Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Enabling SPEED for near Real-time Earthquakes Simulations

Paride Dagna[a]*

[a]*CINECA-SCAI Department, Via R. Sanzio 4, Segrate (MI) 20090, Italy*

Abstract

SPEED (Spectral Element in Elastodynamics with Discontinuous Galerkin) is an open source code, jointly developedby the Department of Structural Engineering and of Mathematics at Politecnico di Milano, for seismic hazard analyses.
In this paper, performanceresults, which come from the optimization and hybridization work done on SPEED, tested on the CINECA Fermi BG/Q supercomputer will be shown. A comparison between the pure MPI and the hybrid SPEED versions on three earthquake scenarios, with increasing complexity, will be presented and a detailed analysisof the advantages that come from hybridization and optimization of the computing and I/O phases will be given.

1. Introduction

The variety and extent of impacts caused by the destructive earthquakes of recent years on the built and natural environment, such as during the 2008 Wenchuan (China), 2009 L'Aquila (Italy) and 2010-2011 Christchurch (New Zealand) earthquakes, revealed dramatically the need for improving the tools for seismic risk assessment. In the last twenty years there has been an impressive progress worldwide towards the development of deterministic ground shaking scenarios used as input for seismic hazard and risk assessment studies, relying on numerical simulation of seismic wave propagation under realistic tectonic and geo-morphological conditions.

Nowadays, standard seismic hazard analysis is achieved through a well-consolidated approach, namely Probabilistic Seismic Hazard Analysis [1].As a basic ingredient, the method requires a suitable tool for predicting the earthquake ground motion produced at the target site by potential earthquakes along any possible rupture areas of the seismogenic source.

The standard approach is through Ground Motion Prediction Equation (GMPE). However, despite their simplicity, GMPEs may not be suitable to reproduce specific ground motion features due to the scarcity of the calibration data set in the near field of an earthquake (i.e.: Christchurch, New Zealand, or recent Emilia-Romagna, Italy) or for very large subduction events (i.e.: Maule, Chile, or Tohoku, Japan).

* Corresponding author. *E-mail address*: p.dagna@cineca.it

Therefore, under such conditions, the only way to estimate accurately the seismic hazard might to adopt a more physical approach capable to limit the systematic bias between data and prediction [2].

However, to include in a single physics-based model the coupled effects of the seismic source, the propagation path through complex geological structures and localized superficial irregularities, such as alluvial basins orsynthetic structures, still poses challenging demands on computational methods and resources due to the coexistence of very different spatial scales, from a few tens of kilometers, with reference to the seismic fault, up to a few meters, or even less, when considering some structural elements.

Motivated by these considerations, the open-source code SPEED (Spectral Element in Elastodynamics with Discontinuous Galerkin), was developed jointly by the Department of Structural Engineering and of Mathematics at Politecnico di Milano[3].This paper presents an improved hybrid version or the code that allows near real time simulations on peta scaling architectures with high efficiency.

SPEED is written in Fortran90 and conforms strictly to the Fortran95 standard. The SPEED package uses parallel programming based upon the Message Passing Interface (MPI) library relying on the domain decomposition paradigm. The mesh generation may be accomplished by a third party software, e.g. CUBIT [4]and then exported in a compatible format. Load balancing is facilitated by graph partitioning based on the open-source library METIS [5], which is included in the package. The I/O operations accomplished by SPEED during its execution do not require external libraries.

The paper is organized as follows. In Section 2 the main features of the HPC system used are listed, in Section 3 the bottlenecks of the original pure MPI version of SPEED are highlighted and the new hybrid parallelization strategy is presented.Performance and results analysis areshown in Section 4.

2. Tier-0 system specifications and installation notes

All the tests that will be presented in this document have been executed on Fermi, a Tier-0 machine which is at present the main CINECA's HPC facility.

Fermi [6] is an IBM BlueGene/Q system composed of 10.240 PowerA2 sockets running at 1.6GHz, with 16 cores each, totaling 163.840 compute cores and a system peak performance of 2.1 PFlop/s. The interconnection network is a very fast and efficient 5D Torus.Fermi is one of the most powerful machines in the world, and it has been ranked #9 in the top 500 supercomputer sites list published in November 2012.

SPEED was built with IBM XL compilers and BG/Q system proprietary MPI. The code has been profiled using the TAU Performance System®analyzer [7].

3. Hybridization strategy

The algorithm implemented in SPEED can be subdivided in three different stages, as described in Figure 1: in the first two partsof the algorithm (1-2) the parameters for the parallel computation are set, while in the latter part (3) the time advancing scheme is performed and the output results are written.

After reading the input data (1.a) and accomplishing the domain decomposition (1.b) the master process calls the mesh partitioner METIS to decompose the computational domain and to generate local meshes for all the other processes. The latter are written into separate files, and then read by each MPI process(1.c). For parametric simulations the domain decomposition is executed once.

The second stage of the algorithm is related to the setup of arrays and variables employed within the time marching scheme (2.a). This also includes the preparation of buffers for MPI communications (2.b).In the last part of the algorithm the time advancing scheme is executed. For each time step in the iterative loop, three different kinds of operations are performed: local computations (3.a), MPI communications (3.b) and output writing (3.c).

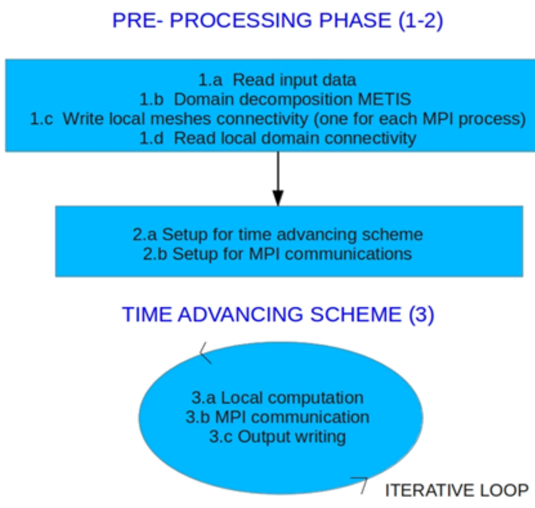
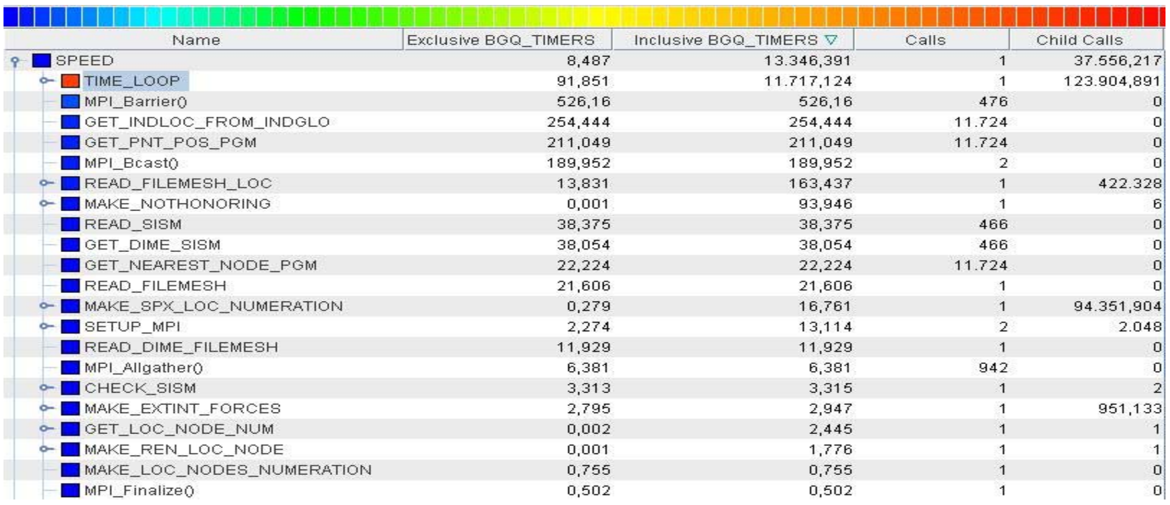

PRE- PROCESSING PHASE (1-2)
1.a Read input data
1.b Domain decomposition METIS
1.c Write local meshes connectivity (one for each MPI process)
1.d Read local domain connectivity
2.a Setup for time advancing scheme
2.b Setup for MPI communications
TIME ADVANCING SCHEME (3)
3.a Local computation
3.b MPI communication
3.c Output writing
ITERATIVE LOOP

Fig. 1. SPEED Workflow.

In Figure 2,obtained with the TAU profiler, ordered according to the inclusive time in seconds (time spent in the routine and its descendants), the exclusive and inclusive time and the number of calls and child calls is presented. Looking at the image it’s clear that the “TIME_LOOP” routine, called by “SPEED”, which is the main routine of the whole code, is the most critical and time consuming section. This is true above all for large meshes and long simulation times, since the time advancing scheme is carried out. For this reason the time loop routine ad its descendants became the object of a deeper investigation.

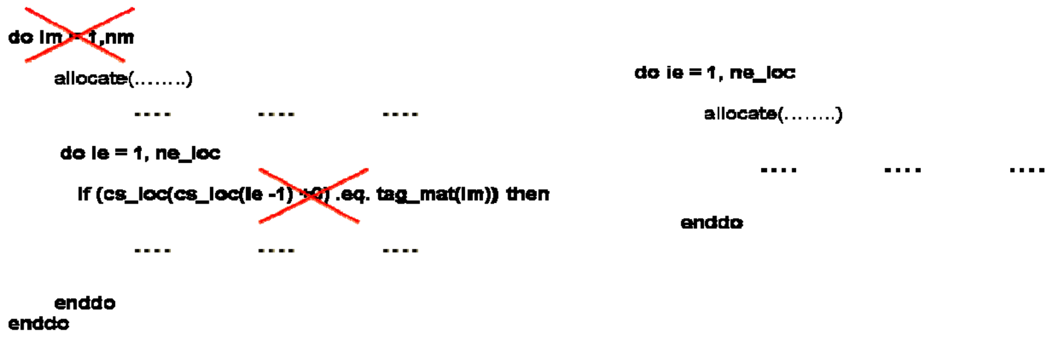


Name	Exclusive BGQ_TIMERS	Inclusive BGQ_TIMERS ▽	Calls	Child Calls
SPEED	8,487	13.346,391	1	37.556,217
TIME_LOOP	91,851	11.717,124	1	123.904,891
MPI_Barrier()	526,16	526,16	476	0
GET_INDLOC_FROM_INDGLO	254,444	254,444	11.724	0
GET_PNT_POS_PGM	211,049	211,049	11.724	0
MPI_Bcast()	189,952	189,952	2	0
READ_FILEMESH_LOC	13,831	163,437	1	422.328
MAKE_NOTHONORING	0,001	93,946	1	6
READ_SISM	38,375	38,375	466	0
GET_DIME_SISM	38,054	38,054	466	0
GET_NEAREST_NODE_PGM	22,224	22,224	11.724	0
READ_FILEMESH	21,606	21,606	1	0
MAKE_SPX_LOC_NUMERATION	0,279	16,761	1	94.351,904
SETUP_MPI	2,274	13,114	2	2.048
READ_DIME_FILEMESH	11,929	11,929	1	0
MPI_Allgather()	6,381	6,381	942	0
CHECK_SISM	3,313	3,315	1	2
MAKE_EXTINT_FORCES	2,795	2,947	1	951,133
GET_LOC_NODE_NUM	0,002	2,445	1	1
MAKE_REN_LOC_NODE	0,001	1,776	1	1
MAKE_LOC_NODES_NUMERATION	0,755	0,755	1	0
MPI_Finalize()	0,502	0,502	1	0

Fig. 2. SPEED, view of the time spent in the main routines of the code.

Concerning the computing phase,in the TIME_LOOP routine, two nested loops (fig. 3) were performed to span all the subdomain elements ordered by a progressive material indexand a conditional clause was present at the beginning of the inner loop so that operations were performed only after checking if the current element belonged to the correct material.

Looking carefully at the logic of the algorithm it was found that the ordered loop over materials and the conditional clause were not necessary. Consequentlythe external loop on materials was removed and the internal loop adapted in order to span all the elements for each subdomain (fig 3).



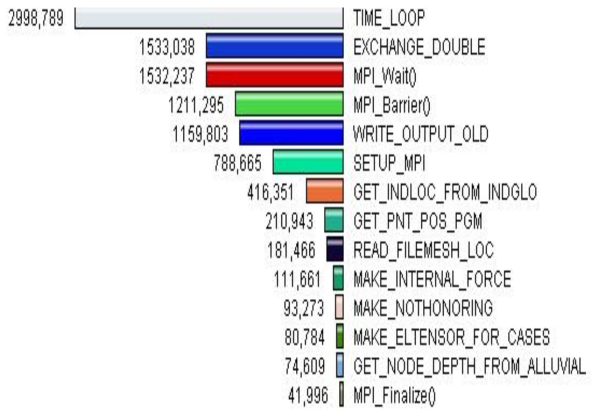
```
~~do im = 1,nm~~
     allocate(........)
              ....        ....        ....
      do ie = 1, ne_loc
        if (cs_loc(cs_loc(ie -1) ~~+0)~~ .eq. tag_mat(im)) then
              ....        ....        ....

     enddo
enddo
```

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

Fig. 3. SPEED, optimization of the TIME_LOOP routine.

Since, operations made on different elements are independent, the solution just proposed allowed to add, at this level, a further Open-MP parallelization. The optimized hybrid version of SPEED was then obtained creating a parallel region before the loop over subdomain elements.

Measuring the exclusive time per call it was found that the old I/O routine “WRITE_OUTPUT_OLD” in which I/O operations were coded was a serious bottleneck to scalability (fig. 4, image on the left). The original algorithm implemented in this function stated that each MPI process can write multiple files but only one row was added at a time. As a consequence, for large scenarios, with many cores, required managing thousands of files in parallel by the file system, causing an unacceptable slowdown or even the complete stall of the simulation.



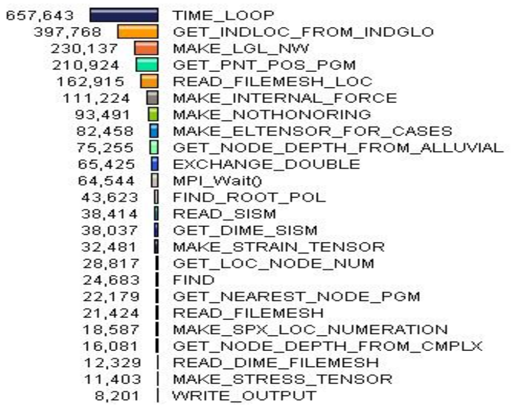


Fig. 4. SPEED, optimization of the I/O and computing phase. Times are in seconds. Old routine on the left “WRITE_OUTPUT_OLD”, new routine on the right “WRITE_OUTPUT”. Old “TIME_LOOP” routine on the left, new “TIME_LOOP” routine on the right.

A new strategy and a new routine was implemented (WRITE_OUTPUT) where each MPI process manages only one file and multiple rows can be written at a time, resulting in a dramatic reduction of the time spent for I/Ooperations (fig. 4, image on the right). The optimizations described allowed to obtain an overall 5x speed-up of the TIME_LOOP routine (fig. 4).

The latter optimized version, not only improves the performance of SPEED in terms of the overall computational time, but also solves a great memory constraint present in the pure MPI version.In fact in the pure MPI version only the single MPI process is able to work on its own chunk of data whereas in the hybridized version each MPI process can take advantage of a selected number of OpenMP threads to work on the same chunk of data. This benefit can be a key turning point for a more effective memory usage of the available hardware when real earthquake scenarios are faced.Within this kind of problems, stated a defined value of memory and of computational cores per node, the number of MPI process that can be instantiated on the single node is not arbitrarysince it is bounded by the available memory on the node and the memory needed by each MPI task to allocate the data used within each subdomain. In the pure MPI version this value represent also the number of usable computational cores per node whereas on the hybridized version each MPI process can call several OpenMP threads to fully exploit the multi-core hardware configuration. For this reasons in the pure MPI version to simulate real earthquake scenarios, only a subset of the available cores per node can be used,wasting a value of computing resourcesper node that is related to the memory request of the sub-process and to the memory availability of the hardware configuration. Using the hybridizedversion of the code, as shown in sections 4.1 and 4.2, the value of MPI tasks and OpenMP threadsper computational node can be mixed to fully exploit the hardware availability.

In our hardware configuration, as presented in section 2, the IBM PowerA2 processors allow to use 16GB of ram and 16 computational cores per node. So that to face the two test cases analyzed (see section 4.1, 4.3 in the forthcoming), in order to respect the memory constraint of the single computational node, we were obliged to use only 8 cores and 4 cores per node respectively in the pure MPI version of the code.So that for instance to submit the requested degree of parallelism of 64 we needed to reserve on the machine 128 and 256 cores respectively wasting a relevant amount of unused computational cores. By converse in the hybridized version of the code we were able to use for each test case all the cores per node available by splitting the overall degree of parallelism request in MPI tasks and OpenMP threads.So that for instance the reserved128 and 256 cores can be fully used by means of 64 MPI process plus 2 OpenMP threads and 64 MPI process plus 4 OpenMP threads.

For these reasons in the forthcoming, in order to highlight the benefits obtained by hybridized version of the code with the respect to the pure MPI one, we performed the speed-up evaluation by normalizing the elapsed-time using either the number of cores used and the number of cores reserved.

4. Test cases and results analysis

In sections 4.1 and 4.2 the selected test cases and the related performance and scalability results obtained with thepure MPI and hybrid version of SPEED will be presented.

Because of the memory issues explained in chapter 3, the pure MPI version had to be run using only half of the sixteen cores available on each node for tests 4.1 and 4.2 and only 4 of the sixteen cores available on each node for benchmark 4.3.

For this reason in sections 4.1 and 4.2 a further test, based on the number of cores necessarily reserved by the pure MPI version because of the memory constraints, is inserted to demonstrate how the hybrid solution which is not memory bounded, allows to fully exploit the features of the BG/Q PowerA2 processors,capable of managing up to 4 threads per core for a total amount of 64 processes per node [8].

The results of the pure MPI runs refer to thecode with improved I/O,because of the issues explained in section 3, over 1024 coresthe simulations stalledwith the old I/O routine.

In section 4.3 a further performance and scalability test on a larger domain, referring to the Po plain, is shown.

4.1. Conforming mesh - Layer Over Halfspace

The problem is known in literature with the acronym LOH (Layer Over Halfspace) [9] and it is currently a reference benchmark for different advanced numerical codes for seismic wave propagation. The computational domain is O = [30 × 30 × 17] km, with 814.833hexahedral elements, varying from size of 100 m, in the first quadrant, to 300 m in the remaining part of the domain. The top layer has a thickness of 1 km.

In table 1 the speed up and efficiency of the pure MPI and hybrid version are reported. Due to the very high degree of parallelism tested (up to 8k cores) and to the extremely large computational time requested by the single core test (stimated in about 300 days) we started our analysis starting from a minimum degree of parallelism equal to 64.The results show how the combination of MPI tasks and OpenMP threads allow increasing the performance, using the same number of cores, because of the reduced MPI communication at the interfaces between subdomains.

Table 1. Speed-up and efficiency of the pure MPI and hybrid OpenMP + MPI .

Pure MPI version				Hybrid MPI + OpenMP					Hybrid vs Pure MPI
Cores #	Time (sec)	Speed up	Efficiency	MPI tasks	OpenMP threads	Time (sec)	Speed up	Efficiency	
64	469800	64	1	64	0	469800	64	1	1,00
512	63000	477	0,93	64	8	60250	499	0,97	1,05
1024	35500	847	0,83	64	16	35500	847	0,83	1,00
1024				512	2	31500	955	0,93	1,13
2048	18900	1591	0,78	512	4	15950	1885	0,92	1,18
2048				1024	2	16650	1806	0,88	1,14
4096	10800	2784	0,68	512	8	8400	3579	0,87	1,29
4096				1024	4	8600	3496	0,85	1,26
4096				2048	2	8200	3667	0,90	1,32
8192	7407	4059	0,50	1024	8	4500	6682	0,82	1,65
8192				2048	4	4420	6803	0,83	1,68
8192				4096	2	5420	5547	0,68	1,37

Referring to table 1, in figure 5 the speed-up obtained from pure MPI runs and the best mixing of MPI tasks and OpenMP threads is shown.

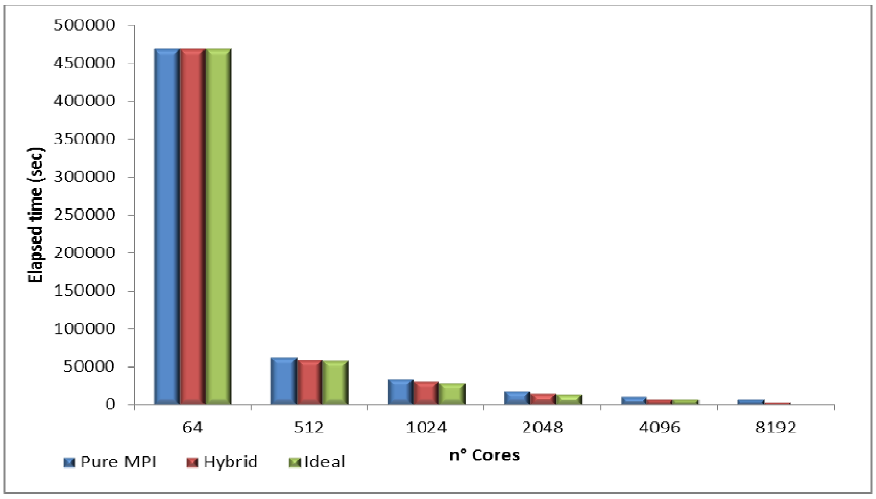


Fig. 5. SPEED, speed-up of pure MPI and mixed OpenMP + MPI solutions.

As underlined in the introduction of chapter 4, because of the huge memory request the pure MPI version had to be run using only half of the available cores on each node. In order to verify how the hybrid solution, which is not affected by memory constraints, outperforms the pure MPI one, the LOH test was conducted using the same number of cores necessarily reserved for the pure MPI runs (table 2, fig. 6).

Table 2.Comparison between original pure MPI and the hybrid OpenMP + MPI. The speed up of the hybrid version is computed in relation to the timings of the pure MPI maintaining the same number of cores reserved.

Pure MPI version			Hybrid MPI + OpenMP				Hybrid vs Pure MPI
Cores Reserved	Cores Used	Time (sec)	MPI tasks	OpenMP threads	OpenMP per core	Time (sec)	
128	64	469800	64	2	1	237600	1,98
128			64	4	2	134000	3,51
128			64	8	4	100350	4,68
1024	512	63000	512	2	1	31500	2,00
1024			512	4	2	16650	3,50
1024			512	8	4	8200	4,67
2048	1024	35500	1024	2	1	16650	2,13
2048			1024	4	2	8200	3,76
2048			1024	8	4	5420	5,26
4096	2048	18900	2048	2	1	8200	2,30
4096			2048	4	2	4600	4,11
4096			2048	8	4	3600	5,25
8192	4096	10800	4096	2	1	5420	1,99
8192			4096	4	2	2800	3,86
8192			4096	8	4	2070	5,22

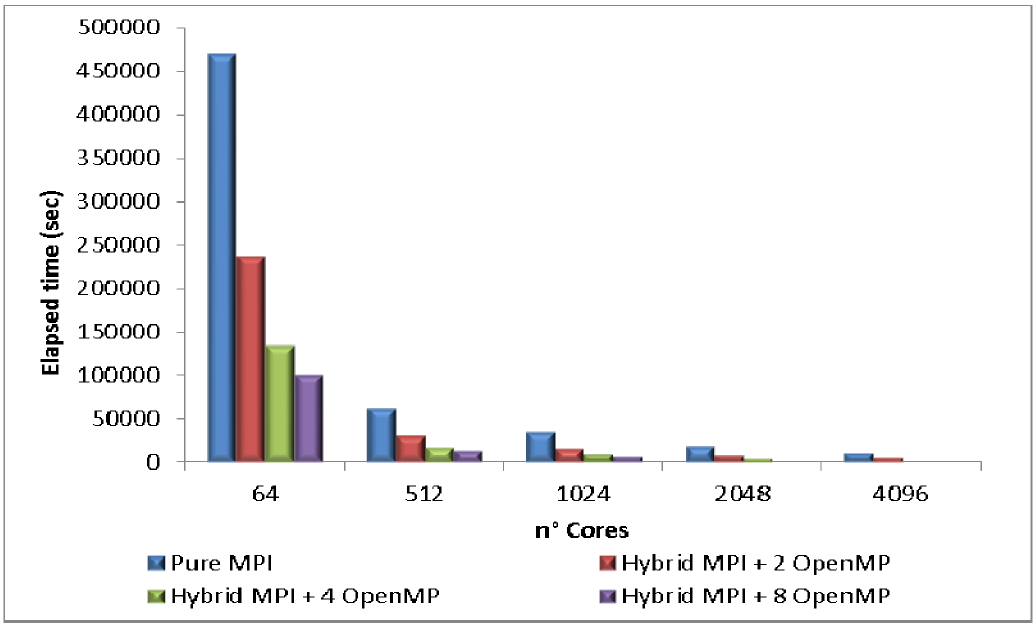


Fig. 6.Comparison between original pure MPI and the hybrid OpenMP + MPI. The speed up of the hybrid version is computed in relation to the timings of the pure MPI maintaining the same amount of cores reserved.

4.2. Non conforming mesh–Layer Over Halfspace

The problem is the same of section 4.1 but for the non conforming mesh the number of hexahedral elements is 70.228, having a size of around 400 m in the upper layer (1 km thickness) and a size of around 650 m in the lower layer (16 km thickness).

Following the same procedure of section 4.1, in table 3 the speed up and efficiency of the pure MPI and hybrid version, maintaining the same number of cores, is reported. The reasons that cause a faster decrease of the parallel efficiency compared to the conforming mesh are explained in chapter 5.

Table 3. Speed-up and efficiency of the pure MPI and hybrid OpenMP + MPI .

Pure MPI version				Hybrid MPI + OpenMP					Hybrid vs Pure MPI
Cores #	Time (sec)	Speed up	Efficiency	MPI tasks	OpenMP threads	Time (sec)	Speed up	Efficiency	
64	104500	64	1,00	64	0	104500	64	1,00	1,00
512	23600	283	0,55	64	8	14350	466	0,91	1,64
1024	16800	398	0,39	64	16	9450	708	0,69	1,78
1024				512	2	11250	594	0,58	1,49
2048	10100	662	0,32	512	4	6200	1079	0,53	1,63
2048				1024	2	7300	916	0,45	1,38

Also for non-conforming meshes, reducing the number of subdomains and MPI tasks and increasing the number of OpenMP threads, with the hybrid solution a better speed-up is achievable (fig. 7).

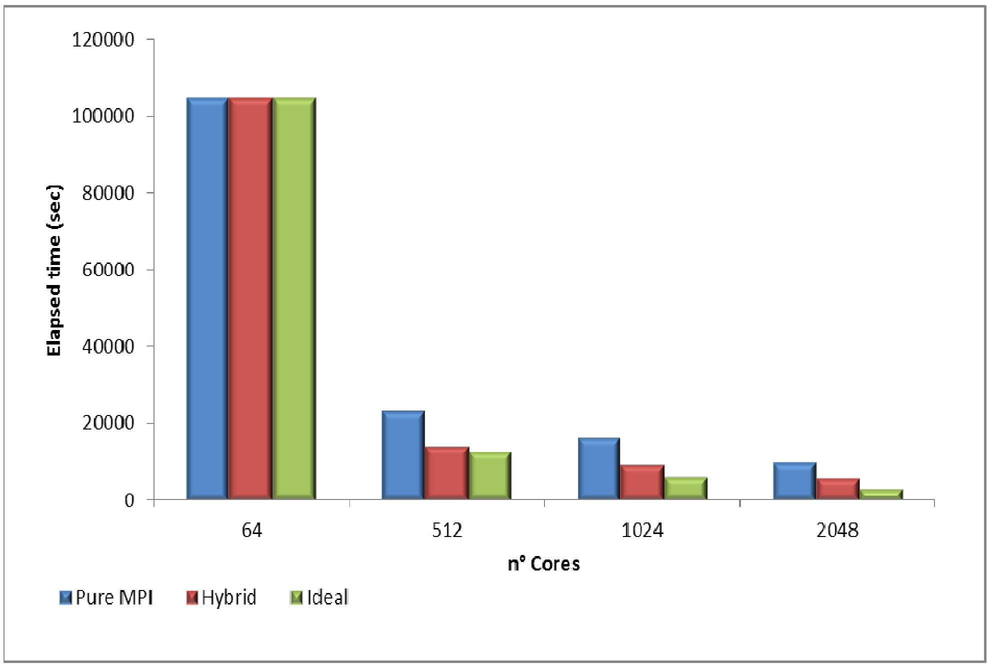


Fig. 7. SPEED, speed-up of pure MPI and mixed OpenMP + MPI solutions.

In the same way of section 4.1 the test reported in table 4 and image 8 confirm the increasing of performances achievable with the hybrid solution due to the better exploitation of the BG/Q architecture.

Table 4. Comparison between original pure MPI and the hybrid OpenMP + MPI. The speed up of the hybrid version is computed in relation to the timings of the pure MPI maintaining the same number of cores reserved**.**

Pure MPI version			Hybrid MPI + OpenMP				Hybrid vs Pure MPI
Cores Reserved	Cores Used	Time (sec)	MPI	OpenMP	OpenMP per core	Time (sec)	
128	64	104500	64	2	1	54000	1,94
128			64	4	2	30150	3,37
128			64	8	4	26550	3,94
1024	512	23600	512	2	1	11250	2,10
1024			512	4	2	6700	3,52
1024			512	8	4	4500	5,24
2048	1024	16800	1024	2	1	7300	2,30
2048			1024	4	2	4500	3,73
2048			1024	8	4	3280	5,12
4096	2048	10100	2048	2	1	5300	1,91
4096			2048	4	2	3150	3,21
4096			2048	8	4	2290	4,41

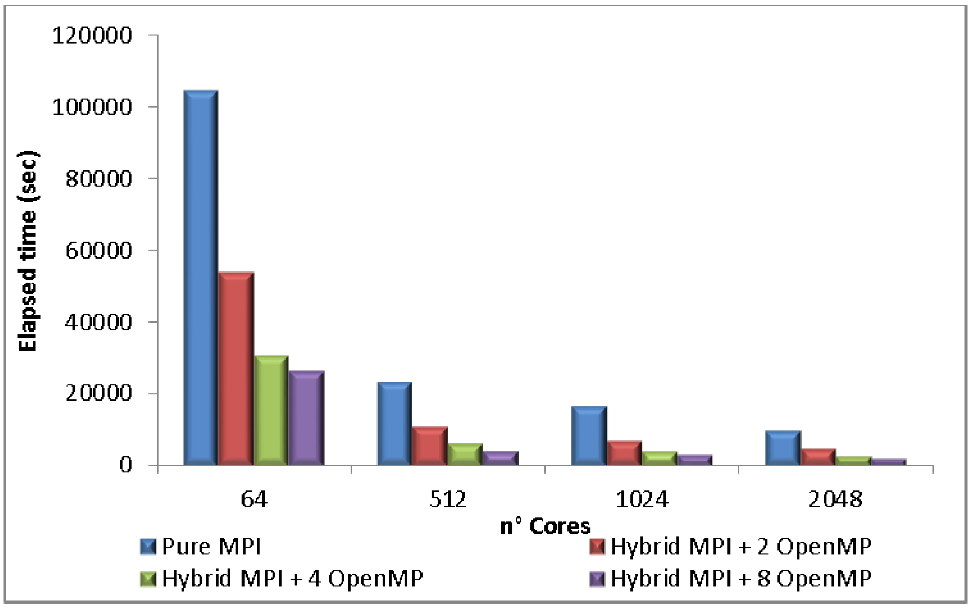


Fig. 8. Comparison between original pure MPI and the hybrid OpenMP + MPI. The speed up of the hybrid version is computed in relation to the timings of the pure MPI maintaining the same amount of cores reserved.

4.3. Structured mesh - Po Plain, Emilia-Romagna

The test case considered is the MW 6.0 29th May 2012 earthquake that struck the Po Plain, Emilia-Romagna, North-eastern Italy. The model extends over a volume of about 74x51x20 km^3 and is discretized using an unstructured hexahedra mesh with characteristic element size ranging from ~ 150 m at the surface to ~ 1400 m at the bottom of the model.

In this test only the hybrid version is used in order to show the scalability obtained from the optimization done on the “setup” and “computing phase” on big real scenarios.

The number of cores (# Cores) is obtained choosing 4 OpenMP threads for each MPI process, (e.g. 1024 cores = 4 threads per 256 MPI processes).

In table 5 and fig. 9 the speed-up is shown up to 16384 cores.

Table 5. Speedup for the test case. The total wall time is divided in two parts: setup operations and time integration.

# Cores	Wall time (s)	Speedup	Setup (s)	Speedup	Time-loop (s)	Speedup
1024	90176	1024	22176	1024	68000	1024
2048	39300	2349	11050	2055	28250	2464
4096	20100	4594	5496	4131	14604	4768
8192	9900	9327	2900	7830	7000	9947
16384	5700	16200	2300	9873	3400	20480

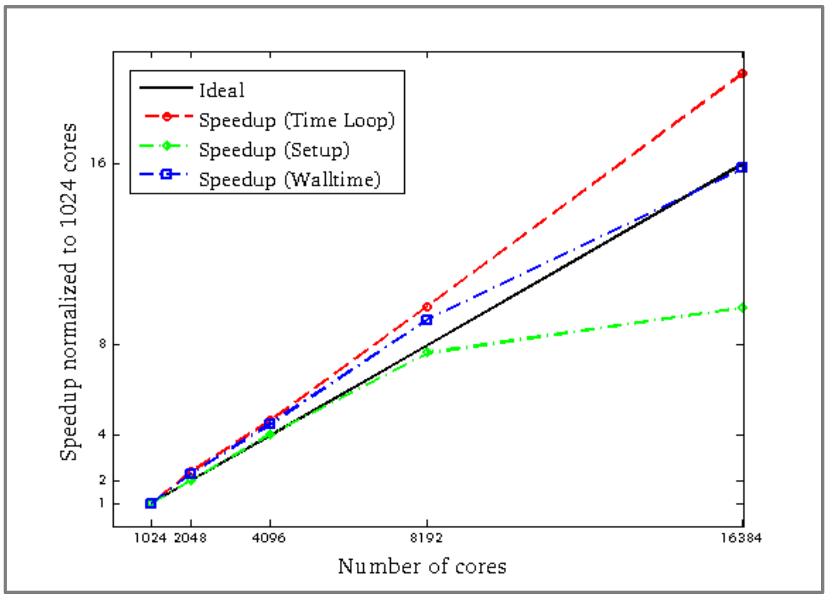


Fig. 9. Parallel efficiency of the code SPEED. Speedup normalized to 1024 cores versus number of cores.

The super linear speedup obtained during the “time-loop” execution is due to cache effects of the BG/Q architecture.

5. Result Analysis and discussion

The tables and graphs of sections 4.1 and 4.2 show how the hybrid optimized version is more performingthan the pure MPI one and able to exploit almost all the computing power of the PowerA2 processors. Using shared memory inside of subdomains,with OpenMP threads, it’s possible to reduce domain decomposition among MPI tasks and consequently the amount of MPI communications, increasing the scalability. This behavior is more evident in non conforming meshes.In the non conforming mesh the number of operations done by MPI tasks differs for the tasks that have to deal with interface elements where additional terms must be computed to control the jump of the solution among non conforming elements. For this reason load balancing and domain decomposition is not so straightforward and efficiency decreases more quickly than for conforming meshes.

Anyway, as listed in table 3, hybridization allow to maintain an acceptable scalability also for non conforming meshes up to thousands of cores.

The frequencyat which the PowerA2 sockets run is not very high (1.6GHz) but, as underlined in section 4, they have the capability of managing up to 4 processes/threads per core for a total amount of 64 processes/threads per node.

Using this feature,which is exploitable only with the hybrid version,with 4 OpenMP threads per coreit’s possible to run up to 5.26 times faster respect to the pure MPI one reserving the same number of cores,demonstrating also the good threads management of the multithreaded PowerA2 architecture.

As underlined in section 4, in order to avoid the stall of simulations over 1024 cores, the pure MPI tests were conducted with the optimized I/O routine. The same tests, even if feasible with the old original I/O version, would have been orders of magnitude slower.

In section 4.3 a scalability test of the hybrid version was performed on a structured mesh on the earthquake that struck the Po Plain, Emilia-Romagna, North-eastern Italy. Because of the memory required by this simulation, only 4 MPI processes per node could be used.

As shown in Fig. 9 the efficiency of the “time loop” phase shows a growing super-linear effect due to the increasinglybetter fit of the loops on the subdomains inside the processor’s caches. Also the “set-up” phase is performing ideally up to 8192 cores. At 16384 cores the scalability of this phase is compromised by the MPI communication among processors needed to build the map of the communicating faces.

Looking at the whole simulation, the speed-up of SPEEDis near to ideal values up to 16384 cores and a good efficiency is expected even fora larger number of cores, confirming that the optimized hybrid version allow managing scenarios intractable by the original pure MPI one.

6. Conclusions

As shown in fig. 9, the high parallel efficiency of the new hybrid version of SPEED, gives the possibility to run simultaneously parametric simulations on a Tier-0 machine, dramatically reducing the execution time and making such deterministic simulations feasible, producing reliable “real-time” results. Moreover, the complexity of the models considered makes these deterministic simulations unaffordable not only on a single processor machine but also on Tier-1 clusters.

In the near future the results of 3D ground-motion simulations obtained with SPEED, will be used to improve the physical reliability of the numerical scenarios; this will allow, on one side, to estimate more precisely the seismic hazard (i.e.: the spatial correlation of ground motion), on the other side, to understand which physical parameters (i.e.: directivity, slip pattern) play a crucial role in terms of potential damages to the infrastructures.

7. Acknowledgements

This work was financially supported by the PRACE-2IP project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI- 283493**.** The performance results were achieved using the PRACE Research Infrastructure resources Fermi, CINECA, Italy.

8. References

1. Cornell C.A. Engineering Seismic Risk Analysis. "Bulletin of the Seismological Society of America", 58(6):1583-1606 (1968).
2. Boroschek R., Contreras V., Kwak D. Y. and Stewart J. P. 2012. *Strong Ground Motion Attributes of the 2010 Mw 8.8 Maule, Chile, Earthquake.* Earthquake Spectra, **28(1):** 19–38.
3. Stupazzini M., Mazzieri I. et al. SPEED project : http://mox.polimi.it/it/progetti/speed/
4. CUBIT Toolkit : http://cubit.sandia.gov
5. METIS library : http://people.sc.fsu.edu/~jburkardt/c_src/metis/metis.html
6. CINECA, Fermi User Guide. http://www.hpc.cineca.it/content/ibm-fermi-user-guide
7. TAU Performance System® analyzer. http://www.cs.uoregon.edu/research/tau/home.php
8. IBM System Blue Gene Solution: Blue Gene/Q Application Development, chapter 3.
9. Day S.M., Bradley C.R. 2001. Memory-efficient simulation of anelastic wave propagation. Bulletin of theSeismological Society of America, 91(3): 520–531.