

# **HPC IN COMPUTATIONAL MECHANICS**

# The world made discrete: from PDE's to computer programs

## □ General Form of PDE's for Engineering Systems

Given  $f : \Omega \rightarrow \mathbb{R}^{n_{sd}}$  e  $r : \Gamma \rightarrow \mathbb{R}^{n_{sd}}$ ,  $n_{sd} = 1, 2$ , or  $3$  and  $t \in I = [0, T]$  find  $u = u(x, t)$ ,  $x \in \mathbb{R}^{n_{sd}}$ , such as:

$$\mathcal{L}(u) + f = 0 \quad \text{in} \quad \Omega \times I, \quad \Omega \subset \mathbb{R}^{n_{sd}} \quad (1)$$

$$\mathcal{M}(u) + r = 0 \quad \text{in} \quad \Gamma \times I \quad (2)$$

$$u(x, 0) = u_o(x) \quad x \in \Omega \quad (3)$$

## Governing Equations in Eulerian Framework

### Navier–Stokes Equations

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} - \nu \Delta \mathbf{u} + \frac{1}{\rho} \nabla p = \mathbf{q} + \mathbf{f}(T, \mathbf{c}) \quad \text{in } \Omega$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega$$

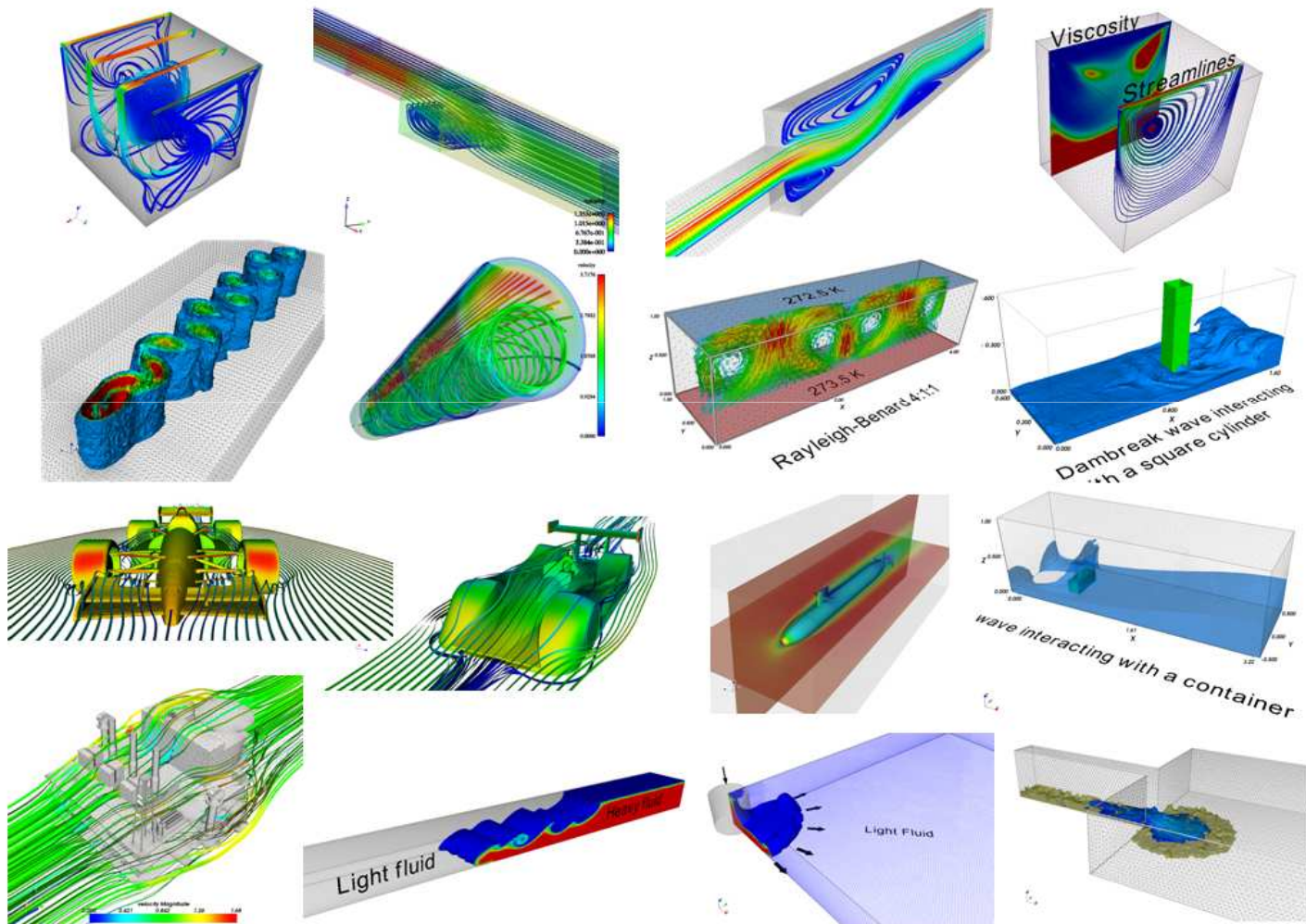
### Energy Transport Equation

$$\rho c_p \frac{\partial T}{\partial t} + \rho c_p \mathbf{u} \cdot \nabla T - \nabla \cdot (k \nabla T) = h_1(T, \mathbf{c}) \quad \text{in } \Omega$$

### Mass Transfer Equations

$$\frac{\partial \mathbf{c}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{c} - \nabla \cdot (K \nabla \mathbf{c}) = \mathbf{h}_2(T, \mathbf{c}) \quad \text{in } \Omega$$

# Parallel FEM Solver for Coupled Viscous Flow and Transport





## Eulerian Governing Equations

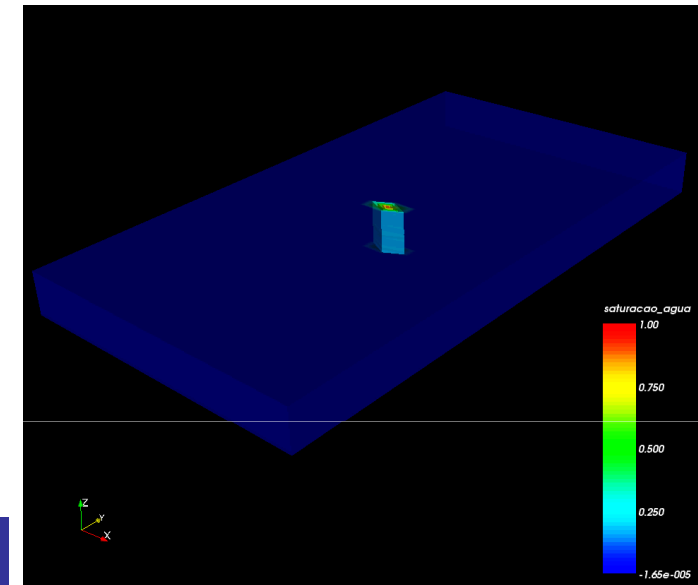
- Multi-phase Darcy-flow in Porous Media:

$$\mathbf{u}_\pi = -\frac{\kappa \mathbf{K}_{ij}}{\mu_\pi} \frac{\partial \Phi_\pi}{\partial x_j}$$

$$\Phi_\pi = p_\pi - \rho_\pi \mathbf{g} \cdot \mathbf{z}$$

$$\frac{\partial (S_\pi \rho_\pi \phi)}{\partial t} = \nabla \cdot \left( \frac{\mathbf{K}_{ij}}{\mu_\pi} \frac{\partial \Phi_\pi}{\partial x_j} \rho_\pi \right) + \rho_\pi q_\pi$$

$$\pi = 1, 2, \dots, n_{\text{phases}}$$



## Governing Equations in Lagrangian Framework

- Equation of Motion for Solids and Structures:

$$\rho u_{i,tt} - \sigma_{ij,j} + f_i = 0 \text{ in } \Omega \times I$$

$$u_i = g_i \text{ in } \Gamma_g \times I$$

$$\sigma_{ij}n_j = h_i \text{ in } \Gamma_h \times I$$

$$u_i(x, 0) = u_{o_i}(x) \quad x \in \Omega$$

$$u_{i,t}(x, 0) = \dot{u}_{o_i}(x) \quad x \in \Omega$$

## Lagrangian Governing Equations

### □ Remarks:

# 1 Material Law

$$\sigma_{ij} = C_{ijkl} \epsilon_{kl}$$

# 2 Material Non-linearity

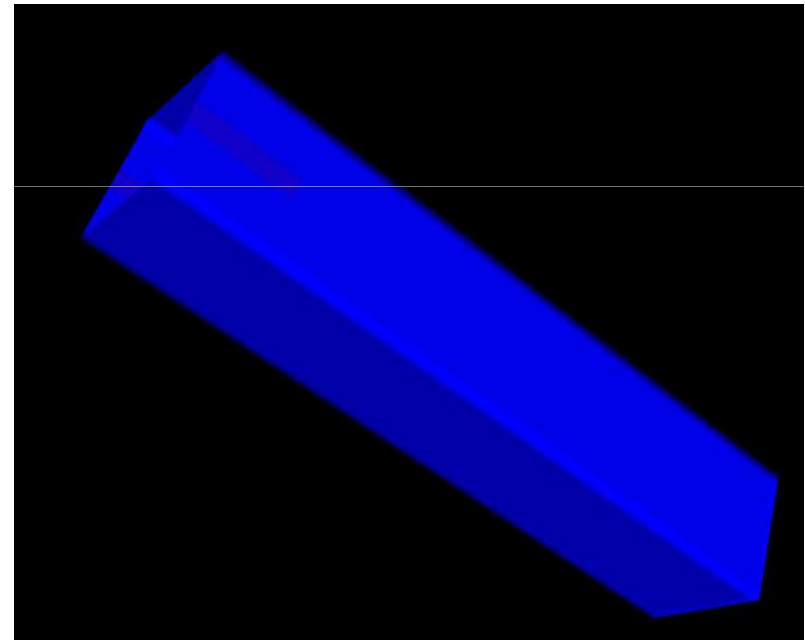
$$\epsilon = \epsilon^e + \epsilon^p$$

$\epsilon^p$  = plastic strains

# 3 Large Displacements

$$\epsilon_{ij} \neq \frac{1}{2}(u_{i,j} + u_{j,i})$$

# 4 Contact



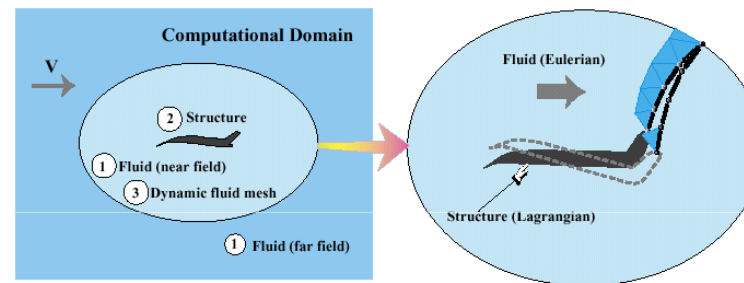
courtesy of J. Alves

## Arbitrary Eulerian Lagrangian Governing Equations

- **Incompressible NS equations in ALE frame moving with velocity  $w$ :**

$$\rho \left[ \frac{du_a}{dt} + (u_b - w_b) \frac{\partial u_a}{\partial x_b} \right] - \frac{\partial \tau_{ab}}{\partial x_b} + \frac{\partial p}{\partial x_a} = 0$$

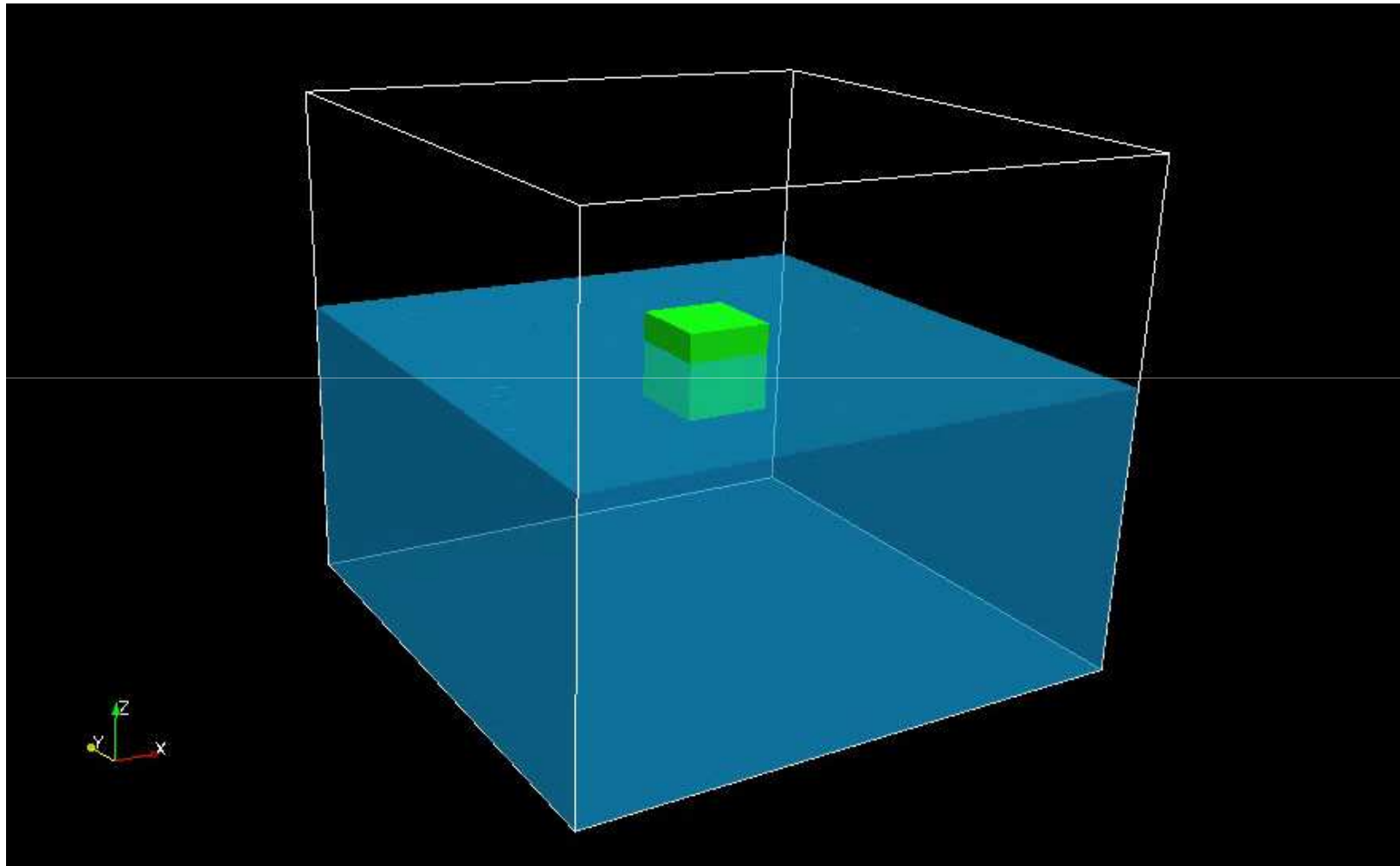
$$\frac{\partial u_a}{\partial x_a} = 0$$



From Felippa, Park and Farhat (CMAME, 2001)

- **Velocity  $w$  is conveniently adjusted to Eulerian ( $w=0$ ), far from moving object to Lagrangian ( $w=u$ ) on the fluid-structure interface.**
- **Fluid is considered attached to the body.**
- **Need to solve extra-field equation to define mesh movement: our choice is to solve the Laplacian.**

# Fluid-structure interaction with free-surface



# FEM Discretization

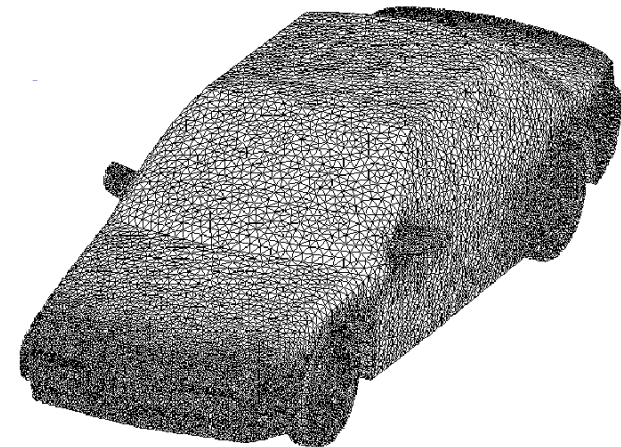
- **Good mathematical background and ability to handle complex geometries by using unstructured grids**

Let  $\mathcal{E}$  be the set of all elements in the discretization of  $\Omega$  in sub-domains  $\Omega^e$ ,  $e = 1, 2, \dots, n_{el}$ , such as,

$$\overline{\Omega} = \bigcup_{e=1}^{n_{el}} \overline{\Omega}^e \quad ; \quad \bigcap_{e=1}^{n_{el}} \Omega^e = \emptyset$$

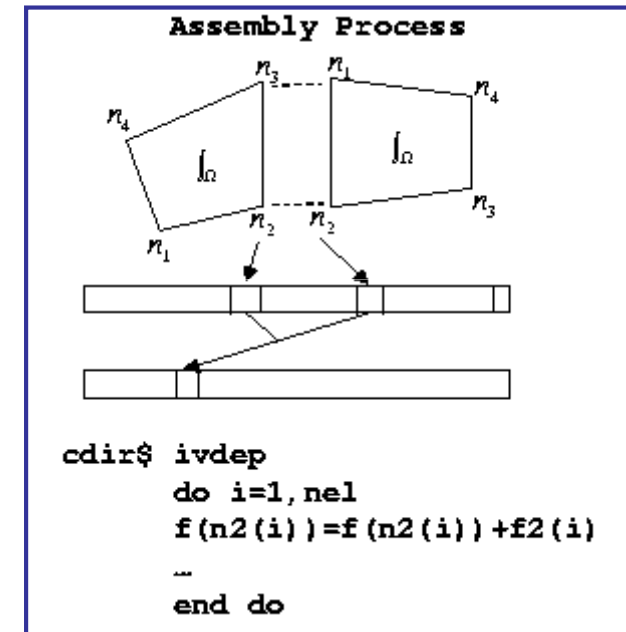
We associate to  $\mathcal{E}$  finite dimension spaces

$$H^{kh} = \{ \phi^h / \phi^h \in C^0(\overline{\Omega}), \phi^h / \Omega^e \in P^k \} \quad k = 1, 2, \dots$$



# FEM Computing Issues

- **FEM is a unstructured grid method characterized by:**
  - Discontinuous data – no i-j-k addressing
  - Gather-scatter operations
  - Random memory access patterns
  - Data dependence
  - Minimize indirect addressing is a must
  - Memory complexity  $O(\text{mesh parameters})$

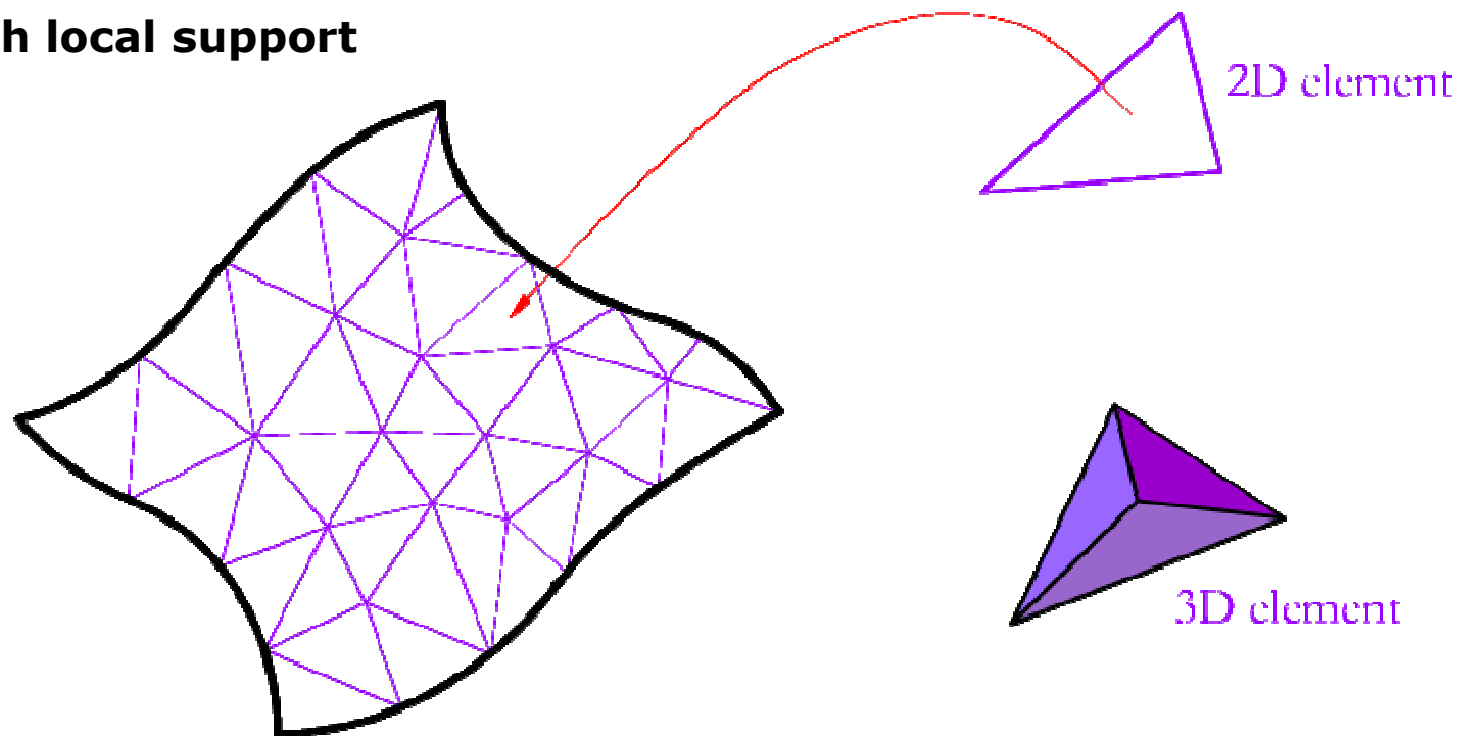


# **PART I: FE SIMULATION CODE**



# Finite Element Method

- **Based on a variational formulation for a given PDE; basis functions with local support**



- **Flexible**
- **Unstructured grid generation in 3D can be difficult**

# FE Simulation Code: Major Components

Pre-processing

Input data

Time integration loop

Nonlinear iteration loop

Form system of linear equations

Solve system of linear equations

End NL loop

Update time step

Output results

End time loop

Post-processing

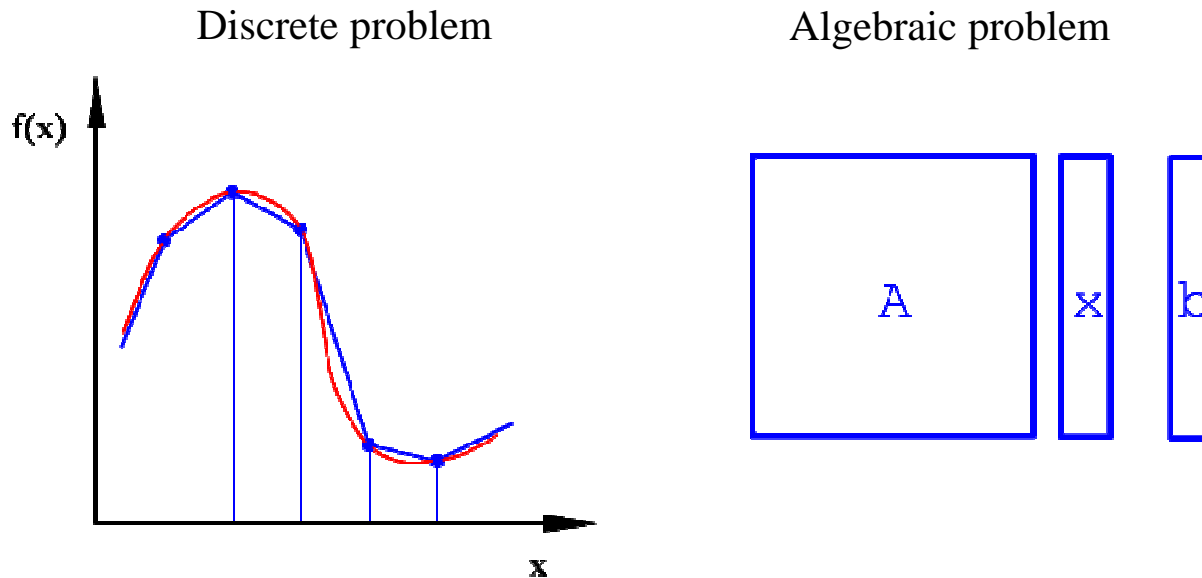
Visualization

Complexity  $O(n^{4/3})$ ,  $n$  #unknowns

Optimal solvers require  $O(n)$  work per time step, and time accurate integration often implies  $O(n^{1/3})$  time steps.

# Major Components

- **Always ends in a system of linear equations**



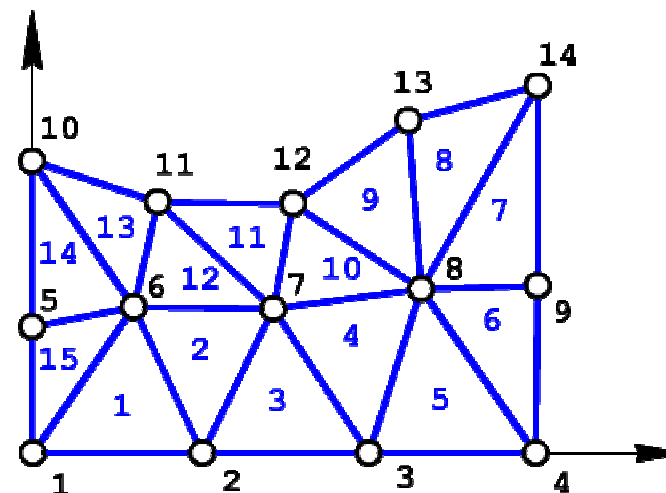
- **Often linear system too big to be solved with a direct method based on Gauss elimination**
- **Iterative solvers are preferred for large systems**
- **HPC allows solving really big systems!**

# Computational Representation of an Unstructured Mesh

nodal coordinates

`real*8 x(ndim,nnos)`

id	x(1,id)	x(2,id)
1	0.00	0.00
2	0.80	0.00
3	1.60	0.00
4	2.40	0.00
5	0.00	0.60
6	0.50	0.70
7	1.15	0.70
8	1.85	0.75
9	2.40	0.80
10	0.00	1.40
11	0.60	1.20
12	1.25	1.20
13	1.80	1.60
14	2.40	1.75



element connectivity

`integer ien(nnoel,nel)`

ie	ien(1,ie)	ien(2,ie)	ien(3,ie)
1	1	2	6
2	2	7	6
3	2	3	7
4	3	8	7
5	3	4	8
6	4	9	8
7	8	9	14
8	8	14	13
9	8	13	12
10	7	8	12
11	7	12	11
12	6	7	11
13	6	11	10
14	5	6	10
15	1	6	5

**ndim**: number of space dimensions

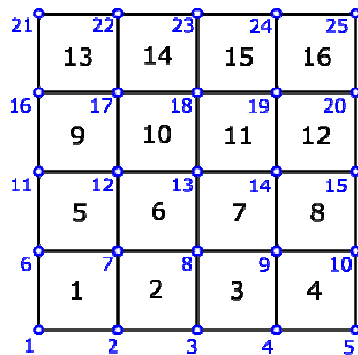
**nnos**: number of nodes

**nnoel**: number of element nodes

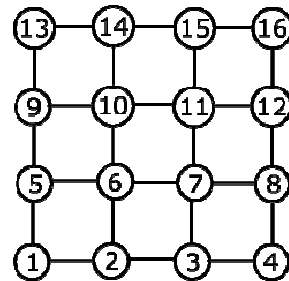
**nel**: number of elements

# Meshs, Graphs and Matrices

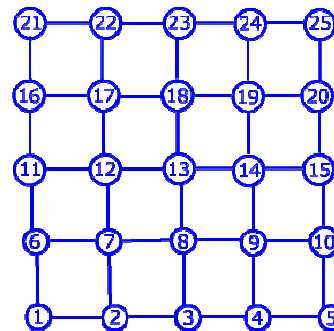
Finite difference grid



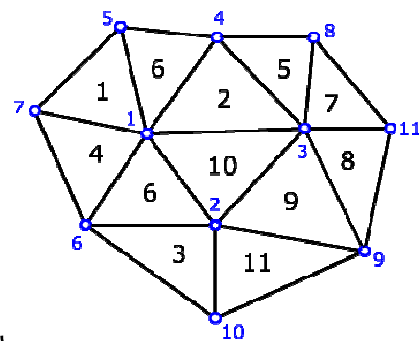
Dual graph (cells)



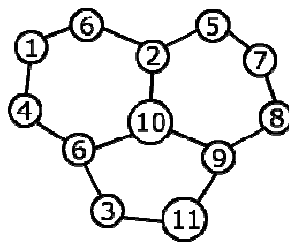
Grid points graph



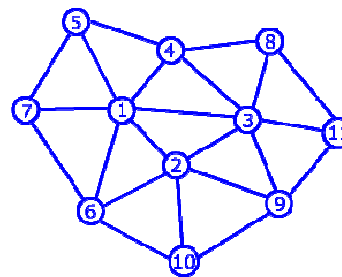
Unstructured mesh



Dual graph (elements)



Nodal graph



Sparse Matrix

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} & & & & & & \\ a_{2,1} & a_{2,2} & a_{2,3} & & & a_{2,6} & & a_{2,9} & a_{2,10} & & & & \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & & & & a_{3,8} & a_{3,9} & & a_{3,11} & & \\ a_{4,1} & & a_{4,3} & a_{4,4} & a_{4,5} & & & a_{4,8} & & & & & \\ a_{5,1} & & & a_{5,4} & a_{5,5} & & a_{5,7} & & & & & & \\ a_{6,1} & a_{6,2} & & & & a_{6,6} & a_{6,7} & & & a_{6,10} & & & \\ a_{7,1} & & & & a_{7,5} & a_{7,6} & a_{7,7} & & & & & & \\ & & a_{8,3} & a_{8,4} & & & & a_{8,8} & & & a_{8,11} & & \\ & a_{9,2} & a_{9,3} & & & & & & a_{9,9} & a_{9,10} & a_{9,11} & & \\ & a_{10,2} & & & & a_{10,6} & & & a_{10,9} & a_{10,10} & & & \\ & & a_{11,3} & & & & & a_{11,8} & a_{11,9} & & a_{11,11} & & \end{bmatrix}$$

geometrical representation

relational representation

algebraic representation

# Krylov Space Iterative Solution Methods

- ❑ Iterative methods more used in practice for non-symmetric systems  $Ax=b$ : **GMRES**
- ❑ **GMRES solution update:**  $x_k = x_0 + y_k$
- ❑  $y_k$  computed as the best approximation in the Krylov space:

$$K_m = \text{span}[r_0, Ar_0, A^2r_0, \dots, A^{k-1}r_0]$$

minimizing the residual

$$||r_k|| = \min_y ||r_0 + Ay||$$

- ❑ In practice applied in cycles

See more in: Y. Saad, Iterative Methods for Sparse Linear Systems, Boston, PWS Publishing Company, 1996

## GMRES Method

```

r=b-Ax ,   v1=r/||r|| ,   tol=||r|| , k=0
While tol > etol ||b|| and k<kmax do
    k=k+1

    For j=1,...,k
        hjk=(Avk)T vj
    vk+1= Avk - sum(hjk vj), j=1,...,k
    hk+1,k= || vk+1 ||
    vk+1 = vk+1 / || vk+1 ||
    e1=(1,0,...,0)T in R(k+1)

    Minimize ||tol e1 - Hk yk|| over R(k+1) to
    obtain yk

    tol = ||tol e1 - Hk yk||
End while.

```

# Preconditioning

- ❑ **GMRES convergence depends on spectra of  $A$**
- ❑ **Preconditioning main idea:**  
$$MAx = Mb$$
- ❑  **$M$  close to  $A^{-1}$  and chosen such as:**
  - $M$  is simple to build and implement
  - $M$  action diminishes the iteration count
- ❑ **Possible implementations**
  - Right:  $AMy = b$
  - Left:  $MAx = Mb$
  - Left-Right:  $MAMy = Mb$



# Main GMRES Operations

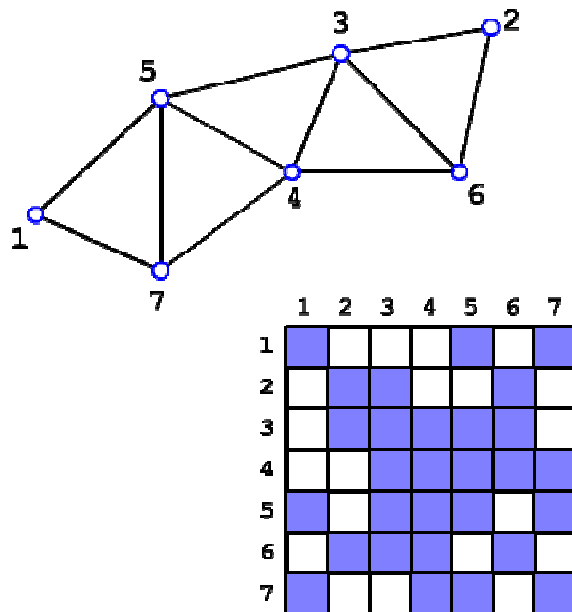
- ❑ **Dot Products**
- ❑ **Vector updates (SAXPY's)**  
$$y = y + ax$$
- ❑ **Matrix-vector products (matvec)**
- ❑ **Preconditioning, auxiliary system solve**

# Common Preconditioning Matrices

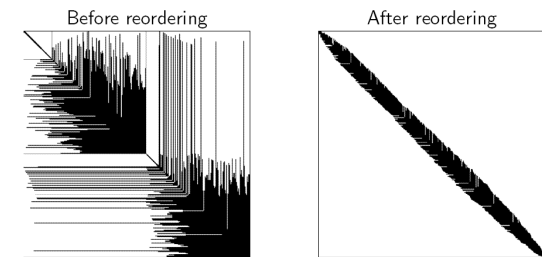
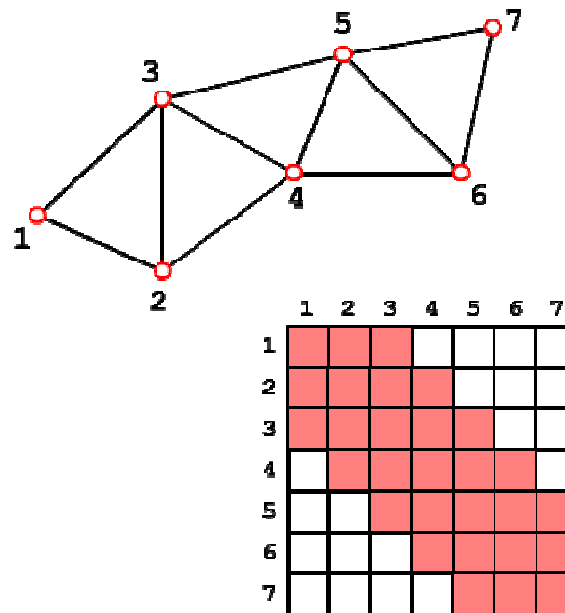
- ❑ **Simplest:  $M = \text{diag}(A)$** 
  - easy to implement, slow convergence
- ❑ **Nodal Block-Diagonal Preconditioning :**
  - $M$  = nodal block-diagonals of  $A$ , that is,  
block  $4 \times 4$ ,  $i=1,2, \dots, \text{nnos}$
- ❑ **Incomplete Factorization:  $M = \text{ILU}(A)$** 
  - Complex, hard to implement in parallel
  - Allows different levels of fill-in,  $\text{ILU}(0)$ ,  $\text{ILU}(1)$  ...

# Nodal Degrees-of-Freedom Reordering

- Matrix profile is directly related to the node (degrees-of-freedom) order
- Reordering algorithms such as Reverse Cuthill Mckee reorder the unknowns to reduce bandwidth (profile)
- NP-complete graph problem



basic example



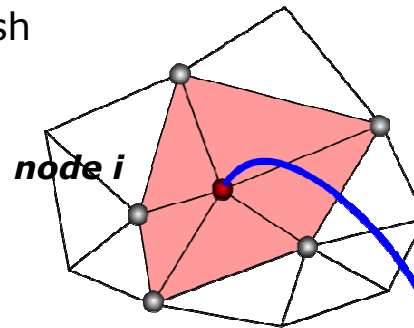
real  
problem

## Storing sparse matrices

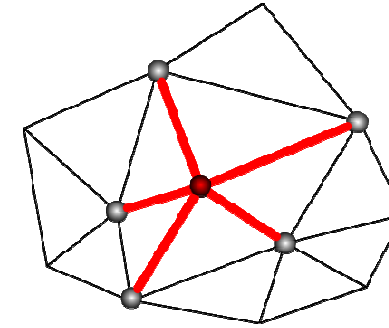
- ❑ **Local nature of FEM approximation yields sparse matrices, with few nonzero terms ( $\sim 90\%$  in 3D);**
- ❑ **There are several schemes to store efficiently such sparse matrices;**
- ❑ **These schemes store only the nonzero terms in compact form;**
- ❑ **We show some of these schemes, discuss the related computational issues;**

# Data Structures are Grid-based

Finite Element  
Mesh

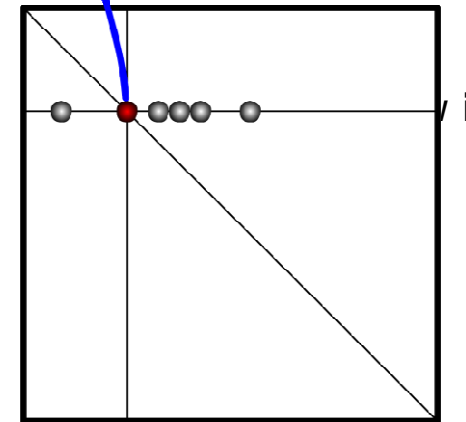


Nodal Graph



EBE, EDE or CSR; all data structures lead to problems with sparse Jacobian matrices; many tasks can leverage off an efficient set of tools for manipulating sparse data structures

**K=**



# General Ideas on Data Structures

- ❑ A proper data structure is crucial to achieve a good *performance X storage space* balance;
- ❑ Basic linear algebra kernels (e.g., matvec products) depend on the data structure to hold the matrix;
- ❑ Compact data structures to store sparse matrices are very convenient for direct and iterative linear equation solvers;
- ❑ Modern packages for linear equation solving (e.g., Petsc Trilinos, etc...) often support standard sparse data structures (e.g., CSR, BSR).
- ❑ Difficulties related to locality of data and impact of multi-core procs
- ❑ See Petsc web-page: <http://www.mcs.anl.gov/petsc/petsc-as/>

# Sparse Data Structures

- ❑ **DNS: Dense**
- ❑ **BND: Linpack Banded;**
- ❑ **COO: Coordinate;**
- ❑ **CSR: Compressed Sparse Row;**
- ❑ **CSC: Compressed Sparse Column;**
- ❑ **MSR: Modified CSR;**
- ❑ **ELL: Ellpack-Itpack;**
- ❑ **DIA: Diagonal;**
- ❑ **BSR: Block Sparse Row**
- ❑ **SSK: Symmetric Skyline;**
- ❑ **NSK: Nonsymmetric Skyline;**
- ❑ **JAD: Jagged Diagonal;**
- ❑ **EBE: Element-by-Element;**
- ❑ **EDS: Edge-by-Edge**

# CSR: Compressed Sparse Row

$$\mathbf{A} = \begin{bmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{bmatrix}$$

- Stores the assembled matrix
- Supported by most packages;
- Hard to extract performance;
- Easy to build ILU preconditioners;

AA(nnz) =	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.
JA(nnz) =	1	4	1	2	4	1	3	4	5	3	4	5
	1	2	3	4	5	6	7	8	9	10	11	12
IA(n+1) =	1	3	6	10	12	13						
	1	2	3	4	5	6						

AA: nonzero coefficients of A

JA: column indexes of nonzero entries

IA: #nonzeros in the lines



# Matrix-vector product CSR (SERIAL)

```
do i = 1, n
    k1 = ia(i)
    k2 = ia(i+1)-1
    y(i) = dotproduct(a(k1:k2),x(ja(k1:k2)))
End do
```

$$\mathbf{A} = \begin{bmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{bmatrix}$$

AA(nnz)=	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.
JA(nnz)=	1	4	1	2	4	1	3	4	5	3	4	5
IA(n+1)=	1	3	6	10	12	13						
	1	2	3	4	5	6						

**n** is the number of lines in A, **dotproduct** is the scalar product between (**a(k1:k2)**) and RHS vector.

Level 1 BLAS routine **ddot** usually employed in **dotproduct**

*Read more:* Williams et al, Optimization of sparse matrix–vector multiplication on emerging multicore platforms, Parallel Computing 35 (2009) 178–194

# BSR: Block Sparse Row

- Good for storing block coefficients (as in the case of many nodal degrees-of-freedom);
- Eventually will store null coefficients within the blocks;
- More complex to work than CSR

$$\mathbf{A} = \begin{bmatrix} \begin{array}{cc|cc|cc} \color{red}{1.} & \color{magenta}{2.} & 0. & 0. & \color{red}{3.} & \color{magenta}{4.} \\ \color{blue}{5.} & \color{green}{6.} & 0. & 0. & \color{blue}{7.} & \color{green}{8.} \end{array} & \begin{array}{cc|cc} 0. & 0. & 9. & 10. \\ 0. & 0. & 13. & 14. \end{array} & \begin{array}{cc|cc} 11. & 12. & 17. & 20. \\ 15. & 16. & 22. & 24. \end{array} \\ \hline \begin{array}{cc|cc} 17. & 18. & 0. & 0. \\ 22. & 23. & 0. & 0. \end{array} & \begin{array}{cc|cc} 20. & 21. & 24. & 25. \end{array} \end{bmatrix}$$

$$\mathbf{AA} = \begin{bmatrix} \color{red}{1.} & \color{red}{3.} & 9. & 11. & 17. & 20. \\ \color{blue}{5.} & \color{blue}{7.} & 13. & 15. & 22. & 24. \\ \color{magenta}{2.} & \color{magenta}{4.} & 10. & 12. & 18. & 21. \\ \color{green}{6.} & \color{green}{8.} & 14. & 16. & 23. & 25. \end{bmatrix}$$

$$\mathbf{JA} = \begin{bmatrix} 1 & 5 & 3 & 5 & 1 & 5 \\ \color{blue}{1} & \color{blue}{2} & \color{blue}{3} & \color{blue}{4} & \color{blue}{5} & \color{blue}{6} \end{bmatrix}$$

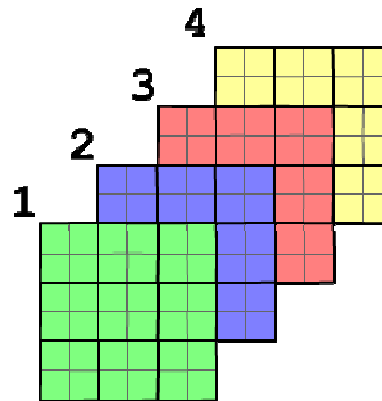
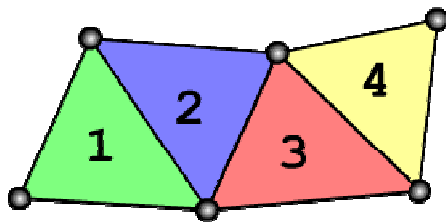
$$\mathbf{IA} = \begin{bmatrix} 1 & 3 & 5 & 7 \\ \color{blue}{1} & \color{blue}{2} & \color{blue}{3} & \color{blue}{4} \end{bmatrix}$$

# EBE: Element-by-Element

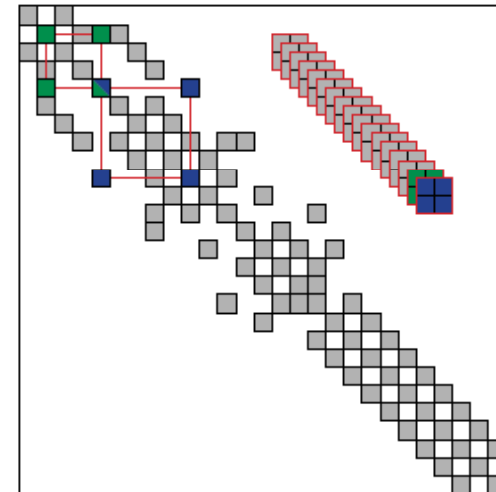
- ❑ **Storage scheme where element coefficients are individually stored;**
- ❑ **As coefficients are stored elementwise, no global (assembled) matrix is built, that is, unassembled element contributions are stored;**
- ❑ **Needs specially built preconditioners;**
- ❑ **See more:**
  - Hughes, Ferencz, Hallquist, Large-scale vectorized implicit calculations in solid mechanics on a Cray X-MP/48 utilizing EBE preconditioned conjugate gradients, *Computer Methods in Applied Mechanics and Engineering*, 61(2):215-248, 1987
  - E. Barragy and G. F. Carey, A Parallel Element-by-Element Solution Scheme, *IJNME*, 26, 2367-2382, 1988
  - F Shakib, T J R Hughes, Z Johan , A multi-element group preconditioned GMRES algorithm for nonsymmetric systems arising in finite element analysis, *Computer Methods in Applied Mechanics and Engineering*, 75: 415-456, 1989.

# EBE: Element-by-Element

- Consider the mesh below with triangular elements (3 nodes/element) with 2 degrees-of-freedom per node ( $ng1$ ):



$t(ng1 * nnoel, ng1 * nnoel, nel)$



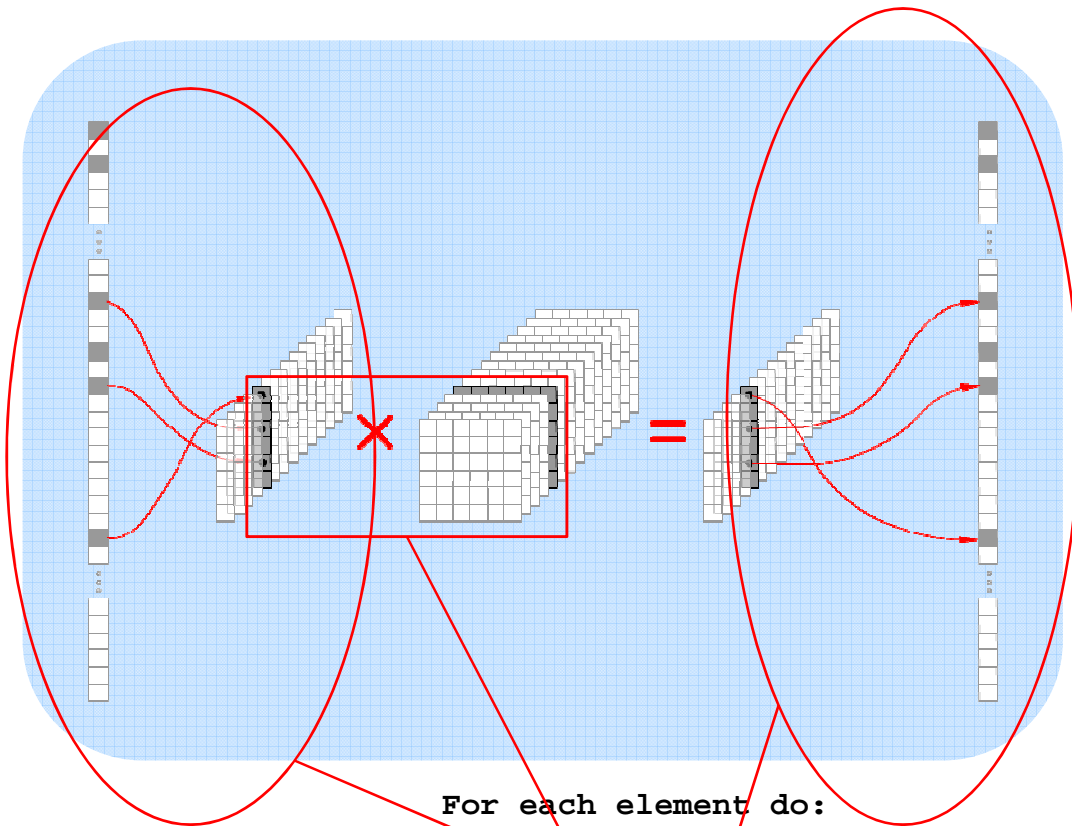
with:

**nel** : #elements

**nnoel**: #nodes per element

**ng1** : #degrees-of-freedom per node

# EBE matrix-vector product (SERIAL)



For each element do:

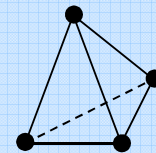
Localize (gather) coefficients from global vector

Do local matvec

Scatter and add result in global vector

Enddo

```
do ie = 1, nel
  neq1 = lm(1,ie)
  neq2 = lm(2,ie)
  neq3 = lm(3,ie)
  neq4 = lm(4,ie)
  ...
  retrieve and multiply 16 coefs
  ...
  p(neq1) = p(neq1) + ap1
  p(neq2) = p(neq2) + ap2
  p(neq3) = p(neq3) + ap3
  p(neq4) = p(neq4) + ap4
end do
```

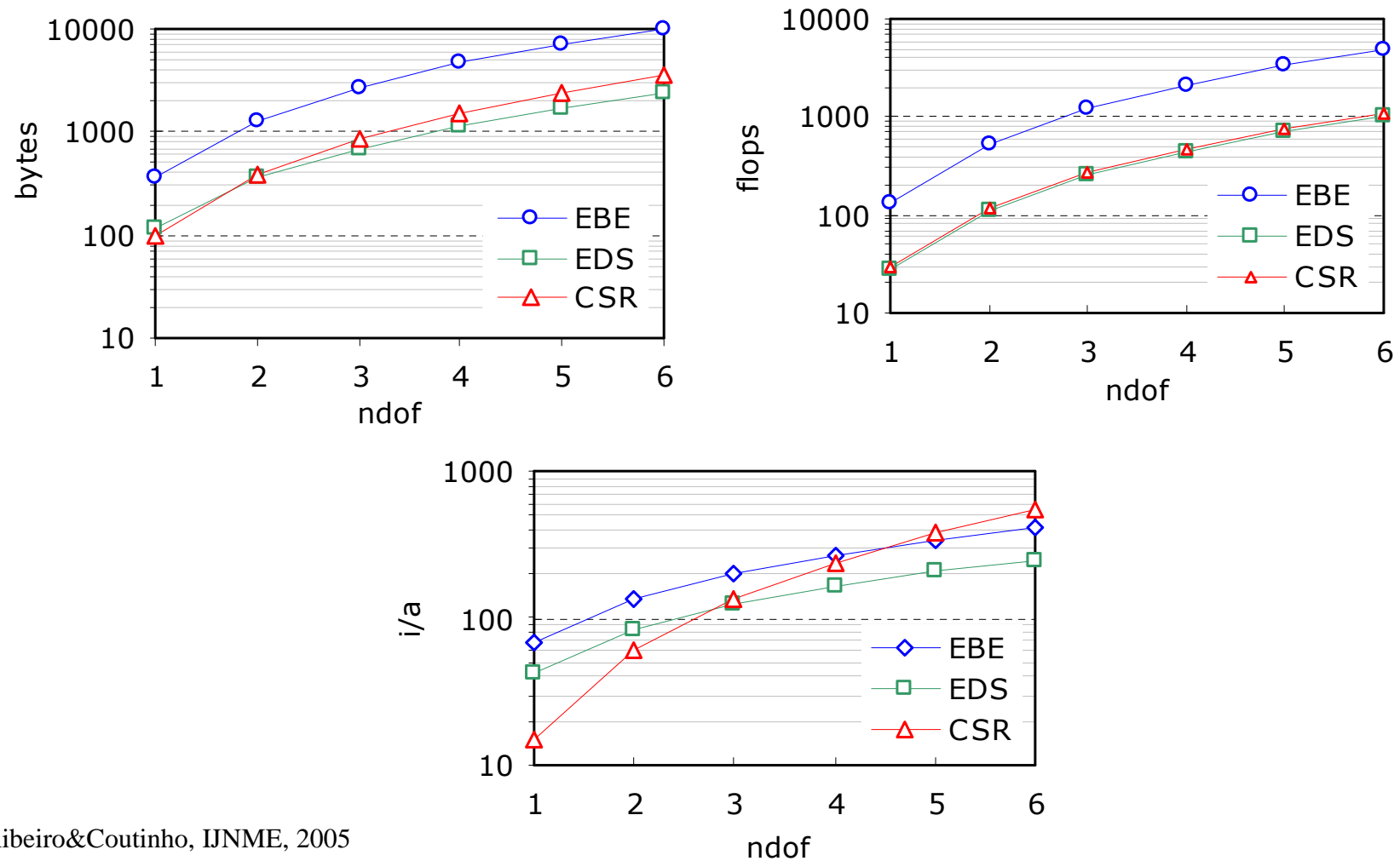


# EDS: Edges-by-Edges

- ❑ **Similar to EBE but here we store coefficients related to edges of the global sparse matrix;**
- ❑ **Scatter and add of edge coefficients performed only at matrix-vector product, as in EBE scheme;**
- ❑ **Several advantages for simplex elements;**
- ❑ **However, preconditioning can be difficult;**
- ❑ **See:**
  - F. L. B. Ribeiro and A. L. G. A. Coutinho, Comparison between element, edge and compressed storage schemes for iterative solutions in finite element analyses, *Int. J. Numer. Meth. Engng*, 63:569–588, 2005
  - R. Lohner, Applied CFD Techniques: An Introduction Based on Finite Element Methods, 2<sup>nd</sup> edition, Wiley, 2008

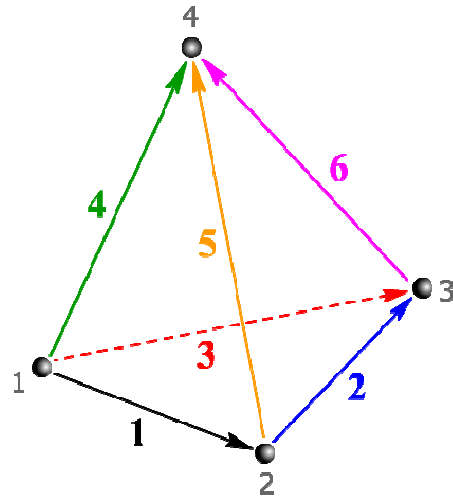


# Computational complexity of sparse matvec data with CSR, EBE, EDS for linear tets grids



from Ribeiro&Coutinho, IJNME, 2005

## EDS: Edges-by-Edge (cont.)



$$\mathbf{J}^e = \begin{bmatrix} \mathbf{J}_{11} & \mathbf{J}_{12} & \mathbf{J}_{13} & \mathbf{J}_{14} \\ \mathbf{J}_{21} & \mathbf{J}_{22} & \mathbf{J}_{23} & \mathbf{J}_{24} \\ \mathbf{J}_{31} & \mathbf{J}_{32} & \mathbf{J}_{33} & \mathbf{J}_{34} \\ \mathbf{J}_{41} & \mathbf{J}_{42} & \mathbf{J}_{43} & \mathbf{J}_{44} \end{bmatrix}$$

$$\mathbf{J}^e = \mathbf{T}_1^e + \mathbf{T}_2^e + \mathbf{T}_3^e + \mathbf{T}_4^e + \mathbf{T}_5^e + \mathbf{T}_6^e$$

“...algebraic disassembling of a tet into edges...”

$$\mathbf{T}_1^e = \begin{bmatrix} \mathbf{T}_{11}^1 & \mathbf{T}_{12}^1 & 0 & 0 \\ \mathbf{T}_{21}^1 & \mathbf{T}_{22}^1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{T}_2^e = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \mathbf{T}_{22}^2 & \mathbf{T}_{23}^2 & 0 \\ 0 & \mathbf{T}_{32}^2 & \mathbf{T}_{33}^2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{T}_4^e = \begin{bmatrix} \mathbf{T}_{11}^4 & 0 & 0 & \mathbf{T}_{14}^4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{T}_{41}^4 & 0 & 0 & \mathbf{T}_{44}^4 \end{bmatrix}$$

$$\mathbf{T}_3^e = \begin{bmatrix} \mathbf{T}_{11}^3 & 0 & \mathbf{T}_{13}^3 & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{T}_{31}^3 & 0 & \mathbf{T}_{33}^3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{T}_5^e = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \mathbf{T}_{22}^5 & 0 & \mathbf{T}_{24}^5 \\ 0 & 0 & 0 & 0 \\ 0 & \mathbf{T}_{42}^5 & 0 & \mathbf{T}_{44}^5 \end{bmatrix}$$

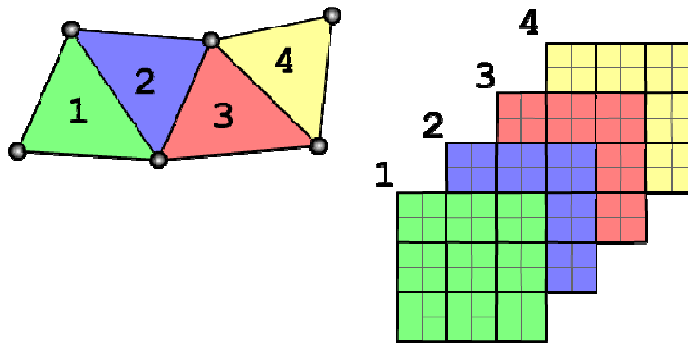
$$\mathbf{T}_6^e = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{T}_{33}^6 & \mathbf{T}_{34}^6 \\ 0 & 0 & \mathbf{T}_{43}^6 & \mathbf{T}_{44}^6 \end{bmatrix}$$

In practice, nodal block-diagonals are also used to build preconditioner

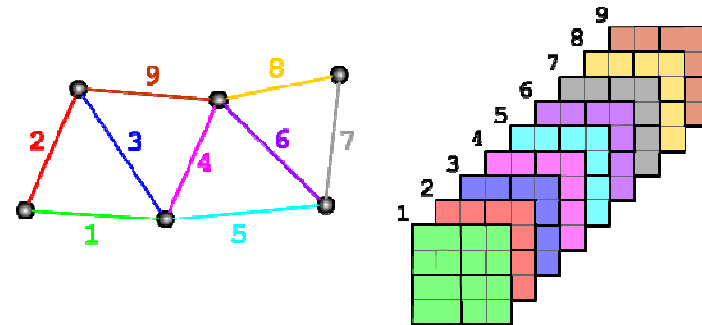


## EDS: Edge-by-Edge

- A closer look into EBE and EDS computational complexity:



$t(\text{ngl} \times 3, \text{ngl} \times 3, \text{nel})$



$t(\text{ngl} \times 2, \text{ngl} \times 2, \text{nedges})$

### Computational cost

Data structure	Memory	Flop	i/a
Element	1056 <i>nnodes</i>	2112 <i>nnodes</i>	1408 <i>nnodes</i>
Edge	224 <i>nnodes</i>	448 <i>nnodes</i>	448 <i>nnodes</i>

Linear tets


u-p fully coupled incompressible flow (4 *dofs*)

$\text{nel} \approx 5.5 \times \text{nnodes}$ ,  $\text{nedges} \approx 7 \times \text{nnodes}$ .

# EDS matrix-vector product (SERIAL)

## Edges (1 degree of freedom)

```
do ie = 1, nedges
  neq1 = lm(1,ie)
  neq2 = lm(2,ie)
  ...
  retrieve and multiply 4 coefs.
  ...
  p(neq1) = p(neq1) + ap1
  p(neq2) = p(neq2) + ap2
end do
```



### STORAGE REQUIREMENTS:

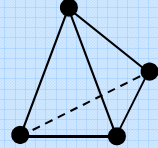
For 4 degrees of freedom:

Elements: (*nel* x 192 coefs.) + BDiag

Edges: (*nedges* x 32 coefs) + BDiag

## Elements (1 degree of freedom)

```
do ie = 1, nel
  neq1 = lm(1,ie)
  neq2 = lm(2,ie)
  neq3 = lm(3,ie)
  neq4 = lm(4,ie)
  ...
  retrieve and multiply 16 coefs
  ...
  p(neq1) = p(neq1) + ap1
  p(neq2) = p(neq2) + ap2
  p(neq3) = p(neq3) + ap3
  p(neq4) = p(neq4) + ap4
end do
```



# Further Improvements n EDS Matvec Product

- **Gijzen Modified:**

$$\mathbf{A}\mathbf{p} = \mathbf{B}(\mathbf{A})\mathbf{p} + \sum_{s=1}^{nedges} \mathbf{A}^s \left[ \mathbf{A}^s - \mathbf{B}(\mathbf{A}^s) \right] \mathbf{p}^s$$

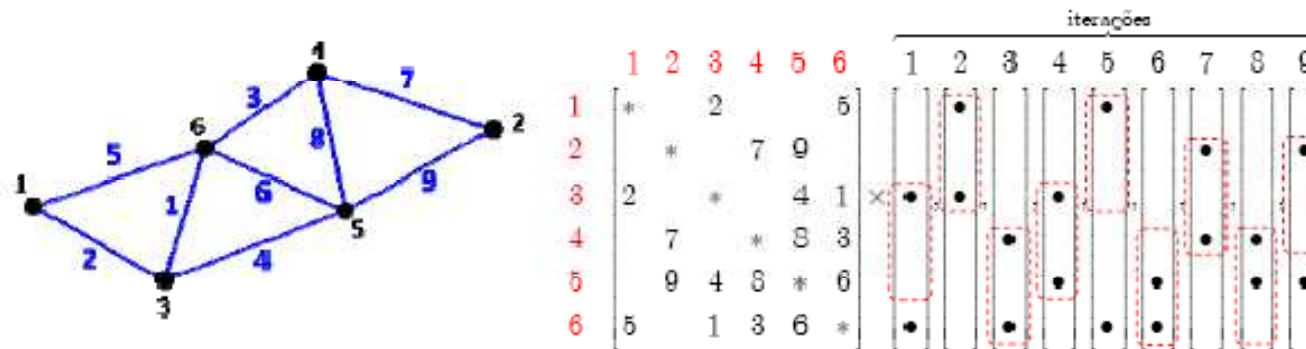
- **Note that this modification imply in storing just off-diagonal edge coefficients, which are global coefficients;**
- **B stores the nodal-block diagonals, which are global**

# Reordering Graph in Unstructured Grid Computations

- ❑ Improve cache utilization
- ❑ Minimize data movement in memory hierarchy
- ❑ Improve data locality
- ❑ Minimize indirect addressing effects
- ❑ Reorder nodes and edges
- ❑ Maximize processor performance

Coutinho, Martins, Sydenstricker, Elias. *Performance comparison of data reordering algorithms for sparse matrix-vector multiplication in edge-based unstructured grid computations*, IJNME, 2006.

# Data Locality Effects in EDS Matvec Product

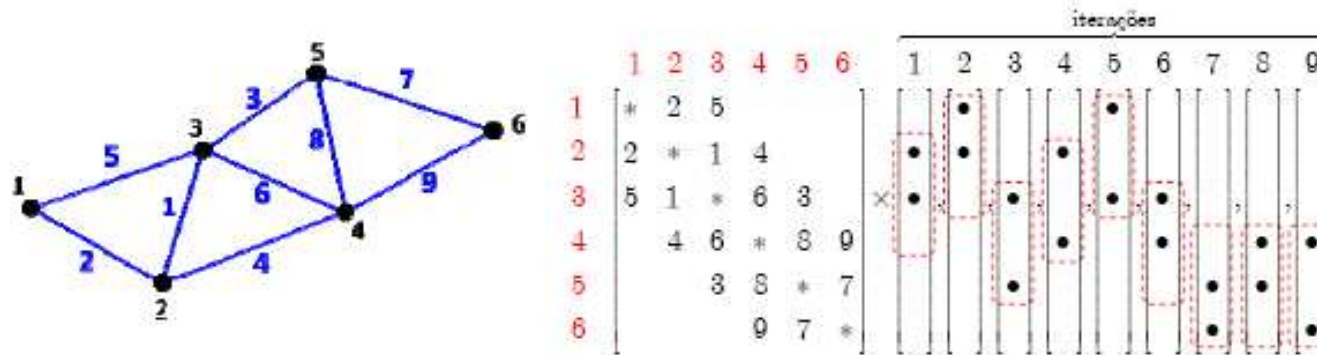


Original mesh ordering



Edge reordering only

# Data Locality Effects in EDS Matvec Product



Node reordering only



Edge and node reordering

# EdgePack<sup>©</sup>

## □ Overview

- Parallel data reordering package for optimizing edge-based computations in unstructured grids
- Develop an automatic procedure to determine which data combination for a given computer platform is the best one in terms of processing time
- Build a matrix-vector product routine library for symmetric and non-symmetric matrices for tetrahedral and hexahedral elements based on nodal renumbering and edge renumbering algorithms
- Support user on quickly developing finite element codes portable on parallel (shared, distributed or both) and multi-core platforms
- See:
  - Marcos Martins, Renato Elias, Alvaro Coutinho, EdgePack: A Parallel Vertex and Node Reordering Package for Optimizing Edge-Based Computations in Unstructured Grids, LNCS, 4395: 292-304, 2007

# EdgePack<sup>©</sup>

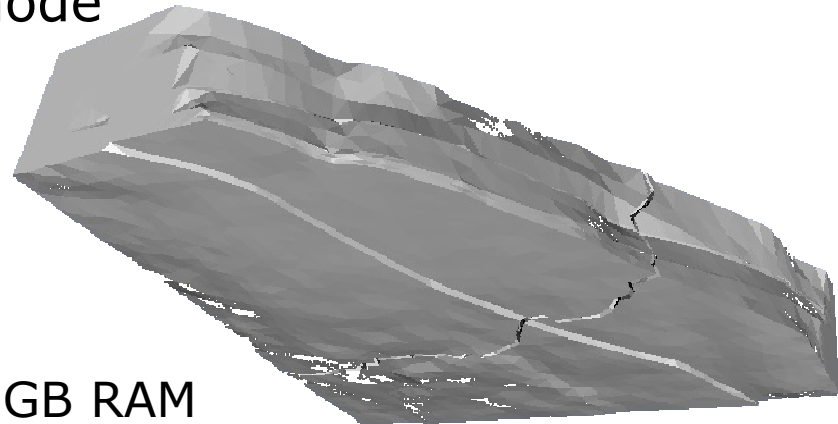
## □ Structure

- EdgePackPre
  - Performs nodal and edge orderings according to the number of degrees of freedom per node
  - Data locality and reuse
  - Data prepared for serial or parallel processing (shared or distributed memory platform)
  - Mesh partition for parallel processing
- Probe
  - Determines the best data configuration experimentally out of hundreds of options for current platform by sampling the time spent for matrix-vector products and element matrix disassembling into edges
  - Results used to mesh weighting
- EdgePackPro
  - Library of routines built in Fortran90 for implementation of finite element code based on edges
  - Based on MPI and OpenMP directives



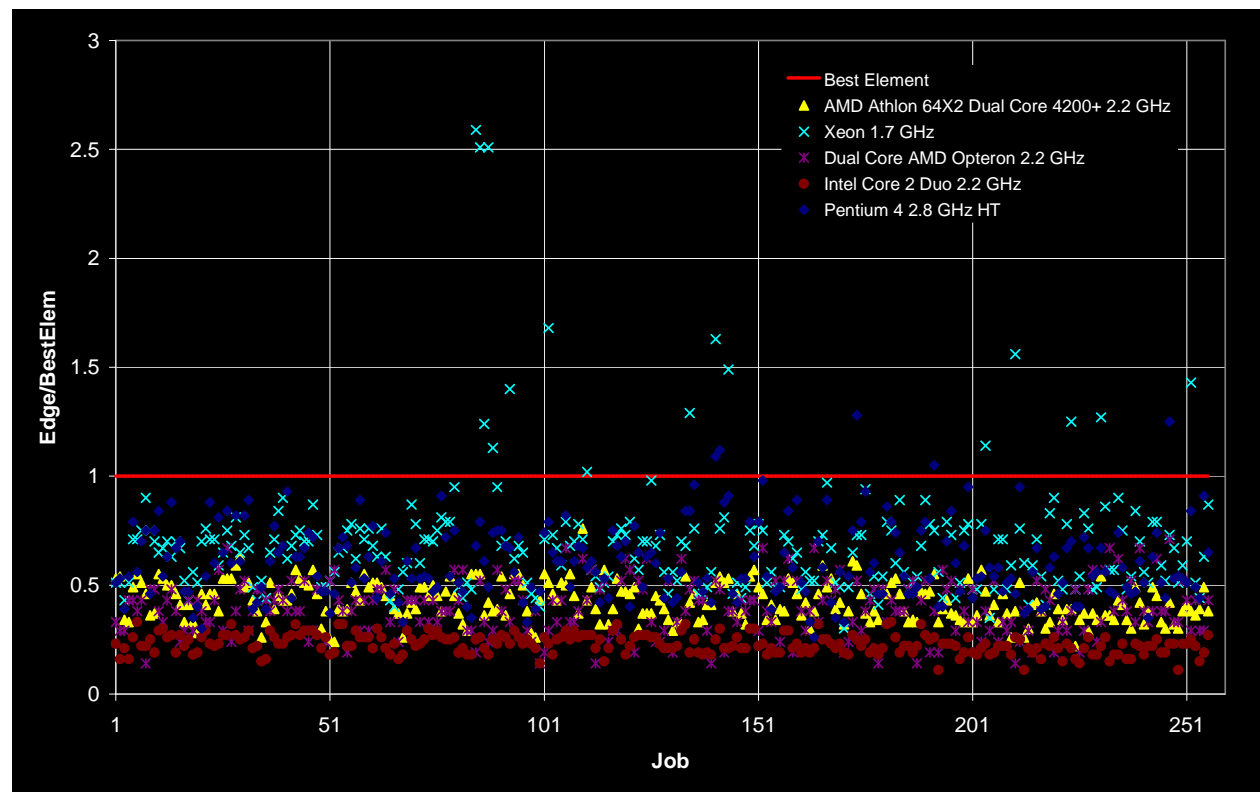
# EdgePack - EdgeTRM

- **Transient thermal conduction through a sedimentary basin (South America)**
  - Linear and symmetric operator
  - 1 degree of freedom per node
  - Mesh
    - 26,253 Nodes and 141,766 Elements
  - Computational platforms
    - Xeon 1.7 GHz, 4 GB RAM
    - Pentium 4 2.8 GHz HT, 2 GB RAM
    - Intel Core 2 Duo 2.2 GHz, 2 GB RAM
    - AMD Athlon 64X2 Dual Core 4200+ 2.2 GHz, 2 GB RAM
    - Dual Core AMD Opteron 2.2 GHz, 16 GB RAM



# EdgePack - EdgeTRM

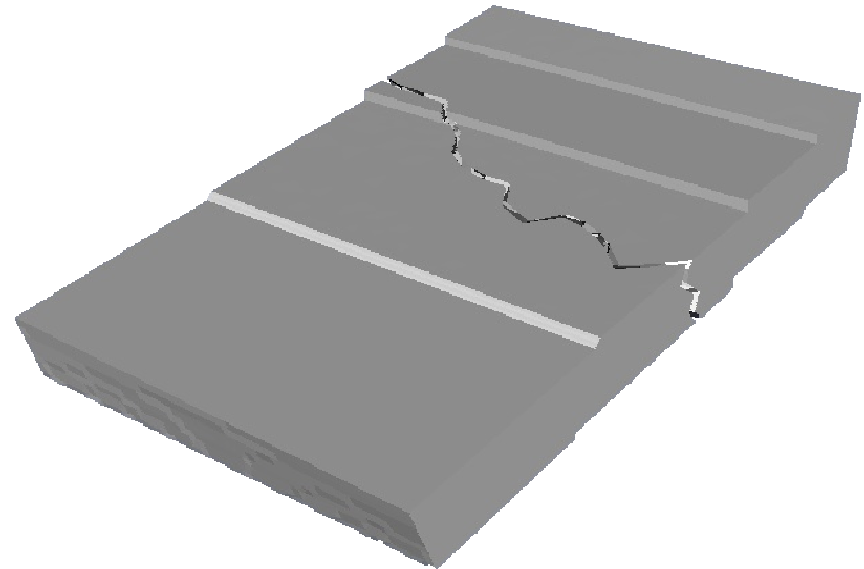
- **Transient thermal conduction through a sedimentary basin**
  - Data probing results: rank 0



# EdgePack - EdgeCSM

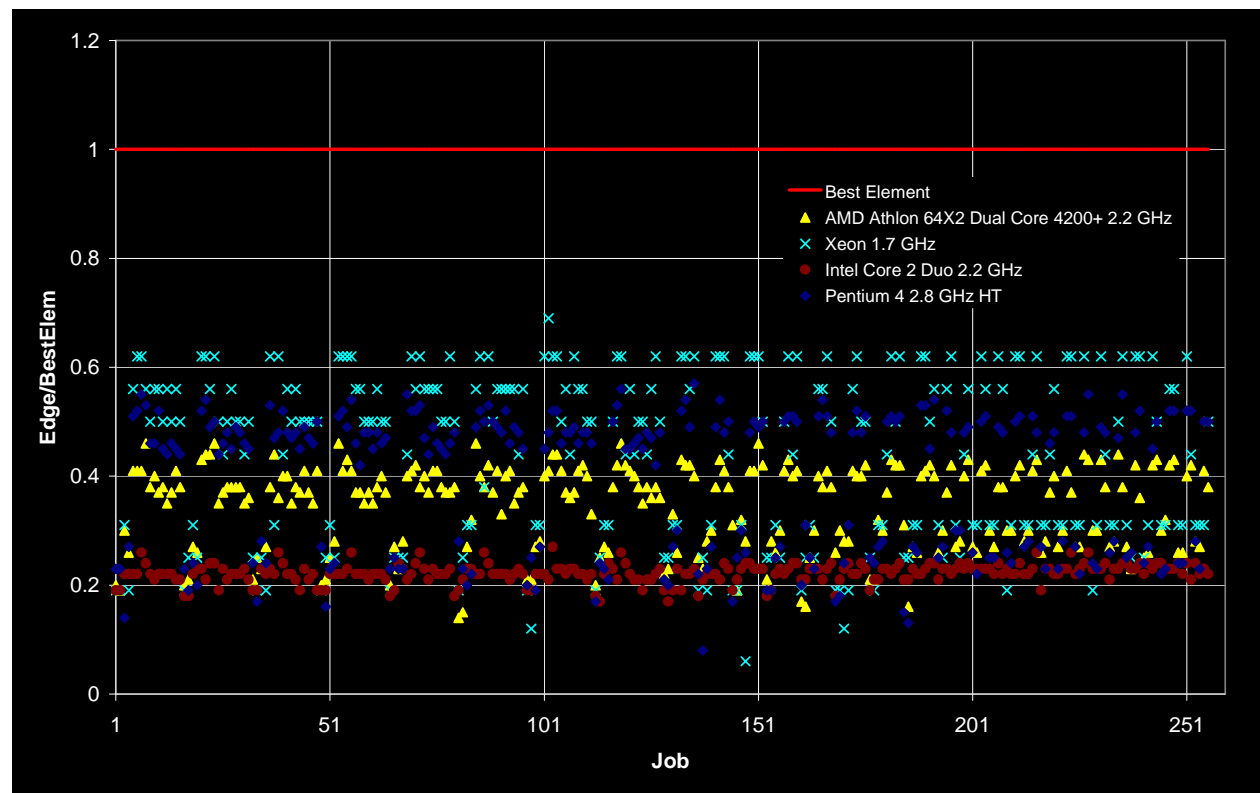
## □ **Elasto-plastic behavior of a sedimentary basin (South America)**

- Non-linear and symmetric operator
- 3 DOF per node
- Mesh
  - 28,814 Nodes and  
136,738 Elements
- Computational platforms
  - Xeon 1.7 GHz, 4 GB RAM
  - Pentium 4 2.8 GHz HT, 2 GB RAM
  - Intel Core 2 Duo 2.2 GHz, 2 GB RAM
  - AMD Athlon 64X2 Dual Core 4200+ 2.2 GHz, 2 GB RAM



# EdgePack - EdgeCSM

- **Elasto-plastic behavior of a sedimentary basin**
  - Data probing results



# **PART II: LET'S GO PARALLEL**

# Parallelizing computational mechanics codes

## □ **Basic concerns:**

- What to parallelize?
- What parallel model to adopt?
- What the implications of the chosen parallel model?
  - Which architecture the program will run?
  - What data structures?
  - What about efficiency? What about scaling?
  - Is it easy to implement and maintain?
- Should I adapt an existing code or better from the scratch?

## □ **Hands-on!**

- Where do I begin?
- How am I sure that I'm getting the right answers?

# What have to be parallelized?

- ❑ **Rule of thumb:** the same code should run in serial and parallel mode! *"... A serial program is just a parallel program running in 1 processor, process, task, thread ..."*
- ❑ **We should consider:**
  - All the program will be parallel or just some parts?
  - What parts deserve to be parallel? *"... consider computational effort and amount of data ..."*
  - Do the selected algorithms and methods can be efficient parallelized?
  - Reads and writes are parallel?
  - KISS programming approach

# What parallel model ?

- ❑ **Threaded parallelism / OpenMP: =**

  - “Easy” to implement, low scalability, problems with memory dependences
  
- ❑ **Distributed memory – MPI:**

  - “Hard” to implement; high scalability, data should be partitioned among processors, load balacing maybe problematic.
  
- ❑ **Hybrid: MPI+OpenMP**

  - Difficult to implement, flexible, memory dependences and partitioning should be taken care



# Adapt an existing code or better from the scratch?

## □ That depends:

- What is the program size?
- How complex is the code? *"... codes that most of computational effort are on few sections (routines, objects) may benefit from parallelism by small strategic interventions without drastic modifications ..."*
- Is the program documented?
- Is the documentation really good?
- Conversion will be done by the same programmer of the original serial code?

## □ Consider:

- Is it really worth begin from the scratch?! *"... sometimes adaption takes longer than recreation ..."*
- Start from the scratch adding components from the old code;
- Start from the scratch with new technologies;

- **Parallelism is a strong factor to be considered in the design of new software.** *"... Even though the 1st version is serial, the programmer should be well aware that most probably this have to be done in the future and provide enabling mechanisms to support the migration..."*

# Where to start?

- Consider again the major components of our FE code

Pre-processing

Input data

Time integration loop

Nonlinear iteration loop

Form system of linear equations

Solve system of linear equations

End NL loop

Update time step

Output results

End time loop

Post-processing

Visualization

Jacobian, residual

Matvec, preconditioning

# Where to start?

## □ What all FE problems generally do?

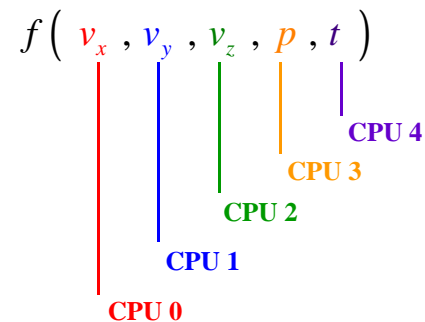
- We evaluate integrals to generate stiffness matrices and residuals; that depends basically on what element and how many we have;
  - The inner part of this program usually solves a linear system of equations; with iterative methods that means matvecs and preconditioning;
  - System solves are repeated many times
    - CONCLUSION: [real gains can be obtained accelerating system generation and solution and/or reducing how many times it is solved](#)
  - Computational effort is directly related to:
    - Solver algorithm
    - Discretization → number of unknowns, elements or edges
    - Underlying physics → number of degrees of freedom per node
    - Conditioning of linear system of equations
    - Other factors → data locality for instance
- that is, we found some points to explore!

# Some ways to think about parallelism

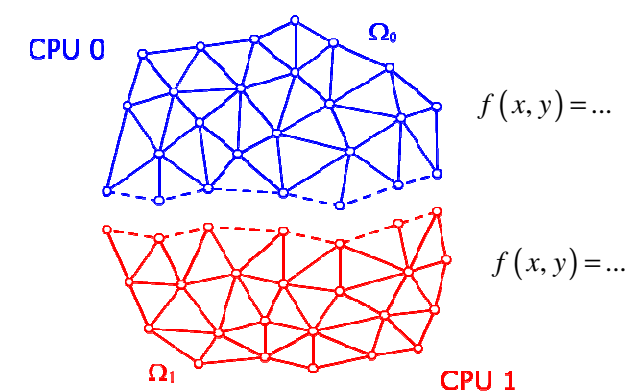
## Functions

$f_1(x, y, z, t) = \dots \rightarrow \text{CPU 0}$   
 $f_2(x, y, z, t) = \dots \rightarrow \text{CPU 1}$   
 $\vdots$   
 $f_n(x, y, z, t) = \dots \rightarrow \text{CPU } n$

## Degrees of freedom

$f(v_x, v_y, v_z, p, t)$   


## Domain



- In the first example several functions (tasks) run at the same time; if the function is the whole FE code, we have what is called now Many Task Computing parallel model.
- In the second example we are limited by the number of degrees-of-freedom
- In the third example we assign a part of the domain to each processor and the FE code runs on every part. However interface data have to be updated

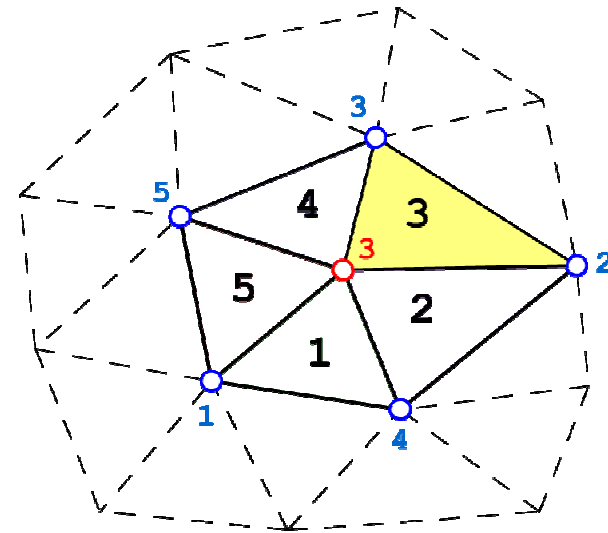
## OpenMP in FE codes

- ❑ **OpenMP can be easily adopted by any existing FE code, once all memory dependencies problems are removed from the code;**
- ❑ **Any loop with no memory dependencies can be parallelized with OpenMP;**
- ❑ **It's better to insert OpenMP directives incrementally, starting from the most computationally intense loop and checking efficiency and results after each intervention;**

# Memory Dependency in OpenMP

- Consider the following element loop executed in 2 parallel threads:

```
C$OMP PARALLEL DO
do i=1,nel
    ! Recover element nodes
    x(no) = x(no) + a
enddo
C$OMP END PARALLEL DO
```

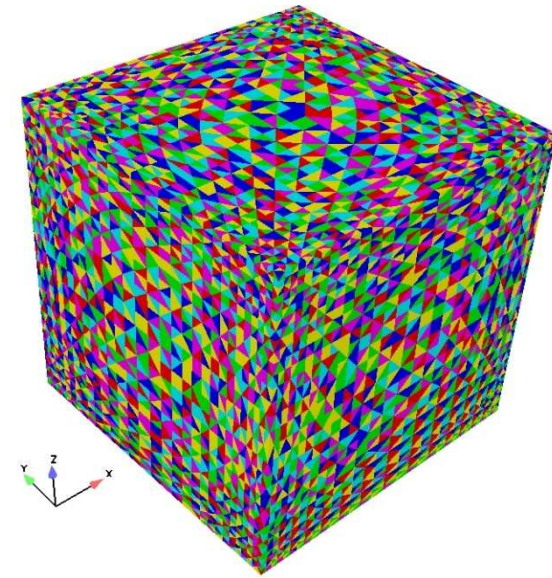


- In the above loop the 2 threads are **COMPETING** to modify the contents of x for node **3** since each thread is working in its element and its data.
- CONCLUSION:** In the above loop threads cannot share information related to node **3**

# Solving memory dependence in unstructured grids

- ❑ To solve the memory dependence problem in unstructured grids we just have to split the elements in the mesh in blocks that do not share any node.
- ❑ This procedure is known as mesh coloring or mesh blocking.

```
ielm = 0
do icor = 1, ncores
  nvec = ielblk(icor)
  C$OMP PARALLEL DO
  C$OMP& FIRSTPRIVATE(NVEC)
    do i = ielm+1, ielm+nvec
      ! Recover element nodes
      x(no) = x(no) + a
    enddo
  C$OMP END PARALLEL DO
  ielm = ielm+nvec
enddo
```



- ❑ Mesh coloring affects node reordering
- ❑ We use a greedy algorithm to color the mesh;

# EBE/EDS Matvec with OpenMP

## □ Exemple: EDS with 1 degree of freedom

```

iside = 0
DO iblk = 1, nedblk
  nvec = iedblk(iblk)
  !DIR$ IVDEP
  !$OMP PARALLEL DO
    DO ka = iside+1, iside+nvec, 1
      neq1 = lm(1,ka)
      neq2 = lm(2,ka)
      ...
      ! retrieve and multiply 4 coefs.
      ...
      p(neq1) = p(neq1) + ap
      p(neq2) = p(neq2) + ap
    ENDDO
  !$OMP END PARALLEL DO
  iside = iside + nvec
ENDDO

```

In some compilers tells that the code below have no memory dependencies



# CSR matvec with OpenMP

```

!$OMP PARALLEL DO
DO i = 1, n
    k1 = ia(i)
    k2 = ia(i+1)-1
    y(i) = dotproduct(a(k1:k2),x(ja(k1:k2)))
ENDDO
!$OMP END PARALLEL DO
    
```

$$\mathbf{A} = \begin{bmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{bmatrix}$$

AA(nnz)=	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.
JA(nnz)=	1	4	1	2	4	1	3	4	5	3	4	5
IA(n+1)=	1	3	6	10	12	13						
	1	2	3	4	5	6						

# Parallelism in Distributed Memory Machines

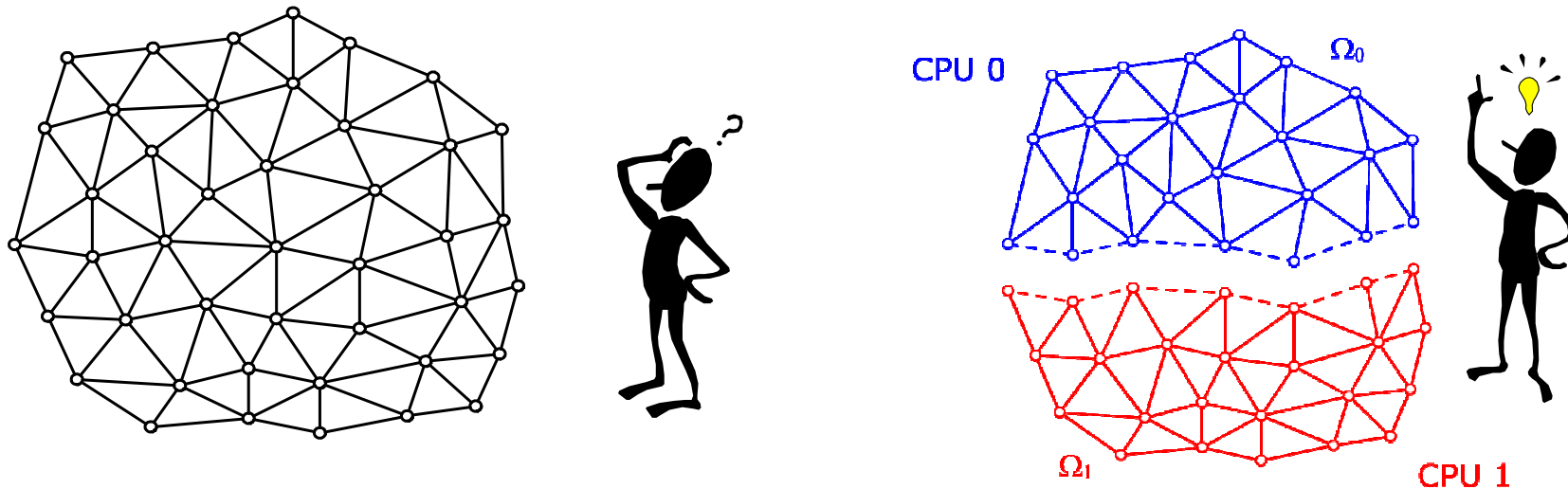
## **M**essage **P**assing **I**nterface

# Generalities

- ❑ Easy to understand → hard to implement
- ❑ Although designed for distributed memory systems, also works on shared memory machines, making the approach very flexible and portable;
- ❑ Basically convert a big problem into several small ones and assign each small problem to a processor (process);
- ❑ Many **identical copies** of the same program are executed simultaneously;
- ❑ Every copy acts independently only in its part of the problem;
- ❑ **Common parts are synchronized** by message passing operations using routines available in MPI;
- ❑ Thus we need to take care of explicit coding of information exchanging among several processes using MPI functions (MPI\_BROADCAST, MPI\_SEND, MPI\_ALLREDUCE, ...)

# What and how?

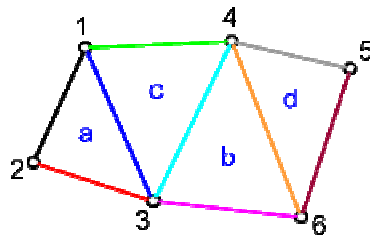
- The most intuitive way of applying MPI to computational mechanics is by partitioning the problem domain into subdomains.



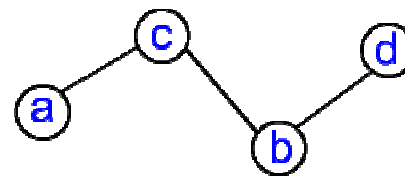
**Conclusion: we need to learn how to partition a given problem**

# Mesh (and graph) Partitioning

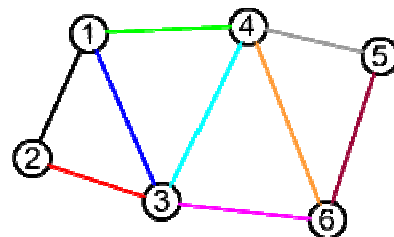
- Consider the mesh below and its associated graphs and sparse matrix



Mesh



Dual graph

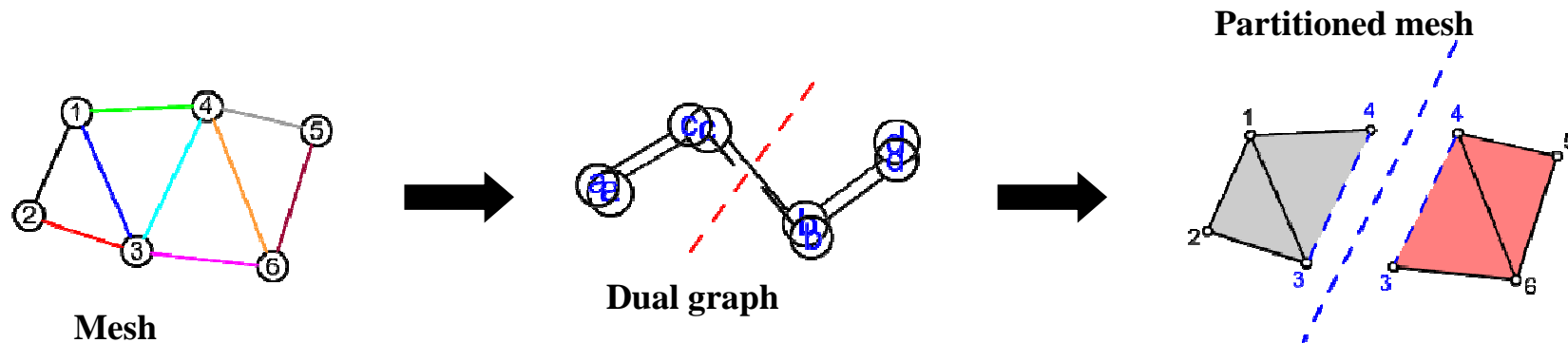


Nodal graph

11	12	13	14		
21	22	23			
31	32	33	34		36
41		43	44	45	46
			54	55	56
		63	64	65	66

matrix

# Dual Partition (or by element)



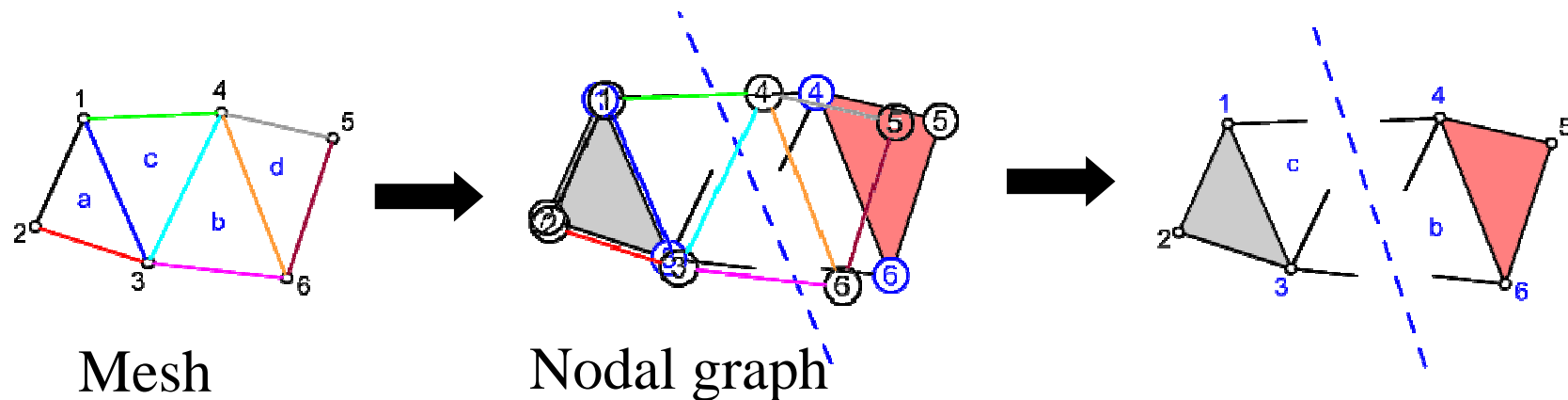
Note that nodes 3 and 4 belong to both sub-mesh simultaneously, that is, these nodes belong to the interface between the 2 partitions;

In practice we need to keep interface information compatible (equal or sincronized)

**Conclusion:** Communication volume (that is, the amount of information sent/received) is directly proportional to the number of interface nodes

## Nodal Partition

- **Alternatively, we can partition the nodal graph. In practice the choice depends on the underlying data structures (element based/node based)**



# Important Aspects on Mesh Partitioning

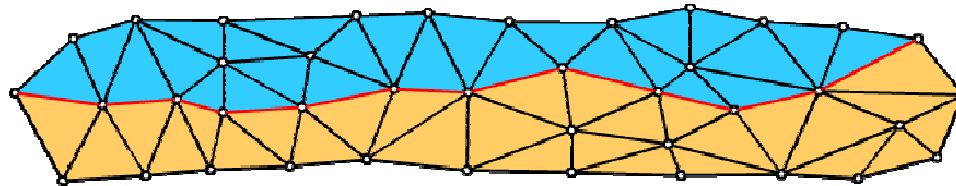
- ❑ **A good partitioning should:**

  - Minimize communication volume while maintaining load balancing;

    - Minimize communication volume means minimize the number of edge cuts in the graph (nodal or dual) ;
    - Balancing load means distribute uniformly the number of vertices in each partition;
  - NOTE: in heterogeneous systems, with different machines, partition should take this into account, generally by assigning different weights to the graph vertices.

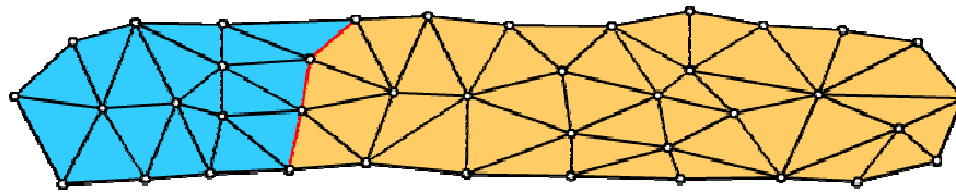


# Partitioning Quality



**HIGH communication**

**Load BALANCED**



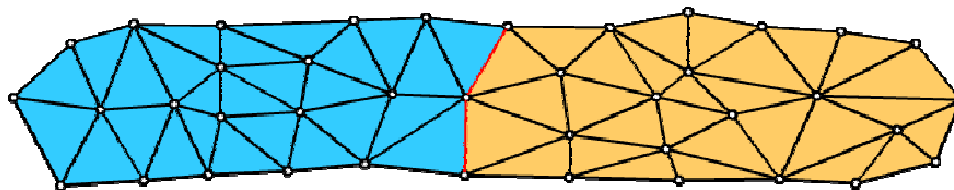
**LOW communication**

**Load (UN)BALANCED**

**Good or bad?! Depends on the system...**

Pentium 4

Nehalem



**LOW communication**

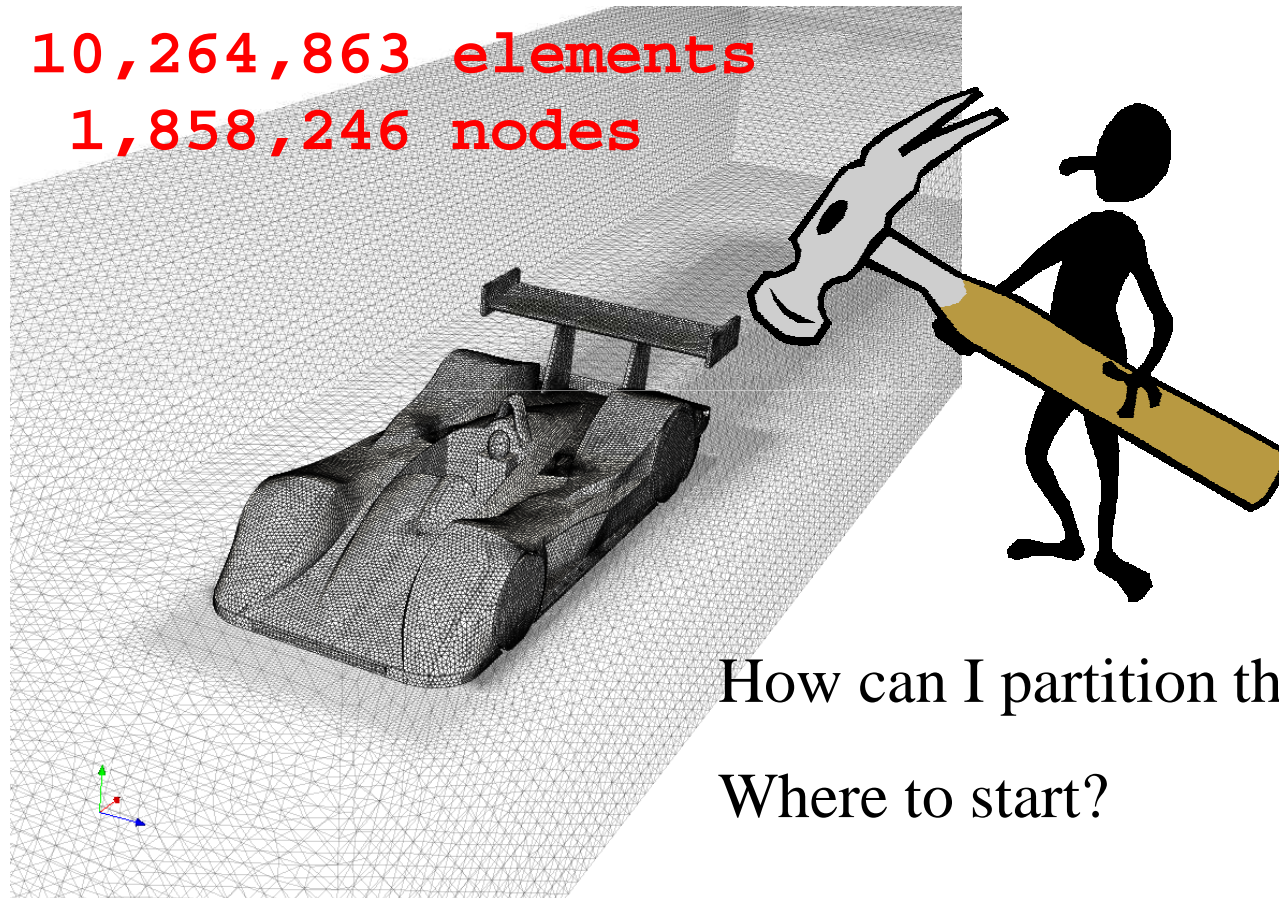
**Load BALANCED**

Nehalem

Nehalem

# Partitioning in practice

10,264,863 elements  
1,858,246 nodes



How can I partition this mesh?

Where to start?

## Software for mesh partitioning

- **Chaco**
  - <http://www.cs.sandia.gov/~bahendr/chaco.html>
- **Jostle**
  - <http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/>
- **Metis/ParMetis**
  - <http://glaros.dtc.umn.edu/gkhome/views/metis/index.html>
- **PARTY**
  - <http://wwwcs.uni-paderborn.de/fachbereich/AG/monien/RESEARCH/PART/party.html>
- **Scotch**
  - <http://www.labri.fr/perso/pelegrin/scotch/>
- **S-Harp**

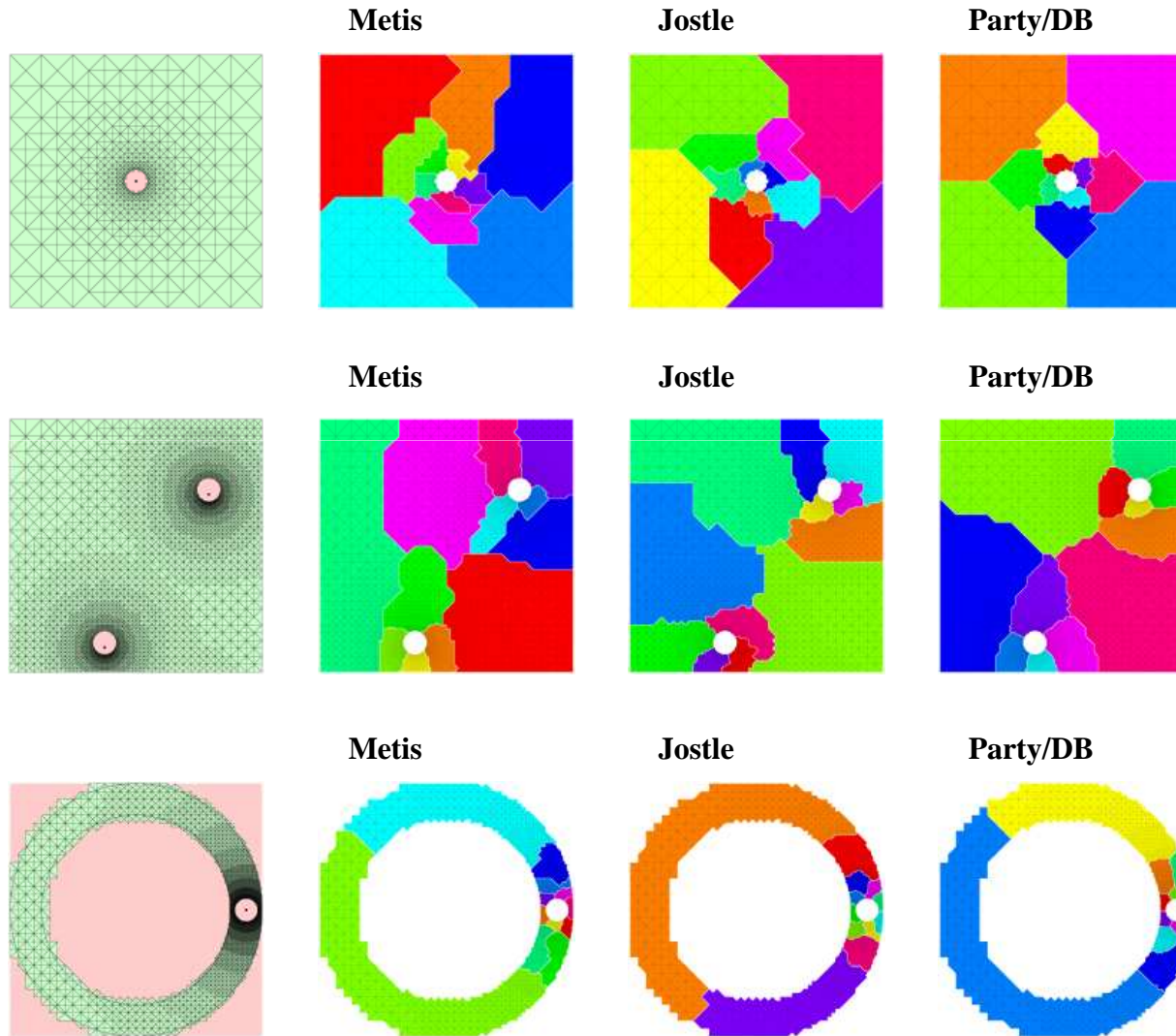
	Chaco	Jostle	Metis	ParMetis	PARTY	SCOTCH	S-HARP
Geometric Schemes	•				•		•
Coordinate Nested Dissection					•		
Recursive Inertial Bisection	•						•
Space-filling Curve Methods				•			
Spectral Methods	•				•		•
Recursive Spectral Bisection	•						
Multilevel Spectral Bisection	•						
Combinatorial Schemes	•				•	•	
Levelized Nest Dissection					•	•	
KL/FM	•				•	•	
Multilevel Schemes	•	•	•	•	•	•	
Multilevel Recursive Bisection	•		•	•		•	
Multilevel k-way Partitioning		•	•	•			
Multilevel Fill-reducing Ordering			•	•			
Dynamic Repartitioners		•		•			•
Diffusive Repartitioning		•		•			•
Scratch-Remap Repartitioning				•			
Parallel Graph Partitioners		•		•			•
Parallel Static Partitioning		•		•			•
Parallel Dynamic Partitioning		•		•			•
Other Formulations		•	•	•			
Multi-constraint Graph Partitioning		•	•	•			
Multi-objective Graph Partitioning			•				

# Comparing algorithms for mesh partition



	Number of Trials	Needs Coordinates	Quality	Local View	Global View	Run Time	Degree of Parallelism
Recursive Spectral Bisection	1	no	●●●●	○	●●●●	■■■■	▲▲
Multilevel Spectral Bisection	1	no	●●●●	○	●●●●	■■■	▲▲
Multilevel Spectral Bisection-KL	1	no	●●●●●●	●●	●●●●	■■■	▲▲
Multilevel Partitioning	1	no	●●●●●●	●●	●●●●	■■	▲▲
Levelized Nested Dissection	1	no	●●	○	●●	■■	▲▲
Kernighan-Lin	1	no	●●	●●	○	■■	▲
	10	no	●●●●○	●●	●●○	■■■	▲▲
	50	no	●●●●	●●	●●	■■■■	▲▲
Coordinate Nested Dissection	1	yes	●	○	●	■	▲▲▲
Recursive Inertial Bisection	1	yes	●●	○	●●	■	▲▲▲
Recursive Inertial Bisection-KL	1	yes	●●●●	●●	●●	■■	▲
Geometric Sphere-cutting	1	yes	●●	○	●●	■	▲▲▲
	10	yes	●●●●○	○	●●●●○	■■	▲▲▲
	50	yes	●●●●	○	●●●●	■■■	▲▲▲
Geometric Sphere-cutting-KL	1	yes	●●●●	●●	●●	■■	▲
	10	yes	●●●●●○	●●	●●●●○	■■■	▲▲
	50	yes	●●●●●●	●●	●●●●	■■■■	▲▲

# Comparing partitioning software



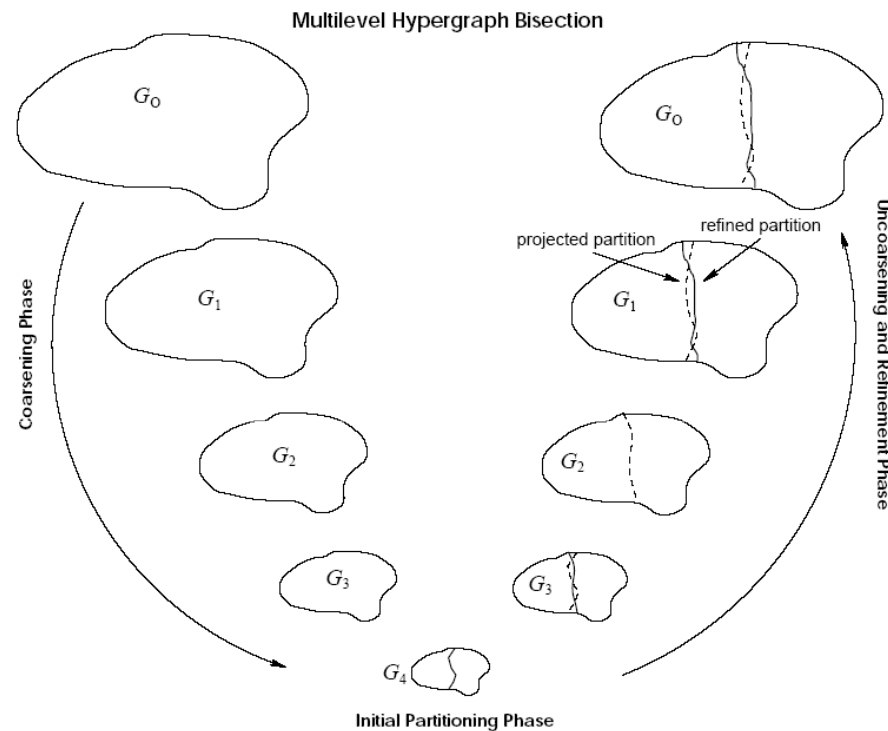
# Metis

## □ What is Metis?

- **Graph** and **mesh** partitioner;
- It has algorithms for bandwidth reduction;
- Extremely fast;
- Partitions are **balanced** or **not (weighted partitioning)**;
- Based on methods that minimize the number of edge cuts;
- Written in **C** with **Fortran** interface;
- Parallel version available (**ParMetis**);
- **Free and open software.**
- Further info: <http://www-users.cs.umn.edu/~karypis/metis/>
- Can be used as stand-alone or library



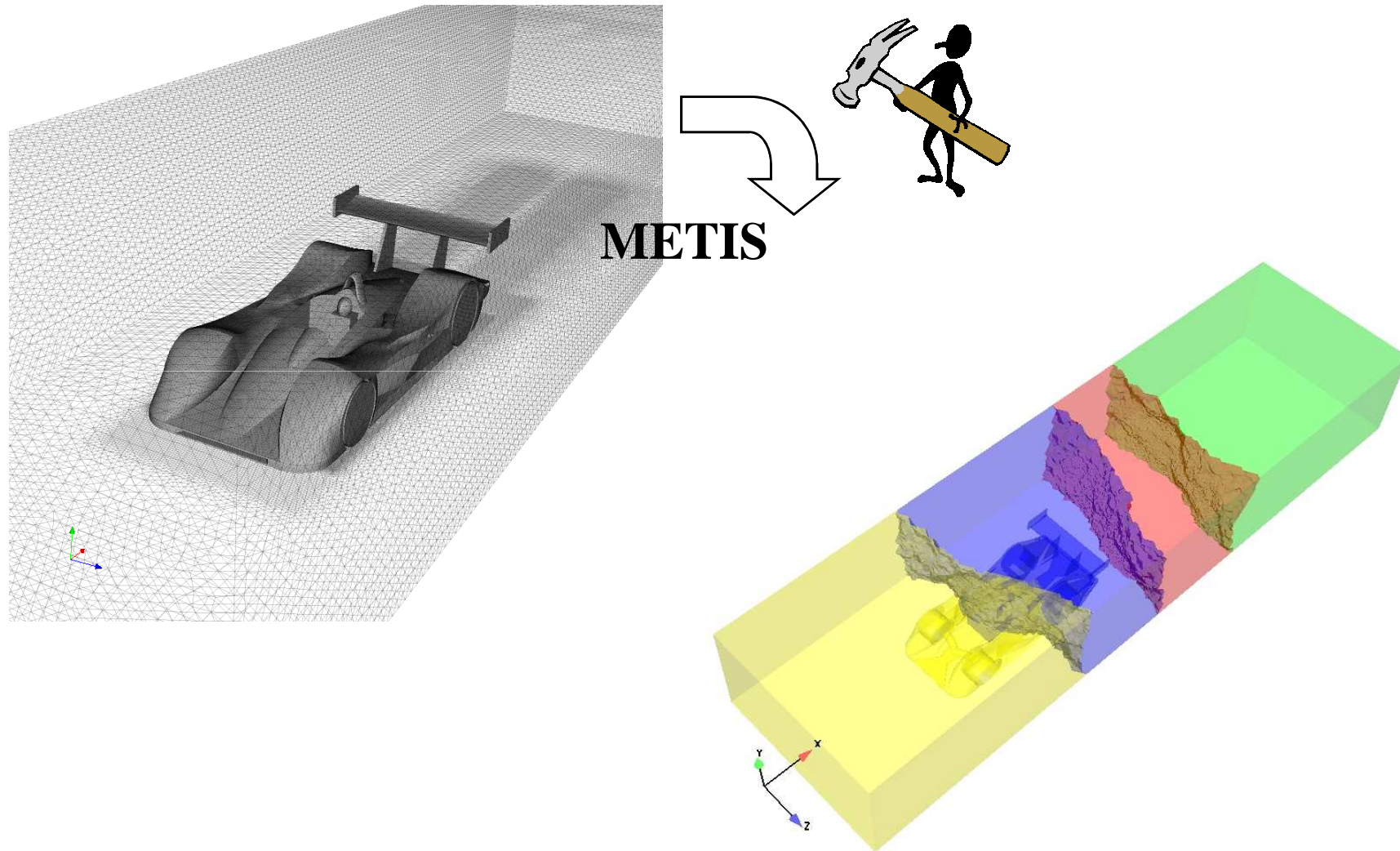
# Graph Partitioning for Distributed Memory Machines



*Figure 1.2.* The various phases of the multilevel hypergraph bisection. During the coarsening phase, the size of the hypergraph is successively decreased; during the initial partitioning phase, a bisection of the smaller hypergraph is computed; and during the uncoarsening and refinement phase, the bisection is successively refined as it is projected to the larger hypergraphs. During the uncoarsening and refinement phase, the dashed lines indicate projected partitionings and dark solid lines indicate partitionings that were produced after refinement.

METIS: <http://www-users.cs.umn.edu/~karypis/metis/index.html>

# Mesh Partitioned!





# Components of Metis Library

- **Graph partitioning:**
  - METIS\_PartGraphRecursive;
  - METIS\_PartGraphKway;
  - METIS\_PartGraphVKway;
  - METIS\_mCPartGraphRecursive;
  - METIS\_mCPartGraphKway;
  - **METIS\_WPartGraphRecursive;**
  - **METIS\_WPartGraphKway;**
  - METIS\_WPartGraphVKway;
- **Mesh partitioning:**
  - **METIS\_PartMeshNodal;**
  - **METIS\_PartMeshDual;**
- **Sparse matrices reordering**
  - METIS\_EdgeND;
  - METIS\_NodeND;
  - METIS\_NodeWND
- **Auxiliary routines:**
  - METIS\_MeshToNodal;
  - **METIS\_MeshToDual;**
  - METIS\_EstimateMemory.

# Using Metis (where to start?)

## ❑ Download files in:

- <http://glaros.dtc.umn.edu/gkhome/metis/metis/download>

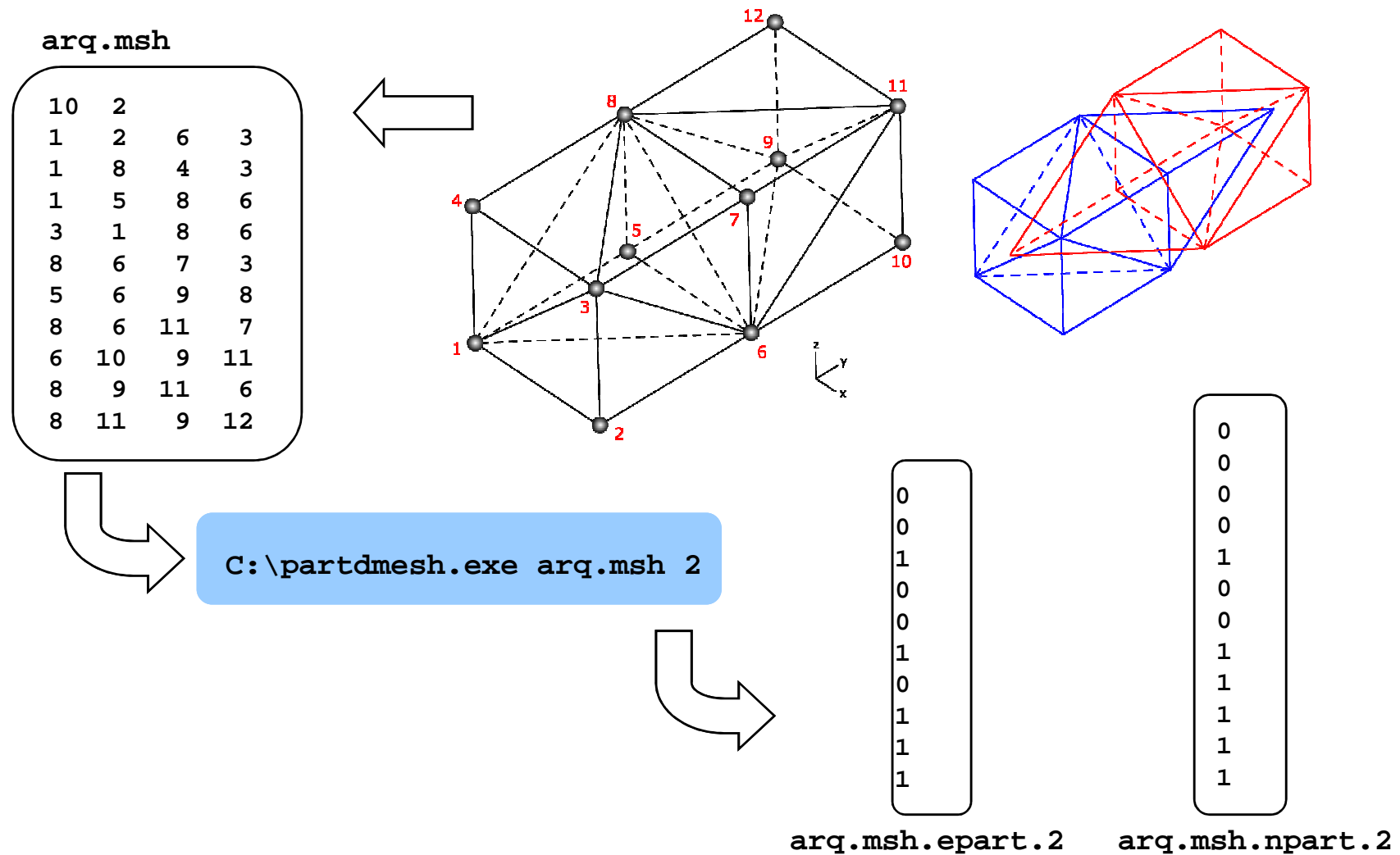
### Metis-4.0.zip (precompiled for Win32)



### Metis-4.0.tar.gz (source code)



## Using Metis in stand-alone mode (Practical example)



# Metis with FORTRAN

## *(using the library)*

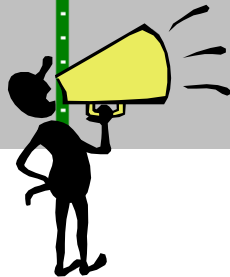
```

subroutine FragMesh(log)

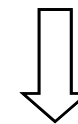
use files
use sizes
use meshdata
use Partitions
use libmetis
implicit double precision (a-h, o-z)
    
```

```

call METIS_PartMeshDual(
    nel , ! number of elements
    nnos , ! number of nodes
    ien , ! element nodal connectivity
    2 , ! Type of element(2=tetrahedra)
    1 , ! 1= Fortran style
    nprocs , ! number of partitions
    ncutedg , ! Amount of edge cut
    iel_p , ! Element partitions
    ino_p ) ! Node partitions
    
```



file libmetis.a



```

c.
c.  Interface module used by METIS library
c.
MODULE libmetis
INTERFACE
    ! PARTMESHDUAL ROUTINE
    SUBROUTINE METIS_PartMeshDual ( ne      ,
                                   nn      ,
                                   elmnts ,
                                   etype  ,
                                   numflag,
                                   nparts ,
                                   edgecut,
                                   epart  ,
                                   npart  )

!DEC$ ATTRIBUTES C :: METIS_PartMeshDual
!DEC$ ATTRIBUTES REFERENCE :: ne
!DEC$ ATTRIBUTES REFERENCE :: nn
!DEC$ ATTRIBUTES REFERENCE :: elmnts
!DEC$ ATTRIBUTES REFERENCE :: etype
!DEC$ ATTRIBUTES REFERENCE :: numflag
!DEC$ ATTRIBUTES REFERENCE :: nparts
!DEC$ ATTRIBUTES REFERENCE :: edgecut
!DEC$ ATTRIBUTES REFERENCE :: epart
!DEC$ ATTRIBUTES REFERENCE :: npart

    INTEGER :: ne, nn, etype, numflag, nparts,
               edgecut, elmnts(1:4,1:ne), epart(1:ne), npart(1:nn)
END SUBROUTINE METIS_PartMeshDual
END INTERFACE
END MODULE libmetis
    
```

Link your program with the library

```
ifort -o <meuprog> *.o libmetis.a
```

# FAQ (Frequently Asked Questions)

- ❑ **Does Metis partition hybrid meshes (more than one element type)?**

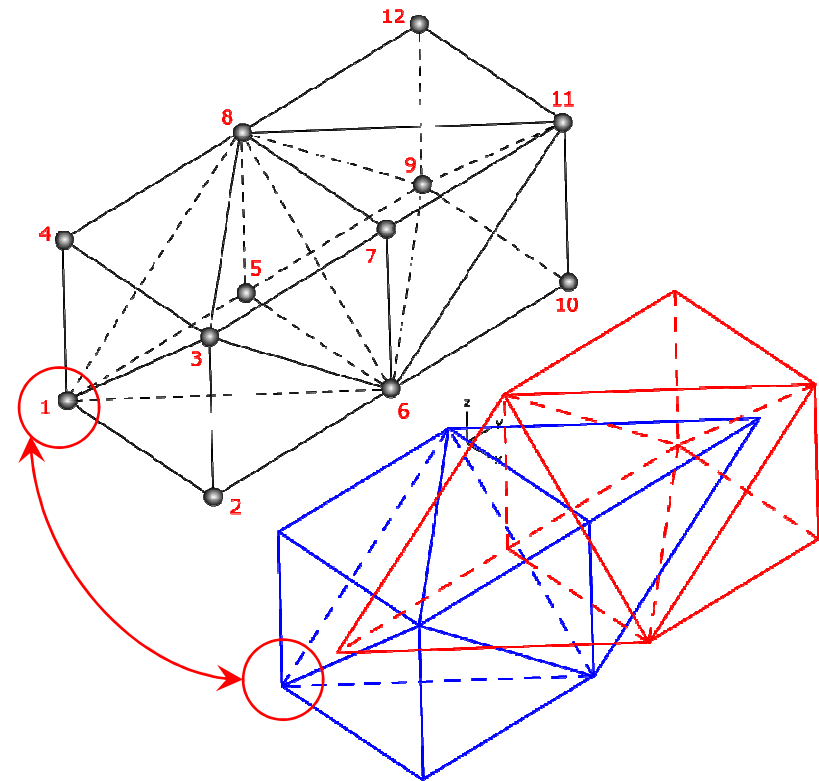
  - Yes and no: Metis partitions GRAPHS, that is, Metis will partition every mesh converted to either a nodal or dual graph.
  - OBS.: Metis parallel version, ParMetis, has native support for hybrid mesh partitioning.
- ❑ **How do I discover the interface nodes?**

  - Metis does not return explicitly this information. However, it's very simple to obtain. Every time a node is in a partition with number different from its element means that is an interface node.

# Metis: Discovering interface nodes

ie	nó1	nó2	nó3	nó4	part
1	1	2	6	3	0
2	1	8	4	3	0
3	1	5	8	6	1
4	3	1	8	6	0
5	8	6	7	3	0
6	5	6	9	8	1
7	8	6	11	7	0
8	6	10	9	11	1
9	8	9	11	6	1
10	8	11	9	12	1

nó	part
1	0
2	0
3	0
4	0
5	1
6	0
7	0
8	1
9	1
10	1
11	1
12	1

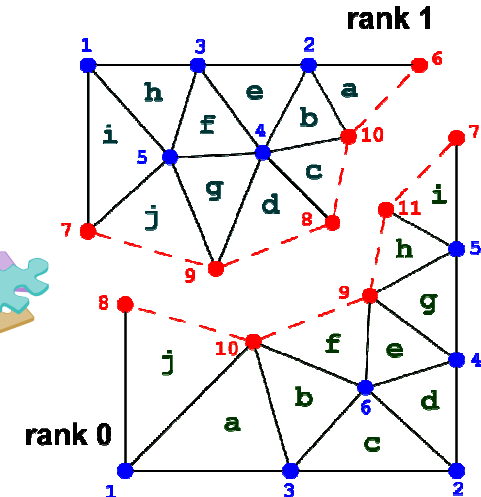
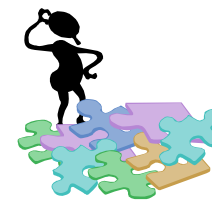
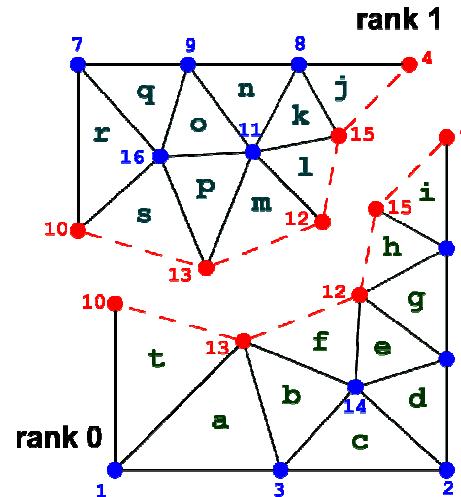
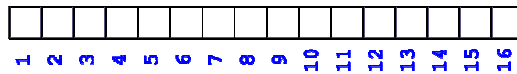
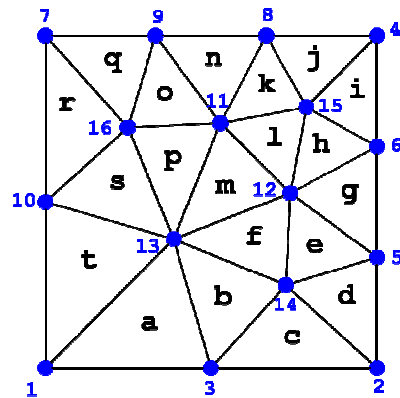


Note that node 1 belongs to partition 0 and belongs to element 3, that belongs to partition 1, that is, node 1 is an interface node!

## Beyond partitioning

- ❑ **After mesh partitioning the original problem is now divided into several smaller sub-problems, connected by the interface nodes;**
- ❑ **A good practice is to independently number every mesh entity (nodes, elements, edges) in each partition;**
- ❑ **There is several ways of performing this ordering. This is the simplest!**
- ❑ **Note that interface data will be replicated in each partition;**
- ❑ **We will see briefly later another way, much more complicated.**

# Reordering the partitions

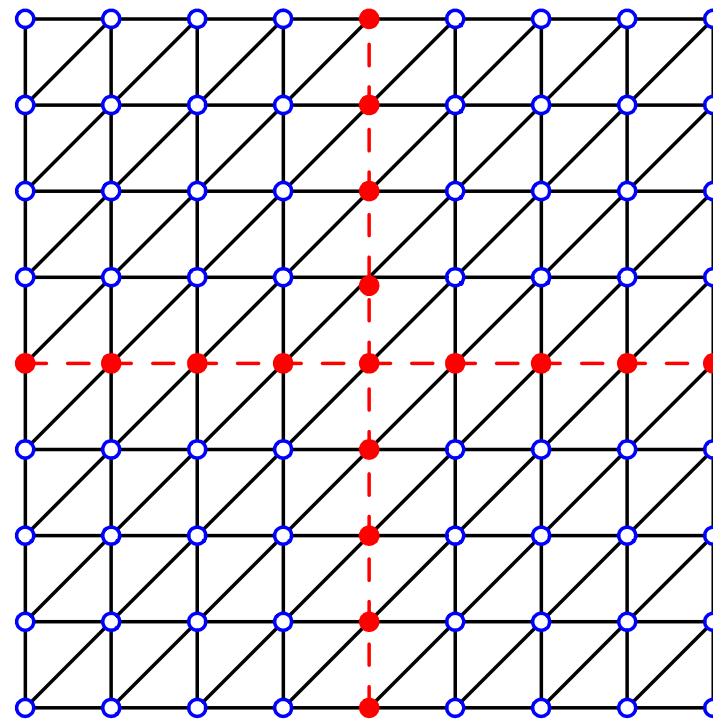


After this reordering we can treat each partition as a completely independent problem and update interface data just when needed.



# Interface Treatment

- ❑ **What is better?**
  - Exchange few big messages or a lot of small ones?



# Thinking in parallel

- ❑ **MPI programs normally have a master process: rank 0**
  - rank 0 normally controls main program flow
  - rank 0 must also be used to compute – ***"don't waste resources!!!"***
- ❑ **MPI programs should also work in serial mode (1 processor)**
- ❑ **Results from a parallel run should match the serial run results**
- ❑ **IMPORTANT TIP: Utilize pre-processing macros for producing 100% serial versions of your code (we you see examples)**

# Building MPI programs

- ❑ **Who does what?**
- ❑ **Who reads data?**
- ❑ **Every process reads its own or rank 0 reads and send to all others?**
- ❑ **What should be communicated/synchronized?**
- ❑ **When I need to communicate/synchronize processes?**
- ❑ **Who will output to the screen?**
- ❑ **Who and how will output be written?**
- ❑ **Output files have to be concatenated?**
- ❑ *"... Debugging MPI programs is very trick, we must write on the screen carefully ..."*

# Assuring portability

## Original source code

```

program foo
#ifdef MPICODE
include 'mpif.h'
character*60 message
dimension :: istatus(MPI_STATUS_SIZE)
#endif
integer :: nprocs, myrank
irank=0; nprocs=1
#ifdef MPICODE
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
#endif
if (myrank.eq.0) then
    ! Inicie algo
#endif
#ifdef MPICODE
else
    ! Termine algo iniciado no rank 0
#endif
endif
#ifdef MPICODE
call MPI_BARRIER(MPI_COMM_WORLD,ierr)
call MPI_FINALIZE(ierr)
#endif
stop
end program
    
```

Serial code

ifort -c -fpp foo.f90

```

program foo
integer :: nprocs, myrank
irank=0; nprocs=1
if (myrank.eq.0) then
    ! faça algo
endif
stop
end program
    
```

Parallel code

ifort -c -fpp MPICODE foo.f90

```

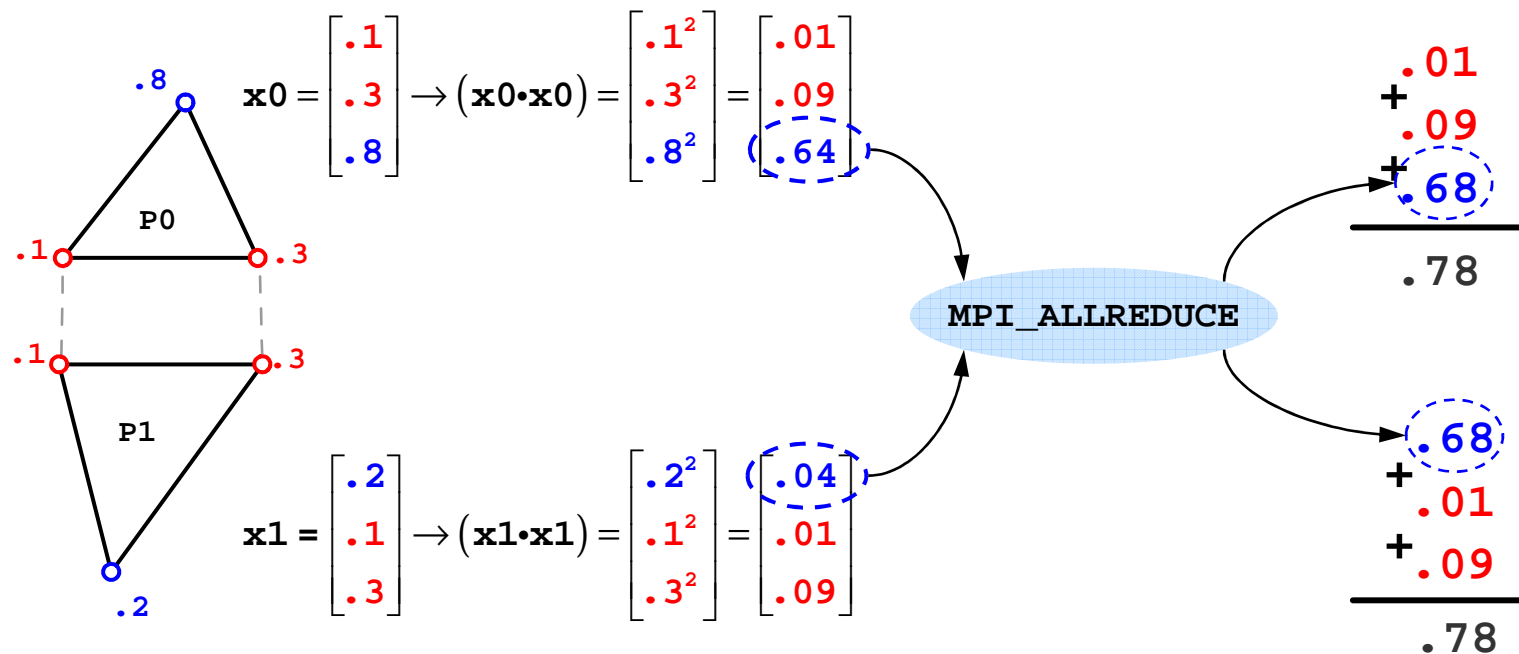
program foo
include 'mpif.h'
character*60 message
dimension :: istatus(MPI_STATUS_SIZE)
integer :: nprocs, myrank
irank=0; nprocs=1
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
if (myrank.eq.0) then
    ! Inicie algo
else
    ! Termine algo iniciado no rank 0
endif
call MPI_BARRIER(MPI_COMM_WORLD,ierr)
call MPI_FINALIZE(ierr)
stop
end program
    
```

# Scalar product in MPI

## Serial

$$\mathbf{x} = \begin{bmatrix} .8 \\ .1 \\ .2 \\ .3 \end{bmatrix} \rightarrow (\mathbf{x} \cdot \mathbf{x}) = \begin{bmatrix} .8^2 \\ .1^2 \\ .2^2 \\ .3^2 \end{bmatrix} = \boxed{.78}$$

## Parallel



# Algorithm for parallel dot product

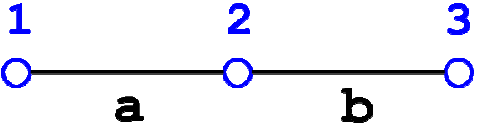
```

function pddot(n,nit,dx,dy)
use mpidefs
! n= #internal values, nit=#interface values
integer :: n, nit, ddot1, ddot2, drec, pddot
real*8   :: dx(1), dy(1)

#ifdef MPICODE
if (nprocs.ne.1) then
    ddot1 = ddot( n,dx( 1),1,dy( 1),1) ! BLAS ddot local
    ddot2 = ddot(nit,dx(n+1),1,dy(n+1),1) ! BLAS ddot interface
    CALL MPI_ALLREDUCE( ddot1, drec, 1, MPI_DOUBLE_PRECISION, &
                        MPI_SUM, MPI_COMM_WORLD, ierr )
    pddot = drec + ddot2                ! global ddot
else
#endif
    pddot = ddot(n,dx(1),1,dy(1),1)
#ifdef MPICODE
endif
#endif
end function
    
```

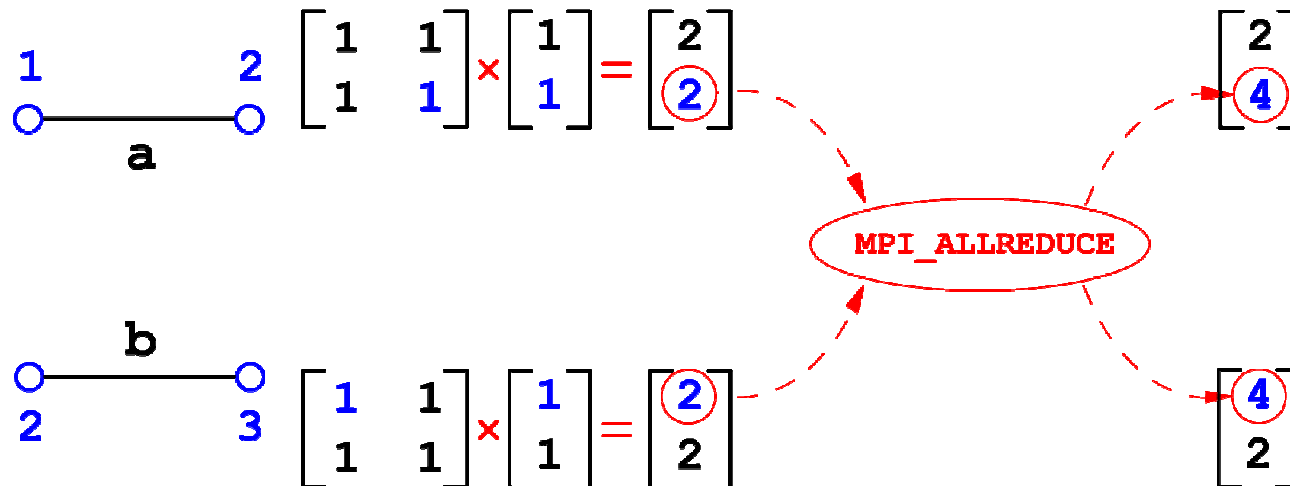
# MPI Matrix-vector product

Global Matvec



$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 2 \end{bmatrix}$$

MPI Matvec



## MPI EDS Matrix-Vector Product

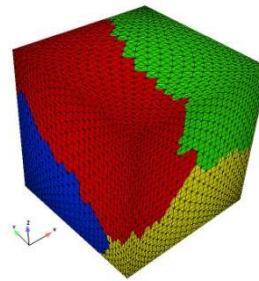
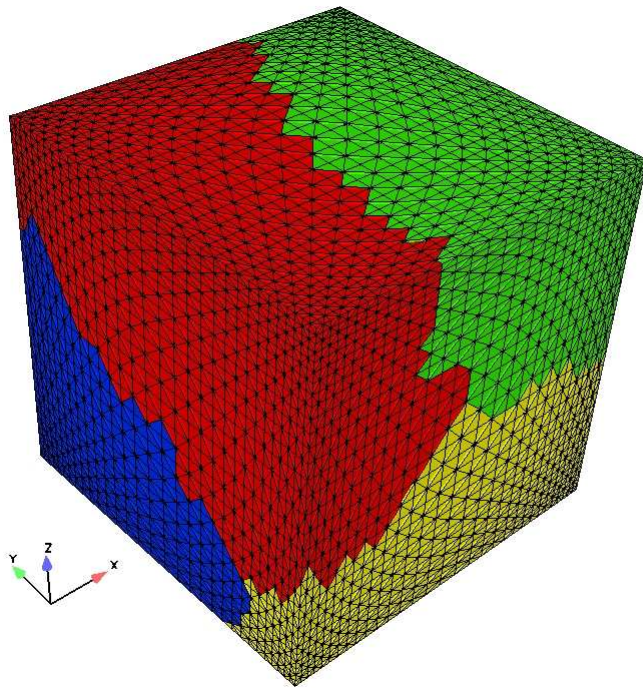
```
do ie = 1, nedges
  neq1 = lm(1,ie)
  neq2 = lm(2,ie)
  ...
  retrieve and multiply 4 coeffs.
  ...
  p(neq1) = p(neq1) + ap
  p(neq2) = p(neq2) + ap
enddo

#ifdef MPICODE
! Adding interface contributions
call MPI_AllReduce
#endif
```



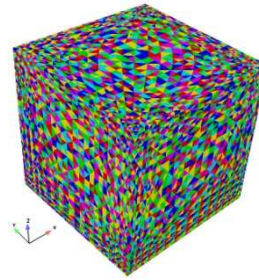
# Hybrid parallelism

- How do we work with data partitioning and memory dependency?



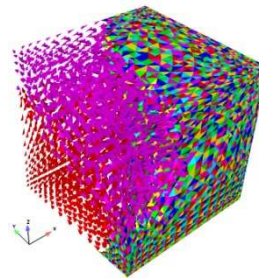
**Mesh partitioning (Metis)**

Distributed memory (MPI)



**Mesh coloring**

Threaded parallelism (OpenMP)



**Data partitioning + Mesh coloring**

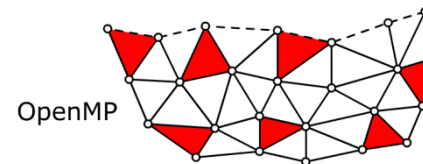
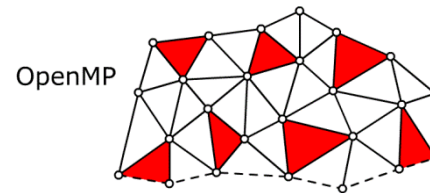
# Hybrid matrix-vector product (OpenMP+MPI)

```

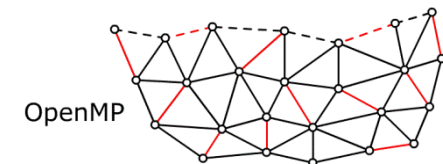
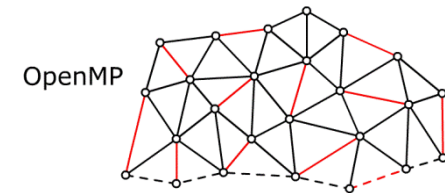
iside = 0
DO iblk = 1, nedblk
  nvec = ia_edblk(iblk)
  !dir$ ivdep
  !$OMP PARALLEL DO
    DO ka = iside+1, iside+nvec, 1
      ...MATVEC computations...
    ENDDO
  !$OMP END PARALLEL DO
ENDDO
...over interface nodes...
#ifdef MPICODE
call MPI_AllReduce
#endif

```

**Element-by-Element**



**Edge-by-Edge**



## Other important operations

- ❑ **Every other operation involving assembling of nodal values should be treated (scatter and add) similarly**
  
- ❑ **Example operations:**
  - Residual vector assembly;
  - Nodal block-diagonal preconditioners.

# Nodal Block-Diagonal Preconditioning

- Nodal block-diagonals are naturally computed when integrating element matrices and can be used as serial or parallel preconditioners
- The nodal block-diagonal matrix has the following structure :

$$W = \begin{bmatrix} \boxed{\begin{matrix} \times & \times \\ \times & \times \end{matrix}} & & & \\ & \boxed{\begin{matrix} \times & \times \\ \times & \times \end{matrix}} & & \\ & & \boxed{\begin{matrix} \times & \times \\ \times & \times \end{matrix}} & \\ & & & \ddots \\ & & & & \boxed{\begin{matrix} \times & \times \\ \times & \times \end{matrix}} \end{bmatrix}$$

$$Ax = b$$

$$W^{-1}Ax = W^{-1}b$$

- In practice nodal block-diagonals (generally with dimensions  $n_{gl} \times n_{gl}$ ) are inverted block-by-block and multiplied by RHS vector at the beginning of the iterative method;
- During iterations, after computing matvec we apply the preconditioner.

# MPI Nodal-Block Diagonal

```

subroutine LinearSolution (A, W, x, b)
! Solve  $W^{-1} A x = W^{-1} b$ 

#ifdef MPICODE
! Globalize interface blocks
#endif

do i=1,nblocos
! invert  $W_i$ 
enddo

! Multiply  $W^{-1} b$ 

call GMRES(A, W, x, b)
! Matvec           $z \leftarrow A x$ 
! Multiply         $W^{-1} z$ 

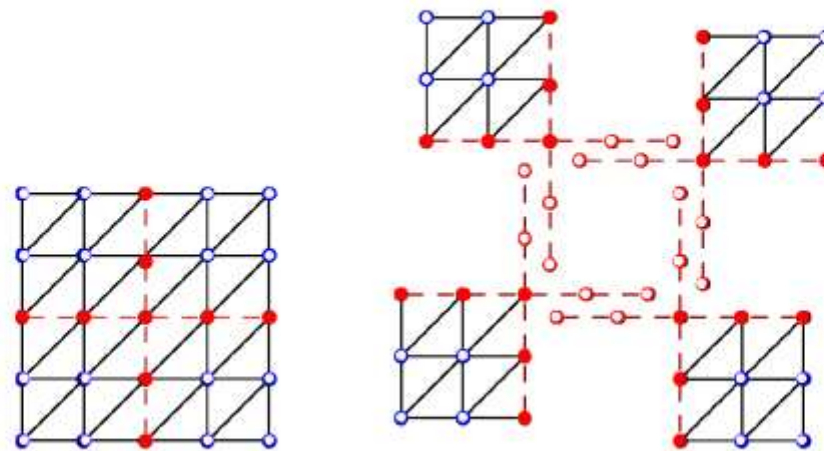
end solution

```

# MPI Collective Communications

## MPI collective

1. All shared equations are synchronized in just one collective operation;
2. Easy to implement;
3. Poor performance for massive parallelism;
4. Some (small) improvements can be done (... *but there's no miracle...*).

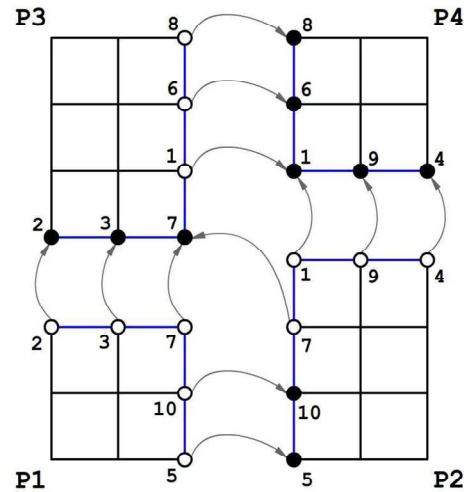


(a) Original mesh

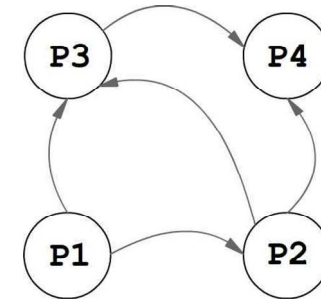
(b) Redundant communication

# P2P Subdomain Communication

## Master-Slave subdomain relationship



(a) Mesh Partition



(b) Communication Graph

Exchange information between neighboring processors implemented in two stages:

- (i) slaves processes send their information to be operated by masters
- (ii) solution values are copied from masters to slaves.

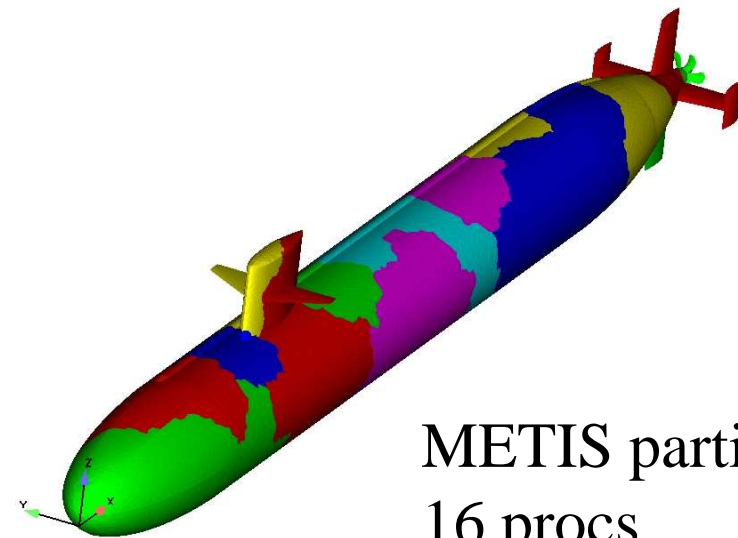
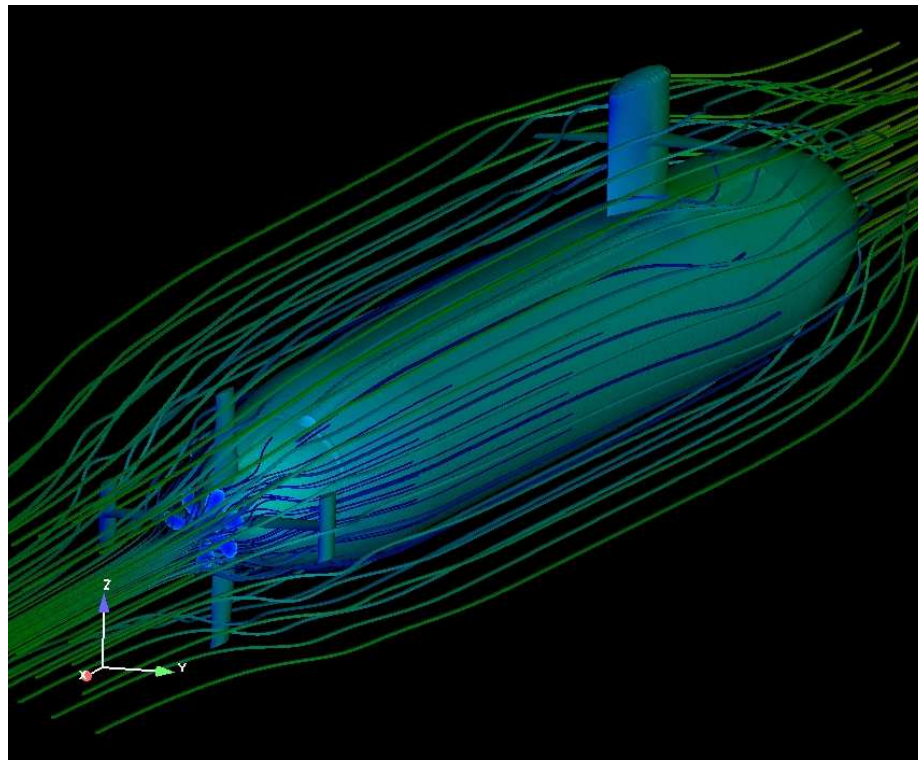
EdgeCFD uses non-blocking send and receive MPI primitives

*See also: Karanam, Jansen, Whiting, Geometry based pre-processor for parallel fluid dynamic simulations using a hierarchical basis, Engineering with Computers (2008)*

# **SOME EXAMPLES**



# Flow around a Los Angeles Class Submarine

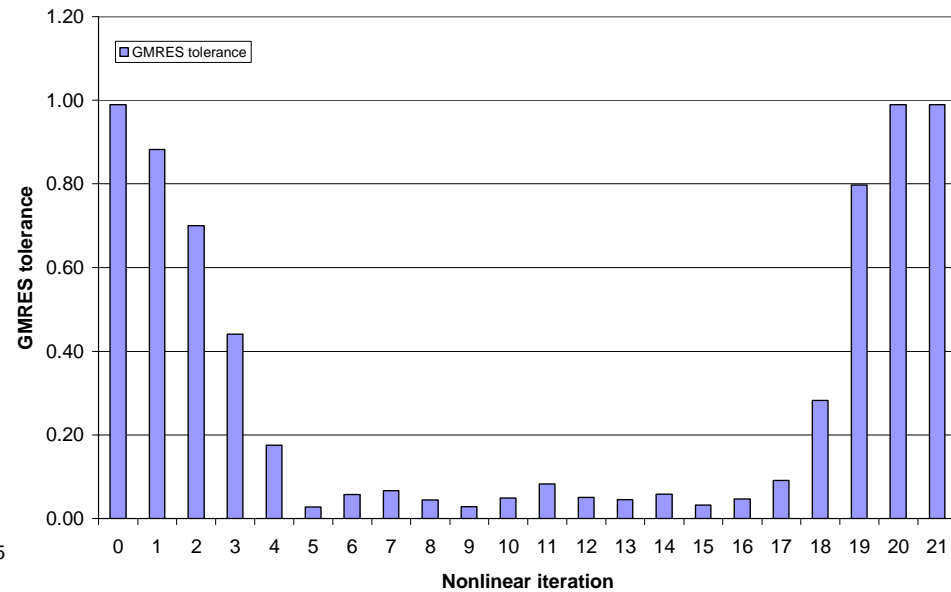
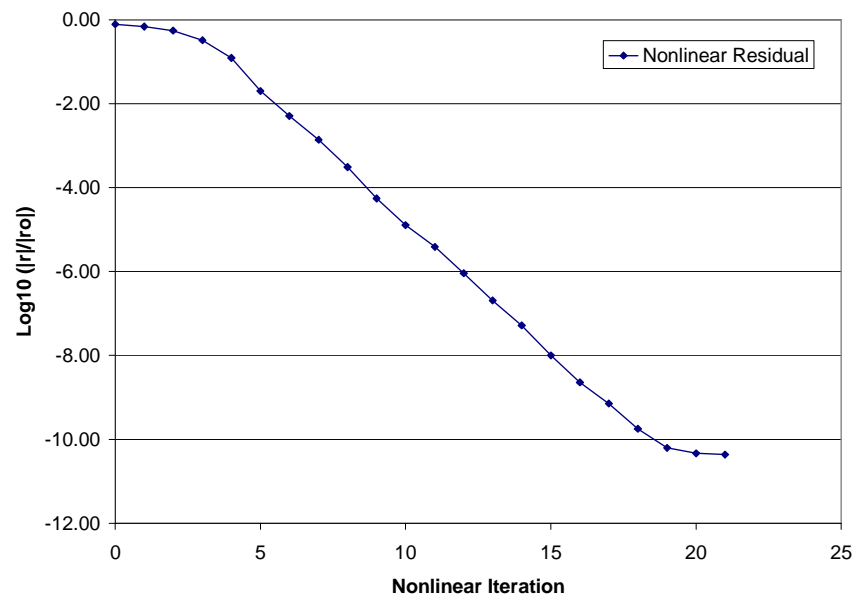


METIS partition  
16 procs

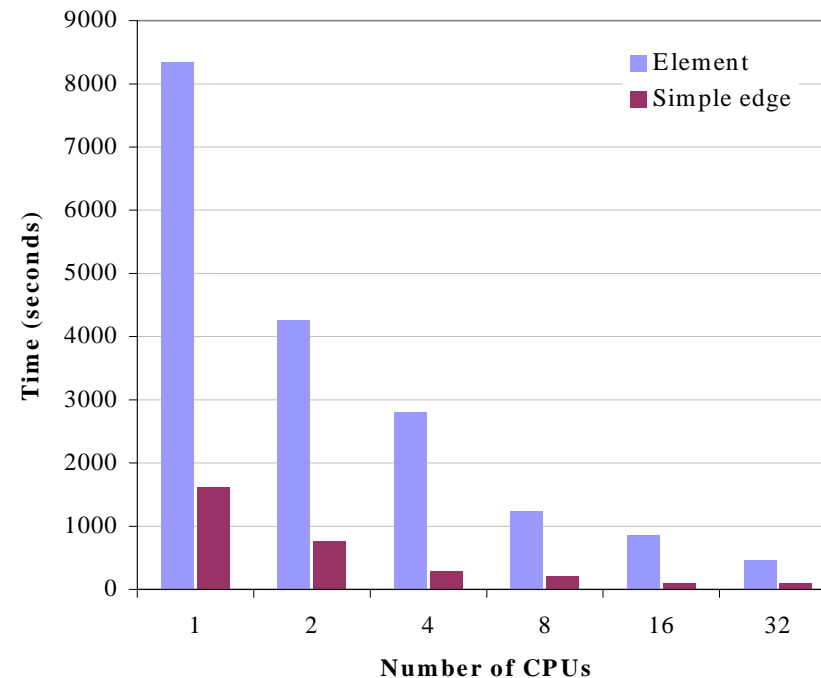
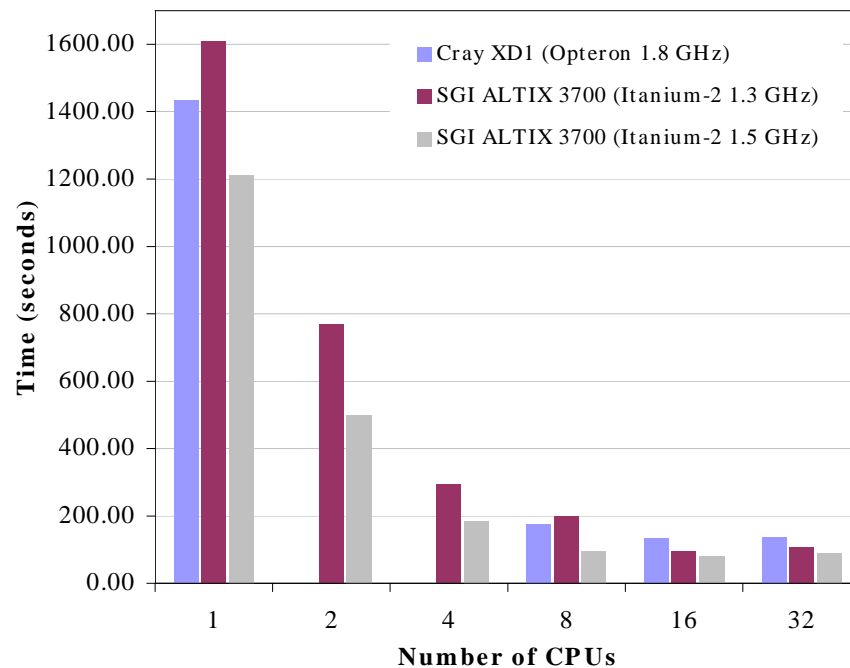
504,947 tetrahedral elements,  
998,420 edges and 92,564 nodes

Renato N. Elias, Marcos A. D. Martins, Alvaro L. G. A. Coutinho: Parallel Edge-Based Inexact Newton  
Solution of Steady Incompressible 3D Navier-Stokes Equations. Euro-Par 2005: 1237-1245

# Inexact Newton Behavior

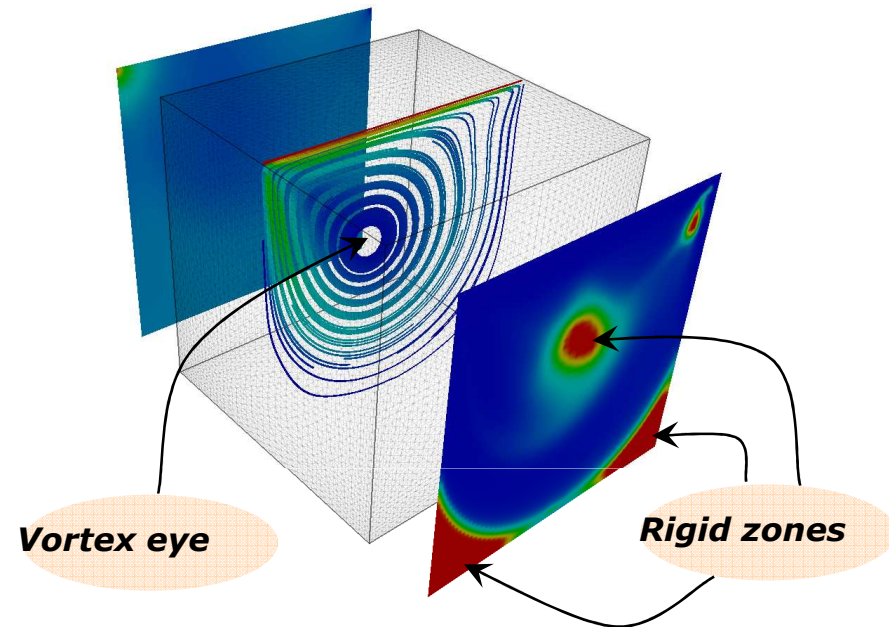
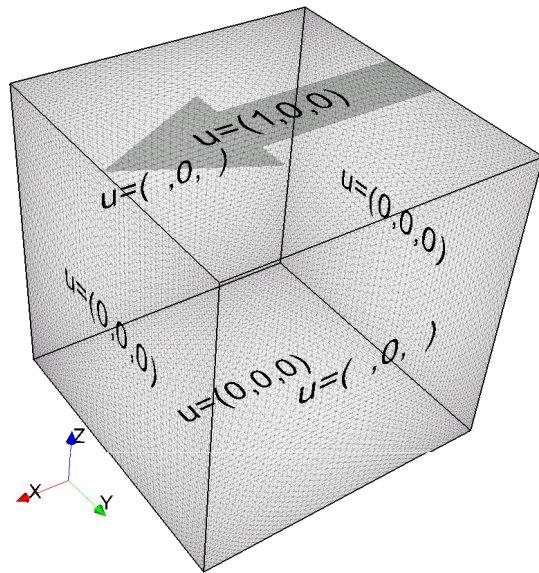


# Parallel Performance



(Left) Message passing performance in SGI Altix and Cray XD1 – edge-based data structure,  
 (Right) Data structure comparisons on SGI Altix (MPI).

# Bingham Flow in a Lid Driven Cavity



PROBLEM SIZES

	31x31x31	51x51x51	71x71x71	101x101x101
<b>Tetrahedra</b>	148,955	663,255	1,789,555	5,151,505
<b>Edges</b>	187,488	819,468	2,193,048	6,273,918
<b>Nodes</b>	32,768	140,608	373,248	1,061,208
<b>Equations</b>	117,367	525,556	1,421,776	4,101,106

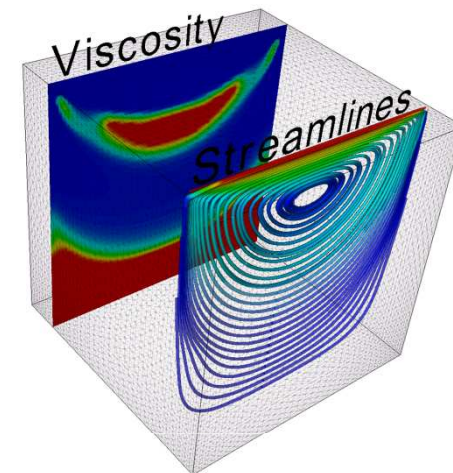
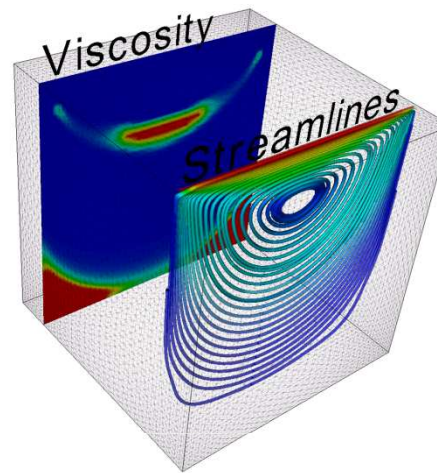
R. N. Elias, M. A. D. Martins, A. L. G. A. Coutinho, Parallel edge-based solution of viscoplastic flows with the SUPG/PSPG formulation, Comput. Mech. (2006) 38: 365–381

# Validation

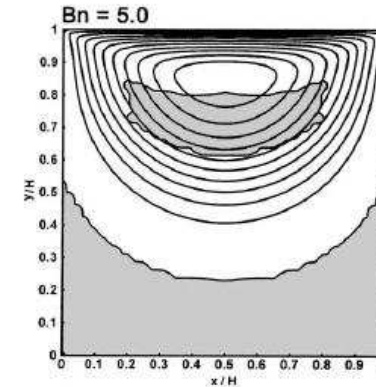
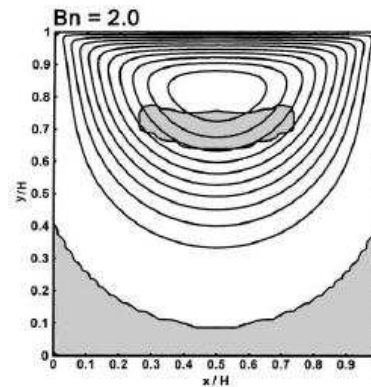
(**Reynolds = 1**, and **Bingham numbers**  $Bn = \frac{\sigma_y h}{u \mu} = 2 \text{ and } 5$ )

## Viscosity/Streamlines

This work using a linear tetrahedra  
mesh 51x51x51



**Yielded/unyielded zones**  
**Mitsoulis and Zisis** using a  
biquadratic quadrilateral  
mesh 40x40

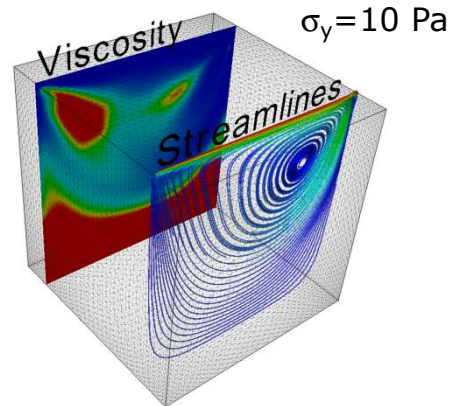


Mitsoulis, E. and Zisis, Th., *Flow of Bingham Plastics in a Lid-Driven Square Cavity*, J. Non-Newtonian Fluid Mech., 2001 (101):173-180

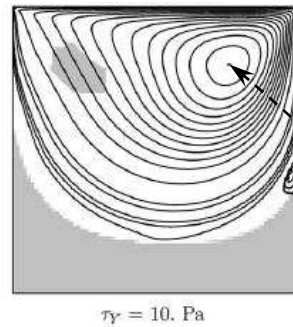


# Result Comparisons

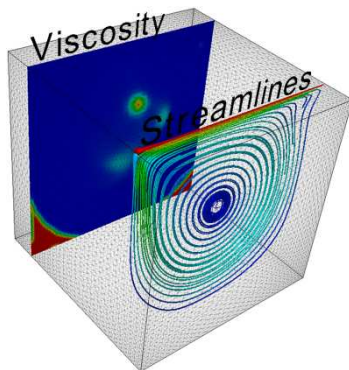
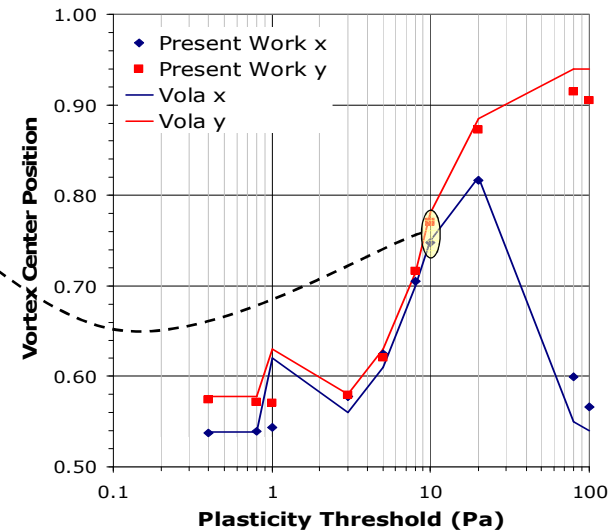
(Reynolds = 1000)



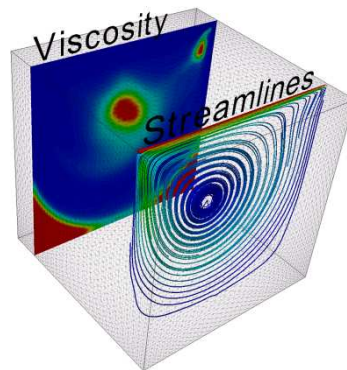
**This work**  
(mesh 51x51x51)



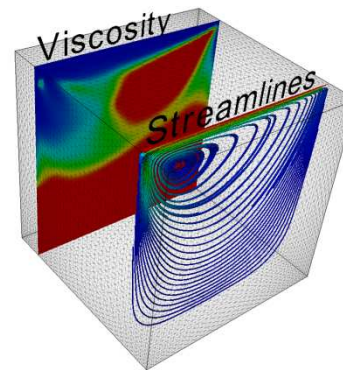
**Vola et al**  
(mesh 80x80)



$\sigma_y = 1 \text{ Pa}$



$\sigma_y = 5 \text{ Pa}$

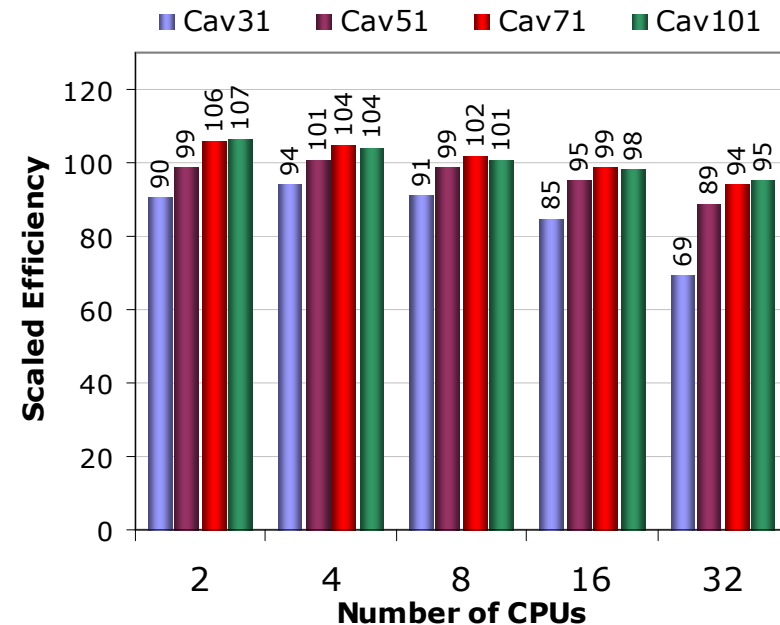
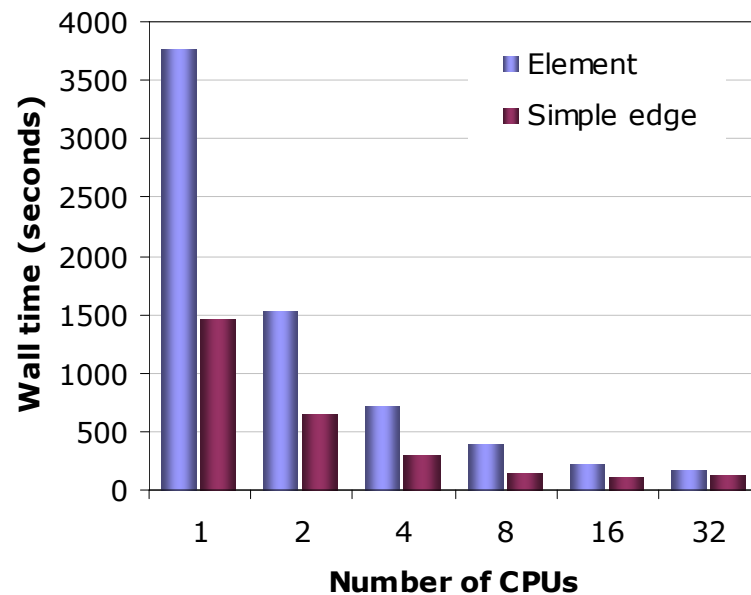


$\sigma_y = 20 \text{ Pa}$

$$\mu(\dot{\gamma}) = \begin{cases} \mu_0 + \frac{\sigma_y}{\dot{\gamma}} & \text{if } \dot{\gamma} > \frac{\sigma_y}{\mu_r - \mu_0} \\ \mu_r & \text{if } \dot{\gamma} \leq \frac{\sigma_y}{\mu_r - \mu_0} \end{cases}$$

Vola, D., Boscardin, L. and Latché, J. C.,  
*Laminar Unsteady Flows of Bingham  
Fluids: a Numerical Strategy and Some  
Benchmark Results*, J. Comput. Phys., 2003  
(187):441-456

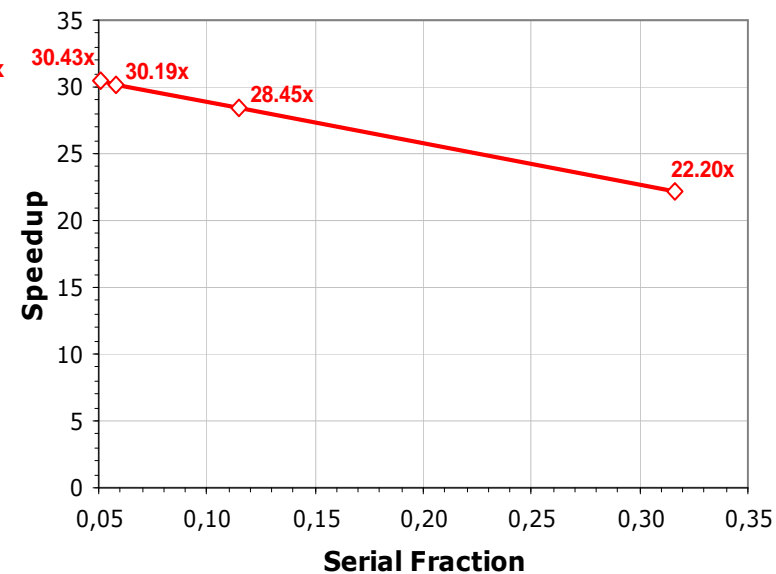
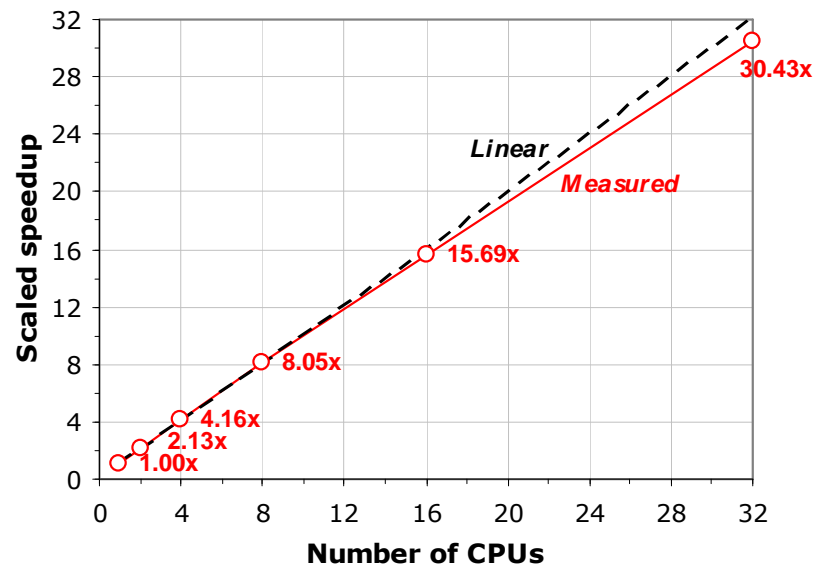
# Data Structure and Parallel Performance



SGI Altix jan/2005

# Parallel Performance

(Leaky Lid Driven Cavity Flow on SGI Altix)





# Performance on Current Processors

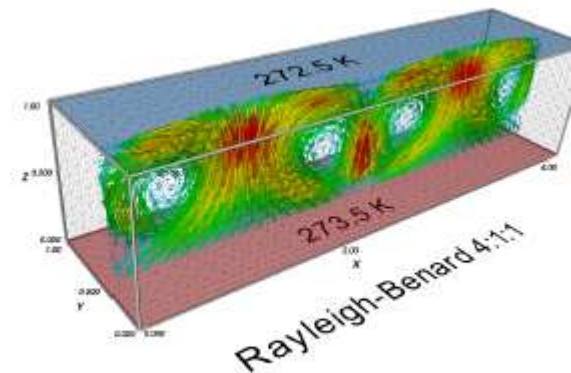
- Rayleigh-Benard ( $Ra=30,000$ ,  $Pr=0.71$ )

Assumptions:

1. All mesh entities were reordered to explore memory locality;
2. Same mesh ordering to all systems;
3. Same compiler (Intel) and compilation flags (-fast)

Mesh sizes

	MSH1	MSH2
Tetrahedra	178,605	39,140,625
Nodes	39,688	7,969,752
Edges	225,978	48,721,528
Flow equations	94,512	31,024,728
Transport equations	36,080	7,843,248



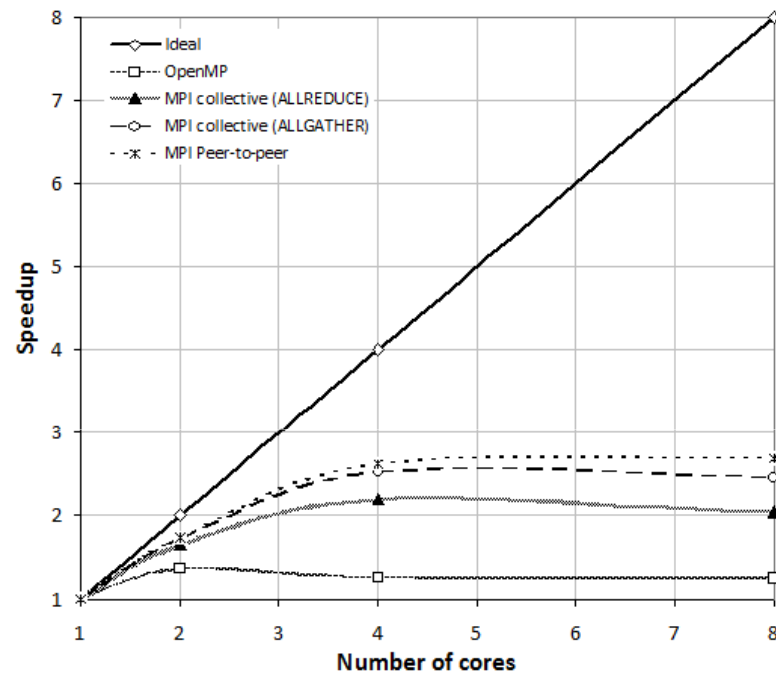
Natural convection problem

Table 1. Systems summary

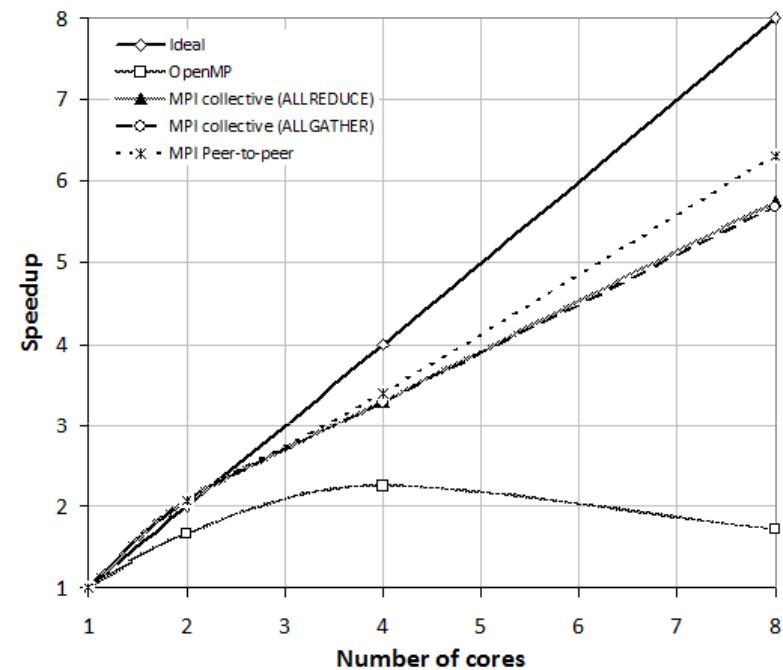
# Test Systems

	Altix-ICE	Cluster Dell	Nehalem server	Ranger
<b>Processor</b>	Intel Xeon X5355	Intel Xeon E5450	Intel Xeon X5560 (i7)	AMD Opteron 8354
<b>Codename</b>	Clovertown	Harpertown	Nehalem	
<b>Launch date</b>	2006	2007	2009	2007
<b>Clock speed</b>	2.66 GHz	3.00 GHz	2.8 GHz	2.2 GHz
<b>Number of nodes</b>	32	16	1	3,936
<b>Sockets per node</b>	2	2	2	4
<b>Cores per socket</b>	4	4	4	4
<b>L1 cache</b>	32 KB (no sharing)	32 KB (no sharing)	32 KB (no sharing)	64 KB (no sharing)
<b>L2 cache</b>	4 MB (0,2)(4,6)	6 MB (0,4)(2,6)	256 KB (no sharing)	512 KB (no sharing)
<b>L3 cache</b>	—	—	8 MB (0,1,2,3)	2 MB (shared)
<b>Operating system</b>	SUSE 10 sp2	Red Hat Advanced Server 5.3	CentOS 5.3	CentOS 4.5
<b>Compiler</b>	Intel Fortran 10.1	Intel Fortran 11.1	Intel Fortran 11.1	Intel Fortran 10.1
<b>Compiler flags</b>	-fast	-fast	-fast	-fast

# Parallel models speedup per processor (intra-node)



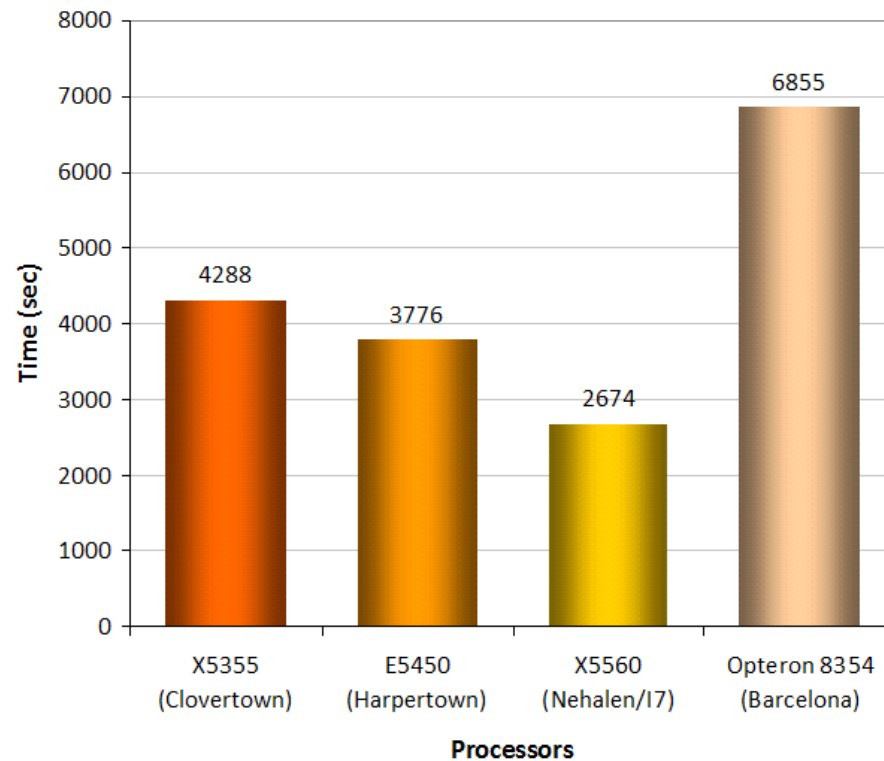
(a) SGI Altix-ICE (Clovertown)



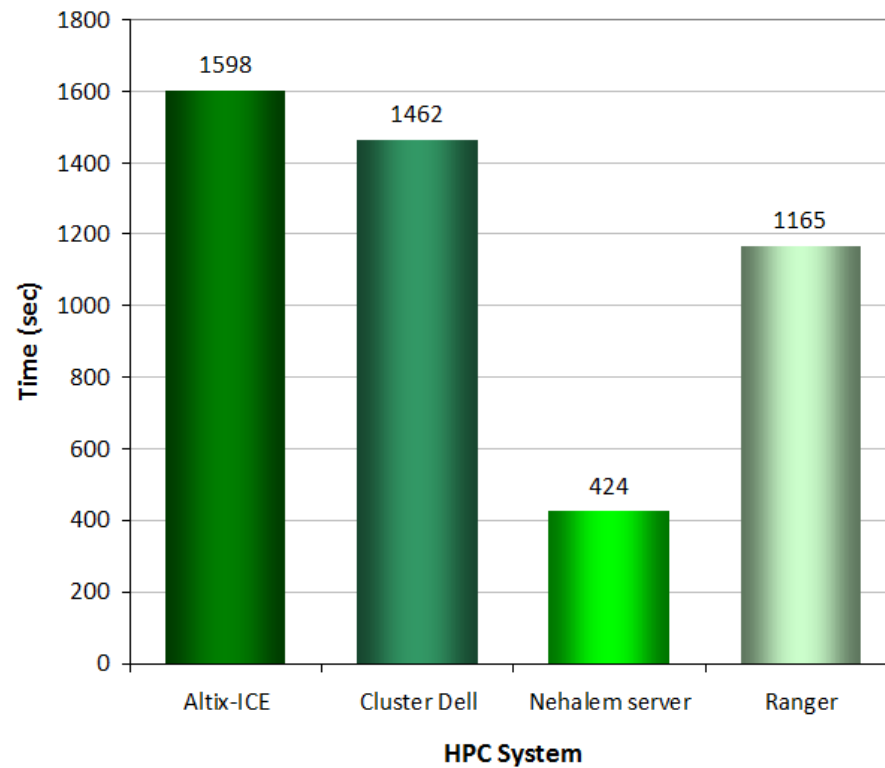
(b) Nehalem server (Nehalem / i7)

All tests in MSH1  
which is relatively small for a large number of cores

# Processor performance

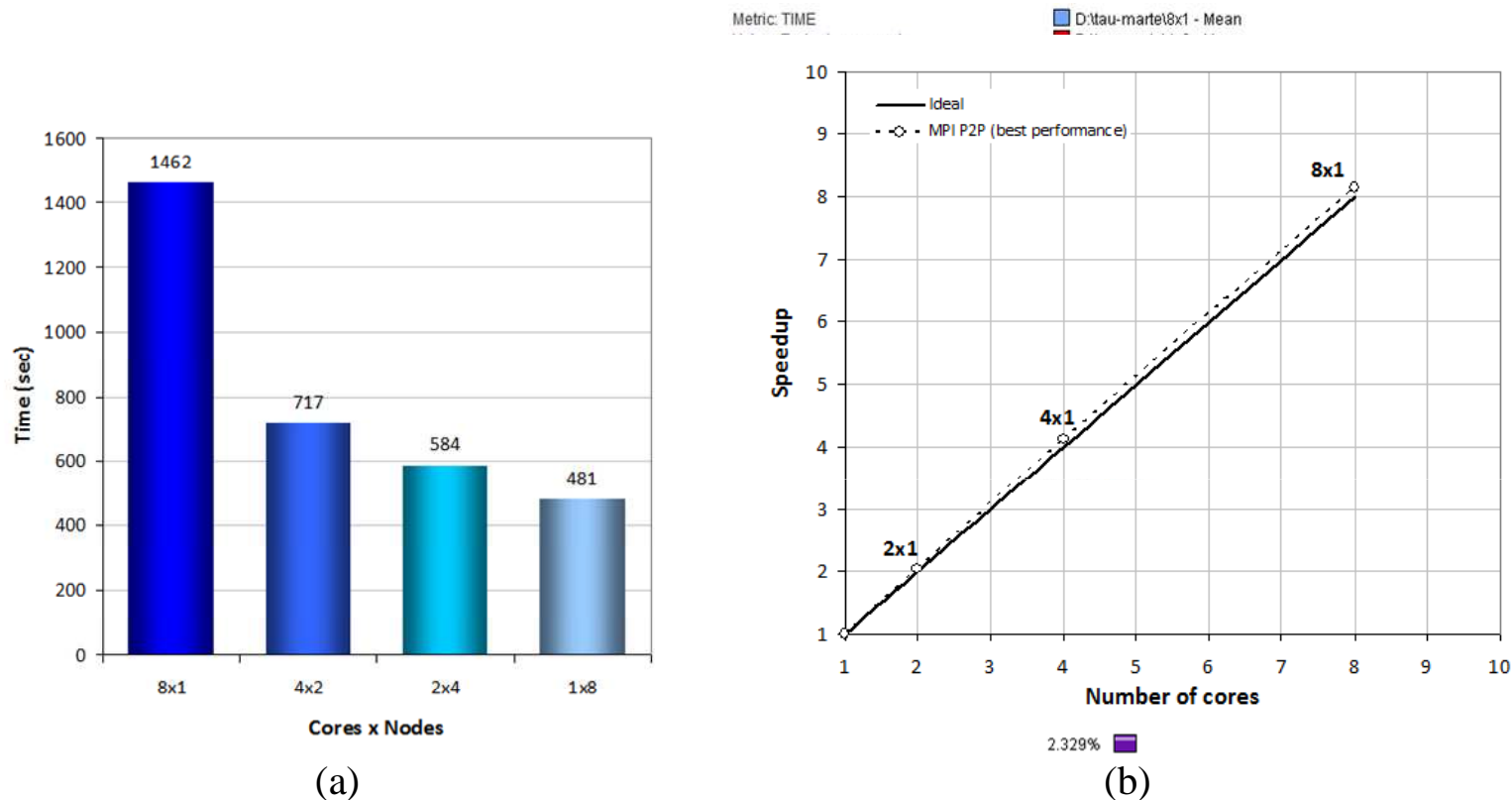


(a) CPU (serial run)



(b) System (8 cores, 1 node)

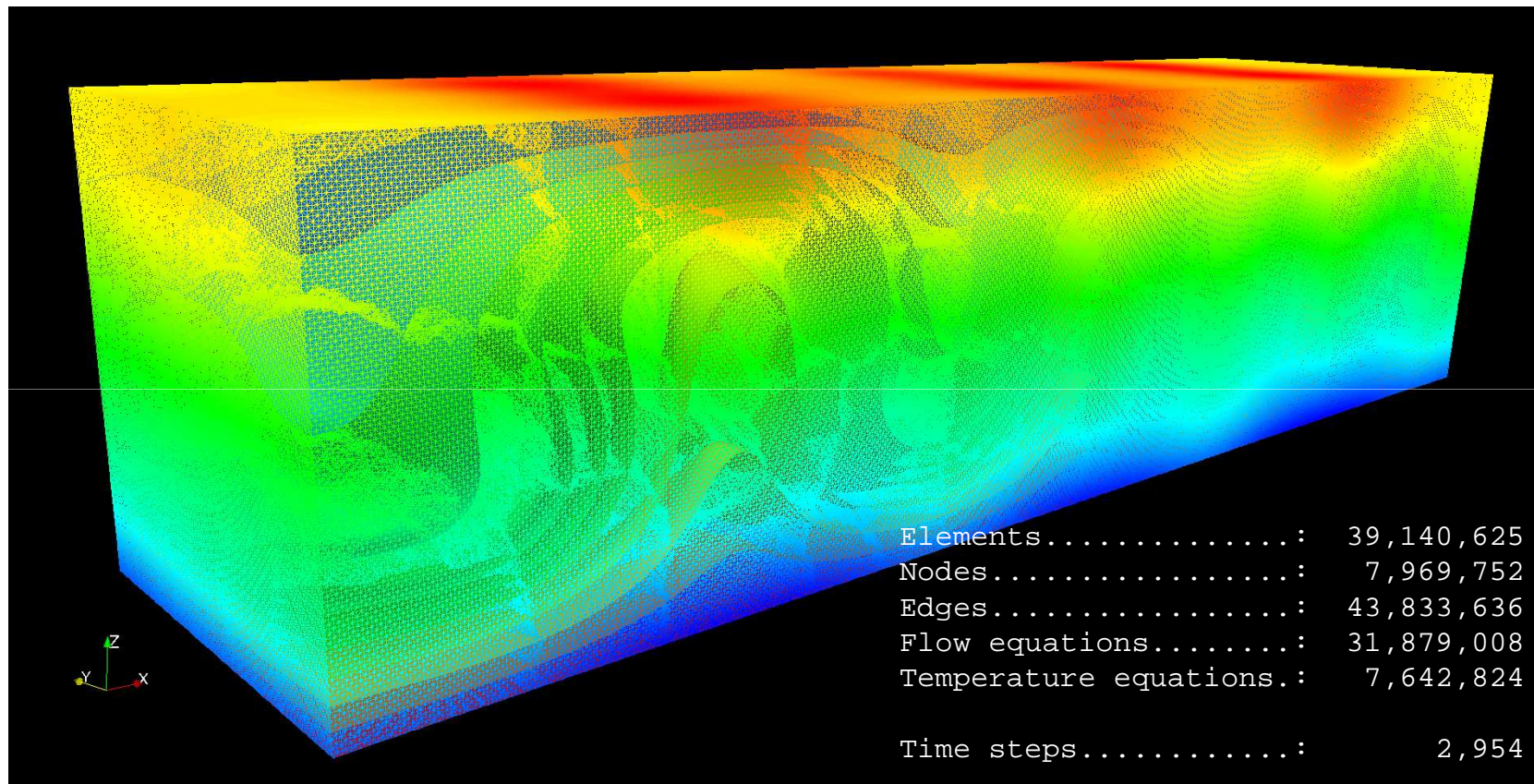
# Effects of Process Placement



**Figure 1. Performance impact according to cores x nodes distribution on Cluster Dell**

Similar results can be found in: Jeff Diamond, Byoung-Do Kim, Martin Burtscher, Steve Keckler, Keshav Pingali and Jim Browne, Multicore Optimization for Ranger, TeraGrid09, <http://www.teragrid.org/tg09/>

# Rayleigh-Benard 4:1:1 - 501×125×125 mesh



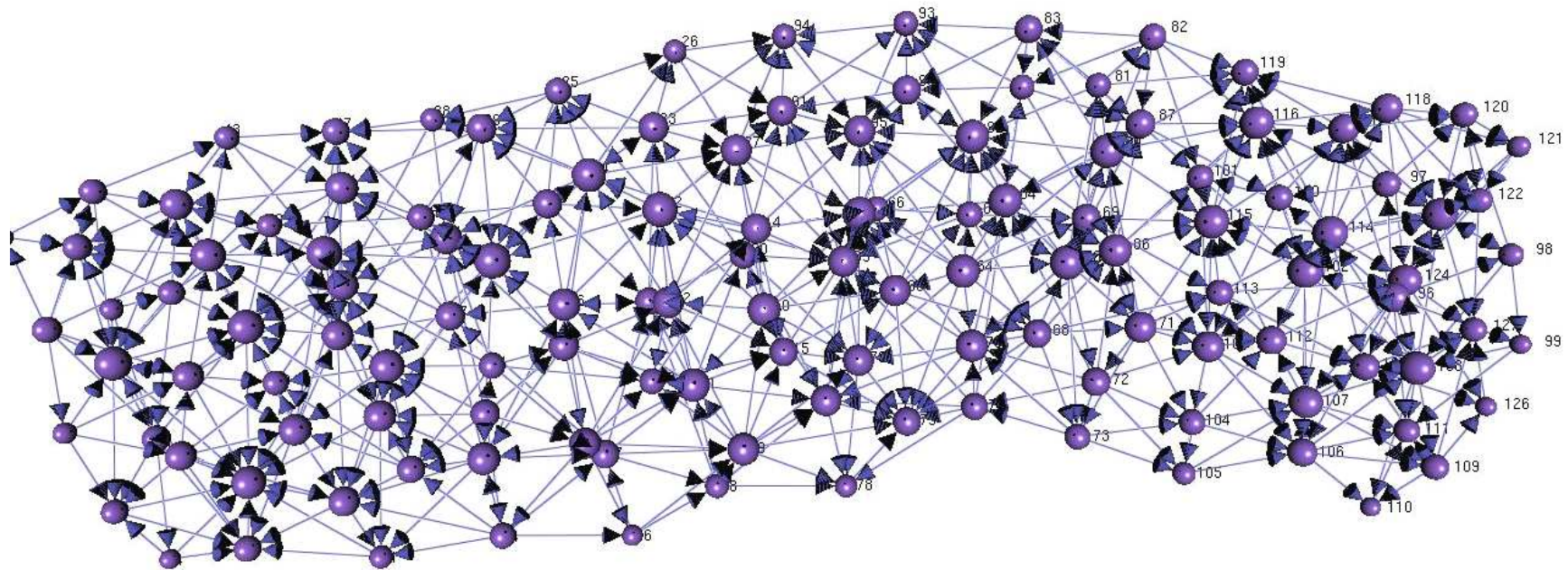
Mesh generation: SGI Altix 450 128GB RAM

Solver: SGI Altix ICE 8200, 128 cores, Cluster Dell 64 cores, MPI-P2P



# P2P Communication Graph

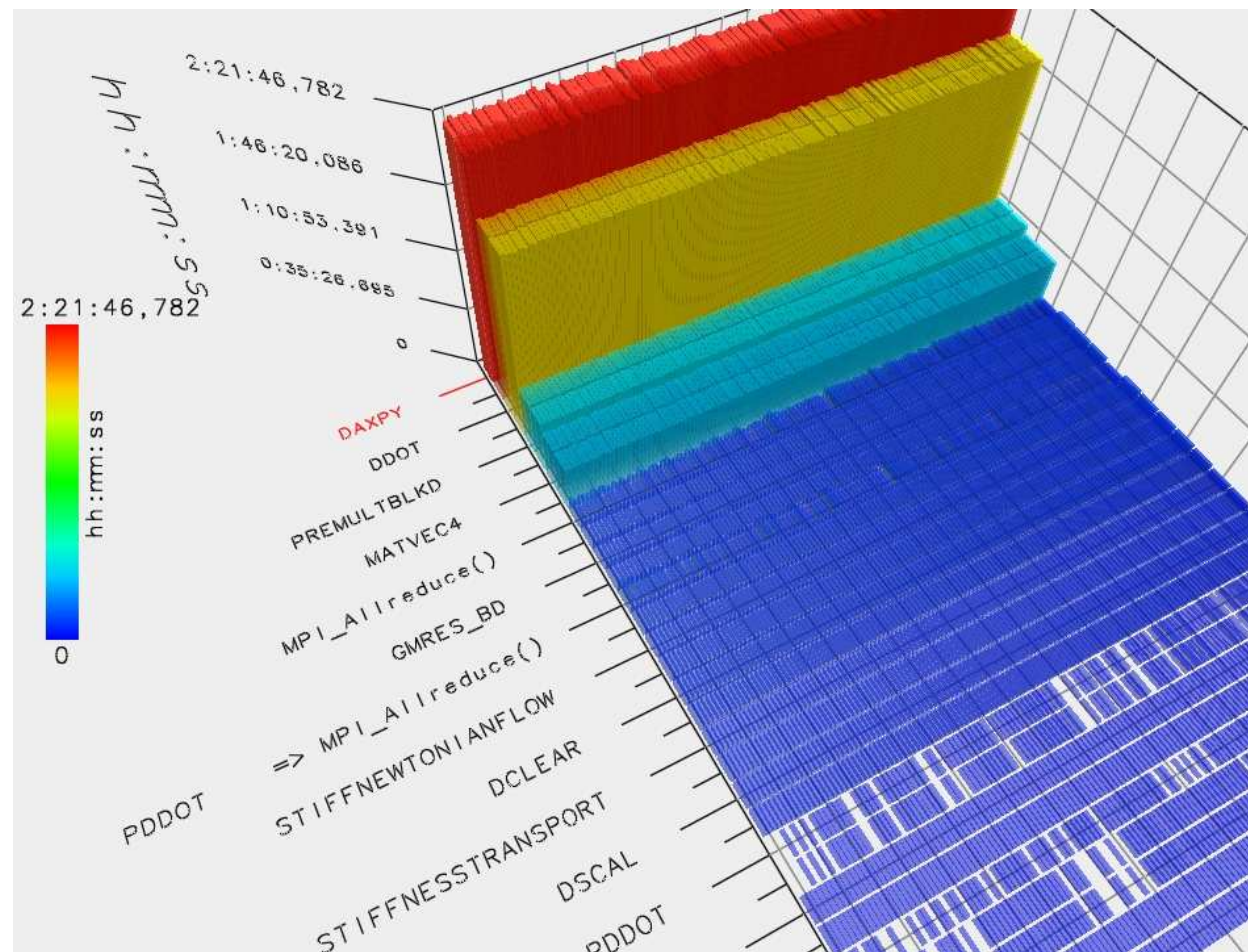
## SGI Altix ICE 8200, 128 cores



Arrows indicate communication direction in sending operations  
Communication is reversed in receiving

Graph visualization tool: Nodes3D, <http://brainmaps.org/index.php?p=desktop-apps-nodes3d>

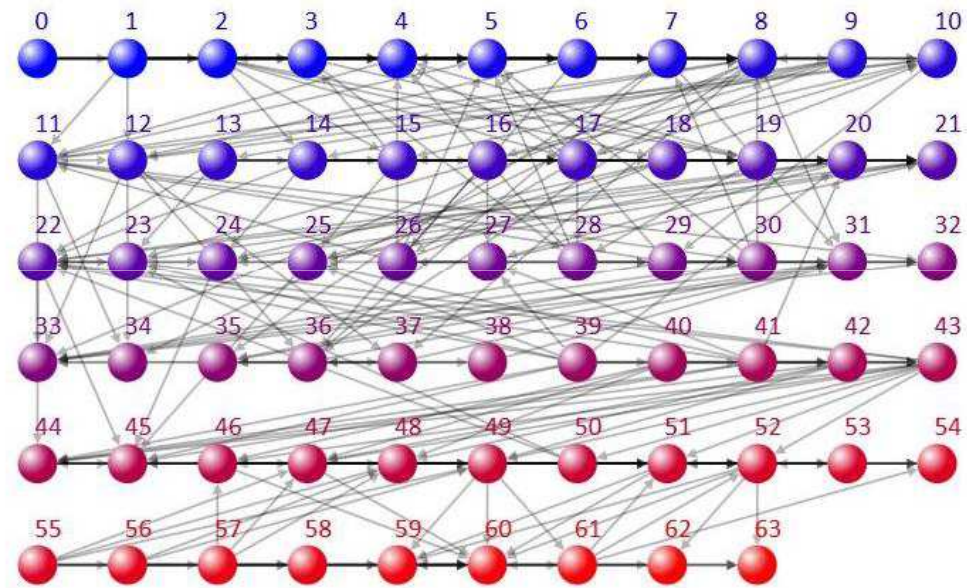
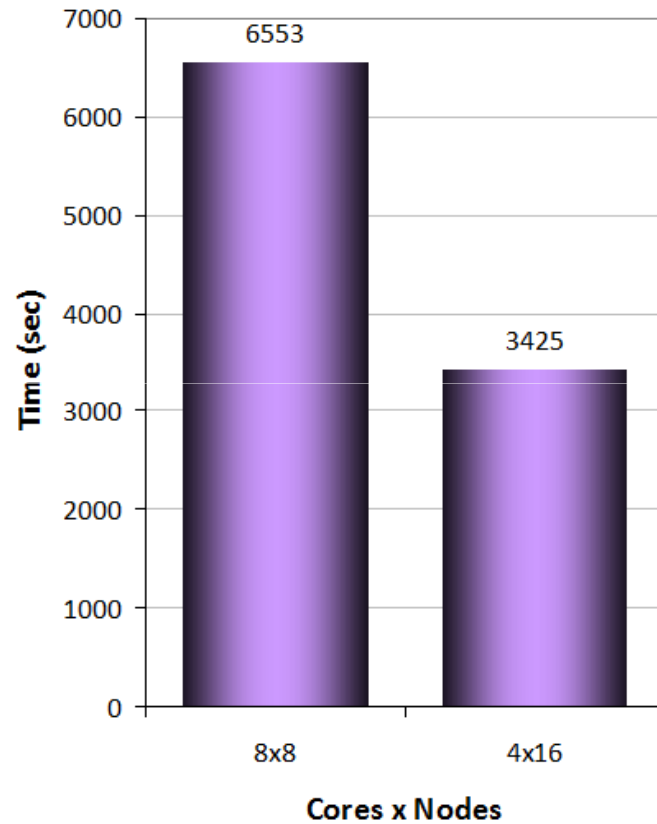
# TAU Parallel Profiling



SGI Altix ICE 8200, 128 cores



# Cluster Dell, 64 cores, process placement

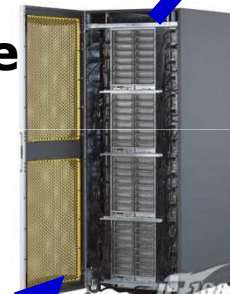


Communication graph

Time spent in 10 time steps

# What we have learned from the applications

- ❑ **HPC can transform engineering and science**
- ❑ **Porting a code is not the issue: performance needs code reformulation and new data structures**
- ❑ **Focus is not the hardware: we need stable and effective programming models, scaling upwards**

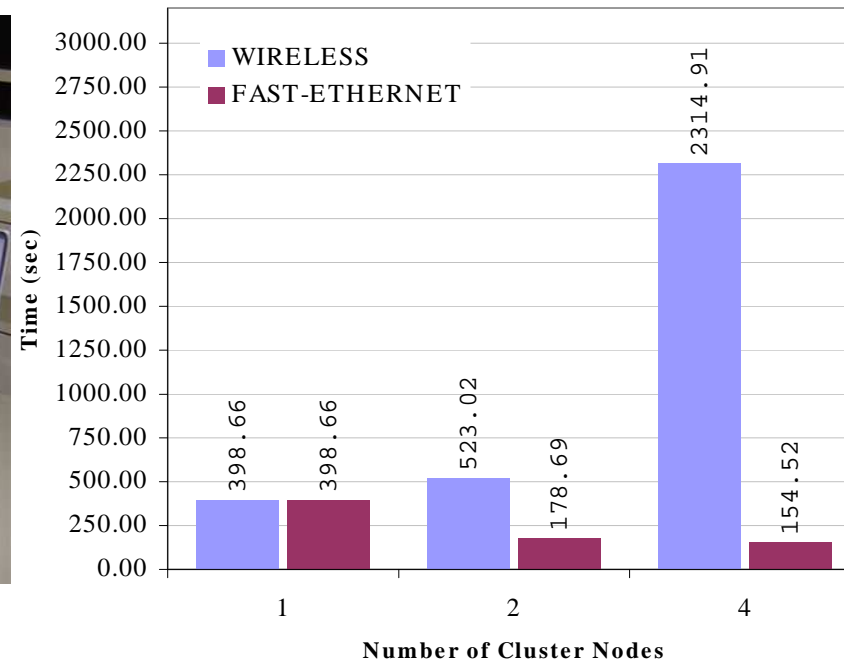


# Wireless Cluster Experiment

- ❑ mini-cluster formed by 4 laptops and a wireless/fast-ethernet network
- ❑ 2 Intel Centrino 1.6 GHz/512Mb, 1 Intel Centrino 1.3 GHz /512Mb and 1 Intel Pentium 4 2.4 GHz/512Mb
- ❑ Interconnection: Linksys Wireless-B Hub, IEEE 802.11b/2.4GHz/11Mbps or Fast-Ethernet 10/100Mbps network



# Wireless Cluster Experiment



(Left) Minicluster mobile wireless/fast-ethernet,  
 (Right) Performance comparison between wireless and fast-ethernet networks.

## Some comments

- ❑ Older Intel Xeon processors dramatically suffer when large workloads are imposed to a single node;
- ❑ Processor placement has great influence in older Xeon based systems;
- ❑ Nehalem (Core I7) has several improvements over its predecessors;
- ❑ Third level shared cache (now we have a true quad core...);
- ❑ Fast interconnect channel among processors (well, sounds like AMD Hyper Transport...).
- ❑ Point-to-Point MPI model is the best strategy to reach good parallel performance;
- ❑ OpenMP performance is still poor, but it's getting better (for the same implementation!);
- ❑ Many-core and GPU paradigms are bringing back threaded parallelism...

## More reading on recent stuff

- ❑ B. Wylie, M. Geimer, M. Nicolai and M. Probst, Performance Analysis and Tuning of the XNS CFD Solver on BlueGene/L, in Lecture Notes in Computer Science Vol. 4757, Proceedings of the 14th EuroPVM/MPI Conference, Springer (2007) 107–116
- ❑ G. Houzeaux, M. Vázquez, R. Aubry, J.M. Cela, A massively parallel fractional step solver for incompressible flows, Journal of Computational Physics 228 (2009) 6316–6332
- ❑ O. Sahni, M. Zhou, M. S. Shephard, K. E. Jansen, Scalable Implicit Finite Element Solver for Massively Parallel Processing with Demonstration to 160K cores, Supercomputing 2009
- ❑ S. Turek, D. Göddeke, C. Becker, S. H.M. Buijssen and H. Wobker, FEAST - Realisation of Hardware-oriented Numerics for HPC Simulations with Finite Elements, accepted for publication in Concurrency and Computation, Special Issue Proceedings of ISC'08, Dec. 2009

## Final Remarks

- ❑ **Computational Engineering and Science changed the way we view engineering**
- ❑ **There is no general approach**
- ❑ **Integrated approach: HPC, Visualization, Storage and Communications**
- ❑ **Challenges:**
  - Managing complexity: programming models, data structures and computer architecture → performance
  - Understanding the results of a computation: visualization, data integration, knowledge extraction
  - Collaboration: grid, web, data security

## Acknowledgements

- ❑ **Collaborators:** J. Alves, L. Landau, F. Rochinha, A. Loula (LNCC), S. Malta (LNCC), P. Sampaio (IEN), G. Carey (UT-Austin), T. Tezduyar (Rice)
- ❑ **Students (and ex):** M. Martins, M. Cunha, R. Sydenstricker, L. Catabriga, C. Dias, A. Valli, P. Hallak, I. Slobodcicov, P. Antunes, D. Souza, P. Sesini, A. Silva, R. Elias. A. Mendonça, W. Ney, J. Camata, A. Rossa
- ❑ **Funding:** CNPq, CAPES, FINEP/CTPetro, IBM, ANP, Petrobras
- ❑ **Computational Resources:** NACAD, TACC