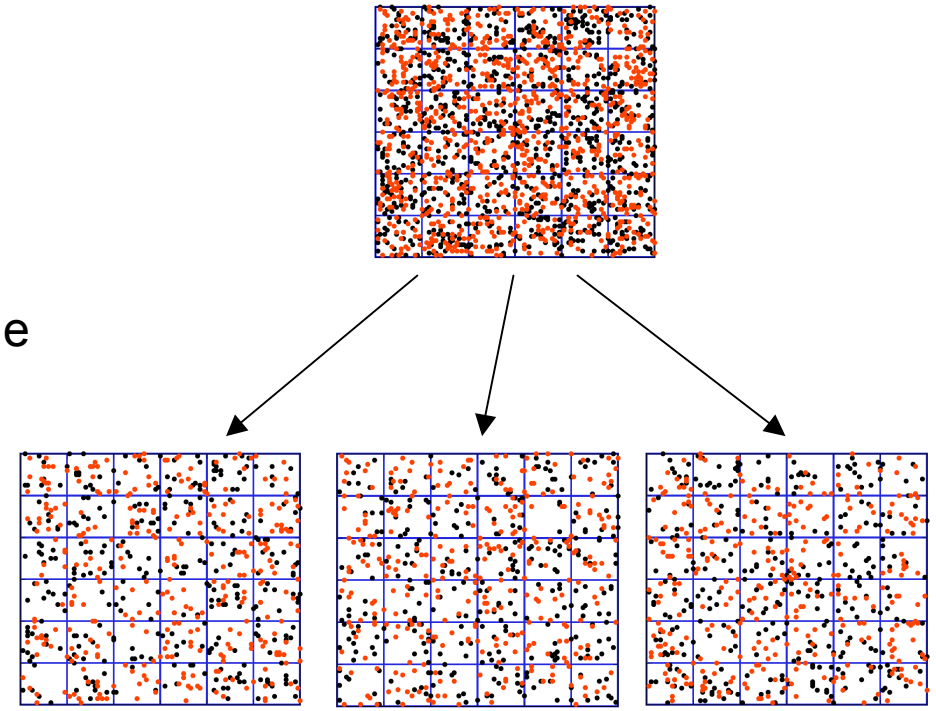
  relativi alle particelle

- Due strategie possibili:
  - decomposizione “per particelle”
  - decomposizione “per dominio”

Decomposizione per particelle

- Dominio **replicato**, particelle **distribuite**

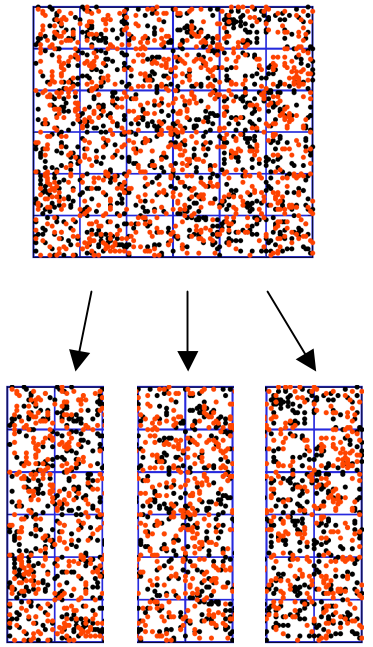
- Pro:
  - **carichi** perfettamente **bilanciati**
  - **nessuna migrazione** di particelle da un nodo all'altro
  - implementazione relativamente **semplice**



- Contro:
  - memoria relativa ai campi e alle densità replicata (la **risoluzione massima** ottenibile **non scala** perfettamente con $\mathbf{N_{nodi}}$)
  - necessaria una ***riduzione*** tra nodi delle densità (**comunicazione**: lo ***speed up*** non scala perfettamente con N_{nodi})

Decomposizione per dominio

- Dominio **distribuito**; particelle **distribuite**



- Pro:
  - memoria relativa a campi e densità distribuita (**risoluzione** spaziale massima ottenibile $\propto$ $\mathbf{N_{nodi}}$)
  - non è necessaria *riduzione* tra nodi delle densità (*speed up* $\propto N_{nodi}$)
  - eccezione: confine tra sub-domini adiacenti

    (v. prossima *slide*)

- Contro:
  - **migrazione** di particelle da un nodo all'altro: *check* su ogni particella dopo il *pushing* ed eventuale riassegnazione al nuovo nodo (comunicazione)
  - possbile **sbilanciamento dei carichi** con perdita dello scaling quasi ideale di risoluzione ottenibile e *speed up* con N_{nodi}
  - cura: **bilanciamento dinamico** (riassegnazione di dominio e particelle quando necessario)
  - più difficile da implementare

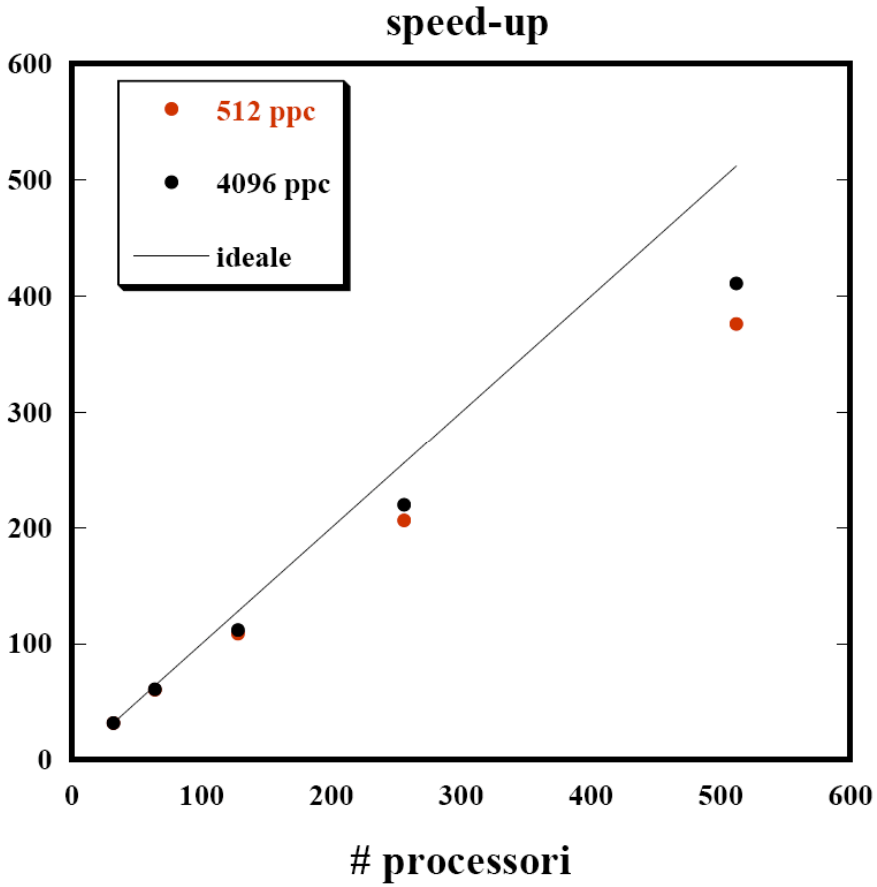
Parallelizzazione su architettura *shared memory*

- I **processori** condividono la memoria
- Distribuiamo le iterazioni dei *loops* sulle particelle tra i diversi processori mediante direttive **OpenMP**

- Distribuzione delle iterazioni del *loop* relativo al ***pushing***: è possibile che una particella (iterazione) **legga** i campi da locazioni di memoria lette simultaneamente da altra particella; **scrive** però le nuove coordinate su locazioni di memoria non toccate da altre particelle
- Sufficiente *privatizzare* le variabili “locali”

- Distribuzione delle iterazioni del *loop* relativo al **calcolo delle densità**: particelle diverse, spazialmente vicine, possono cercare di **scrivere** simultaneamente (***race conditions***) sulla stessa locazione di memoria (es.: densità di carica in un certo vertice di cella)
- Si introducono copie ***private*** (una per processore) degli *arrays* da aggiornare e si effettua la ***riduzione*** delle copie al termine del *loop*

Performances



Cresco; 256 nodi Intel-Xeon
con 2 processori quad core per nodo

Performances: modello predittivo

- Decomposizione ottimale (dominio-dominio)

- Memoria per nodo ed elapsed time per passo richiesti:

$$\mathbf{M_{DD}} \approx \frac{\mathbf{N_{cell}}}{\mathbf{n_{node}}}[\mathbf{m_{field}} + \mathbf{m_{density}} + (\mathbf{m_{part}} + \delta\mathbf{m_{part}})\mathbf{n_{ppc}}]$$

$$\mathbf{t_{DD}} \approx \frac{\mathbf{N_{cell}}}{\mathbf{n_{node}}}[\mathbf{t_{FFT}} \log_{\mathbf{2}} \mathbf{N_{cell}} + \mathbf{t_{int}} \frac{\mathbf{n_{ppc}}}{\mathbf{n_{proc}}} + \mathbf{t_{com}} \left(\frac{\mathbf{n_{node}}}{\mathbf{N_{cell}}}\right)^{\frac{1}{3}} \mathbf{n_{ppc}} + \mathbf{t_{sort}} \left(\frac{\mathbf{n_{node}} \mathbf{n_{proc}}}{\mathbf{N_{cell}}}\right)^{\frac{1}{3}} \mathbf{n_{ppc}}]$$

Performances: modello predittivo (cont.)

- Vincoli:
  - memoria per nodo effettivamente disponibile

$$\mathbf{M_{DD} \leq M_0 = m_0 n_{proc}}$$

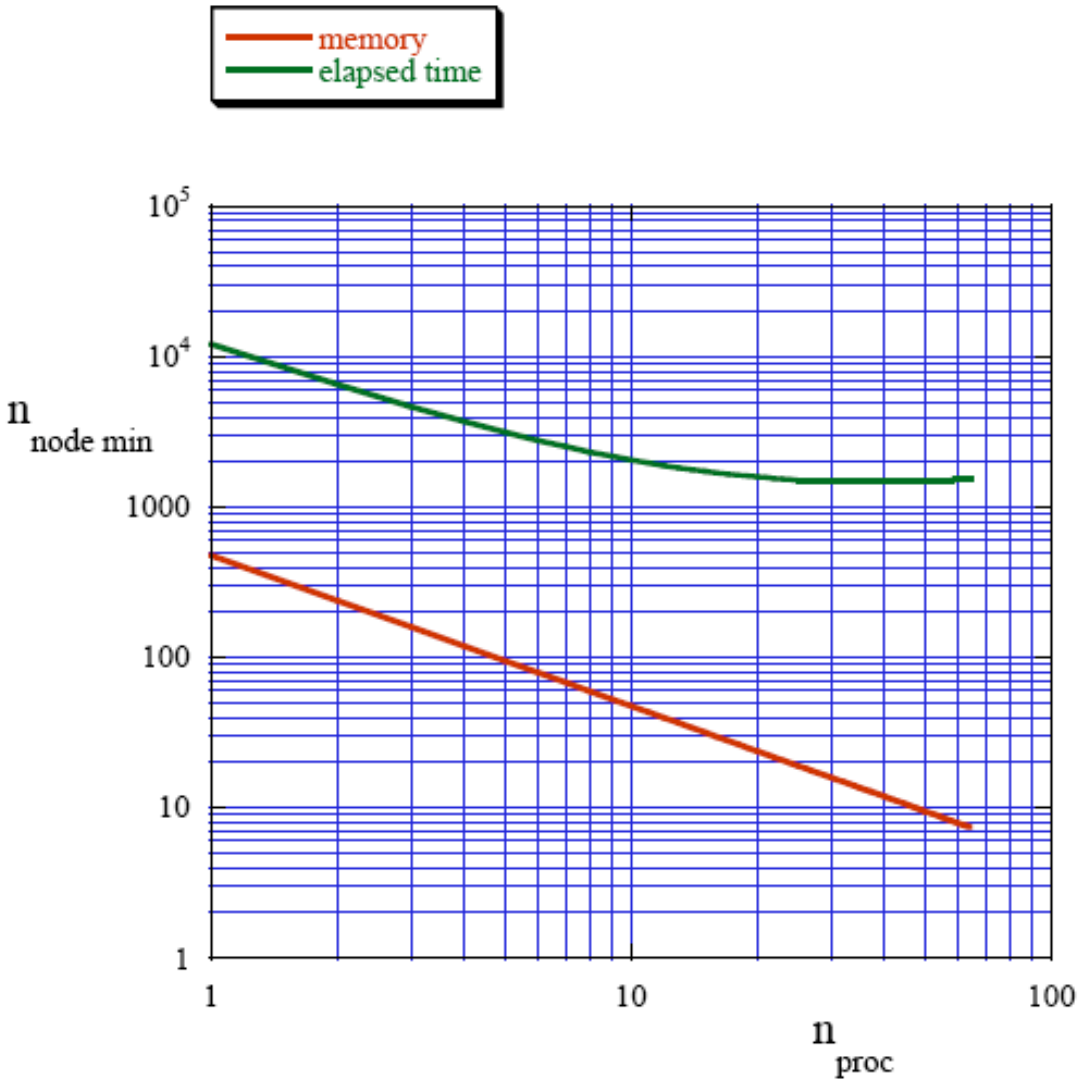
  - tempo di simulazione complessivo ragionevole

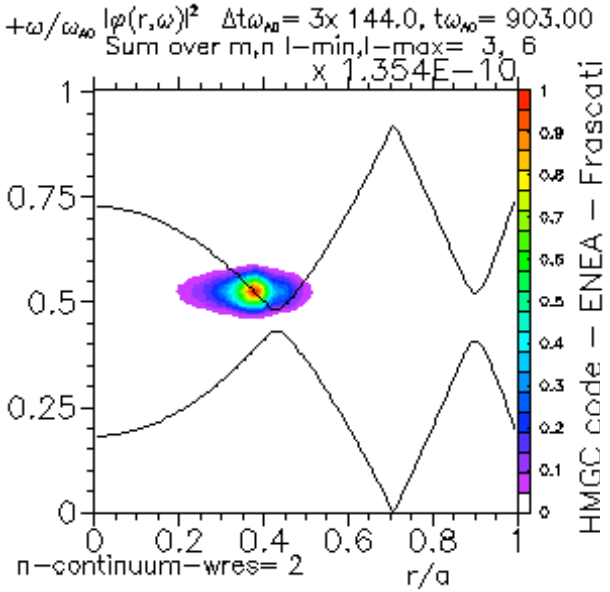
$$\mathbf{t_{tot} \equiv n_{step} t_{DD} \leq t_{lim}}$$

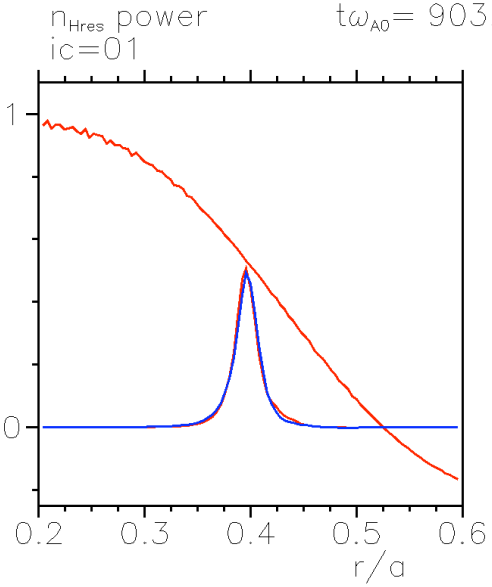
- Parametri:

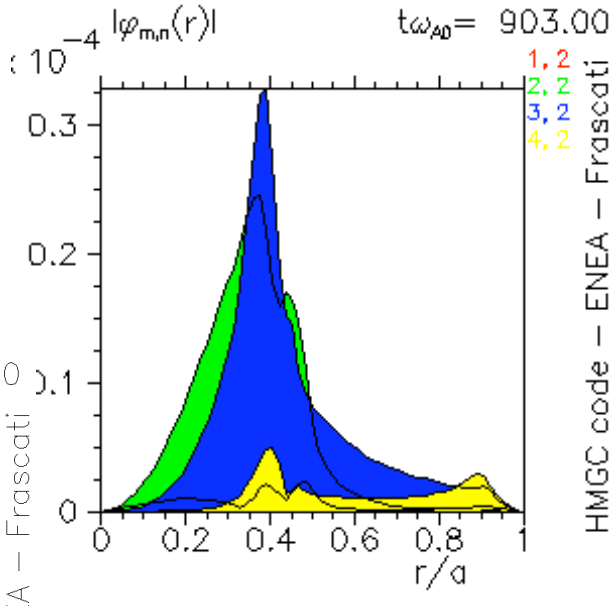
m_{part}=64 bytes, m_{field}=48 bytes, m_{press}=8 bytes, δm_{part}=24 bytes,
$t_{int} \approx 2.35 \times 10^{-5}$ sec (su processore AMD 2.4GHz)
t_{com}/t_{int}=0.5, t_{FFT}/t_{int}=0.1, t_{sort}/t_{int}=1
$n_{step} = 10^4$, n_{ppc}=200
m_0=4 Gbytes, t_{lim}=100 ore

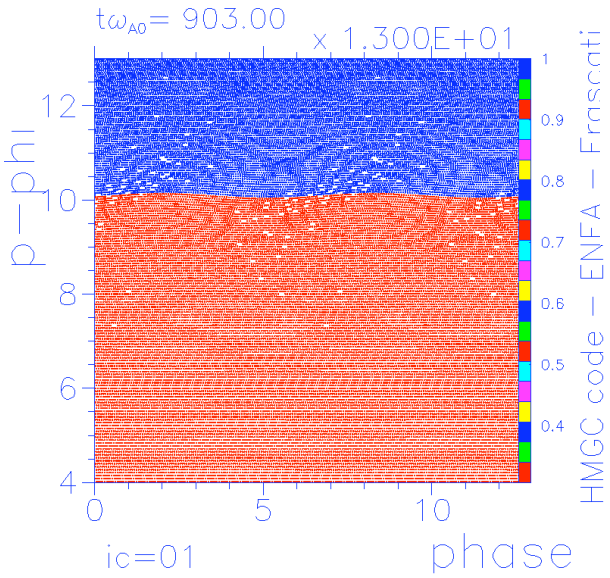
Performances: modello predittivo (cont.)

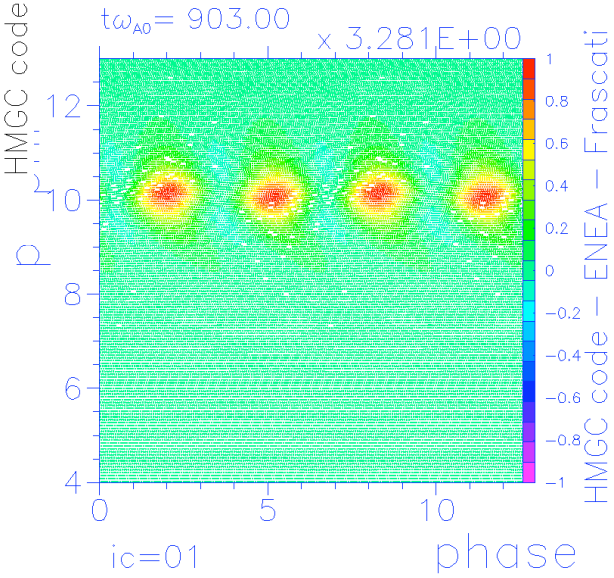



+ω/ω_A0 |φ(r,ω)|² Δtω_A0= 3x 144.0, tω_A0= 903.00
Sum over m,n l-min,l-max= 3, 6
x 1.354E-10
n-continuum-wres= 2
r/a
HMGC code – ENEA – Frascati


n_Hres power
ic=01
tω_A0= 903.00
r/a


|φ_m,n(r)|
tω_A0= 903.00
1, 2
2, 2
3, 2
4, 2
r/a
HMGC code – ENEA – Frascati


tω_A0= 903.00
x 1.300E+01
p-phi
ic=01
phase
HMGC code – ENFA – Frascati


tω_A0= 903.00
x 3.281E+00
ic=01
phase
HMGC code – ENEA – Frascati