9th Advanced School on PARALLEL COMPUTING

Hybrid MPI+OpenMP Optimization,BGQ exploitation

SPEED – TIGRA - OpenFOAM

Paride Dagna – p.dagna@cineca.it
SuperComputing Applications and Innovation Department

CINECA

February 11 - 15, 2013

Outline

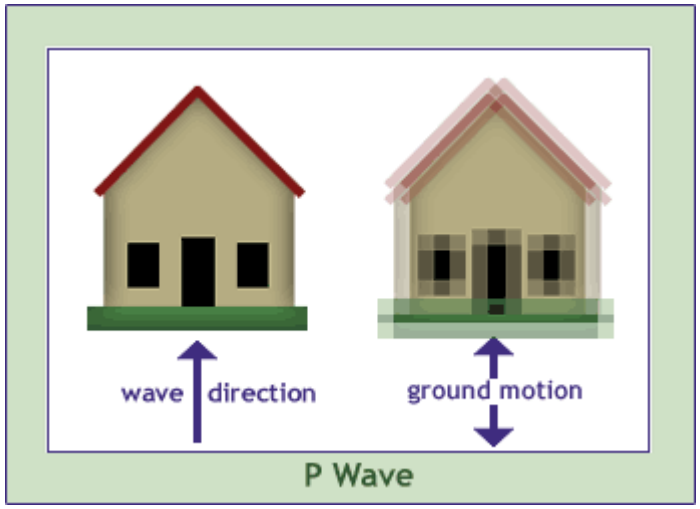
- **SPEED**
  - Introduction
  - BGQ Optimization strategy
  - Benchmark
  - Future work

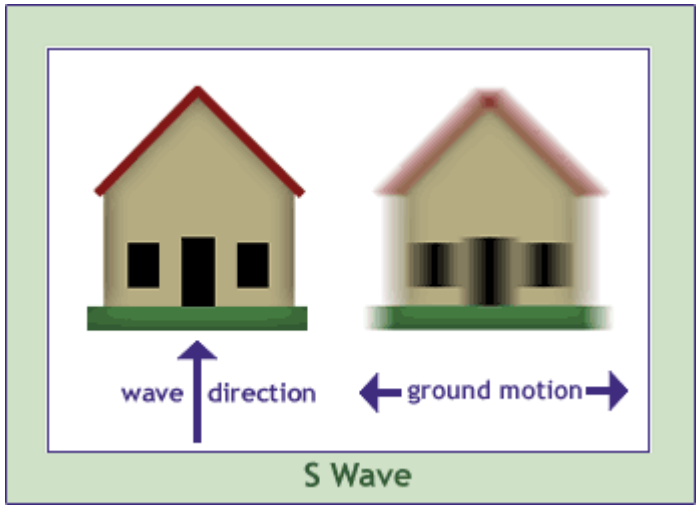
Introduction

- SPEED (SPectral Elements in Elastodynamics with discontinuous Galerkin)
  - open-source numerical code for the simulation of seismic wave propagation in two and three-dimensional heterogeneous geological media
  - Fortran
  - METIS for mesh partitionig
  - MPI communication library
  - OpenMP directives (inter-node)
- designed with the aim of simulating large-scale seismic events, allowing the evaluation of the typically multi-scale wave propagation problems

Introduction

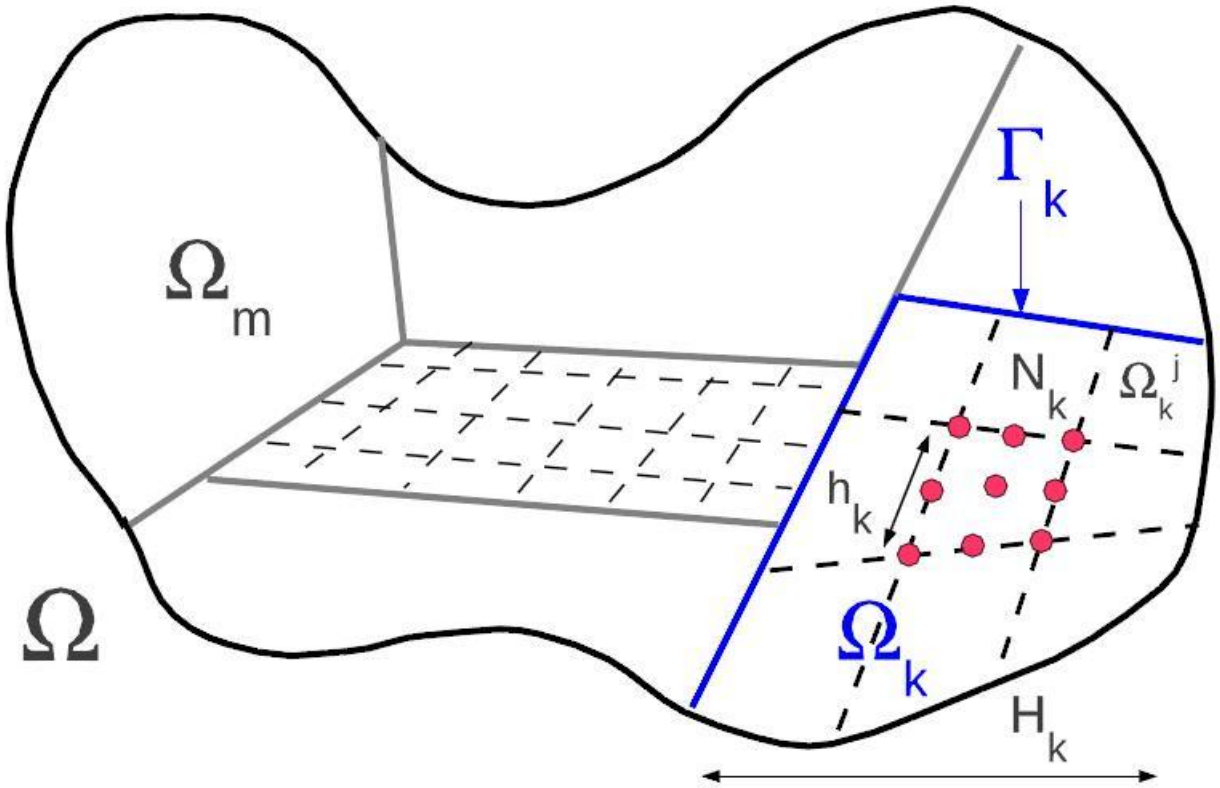
- Seismic waves are propagating vibrations that carry energy from the source of the shaking outward in all directions.
- There are different kind of seismic waves. The most important ones are:
  - Compressional or P (primary)
  - Transverse or S (secondary)
- An earthquake radiates P and S waves in all directions.





Numerical Approach

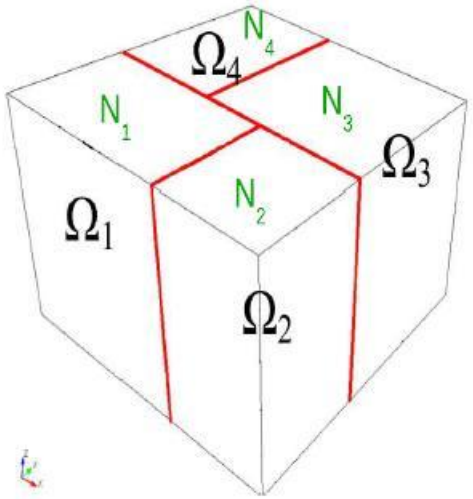
- Discontinuity across macro-domains.

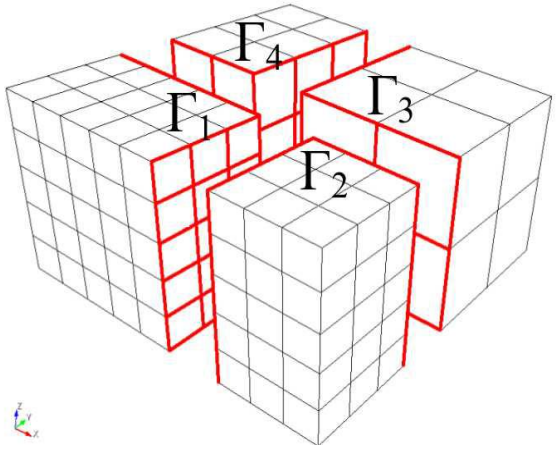


- Continuity within macro-domains (union of conforming spectral elements).

Numerical Approach

- SubdivideΩinto R non-overlapping macro-regions Ω_k: i.e., $\overline{\Omega} = \bigcup_{k=1}^{R} \overline{\Omega}_k$
- For each define a polynomialapproximationoforder .
- For each k , introduce a conforming partition : $\mathcal{T}_{h_k}$, i.e., $\overline{\Omega}_k = \bigcup_{j=1}^{J_k} \overline{\Omega}_k^j$
- Subdivide the skeleton in M elementary components (faces) as intersection of interfaceboundaries
- Non-Conforming Spectral Element Method in elastodynamics
- DiscontinuousGalerkinSpectralElementMethod
  - Weakly continuity across interfaces (polynomials are globally discontinuous)
  - Non-conformingmeshes
  - Non-uniformpolynomialapproximationdegrees





Christchurch earthquake 22.02.2011

New Zealand

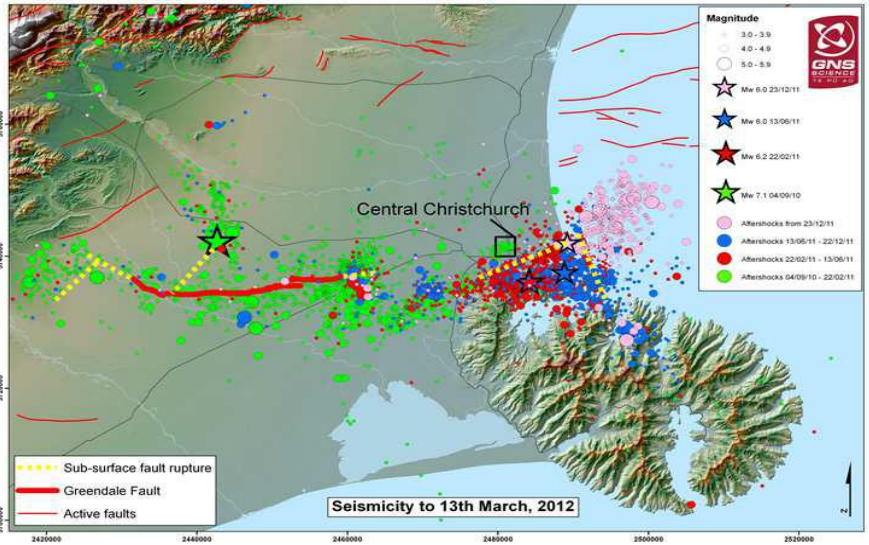
Canterbury Plains

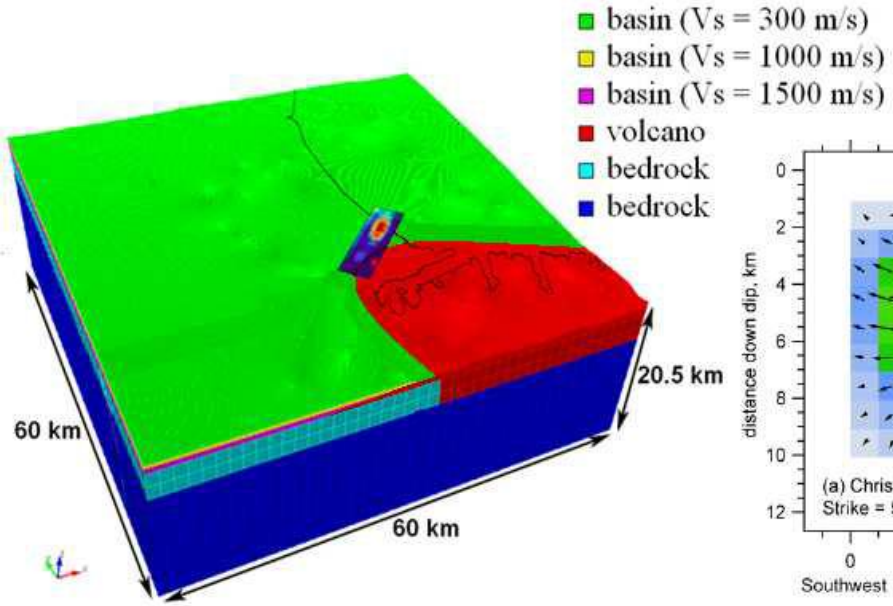
Christchurch (CBD)

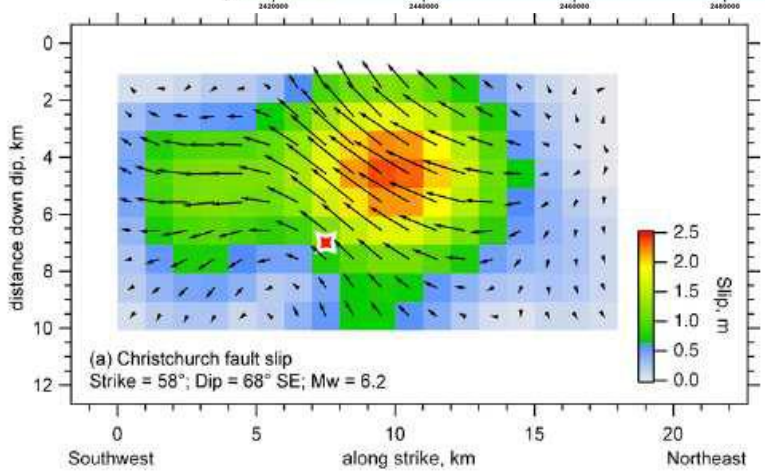
- Geological map of Christchurch provided by GNS (New Zealand research institute) [Forsyth et al. 2008]
- Geological cross-section: numbers refer to the stations within a 40 km radius from the epicenter

From the geological to the 3D model

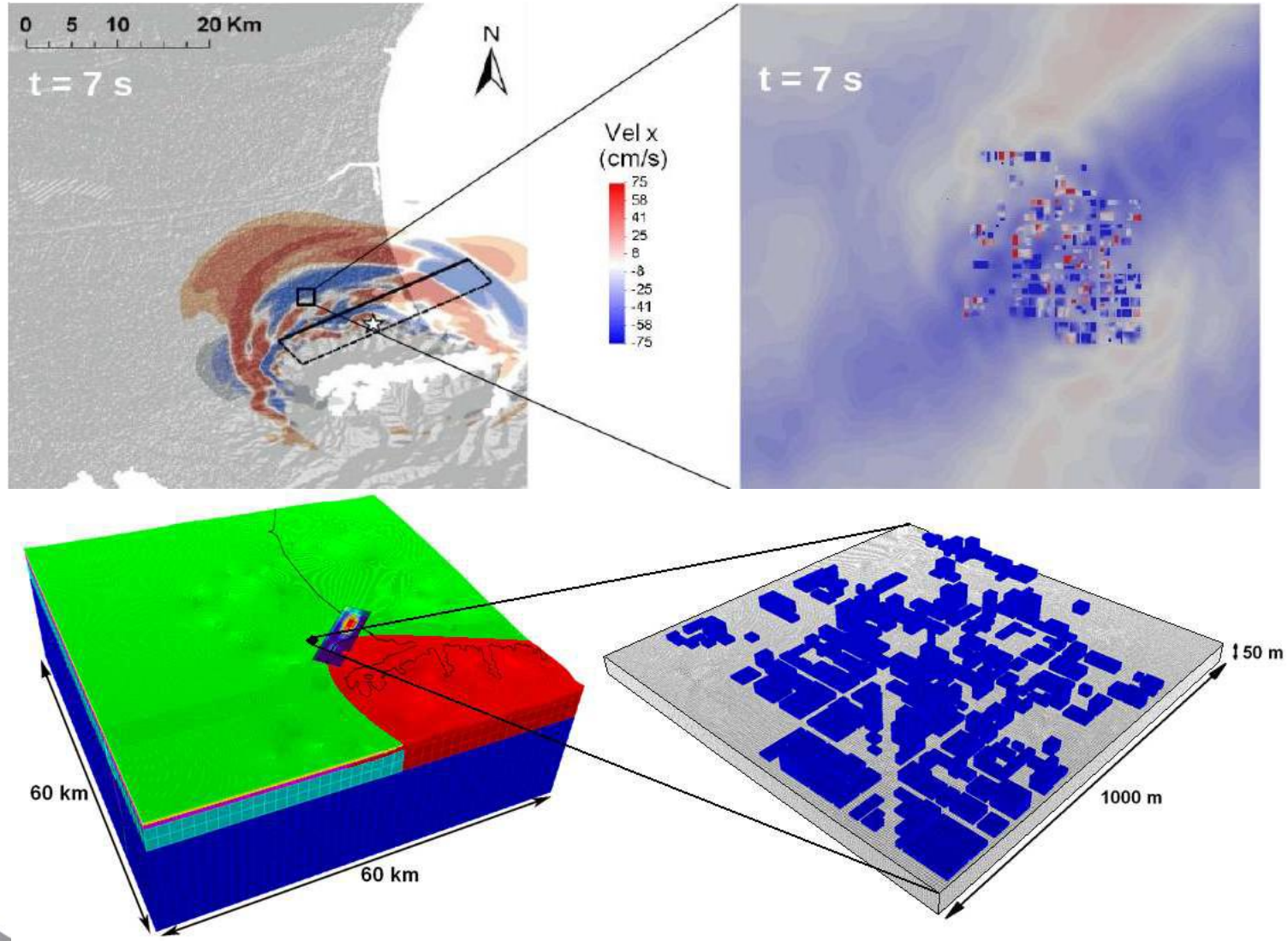
- **Input**
  - 3D geological and geophysical data describing the deep and sub-surface structure
  - Kinematic finite fault model for the seismic source







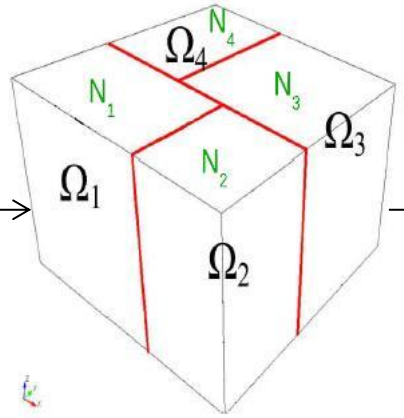
Simulated displacement of the CBD



SPEED Workflow

MPI Master Process

Read data input



Preprocessing phase

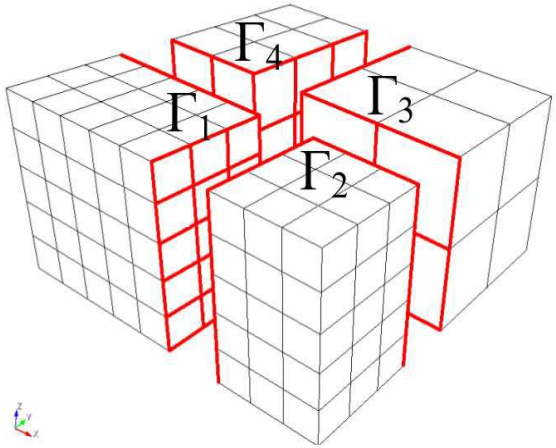
Domain decomposition - metis

Each MPI process receives its own subdomain

Computational and I/O Phase

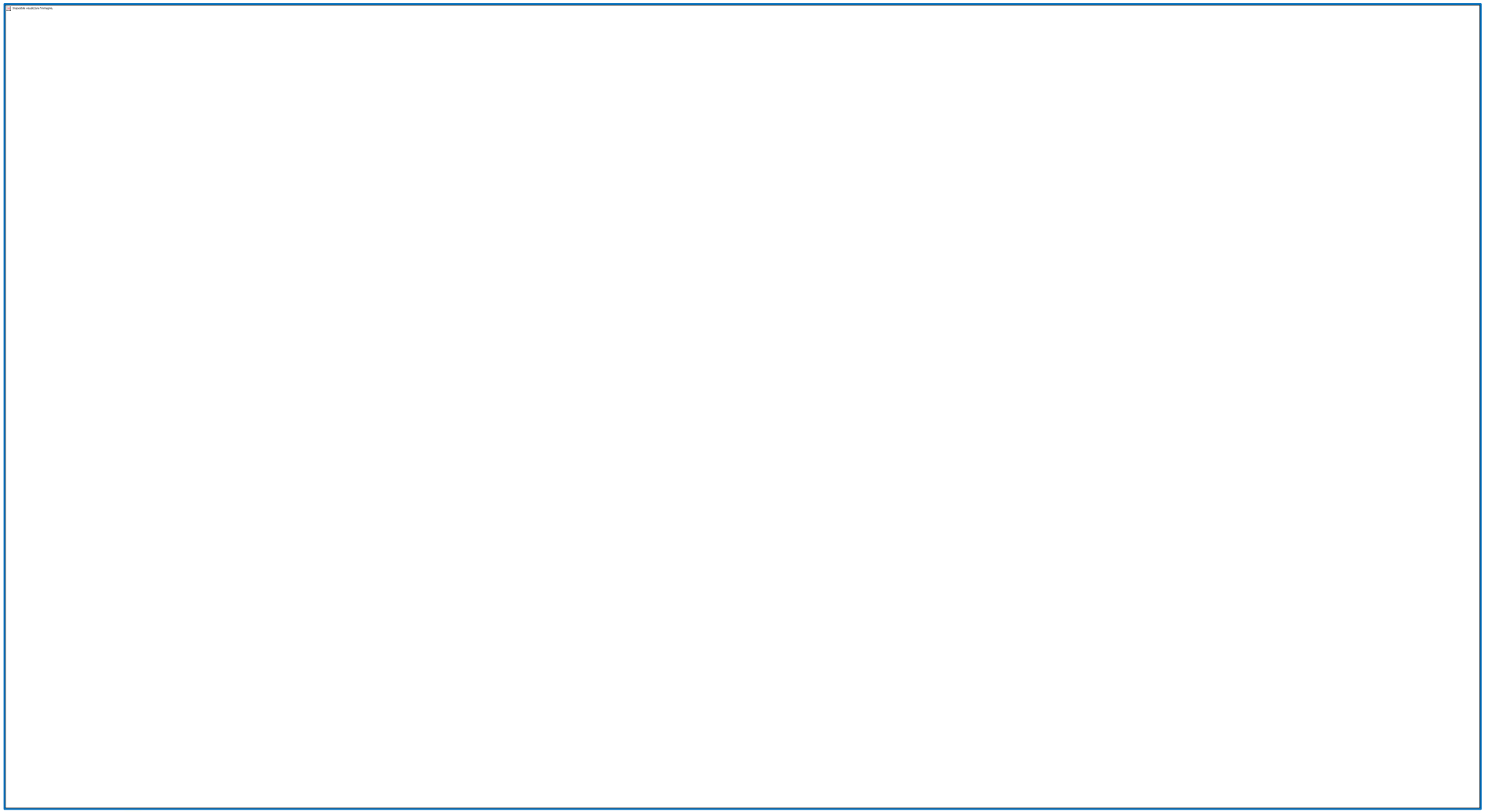
All MPI Processes

Iterative loop



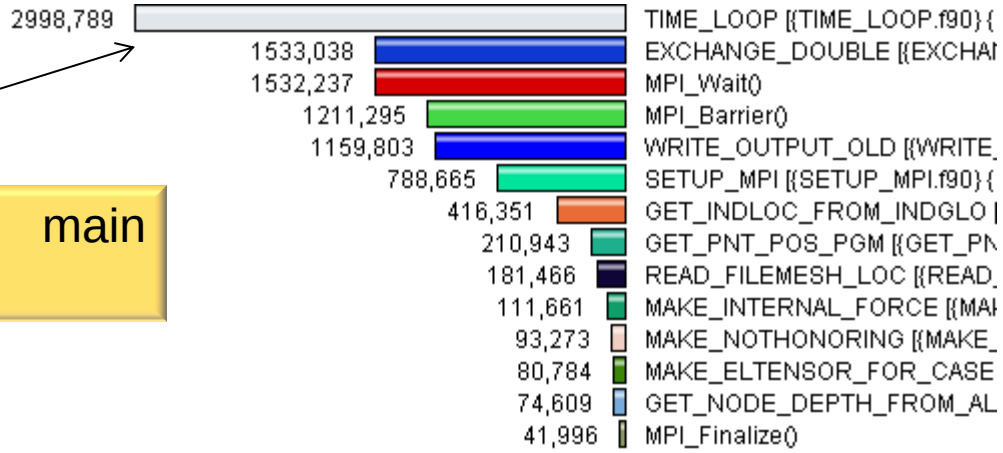
SPEED Optimization Procedure

SPEED Original version - Profiling

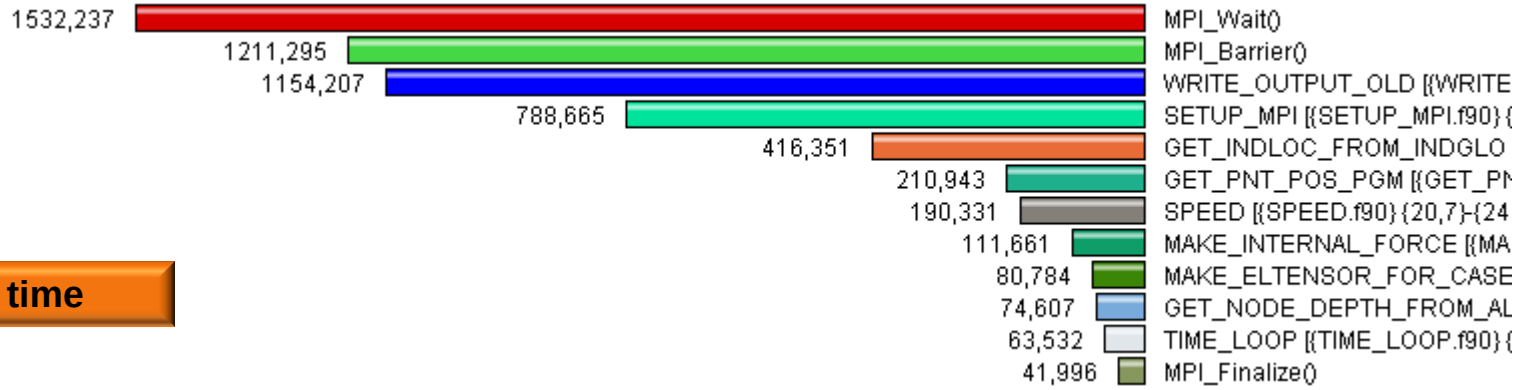


SPEED Original version
Main routines and bottlenecks

Inclusive time



The routine TIME_LOOP is the main routine of the computational phase



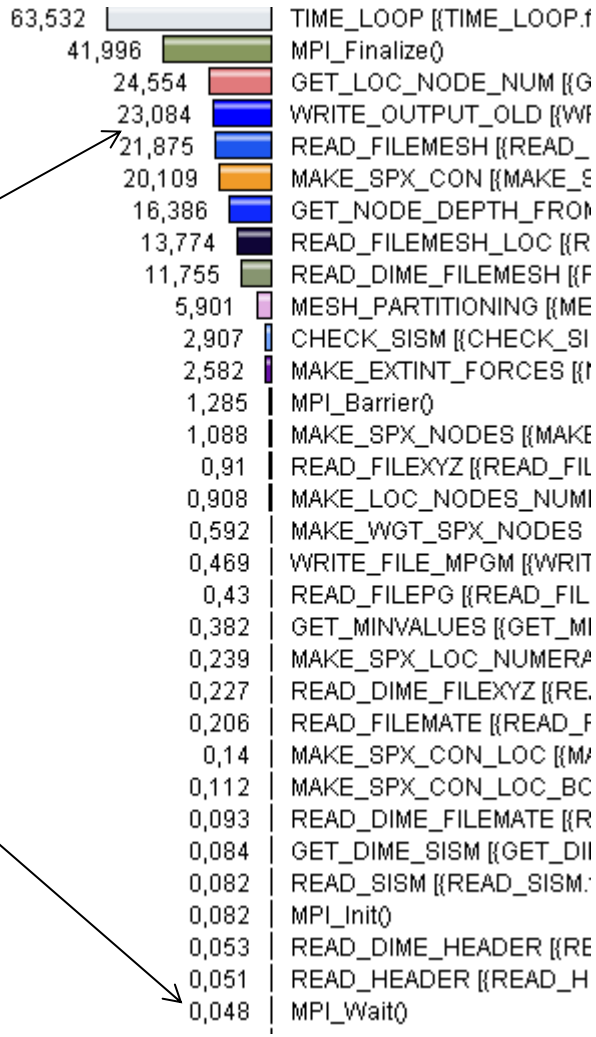
Exclusive time

SPEED Original version - Bottlenecks

Exclusive time per call

The time per call spent in the I/O routine WRITE_OUTPUT_OLD is a bottleneck

The time per call spent in MPI_WAIT each iterartion is very small, it's not a bottleneck for scalability



SPEED Original version – MPI bug

- In the original version this piece of code causes the hang of simulation for some configurations of cores and MPI processes

The «tag» variable is not initialized and takes random and sometimes negative values

```
do ip = 1, nproc
      if (proc_recv(ip).gt.0) then
        call MPI_IRECV(buff_recv(ir:(ir +proc_recv(ip) -1)),&
            proc_recv(ip),SPEED_DOUBLE,(ip -1),&
            tag,comm,request(ip),ierr)
      endif
      ir = ir +proc_recv(ip)
enddo

    is = 1
do ip = 1, nproc
      if (proc_send(ip).gt.0) then
        call MPI_SEND(buff_send(is:(is +proc_send(ip) -1)),&
            proc_send(ip),SPEED_DOUBLE,(ip -1),&
            tag,comm,ierr)
      endif
      is = is +proc_send(ip)
 enddo

 do ip = 1, nproc
      if (proc_recv(ip).gt.0) then
        call MPI_WAIT(request(ip),status,ierr)
      endif
enddo
```

MPI bug solution

- In the new version all MPI calls are non blocking and in the tag variable is now initialized

The «tag» variable has been replaced by MPI_ANY_TAG

```
do ip = 1, nproc
      if (proc_send(ip).gt.0) then
        call MPI_ISEND(buff_send(is:(is +proc_send(ip) -1)),&
            proc_send(ip),SPEED_DOUBLE,(ip -1),&
           tag,comm,requests(ip),ierr)
      endif
      is = is +proc_send(ip)
enddo

    ir = 1
do ip = 1, nproc
      if (proc_recv(ip).gt.0) then
        call MPI_IRECV(buff_recv(ir:(ir +proc_recv(ip) -1)),&
            proc_recv(ip),SPEED_DOUBLE,(ip -1),&
           MPI_ANY_TAG,comm,request(ip),ierr)
      endif
      ir = ir +proc_recv(ip)
 enddo
 do ip = 1, nproc
      if (proc_send(ip).gt.0) then
        call MPI_WAIT(requests(ip),status,ierr)
      endif
 enddo
do ip = 1, nproc
     if (proc_recv(ip).gt.0) then
       call MPI_WAIT(request(ip),status,ierr)
     endif
enddo
```

SPEED Original version – Time loop routine

Main routine for the computing phase

Block repeated 8 times for different tasks

```
do im = 1,nm
        nn = sdeg_mat(im) +1

        allocate(……..)

            ….          ….          ….

        do ie = 1, ne_loc

            if (cs_loc(cs_loc(ie -1) +0) .eq. tag_mat(im)) then

            ….          ….          ….


        enddo
enddo
```

Loop on materials

Loop on subdomains elements

Check that the element has the correct material

SPEED Optimized version
Time loop routine

```
do im = 1,nm
      allocate(……..)
            ….        ….        ….
      do ie = 1, ne_loc
        if (cs_loc(cs_loc(ie -1) +0) .eq. tag_mat(im)) then
            ….        ….        ….
      enddo
enddo
```

```
do ie = 1, ne_loc
      allocate(……..)
            ….        ….        ….
enddo
```

- Looking carefully at the logic of the algorithm we find out that the ordered loop over materials and the conditional clause to check if an element belongs to the group of elements within that material are not necessary.
- We can just loop over all the elements without caring about which material the element belongs to.

SPEED Hybrid Optimized version
Time loop routine

```
!$OMP PARALLEL &
!$OMP PRIVATE(ie,im,nn,ct,ww,dd,ux_el,uy_el, ….) &
!$OMP PRIVATE(duzdy_el,duxdz_el,duydz_el,…..) &
!$OMP PRIVATE(rho_el,lambda_el,mu_el,…..) &
!$OMP PRIVATE(k,j,i,is,in,iaz)

!$OMP DO
        do ie = 1, ne_loc

           if (cs_loc(cs_loc(ie -1) +0) .eq. tag_mat(im)) then

              ….        ….        ….

        enddo
enddo
```

Creation of parallel region

Check for shared and private variables

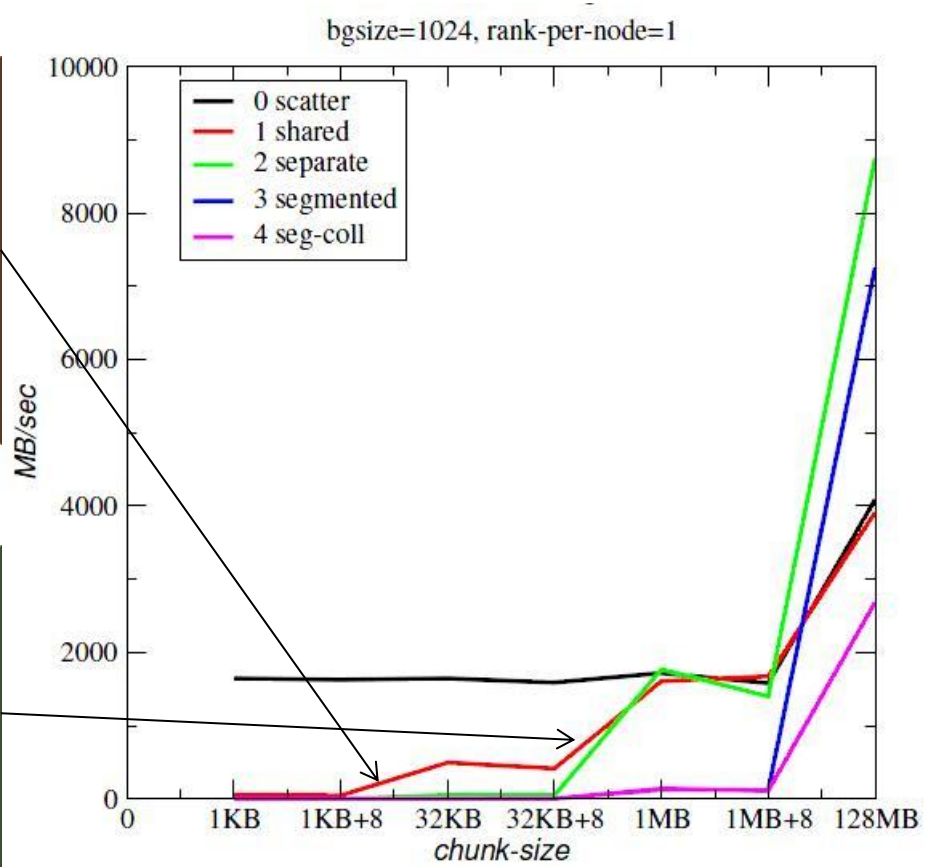
Loop parallelization

- The hybrid optimized version **solves the memory issue** allowing to chose the best configuration between MPI processes and OpenMP threads.
- **No more cores and resource wasting**, now all the computing power of IBM PowerA2 processors can be used.

SPEED Hybrid Optimized version I/O optimization

- **Original MPI version**
  - Each MPI process writes multiple files
  - Only one row is written in each file
  - Thousands of files to be managed in parallel by the file system for large simulations

- **Hybrid Otimized version**
  - Each MPI process writes only one file
  - Multiple rows are written
  - Many few files to be managed in parallel by the file system



SPEED Hybrid Optimized version

- **Using 4 OpenMP threads and the optimized I/O routine we ha a 5x speedup**

Inclusive time

639,314	TIME_LOOP [{TIME_LOOP.pomp.f90}{21,8}-{3458,29}]
367,465	MPI_Barrier()
309,877	parallel fork/join [OpenMP]
297,049	MAKE_NOTHONORING [{MAKE_NOTHONORING.pomp.f90}{2
290,352	GET_NODE_DEPTH_FROM_ALLUVIAL [{GET_NODE_DEPTH_
263,177	parallel [OpenMP location: file:/gpfs/scratch/userinternal/pdagna
251,061	EXCHANGE_DOUBLE [{EXCHANGE_DOUBLE.pomp.f90}{21,6
250,154	MPI_Wait()
220,855	GET_PNT_POS_PGM [{GET_PNT_POS_PGM.pomp.f90}{21,7}-
201,129	parallel begin/end [OpenMP]
200,884	for enter/exit [OpenMP]
190,612	do [OpenMP location: file:/gpfs/scratch/userinternal/pdagna00/P
157,878	READ_FILEMESH_LOC [{READ_FILEMESH_LOC.pomp.f90}{2
74,858	MAKE_LGL_NW [{MAKE_LGL_NW.pomp.f90}{21,6}-{109,32}]
70,274	MAKE_SPX_CON_LOC [{MAKE_SPX_CON_LOC.pomp.f90}{21
59,735	MAKE_SPX_CON_LOC_BOUND [{MAKE_SPX_CON_LOC_BOU
50,095	MAKE_INTERNAL_FORCE [{MAKE_INTERNAL_FORCE.pomp.f
39,343	READ_SISM [{READ_SISM.pomp.f90}{21,7}-{125,30}]
38,954	GET_DIME_SISM [{GET_DIME_SISM.pomp.f90}{21,7}-{114,34}]

Exclusive time

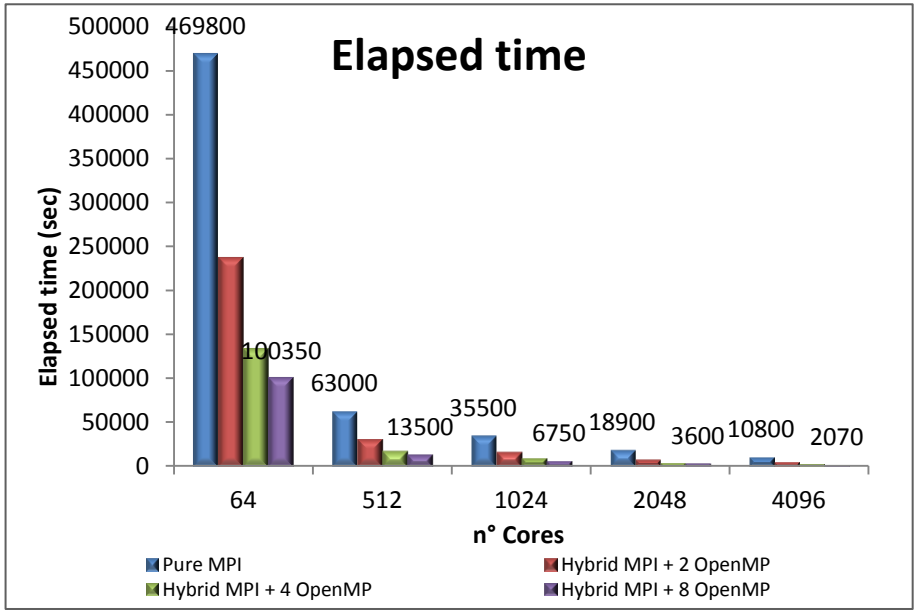
250,154	MPI_Wait()
220,855	GET_PNT_POS_PGM [{GET_PNT_POS_PGM.pom
157,878	READ_FILEMESH_LOC [{READ_FILEMESH_LOC.p
72,468	parallel [OpenMP location: file:/gpfs/scratch/userint
70,274	MAKE_SPX_CON_LOC [{MAKE_SPX_CON_LOC.p
59,59	MAKE_LGL_NW [{MAKE_LGL_NW.pomp.f90}{21,6
58,744	MAKE_SPX_CON_LOC_BOUND [{MAKE_SPX_CO
50,095	MAKE_INTERNAL_FORCE [{MAKE_INTERNAL_FO
39,343	READ_SISM [{READ_SISM.pomp.f90}{21,7}-{125,3
38,954	GET_DIME_SISM [{GET_DIME_SISM.pomp.f90}{21
38,597	parallel do [OpenMP location: file:/gpfs/scratch/user
28,897	GET_LOC_NODE_NUM [{GET_LOC_NODE_NUM.
28,29	MAKE_ELTENSOR_FOR_CASES [{MAKE_ELTENS
25,177	FIND [{FIND.pomp.f90}{21,7}-{51,23}]
23,37	GET_NEAREST_NODE_PGM [{GET_NEAREST_N
22,295	TIME_LOOP [{TIME_LOOP.pomp.f90}{21,8}-{3458,
21,67	do [OpenMP location: file:/gpfs/scratch/userinternal
21,112	READ_FILEMESH [{READ_FILEMESH.pomp.f90}{
20,112	MAKE_SPX_CON [{MAKE_SPX_CON.pomp.f90}{2
18,079	MAKE_SPX_LOC_NUMERATION [{MAKE_SPX_LO
14,382	FIND_ROOT_POL [{FIND_ROOT_POL.pomp.f90}{
12,89	MAKE_STRAIN_TENSOR [{MAKE_STRAIN_TENSO
12,049	READ_DIME_FILEMESH [{READ_DIME_FILEMESH
6,396	parallel [OpenMP location: file:/gpfs/scratch/userint
6,115	MESH_PARTITIONING [{MESH_PARTITIONING.po
4,509	MAKE_STRESS_TENSOR [{MAKE_STRESS_TENS
4,485	GET_NODE_DEPTH_FROM_CMPLX [{GET_NODE
4,276	WRITE_OUTPUT [{WRITE_OUTPUT.pomp.f90}{20

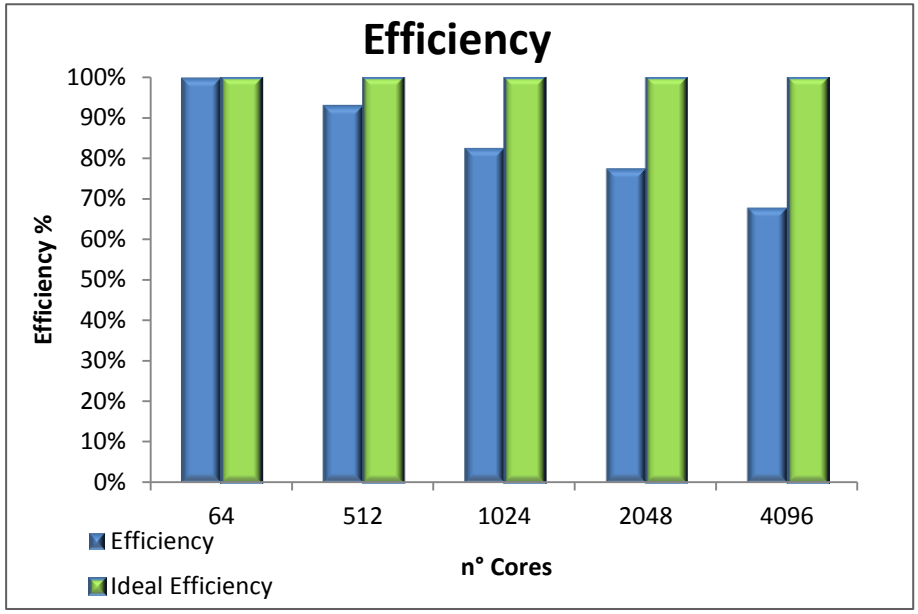
I/O is 6-10 time faster

SPEED Hybrid New version vs Original MPI

Conforming mesh

- **Number of unused cores in the original MPI version : 8 per node**
- **In the new Hybrid version it's possible to have up to 64 processes per node**
  - **Full computing power usage**
  - **Up to six times faster using the same bg_size**

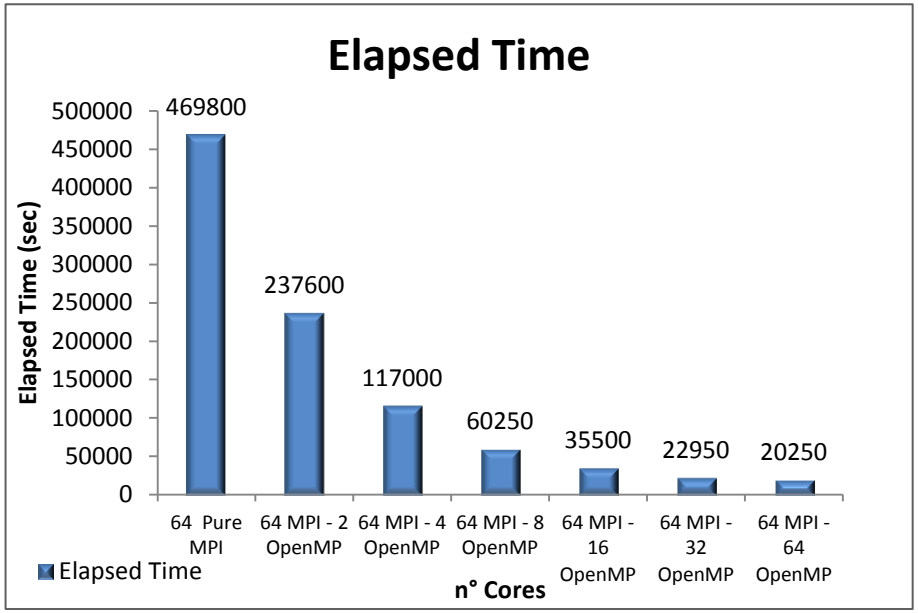


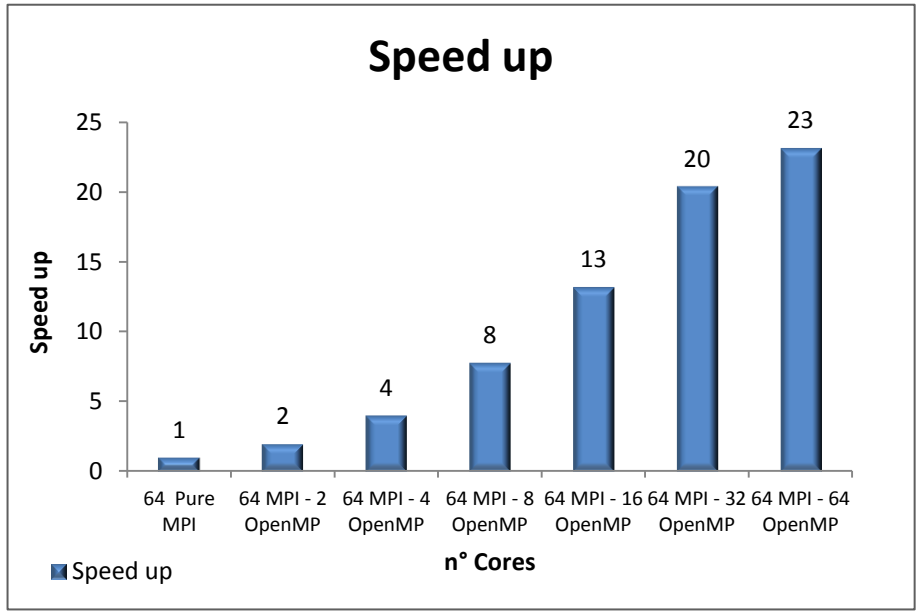


SPEED Hybrid New version vs Original MPI

Conforming mesh

- **Can we go faster?**
  - **Selecting the optimal value of MPI processes it's possible to go up to 23 times faster using also OpenMP threads**

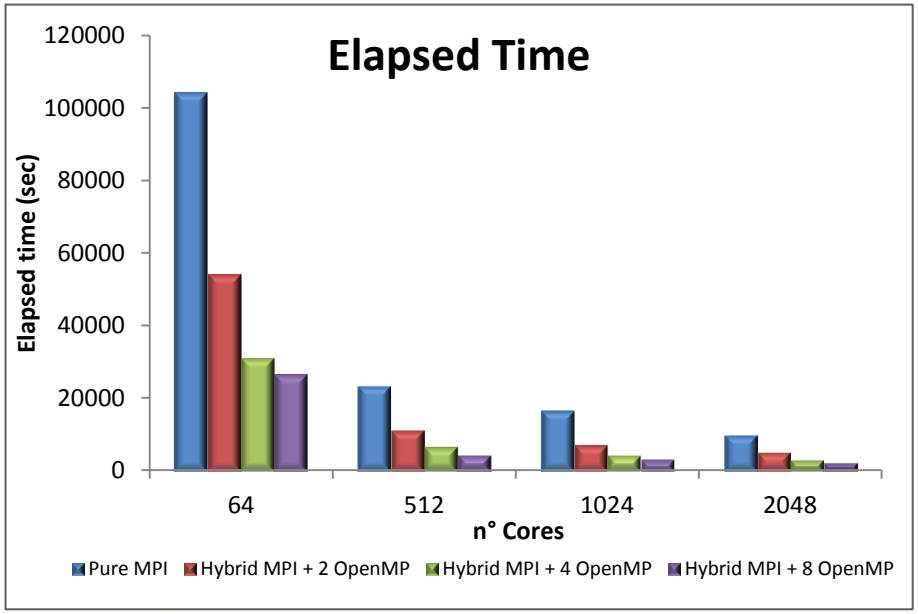


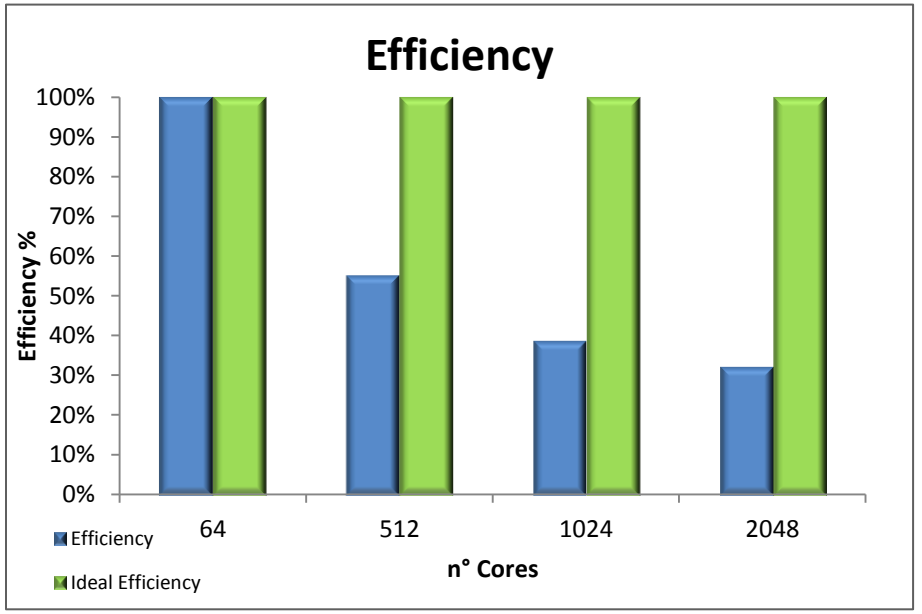


SPEED Hybrid New version vs Original MPI

Non Conforming mesh

- **Number of unused cores in the original MPI version : 8 per node**
- **Efficiency decreases faster**
- **In the new Hybrid version it's possible to have up to 64 processes per node**
  - **Full computing power usage**
  - **Up to six times faster using the same bg_size**

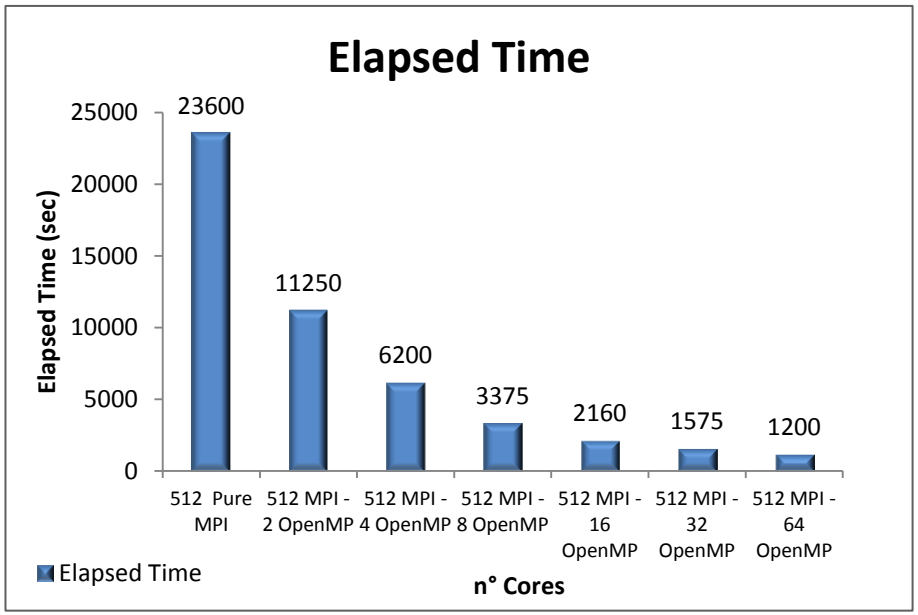


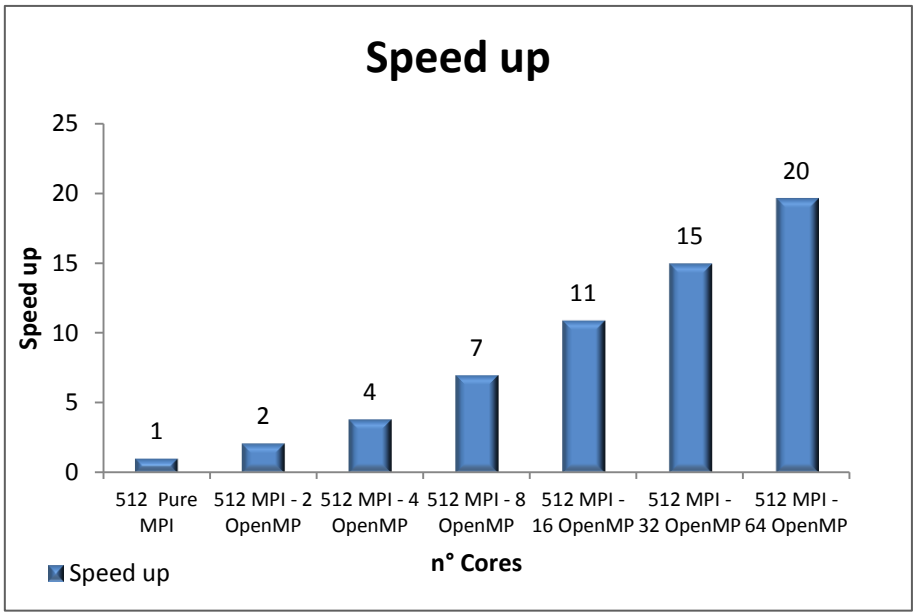


SPEED Hybrid New version vs Original MPI

Non Conforming mesh

- **Can we go faster?**
  - **Selecting the optimal value of MPI processes it's possible to go up to 23 times faster using also OpenMP threads**





SPEED Future work

- Further optimizations
  - Cache blocking techniques in the computational routines
  - Preprocessing phase optimization

TIGRA Optimization BGQ Exploitation

Outline

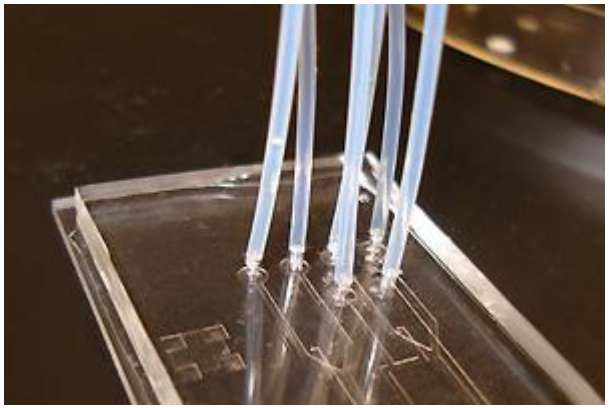
- **TIGRA**
  - Introduction
  - Optimization strategy
  - Benchmark
  - Future work

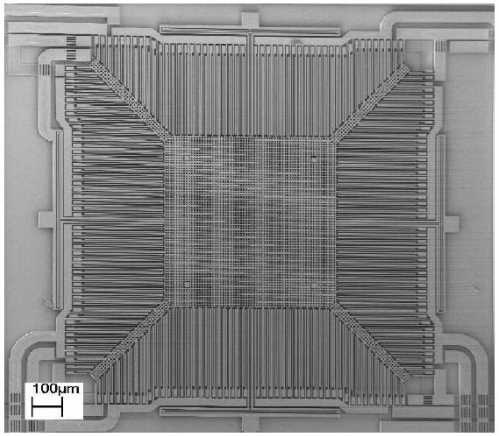
Introduction

- TIGRA
  - open-source numerical code for the simulation of turbulence and instability in rarefied gas flows.
  - Fortran
  - BLAS for linear algebra operations
  - MPI communication library
  - OpenMP directives (inter-node)
- designed with the aim of giving accurate solutions for low Mach number gas flows, such as those occurring in micro-electro-mechanical systems (MEMS) where the DSMC method becomes computationally inefficient due to the large statistical noise.

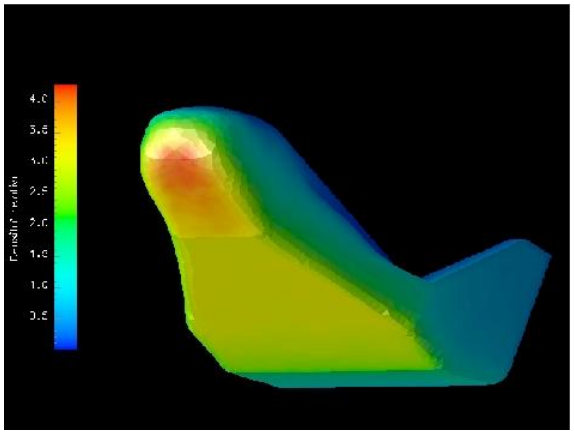
TIGRA – Application Fields

- Microfluidics devices
  - Micropomps
  - Microvalves





- Aerothermodynamics
  - Space shields
  - Improvement of the aerodynamics of a spacecraft in the re-entry phase

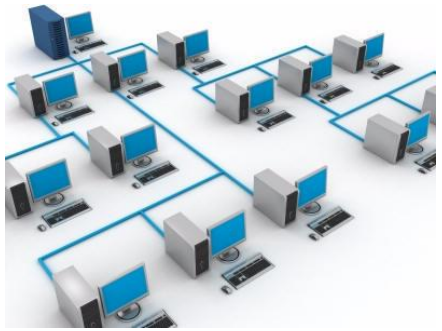


TIGRA – Architectures

- Developed with the aim of using all the available hpc architectures
  - Hybrid MPI – OpenMP
  - Hybrid MPI - CUDA

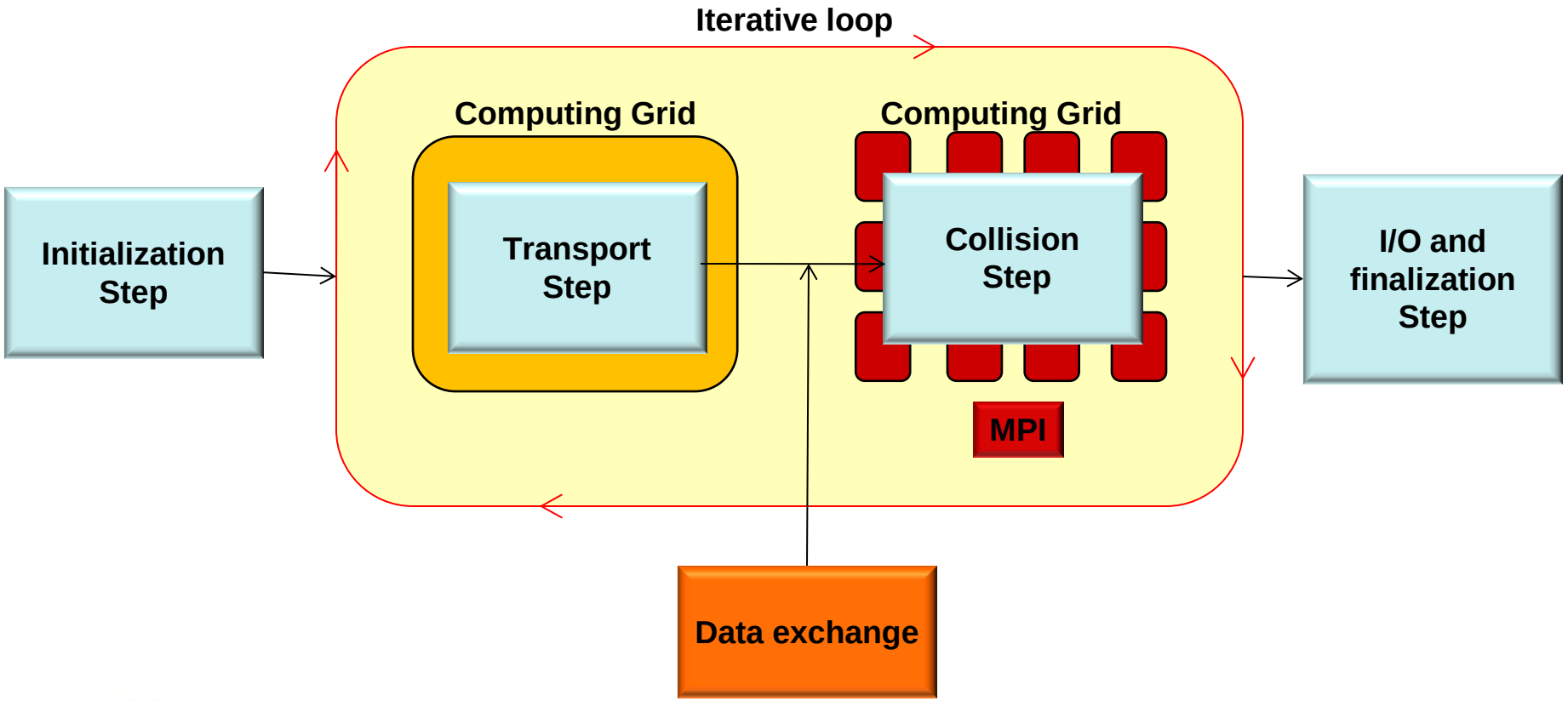
Fermi

PLX



LAN of PC

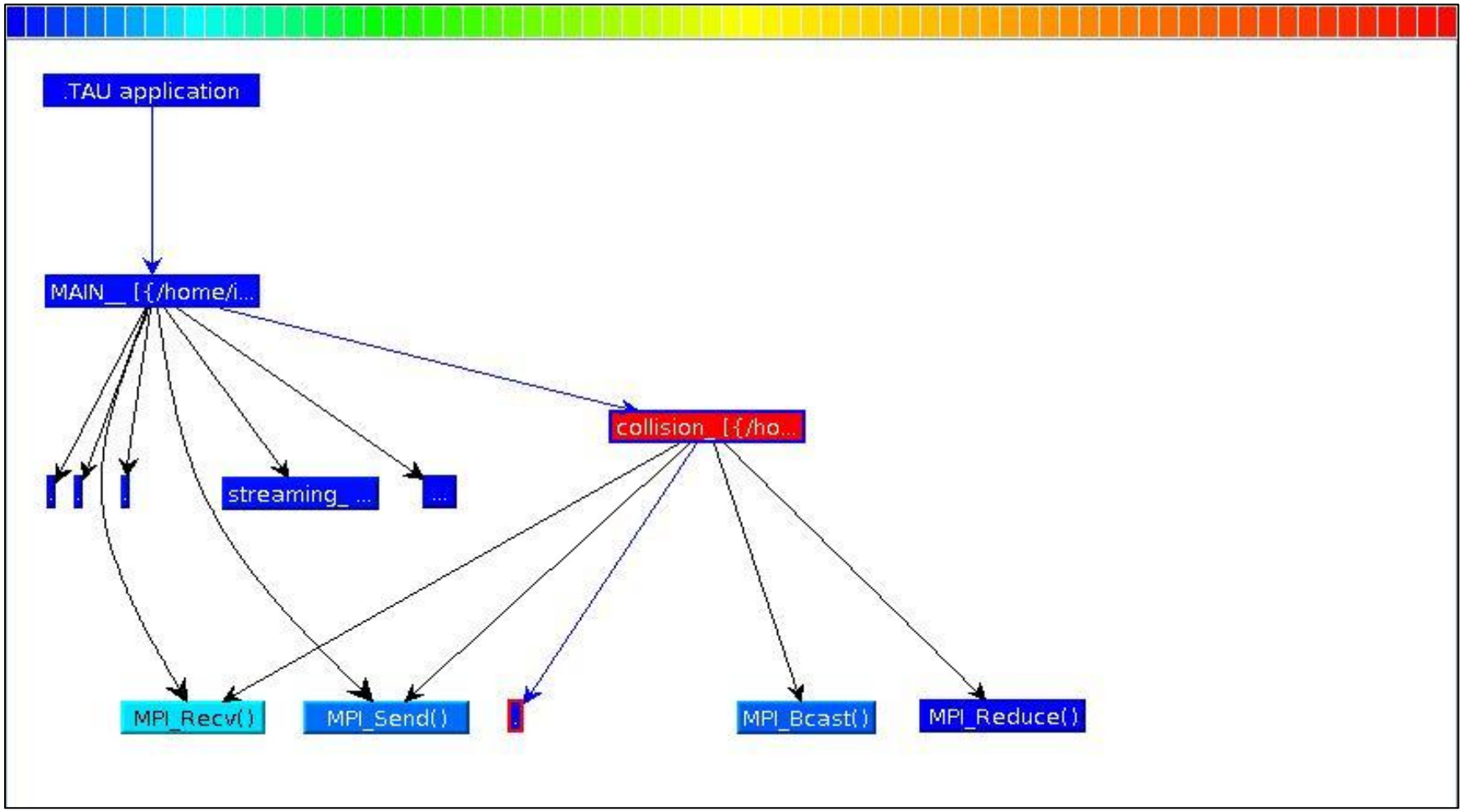
TIGRA Workflow – MPI Version



TIGRA Domain of simulation

- Computational parameters
  - *NX;NY*, number of physical cells in the *x* and *y* directions
  - 3D space of velocities
  - Matrices C containing interaction between particles of dimension N3xN6 where N is tipically in the range 5-10.
- In the collision step for each physical cell we have to loop over the C matrices in subblocks of dimension N3xN3 and compute a matrix-vector multiplication of order (N3,N3)xN3
- Simulation time is proportional to N,NX and NY.
- For each physical cell the number of operations is

TIGRA Original version - Profiling

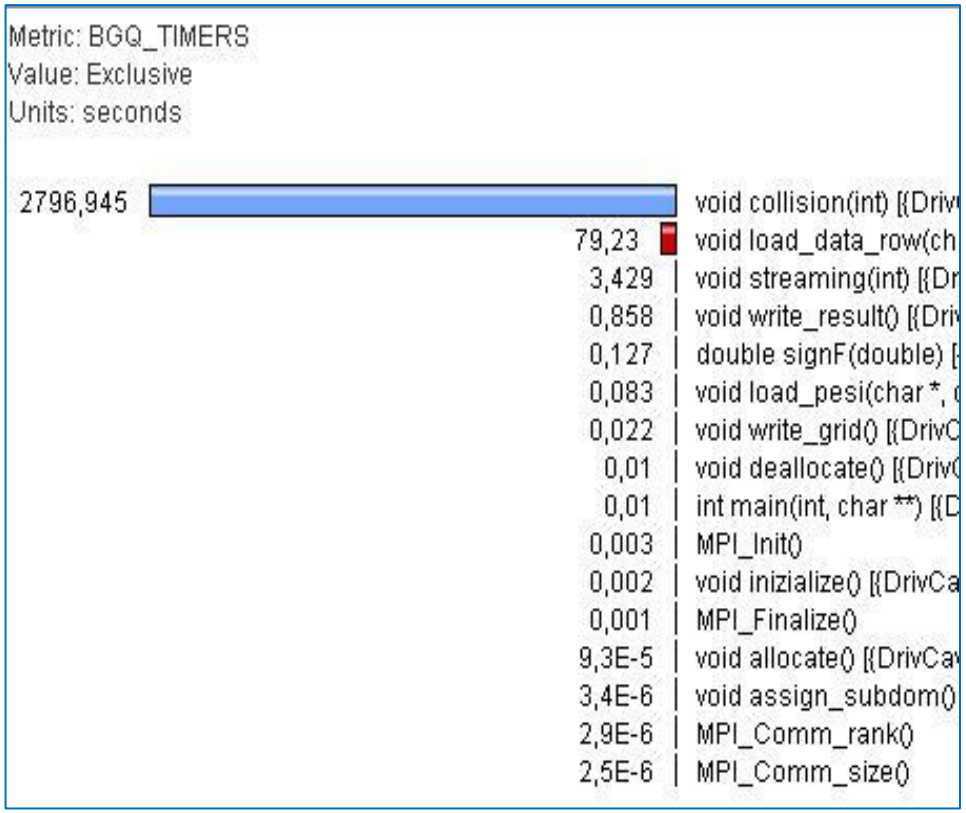


TIGRA Original version - Profiling

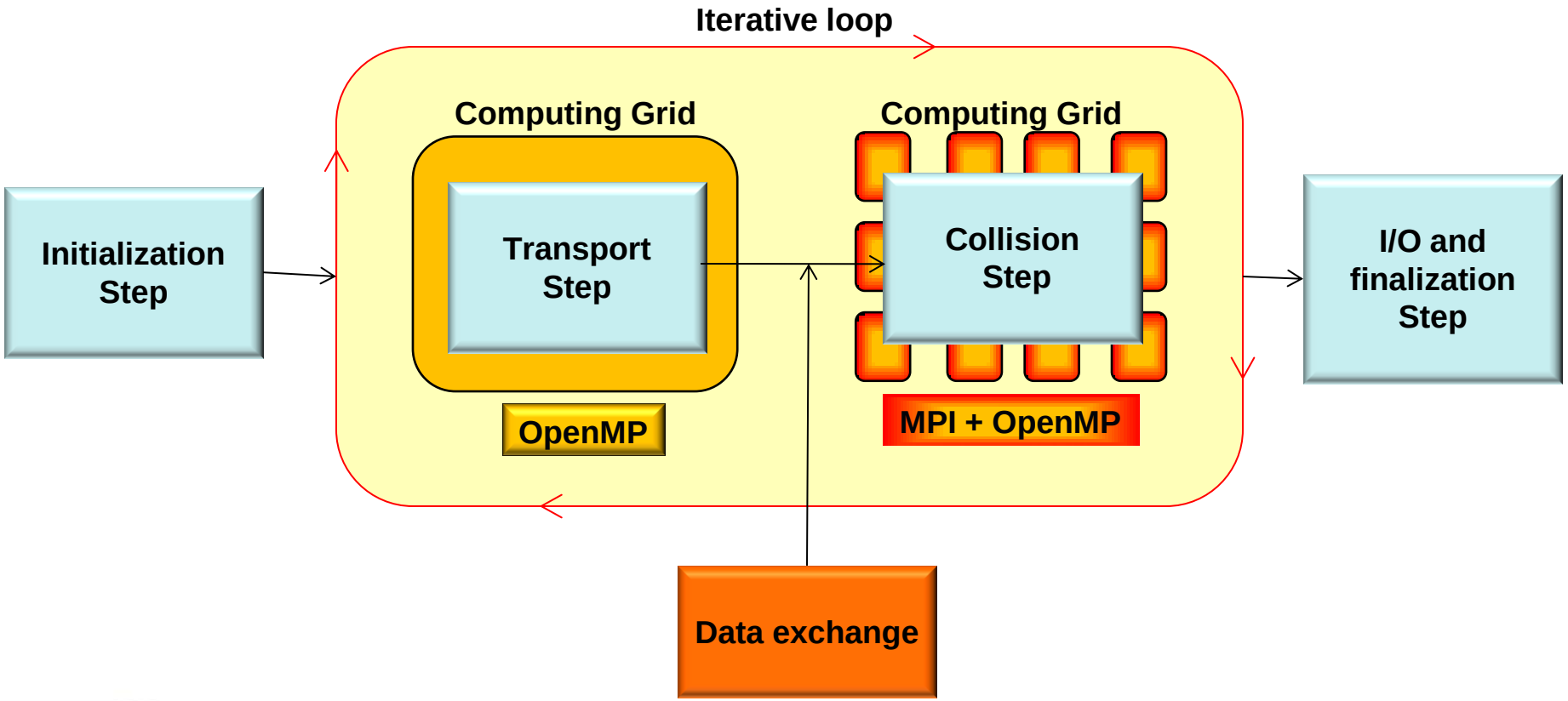
Name △	Exclusive TIME	Inclusive TIME	Calls	Child Calls
.TAU application	0.026	237.489	1	1
MAIN__ [{/home/interni/dagna/bando_lisa/lisa043/DriCavBHS/test_8nodi_2mpixnodc	0.026	237.463	1	38
MPI_Comm_rank()	0	0	1	0
MPI_Comm_size()	0	0	1	0
MPI_Finalize()	0.717	0.717	1	0
MPI_Init()	1.458	1.458	1	0
MPI_Recv()	16.821	16.821	16	0
collision_ [{/home/interni/dagna/bando_lisa/lisa043/DriCavBHS/test_8nodi_2mpi	183.872	217.187	9	135
MPI_Allreduce()	0.03	0.03	36	0
MPI_Bcast()	33.271	33.271	36	0
MPI_Reduce()	0.006	0.006	27	0
MPI_Send()	0.007	0.007	36	0
streaming_ [{/home/interni/dagna/bando_lisa/lisa043/DriCavBHS/test_8nodi_2m	1.253	1.253	9	0

TIGRA Original version - Profiling

- The computing time ratio between collision and streaming step is of the order of
- Number of operations in the collision step)
- Number of operations in the streaming step)



TIGRA Workflow – Hybrid MPI - OpenMP



TIGRA MPI decomposition

- Collision step
  - Matrices are decomposed using MPI along the N6 rows dimension but they can't be smaller than N3xN3
  - C matrices are independent in the physical space

```
for(i=0; i<NX;i++){
    for(j=0; j<NY;j++){
      for(k=0;k<N3;k++) {
        initialization of vectors G(4N3)

      }
      for(k=0;k<N3;k++){
        ini=k*N3;
        fin=ini+N3-1;
        for(l=0;l<N3;l++){
          for(m=0;m<N3;m++){

                initialization of matrix C(N3,N3)
          }
        }
        dgemv (C,G)
        }
      }
}
```

MPI decomposition

Initialization of subblock of C matrix (N3,N3)

TIGRA MPI decomposition

Collision step

Each MPI process takes a subset on the N3 blocks of dimension (N3,N3)

gXX(NX,NY,N3)

Necessary operation because during the streaming phase the MPI processes need the entire gXX arrays

```
for(i=0; i<NX;i++){
    for(j=0; j<NY;j++){
      for(k=0;k<N3;k++) {
        initialization of vectors G(4N3)

      }
      for(k=0;k<sub_N3;k++){        <- MPI decomposition
        ini=k*N3;
        fin=ini+N3-1;
        for(l=0;l<N3;l++){
          for(m=0;m<N3;m++){

                initialization of matrix C(N3,N3)
          }
        }
        dgemv (C,G,res)
        gXX[j+i*NY+k*NY*NX]+= dt * ddot(G,res);
        }
    }
}
```

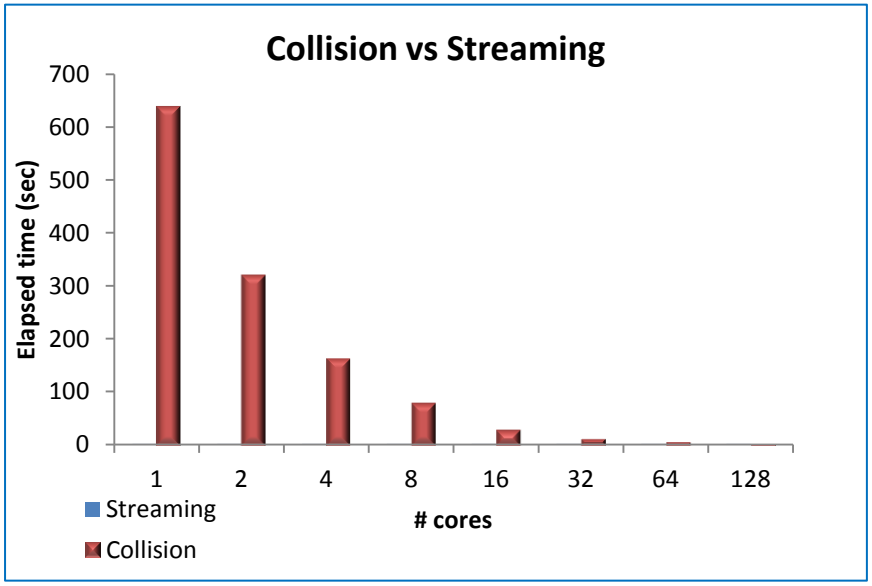
Master process receives all the contributions from the slaves processes to build the entire array gXX and then broadcasts the array to all.

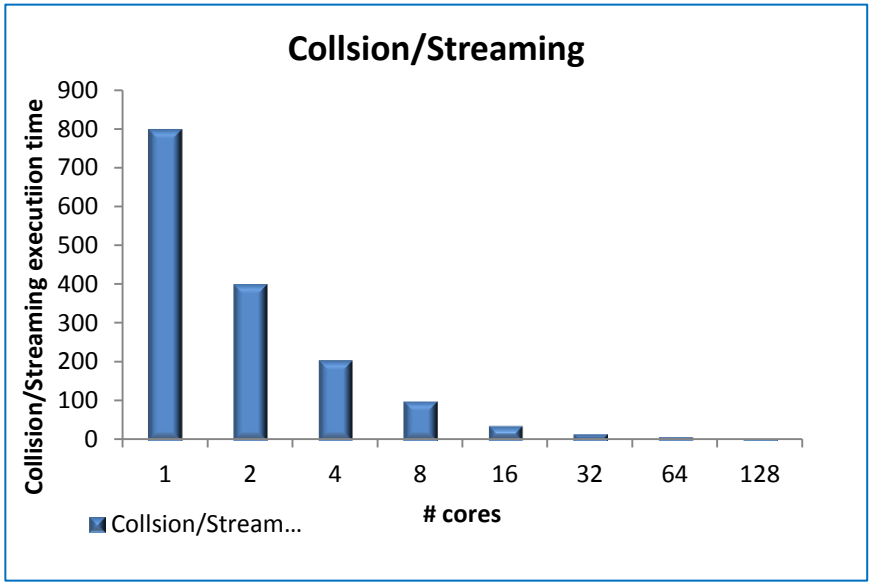
```
MPI_Bcast (gXX, NX*NY*N3, MPI_DOUBLE , 0, MPI_COMM_WORLD );
```

TIGRA MPI version

- When the number of MPI processes increase the execution time of the routines collision and streaming becomes comparable.

n° Cores	Elapsed Time Streaming (sec)	Elapsed Time Collision (sec)	Collsion/Streaming
1	0,8	640	800
2	0,8	322	403
4	0,8	165	206
8	0,8	81	101
16	0,8	30	38
32	0,8	13	16
64	0,8	7	9
128	0,8	3,5	4





TIGRA Hybrid version MPI +OpenMP

Collision step

Indipendent along i,j

gXX(NX,NY,N3)

MPI decomposition

```
#pragma omp parallel default(shared) private(i,j,….)
{  …   …     …   ….
#pragma omp for
for(i=0; i<NX;i++){
     for(j=0; j<NY;j++){
       for(k=0;k<N3;k++) {
         initialization of vectors G(4N3)
        }
       for(k=0;k<sub_N3;k++){
          ini=k*N3;
          fin=ini+N3-1;
          for(l=0;l<N3;l++){
            for(m=0;m<N3;m++){

                    initialization of matrix C(N3,N3)
            }
          }
          dgemv (C,G,res)
          gXX[j+i*NY+k*NY*NX]+= dt * ddot(G,res);
          }
       }
 }
} // close parallel region
  …   ….   …   …
Master process receives all the contributions from the slaves processes
to build the entire array gXX and then broadcasts the array to all.
MPI_Bcast (gXX, NX*NY*N3, MPI_DOUBLE , 0, MPI_COMM_WORLD );
```

TIGRA Hybrid version MPI +OpenMP

Streaming step

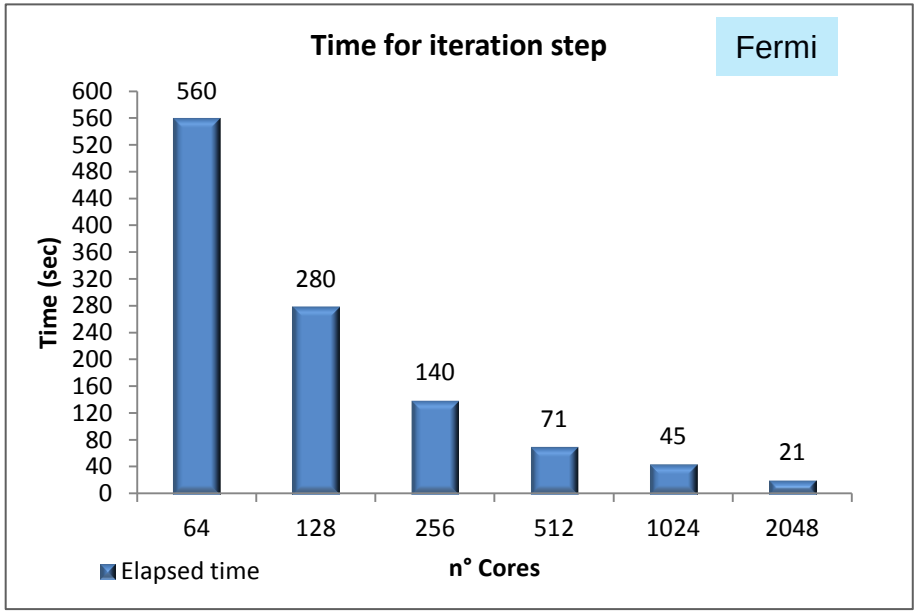
Indipendent along i,j

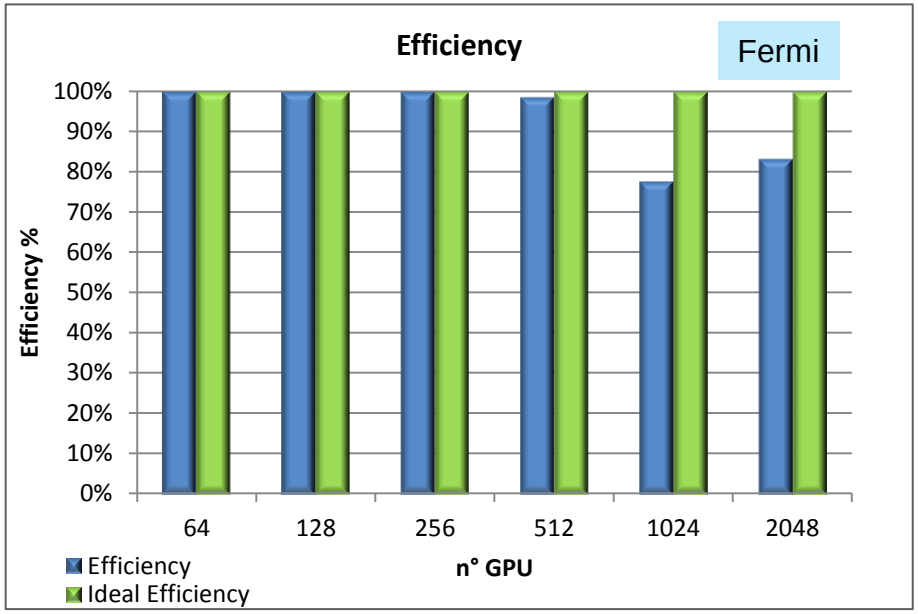
Indipendent along i,j

```
for(k=0;k<N3;k++){
#pragma omp parallel default(shared) private(i,j….)
{
#pragma omp for
  for(j=0;j<NY;j++){
    for( i=0;i<NX-2;i++){

     …   …   …   …

    }
  }

#pragma omp for
  for(j=0;j<NY;j++){
    for( i=0;i<NX-2;i++){
      gPP[j+(i+1)*NY+k*NY*NX]
          …  ….   …..   ….
    }
  }
} // end of parallel region
} // end of k loop
```

TIGRA - Benchmark

2D Grid Dimension : 200 x 200
Number of nodes in the velocity space : 5
Number of floating point operations per iteration step : *O*



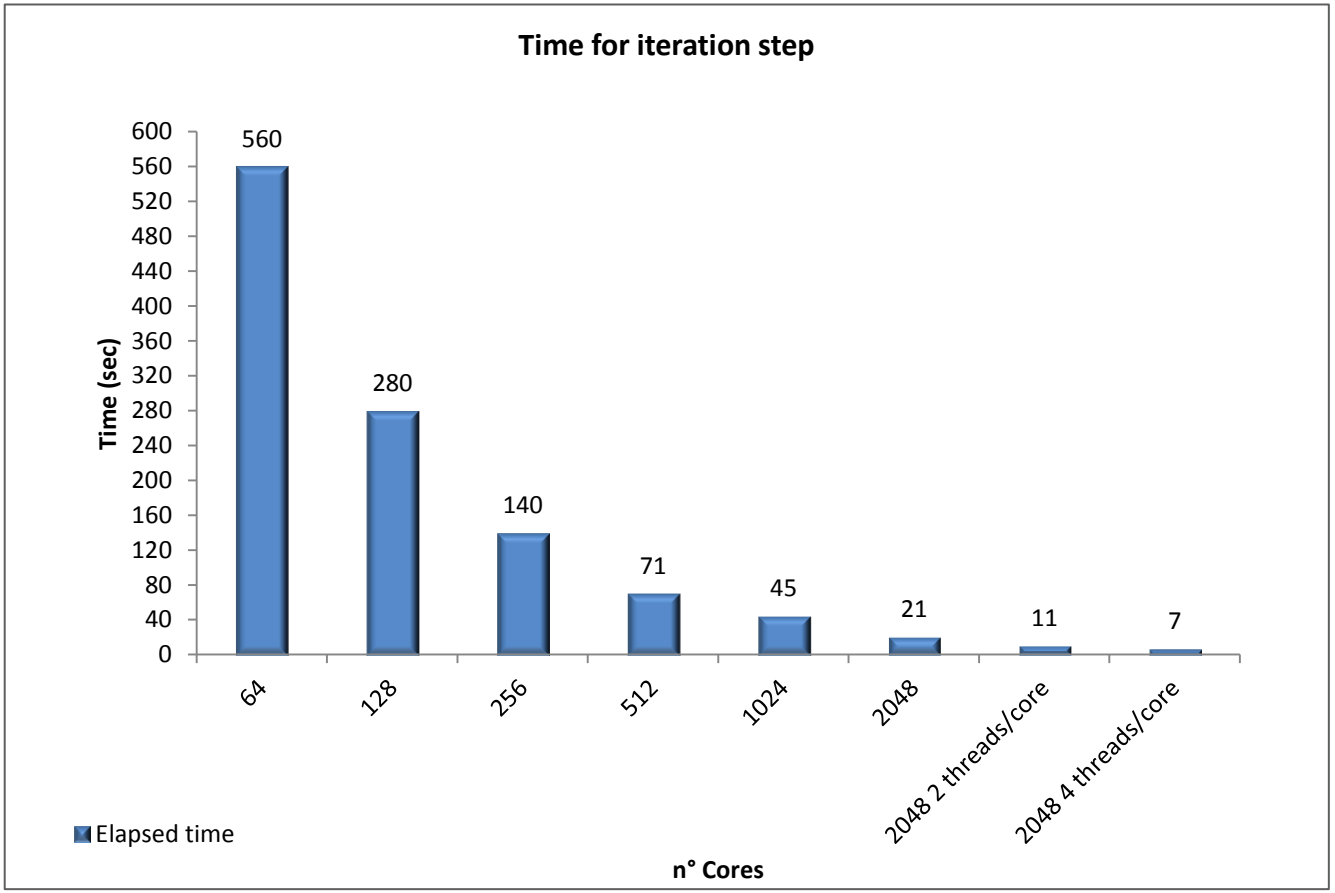


Processors : IBM PowerA2, 1.6 GHz
Node : 16 cores
RAM : 16 GB/Node
Interconnection: Proprietary Torus 5D.

TIGRA - Benchmark

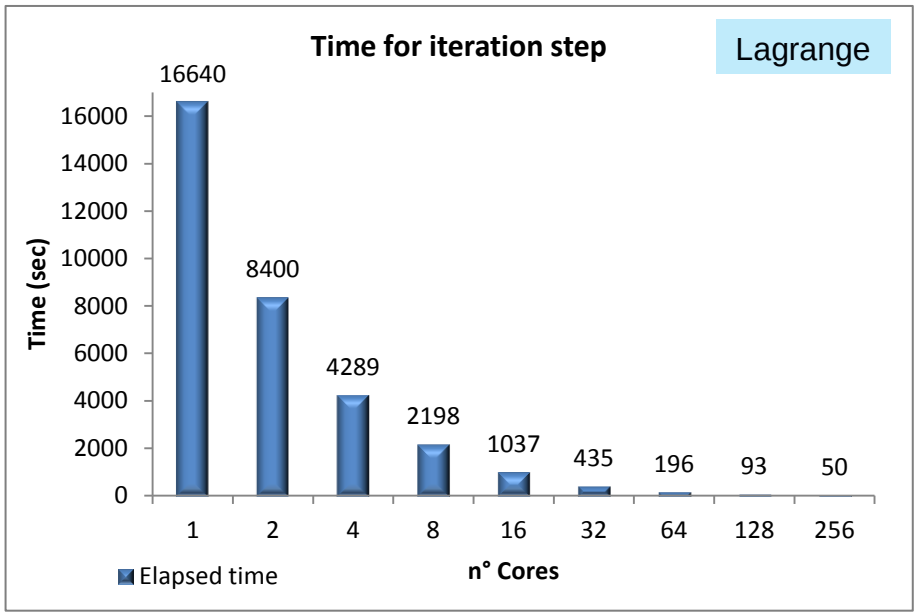
2D Grid Dimension : 200 x 200
Number of nodes in the velocity space : 5
Number of floating point operations per iteration step : *O*

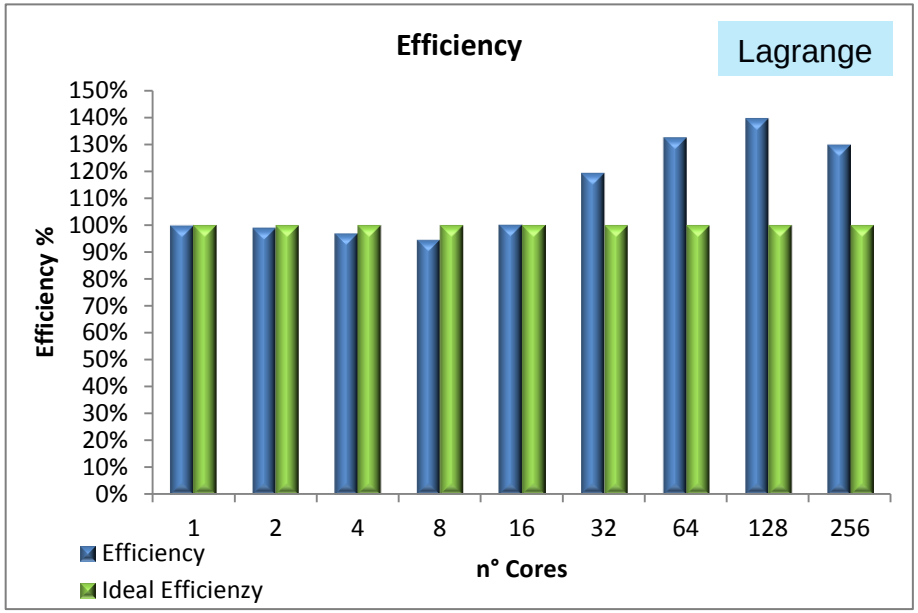
- Using PowerA2 4-way simultaneous multithreaded core capability we can obtain further 3X speedup



TIGRA - Benchmark

2D Grid Dimension : 200 x 200
Number of nodes in the velocity space : 5
Number of floating point operations per iteration step : *O*

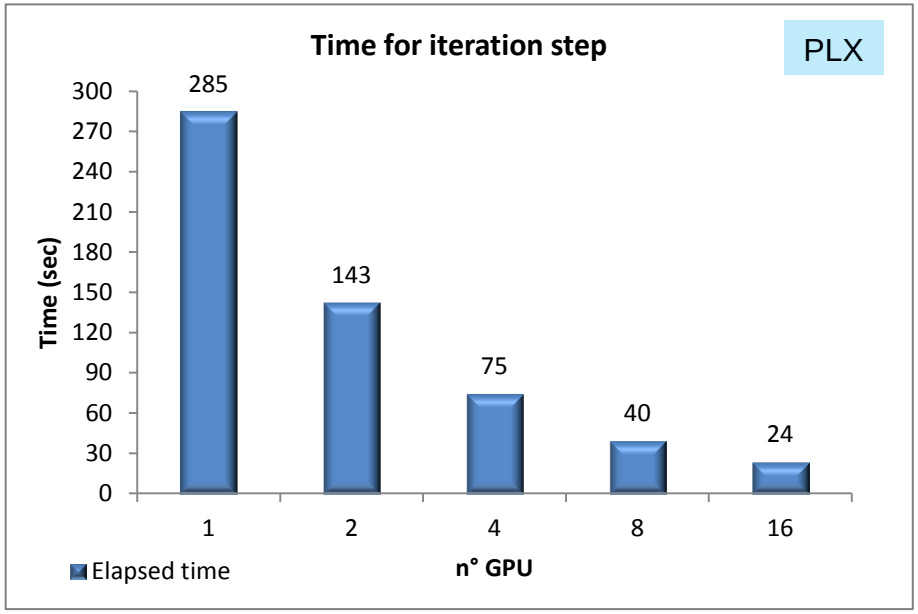


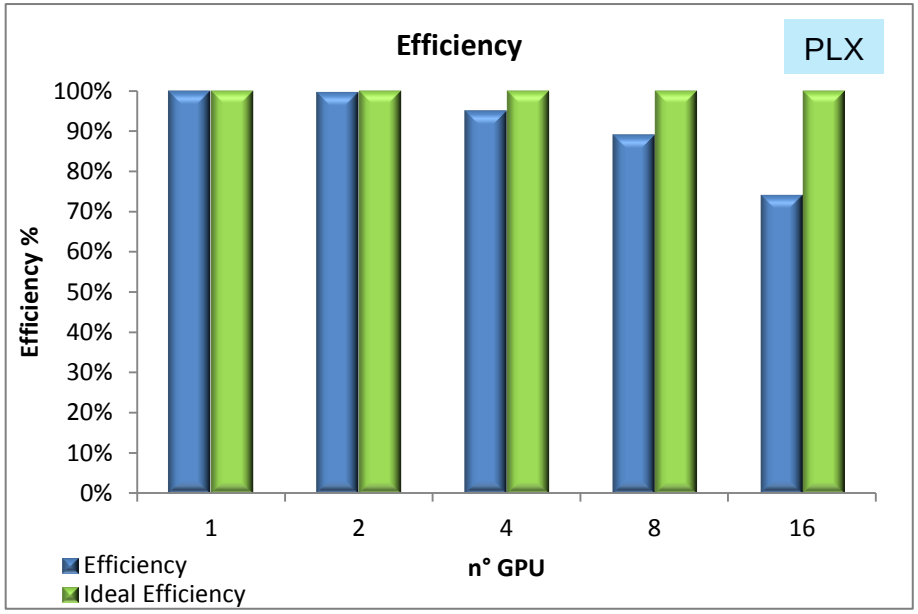


Processors : Intel Xeon X5660 2.80 GHz exacore
Node : 12 cores
RAM : 24 GB/Node
Interconnection : Infiniband QDR 40 Gb/s

TIGRA - Benchmark

2D Grid Dimension : 200 x 200
Number of nodes in the velocity space : 5
Number of floating point operations per iteration step : *O*

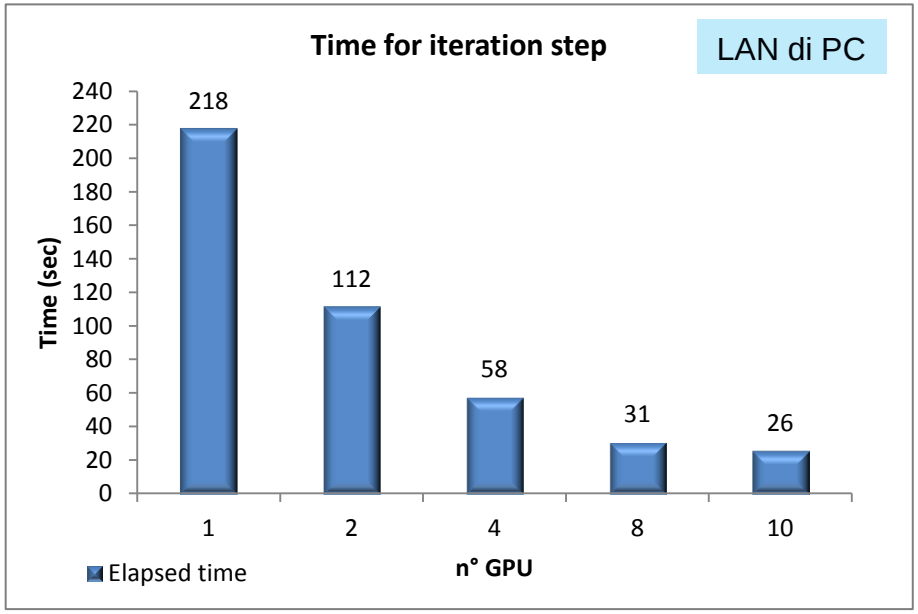


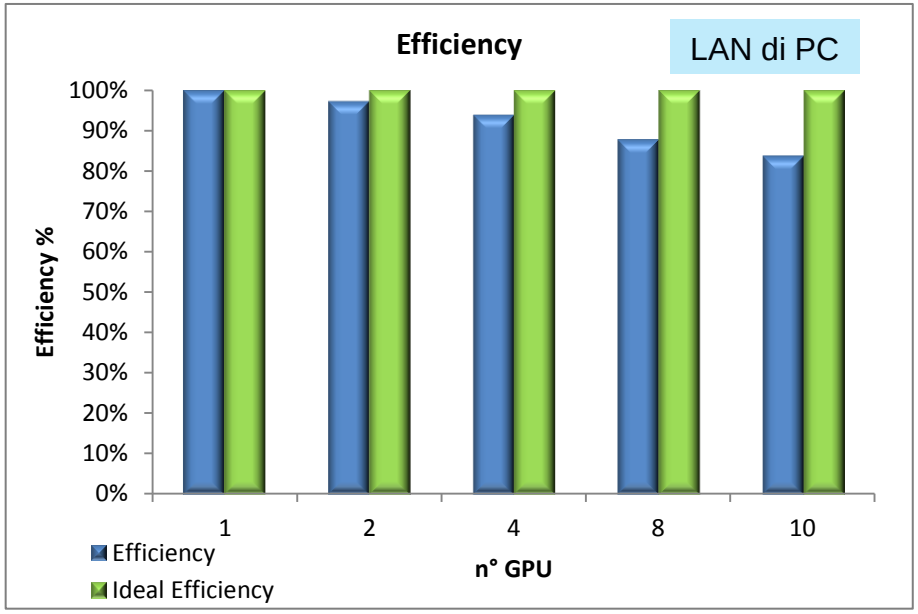


Processors: Intel Xeon X5660 2.40 GHz exacore
Node : 12 cores
RAM : 48 GB/Node
Interconnection : Infiniband QDR 40 Gb/s
GPU : Tesla M2070 1.1 GHz
RAM GPU : 6GB

TIGRA - Benchmark

2D Grid Dimension : 200 x 200
Number of nodes in the velocity space : 5
Number of floating point operations per iteration step : *O*

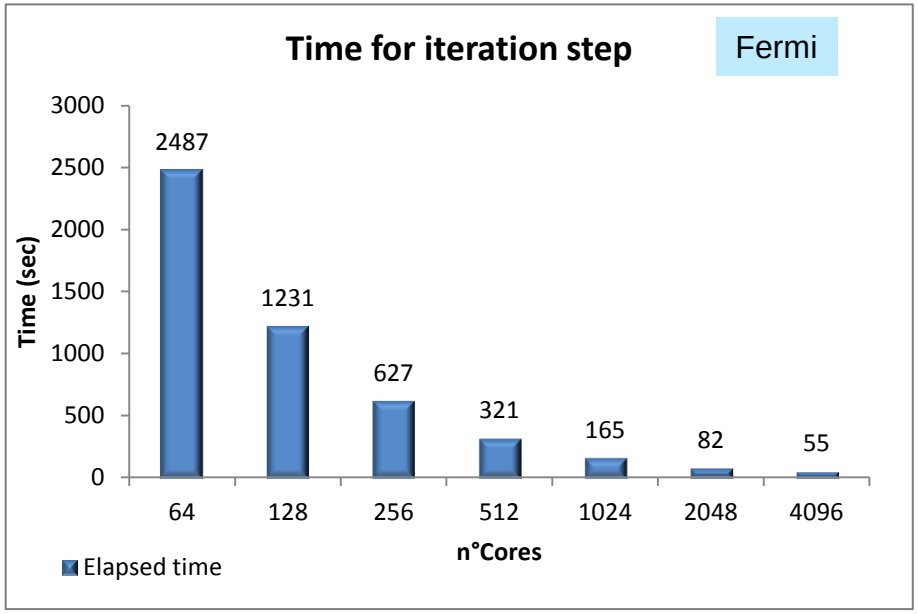


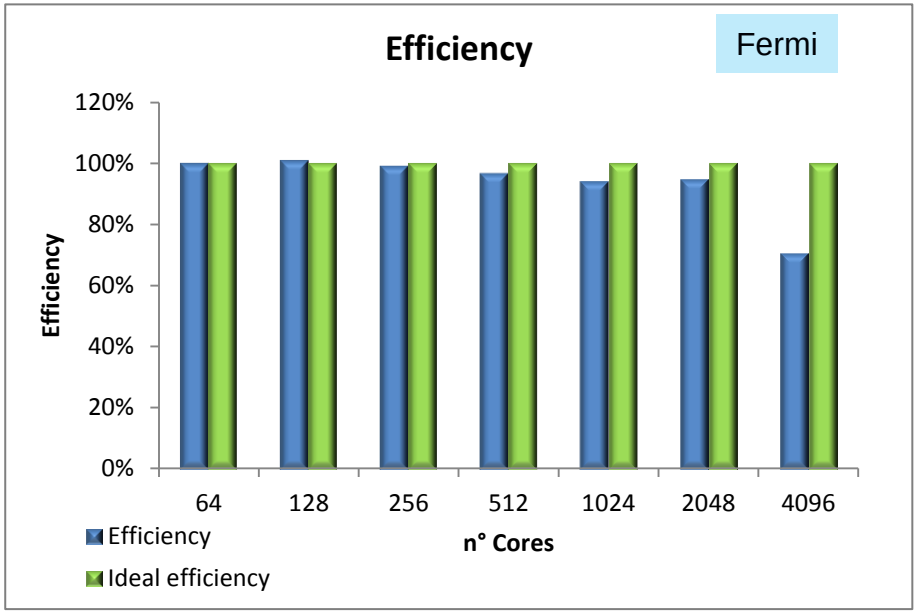


Processors : Intel Xeon 3GHz Quad Core
Interconnesion : LAN 1 Gb/s
GPU : GTX 480 1.4GHz
RAM GPU : 1GB

TIGRA - Benchmark

2D Grid Dimension : 64 x 64
Number of nodes in the velocity space : 7
Number of floating point operations per iteration step : *O*

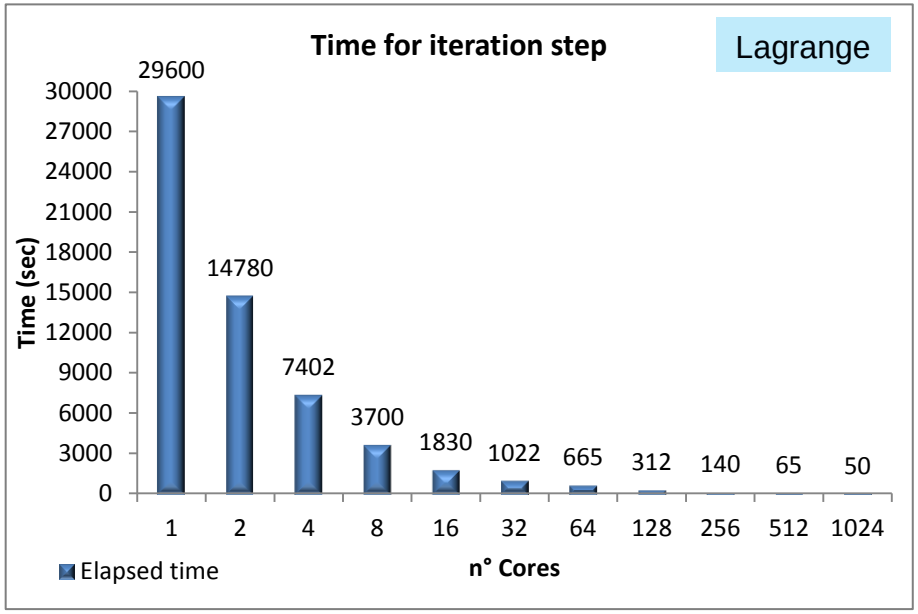


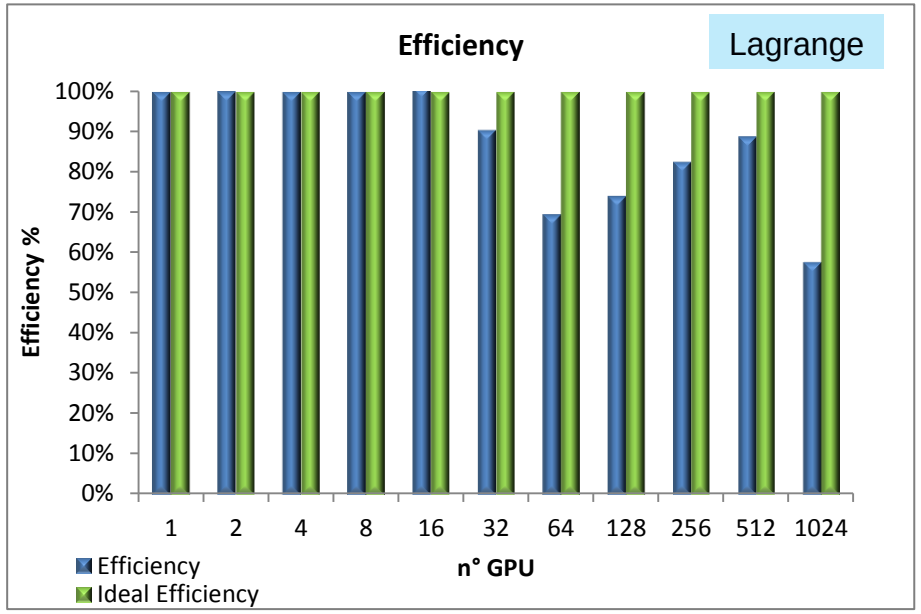


Processors : IBM PowerA2, 1.6 GHz
Node : 16 cores
RAM : 16 GB/Node
Interconnection: Proprietary Torus 5D.

TIGRA - Benchmark

2D Grid Dimension : 64 x 64
Number of nodes in the velocity space : 7
Number of floating point operations per iteration step : *O*

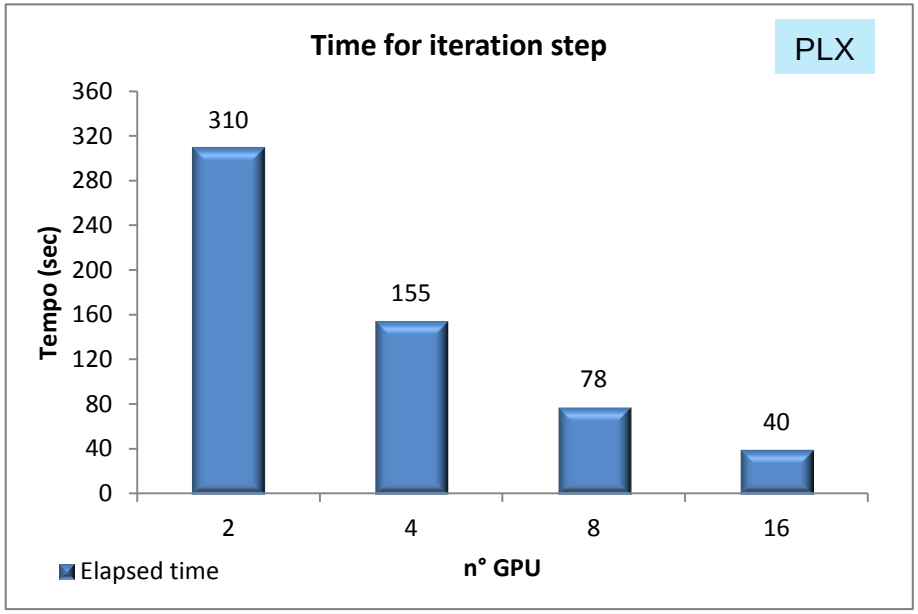


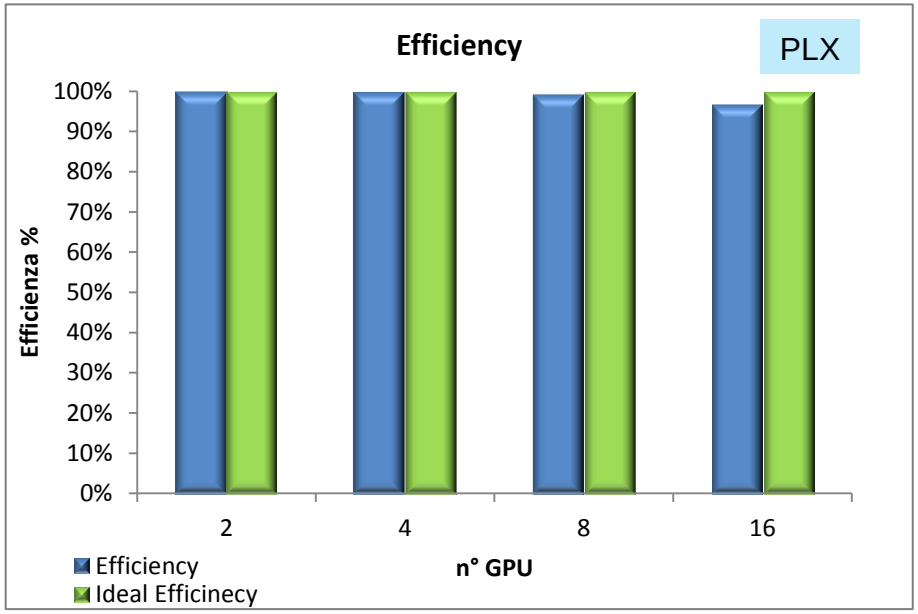


Processors : Intel Xeon X5660 2.80 GHz exacore
Node : 12 cores
RAM : 24 GB/Node
Interconnection: Infiniband QDR 40 Gb/s

TIGRA - Benchmark

2D Grid Dimension : 64 x 64
Number of nodes in the velocity space : 7
Number of floating point operations per iteration step : *O*





Processors : Intel Xeon X5660 2.40 GHz exacore
Nodes : 12 cores
RAM : 48 GB/Node
Interconnesion : Infiniband QDR 40 Gb/s
GPU : Tesla M2070 1.1 GHz
RAM GPU : 6GB

TIGRA – Future work

- Cache blocking techniques
- Introduction of a second layer MPI to allow high scalability even for small values of N

OpenFOAM Optimization BGQ Exploitation

Outline

- **OpenFOAM**
  - Introduction
  - Optimization strategy
  - Benchmark
  - Future work

OpenFOAM - Introduction

- The OpenFOAM® (Open Field Operation and Manipulation) CFD Toolbox is a free, open source CFD software package produced by OpenCFD Ltd.
- It has a large user base across most areas of engineering and science, from both commercial and academic organisations.
- OpenFOAM has an extensive range of features to solve anything from complex fluid flows involving chemical reactions, turbulence and heat transfer, to solid dynamics and electromagnetics.

OpenFOAM - Introduction

- Solver Capabilities
  - Incompressible flows
  - Multiphase flows
  - Combustion
  - Buoyancy-driven flows
  - Conjugate heat transfer
  - Compressible flows
  - Particle methods (DEM, DSMC, MD)
  - Other (Solid dynamics, electromagnetics)

- Library Functionality
  - Turbulence models
  - Transport/rheology models
  - Thermophysical models
  - Lagrangian particle tracking
  - Reaction kinetics / chemistry

OpenFOAM – Known scalability bottlenecks

- Communication structure

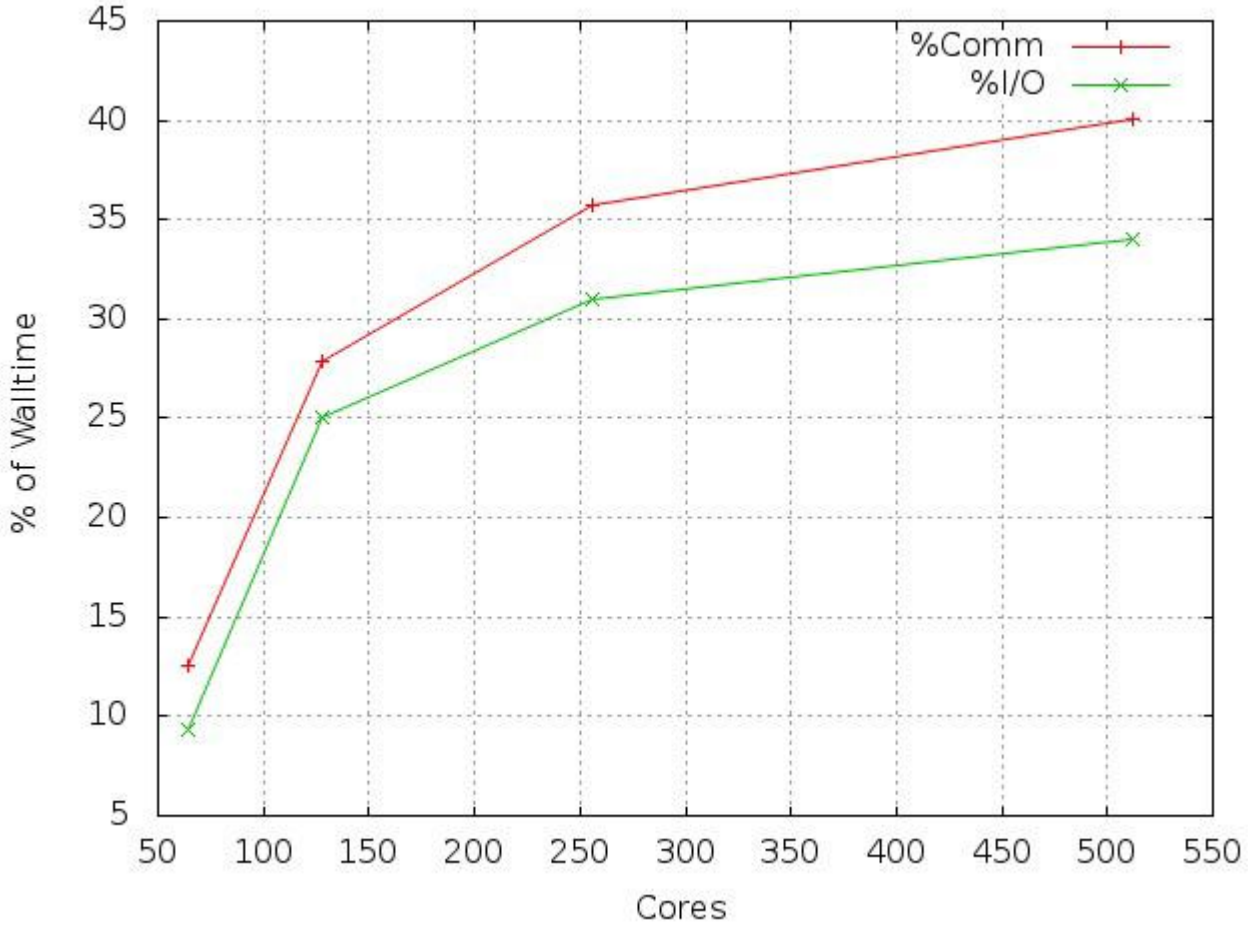
  ➢ Blocking MPI calls (MPI_Waitall, MPI_Allreduce) become dominant as the number of MPI processes increases. Processes communicate with one another through reading and writing of files.

- I/O design

  ➢ The number of files created and read doubles with each doubling of number of processes. The average file size is at same time approximately halved.
  ➢ In OpenFOAM each MPI process creates multiple files and updates them reading/writing small chunks of data

OpenFOAM – MPI and I/O bottlenecks

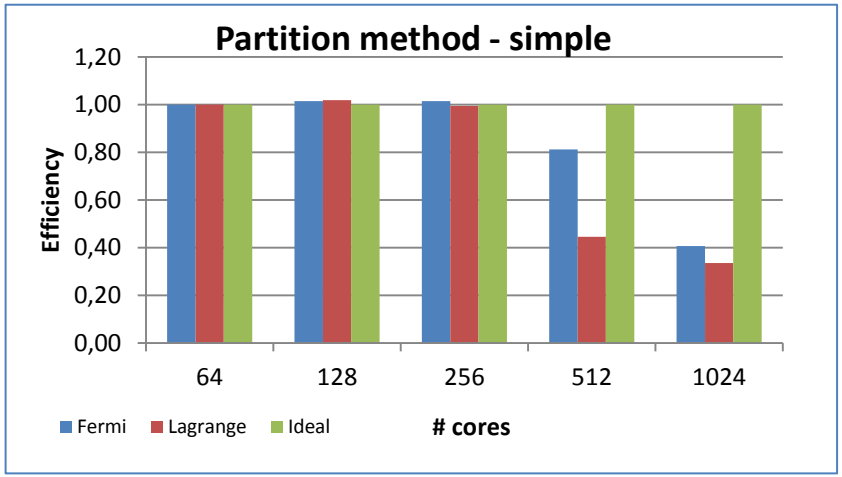
- In the region of 512 cores or more, the remaining portion of time left for the actual solver calculations is seen to be a maximum of 26% of the overall execution time.



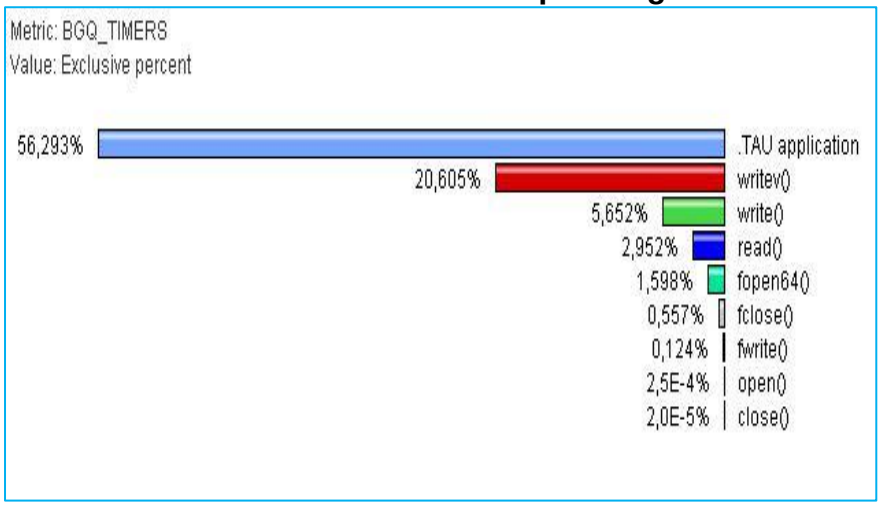
% of walltime taken by Communication and I/O (interFoam solver)

OpenFOAM - I/O bottleneck

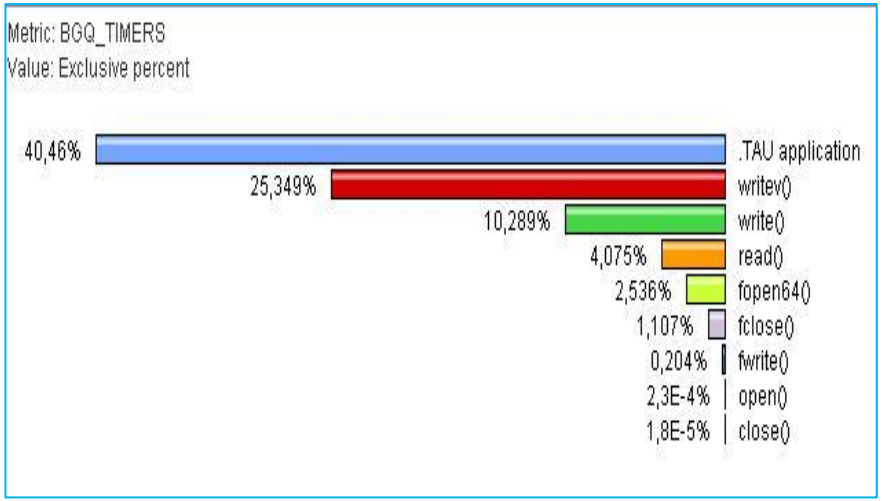
Test case : Cavity 3D
Mesh elements : 10.000.000



I/O profiling : 512 cores

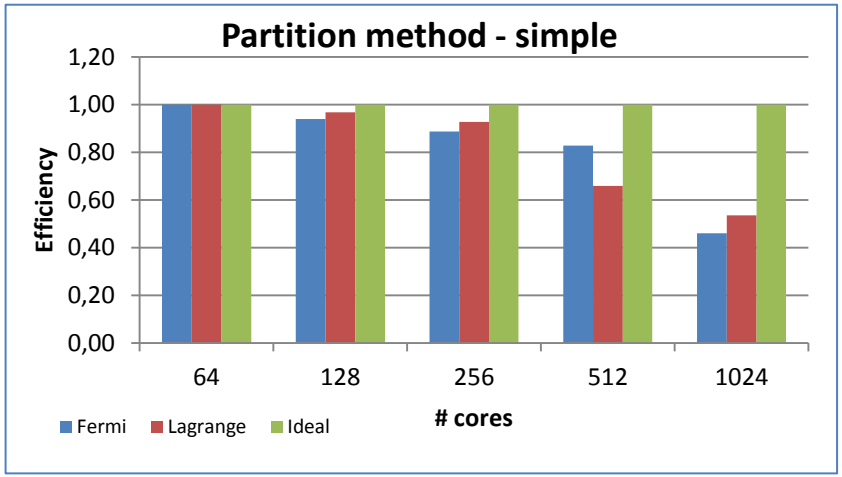


I/O profiling : 1024 cores

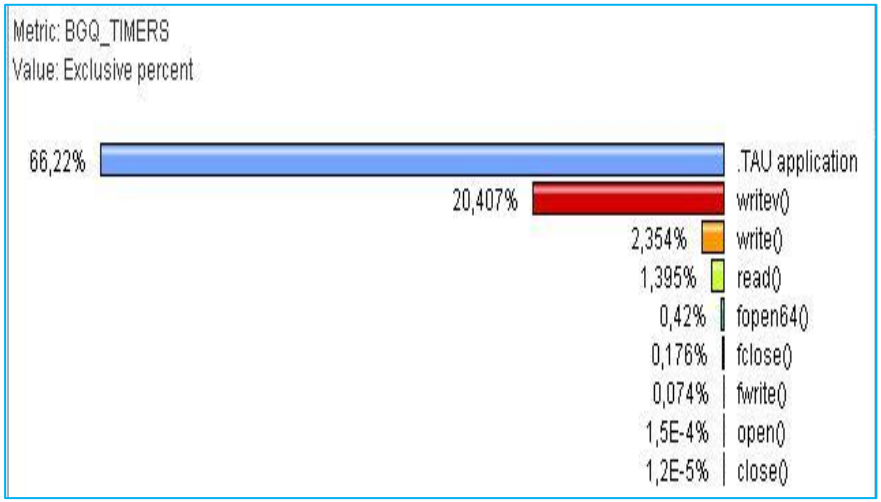


OpenFOAM - I/O bottleneck

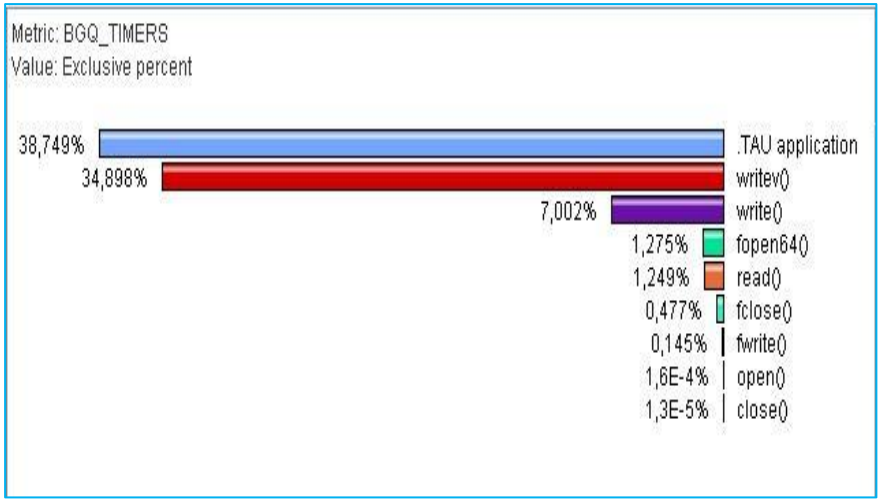
Test case : Cavity 3D
Mesh elements : 20.000.000



I/O profiling : 512 cores



I/O profiling : 1024 cores



OpenFOAM – I/O possible optimizations

- Currently within OpenFOAM process communication and file I/O are heavily linked as reading and writing to files is the chosen method of sharing data.
- Monitoring file timestamps with such frequency blocks communication and eliminate any hope of good scaling.
- OpenFOAM uses a IOStream object to read and write updated parameters. IOstream is used throughout the application of more than a million lines of C++ code.
- A new scheme of communication and I/O are under such conditions not easily introduced. Though, it is worth investigating whether a I/O-library like parallel-NetCDF can be used by OpenFOAM.

OpenFOAM – Solver Hybridization strategy

- A multi-threaded hybrid MPI/OpenMP version of the solvers should mitigate the time spent in MPI routines with the increase in the number of cores.

- Test case : Lid-driven Cavity 3D
  - Number of mesh elements: 10.000.000 and 20.000.000
  - OpenFOAM solver for incompressible laminar Navier-Stokes equations : IcoFoam
  - Solver : PCG
  - Preconditioner : Diagonal

- The most relevant operations performed during each PCG iterative cycle are **scalar products, preconditioning steps and matrix-vector multiplications.**

OpenFOAM – Solver Hybridization strategy

File PCG.c

Only master thread

Executes Matrix-Vector multiplications. Called by all MPI processes. Contains OpenMP directives.

```
// --- Check convergence, solve if not converged
#pragma omp master
    {
   convergenceCheck = solverPerf.checkConvergence(tolerance_, relTol_);
    }
#pragma omp barrier
 if (!convergenceCheck)
            {
                  // --- Solver iteration
                  do
                  {
                 …          …          …          …
 // --- Update preconditioned residual
                 matrix_.AmulOmp(wA, pA, interfaceBouCoeffs_, interfaces_, cmpt);

                 …          …          …          …
                 }
            }
```

OpenFOAM - Linear algebra routine Hybridization strategy

File lduMatrixATmul.C

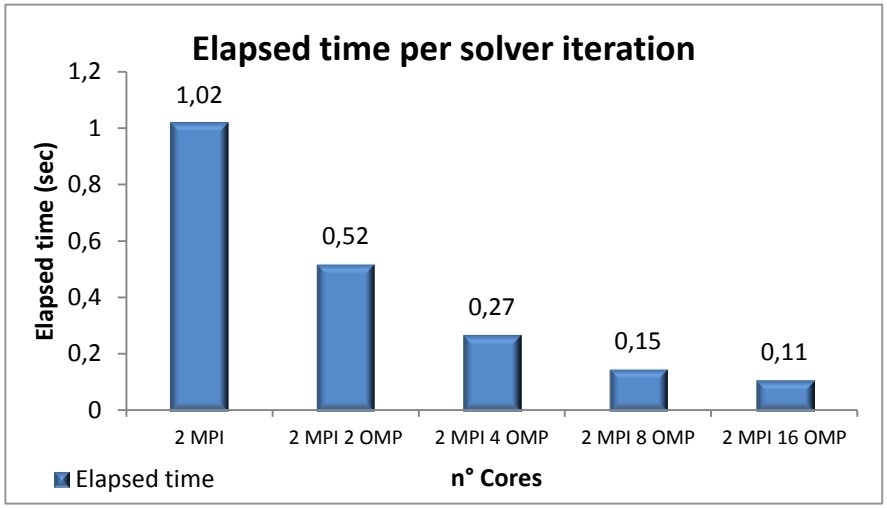
Only master thread

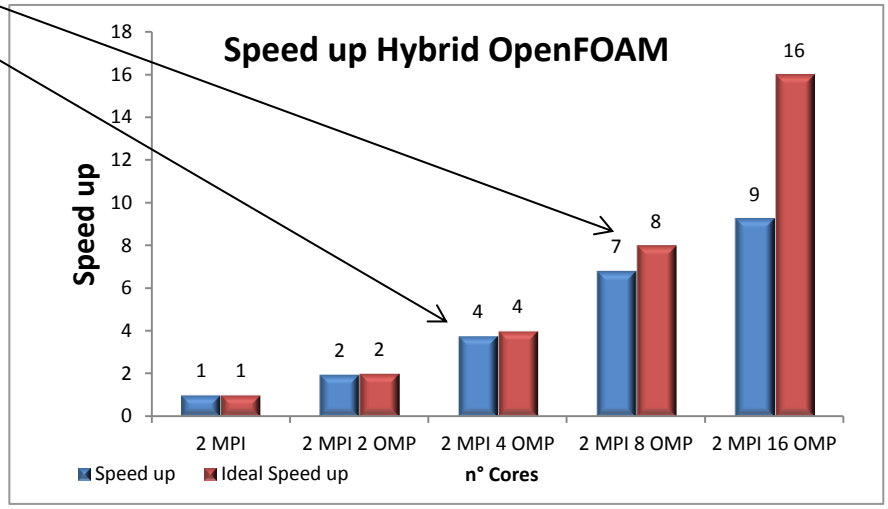
Executes Scalar and Matrix-Vector multiplications.

```
#pragma omp master
    {
        // Initialise the update of interfaced interfaces

                ...         ...         ...         ...
    }
#pragma omp barrier

#pragma omp for schedule(static)
for(label cell=0; cell < nCells; cell++)
    {
        ApsiPtr[cell] = diagPtr[cell]*psiPtr[cell];

     }
#pragma omp for schedule(static)
for(label cell=0; cell < nCells; cell++)
        {
for(register label jj=cell2uPtrOrd[cell]; jj < cell2uPtrOrd[cell+1]; jj++)
            {
                ApsiPtr[cell] += lowerPtr[faceOrd[jj]]*psiPtr[lPtrOrd[jj]];
            }
            for(register label jj=cell2lPtr[cell]; jj < cell2lPtr[cell+1]; jj++)
            {
                ApsiPtr[cell] += upperPtr[jj]*psiPtr[uPtr[jj]];
            }
        }
```

OpenFOAM - Hybrid vs Pure MPI single-node

- **The speed up of the hybrid version of the PCG solver and the linear algebra routines is ideal up to 4 OpenMP threads and acceptable up to 8 OpenMP threads**

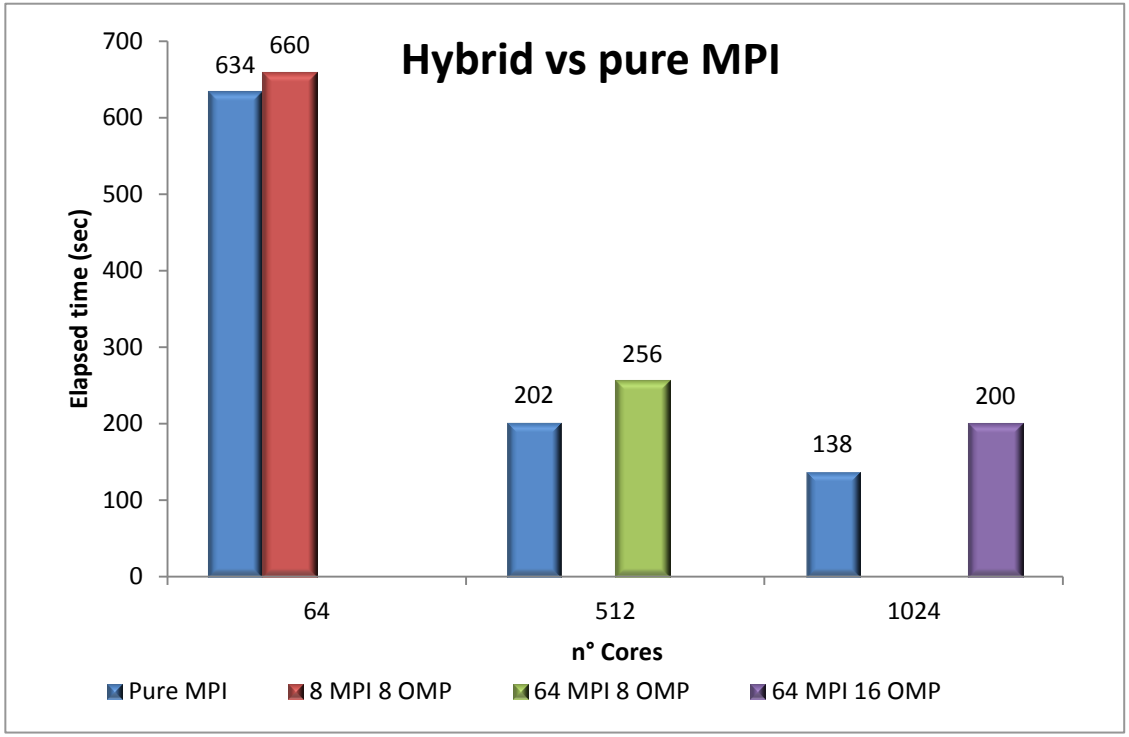




OpenFOAM - Hybrid vs Pure MPI multiple-nodes

- In simulations where OpenFOAM is launched with a large number of cores, the hybrid version should reduce MPI communications, improving scalability.
- But using the same number of cores the pure MPI version is faster respect to the hybrid one.

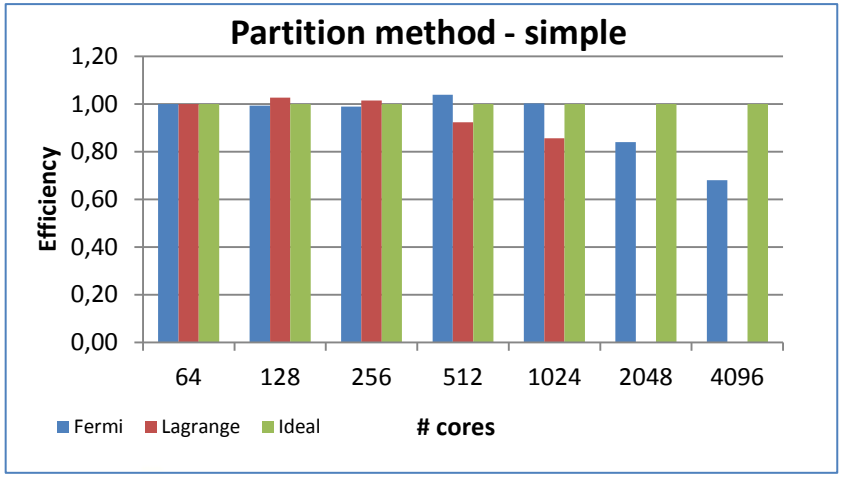
Why ?



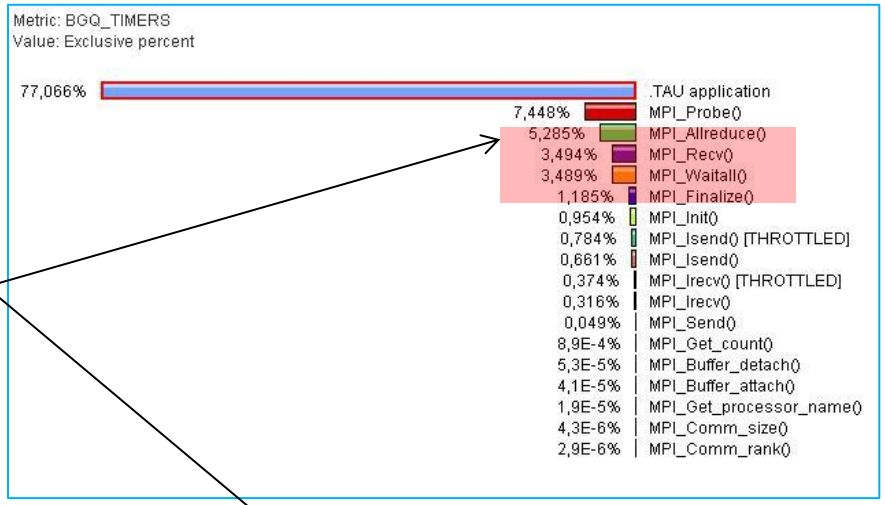
OpenFOAM - Hybrid Performance Issues

Test case : Cavity 3D
Mesh elements : 10.000.000

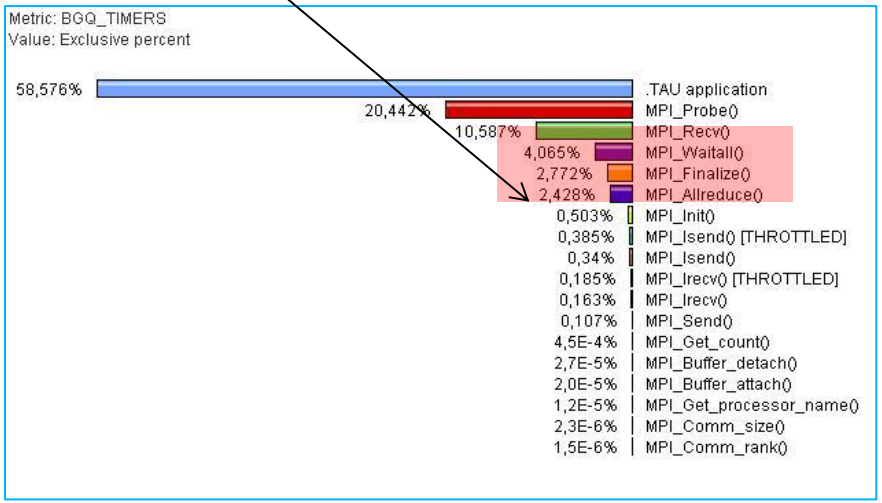
Time spent for MPI communications and above all for MPI_Allreduce which is called during linear system solution is too short



MPI profiling : 1024 cores



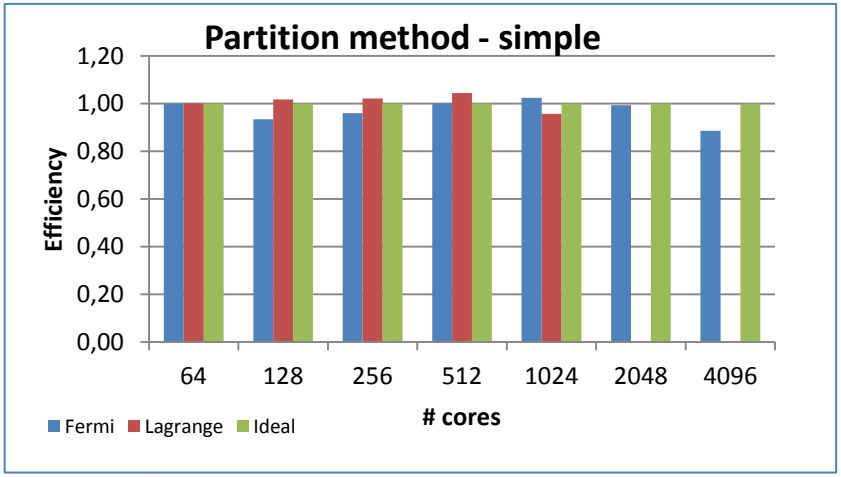
MPI profiling : 2048 cores



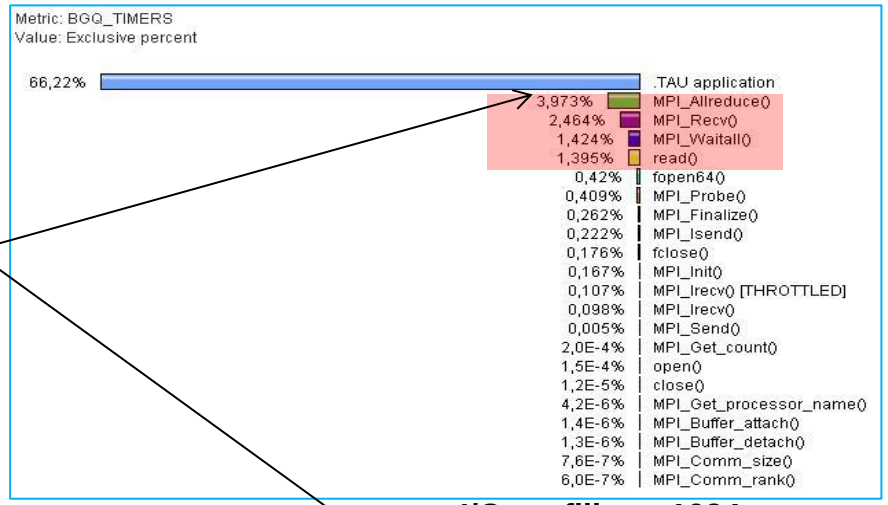
OpenFOAM - Hybrid Performance Issues

Test case : Cavity 3D
Mesh elements : 20.000.000

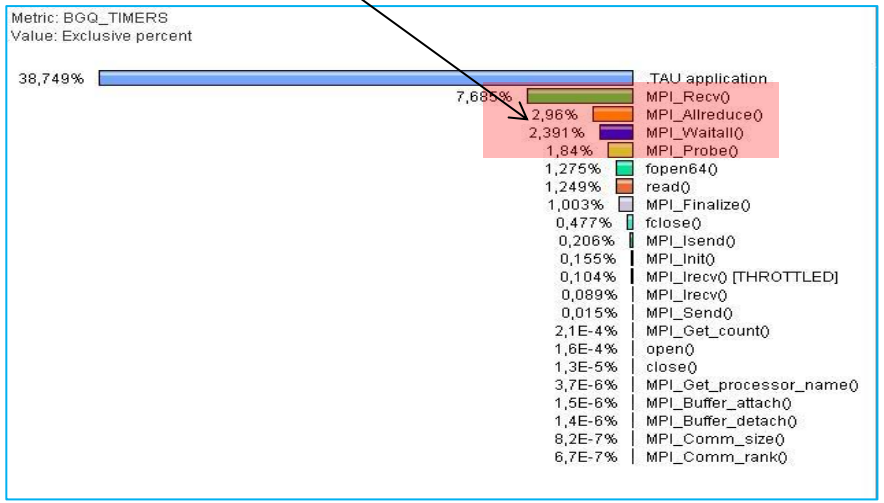
Time spent for MPI communications and above all for MPI_Allreduce which is called during linear system solution is too short



MPI profiling : 1024 cores



I/O profiling : 1024 cores



OpenFOAM - Future work

- Look for possible optimizations of the hybrid parts.

- Performance checking of the hybrid version on real and complex meshes where MPI communication has a more important role.