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

Rafael Keller Tesser[1(✉)], Lucas Mello Schnorr[1], Arnaud Legrand[2], Fabrice Dupros[3], and Philippe Olivier Alexandre Navaux[1]

[1] Informatics Institute, Federal University of Rio Grande do Sul, Porto Alegre, Brazil
{rktesser,schnorr,navaux}@inf.ufrgs.br
[2] Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble, France
arnaud.legrand@imag.fr
[3] BRGM, Orléans, France
f.dupros@brgm.fr

Abstract. Finite difference methods are commonplace in scientific computing. Despite their apparent regularity, they often exhibit load imbalance that damages their efficiency. We characterize the spatial and temporal load imbalance of Ondes3D, a seismic wave propagation simulator. We reveal that this imbalance originates from the nature of the input data and from low-level CPU optimizations. Such dynamic imbalance should therefore be quite common and is intractable by any static approach or classical code reorganization. An effective solution, with few code modifications, combines domain over-decomposition and dynamic load balancing (e.g., with AMPI), migrating data and computation at the granularity of an MPI rank. It generally requires a careful tuning of the over-decomposition level, the load balancing heuristic and frequency. These choices are quite dependent on application and platform characteristics. In this paper, we propose a methodology that leverages the capabilities of the SimGrid framework to conduct such study at low experimental cost. It combines emulation, simulation, and application modeling that requires minimal code modification and yet manages to capture both spatial and temporal load imbalance, faithfully predicting its overall performance. We compare simulation and real executions results and show how our strategy can be used to determine the best load balancing configuration for a given application/hardware configuration.

Keywords: Load balancing and over-decomposition · Performance prediction · Simulation · Geophysics FDM application

1 Introduction

The Ondes3D seismic wave propagation simulator [7], developed by computational science researchers at the French Geological and Mining Research Bureau

F.F. Rivera et al. (Eds.): Euro-Par 2017, LNCS 10417, pp. 192–205, 2017.
DOI: 10.1007/978-3-319-64203-1_14

(BRGM), is a typical iterative application tailored for homogeneous HPC platforms. Unfortunately like many other similar applications, Ondes3D suffers from scalability issues [6] 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, Ondes3D presents both non-trivial spatial and temporal load imbalance.

The performance of Ondes3D could be improved by partially rewriting it [13] to run on modern heterogeneous HPC platforms. The undesired side-effect is that computational science researchers, the people who actually understand the physics behind the code, often become incapable to contribute anymore. An alternative way to improve performance with less intrusive modifications is to rely on domain over-decomposition and runtimes that support dynamic process migration, as implemented by Charm++ [11]. In the specific case of legacy iterative MPI applications, one may employ Adaptive MPI (AMPI) [10], 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 task that can be dynamically migrated when necessary. The migration phase is triggered when the `MPI_Migrate` operation is called. The load balancer decides the new task mapping based on previously collected load measurements.

Such porting has already been applied to Ondes3D in a previous work [12], enabling spatial load imbalance to be dynamically mitigated. However, anticipating performance gains when using such adaptive HPC runtimes is usually difficult. Finding the best configuration for 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. Such parameter tuning is platform-specific, 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 codes. This methodology accelerates the evaluation of over-decomposition coupled with dynamic load balancing with almost no modification of the target application. Our approach relies on the SMPI emulation and trace replay mechanisms of SimGrid [5] to simulate the computation/communication behavior of the application and to mimic the behavior of the load balancing heuristics. 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 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 our validation is conducted only with Ondes3D and two earthquake scenarios (Chuetsu-Oki and Ligurian), we believe that it has nothing specific to it. Our strategy could be applied to any iterative MPI application.

Section 2 presents a detailed analysis of the spatial and temporal load imbalances in Ondes3D. Section 3 details our evaluation workflow and its validation procedure. In Sect. 4, we compare our method against real executions, and confirm the usefulness of our simulation for load balancing parameter tuning.

Section 5 presents related work on simulation-based tools, justifying our choices. Section 6 concludes the paper, listing major contributions and future work. More details on experiments, analysis, and simulation workflow can be found in an extended version at https://hal.inria.fr/hal-01391401.

2 Ondes3D: A Typical Imbalanced MPI Code

Ondes3D is a simulator to conduct seismic hazard assessment at regional scale. It approximates the differential equations governing the elastodynamics of rock medium using finite-differences methods (FDM). The 3D domain is statically partitioned in cuboids, as depicted in Fig. 1. Each iteration (see Fig. 2a) corresponds to a given time step and consists in calling three macro kernels (`Intermediates`, `Stress`, and `Velocity`) that apply a series of micro kernels (example in Fig. 2b) to the whole domain. Message passing consists in asynchronous neighborhood communications intertwined with the three macro kernels. There is no global barrier, each process evolves asynchronously up to some extent.

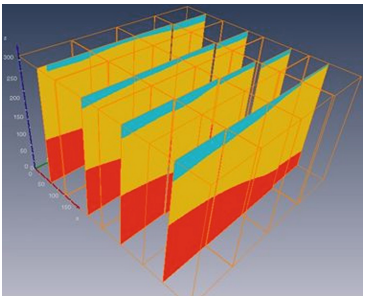


Fig. 1. 3D rock medium, with a 4×4 domain decomposition; each process calculates a cuboid.

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

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

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

}
```

(a) 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);
}
```

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

Fig. 2. The Ondes3D application: (a) the three macro kernels of the main loop, with intertwined neighborhood communications; (b) and the `CPML4` micro kernel.

Ondes3D suffers from load imbalance that limits its scalability despite its regularity (cuboids have the same geometry; code is always the same). Extra-computation dealing with boundary conditions has been previously identified [6]

as the main source of spatial imbalance. In Sect. 2.1, we report another source of spatial imbalance caused by the heterogeneous rock substrate. Temporal imbalances had been overlooked due to the regular shape of the code. In Sect. 2.1, we show that temporal imbalance is stronger than the spatial one. Evidences of its origin are related to low level optimizations taking place inside the CPU.

We have used a Mw6.3 earthquake workload [2] identified as Ligurian. Code compilation uses GCC 6.1.1 with `-O3` and PAPI [14] instrumentation. While we report results only for this setup, we have observed the issues with other workloads, CPUs (Xeon X3440, X5650, E5-2630, and i7 4600M), and compilers.

2.1 Identifying New Sources of Load Imbalance

Spatial Imbalance Due to Heterogeneous Rock Medium. Figure 3 depicts a 16×16 domain decomposition where each cell in the cartesian grid represents one of the 256 processes, each in charge of a cuboid subdomain. The color in the heatmap indicates the total computational load per process during the first iteration, before the main earthquake event that originates in the $(13, 5)$ subdomain coordinate. 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.

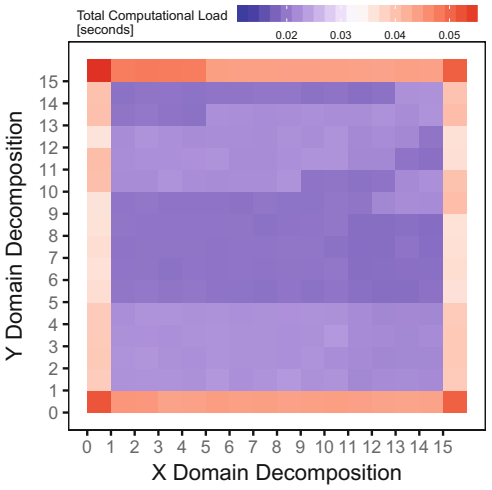
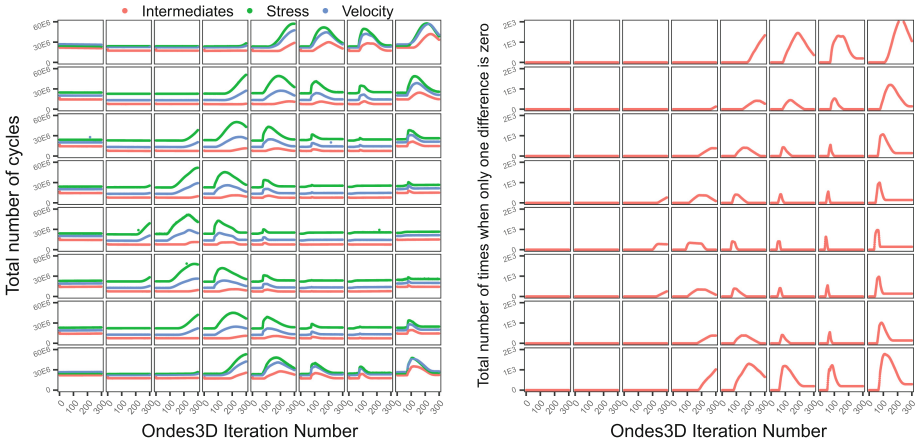


Fig. 3. Spatial imbalance for the first iteration represented by a color gradient for each rank in a 16×16 grid (256 processes). (Color figure online)

Temporal Imbalance Due to Low-Level CPU Optimizations. The Ondes3D code does not exhibit any structure (convergence loops, refinements, thresholds) that could lead to an evolution of computation load along simulation iterations. There are conditional branches (see Fig. 2b), but they are related solely to absorbing boundary conditions. Yet, as illustrated in Fig. 4a, 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



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

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

Fig. 4. Load imbalances for the Ligurian workload: (a) spatial load imbalance; (b) temporal load imbalance for three kernels; and (c) `CPML4` substrate values evaluating to distinct values.

(in the vertical axis of each box) the total number of cycles (`PAPI_TOT_CYC`) per macro kernel 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.

We take the `CPML4` kernel (see Fig. 2b) to explain the origin of this dynamic computational cost. `CPML4` represents well all the 24 small inlined kernels. It is called by the `Intermediates` macro kernel that iterates over the cuboid subdomain with three nested loops. For each subdomain's cell, the `CPML4` kernel is called nine times with different parameters, resulting in many calls for each process and time step. The values `dx` and `dt` are constants, while variables `x1`, `x2`, `x3`, and `x4` represent how the rock medium state unfolds along the iterations.

Let us consider the `x1`, `x2`, `x3`, and `x4` arguments of the `CPML4` kernel (Fig. 2b). They are used in the `return` statement, considered by the FPU for arithmetic evaluation. We instrumented the `CPML4` kernel to count how many times per time step and per process these differences are equal to zero (let us name these numbers $n^0_{2,1}$ for `x2-x1` and $n^0_{4,3}$ for `x4-x3`). The difference $|n^0_{2,1} - n^0_{4,3}|$ (Fig. 4b) perfectly correlates with the computational load change (Fig. 4a) and with the growth of the branch miss-prediction counters. Intuitively, this value measures how often only one of the two differences is zero. This hypothesis has been confirmed with a manual instrumentation of the `CPML4` kernel, recording its duration for each call (in cycles) along with the result of the two differences (`x2-x1` and `x4-x3`). 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. All other small inlined kernels share the same structure of `CPML4`. It is thus the aggregated contribution of all these small additional cycles that generates the temporal load variation.

2.2 Need for Dynamic Load Balancing: The AMPI Approach

Modeling and predicting the Ondes3D load imbalance is hard, as it strongly depends on the initial and evolving conditions of the earthquake simulations. Even if we could rewrite Ondes3D to allow uneven domain decomposition, some periodic data/computation re-balancing would still be required to cope with temporal load imbalance. We thus employ a simpler approach by mixing load balancing at runtime with over-decomposition, using Charm++'s Adaptive MPI [10] (AMPI). This framework enables over-decomposition, i.e., dividing the problem domain in more tasks than the number of available cores. Each task becomes a user-level thread suitable for migration. Load balancing heuristics, sensitive to load variations from the near past, can periodically redistribute load.

Porting from MPI to AMPI requires three application changes. First, there should be no global or static variables, to avoid data sharing among tasks. Second, Pack-and-Unpack functions are necessary to make data migrations possible. And third, the application must call `MPI_Migrate` to indicate when the application has no active communications or open files, and is ready for load balancing.

2.3 Costly Tuning of Load Balancing Parameters in Real Platforms

Many parameters influence the effectiveness of the load balancing. Some **load balancing heuristics** are more scalable than others (e.g., centralized *vs* distributed). The **level of over-decomposition** defines the granularity for the load balancer. As over-decomposition increases, we also increase the communication cost. At some point, such cost exceeds the benefit of load balancing. Likewise, **the number of processors** is a critical parameter in the overall performance. Finally, fine-tuning the **frequency of load balancing** is essential to obtain good performance since frequent calls might become overhead, hiding any load balancing benefits. 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.

Using real executions to evaluate the load balancing benefits present several difficulties. The optimal configuration often depends on application and platform characteristics. Running the same earthquake simulation many times at scale on a production system solely to determine such parameters is both resource and time consuming. To overcome this, we propose a lightweight simulation workflow to avoid the burden of real executions. Performance gains are evaluated with few code changes (even before AMPI porting), and the application needs to be executed only once. Such approach saves development and evaluation time.

3 Simulated Adaptive MPI (SAMPI)

Our workflow relies on SimGrid's SMPI, which offers two key features we have built upon. First, SMPI's flexibility allows to study MPI applications either in *emulation mode* or through *trace-replay*. In emulation, unmodified MPI applications are sequentially executed on top of the simulator, in a controlled way. In trace replay, the events of an MPI application are replayed on top of the simulator, in a small fraction of the time it takes to finish a normal run at full scale. Second, SMPI builds on the hybrid flow-level network models of SimGrid [4] that allow to faithfully model network contention, which is essential in our context.

SMPI has been modified to simulate AMPI in three ways. (1) The API is extended with the non-standard `MPI_Migrate` function both in the emulation mode (to generate an event in the trace) and in the trace replay. When replaying with load-balancing, this function calls the `MPI_Barrier` function, the load balancing heuristic to define a new mapping, and simulates all task migrations. (2) We have manually extracted and slightly adapted two centralized load balancers (LB) by hand: **GreedyLB** and **RefineLB**. We removed internal references to the original Charm++ implementation, making sure that the heuristic remains intact. A few trace replay routines also had to be modified to collect the load data that is fed to these heuristics. (3) The migration payload is estimated by trapping `malloc` function calls in emulation, which is prone to migration cost understimation. We rely on SimGrid's contention-aware network models when sending the data of the migrated task from its original location to its destination.

Tracing one workload requires to run the code for real, hence it takes 3–5 h with SMPI's emulation on a laptop. Then, while exploring parameters, it can be replayed many times with SAMPI and an LB configuration (frequency, heuristic). Each configuration simulation takes only a few minutes on a laptop.

4 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 is 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 adhoc 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. We use this performance metric, to compare reality and simulation both qualitatively and quantitatively.

Real measurements have been collected in 16 nodes of the Parapluie cluster (part of Grid'5000 [3]). Each node has two 12-core 1.7 GHz AMD Opteron 6164 HE processors, interconnected through a 20 G Infiniband 4x QDR network.

We tested two very different earthquake scenarios in Ondes3D. The first one is the Mw6.6 Niigata *Chuetsu-Oki* (2007) from Japan [1]. Running the full simulation (6000 time steps) takes an unreasonable amount of time, mainly because

many runs are needed to obtain statistically significant results. We limited this simulation to the first 500 time steps to keep a reasonable experimental time. We also reduced the number of cells to $300 \times 300 \times 150$. The second simulated scenario is the same used in Sect. 2, with $500 \times 350 \times 130$ cells.

4.1 Validation: Comparing SAMPI (Simulation) Against AMPI

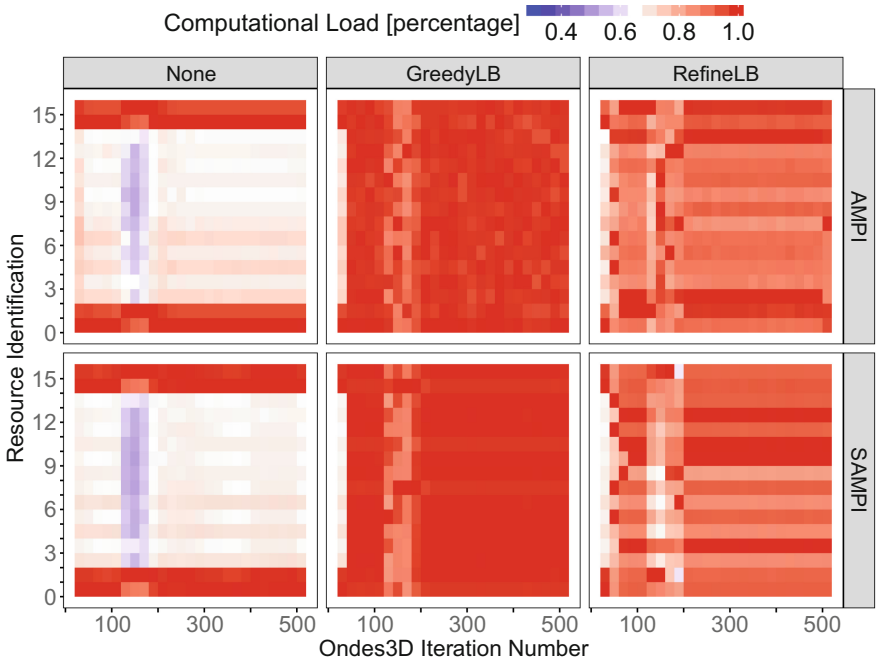
In our validation experiments, we fix the domain decomposition to 64 tasks (always mapped to 16 processes) and call `MPI_Migrate` every 20 time steps. From our experience, this configuration is relatively good and allows to focus our evaluation on sound scenarios. The comparison of SAMPI with AMPI for situations without load balancer, with GreedyLB and with RefineLB, is depicted for the two workloads: Chuetsu-Oki in Fig. 5, and Ligurian in Fig. 6.

Per-Process Computational Load Analysis. The heatmaps in Figs. 5a (Chuetsu-Oki) and 6a (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), with a given load balancer (no load balancing on the left column, Greedy in the center, and Refine on the right). The real and simulated load distribution are very similar, showing the ability of our workflow to capture the complex behavior of AMPI in simulation.

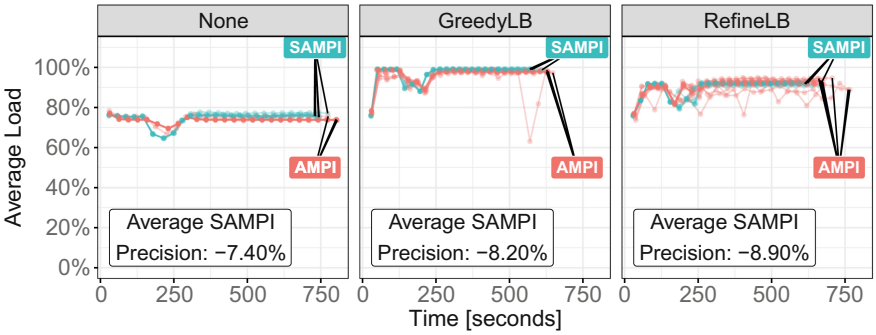
Figure 5a shows that for Chuetsu-Oki, the case without load balancing leads to many underutilized resources (white and bluish regions). Both LB seem to significantly improve this situation by making processes 2 to 13 receive more load. GreedyLB achieves a much better load balancing than RefineLB (being more conservative) and this is visible in simulation as well as in real execution traces. The load structure for the Ligurian workload is quite different (see Fig. 6a). 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. We observe a much more even computational load across processes but not as good as for the Chuetsu-Oki workload.

The heatmap views are based on one run for each case. Any new execution (either real or in simulation from a new trace) leads to slightly different outcomes. Thus, focusing on the load of a given core at a given time-step is not really meaningful. From such view, it seems that GreedyLB is the best choice from the load balancing perspective, but communication (both from the application and load balancer) should also be taken into account. In the following, we provide makespan analyses using the average load as a function of the execution time.

Average Load and Makespan Comparison Analysis. The plots in Figs. 5b (Chuetsu-Oki) and 6b (Ligurian) depict the evolution of the average load for each core. This metric (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



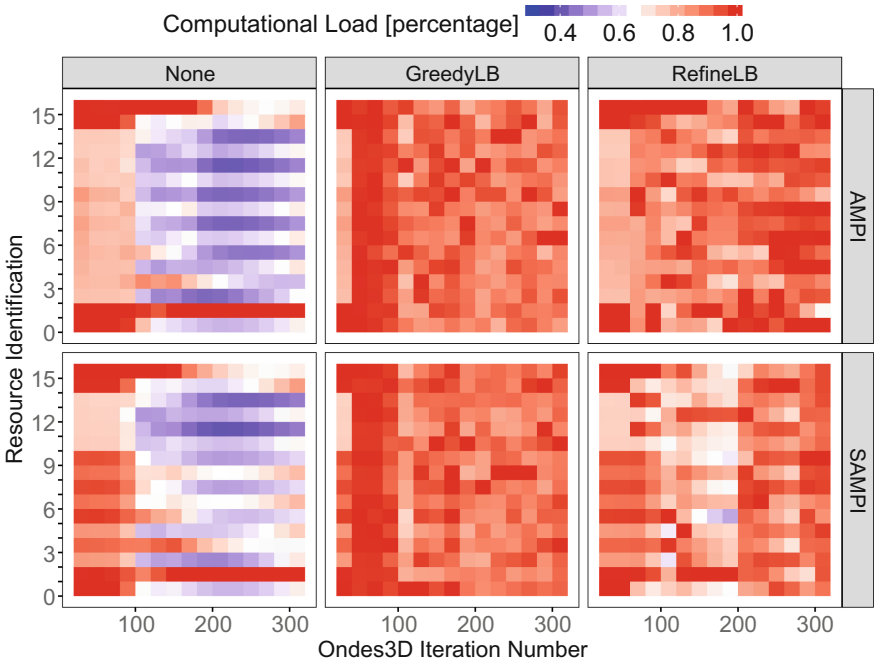
(a) Per-process computational load.



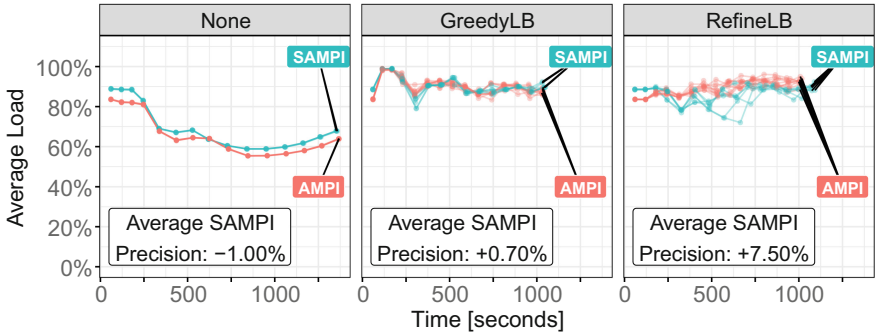
(b) Average Load and makespan.

Fig. 5. Comparison of SAMPI (simulation) against AMPI (reality) for the Chuetsu-Oki workload; the top row shows six heatmaps (no LB, Greedy, and Refine) illustrating the computation load (color gradient) for each iteration and all 16 processes; the bottom row shows the average aggregated load along time, with the makespan of multiple runs. (Color figure online)

moments when the metric is computed (when `MPI_Migrate` starts, at the end of the LB interval); lines show the trend. Horizontal facetting indicates the metric without load balancing, with GreedyLB and with RefineLB.



(a) Per-process computational load.



(b) Average Load and makespan.

Fig. 6. Comparison of SAMPI against AMPI for the Ligurian workload. (Color figure online)

For the Chuetsu-Oki workload (Fig. 5b), GreedyLB performs better than RefineLB, both in simulation as in real life. One could expect GreedyLB to be worse instead, due to the larger amount of migrations. 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, we see that SAMPI is slightly too optimistic across several runs. That being said, such inaccuracy would not affect our choice of load balancer. There is a significant variability

in real executions (perfect isolation is tough to achieve on a cluster), being generally larger than in the simulations. Simulation variability comes from the use of different inputs to trace replay. For the Ligurian workload (Fig. 6b), as on the previous scenario, both simulation and real life have similar load unfolding, except for RefineLB, where SAMPI is slightly more pessimistic than real life.

Our simulation mimics in a realistic way the evolution of the load distribution of real executions, which is one of the main aspects we are trying to obtain. There remains some minor inaccuracies in absolute time prediction: ≈9% for all configurations of the Chuetsu-Oki workload, and varying from ≈1% to ≈8% in the Ligurian. We are currently investigating their origin. Yet, since the trends remain correct, this does 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.

4.2 Tuning Load-Balancing Parameters with Simulation

We investigate the parameter space of AMPI using our SAMPI workflow. We measure four configurations for load balancing interval; and five levels of over-decomposition. We focus on the Ligurian workload, since it is much larger than the Chuetsu-Oki and parameter tuning is likely to be more useful.

The Influence of Load Balancing Frequency. We measure the makespan of Ondes3D with different load balancing intervals. A call to `MPI_Migrate` is present for each task at the end of every time step. During the simulation with SAMPI, we control and 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 influence of the load balancing frequency (horizontal axis) on the makespan (vertical axis) of a 16×4 task configuration. In this setting, it turns out that LB frequency has no or little influence in the performance attained when using GreedyLB or RefineLB. Even though GreedyLB balances the load carelessly whereas RefineLB is much more conservative, the communication performance of the system is sufficiently good to hide the migration costs.

The Influence of Decomposition Level. Another important performance affecting parameter is the over-decomposition level. The influence of over-decomposition on the makespan of Ondes3D, when calling `MPI_Migrate` every 20 time steps, is depicted in Fig. 7 (right plot). The average makespan (vertical axis) is shown as a function of five over-decomposition configurations (horizontal). In the absence of load balancing (None), over-decomposing is, as expected, generally deterring since this creates extra-communication between tasks. Yet having more and smaller tasks allows for a better redistribution of the load. The RefineLB sweet spot is reached with a 16×8 decomposition (≈13% gain over the original version). However, for GreedyLB the decomposition level should be

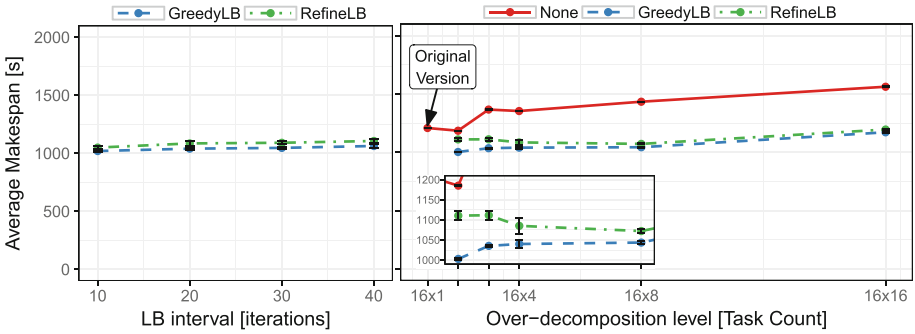


Fig. 7. Simulated makespan predictions for the Ligurian earthquake simulation with (left) four load balancing intervals (in number of iterations) and (right) with six over-decomposition levels (1, 2, 3, 4, 8, and 16) on 16 cores.

as small as possible (which leads to ≈19% gain over the original version), which is again explained by the fact that its careless migrations scale very badly. In the end, the 16×2 GreedyLB configuration is slightly better than the 16×4 RefineLB configuration but exhibits quite different load balancing behaviors.

From a series of similar simple studies using SAMPI, it appears that, for this application, RefineLB executed every 20 time steps with an over-decomposition level of 8 provides, in general, a decent performance and gracefully handles a larger number of nodes. This parameter combination has been tested a real execution of the Chuetsu-Oki simulation on a 12-node cluster (288 cores) at BRGM. We obtained an ≈36% faster execution than the original unbalanced execution. Further tuning can be done at low cost using SAMPI to guide the analyst toward a better configuration.

5 Related Work

The SAMPI workflow we propose mostly depends on two factors. First, a faithful model of modern HPC networks and MPI implementations are 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 helpful to select the approach most suited to the resources at hand. There is a plethora of simulation tools to study MPI applications [5] and at least four of them support both modes and could thus have been modified: BigSim [17], SST/Macro [15], xSim [8], and SimGrid [5] (through SMPI). BigSim is part of Charm++, thus supporting the AMPI applications simulation, such as our Ondes3D code. Although linked to Charm++, BigSim is uncapable to change the load balancing parameters during trace replay and this would require major code modifications. SST-Macro allows both trace replay through the DUMPI module and emulation through skeletonization. Although SST-macro is flexible with many network models, including flow-based ones, its emulation support still seems

unsufficiently mature to run an application as complex as Ondes3D. Finally, xSim mostly focuses on extreme-scale executions and its validity remains questionable at small scale [9]. Furthermore, the source code of xSim is currently unavailable.

For this work, we therefore chose to rely on the free software SimGrid, whose SMPI interface allows both emulation and trace replay of MPI applications. SMPI leverages SimGrid's thoroughly validated flow communication models [16], while also accounting for specific characteristics of MPI implementations [5]. Hence, SMPI allows us to collect accurate execution traces from emulation, and its replay mechanism allows us to quickly simulate one execution many times.

6 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) A validated simulator called SAMPI that simulates over-decomposition and AMPI load balancing. This simulator is integrated in the open-source SimGrid framework, and allows the fast and faithful exploration of different load balancing scenarios from a single execution trace. (c) A sensibility analysis showing both the importance of activating a load balancer (≈20–30% gains), and the rather low influence of specific load balancing parameters in the Ondes3D makespan.

As future work, we plan to build on other Ondes3D characteristics to understand how spatial aggregation and trace extrapolation can be used together to further accelerate the simulations.

Acknowledgements. We thank CAPES/Cofecub 764-13, FAPERGS/Inria ExaSE, FAPERGS Green-Cloud, CNPq 447311/2014-0, CNRS/LICIA Intl. Lab, the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E Project, grant 689772. Some experiments were carried out at the Grid'5000 platform (https://www.grid5000.fr), with support from Inria, CNRS, RENATER and several other organizations.

References

1. Aochi, H., Ducellier, A., Dupros, F., Delatre, M., Ulrich, T., Martin, F., Yoshimi, M.: 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 Appl. Geophys. **170**(1–2), 43–64 (2013)

2. Aochi, H., Ducellier, A., Dupros, F., Terrier, M., Lambert, J.: 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. p. 6, September 2011
3. Balouek, D., et al.: Adding virtualization capabilities to the Grid'5000 testbed. In: Ivanov, I.I., Sinderen, M., Leymann, F., Shan, T. (eds.) CLOSER 2012. CCIS, vol. 367, pp. 3–20. Springer, Cham (2013). doi:10.1007/978-3-319-04519-1_1
4. Bédaride, P., et al.: Toward better simulation of MPI applications on Ethernet/TCP networks. In: Jarvis, S.A., Wright, S.A., Hammond, S.D. (eds.) PMBS 2013. LNCS, vol. 8551, pp. 158–181. Springer, Cham (2013). doi:10.1007/978-3-319-10214-6_8
5. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. Parallel Distrib. Comput. **74**(10), 2899–2917 (2014)
6. Dupros, F., Do, H.T., Aochi, H.: On scalability issues of the elastodynamics equations on multicore platforms. In: International Conference on Computer Science, Procedia Computer Science, p. 9. Elsevier, Barcelone, June 2013
7. Dupros, F., Martin, F.D., Foerster, E., Komatitsch, D., Roman, J.: High-performance finite-element simulations of seismic wave propagation in three-dimensional nonlinear inelastic geological media. Parallel Comput. **36**(5–6), 308–325 (2010)
8. Engelmann, C.: Scaling to a million cores and beyond: using light-weight simulation to understand the challenges ahead on the road to exascale. Future Gener. Comput. Syst. **30**, 59–65 (2014)
9. Engelmann, C., Naughton, T.: A network contention model for the extreme-scale simulator. In: Press, A. (ed.) 34th IASTED International Conference on Modelling, Identification and Control (MIC) (2015)
10. Huang, C., Lawlor, O., Kalé, L.V.: Adaptive MPI. In: Rauchwerger, L. (ed.) LCPC 2003. LNCS, vol. 2958, pp. 306–322. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24644-2_20
11. Kalé, L., Krishnan, S.: CHARM++: a portable concurrent object oriented system based on C++. In: Proceedings of OOPSLA 1993, pp. 91–108. ACM Press (1993)
12. Keller Tesser, R., Lima Pilla, L., Dupros, F., Navaux, P., Mehaut, J.F., Mendes, C.: Improving the performance of seismic wave simulations with dynamic load balancing. In: International Conference Parallel, Distributed and Network-Based Processing (2014)
13. Martinez, V., Michéa, D., Dupros, F., Aumage, O., Thibault, S., Aochi, H., Navaux, P.O.A.: Towards seismic wave modeling on heterogeneous many-core architectures using task-based runtime system. In: SBAC-PAD. IEEE Computer Society (2015)
14. Mucci, P.J., Browne, S., Deane, C., Ho, G.: PAPI: a portable interface to hardware performance counters. In: Proceedings of the Department of Defense HPCMP Users Group Conference, pp. 7–10 (1999)
15. Rodrigues, A.F., Hemmert, K.S., Barrett, B.W., Kersey, C., Oldfield, R., Weston, M., Risen, R., Cook, J., Rosenfeld, P., CooperBalls, E., et al.: The structural simulation toolkit. ACM SIGMETRICS Perform. Eval. Rev. **38**(4), 37–42 (2011)
16. Velho, P., Schnorr, L.M., Casanova, H., Legrand, A.: On the validity of flow-level TCP network models for grid and cloud simulations. ACM Trans. Model. Comput. Simul. **23**(4), 23:1–23:26 (2013)
17. Zheng, G., Kakulapati, G., Kale, L.: Bigsim: a parallel simulator for performance prediction of extremely large parallel machines. In: Parallel and Distributed Processing Symposium, Proceedings, 18th International, p. 78, April 2004