Fourth Brazil-France Workshop

On High Performance Computing and Scientific Data Management Driven by Highly Demanding Applications

15-18 September, Gramado, Brazil 2014

Emerging Challenges for EdgeCFD Simulations in Many-core Architectures

Renato N. Elias[1], *José Camata*[1], *Albino Aveleda*[1], *Alvaro Coutinho*[1]

[1] ***High Performance Computing Center (NACAD)***
*{**rnelias**, camata, bino, alvaro}@nacad.ufrj.br*
www.nacad.ufrj.br
Civil Engineering Department (PEC/COPPE)
www.pec.coppe.ufrj.br
Federal University of Rio de Janeiro (UFRJ)
www.ufrj.br

Outline

- ❑ *Introduction & Motivation*
- ❑ *What's EdgeCFD*
  - – *Main aspects and algorithms*
- ❑ *What's Xeon-Phi*
- ❑ *EdgeCFD on Xeon-Phi*
- ❑ *Test-case*
- ❑ *Results*
- ❑ *Closure and discussions*

Introduction

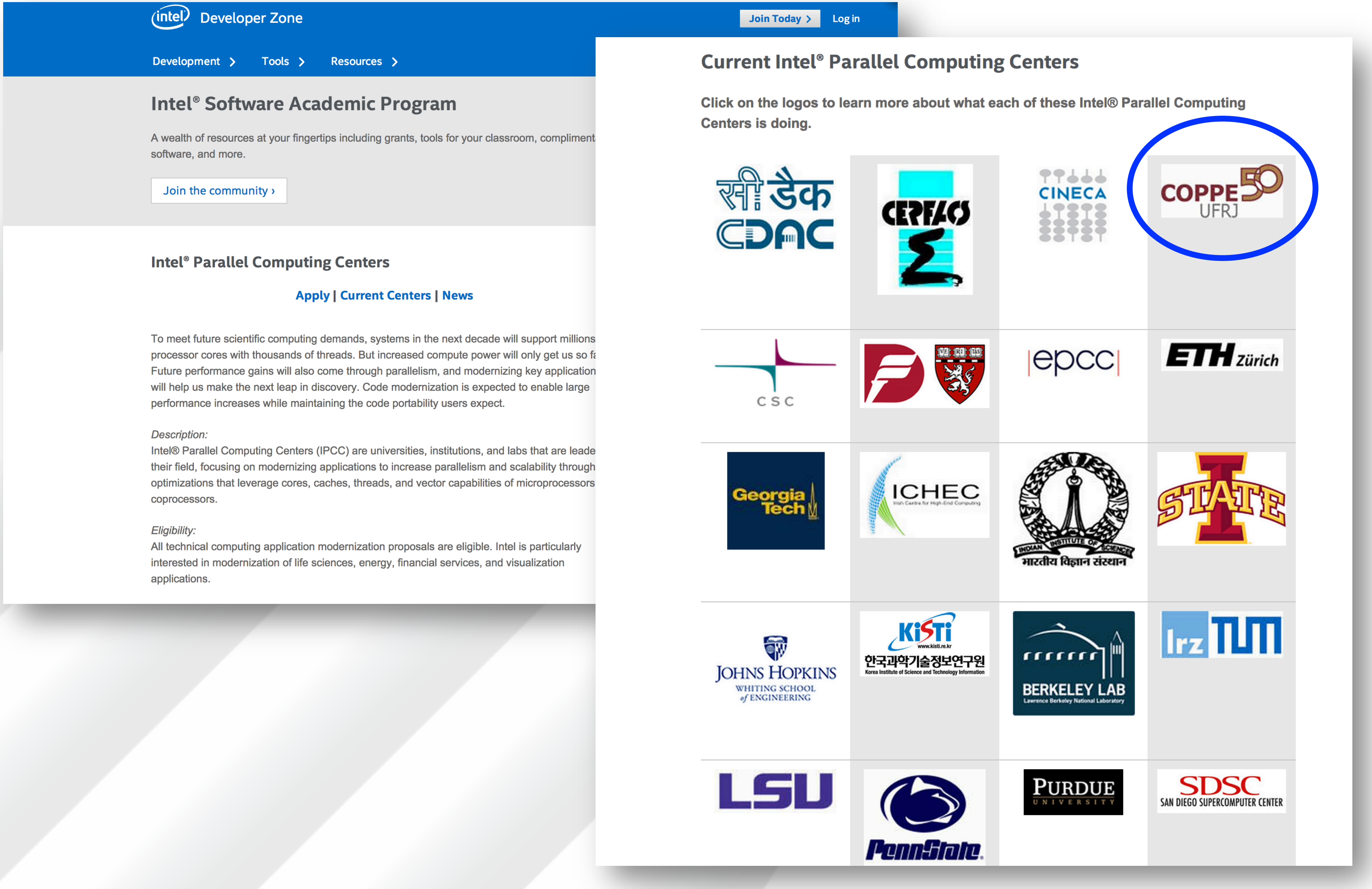
- ***New HPC systems are getting***
  - *Massive in slower cores = "Many-cores"*
  - *Several parallelism and vectorization combinations/possibilities*
- ***1st on top500.org (JUN14)***
  - *Tianhe-2/China: 3.12M of cores (**Intel Xeon Phi**)*
  - *33.86 Pflops in HPL*

Questions/Concerns

1. ***Are applications ready for these new HPC systems?***
2. ***How should we take advantage of them?***
3. ***How do CoProcessors compare to "traditional" CPUs?***
   - *Speed?*
   - *Cost?*
   - *Complexity?*
4. ***Are CoProcessors a viable solution for any kind of problem?***

Introduction/Motivation:

- ***It's an active and recent topic in HPC Community***
- ***Related works:***
  - *M. Vasquez et al @ BSC. "Xeon Phi Performance for HPC-based Computational Mechanics Codes", Partnership for Advanced Computing in Europe*
  - *Venetis et al @ NTUA/Greece and TUD/Germany. "Porting FEASTFLOW to the Intel Xeon Phi: Lessons Learned"*
  - *J. Waltz @ LANL. "Performance Analysis of a 3D Unstructured Mesh Hydrodynamics Code on Multi- and Many-Core Architectures to appear in IJNMF*



Intel® Software Academic Program

A wealth of resources at your fingertips including grants, tools for your classroom, compliment software, and more.

Intel® Parallel Computing Centers

Apply | Current Centers | News

To meet future scientific computing demands, systems in the next decade will support millions processor cores with thousands of threads. But increased compute power will only get us so fa Future performance gains will also come through parallelism, and modernizing key application will help us make the next leap in discovery. Code modernization is expected to enable large performance increases while maintaining the code portability users expect.

Description:
Intel® Parallel Computing Centers (IPCC) are universities, institutions, and labs that are leade their field, focusing on modernizing applications to increase parallelism and scalability through optimizations that leverage cores, caches, threads, and vector capabilities of microprocessors coprocessors.

Eligibility:
All technical computing application modernization proposals are eligible. Intel is particularly interested in modernization of life sciences, energy, financial services, and visualization applications.

Current Intel® Parallel Computing Centers

Click on the logos to learn more about what each of these Intel® Parallel Computing Centers is doing.

सी डैक CDAC
CERFACS
CINECA
COPPE 50 UFRJ
CSC
epcc
ETH Zürich
Georgia Tech
ICHEC
Indian Institute of Science भारतीय विज्ञान संस्थान
STATE
Johns Hopkins Whiting School of Engineering
KiSTi 한국과학기술정보연구원
Berkeley Lab
lrz TUM
LSU
PennState
Purdue University
SDSC San Diego Supercomputer Center

What are we interested in?

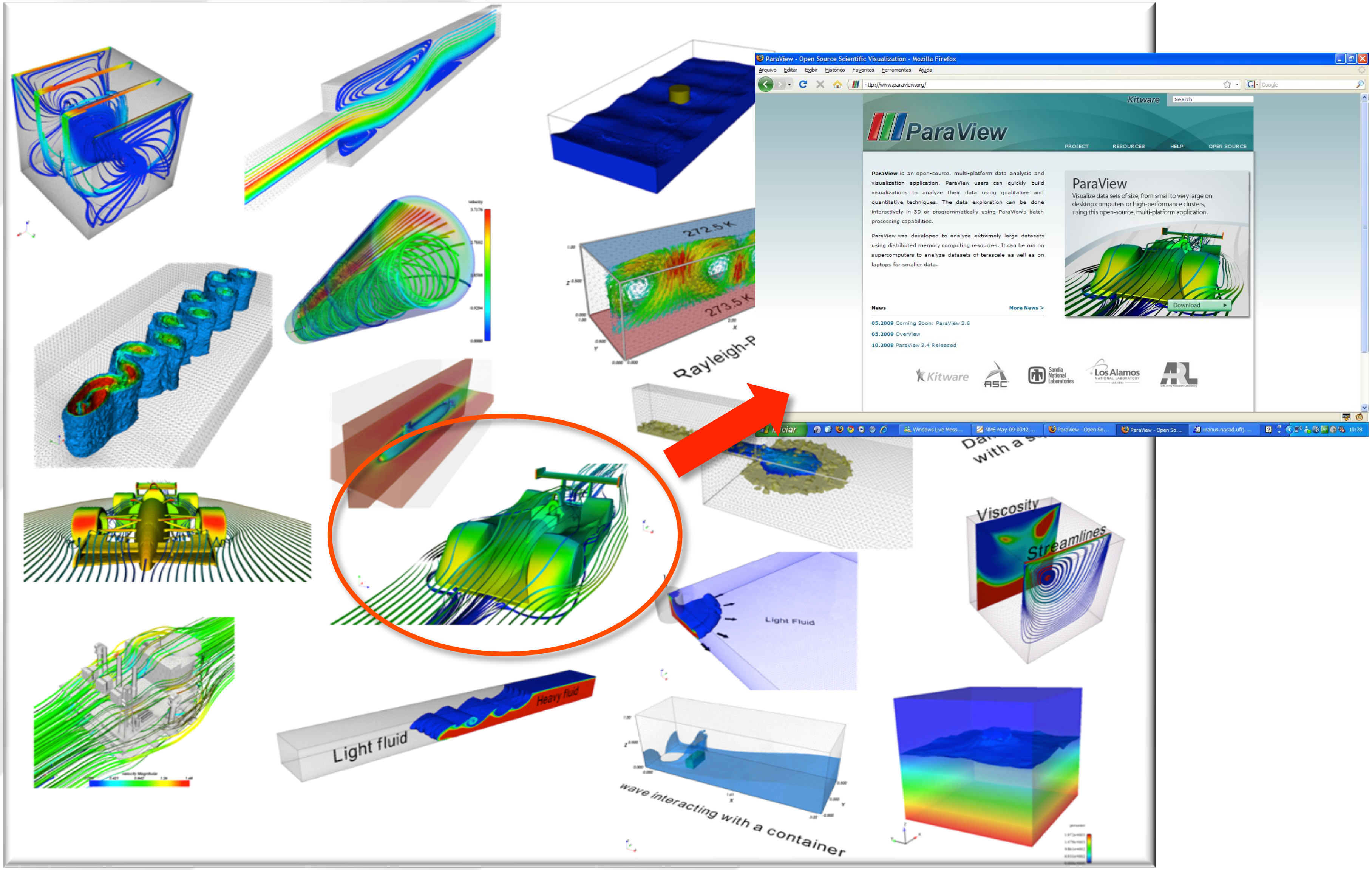
- ❑ ***Prepare our software for these new architectures;***
- ❑ ***EdgeCFD: A Stabilized Finite Element Simulator for Fluid Flow Problems.***
  - *Inherit/incorporate results from theoretical and applied researches in our group*
  - *Helps other projects as a simulation tool*

***Edge**: Data structure used in the software*

+

***CFD**: Computational Fluid Dynamics*

EdgeCFD in Action



ParaView is an open-source, multi-platform data analysis and visualization application. ParaView users can quickly build visualizations to analyze their data using qualitative and quantitative techniques. The data exploration can be done interactively in 3D or programmatically using ParaView's batch processing capabilities.

ParaView was developed to analyze extremely large datasets using distributed memory computing resources. It can be run on supercomputers to analyze datasets of terascale as well as on laptops for smaller data.

ParaView

Visualize data sets of size, from small to very large on desktop computers or high-performance clusters, using this open-source, multi-platform application.

News More News >

05.2009 Coming Soon: ParaView 3.6

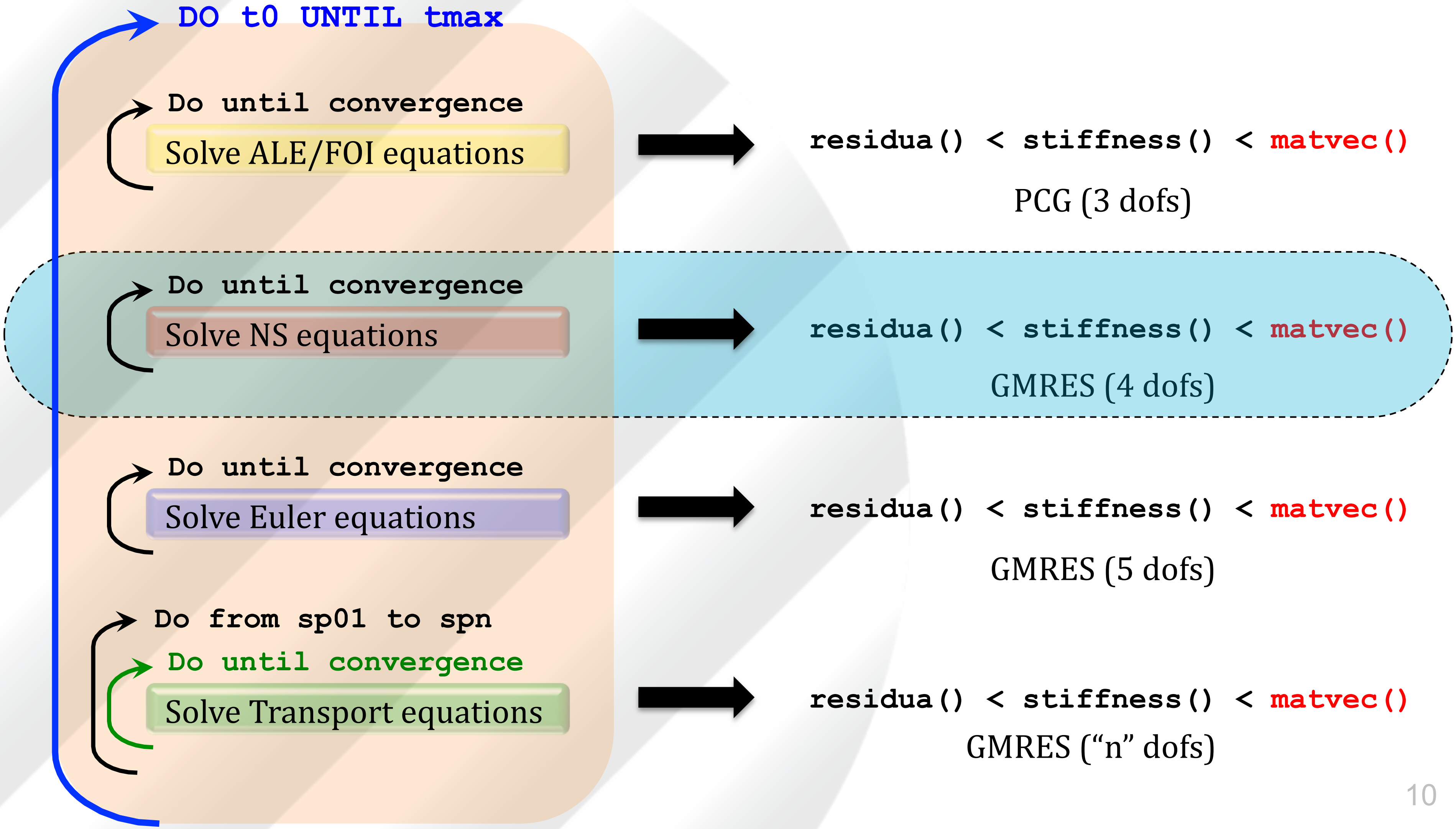
05.2009 OverView

10.2008 ParaView 3.4 Released

Some "Goodies and Baddies"

- ***Goodies*:**
  - *Built as a hybrid parallel code **from scratch***
  - ***Highly portable:** *any* Fortran compiler is enough to compile…*
    - *We extremely minimize external library dependencies;*
    - *Only Metis and MPI is required;*
    - *Build process is very simple for any platform (Mac Osx, Windows or *Nix) ☺*
  - ***Efficient:** at least, Intel Vtune and TAU say that… ☺*
    - *It's able to apply data ordering according to the different architecture.*
    - *Employs fast non-linear and linear solvers*
    - *Efficient data structure (by edges)*
    - *Relies on good programming practices for HPC architectures*
- ***Baddies***
  - *No Open Source ☹*
  - *No "fancy" **UI's** ☹ (pre-processing → GMSH and post-processing → ParaView)*
  - *Only **linear tetrahedra** elements are supported… (but any element can be dissassembled in edges)*

Loops and Computational Costs…



DO t0 UNTIL tmax

Do until convergence

Solve ALE/FOI equations

residua() < stiffness() < matvec()

PCG (3 dofs)

Do until convergence

Solve NS equations

residua() < stiffness() < matvec()

GMRES (4 dofs)

Do until convergence

Solve Euler equations

residua() < stiffness() < matvec()

GMRES (5 dofs)

Do from sp01 to spn

Do until convergence

Solve Transport equations

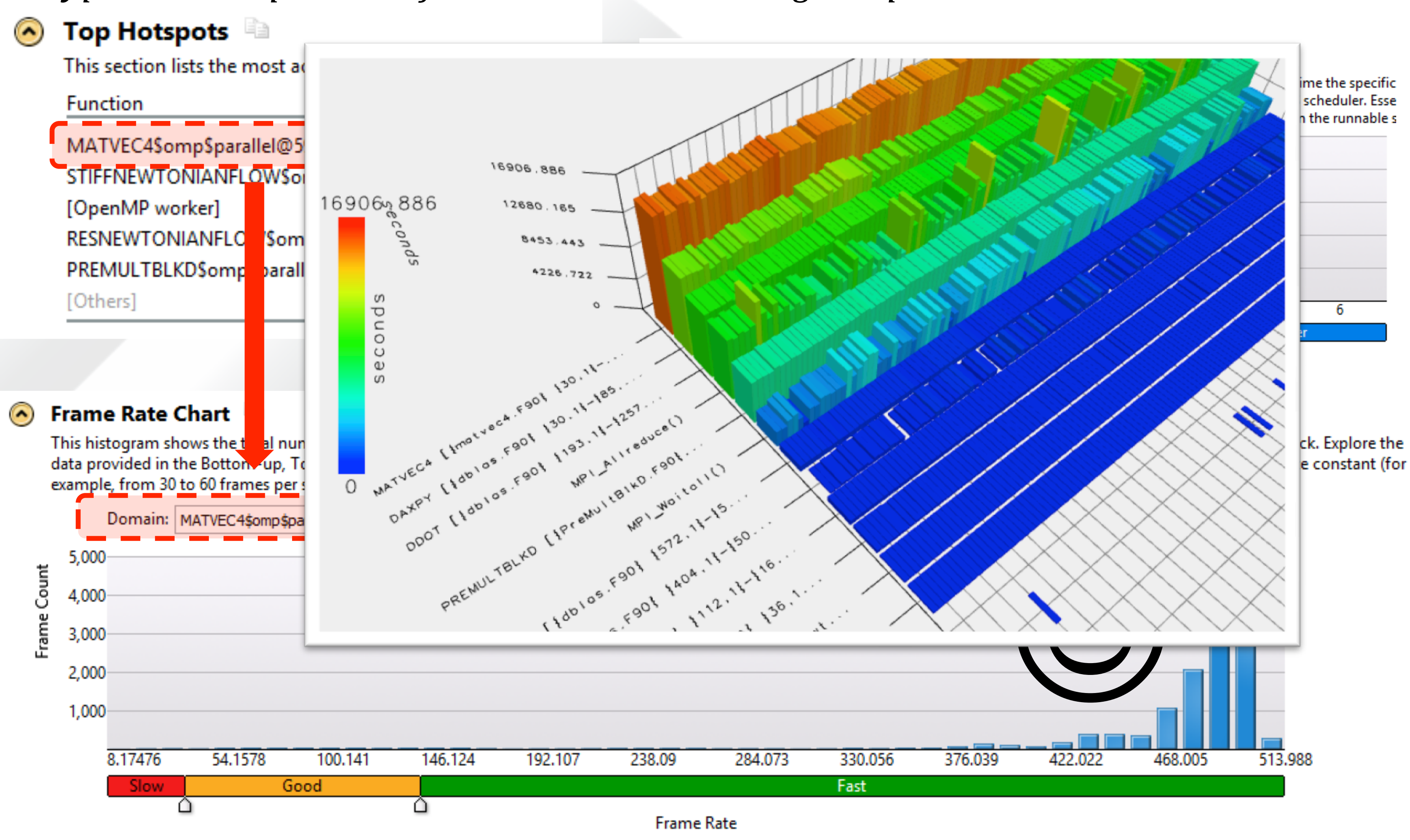
residua() < stiffness() < matvec()

GMRES ("n" dofs)

Dof = degrees of freedom

Performance analysis

Typical incompressible flow simulation running in OpenMP



Top Hotspots

This section lists the most a

Function

MATVEC4$omp$parallel@5

STIFFNEWTONIANFLOW$o

[OpenMP worker]

RESNEWTONIANFLO $om

PREMULTBLKD$omp arall

[Others]

ime the specific
scheduler. Esse
n the runnable s

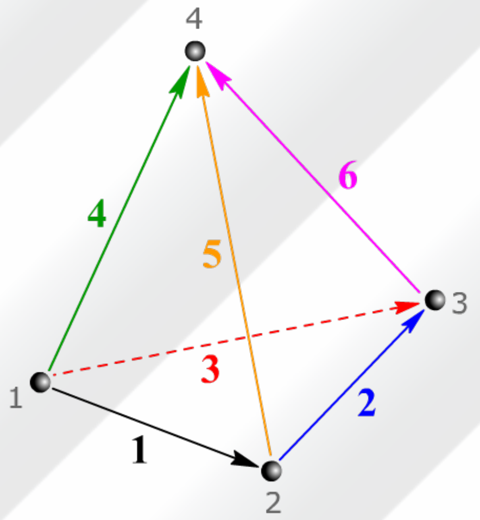
Frame Rate Chart

This histogram shows the t al nu
data provided in the Botton up, To
example, from 30 to 60 frames per s

ck. Explore the
e constant (for

Domain: MATVEC4$omp$pa

EDS: Edges-by-Edge (cont.)



"…algebraic disassembling of a tet into edges…"

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

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

$$\mathbf{T}_1^e = \begin{bmatrix} \mathbf{T}_{11}^1 & \mathbf{T}_{12}^1 & 0 & 0 \\ \mathbf{T}_{21}^1 & \mathbf{T}_{22}^1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{T}_2^e = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \mathbf{T}_{22}^2 & \mathbf{T}_{23}^2 & 0 \\ 0 & \mathbf{T}_{32}^2 & \mathbf{T}_{33}^2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

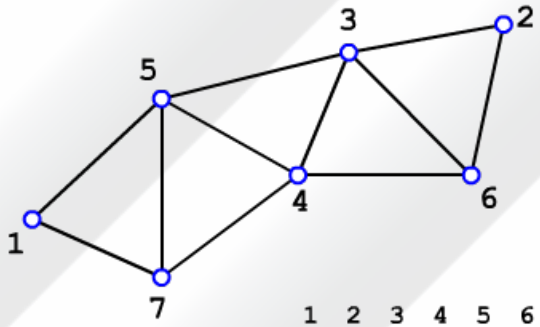
$$\mathbf{T}_4^e = \begin{bmatrix} \mathbf{T}_{11}^4 & 0 & 0 & \mathbf{T}_{14}^4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{T}_{41}^4 & 0 & 0 & \mathbf{T}_{44}^4 \end{bmatrix} \qquad \mathbf{T}_3^e = \begin{bmatrix} \mathbf{T}_{11}^3 & 0 & \mathbf{T}_{13}^3 & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{T}_{31}^3 & 0 & \mathbf{T}_{33}^3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{T}_5^e = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \mathbf{T}_{22}^5 & 0 & \mathbf{T}_{24}^5 \\ 0 & 0 & 0 & 0 \\ 0 & \mathbf{T}_{42}^5 & 0 & \mathbf{T}_{44}^5 \end{bmatrix} \qquad \mathbf{T}_6^e = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{T}_{33}^6 & \mathbf{T}_{34}^6 \\ 0 & 0 & \mathbf{T}_{43}^6 & \mathbf{T}_{44}^6 \end{bmatrix}$$

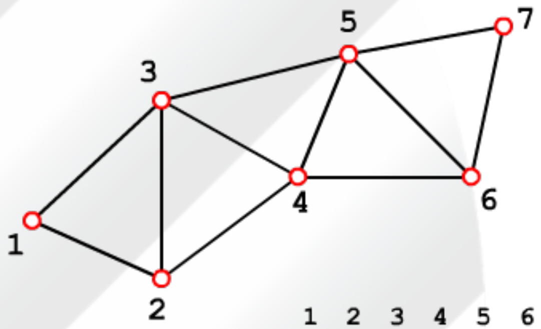
In practice, nodal block-diagonals are also used to build preconditioner

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*

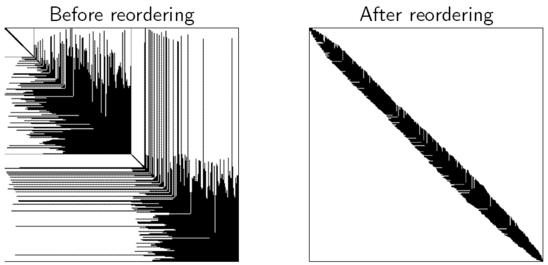


	1	2	3	4	5	6	7
1	■				■		■
2		■	■			■	
3		■	■	■	■	■	
4			■	■	■	■	■
5	■		■	■	■		■
6		■	■	■		■	
7	■			■	■		■



	1	2	3	4	5	6	7
1	■	■	■				
2	■	■	■	■			
3	■	■	■	■	■		
4		■	■	■	■	■	
5			■	■	■	■	■
6				■	■	■	■
7					■	■	■

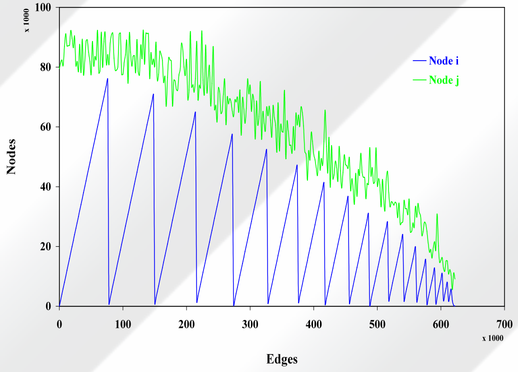
basic example

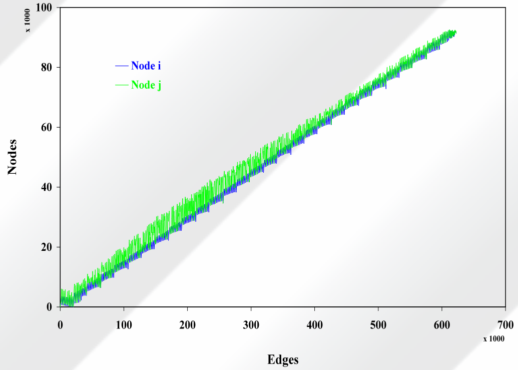


real problem

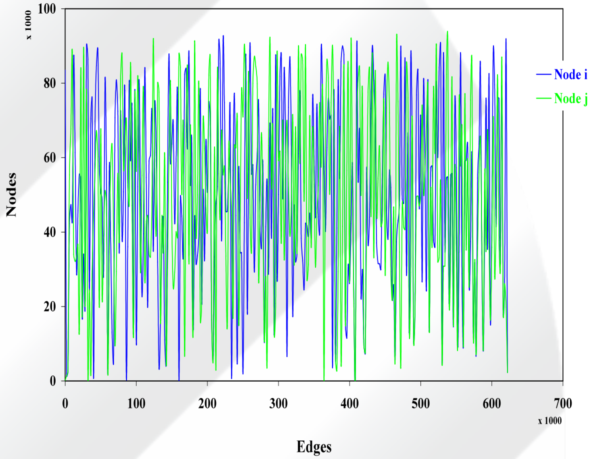
Memory Access Patterns

Reordering for minimizing i/a





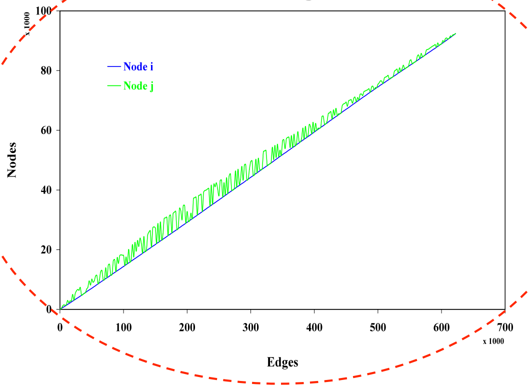
Improving data locality with NO memory dependencies

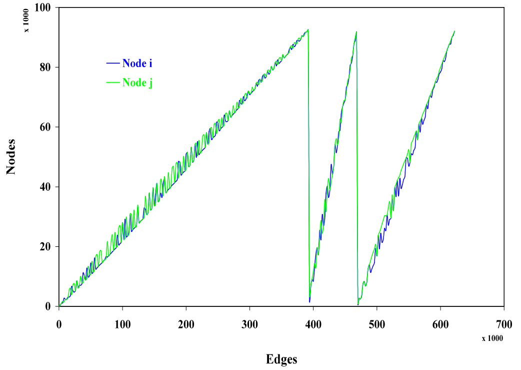


Original edge order

Data reordering schemes provided by *EdgePack®*

Improving data locality: RCM vertex reordering and edge reordering in ascending vertex order

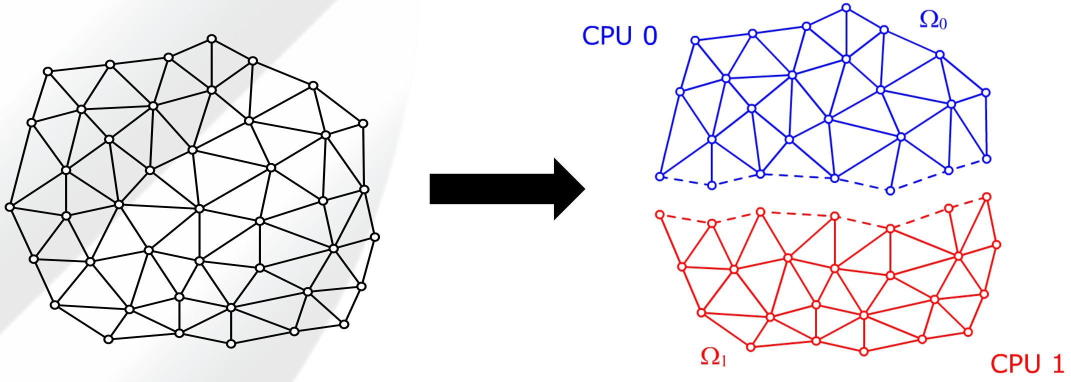




Improving data locality for each superedge type. Superedge improves the use of CPU registers

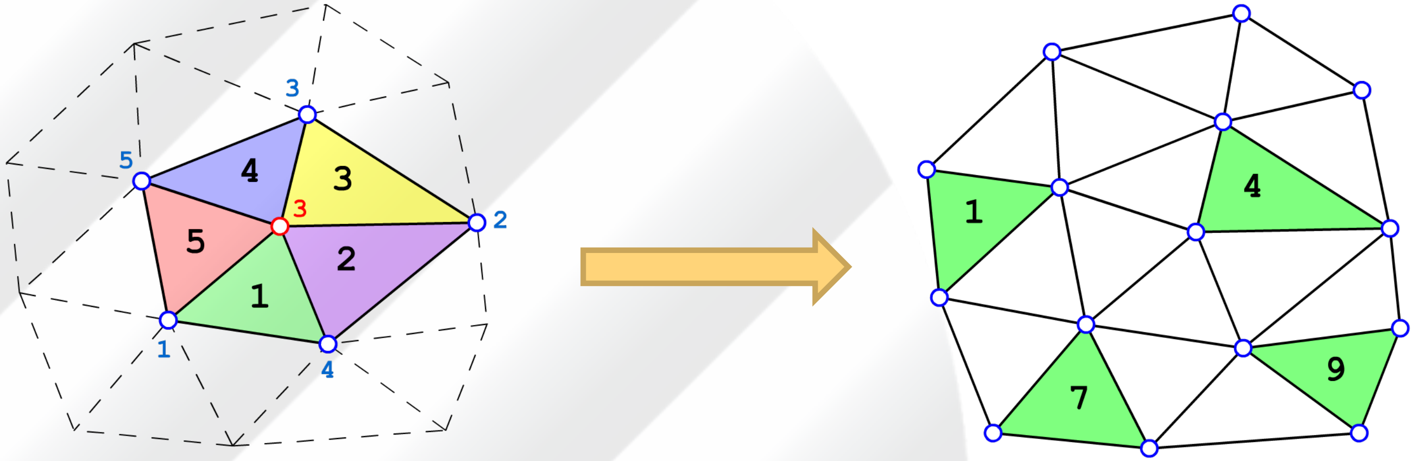
Distributed Memory Parallelism

- ***Standard Approach:***
  - *Takes a mesh and give it to a partitioner (**Metis**)*
  - *Once partitioned, **keeps the data parallel***
  - *Global IDs → **Local IDs***
  - *Each process records its own files for **parallel I/O***
  - *Shared information is synchronized by **p2p non-blocking communications***
  - *No ghost/halo information employed*



Shared Memory Parallelism

- ***Standard blocked loops to remove memory dependency***



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

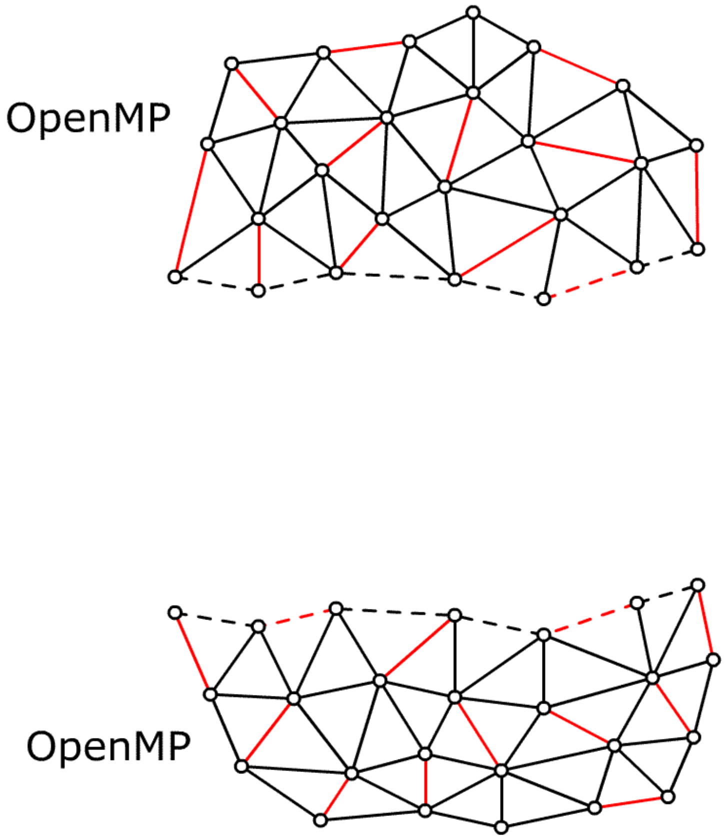
```
ielm = 0
do icor = 1, ncores
   nvec = ielblk(icor)
!$OMP  DO
   do i = ielm+1, ielm+nvec
      ! Retrieve element nodes
      x(no) = x(no) + a
   enddo
!$OMP END DO
   ielm = ielm+nvec
enddo
```

Hybrid Matrix-Vector Product

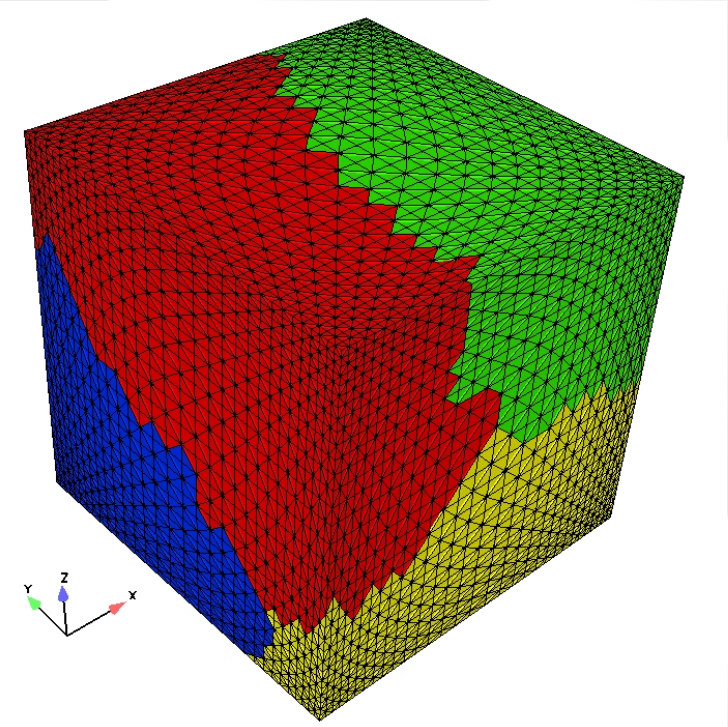
```
iside = 0
DO iblk = 1, nedblk
nvec  = ia_edblk(iblk)
!dir$ ivdep
!$OMP DO
   DO ka = iside+1, iside+nvec, 1
      ...MATVEC computations...
   ENDDO
!$OMP END DO
ENDDO
```

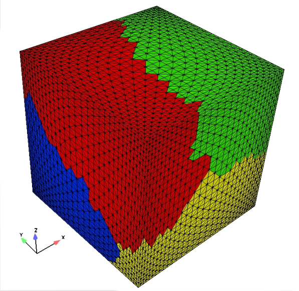
```
...over interface nodes...
#ifdef MPICODE
call MPI_AllReduce
#endif
```

Edge-by-Edge



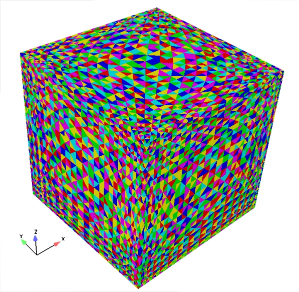
Data Structure for Hybrid Paralelism





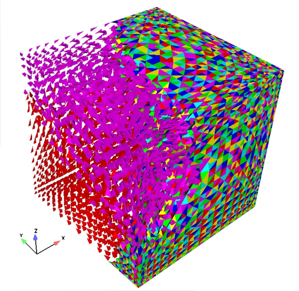
Metis partitioning

Distributed (or shared) memory with MPI



Mesh coloring

Vectorization and/or shared memory parallelism (OpenMP)



Data partitioning + Mesh coloring

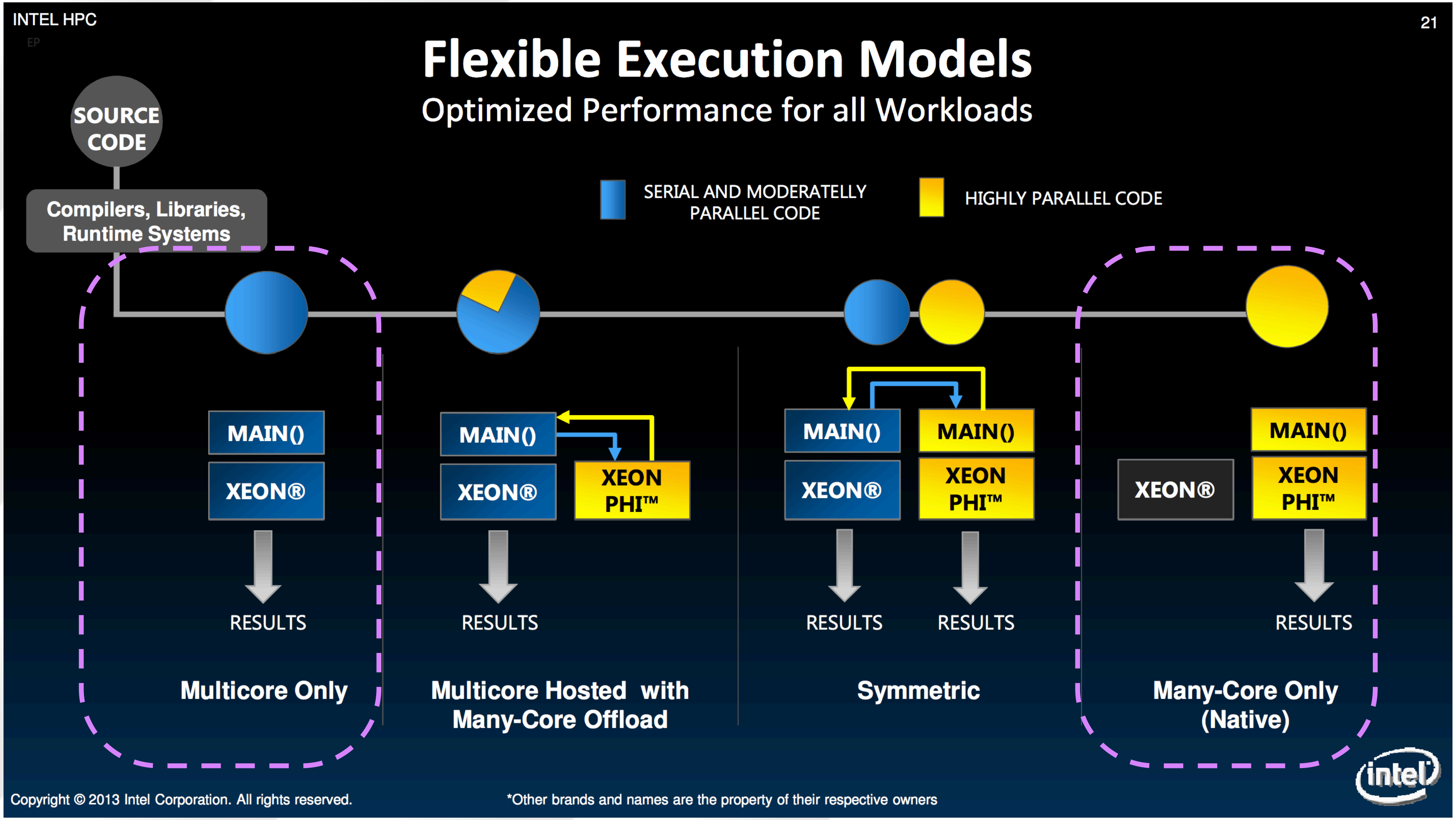
Intel Xeon Phi

- ***Intel's Many-Integrated-Core (MIC) architecture***
- ***In few words:***
  - *"A PCI express board with 57-61 cores supporting up to 4 threads each (228-244 threads per board)*
- ***Dense and simplified collection of (old) processors***
  - *Based on Pentium P54c x86 architecture, with 64-bits, vector capabilities and cache coherent.*
  - ***It was made to support existing HPC applications***
- ***Each board runs a simplified Linux***
  - *Has ip address and is fully functional. In other words, **it's a computing node inside a computing node***

EdgeCDF on Xeon-Phi

- ❑ ***As simple as call the compiler with a new compilation flag (`-mmic`) ☺***
- ❑ ***Ok, but does it scale? Let's see on next slides...***
- ❑ ***Why not GPGPU's?***
  - *It would require* ***deep structural code changes*** *(data structure, reordering schemes, etc...)*
  - *EdgeCFD is not focused on specific platforms. We seek for a* ***good and overall*** *performance to keep portability!*
  - *Planning some tests with PGI CUDA Fortran compiler @ CRAY*

Xeon-Phi Execution Models



INTEL HPC

EP

21

Flexible Execution Models

Optimized Performance for all Workloads

SOURCE CODE

Compilers, Libraries, Runtime Systems

🟦 SERIAL AND MODERATELLY PARALLEL CODE

🟨 HIGHLY PARALLEL CODE

MAIN()
XEON®
RESULTS
Multicore Only

MAIN()
XEON®
XEON PHI™
RESULTS
Multicore Hosted with Many-Core Offload

MAIN()
XEON®
MAIN()
XEON PHI™
RESULTS
RESULTS
Symmetric

XEON®
MAIN()
XEON PHI™
RESULTS
Many-Core Only (Native)

Copyright © 2013 Intel Corporation. All rights reserved.

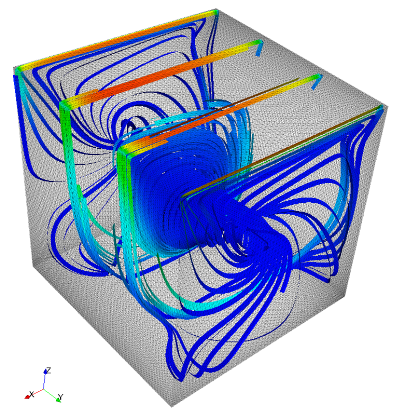
*Other brands and names are the property of their respective owners

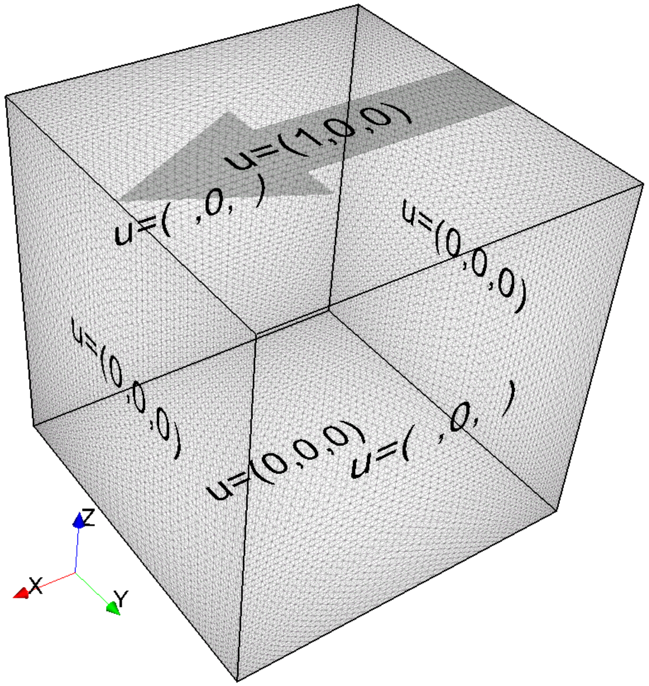
EdgeCFD Tests on Xeon Phi

- ***"Blind"/Naive test***
  - *Upload, compile and run! Nothing changed and/or tuned besides compiler optimizations (**-O3**)*
- ***Execution Models***
  - *Host only*
  - *Mic only*
- ***Parallel Models***
  - *OpenMP and MPI on Host*
  - *OpenMP and MPI on Mic*
  - *Hybrid on Mic*
- ***Only strong scalability!***
  - *How solution time behaves as the number of cores increase for a fixed effort/ problem.*

Test Problem

- ***Cavity Flow Problem (Incompressible Flow)***
  - **50 Time steps**
  - **Edges: 133,849**
  - **Tets.: 108,104**
  - **Nodes: 20,589**





NOTE

- *It's a small problem! Could be easily ran on a laptop!*

BUT…

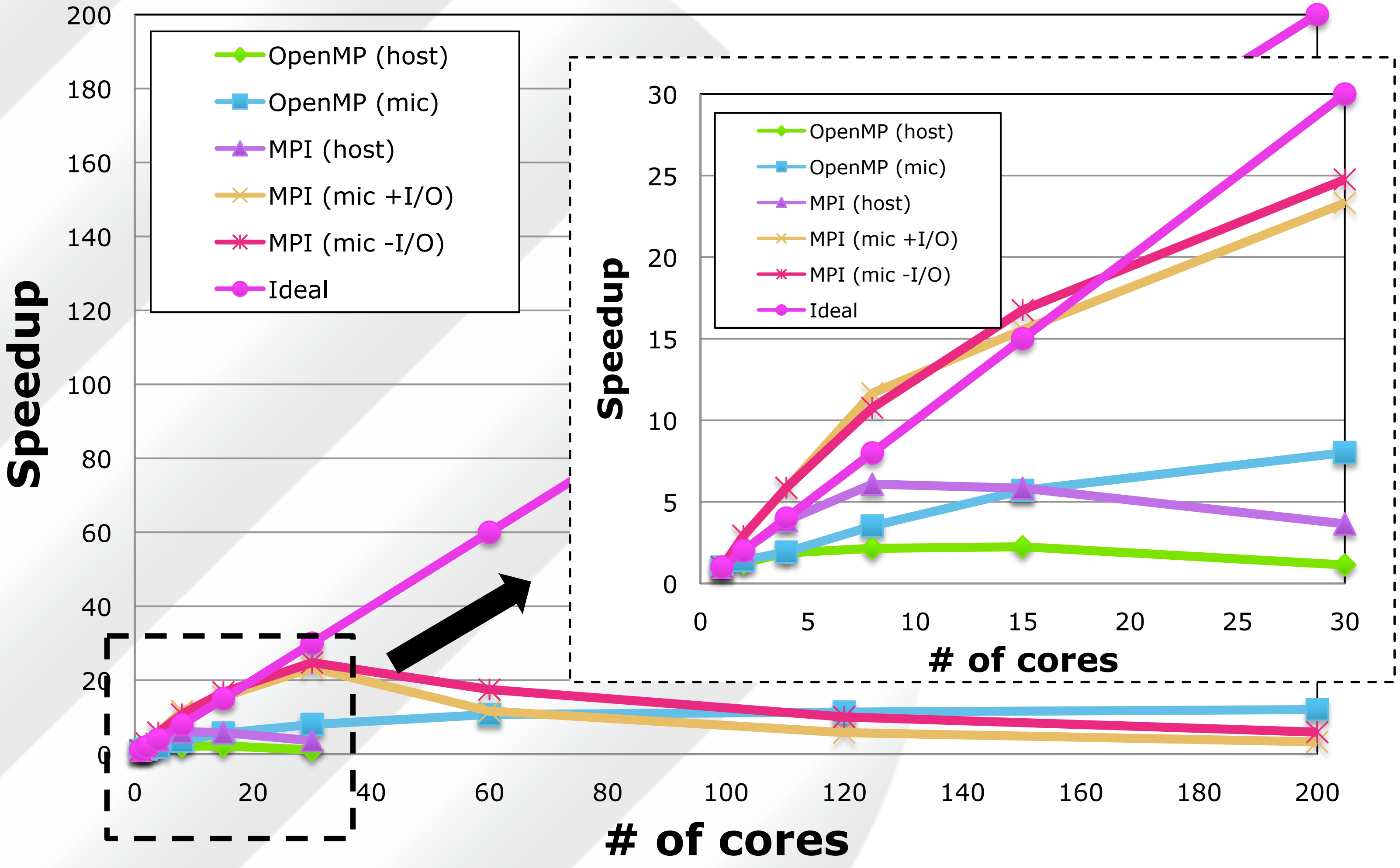
- *Can be considered as a MPI partition of a bigger problem placed on a computing node… **we're just trying to measure Xeon-Phi performance alone***

Hardware

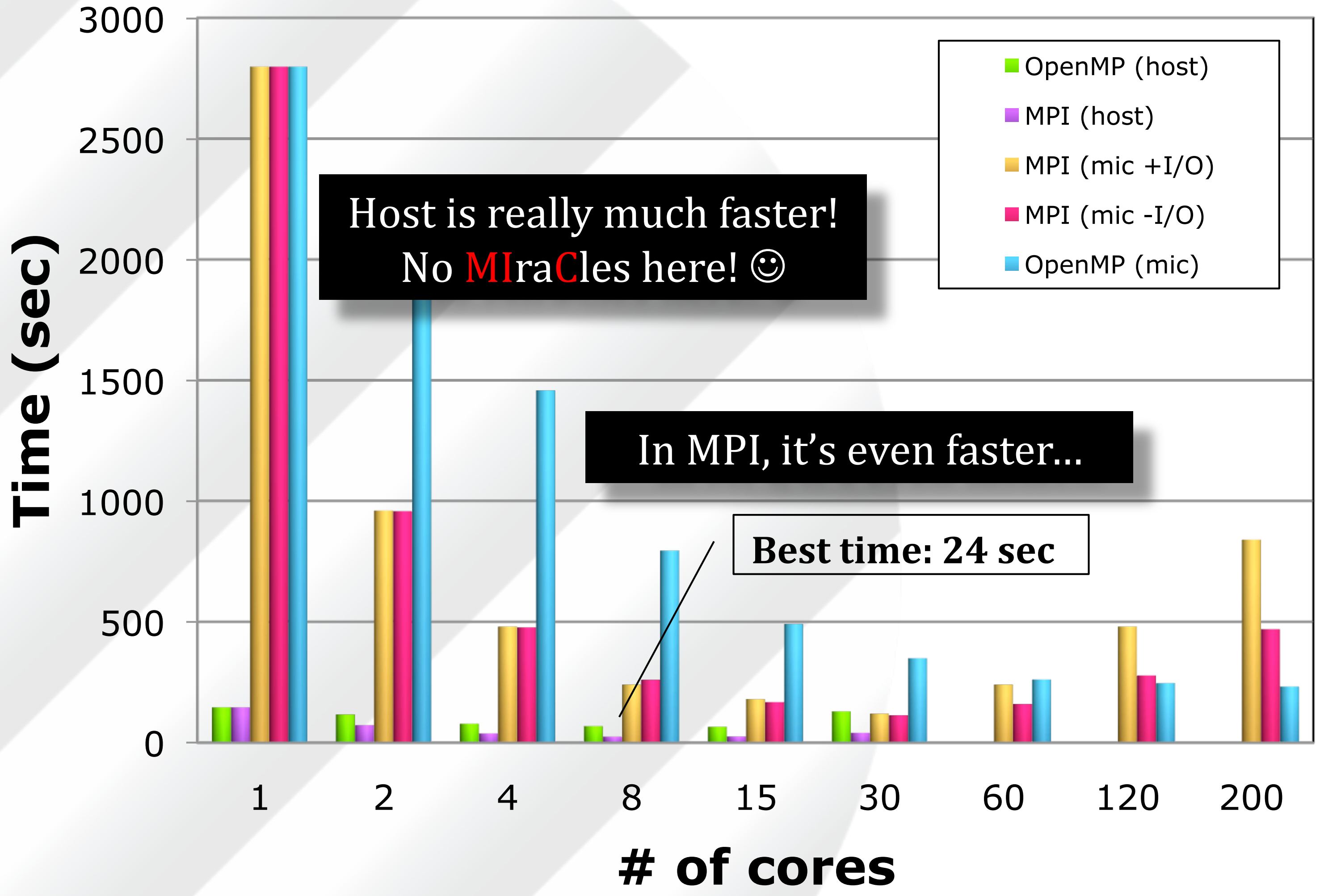
- ❑ ***HOST: Dell server (~$4.2k @ Amazon 2T14)***
  - *Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz*
  - *2 x 8 cores, 2 threads/core, up to 32 threads or **64 hyperthreaded***
  - *64GB memory*
- ❑ ***Xeon Phi 3120A CoProcessor (~ $1.55k @ Amazon 2T14)***
  - *6GB memory (12 memory channels @ 240GB/s)*
  - *28,5 MB cache memory*
  - *1.053 GHz Clockspeed*
  - *57 Cores / 4 threads per core = up to 228 Threads*

But..., hey, Xeon-Phi does not live alone
It needs a host anyway... ☹

SpeedUp


Speedup
200
180
160
140
120
100
80
60
40
20
0
0
20
40
60
80
100
120
140
160
180
200
# of cores
OpenMP (host)
OpenMP (mic)
MPI (host)
MPI (mic +I/O)
MPI (mic -I/O)
Ideal
Speedup
30
25
20
15
10
5
0
0
5
10
15
20
25
30
# of cores

Host x Mic Performance

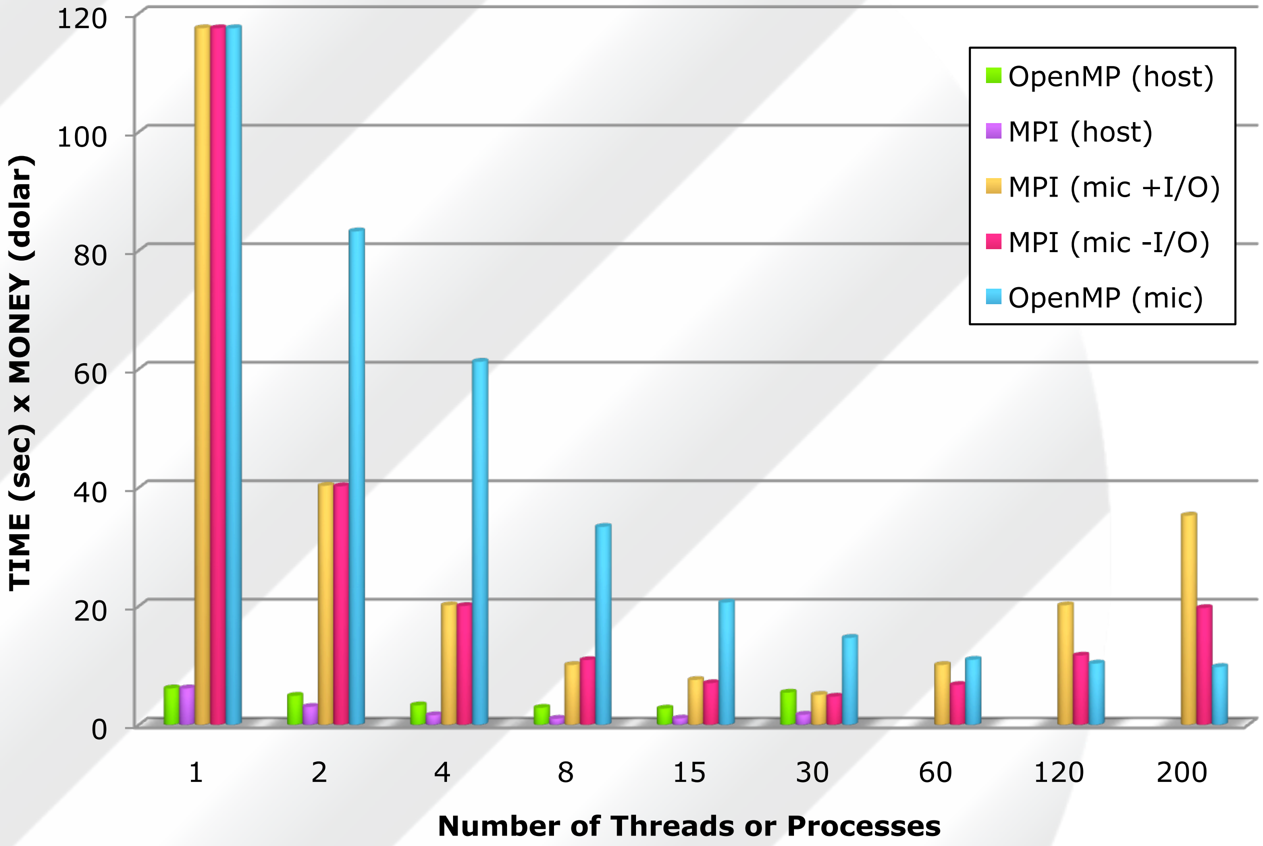


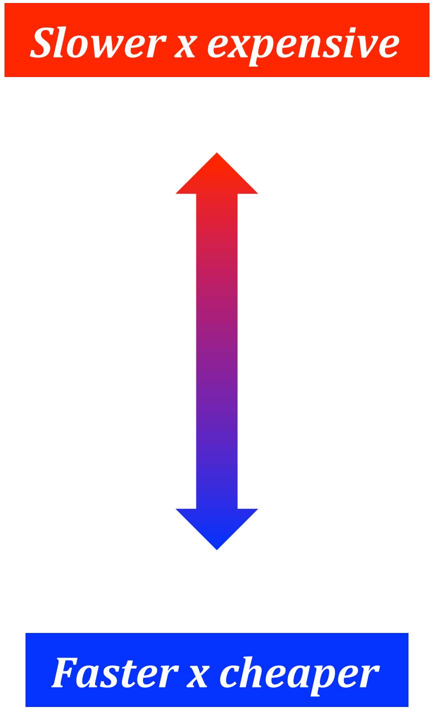
Host is really much faster!
No MIraCles here! ☺

In MPI, it's even faster...

Best time: 24 sec

Money x Time Comparison



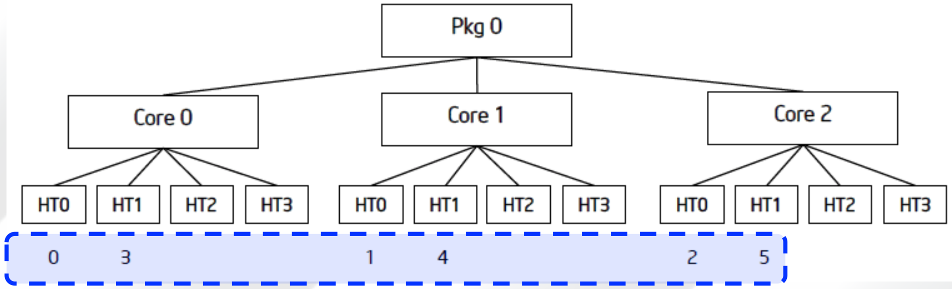


Host: ~$4.2k @ Amazon 2T14
Xeon-Phi: ~$1.55 @ Amazon 2T14

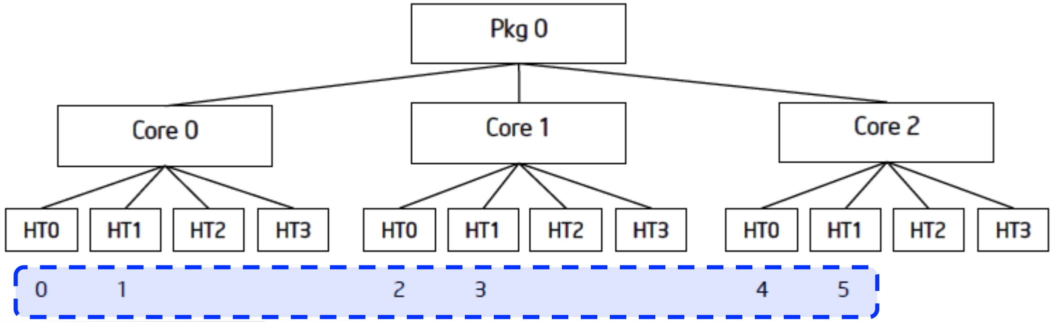
NOTE: Mics need a host but it was considered alone in this comparison

Thread Affinity on Xeon-Phi

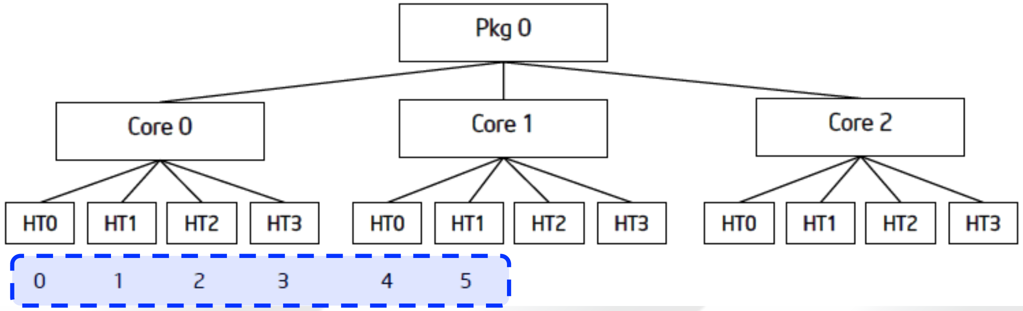
Scatter



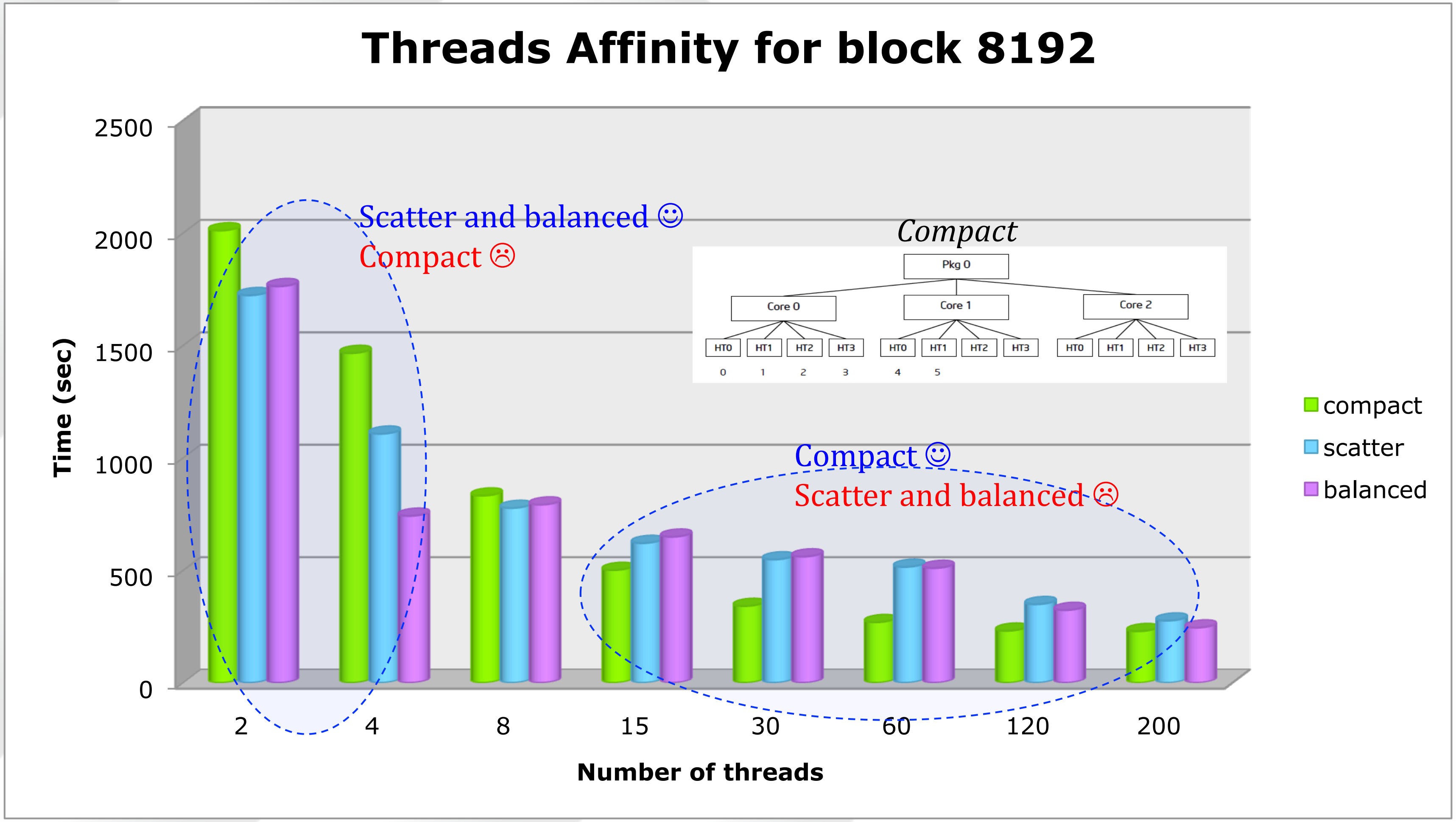
Balanced



Compact



How does thread affinity affect performance?



Scatter and balanced ☺

Compact ☹

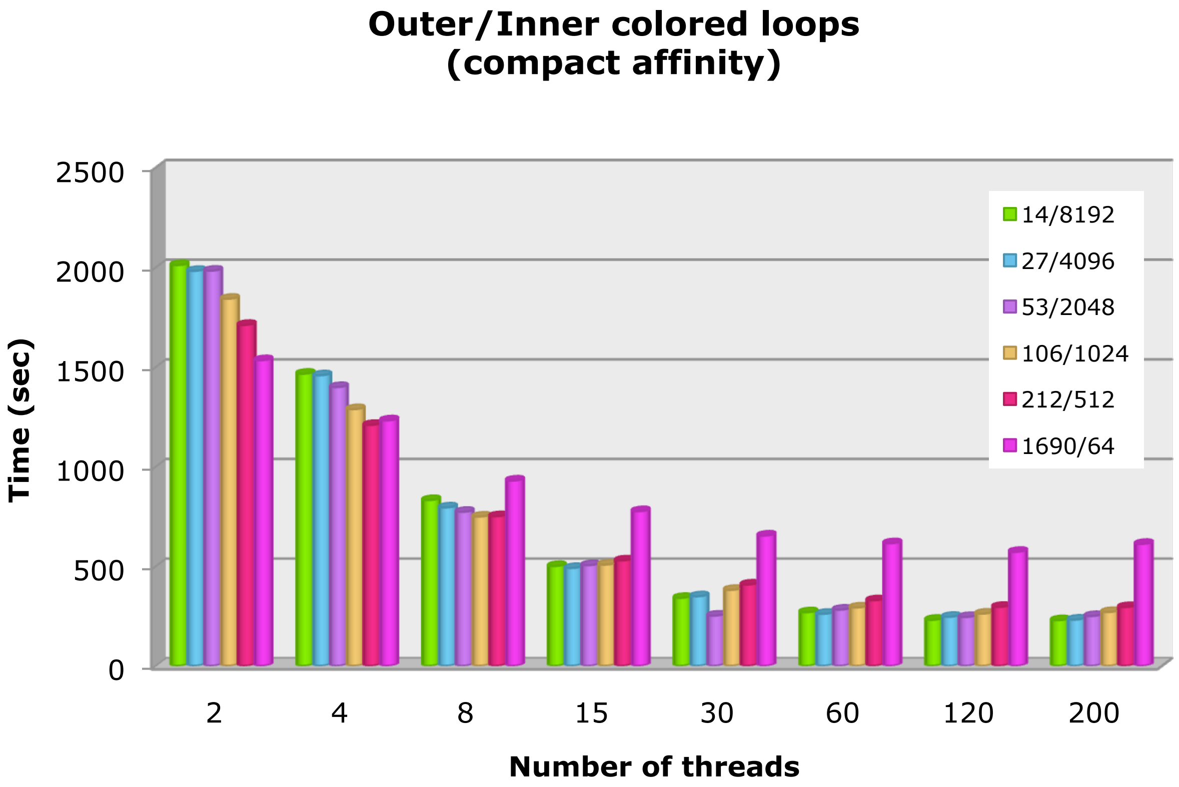
Compact

Compact ☺

Scatter and balanced ☹

How does block size affect EdgeCFD Performance

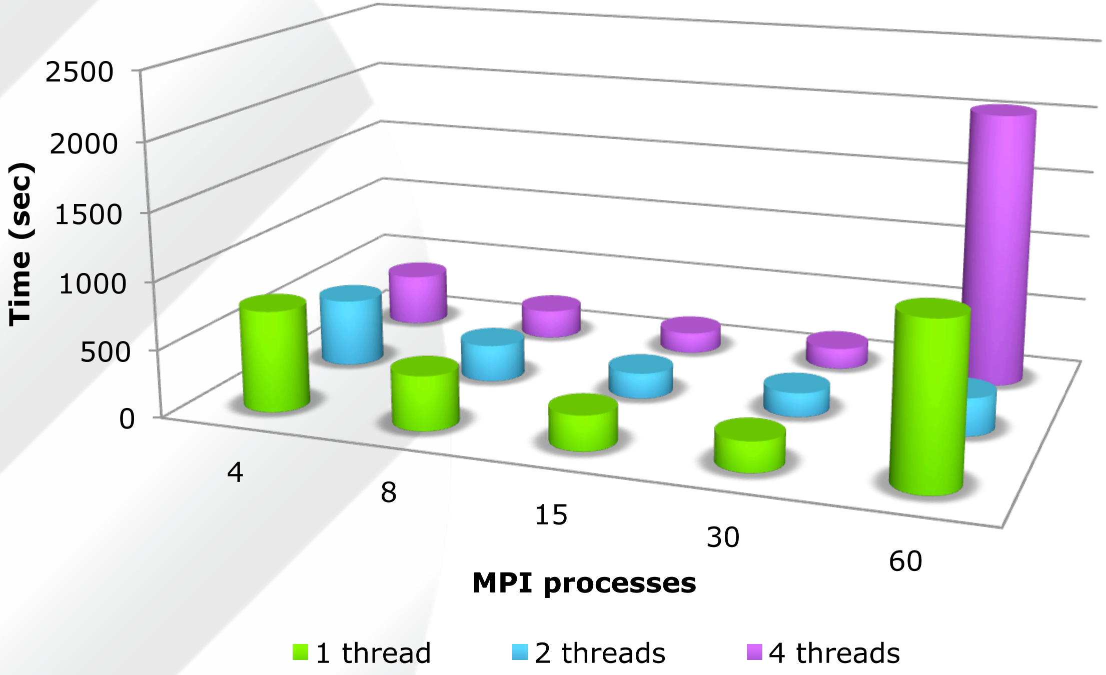
- ❑ *Block sizes => Bigger blocks means denser inner loops and less barrier synchronization*



```
iside = 0
DO iblk = 1, nedblk
nvec  = ia_edblk(iblk)
!$OMP DO
   DO ka = iside+1, iside+nvec, 1
      ...MATVEC computations...
   ENDDO
!$OMP END DO ← BARRIER!!!!
iside = iside + nvec
ENDDO
```

Hybrid Performance on Xeon-Phi

- *Parallel I/O cost is inversely proportional to the number of MPI process;*
- *60 threads trying to write on the same disk at once* ☹
- *Solution: CoProcessing could help to overcome this problem , but computational mechanics software still needs to save some data on disk…*



Summary and Conclusions (1/3)

	Best @ Host		Best @ Mic	
Overall	24.0	MPI / 8 procs	113.0	MPI / 30 procs -I/O
OpenMP	65.0	15 threads	232.0	200 threads
MPI	24.0	8 procs	113.0	30 procs
cost x time	1.0	$ seconds	5.0	$ seconds

- *Xeon-Phi could be **5x cheaper or 5x faster** from this comparison*
  - ***Cost-wise**, for our application, **regular processors are the best investment** (yet!)*
- ***New algorithms** for improving many-cores parallelism are necessary **for unstructured methods** running on newer architectures based on Mics;*
  - *Or we should leave Mics for other more specific applications (codes with predictable memory access pattern…)*

Conclusions (2/3)

- ❑ ***Xeon-Phi is indeed a flexible*** *alternative to get into microprocessors world (but* ***not the most efficient*** *at first… ☹)*
  - *Microprocessors pose other options and complexity about parallel and vectorization combinations;*
  - *Flexibility comes with a (high) price.*
- ❑ *Some* ***parameters*** *on EdgeCFD may affect* ***OpenMP performance***
  - *outer/inner blocked loops ratio;*
  - *Memory access pattern (not deeply investigated yet…);*

Conclusions (3/3)

- ***Thread affinity** is also important when squeezing the best **performance on Xeon-Phi**.*
- *Xeon-Phi, although support MPI **natively**, it must be used with care due to I/O costs...*
  - *Offloads to host might mitigate this issue...*
- *In very few words, the **main issues using Xeon-Phi natively** are:*
  - *Memory access pattern for unstructured mesh methods;*
  - *Thread synchronization in OpenMP;*
  - *Disk use in MPI.*

Trying to Answer Our Questions:

- ***Are old applications ready for these new HPC systems?***
  - *Definitely not!*
- ***How could/should we take advantage of them?***
  - *We don't know (yet). But we already have some guesses….*
- ***How do CoProcessors compare to "traditional" CPUs?***
  - *They are (much) slower and expensive as well*
- ***Are CoProcessors a viable solution for any kind of problem?***
  - *Our results pointed that they're more suitable for applications with a predictable memory access pattern;*
  - *We have room for improvements in new algorithms here…*

Thanks for your attention!

Acknowledgment to:

National Council for Scientific and Technological Development CNPq/Brazil Proj.: 482886/2012-9 for the financial funding

High Performance Computing Center (NACAD/COPPE/UFRJ) for the Computational Resources