

Using Simulation to Evaluate and Tune the Performance of Dynamic Load Balancing of an Over-decomposed Geophysics Application

Rafael Keller Tesser, Lucas Mello Schnorr, Arnaud Legrand, Fabrice Dupros, Philippe O A Navaux

▶ **To cite this version:**

Rafael Keller Tesser, Lucas Mello Schnorr, Arnaud Legrand, Fabrice Dupros, Philippe O A Navaux. Using Simulation to Evaluate and Tune the Performance of Dynamic Load Balancing of an Over-decomposed Geophysics Application. [Research Report] RR-9031, INRIA Grenoble - Rhone-Alpes. 2017, pp.21. hal-01391401v2

HAL Id: hal-01391401
https://inria.hal.science/hal-01391401v2

Submitted on 16 Feb 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

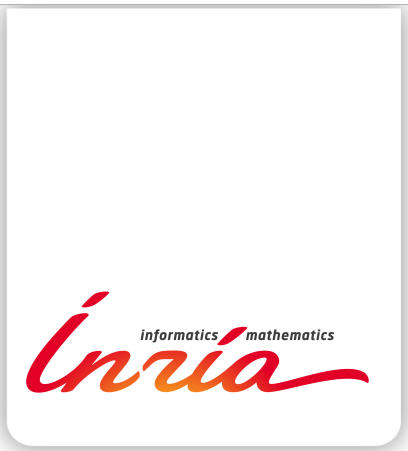


Using Simulation to Evaluate and Tune the Performance of Dynamic Load Balancing of an Over-decomposed Geophysics Application

Rafael Keller Tesser, Lucas Mello Schnorr, Arnaud Legrand, Fabrice Dupros, Philippe O. A. Navaux

RESEARCH REPORT
N° 9031
February 2017
Project-Team POLARIS

ISSN 0249-6399 ISRN INRIA/RR--9031--FR+ENG



Using Simulation to Evaluate and Tune the Performance of Dynamic Load Balancing of an Over-decomposed Geophysics Application

Rafael Keller Tesser*, Lucas Mello Schnorr*, Arnaud Legrand†, Fabrice Dupros‡, Philippe O. A. Navaux*

Project-Team POLARIS

Research Report n° 9031 — February 2017 — 21 pages

* Informatics Institute, Federal University of Rio Grande do Sul, Porto Alegre, Brazil
† Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, F-38000 Grenoble, France
‡ BRGM, Orléans, France

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Abstract: Finite difference methods are, in general, well suited to execution on parallel machines and are thus commonplace in High Performance Computing. Yet, despite their apparent regularity, they often exhibit load imbalance that damages their efficiency. In this article, we first characterize the spatial and temporal load imbalance of Ondes3D, a seismic wave propagation simulator used to conduct regional scale risk assessment. Our analysis reveals that this imbalance originates from the structure of the input data and from low-level CPU optimizations. Such dynamic imbalance should, therefore, be quite common and can not be solved by any static approach or classical code reorganization. An effective solution for such scenarios, incurring minimal code modification, is to use AMPI/CHARM++. By over-decomposing the application, the CHARM++ runtime can dynamically rebalance the load by migrating data and computation at the granularity of an MPI rank. We show that this approach is effective to balance the spatial/temporal dynamic load of the application, thus drastically reducing its execution time. However, this approach requires a careful selection of the load balancing algorithm, its activation frequency, and of the over-decomposition level. These choices are, unfortunately, quite dependent on application structure and platform characteristics. (i.e., number of processors and their speed; network topology, bandwidth, latency). Therefore, we propose a methodology that leverages the capabilities of the SimGrid simulation framework and allows to conduct such study at low experimental cost. Our approach relies on a combination of emulation, simulation, and application modeling that requires minimal code modification and yet manages to capture both spatial and temporal load imbalance and to faithfully predict the performance of dynamic load balancing. We evaluate the quality of our simulation by systematically comparing simulation results with the outcome of real executions and demonstrate how this approach can be used to quickly find the optimal load balancing configuration for a given application/hardware configuration.

Key-words: Geophysics application, Load balancing, Simulation, SimGrid, MPI

Utilisation de la simulation pour évaluer et optimiser les performances d'une application de géophysique à l'aide de surdécomposition et d'équilibrage de charge dynamique

Résumé : Les méthodes aux différences finies sont en général bien adaptées au machines parallèles et donc assez courantes dans le domaine du calcul à haute performance. Pourtant, en dépit de leurs apparentes régularités, il n'est pas rare qu'elles souffrent d'un déséquilibre de charge dommageable. Dans cet article, nous commençons par caractériser le déséquilibre de charge spatial et temporel d'Ondes3D, une application de simulation de propagation d'ondes sismiques utilisée pour faire de l'évaluation de risque sismique à l'échelle régionale. Notre analyse révèle que ce déséquilibre provient de la nature même des données d'entrées et d'optimisations bas niveau du CPU. Ce type de déséquilibre dynamique est donc a priori relativement courant et ne peut être résolu par des approches statiques ou par des réorganisations de code classiques. Une approche pragmatique et ne nécessitant que des modifications mineures du code consiste à utiliser AMPI/CHARM++. En sur-décomposant l'application, le runtime CHARM++ peut rééquilibrer dynamiquement la charge en migrant les données et les calculs à la granularité du processus MPI. Nous montrons que cette approche permet effectivement de résoudre le problème de déséquilibre spatial et temporel de charge et ainsi de réduire drastiquement le temps d'exécution total. Cependant, cette approche nécessite a priori une sélection minutieuse de l'algorithme d'équilibrage de charge, de la fréquence d'activation ou du niveau de sur-décomposition. Ces choix sont hélas en général assez dépendants de la structure de l'application et des caractéristiques de la plate-forme (i.e., le nombre de processeurs et leur vitesse, la topologie et la vitesse du réseau). Nous proposons donc une méthodologie se basant sur l'environnement de simulation SimGrid et permettant de réaliser ce type d'étude à faible coût. Notre approche repose sur une combinaison d'émulation, de simulation et de modélisation d'application qui ne nécessite que des modifications minimes du code d'origine et permet à la fois de capturer le déséquilibre spatial et temporel et de prédire de façon fiable les performances de l'équilibrage de charge. Nous évaluons la qualité de notre proposition en comparant de façon systématique les résultats de notre simulation avec ceux d'expériences réelles. Nous montrons ensuite comment cette approche peut être utilisée pour déterminer rapidement les paramètres optimaux d'équilibrage de charge pour une configuration applicative/matérielle donnée.

Mots-clés : Application de Géophysique, Équilibrage de charge, Simulation, SimGrid, MPI

1 Introduction

The Ondes3D seismic wave propagation simulator [1], developed by computational science researchers at the French Geological and Mining Research Bureau (BRGM), is a typical iterative application tailored for homogeneous HPC platforms. Unfortunately like many other similar applications, Ondes3D suffers from scalability issues [2] due to the difficulty of evenly distributing the computational load among processes. One of the contributions of this article is to demonstrate that, despite the regularity of the finite difference method kernels it relies on, it presents both spatial and temporal load imbalance.

The performance of Ondes3D could be improved by rewriting it to run on modern heterogeneous HPC platforms. Such process is, however, quite difficult and time consuming. Although there are efforts [3] focusing on parts of the application, this usually leads to an almost complete code transformation. The unwanted side-effect is that computational science researchers, the people that actually understand the physics behind the code, are often unable to contribute anymore.

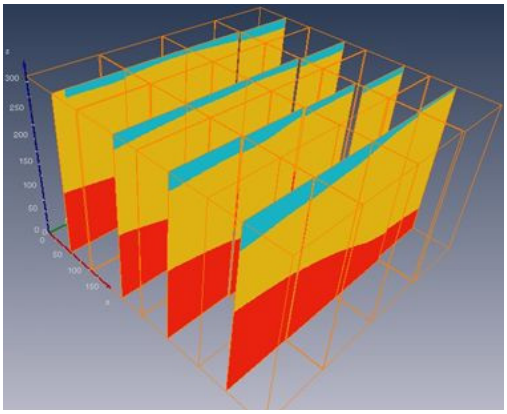
Another possible way to improve performance with minimal code modification is to rely on domain over-decomposition and runtimes that support dynamic process migration, as implemented by Charm++ [4]. In the specific case of legacy iterative MPI applications, one may employ Adaptive MPI (AMPI) [5, 6], which is a full-fledged MPI implementation built over the Charm++ runtime and benefits from its load balancing infrastructure. AMPI encapsulates each MPI rank in a virtual processor that can be dynamically migrated when necessary. The migration phase is triggered when the non-standard `MPI_Migrate` operation is called by the application. In this phase, the load balancer to decides the new process/resource mapping, based on load measurements taken during the last observation period. The direct benefit of this approach, which has already been applied to Ondes3D in a previous work [7], is that any spatio-temporal load imbalance is dynamically mitigated, effectively improving performance.

Discovering beforehand how much performance gain can be obtained using adaptive HPC runtimes is fundamental, but usually difficult to evaluate without considerable effort. Finding the best configuration of AMPI involves conducting real experiments at scale to identify the best (a) over-decomposition level, (b) load balancing heuristic, (c) load balancing frequency, and (d) number of resources to request. These parameters depend directly on the platform configuration, and tuning them is very resource and time-consuming.

In this paper, we propose a simulation-based methodology to evaluate the potential performance benefits brought by adaptive MPI runtimes to legacy MPI codes. This methodology accelerates the evaluation of over-decomposition coupled with dynamic load balancing with almost no modification of the target MPI application. In our approach, instead of running real-world experiments, we rely on SimGrid's SMPI emulation [8, 9] and trace replay mechanisms [10] to simulate the computation and communication behavior of the application and to mimic the behavior of the load balancing heuristics. We show that our methodology is faithful, in terms of total makespan as well as from the load balancing perspective. The application has to be executed only once to obtain a fine-grain trace that can then be replayed multiple times to evaluate the best parameter configuration for a given HPC platform. Since the replay is fast (usually less than a minute on a laptop), it enables a quick inspection of many load balancing parameters.

Although we validate our methodology using a single real-life application (Ondes3D) and two earthquake scenarios (Chuetsu-Oki and Ligurian), we believe that our methodology has nothing specific to it. Our strategy could be applied to any iterative domain based MPI application.

Section 2 presents a detailed analysis of the spatial and temporal load imbalances in Ondes3D. Section 3 presents AMPI and difficulties of the load balancing evaluation process. Section 4 details our proposal, our evaluation workflow and its validation procedure. In Section 5, we



(a) 3D view of the heterogeneous rock medium, with a 4×4 domain decomposition; each process calculates a cuboid.

```
for (ts = 0; ts < N; ts++) {
  Intermediates();

  Stress();
  //Intertwined Asynchronous
  //Neighborhood Communication

  Velocity();
  //Intertwined Asynchronous
  //Neighborhood Communication
}
```

(b) The main loop with three kernels: `Intermediates`, `Stress` and `Velocity`; no global synchronization.

```
static inline double CPML4 (double vp, double dump, double alpha,
                    double kappa, double phidum, double dx, double dt,
                    double x1, double x2, double x3, double x4) {
  double a, b;
  b = exp(-(vp * dump / kappa + alpha) * dt);
  a = 0.0;
  if (abs(vp * dump) > 0.000001)
    a = vp * dump * (b - 1.0) / (kappa * (vp * dump + kappa * alpha));
  return b * phidum + a * ((9. / 8.) * (x2 - x1) / dx -
         (1. / 24.) * (x4 - x3) / dx);
}
```

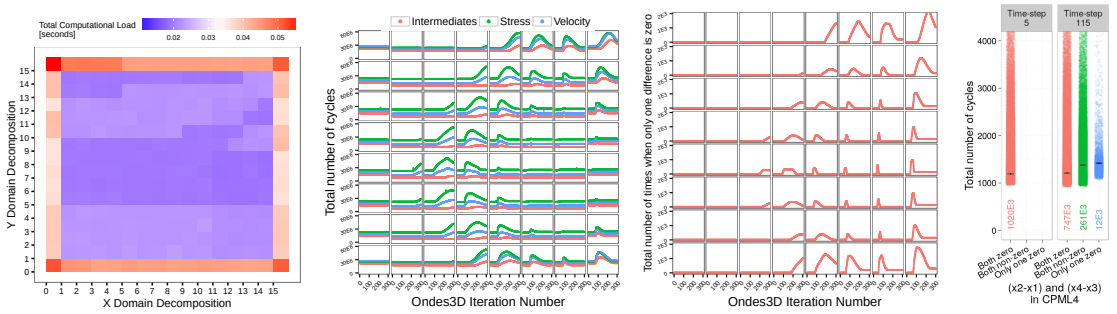
(c) The `CPML4` kernel, called nine times for each cell i, j, k in the `Intermediates` kernel when processing a sub-domain. Variables `x1`, `x2`, `x3`, and `x4` represent the rock medium states (e.g., speed) that evolves along the iterations.

Figure 1: The Ondes3D application: (a) an example of domain decomposition for 16 processes; (b) the three large kernels of the main loop, with intertwined neighborhood communications; (c) a detailed view of the small `CPML4` kernel (out of 24).

compare our methodology against real executions, and confirm the usefulness of our simulation for load balancing parameter exploration. Section 6 presents related work on simulation-based tools, justifying our choices. Section 7 concludes the paper, listing our major contributions and future work.

2 Ondes3D: a Typical Imbalanced MPI Code

Ondes3D [1] is a geophysics simulator to conduct seismic hazard assessment at regional scale. It approximates the differential equations governing the elastodynamics of rock medium using finite-differences numerical methods (FDM). The problem domain is statically partitioned in cuboids, as depicted in Figure 1a. Each iteration (see Figure 1b) corresponds to a given time step and consists in calling three macro kernels (`Intermediates`, `Stress`, and `Velocity`) that apply a series of finite differences micro kernels (see an example in Figure 1c) to the whole domain. Message passing consists in asynchronous neighborhood communications intertwined with the three macro kernels. There is no global barrier, which enables a slightly asynchronous evolution of each process.



(a) Spatial imbalance for the first iteration represented by a color gradient for each rank in a 16×16 grid (256 processes).

(b) Total number of cycles for each kernel along iterations for each rank in a 8×8 grid (64 processes).

(c) How much the differences evaluate to contrasting values in the `CPML4` kernel along iterations, for each of the 64 ranks.

(d) Expected duration (cycles) and number of calls of `CPML4`.

Figure 2: Load imbalances for the Ligurian workload: (a) spatial load imbalance; (b) temporal load imbalance for three kernels; (c) `CPML4` substrate values evaluating to distinct values; and (d) explanining temporal anomaly with `CPML4` argument analysis.

Despite this apparent regularity (processes manage a cuboid with the same geometry and run the same regular code), Ondes3D suffers from load imbalance that limits its scalability. The main source of imbalance was previously identified [2] as the extra-computation needed to deal with boundary conditions. The processes on the border of the domain have thus more work, which causes significant spatial imbalance. In Section 2.1, we report another, more subtle, source of spatial imbalance, caused by the heterogeneous simulation substrate. Until now, due to the highly regular structure of the code, studies had solely focused on spatial imbalance and had overlooked possible temporal imbalance. In Section 2.2, we show that this imbalance is even stronger than the spatial one, and present evidences of its origin being related to low level optimizations taking place inside the CPU. Both spatial and temporal load variations are highly related to the nature of the computation and thus seem quite hard to predict accurately. Yet, we show in Section 2.3 how a coarse grain execution trace can be derived from a fine grain execution trace. Such property is natural for domain-based decomposition and is exploited in Section 4 to speed up simulations.

In the following we provide an in-depth study of the spatial and temporal load imbalance of Ondes3D. To this end, we use a historical Mw 6.3 earthquake workload [11] that arose in Liguria (north-western Italy) and present performance analysis of several simulations with varying numbers of processes. The code has been compiled with GCC 6.1.1 with `-O3` and instrumented with PAPI [12] and SMPI [9]. While we report results only for this particular setup, we have systematically observed similar issues with other workloads, CPUs (Xeon X3440, X5650, E5-2630, and i7 4600M), and compilers.

2.1 Understanding Spatial Imbalance

Figure 2a depicts a 16×16 domain decomposition where each cell in the cartesian grid represents one of the 256 processes and is in charge of a cuboid subdomain. In this heatmap, the color indicates the total computational load per process during the first iteration, before the triggering of the initial shock that originates in the 13×5 subdomain. Processes on the borders demonstrate a much higher computational load (red color) than those located inside the physical domain. Another, much more subtle, source of spatial imbalance (blue shades), depends mostly on the

rock multi-layer configuration of the input (six layers for this scenario). Although minor, such effect exists and solely depends on the substrate geometry.

2.2 Characterizing Temporal Imbalance

The Ondes3D code does not exhibit any structure (convergence loops, refinements, thresholds) that could lead to an evolution of computation load along simulation iterations. As shown in Figure 1c, there are conditional branches, but a closer examination reveals that they are related to absorbing conditions and thus solely to the fixed problem geometry. Yet, as illustrated in Figure 2b, one can observe a variability in computational costs along iterations that is even higher than the spatial variability incurred by the absorbing conditions. This figure details the behavior of all 64 processes (each box in the 8×8 grid), showing (in the vertical axis of each box) the total number of cycles per macro kernel (the `PAPI_TOT_CYC` hardware counter) as a function of the iteration (horizontal axis). The number of cycles seems to follow the earthquake shock progression, standing out around the eightieth iteration.

To explain the origin of this variable computational cost, we use the `CPML4` kernel shown in Figure 1c. This kernel is only one out of the 24 small inlined kernels but is perfectly representative of the other ones. It is called by the `Intermediates` macro kernel that iterates over the cuboid sub-domain with three nested loops. For each cell of the subdomain, the `CPML4` kernel is called nine times with slightly different parameters, resulting in more than a million calls per process and time step in a 64-process simulation of the Chuetsu-Oki workload. In this code, the variables `dx` and `dt` are simulation constants, while variables `x1`, `x2`, `x3`, and `x4` represent the rock medium state (speed, pressure, ...) that unfolds along the iterations.

For completeness, we proposed and verified several hypothesis to explain the temporal load variation, among them: conditional branches in micro kernels (e.g., the `if` in `CPML4` code of Figure 1c), variable duration of math functions depending on their input (e.g., the `exp` call in the `CPML4` code), arithmetic exceptions at the architecture level, compiler level optimizations that could, for example, introduce conditional branches in the assembly code, data cache effects, higher cost of divisions when compared to multiplications, and so on. By either playing with compiler options, carefully looking at the assembly code, or finely instrumenting the different operations, we have ruled out all these assumptions. Nevertheless, we noticed a strong correlation between the evolution of the load and the instruction cache misses (`PAPI_L2_ICM`) and branch miss-predictions (`PAPI_BR_MSP`). As we will show next, this correlation is a consequence and not the cause of the evolution.

Let us consider the `x1`, `x2`, `x3`, and `x4` arguments of the `CPML4` kernel (still in Figure 1c). The `return` statement of this ker the arithmetic expression in the FPU. So we instrumented the `CPML4` kernel to count how many times per time step and per process these differences were equal to zero (let us name these numbers $n^0_{2,1}$for `x2-x1` and $n^0_{4,3}$for `x4-x3`). Looking at them separately indicates no correlation with the temporal load evolution. However the difference $|n^0_{2,1} - n^0_{4,3}|$ (see Figure 2c) is perfectly correlated with the computational load change (see Figure 2b) and with the growth of the branch miss-prediction counters. Intuitively, this value measures how often only one of the two differences is zero.

To confirm this hypothesis, we instrumented the `CPML4` kernel to record its duration for each call (in cycles) along with the result of the two differences (`x2-x1` and `x4-x3`). Since this kernel is called very often, we traced a single rank (which demonstrates a strong variability in time) and only for two very different time steps (iteration 5 with low computational load and iteration 115 with maximal computational load). Figure 2d shows the number of cycles of a single call to the `CPML4` kernel depending whether both differences are zero, non-zero or whether only one of them is zero. Just after simulation starts, in iteration 5, when the seism has not yet reached this region,

both differences are always zero (red points in Figure 2d) and the number of cycles is relatively small. In iteration 115, both differences are non zero (green points in Figure 2d) for more than 25% of the `CPML4` calls and the average execution time is then 168 cycles slower. The FPU optimization to speed up multiplications by zero is thus worthwhile. Interestingly, the situation where only one difference is zero (blue points in Figure 2d) is sporadic (about 1% of calls) but both the average and the minimum execution time are significantly longer. It is interesting to note that the corresponding cells i, j, k for which the condition is true are spatially organized but they do not relate at all to cache alignment and do not generate additional data cache misses but clearly more branch miss-predictions and L2 instruction cache misses that participate in the slow down of the code. The observed duration increase originates from the combination of both a speed-up of multiplications by zero and of branch miss-predictions in the FPU incurred by the irregular sequence of zeros and non-zeros.

Finally, even if we focused on the `CPML4` kernel here, all other small inlined kernels share the same structure. It is thus the aggregated contribution of all these small additional cycles that generates the temporal load variation.

2.3 Spatial Aggregation

The previous analysis shows that both spatial and temporal load variation are nearly smooth but it seems difficult to anticipate the performance by solely looking at the input data, without running the code at least once. Yet, since this code relies on domain-based decomposition, we will now show that it is possible to derive a coarse grain execution trace from a fine grain execution trace. Indeed, the amount of work induced by a given area of the domain at a given time step corresponds to the sum of the amount of work of the corresponding sub-areas. We conducted a number of runs with different simulation configurations to validate this coarse grain workload prediction from a fine-grain domain decomposition.

Let us consider a 16×16 decomposition, as shown on Figure 2a. When using instead a coarse grain 4×4 grid, the computational load of each process is the sum of the 16 corresponding processes in the fine grain 16×16 decomposition. Figure 3 shows the computing time (vertical axis) of each main loop iteration as a function of the time step (horizontal) for each rank (colors). The left facet shows the measurements from an execution with a coarse-grain (4×4) grid. The right one shows the aggregation of a fine-grain (16×16) grid. The difference between the coarse-grain trace and the aggregation from the fine-grain trace is minimal and allows to fully capture the temporal evolution for a given earthquake scenario.

2.4 The Necessity of Dynamic Load Balancing

Despite the regularity of the Ondes3D code, it is fundamentally spatial and temporal imbalanced. We believe that such imbalance is usually overlooked, as it requires a careful and fine analysis to be identified. Therefore, it should also be present in many other so-called regular applications.

Modeling and predicting such load variation is hard, as it strongly depends on the initial and evolving conditions of the earthquake simulations. Even if we could rewrite Ondes3D, using adaptive data structures to allow uneven domain decomposition, some periodic data and computation re-balancing would still be required. That is why we propose to use a simpler approach (from the application developer mindset) by mixing load balancing at runtime with over-decomposition, a strategy in the scope of the AMPI framework [5, 6]. The next section presents our AMPI-aware version of Ondes3D.

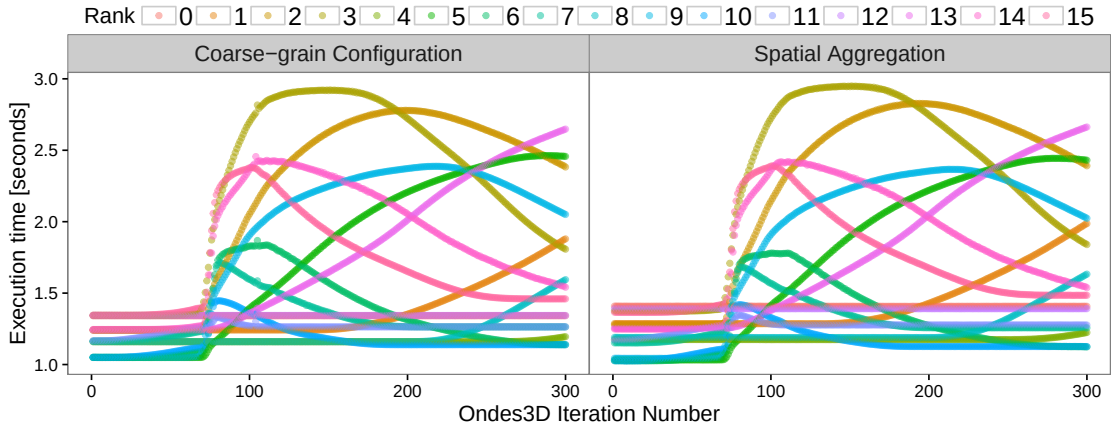


Figure 3: Execution time (vertical axis) along the time steps (horizontal) for each rank (color) considering all three kernels; the left facet is the behavior of a 16-process run, the right facet shows the same, but calculated from a 256-process run.

3 Adaptive MPI: Migration-based Load balancing with Over-Decomposition

AMPI stands for Adaptive MPI [5, 6], and is an MPI implementation built upon the Charm++ framework and runtime [4]. Thus, enabling MPI applications to benefit from the underlying dynamic load balancing infrastructure. Charm++ enables the decomposition of the problem domain in more tasks than the number of available physical cores (i.e., over-decomposition). Tasks are mapped to Virtual Processors (VP), which are implemented as user-level threads suitable for migration. At runtime, many VPs are mapped to the same system process which is pinned to a physical processor. The runtime can migrate these VPs among system processes. Over-decomposition and migration enables Charm++ to periodically redistribute tasks, according to a load balancing heuristic. The heuristics use load information from the near past to define a new VPs mapping. Therefore, the load balancing in AMPI is dynamic since it responds to load fluctuations during runtime.

Application Modifications for AMPI

The application has to be changed in three ways to exploit the AMPI's load balancing functionality. (1) First, relying on user-threads requires the code to have no global or static variables (otherwise they would be shared among VPs). (2) Second, Pack-and-Unpack (PUP) functions must be implemented to serialize dynamically allocated data and allow migrations to take place. Writing these functions can be tedious, since it may entail a profound understanding of the application data structures. (3) The last modification is the addition of a call to `MPI_Migrate`. When triggered, this call indicates that the application is ready for load balancing. The application developer must call this function only when there are no active communications or open files. Typically, this call is placed at the end of the outermost loop iteration (the time-step loop in the case of Ondes3D), being thus called periodically.

Performance-impacting Parameters

A number of parameters influence the effectiveness of the load balancing. Some **load balancing heuristics** are naturally more scalable than others (e.g., centralized *vs* distributed). In particu-

lar, there is a trade-off between achieving a perfect load balancing and the time spent migrating data to achieve it. The best heuristic therefore highly depends on the application dynamicity and on platform characteristics. The **level of over-decomposition** influences how well the load balancer is capable to redistribute the load. In this aspect, the more sub-domains, the better. Conversely, increment the amount of sub-domains generally increases the communication cost at the application level. At some point, such cost exceeds the benefit of load balancing. The number of tasks also increases the amount of work done by the load balancer, which can become significant at large scale. Likewise, **the number of computing resources** is a critical parameter in the overall performance of a parallel program and the more is not always the better. Finally, fine-tuning the **frequency of load balancing** is essential to obtain good performance. Indeed, the more frequent the load balancing, the more likely we are to detect load imbalance and tackle it. Yet, if the balancing is too frequent, it will incur in too much overhead. Moreover, since calling `MPI_Migrate` incurs a global barrier, it may also destroy any natural compensation of load imbalance throughout iterations afforded by asynchronous neighborhood communications.

3.1 Potential Performance Gains are Significant

Ondes3D has been already ported to AMPI, with performance gains up to 28.35% [7] on an 8-node cluster (64 cores). We present new performance measurements on a 12-node cluster (288 cores) at BRGM to illustrate the typical gains of the dynamic load balancing provided by AMPI. This platform is typical of the scale at which Ondes3D is expected to run. It has two AMD Opteron 6344 processors per node, using an Infiniband 40G network (MT27500 Family). We simulated the first 500 time steps of the Mw 6.6 2007 Niigata Chuetsu-Oki earthquake, using a grid dimension ($1152 \times 1152 \times 384$), adjusted to the increased computational power of this cluster.

We evaluate four load balancing heuristics (RefineLB, NucoLB [13], HwTopoLB [14], and HierarchicalLB [15]) with two over-decomposition levels (8 and 16), and a `MPI_Migrate` call every 20 time steps. Results are compared against the pure MPI-based implementation and the AMPI with no load balancing (baseline). RefineLB, part of Charm++, has been chosen because it was the best performing one in our previous work. HierarchicalLB is a distributed hierarchical algorithm. It was tested in two configurations: the first employs HwTopoLB at platform level and NucoLB at host level; the second employs HwTopoLB at both levels.

The results in Figure 4 show the average makespan of at least 10 executions for each case, with 95% of confidence interval. Our AMPI baseline is approximately 6.29% better than MPI. RefineLB is 36.58% faster (using an 8-factor over-decomposition) than the baseline and the best load balancer overall, probably because topology awareness was not essential in this homogeneous cluster with a low-latency network.

These results are better than the ones obtained on our previous work, probably due to a different communication to computation speed ratio on this platform. This clearly demonstrates the significant benefits of dynamic load balancing and over-decomposition to iterative MPI codes such as Ondes3D.

3.2 Difficulties to Evaluate the Benefits of Load Balancing

While, as demonstrated above, evaluating the benefits of load balancing with real execution is feasible, it presents several difficulties. As explained in Section 3, several parameters (heuristic, level of decomposition, frequency, ...) strongly impact performance of the load balancing and the **optimal configuration** is highly dependent on application dynamics and on platform characteristics. The results presented in the previous section required an empirical exploration

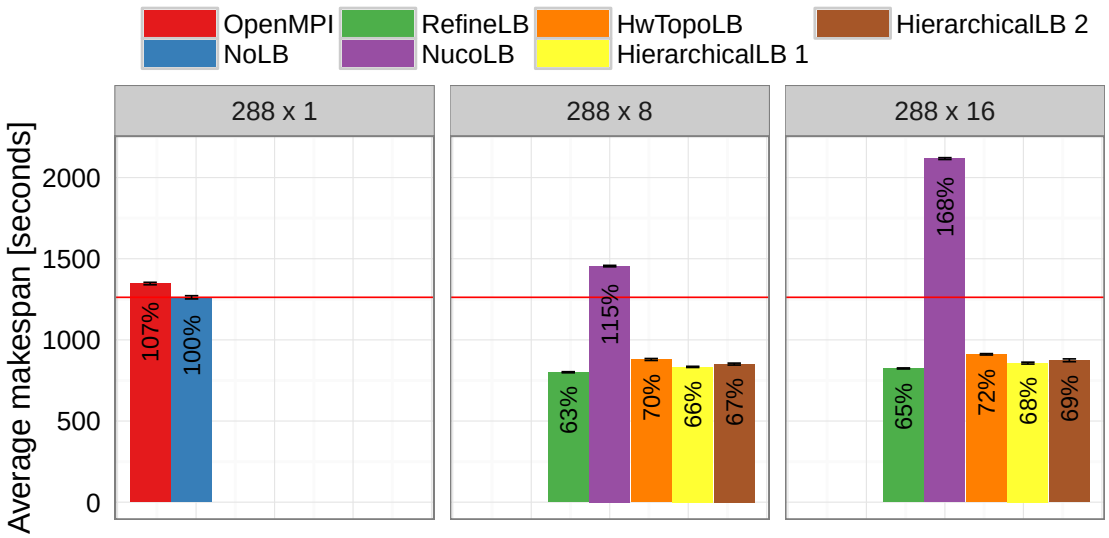


Figure 4: Mean execution time of Ondes3D on 288 cores with different task granularities (VP counts) and load balancers.

of this configuration space. Running the same Earthquake simulation several dozens of times at scale on a production system solely to determine such parameters is not only unsatisfying but also both **resource and time consuming**.

This is why we propose a simulation workflow that addresses these difficulties in a lightweight way. The application benefits can be evaluated with minimal code changes (even if it has not been ported to AMPI yet), and it needs to be ran only once to outline its execution. These features save development and evaluation time. Besides, the tracing process can be done using sequential emulation, which requires a single host.

4 Simulation Workflow to Evaluate the Impact of the Configuration Space on AMPI Performance

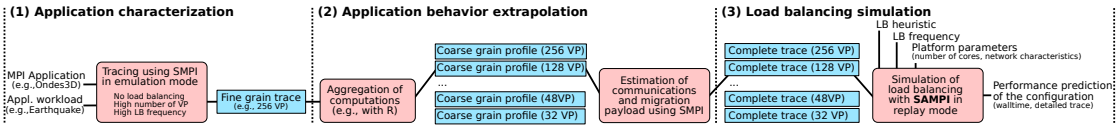


Figure 5: Performance evaluation workflow. The irregular structure of the load of the application over space and time needs to be carefully and finely captured in step 1. It can then be spatially agregated in step 2 to study different levels of over decomposition. The final step 3 allows to quickly study the influence of load balancing parameters.

Our simulation workflow relies on SimGrid's SMPI [9], which offers two key features on which we have built. (1) First, SMPI is quite flexible and allows to study MPI applications either in *emulation mode* or through *trace-replay*. In emulation mode, unmodified MPI applications are sequentially executed, in a controlled way, on top of the simulator. Such approach can be costly in terms of simulation time (all the computations are actually executed) but allows to faithfully capture the behavior of complex applications while using only one core. In trace replay mode,

the trace of an MPI application is replayed on top of the simulator and takes only a small fraction of the time to finish than a normal run at full scale (typically one or two minutes on a single core compared to 15 minutes on a cluster for our Ondes3D simulations), thus upholding our resource requirement goal. (2) Second, SMPI builds on the hybrid flow-level network models of SimGrid [16] that allow to faithfully model communications and contention, which is essential in the context or our study. Next, we described SMPI modifications to simulate AMPI, and our SAMPI workflow.

4.1 Simulated Adaptive MPI (SAMPI)

A few modifications in SMPI have been necessary to simulate AMPI. (1) First, we had to augment the SMPI interface with the non-standard `MPI_Migrate` function both in the emulation mode (to generate an event in the trace) and in the trace replay mechanism. In trace replay, whenever the load-balancing is activated, this function calls the `MPI_Barrier` function, the load balancing heuristic that defines the new process mapping, and simulates all resulting processes migrations. (2) To obtain a faithful behavior of AMPI load balancing heuristics, it is essential to stay as close as possible to their real implementation. We investigated the possibility to directly plug Charm++'s load balancers, but they heavily rely on Charm++ specific elements. Fully porting the Charm++ runtime ecosystem on top of SimGrid would require a significant development effort and a deep knowledge of the Charm++ internals. Instead, we have manually extracted and slightly adapted two centralized load balancers by hand: **GreedyLB** and **RefineLB**. This operation consisted mainly in removing internal references to Charm++, making sure that the heuristic implementation remains intact. A few trace replay routines also had to be modified to collect the load data that is fed to the load balancing heuristics. (3) Finally, the migration cost is accounted for in the simulated execution time. We rely on SimGrid's contention-aware network models when sending the data belonging to the migrated task from its original location to its destination. The migration payload is estimated by trapping `malloc` and `free` functions in emulation.

4.2 Low-cost Workflow to Study Ondes3d-like Applications

We build upon our simulator to propose a low-cost three-step workflow to study Ondes3D, show in Figure 5. Although we evaluate our workflow with this application, we believe that it is generic enough to apply to any imbalanced MPI code that is iterative and relies on static domain decomposition.

(1) First, the original behavior of the application is traced (e.g., using SMPI) at a very fine-grained decomposition level (with the largest number of processes to be studied) and invoking the `MPI_Migrate` function at every iteration. The resulting trace will contain, for all Ondes3D iterations, the computational cost (in micro-seconds or cycles) of the three Ondes3D macro kernels interleaved with the communication pattern (source, destination, payload) of the application. All timings are converted in flops to obtain a fully *Time Independent Trace* [10] (TIT). In this step, instead of SMPI, we could have used a classical tracing on a small size cluster, recording the number of cycles with PAPI instead of real timings as done in [17]. Note that the resulting trace is representative of a given input scenario (earthquake) but not linked to a given platform anymore. **(2)** The second step of the workflow aims at extrapolating the previous fine grain trace to any coarser decomposition level (i.e., using fewer processes). As shown in Section 2, we can spatially aggregate computational costs with information loss. For a given decomposition level, one can create (e.g., using a simple R script) a computational load *profile* for each rank, kernel and iteration of the Ondes3D application. Extrapolating communication patterns can be more

complicated (although techniques such as the ones in [18] could be used) so instead, a simple approach, involving a minor modification of the Ondes3D application, consists in using SMPI in hybrid emulation/trace replay mode: the code is emulated with fewer VPs, as in the first step, but every computationally intensive part is skipped and replaced by the corresponding spatially aggregated computational profile read from the coarse grain profile. As a result, a complete *time independent trace* (with both computations and communications) for a reduced number of VPs is obtained. If building on the spatial aggregation property is not possible, then a trace for every envisioned level of decomposition should be obtained directly as in the first step. **(3)** Finally, the last step simulates the load balancing of a TIT trace obtained in the previous step using SAMPI for a given Load Balancing configuration (heuristic and frequency) and platform. SAMPI builds on the contention-aware network models of SimGrid and is used in trace replay mode, which allows to quickly and easily explore the AMPI parameter space to find the best combination of load balancer heuristic and frequency, decomposition level, for the initial application workload (earthquake).

5 Experimental Results and Evaluation

Several issues should be solved to correctly validate the accuracy of predictions obtained in simulation. Solely comparing the (predicted) makespan of simulations with the one of real-life executions on a few examples may be insufficient to be fully trusted. Yet, comparing detailed execution traces (e.g., with Gantt charts) of an application as complex as Ondes3D is simply impossible. Other ad hoc intermediate and aggregated representations are thus needed. In our context, iterations and load imbalance are of primary importance. Therefore, we decided to track the resource usage per processor and per iteration and to study its evolution both temporally and spatially. In this section, we use this performance metric, to compare reality and simulation both qualitatively and quantitatively.

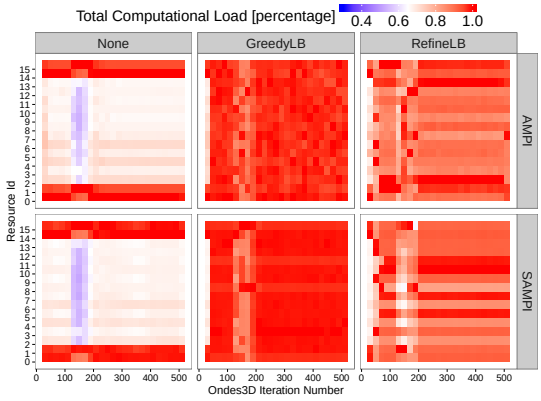
We first report validation of our SAMPI workflow, contrasting it with AMPI. We then present a typical use of SAMPI to identify the best load balancing parameters for a given Ondes3D workload and a given platform.

5.1 Testbed, Validation Methodology and Ondes3D Workload

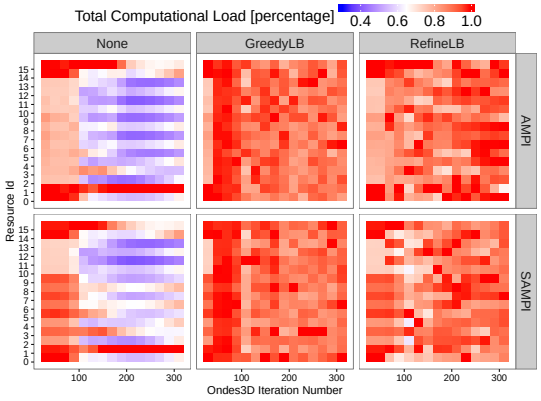
The experiments have been conducted on 16 nodes of the Parapluie cluster, which is part of the Grid'5000 [19]. Each node has two 12-core 1.7GHz AMD Opteron 6164 HE processors and is interconnected through a 20G Infiniband 4x QDR network. Ondes3D was compiled with GCC 4.8.4 with optimization flags `-O2 -march=native`, and the AMPI tests use Charm++ 6.6.1. All experiments and measurements shown below were obtained in a controlled Grid'5000 environment.

5.1.1 Tracing Methodology (comparing AMPI with SAMPI)

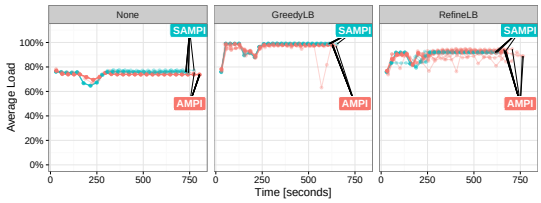
The behavior of AMPI-based Ondes3D executions is registered using Tau [20], integrated in Charm++'s Tau tracemode. We configured AMPI to use MPI for all communication operations and enriched (through manual instrumentation of Ondes3D with the Tau API) the traces with the timestamps of each iteration of the main loop of Ondes3D. Time-independent traces for our trace replay, according to our SAMPI workflow, have been obtained through sequential emulation using SMPI. All emulations have been conducted in one node of the cluster to obtain faithful computational load profiles.



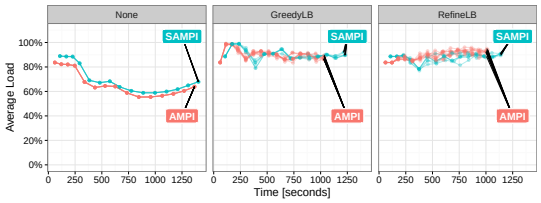
(a) Per-process computational load of the Chuetsu-Oki workload.



(b) Per-process computational load of the Ligurian workload.



(c) Average Load and makespan using the Chuetsu-Oki workload.



(d) Average Load and makespan using the Ligurian workload.

Figure 6: Comparison of SAMPI executions (obtained in simulation) against AMPI runs (reality) for two representative workloads: Chuetsu-Oki (left) and Ligurian (right); for each of them, the top row shows six heatmaps (without load balancing, Greedy, and Refine) illustrating the computation load (color gradient) as a function of the Ondes3D iteration for all 16 processes; the bottom row shows the average aggregated load along time, highlighting the makespan of multiple runs.

5.1.2 Ondes3D Workload Description

We tested two very different earthquake scenarios in Ondes3D. The first one is the Mw 6.6 Niigata *Chuetsu-Oki* (2007) from Japan [21]. Running the full simulation (6000 time steps) takes an unreasonable amount of time, especially because several executions are needed to obtain statistically significant results. So, in order to keep a reasonable duration for the experiments, we limited this simulation to the first 500 time steps. For the same reason, the number of cells in each direction was reduced to $300 \times 300 \times 150$. The second simulated scenario is the Mw 6.3 *Ligurian* (1887) from north-west Italy. It was limited to 300 time steps and the number of cells to $500 \times 350 \times 130$.

5.2 Validation: Comparing SAMPI (simulation) against AMPI

In our validation experiments, we fix the domain decomposition to 64 VPs (always mapped to 16 processes), and call `MPI_Migrate` every 20 time steps. From our experience, this configuration is relatively good and allows to focus our evaluation on sound scenarios. Figure 6 depicts the comparison of simulation outputs with real AMPI traces for situations without load balancer, with GreedyLB and with RefineLB, for the two workloads: Chuetsu-Oki (left) and Ligurian

(right).

5.2.1 Per-process Computational Load Analysis (Heatmaps)

Heatmaps in Figures 6a (Chuetsu-Oki) and 6b (Ligurian) show the computational load (as a color gradient) for each core (in the vertical axis) along the Ondes3D iterations. A reddish color represents higher computational load, while blue represents idleness. Each heatmap corresponds to an execution, either real (AMPI in the top row) or simulated (SAMPI in the bottom row), with a given load balancer (no load balancing on the left column, Greedy in the center, and Refine on the right column). In all configurations, the real and simulated load distribution are very similar, which shows the ability of our workflow to capture the complex behavior of AMPI.

Figure 6a shows that for Chuetsu-Oki, the case without load balancing leads to many underutilized resources (white and bluish regions). Both LB seem to significantly improve this situation by making processes 2 to 13 receive more load. It is interesting to note that GreedyLB achieves a much better load balancing than RefineLB (being more conservative) and that this is visible in simulation as in real execution traces. The load structure for the Ligurian workload is quite different (see Figure 6b). There seems to exist an alternating load irregularity in processes whose ranks belong to the center of the domain decomposition (those with white and bluish colors without load balancing). The Greedy and Refine load balancers are again effective to redistribute the load, since we observe a much more even computational load across processes although not as good as the one obtained for the Chuetsu-Oki workload.

The heatmap views are based on one run for each case (hence 12 runs considering in total both workloads). It should be noted that any new execution (either real or in simulation from a new trace) will lead to slightly different outcome and that focusing on the load of a given core at a given time-step is thus not really meaningful. From such view, it seems that GreedyLB is the best choice from the load balancing point of view, but communication (both from the application and from load balancer) should also be taken into account. In the following, we provide makespan results using the average load as a function of the execution time.

5.2.2 Average Load and Makespan Comparison Analysis

The plots in Figures 6c (Chuetsu-Oki) and 6d (Ligurian) depict the evolution of the average load across the cores participating in the execution. The average load (in vertical axis) is drawn as a function of time (horizontal) for both SAMPI (blue) and AMPI (red). The points along the lines indicate the moments where the metric is calculated (in the beginning of the `MPI_Migrate` operation, at the end of the load balancing interval); lines show the trend. As before, the horizontal facets in each figure indicate the computed metric without load balancing, with Greedy and with Refine load balancers. They enable richer analysis; depicting many runs on a single chart (at least five).

For the Chuetsu-Oki workload (see Figure 6c), we can confirm that GreedyLB performs better than RefineLB, both in simulation as in real life. One could expect GreedyLB to be worse than RefineLB, due to the large amount of migrations it performs. It seems however that, in this case, the default overload tolerance of 1.05 used by RefineLB is too high. Regarding the comparison of SAMPI against the real AMPI, one can note that across several runs, SAMPI is slightly too optimistic. That being said, such inaccuracy would not affect our choice of load balancer. It is interesting to see quite significant variability in the real executions (perfect isolation is tough to achieve on a cluster), being generally larger than in the simulations. Variability of simulations comes from the tracing variations in step 1 of the workflow. For the Ligurian workload (see Figure 6d), as on the previous scenario, both simulation and real life have similar load unfolding.

Again, there is some makespan disparity with SAMPI timings being pessimistic. Yet the trends remain correct and the RefineLB performance is better both in simulation and in real life.

5.2.3 Summary of the Validation Procedure

Our simulation is able to mimic in a realistic way the evolution of the load distribution of real life executions, which is one of the main aspects we are trying to analyze. There is still some inaccuracy in terms of absolute time prediction and we are currently investigating the source of this discrepancy. Yet, since the trends remain correct, this does not not affect the identification of the optimal load balancer in the two investigated scenarios. In the next section, we demonstrate how the SAMPI simulator can be used to explore different load balancing parameters.

5.3 Tuning Load-balancing Parameters with Simulation

In this section, we present experiments to demonstrate the usefulness of our simulation approach, through the investigation of the parameter space of AMPI. Two parameters are evaluated: four configurations for load balancing interval; and five levels of over-decomposition. For these experiments, we decided to focus on the Ligurian scenario, since it is larger and more complex than the Chuetsu-Oki. Therefore, tuning in simulation is more likely to be useful for it.

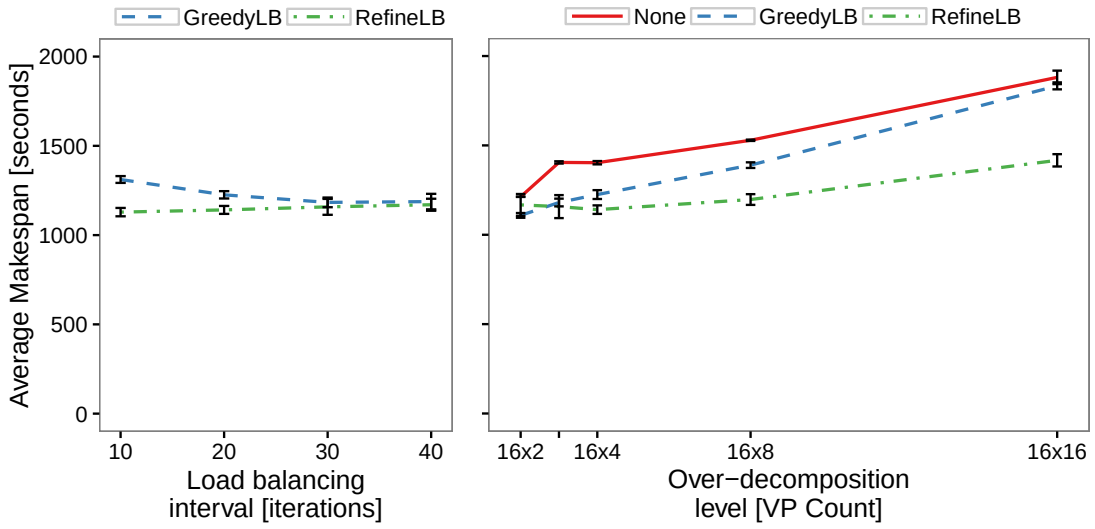


Figure 7: Simulated makespan predictions for the Ligurian earthquake simulation with (left) four load balancing intervals (in number of iterations) and (right) with five over-decomposition levels (2, 3, 4, 8, and 16) on 16 cores.

5.3.1 Investigating the influence of load balancing frequency

In these experiments, we use the previous workflow to study the makespan of Ondes3D with different load balancing intervals. In the traces obtained in the end of Step 2 a call to `MPI_Migrate` is present for each VP at the end of every time step. During the simulation with SAMPI, we control an enforce a different load balancing frequency by actually calling the barrier and the load balancing, for example, only every 10, 20, 30 or 40 iterations. Intuitively, the more frequent the

calls, the better the load balancing but also the more important the barrier and data migration overhead. Figure 7 shows the influcene of the load balancing frequency (horizontal axis) on the makespan (vertical axis) of a 16×4 VP configuration. It is interesting to see that in this scenario, calling GreedyLB too frequently is deterring, whereas it is the opposite for RefineLB. This can be explained by the fact that GreedyLB balances the load without neither minimizing the amount of data migration nor taking into account the current task mapping whereas RefineLB is much more conservative.

In these experiments, we traced 10 application executions, using `smpirun`. Each emulation took approximately 5 hours to finish, which gives us an approximate 50 hours for tracing in the first step of our workflow, using a single computer node. Tracing with Tau as done in [17] would allow to divide this tracing time by the number of computing nodes. Each simulation takes about 200 seconds to finish on a standard laptop equipped with an Intel Core i7 processor. For each of the 10 input traces, we simulated 2 load balancing heuristics with 4 different frequencies. This gives us a total of approximately 4 hours and 26 minutes for the simulations of step 3. Again, since these simulations are all independent, they can be speed up by running them in parallel.

5.3.2 Investigating the influence of decomposition level

Another important performance affecting parameter is the over-decomposition level. The results on how over-decomposition affects the makespan of Ondes3D, when calling `MPI_Migrate` every 20 time steps, are in the right plot of Figure 7. The average makespan (vertical axis) is shown as a function of five over-decomposition configurations (horizontal). In the absence of load balancing (None), overdecomposing is, as expected, deterring since this creates extra-communication between VPs. Yet having more and smaller VPs allows for a better redistribution of the load. This trade-off was explained in Section 3. It is interesting to note that for RefineLB the sweet spot is reached with a 16×4 decomposition. However, for GreedyLB the decomposition level should be as small as possible, which is again explained by the fact that its careless migrations scale very badly. In the end, the 16×2 GreedyLB configuration is equivalent to the 16×4 RefineLB configuration but exhibit quite different load balancing behaviors.

For these simulations, we needed a trace for each over-decomposition level. The fine grain traces are obtained with SMPI emulation (step 1) in about 5 hours. Extrapolating a fine-grain trace into a coarse grain one (step 2) requires a few minutes. Once this is done, each simulation with SAMPI (step 3) runs in about 200 seconds on a laptop. We ran five simulations (one for each over-decomposition level) for each three load balancing heuristics (including absence of load balancing), which takes about 4 hours on a single core of the laptop. Running similar experiments at scale would require the cluster for more than a day.

6 Related Work

The simulation workflow we propose mostly depends on two factors. First, a faithful model of modern HPC networks and MPI implementations is essential since communications play a crucial role in the load balancing trade-offs. Second, the ability to run simulations both in trace-replay and emulation modes is more than helpful to select the approach most suited to the resources at hand. There is a plethora of simulation frameworks and tools to study MPI applications [8] and at least four of them support both emulation and trace-replay and could thus have been modified: BigSim [22], SST/Macro [23], Xsim [24], and SimGrid [8] (through SMPI [9]). BigSim is part of Charm++, thus supporting the simulation of AMPI applications, such as our Ondes3D implementation. Although it is linked to Charm++, it does not allow to change the load balancing parameters during trace replay and this would require major modifications to the

BigSim simulator. SST-Macro [25] allows both trace replay through the DUMPI module and online simulation (emulation) through skeletonization. Although SST-macro is quite flexible, and has many network models, including flow-based ones, its emulation support is not sufficiently mature yet to run an application as complex as Ondes3D. Finally, Xsim mostly focuses on extreme-scale executions, does not fully support emulation of unmodified MPI applications, and its sources are not currently available.

For this work, we therefore chose to rely on the free software SimGrid, whose SMPI [9] interface allows both emulation and trace replay of MPI applications. SMPI leverages SimGrid's thoroughly validated flow communication models [26], while also accounting for specific characteristics of MPI implementations [8]. Hence, SMPI allows us to collect accurate execution traces from emulation, and its replay mechanism allows us to quickly simulate this execution as many times as needed. Traces are completely open enabling an easy transformation through spatial aggregation.

7 Conclusion

We propose a simulation based approach for the performance evaluation and tuning of dynamic load balancing applied to iterative MPI applications. Our approach allows the estimation of performance gains from load balancing at low cost, both in terms of time and of resource requirements. Although we apply it to a geophysics application (Ondes3D), its structure is very typical among legacy MPI applications. Therefore, we believe the usefulness of our approach is not limited to Ondes3D. Our contributions are three-fold: (a) An in-depth analysis of the spatial and temporal load balancing issues found in Ondes3D. The latter demonstrates how dynamic load imbalance can arise even when there is no indication of temporal variability in the code. (b) Implementation and validation of a simulator, called *SAMPI*, that uses SimGrid to simulate over-decomposition and load balancing, mimicking the behaviour of AMPI. This simulator extends the MPI trace replay functionality included in SimGrid, to enable the fast exploration of different load balancing scenarios from a single execution trace. (c) A performance evaluation workflow that uses a combination of sequential emulation and time-independent trace replay to evaluate load balancing scenarios at a small fraction of the cost of real executions.

As future work, we plan to consider spatial aggregation of traces (as shown in Section 2.3) to accelerate the workflow. We also plan to investigate other Ondes3D workloads to understand how the computational load evolves, possibly enabling us to extrapolate the behavior for higher process counts and to better guide and trigger the load balancing. As a side effect of our work, although out of the scope of our initial goals, the gathered knowledge obtained in this work will be employed to improve the performance of the Ondes3D kernels.

Acknowledgements

This work was supported by the projects: CAPES/Cofecub 764-13, FAPERGS/Inria ExaSE, CNPq 447311/2014-0, and the CNRS/LICIA Intl. Lab. Some experiments were carried out using Grid'5000, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several other organizations (see `https://www.grid5000.fr`). These research results have received funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E Project, grant agreement n° 689772.

References

[1] F. Dupros, F. D. Martin, E. Foerster, D. Komatitsch, and J. Roman, "High-performance finite-element simulations of seismic wave propagation in three-dimensional nonlinear inelastic geological media," *Parallel Computing*, vol. 36, no. 5–6, pp. 308 – 325, 2010.

[2] F. Dupros, H.-T. Do, and H. Aochi, "On scalability issues of the elastodynamics equations on multicore platforms," in *ICCS 2013 : International conference on computational science*, ser. Procedia Computer Science. Barcelone, Spain: Elsevier, Jun. 2013, p. 9 p.

[3] V. Martinez, D. Michéa, F. Dupros, O. Aumage, S. Thibault, H. Aochi, and P. O. A. Navaux, "Towards seismic wave modeling on heterogeneous many-core architectures using task-based runtime system," in *SBAC-PAD*. IEEE Computer Society, 2015, pp. 1–8.

[4] L. Kalé and S. Krishnan, "CHARM++: A Portable Concurrent Object Oriented System Based on C++," in *Proceedings of OOPSLA'93*, A. Paepcke, Ed. ACM Press, September 1993, pp. 91–108.

[5] C. Huang, O. Lawlor, and L. Kalé, "Adaptive MPI," in *Lang. and Compilers for Parallel Computing*, ser. LNCS, L. Rauchwerger, Ed. Springer Berlin Heidelberg, 2004, vol. 2958, pp. 306–322.

[6] C. Huang, G. Zheng, S. Kumar, and L. V. Kalé, "Performance evaluation of Adaptive MPI," in *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, March 2006.

[7] R. Keller Tesser, L. Lima Pilla, F. Dupros, P. Navaux, J.-F. Mehaut, and C. Mendes, "Improving the performance of seismic wave simulations with dynamic load balancing," in *Intl. Conf. Parallel, Distributed and Network-Based Processing (PDP)*, Feb 2014, pp. 196–203.

[8] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Paral. and Distr. Comp.*, vol. 74, no. 10, 2014.

[9] P.-N. Clauss, M. Stillwell, S. Genaud, F. Suter, H. Casanova, and M. Quinson, "Single node on-line simulation of MPI applications with SMPI," in *Intl. Par. Distr. Proc. Symp.*, May 2011, pp. 664–675.

[10] F. Desprez, G. S. Markomanolis, and F. Suter, "Improving the accuracy and efficiency of time-independent trace replay," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, Nov 2012, pp. 446–455.

[11] H. Aochi, A. Ducellier, F. Dupros, M. Terrier, and J. Lambert, "Investigation of historical earthquake by seismic wave propagation simulation: Source parameters of the 1887 M6.3 Ligurian, north-western Italy, earthquake," in *8ème colloque AFPS, Vers une maitrise durable du risque sismique*, Champs-sur-Marne, France, Sep. 2011, p. 6 p.

[12] P. J. Mucci, S. Browne, C. Deane, and G. Ho, "Papi: A portable interface to hardware performance counters," in *Proceedings of the department of defense HPCMP users group conference*, 1999, pp. 7–10.

[13] L. L. Pilla, C. P. Ribeiro, D. Cordeiro, C. Mei, A. Bhatele, Navaux, F. Broquedis, J. Mehaut, and L. V. Kale, "A Hierarchical Approach for Load Balancing on Parallel Multi-core Systems," in *Parallel Processing (ICPP), 2012 41st International Conference on*, 2012, pp. 118–127.

[14] L. L. Pilla, C. P. Ribeiro, P. Coucheney, F. Broquedis, B. Gaujal, P. O. A. Navaux, and J.-F. Méaut, "A Topology-Aware Load Balancing Algorithm for Clustered Hierarchical Multi-Core Machines," *Future Generation Computer Systems*, 2013.

[15] L. L. Pilla, "Topology-Aware Load Balancing for Performance Portability over Parallel High Performance Systems," Ph.D. dissertation, UFRGS; Université de Grenoble, Apr. 2014.

[16] P. Bedaride, A. Degomme, S. Genaud, A. Legrand, G. Markomanolis, M. Quinson, L. Stillwell, Mark, F. Suter, and B. Videau, "Toward better simulation of MPI applications on Ethernet/TCP networks," in *PMBS13 - 4th Intl. Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, Nov. 2013.

[17] H. Casanova, F. Desprez, G. S. Markomanolis, and F. Suter, "Simulation of MPI applications with time-independent traces," *Conc. Comp.: Pract. and Exp.*, vol. 27, no. 5, pp. 1145–1168, 2015.

[18] M. Noeth, P. Ratn, F. Mueller, M. Schulz, and B. R. de Supinski, "Scalatrace: Scalable compression and replay of communication traces for high-performance computing," *Journal of Parallel and Distributed Computing*, vol. 69, no. 8, pp. 696 – 710, 2009.

[19] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science, I. Ivanov, M. Sinderen, F. Leymann, and T. Shan, Eds. Springer International Publishing, 2013, vol. 367, pp. 3–20.

[20] S. S. Shende and A. D. Malony, "The tau parallel performance system," *Int. J. High Perform. Comput. Appl.*, vol. 20, no. 2, pp. 287–311, 2006.

[21] H. Aochi, A. Ducellier, F. Dupros, M. Delatre, T. Ulrich, F. Martin, and M. Yoshimi, "Finite difference simulations of seismic wave propagation for the 2007 mw 6.6 Niigata-ken Chuetsu-Oki earthquake: Validity of models and reliable input ground motion in the near-field," *Pure and Applied Geophysics*, vol. 170, no. 1-2, pp. 43–64, 2013.

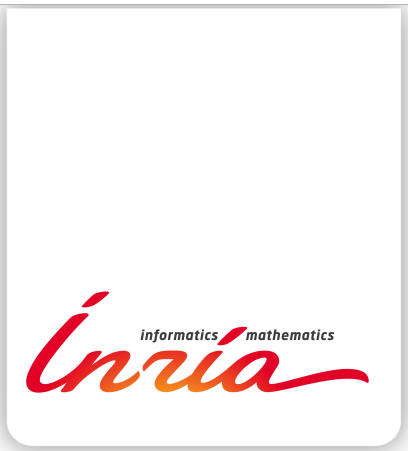
[22] G. Zheng, G. Kakulapati, and L. Kale, "Bigsim: a parallel simulator for performance prediction of extremely large parallel machines," in *Par. and Dist. Proc. Symp., 2004. Proceedings. 18th Int.*, apr 2004, pp. 78–.

[23] C. L. Janssen, H. Adalsteinsson, S. Cranford, J. P. Kenny, A. Pinar, D. A. Evensky, and J. Mayo, "A simulator for large-scale parallel computer architectures," *Int. J. of Dist. Syst. and Tech.*, vol. 1, no. 2, 2010.

[24] C. Engelmann, "Scaling to a million cores and beyond: Using light-weight simulation to understand the challenges ahead on the road to exascale," *Future Generation Computer Systems*, vol. 30, pp. 59 – 65, 2014.

[25] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls *et al.*, "The structural simulation toolkit," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, pp. 37–42, 2011.

[26] P. Velho, L. M. Schnorr, H. Casanova, and A. Legrand, "On the validity of flow-level tcp network models for grid and cloud simulations," *ACM Trans. Model. Comput. Simul.*, vol. 23, no. 4, pp. 23:1–23:26, Dec. 2013.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399