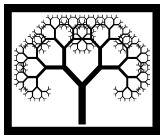
Paper 4



Proceedings of the Eighth International Conference
on Engineering Computational Technology,
B.H.V. Topping, (Editor),
Civil-Comp Press, Stirlingshire, Scotland

Load Balancing for Mesh Based Multi-Physics Simulations in the Arcane Framework

C. Chevalier, G. Grospellier, F. Ledoux and J.C. Weill
CEA, DAM, DIF
Arpajon, France

Abstract

Nowadays, industrial applications frequently need to deal with large and complex numerical simulations. To run, they require the use of large computers, such as TERA-100 at CEA DAM. To efficiently exploit such architectures, be able to have a correct load balance of the computations is a key issue, as the speed of the overall computation is very often given by the speed of the slowest process. This paper presents load balancing issues for large multi-physics simulations, and specially it is focused on engineering aspects. Using the ARCANE framework, we show how to make an efficient use of state-of-the-art partitioning tools. We describe our methodology and we also discuss what improvements are needed on the path towards peta-scale and exa-scale computing.

Keywords: partitioning, load balancing, multi-criteria, multi-physics, distributed memory, high performance computing.

1 Introduction

At CEA DAM, the French research institute on nuclear technologies, a particular effort in developing large scale complex numerical simulations is driven since the end of the 90s by The Simulation Program [1]. One goal of this project consists in designing and in implementing codes that simulate complex physics phenomena related, for example, to laser experiments. Mostly often, these phenomena are lying over several kinds of physics, and therefore codes have been designed to deal with multiple physics at the same time. Moreover, for such large simulations, very large computers such as TERA-100 have to be used efficiently.

On this kind of computer, simulations have to handle distributed memory on large number of processors. To efficiently perform this is a difficult task. This can be

addressed by using development framework such as ARCANE [2]. The latter manages all technical aspects of a simulation code while ensuring a high performance over thousands of cores. To achieve this goal, one of the key problems to solve is how to distribute data on the machine.

This paper presents work and results we have done solving this distribution problem in the ARCANE framework for multi-physics mesh based simulations. However, most of the remarks are generic enough to be valid and relevant for other frameworks.

First, we quickly describe the context of this work, then we show what are the classical ways to theoretically address this distribution issue. Afterwards we describe how we can implement them inside the ARCANE framework. Finally, we provide some results and we discuss about the current limits of state-of-the-art models and implementations.

2 Multi-Physics Code Design with Arcane

In this work, we consider complex mesh-based simulations of hydrodynamics phenomena with particle interactions. From a computational point of view, it means that we have to take into account several constraints. First, physical domains are discretized using 2D and 3D unstructured meshes having any type of cells (*i.e.* Triangles, quadrilaterals, polygons, tetrahedrons, hexaedrons, prisms, polyhedrons.). Secondly, hydrodynamics computations yield in each mesh cell (noted e). They are generally computed locally to cell e or its close neighbourhood (*e.g.*, in 2D, cells sharing an edge with e). Moreover, we have to deal with different kinds of particles that live in the mesh. Each particle arises, moves and dies somewhere in the mesh and not locally to a cell. In other words, the location of particles changes during the simulation process. Finally, the different physics that we have to handle are generally not localised in specific area of the physical domain, meaning that some parts of the mesh get involved in several different computational phases.

Writing a simulation code handling such specifications in distributed memory context is a difficult task. Thus, in order to ease the development of multi-physics simulation codes, the ARCANE framework has been developed at the CEA for a decade (see [2]). ARCANE can be seen as a component-based architecture where an ARCANE component is called a *service* that fulfills a *contract*. The contract defines an expected behaviour, and each contracting service has to provide an implementation of this behaviour. ARCANE framework is C++/C# written and it is aimed at managing all technical aspects (*i.e.* Mesh management, memory management through ARCANE variables, input/output, parallelism (MPI, threads), *etc.*), while ensuring high computing performances of numerical codes on clusters over 10000 cores. Another important goal is to speed-up the development process by providing a set of tools to build, debug, verify and validate numerical codes.

In order to build a multi-physics simulation code with ARCANE, the developer has to build *modules* specific to each physics. Technically, a module is independent

from the others: it can use ARCANE services, but not another module. To exchange data, modules share physical variables. The main principle of an ARCANE-based numerical simulation is to perform a time loop, where, during each time step, some physical modules are called in a specific order. Some ARCANE services can also be called to perform technical operations like synchronisation, load-balancing or output file generation. Figure 1 provides a simple example of a three physics simulation code using the ARCANE framework.

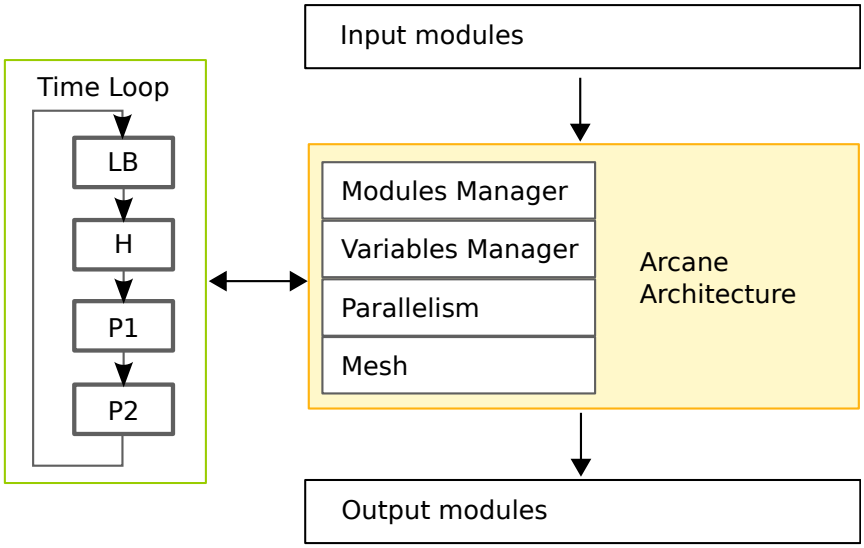


Figure 1: Multi-physics simulation code based on ARCANE framework. Four modules are used: three physics (H, P1 and P2) and one for the load balancing (LB).

For our concerns, the ARCANE service dedicated to load balancing might be called when computations become unbalanced. In order to illustrate the usage of load balancing in a multi-physics simulation, let us consider Figure 2 where six time steps of a simulation involving three physical modules are depicted. Module (H) is related to hydrodynamics while modules P1 and P2 are related to particles (typically Monte Carlo steps). One iteration of multi-physics simulations is generally divided into multiple steps (here, three) separated by synchronisations such as barriers. These synchronisations prevent the code from being handled like a mono-physics one: all processes have then to wait for the end of each phase before starting the next phase. To reduce the computational time, we can minimise the maximum time spent at each step, so we have to load balance each phase. A time representation of the three physics simulation presented in Figure 1 is shown in Figure 2. Load balancing happens at the beginning of iteration 4, and the following iterations are more efficient, as less computing resource is spoilt by process 0 (resp. 1) for P1 (resp. P2).

Visual explanation of load balancing is shown in Figure3. The first picture represents the mesh and its distribution on the two processes at iteration 1. At the beginning of iteration 4, particles have moved and now high concentration areas can be identified. Data distribution has to be changed in order to obtain better load imbalance.

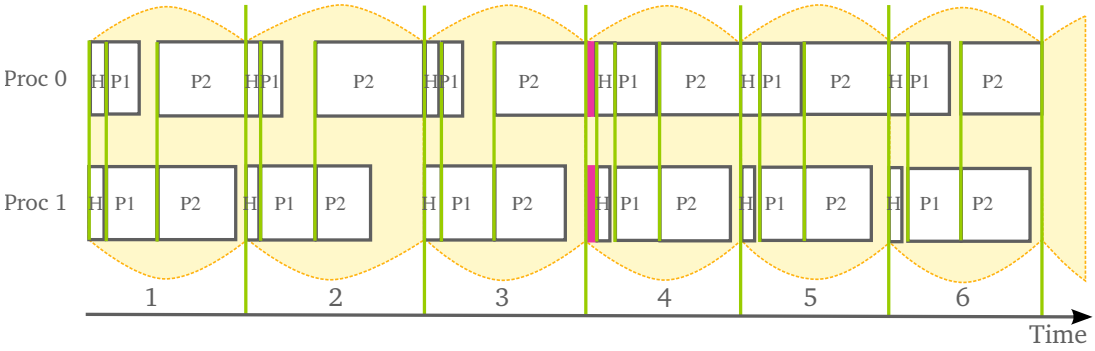
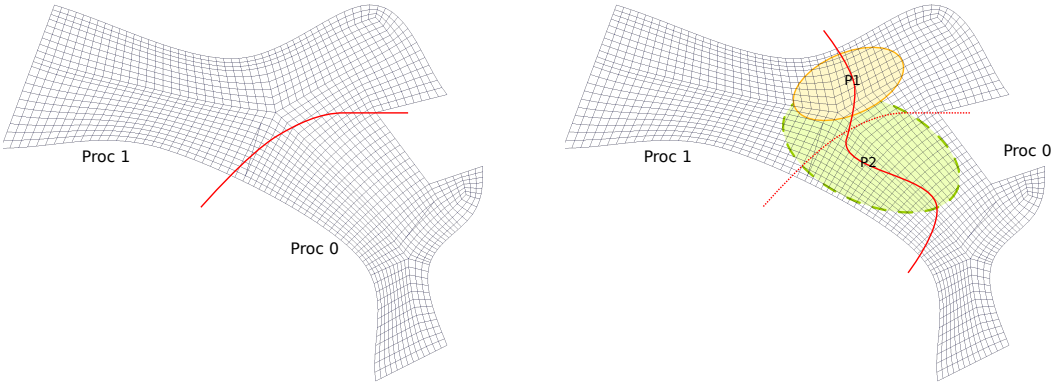


Figure 2: First iterations of a 3 physics simulation on 2 processes. Vertical bars correspond to synchronisation points. Load balancing occurs at the beginning of iteration 4.



(a) Computational domains at iteration 1. Particles are uniformly scattered across the mesh.

(b) Computational domains at iteration 4. Particles are now mostly localised in two areas. A new data distribution is computed in order to balance particles.

Figure 3: Illustration of a 3 steps multi-physics simulation running on two processes.

In the following, we do not explicitly give any details about physical phenomena and models. We have chosen to have a generic framework for load balancing, which requires only a computational point of view of the models, and we want to illustrate that it can be a reasonable choice.

3 Load Balancing and Mesh Based Simulations

This section summarises popular models developed to perform load balancing for mesh simulation. Technical implementation aspects will be discussed in section 4.

We consider simulations executed on large 2D or 3D unstructured meshes. Parallelism is achieved by distributing these meshes on the different processing units and by concurrently computing on these subdivisions. Designing a “good” distribution is

a problem that has been intensively discussed since the 80s. If we discard numerical stability aspects, the popular way to simplify and solve it is to perform load balancing. However Hendrickson [3] warns us that having a good load balance is only a requirement to achieve good performances, it is not sufficient.

The load balancing problem consists in distributing data in such a way that all the computing units have the same amount of work to do. Very often, it is solved using partitioning. In this context, a simulation can be described as a topological entity such as a graph where the graph vertexes correspond to the localisation of the computations, vertex weights represent the computational amount (or the memory) and edges between two vertexes mean that some dependencies exist between the two entities. In this model, distributing the data in an efficient manner over p processes can be computed by partitioning the corresponding graph in p parts, minimising load imbalance and edge-cut.

As mentioned earlier, our simulations are mesh based. In other words, the computational domain is represented by a mesh that can minimally be defined as a set of cells and a set of nodes. Moreover, in simulation codes using ARCANE, computational loops are mainly done over cells. We thus discard the nodal graph model, which consists in defining the graph from the mesh by identifying mesh nodes as graph vertexes and mesh edges as graph edges. Instead, we choose the most popular way to model a mesh as a graph, "*the dual graph representation*". In this model, for 3D meshes, graph vertexes correspond to mesh cells and graph edges correspond to mesh faces or mesh nodes. For instance, if data are exchanged between cells through faces, graph edges will correspond to mesh faces. On the opposite, if the simulation has a nodal formulation, it is better to define the interesting neighbourhood relationship as "two cells are neighbours *iff* they share a node".

As our simulations usually use faces, we consider the dual graph model defined by faces. This model can be extended from a graph to an hypergraph by keeping the definition of vertexes and associating to each vertex one hyperedge corresponding to all the neighbour cells. This provides a slightly richer model for partitioning, communication costs being evaluated accurately by the $\lambda - 1$ hypergraph partitioning metric [4]. Other models to represent mesh simulations as graphs or hypergraphs can be found (see [5]). However, in this paper, we keep the dual graph defined by faces or its hypergraph extension as topological models for load balancing. We discuss more precisely their practical usage in Section 4.

In Section 1, we mentioned that our goal was to provide load balance for multi-physics simulations. At this time, the chosen dual graph model does not handle any information about the multi phases essence of the simulation. The most natural way for the partitioner to find a good load balance for all the different phases of the simulation is to add one computational cost for each phase inside the graph. This can be done by giving vertex weight as a vector of scalars for which each component corresponds to a computing step [6]. By performing multi-criteria partitioning of this multi-weighted graph, we can obtain a distribution that balances computations for each phase. However, finding the best partition for a generic non weighted graph be-

ing already a NP-Hard problem (solved in practice only with heuristics), its extension to multi-weighted graphs can be more problematic to solve.

Therefore, other strategies have been developed, such as the proposal of Walshaw [7], which consists in doing several mono-criterion partitioning of reduced problems. Unfortunately this approach requires that computational domains of the different phases to be disjoint (at least for phases with particles.). As, in our simulations, it is not the case, we have to handle multi-physics differently.

Another approach to deal with some multi-physics simulations is known as domain replication [8, 9]. The mesh is distributed across MPI processes, but with some replicated parts. Indeed, mesh parts that support heavy particles computations are duplicated, and now it is particles that are partitioned between these processes. As we are also concerned about memory issues, we will not allow duplications of data, however, domain replication can theoretically be used conjointly with the developments presented in this paper. In fact, for small meshes, implementation in ARCANE is currently in progress.

4 Load Balancing Implementation in ARCANE

In this section, we will first quickly introduce the ARCANE goals in respect to data distribution. Afterwards, we will present the load balancing implementation inside ARCANE.

The ARCANE load balancing service deals with performance issues due to the distribution and can be called from ARCANE main loop whenever needed. It is only focused on using (hyper-)graph partitioning libraries and is designed to be a tool box to solve load balancing problems.

ARCANE load balancing module consists in three steps:

1. build a "partitioning object" (mostly often a graph) that models the simulation costs ;
2. partition this object according to its costs function ;
3. redistribute the ARCANE entities and variables according to this partitioning.

The first and the third steps are done by ARCANE, while the second is generally performed using one of the libraries described below.

The "partitioning object" is built accordingly to the models presented in section 3. These models are converted accordingly to the partitioning tool used at the second step, so the first and the last steps are pure ARCANE codes.

To summarise, data manipulations are directly performed by ARCANE whereas partitioning computation uses heuristics that are generally implemented in specialised third party libraries such as PARMETIS [10], SCOTCH [11] and ZOLTAN [12]. Results that will follow are obtained with PARMETIS version 3.1.1, SCOTCH 5.1.11 and ZOLTAN from Trilinos 10.6.

These libraries are supported in various ways, to be able to have the best adequacy to the different kinds of problems that can occur in ARCANE based simulation codes. All these libraries provide parallel partitioning for distributed graphs, relying on MPI for communication. They all use multi-level algorithms, and they are known to produce state-of-the-art results, in term of both quality and speed.

Both PARMETIS and SCOTCH are parallel graph partitioners, and in ARCANE, they both exploit the dual graph model introduced in Section 3. Thereby, without being misled by implementation artifacts, we can perform a better evaluation of the model. However, software capabilities differ quite a bit between the two. PARMETIS is able to compute repartitioning (in fact partitioning and remapping) as well as multi-criteria partitioning. SCOTCH, even if it is currently limited to mono criteria static partitioning, can take advantage of more clever heuristics such as diffusion algorithms [13] to improve the partition quality [14, 15, 16].

ZOLTAN provides other kinds of algorithms and models, like geometric and hypergraph partitioning [4] that we exploit in ARCANE. For geometric models, we partition a set of vertexes, which correspond to the cell mass centres, their weights being the same as in the dual graph or hypergraph models. We have led our experiments using RCB [17] algorithm as it allows us to use multi-weighted vertexes. Hypergraph model corresponds to the extension of dual graph model presented in Section 3, but does not allow multiple weights.

5 Realisations and Results

This section summarises quickly the load balancing issues that arise when dealing with dynamic multi-physics simulations. It also describes solutions we have retained.

5.1 Dynamic Load Balancing.

One key aspect of our simulations is the fact that they model dynamic phenomena, which can move in the computational domain, creating load imbalance during the simulation. In order to improve overall running time, ARCANE provides dynamic load balancing.

Dynamic load balancing is implemented directly within ARCANE as a (re)partition stage followed by data redistribution. Note that even if current version of SCOTCH deals only with partitioning, *i.e.* is primarily designed for static load balancing, it is possible to use it as a partitioner in a dynamic context, at the expense that data migration costs are not taken into account. For PARMETIS and ZOLTAN, repartitioning interfaces are used, allowing us to minimise the overall communication cost of both migration and simulation steps [18, 19].

Table 1 presents a comparison between static and dynamic partitioning for a dynamic simulation composed by one hydrodynamics phase and two phases computing on particles. These run-times are given for the beginning of the simulation (first five

Approach	2D test case			3D test case		
	MPI processes			MPI processes		
	128	256	512	128	256	512
`Static`	1,724	985	481	2,157	1,145	619
`PM-old`	1,102	†	535	†	†	†
`PM-new`	719	491	260	1,693	848	522

Table 1: Run-times (s) for multi-physics simulations involving hydrodynamics and two kinds of particles. † represents inability to compute, usually due to memory shortage.

hundred iterations).

Each mesh has one million cells, and twelve million particles of kind P1. 2D test case has ten million particles of kind P2 while 3D case has two hundred fifty million. PARMETIS was used as partitioner, enforcing multi-criteria for non static cases. Static distribution was obtained by splitting the mesh using serial METIS, which provides in this case better quality than PARMETIS. However, after few iterations, computations become highly unbalanced, some processes staying without any particle while the others are dealing with all the complexity. That is the reason why dynamic load balancing seems mandatory, and this phenomenon is amplified when continuing the computations. In the dynamic case, data redistribution can be computed every 10 iterations if the imbalance in running time between processes is greater than a threshold of 20%. As shown in Table 1, periodically computing load balance (methods `PM-old` and `PM-new`) greatly improves performances: here, up to a 2.4 speed-up on 128 processors.

5.2 Costs Evaluation for Dynamic Load Balancing.

Partitioning for dynamic load balancing requires to evaluate individual computational costs for each cell. In a perfect world, we should be able to predict such costs, but unfortunately, for general cases, we can just estimate them by taking into account what happened during the previous iterations.

In order to achieve a complete abstraction towards user's code, the most natural way is to use time measurements as weights for the partitioner. Easy to implement, allowing to use fully opaque physics codes, this was the approach chosen for the original version of ARCANE (like `PM-old`). Nevertheless an important drawback can be identified: it is very sensitive to foreign and side effects. Indeed, it is hard to obtain an accurate value of distribution layout costs by measuring the time spent into the simulation. As weight should be defined for each cell, the best way is to get measurement per cell, but unfortunately, system noise (communications, system load, contexts switching, ...) predominates at this level. Moreover, even the distribution can have side effects at cell level: for example, for a given cell, computational times can vary a lot depending on the previous computations (cache, super-scalar or pipeline

effects ...). Thus, without any instrumentation of physics code, it seems impossible to get an accurate evaluation of partitioning weights. It has been confirmed by some experiments. Especially, on platforms with highly congested networks, we observed that using timings gave us worst computational timings (and not only communication timings) than a more resource free version of the machine.

Therefore, as automatic evaluations of distribution related costs seem inappropriate, more flexibility towards the black box concept for physics modules within ARCANE is needed, allowing us to instrument modules that require load balancing. It is necessary to exhibit quantities that are only related to distribution (*i.e.* deterministic in respect to the data distribution) and that are meaningful regarding computational cost (timings).

Our simulations mainly involve two entities: cells and particles. Hydrodynamics step deals only with cells and its theoretical complexity in time only depends on the number of cells. Particles steps complexities rely on number of particles and potentially also on the number of cells (for example when particles are scanned by looking at the cells). We retain these criteria to characterise distribution costs for ARCANE simulations.

Pertinence of this costs characterisation is illustrated in Table 1 by strategy `PM-new`. `PM-old` uses time measurements information while `PM-new` corresponds to the new approach, using particles counts as characteristics. We can observe that comparing to the full black box approach (`PM-old`), code efficiency is higher (up to a speed-up of 2 on 512 processors). Moreover, the robustness is also improved, allowing us to compute harder instance (like the 3D problem) and to obtain more consistent performances from one run to another.

Therefore, with a not too intrusive instrumentation that simply evaluates the number of relevant particles for each phase, it is possible to give a proper characterisation of the problem to the load balancing algorithm. Table 2 confirms it. The instrumented simulation has a good scalability thanks to dynamic load balancing using SCOTCH. Moreover, we can observe that dynamically load balanced simulation on 512 processors run twice as fast as the static one on 1024 processors. Above, problem becomes over distributed but we can still observe some speed-up.

# MPI	Average computing time (s) per iteration								Time (s)	
	in $[0.0, 0.4]$		in $[0.4, 0.6]$		in $[0.6, 0.8]$		in $[0.8, 1.0]$		to $1 \times 10^{-7} s$	
512	3.9	*×1*	5.1	*×1*	7.3	*×1*	8.9	*×1*	6,610	*×1*
1024 nr	2.2	*×1.8*	8.3	*×0.6*	17.8	*×0.4*	41.8	*×0.2*	12,814	*×0.52*
1024	2.2	*×1.8*	3.5	*×1.5*	4.4	*×1.7*	6.1	*×1.5*	3,559	*×1.86*
2048	1.5	*×2.6*	2.2	*×2.3*	3.5	*×2.1*	4.6	*×1.9*	2,508	*×2.64*
4096	1.0	*×3.9*	1.6	*×3.2*	2.0	*×3.7*	2.9	*×3.1*	2,220	*×2.98*

Table 2: Run-times (s) for a particle simulation (6.88 million cells) load balanced by SCOTCH. Average computing time per iteration is relative to CPU time spent in physics modules (load balancing cost is not taken into account). All cases but *"1024 nr"* use dynamic load balancing.

5.3 Multi-physics and Load Balancing.

Now that a correct quantification of computing costs in each physics module individually is given, we can focus on how to handle different physics and associated costs at the same time.

With some partitioners, we can directly exploit the multi-criteria model introduced in Section 3. In particular, PARMETIS provides multi-criteria partitioning for graphs, while ZOLTAN supports it only for geometric. Besides, ZOLTAN approach cannot insure to meet balance for all criteria. It currently tries to find a contiguous area by doing recursive coordinates bisections, and in our simulations, we observe that it was not always possible to satisfy all balance constraints this way.

To not only rely on PARMETIS and thus to avoid pathological cases (implementation or heuristics problems triggered by "bad" instances, ...), we still want to use SCOTCH or ZOLTAN at least as alternatives. As neither the multi-criteria nor Walshaw's approaches can be used, another model, exploiting mono-criteria partitioning, is needed.

A classical way to convert multi-weights vectors to scalar weights is to compute a linear combination of vector components. Withal, coefficients have to be carefully chosen and have to be closely related to the simulation case to allow this approach to produce decent results. Dependency towards the case specificity being essential, this linear combination approach cannot really be used in an automatic way, so it cannot be used in ARCANE.

The main issue in the previously described approach is that combination occurs on different kinds of quantities have thus to use a quantity which is relevant for each single module, and still significant outside the module. The memory consumption fulfills these requirements. Besides, it is a significant characteristic for data distribution quality. ARCANE modules use ARCANE variables to handle data, so the framework is able to sharply evaluate the allocated memory, even for one single cell (and its associated particles).

We thus have two different approaches to deal with multi-physics simulations: the first one works on balancing computational times through multi-criteria partitioning, the latter one deals with memory load balancing through mono-criterion partitioning.

Figure 4 shows that multi-criteria (`PM-*` and `ZOLTAN-RCB`) partitioners outperform mono-criterion approaches for a multi-physics simulation, even at relatively small scale. Results also confirm that instrumenting the code helps a lot to improve performances, differences between `PM-new` and `PM-old` speaking for themselves.

6 Discussion

Previous experiments showed us that it is possible, using state-of-the-art partitioning softwares, to achieve correct load balancing, even for complex and dynamic numerical simulations. In this section we will discuss what results can be expected from today

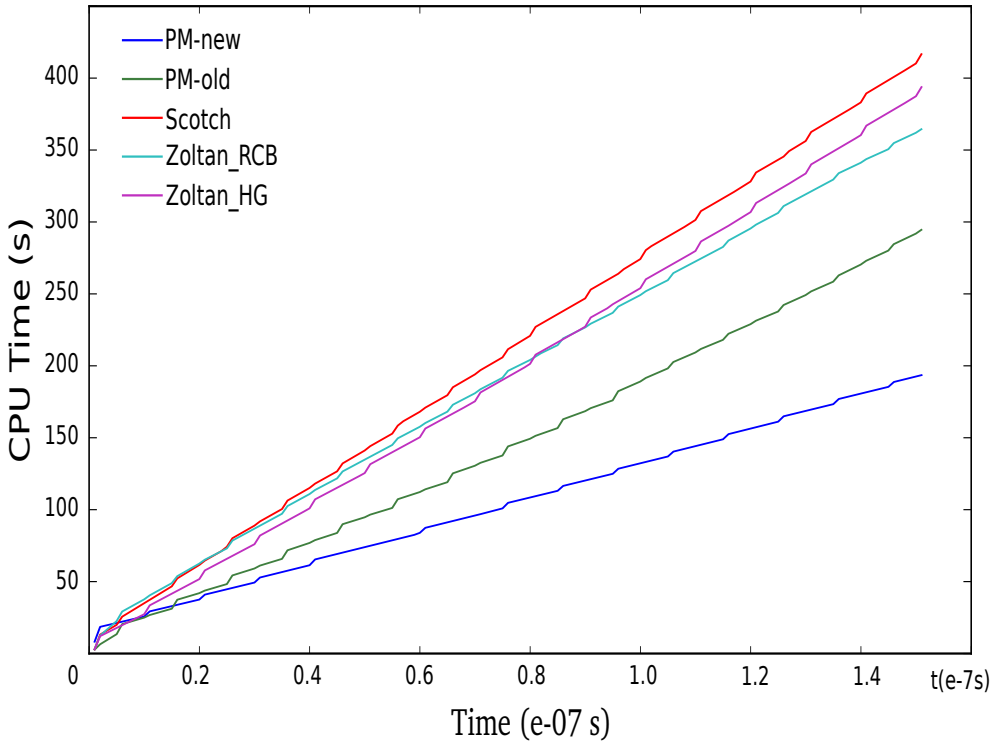


Figure 4: Run-time (s) for a simulation on 64 processors with hydrodynamics and two particles phases. Multi-criteria methods (`PM-old`, `PM-new` and even `ZOLTAN-RCB`) perform better than mono-criteria methods. `PM-old` uses time measurements while all other methods use particles counts.

partitioning tools and then we will present problems that still have to be addressed for exa-scale computations of this kind.

6.1 Observations

We have underlined that using correctly partitioning libraries is not trivial, and that the knowledge of pertinent quantities for describing simulation codes is essential. In the context of a development framework such as ARCANE, it implies that physics modules has to submit relevant data to the load balancing module. External measurements of physics codes (*i.e.* timings) do not provide accurate enough information for computing a good partitioning, so counters for load balancing have to be placed inside the modules. With such an information,the framework is able to automatically perform load balancing.

Once that partitioning tools were fed by the adequate characteristics of the distribution, our experiments brought into light that there is no perfect partitioning software for large multi-physics load balancing in production codes. PARMETIS results are quite interesting, specially because they satisfy multi-criteria balance, but the lack of reliability above few hundreds processes [20, 21] is problematic for a generic frame-

work dedicated to large scale production simulations.

Oppositely, SCOTCH and ZOLTAN insure a good robustness, however they are quite limited by enforcing only scalar weights balance for topological models. ZOLTAN's geometric multi-criteria capabilities are interesting, however the impossibility to obtain non-contiguous parts limits its interest for us.

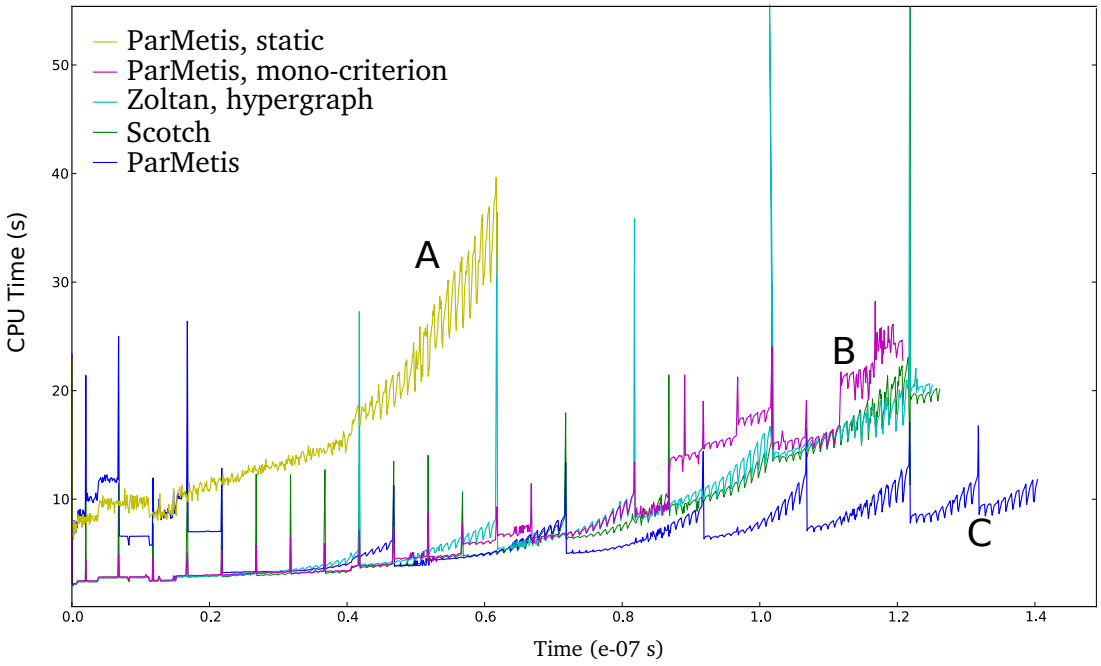


Figure 5: Time (s) per iteration for a 128 MPI processes simulation with hydrodynamics and particles on a 1.72 million cells mesh.

Figure 5 shows results obtained on 128 processors for a dynamic multi-physics simulation with one hydrodynamic phase and one particle one. It summarises the observations we previously made, namely that dynamic load balancing is required, and that multi-criteria partitioning provides better results than mono-criterion ones. To describe in more details, high amplitude peaks correspond to repartitioning, while high frequency oscillations correspond to particle renumbering.

- (A) illustrates that static load balancing is clearly outperformed by any dynamic approach.
- (B) shows that mono-criterion partitioners provide almost similar results. However, PARMETIS (version 3.1.1) has an unpredictable behaviour, due to the 32 bits limit for vertex weight. Since version 4.0, PARMETIS can use a 64 bits weight representation and mono-criterion PARMETIS gives results close to those of ZOLTAN or SCOTCH.
- (C) Multi-criteria partitioning with PARMETIS is the best approach for this case, keeping iteration cost low. We can also observe that particles numbering seems to have less performance impact when using SCOTCH partitioning.

On this example, one can notice that performing load balancing takes a lot of time, and one explanation can be that the graphs are over-distributed across the processes. Indeed, graph partitioning is not intensive in term of computations and generally requires in distributed memory a lot of communications. Moreover, only a small number of information is associated to the graph (weights and connectivity) and there is no point to keep the same distribution of vertexes as the cells of the mesh. In ARCANE, we now reduce the number of processes that really compute the partitioning by using only one MPI process by computational node. On TERA-100, we can divide the number of processes for partitioning by up to 32 and therefore we increase the density of data as well as the computational efficiency of partitioning tools (and also very often the quality of their results). To avoid graph centralisation over-costs, we use this technique when simulation is spread on more than five hundred MPI processes.

6.2 Open Problems

Whether several improvements can be done in software implementations, a lot of concepts also have to be developed for modelling dynamic multi-physics partitioning problems. Indeed, it appears that these problems need slightly different approaches to be solved more efficiently. For example, plenty of interesting works about limiting communications during simulations or even during data redistribution [18, 19, 22] can be found, but in our case of intensive computations simulations, it is not really the main issue. SCOTCH performing only partitioning provides as good results as ZOLTAN using a much more richer repartitioning approach, data communication being cheap comparing to overall computational cost.

On the other hand, some particularities of mesh partitioning have been ignored, in particular the fact that we can use geometry, or, even more important, the distribution impact on mesh data structure. Most of simulation codes exploit ghost layers, duplicating foreign boundary cells on local processes. These cells have a special behaviour inside the code, but still consume memory or computing time and are usually ignored by standard partitioning models. New models, specific to mesh load balancing, have to be developed as impact of data distribution becomes more and more important as the number of processors is increasing.

Partitioning of graphs issued from meshes was for a long time considered as an easier problem than on more generic graphs [23] (from linear algebra, VLSI design, *etc.*). But, in fact, they exhibit characteristics which are not properly handled by current heuristics or implementations. If specific and powerful algorithms have been developed and studied for complex topologies such as power law graphs or social network graphs; simulation graphs are, on the opposite, usually very simple in terms of connectivity but with a very heterogeneous weight distribution. In parallel distributed context, we have observed, that softwares have trouble to meet imbalance criterion on such graphs, even if solutions exist.

7 Conclusion

This paper mainly illustrates that, once again, load balancing is a key issue for enabling efficient parallel and distributed simulations.

We also found that implementing an efficient generic load balancing capability for a simulation development framework such as ARCANE is not a trivial task, even when using well-known models and popular partitioning libraries. We have illustrated that in the context of a generic framework for simulations, instrumenting code is required. As shown in Section 4, this instrumentation results in faster (Figures 4 and 5) and more robust (Table 1) simulations, with all the partitioning libraries. Moreover, for multi-physics simulations, multi-criteria partitioning appears to be a requirement, providing much better results than mono-criterion approaches.

However, we also pointed out that with state-of-the-art tools we can, at best, obtain decent results for medium-sized simulations. For peta-scale computing, current softwares lack functionalities or robustness. Limitations can be related to the model from a data point of view, such as the inability to take ghost cells into account, specially annoying at large scale, essentially because in 3D, interface sizes grow faster than the problem. But, limitations might also be architecture related, in a close future memory hierarchy will definitely have to be taken into account: at the framework level by considering new programming paradigms such as MPI-Threads, or at partitioner level by performing hierarchical partitioning or static (or "dynamic", *i.e.* in simulation) mapping.

References

[1] "The Simulation Program", `http://www.cea.fr/english_portal/defense/the_simulation_program`.

[2] G. Grospellier, B. Lelandais, "The Arcane development framework", in *Proceedings of the 8th workshop on Parallel/High-Performance Object-Oriented Scientific Computing*, POOSC '09, pages 4:1–4:11. ACM, New York, NY, USA, 2009, ISBN 978-1-60558-547-5.

[3] B. Hendrickson, T.G. Kolda, "Graph partitioning models for parallel computing", *Parallel Computing*, 26: 1519 – 1534, 2000.

[4] K. Devine, E. Boman, R. Heaphy, R. Bisseling, U. Çatalyürek, "Parallel Hypergraph Partitioning for Scientific Computing", in *Proc. IPDPS'06*. IEEE, 2006.

[5] E.G. Boman, Ü. Çatalyürek, C. Chevalier, K.D. Devine, *Combinatorial Scientific Computing*, Chapter Parallel Partitioning, Coloring and Ordering in Scientific Computing, pages 351–371, Computational Science Series. CRC Press, 2011.

[6] G. Karypis, V. Kumar, "Multilevel Algorithms for Multi-Constraint Graph Partitioning", in *Proc. Supercomputing*, 1998.

[7] C. Walshaw, M. Cross, K. McManus, "Multiphase Mesh Partitioning", *Appl. Math. Modelling*, 25(2): 123–140, 2000.

[8] H.J. Alme, G.H. Rodrigue, G.B. Zimmerman, "Domain Decomposition Models for Parallel Monte Carlo Transport", *The Journal of Supercomputing*, 18: 5–23, 2001, ISSN 0920-8542, URL http://dx.doi.org/10.1023/A:1008196906753, 10.1023/A:1008196906753.

[9] R.J. Procassini, M.J. O'Brien, J.M. Taylor, "Load Balancing of Parallel Monte Carlo Transport Calculations", in *International Topical Meeting on Mathematics and Computations*, 2005.

[10] G. Karypis, K. Schloegel, V. Kumar, "ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 3.1", Technical report, Dept. Computer Science, University of Minnesota, 2003.

[11] F. Pelligrini, "PT-SCOTCH 5.1 User's Guide", Research rep., LaBRI, 2008.

[12] E. Boman, K. Devine, R. Heaphy, B. Hendrickson, V. Leung, L.A. Riesen, C. Vaughan, Ü. Çatalyürek, D. Bozdag, W. Mitchell, J. Teresco, *Zoltan 3.0: Parallel Partitioning, Load Balancing, and Data-Management Services; User's Guide*, Sandia National Labs, Albuquerque, NM, 2007, Tech. Report SAND2007-4748W.

[13] F. Pellegrini, "A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries", in *Proc. Euro-Par'07*, Volume 4641 of *LNCS*, pages 191–200. Springer, Aug. 2007.

[14] C. Chevalier, *Conception et mise en œuvre d'outils efficaces pour le partitionnement et la distribution parallèle de problèmes numériques de très grande taille*, PhD thesis, Université Bordeaux I, 2007.

[15] C. Chevalier, F. Pellegrini, "PT-SCOTCH: A tool for efficient parallel graph ordering", *Parallel Computing*, 34(6–8): 318–331, 2008.

[16] J.H. Her, F. Pellegrini, "Efficient and scalable parallel graph partitioning", *Parallel Computing*, 2009.

[17] M.J. Berger, S.H. Bokhari, "A partitioning strategy for nonuniform problems on multiprocessors", *IEEE Trans. Computers*, C-36(5): 570–580, 1987.

[18] U. Çatalyürek, E. Boman, K. Devine, D. Bozdag, R. Heaphy, L. Riesen, "Hypergraph-Based Dynamic Load Balancing for Adaptive Scientific Computations", in *Proc. IPDPS'07*. IEEE, 2007.

[19] K. Schloegel, G. Karypis, V. Kumar, "Parallel static and dynamic multiconstraint graph partitioning", *Concurrency and Computation – Practice and Experience*, 14(3): 219–240, 2002.

[20] C. Bekas, A. Curioni, P. Arbenz, C. Flaig, G.H. Van Lenthe, R. Müller, A.J. Wirth, "Extreme scalability challenges in micro-finite element simulations of human bone", *Concurr. Comput. : Pract. Exper.*, 22: 2282–2296, November 2010, ISSN 1532-0626, URL http://dx.doi.org/10.1002/cpe.v22:16.

[21] A. Turk, C. Aykanat, V. Vehbi Demerci, S. von Alfthan, I. Honkonen, "Investigation of load balancing scalability in space plasma simulations", Technical report, PRACE Partnership for Advanced Computing in Europe, 2012.

[22] C. Walshaw, M. Cross, M. Everett, "Mesh partitioning and load-balancing for distributed memory parallel systems", in *Proc. Parallel & Distributed Comput-*

ing for Computational Mechanics, Lochinver, Scotland, 1998.

[23] C. Chevalier, I. Safro, "Comparison of coarsening schemes for the multilevel graph partitioning", in *Proc. LION'3*. Trento, Italy, Jan. 2009.