Simulation with Skeletons of Applications Using Dimemas

postprint version

C. Camarero, C. Martínez, and J. L. Bosque. 2019. Simulation with skeletons of applications using dimemas. In Proceedings of the 16th ACM International Conference on Computing Frontiers (CF '19). Association for Computing Machinery, New York, NY, USA, 274–278.

https://doi.org/10.1145/3310273.3322827

This is the author's version of the work. It is openly available to comply with institutional requirements. Not for redistribution.

Esta versión esta disponible para cumplir exigencias institucionales, no para su distribución.

Cristóbal Camarero[1], Carmen Martínez[1], Jose Luis Bosque[1]
[1]Computer Science and Electronics Department, University of Cantabria
cristobal.camarero, carmen.martinez, joseluis.bosque@unican.es

December 17, 2021

Abstract

Large computer systems, like those in the TOP 500 ranking, comprise about hundreds of thousands cores. Simulating application execution in these systems is very complex and costly. This article explores the option of using application skeletons, together with an analytic simulator, to study the performance of these large systems. With this aim, the Dimemas simulator has been enhanced with the capability of simulating application skeletons. This enhancement allows simulating the skeleton of Lulesh, an application with 90k processes in a single day. In addition, it also generates traces, which is of great value to validate skeletons and simulations.

1 Introduction

The *Structural Simulation Toolkit (SST)* is a structural event-driven simulator developed at Sandia National Laboratories [**?**]. The SST/macro simulator is the macro-scale (system level) component of SST. It simulates MPI applications, typically written in C, with variable number of cores and nodes, detailed network model and different process allocation policies. The simulation is done using application skeletons [**?**]. This means that the computation phases of the application are replaced by numerical models of their duration, and only the communications are actually simulated. Employing application skeletons provides flexibility, scalability and accuracy. Since the skeleton uses the same arguments of the application, a well calibrated skeleton can represent an application independently of the input size. In addition, it also enables simulations for an arbitrary number of processes, even beyond the size of existent machines.

Dimemas [**?**] is an offline simulator, part of the Barcelona Supercomputing Centre (BSC) toolchain, that uses MPI traces to predict the event times at the target machine. This tool has models for point to point communication and resource contention among others. It is commonly utilized together with other two BSC's tools: Extrae and Paraver. Extrae makes MPI traces from real executions by means of a MPI interposition library, thereby constructing the input for Dimemas. Paraver allows visualization of MPI traces in many diverse ways, and it is employed to analyze the results of Dimemas in depth.

In this paper we consider the suitability of simulating with skeletons of MPI applications using an analytical simulator such as Dimemas. With this aim, we have developed *Dimemas online*, which constitutes an extension of the functionality of Dimemas that allows the processing of application skeletons. As it will be shown, Dimemas online increases the flexibility of the simulator compared to trace based execution, as it is not necessary to feed the simulator with a trace obtained from a real execution. In this way, it is straightforward to modify the parameters of the skeleton to simulate any instance of an application. Moreover, Dimemas allows the obtention of simulation traces, thus it is possible to generate traces also from skeleton driven simulations. These traces can subsequently be compared with real traces to validate both the skeleton and the simulator.

Lulesh has been the application to test Dimemas online, and some experiments with it are also presented. In these, we have carried out a study of the accuracy and scalability of Dimemas online using the skeleton of Lulesh. Our baseline will be the results obtained with SST/macro, as this simulator also uses skeletons. Therefore, all the results achieved with Dimemas online will be compared with those obtained with SST/macro with an equivalent configuration. It must be taken into account that SSt/macro 4.0 has been used in all the experiments. Although the simulator supports parallel simulation, our experiments do not take this advantage. Experiments show that Dimemas online gives an accuracy similar to SST/macro. But Dimemas online scales much better, since it needs less memory, reaching up to 90k processes in a computer with 96 GB of main memory. On the other side, SST/macro is faster than Dimemas online. However, it should be taken into account that Dimemas online is a functional extension of an analytical simulator while SST/macro has been conceived as a structural simulator from its design.

This paper is organized as follows. In Section 2 the methodology for simulating using application skeletons is explained. Dimemas online is described in Section 3. Next, in Section 4, some experimental results are shown and analysed. Finally, Section 5 presents the main conclusions of this work.

2 Skeleton Based Methodology

Prior to making skeleton simulations of applications some steps must be followed, regardless of whether Dimemas online, SST/macro, or other simulator is used. Indeed, with the proper macros, a calibrated skeleton can work with different simulators. The main requirement is a *calibrated skeleton* of the application to be simulated. This is obtained after completing two steps: skeletonisation and calibration.

1. The *skeletonisation* of an application implies identifying blocks of code which correspond to computation phases. These code blocks are substituted by functions, that calculate the computation time based on the parameters of the application. To achieve this, the code has to be instrumented, and the model for the computation times is calculated by interpolating the obtained values.

2. The *calibration* is the procedure by which the terms in the arguments devised in the skeletonisation are chosen. This usually requires running the application on a real machine. We used Sandia's toolchain (lwperf and Eiger) to generate the models. This procedure is explained in [**?**] and summarized in Figure 1.

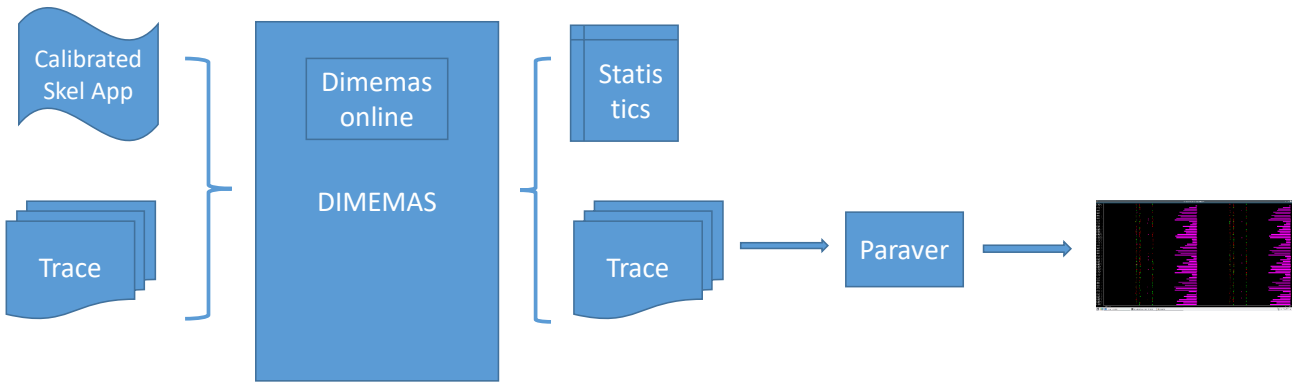

Figure 1: Obtention of a calibrated skeleton.

It is also necessary to have information about the target machine to be simulated. This is represented through parameters like the number of sockets, cores and their frequency, or the memory bandwidth and latency. Also it is possible to model the interconnection network, configuring its topology, the routers, or the bandwidth and latency of the links. In our experimentation, the real machine, used to calibrate the skeleton, and the target machine only differ in the number of compute nodes. If the cores of the real and target machines were different, it would be necessary to correct the models.

Once the skeleton and the target information is available, the skeleton is linked against the simulator (Dimemas online or SST/macro). The target information together with the application arguments are provided to the simulator through a configuration file. The execution of the resulting binary gives the estimations of the application execution on the target machine plus other simulator-dependant outputs.

3 Dimemas Online

3.1 Implementation

The Dimemas simulator is an analytical simulator based on the use of MPI traces. With Dimemas online, simulations can also be performed by using application skeletons. To make this possible, the compilation of Dimemas has been extended such that, in addition to generating a Dimemas binary, it also generates a shared library containing the simulator with the online functionality. This Dimemas library can then be linked against an application or skeleton, resulting on the final executable. In addition to making an online module, it has been necessary to make some changes to other parts of Dimemas. In particular in the reading of data and the processing of communications.

The basic procedure is the same as in SST/macro: the main routine of Dimemas spawns one thread per simulated process, each executing the main routine of the skeleton. These threads behave as the processes of the original application and they can use the function `SKELETON_Compute` to implement the CPU models, as explained before. The communication between these threads is done by the implementation of some MPI functions inside Dimemas. When a thread calls an MPI function (1) it waits until it has no pending actions to be processed by the main routine of Dimemas, (2) it potentially makes data transfers among the threads in the same way as the real MPI function would do, (3) it creates Dimemas' actions in order to include the time of MPI calls in the total simulated time and the corresponding entries in the trace file, (4) it waits for commits or barriers if necessary, and (5) it frees internal structures that no other thread requires. The waits in (1) and (4) take a large portion of the time, as will be noted latter. The Dimemas' actions are processed by Dimemas' main routine in the same way as if they would had been read from a tracefile, this includes the use of analytical models to estimate the duration of the MPI calls in the target machine.

A partial implementation of the MPI library and runtime has been incorporated into Dimemas, so an application can run correctly, both as an application and as a skeleton. In the current prototype, the most frequently used MPI functions have been implemented. Among them are point-to-point communications (blocking and non-blocking), some collective calls (MPI_Barrier, MPI_Bcast, MPI_reduce, MPI_Allreduce, ...) and communicators. Therefore, many HPC applications such as Lulesh, HPCCG [**?**] or CoMD [**?**] can be simulated without any changes. Note that the time associated to executing this partial implementation in the host machine is completely unrelated to the time that Dimemas estimates for the same MPI call to take in the target machine.

3.2 Utilization

Figure 2 shows a sketch of the methodology used to perform a simulation with Dimemas online. As it can be seen, both a trace or a skeleton can be used to perform the simulation. If this is the case, the skeleton has to be linked to the Dimemas library mentioned before, obtaining the final executable.

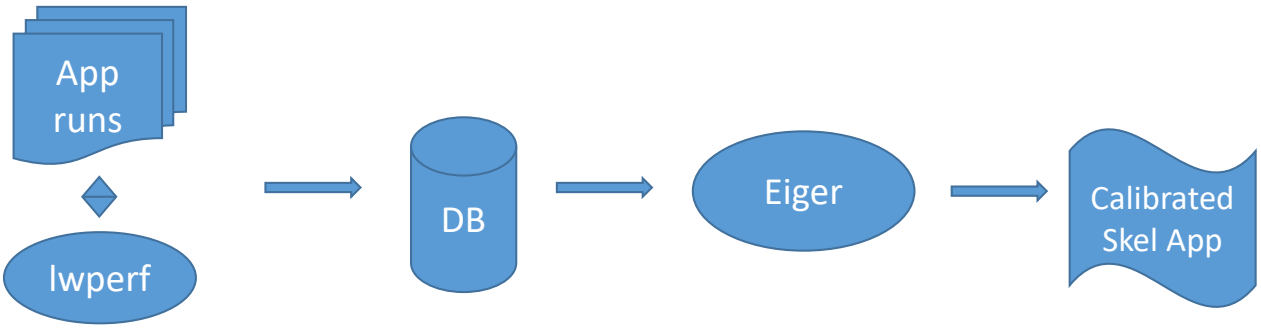

Figure 2: Simulation methodology with Dimemas online.

Then, Dimemas statistics can be obtained. A configuration flag can force a trace of the simulation to be created, which can then be visualized using Paraver. Moreover, this trace can be converted to Dimemas format and then be simulated in Dimemas with different parameters. In Figure 3(a) a Paraver view of a Lulesh trace can be seen, and Figure 3(b) shows the trace generated by the simulation of the Lulesh skeleton with Dimemas online.

These figures show the state of the application during two iterations. Each row corresponds to an MPI process and the abscissa is a measure of time. The color indicates the state of the process at a given time, black meaning the process is computing, and other colors indicating the execution of some MPI call. For example, the pink color is associated to the MPI_Allreduce collective call, it starts when the process enters the call and finishes when all other processes enter and perform the reduction. The time of the reduction itself is negligible, and the processes end the call almost simultaneously, as the separation from pink to black is a vertical line.

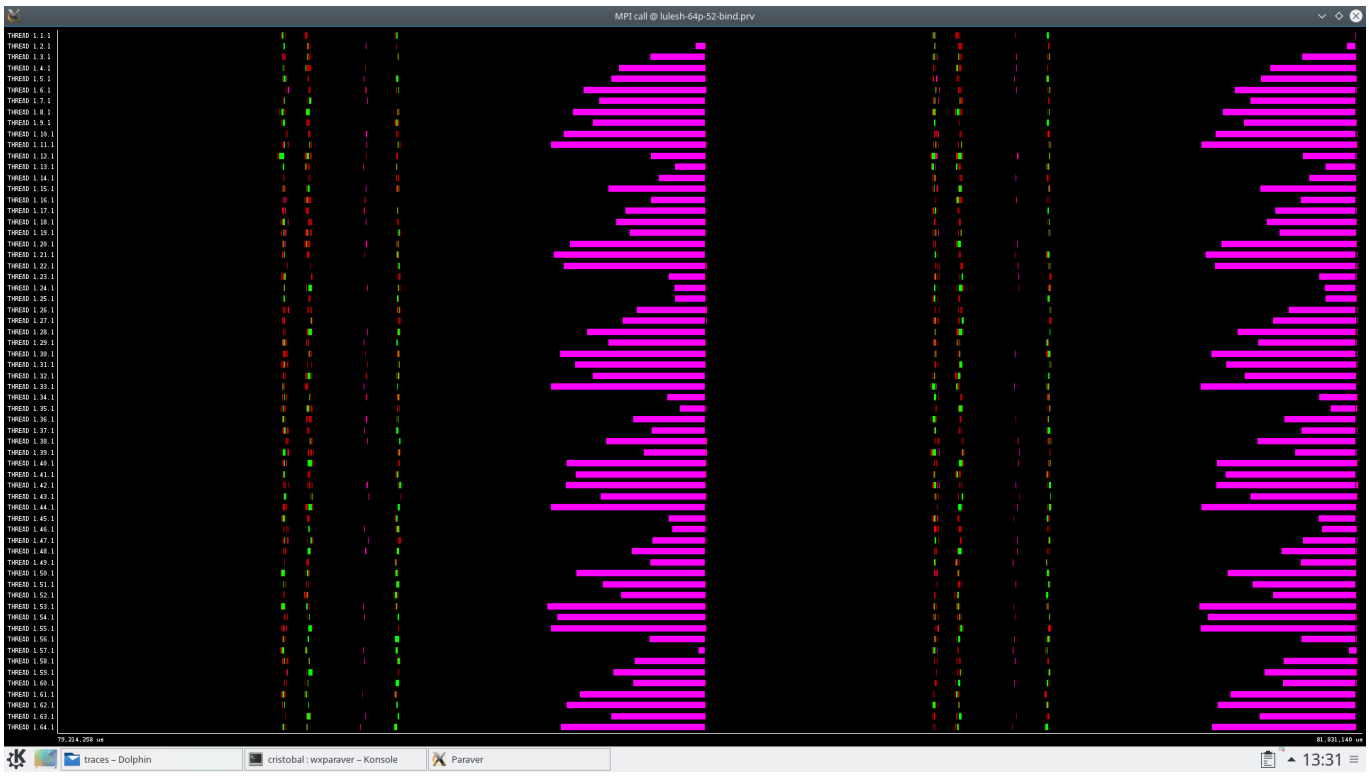

Figure 3: Trace of a Lulesh execution on the ThunderX mini-cluster.

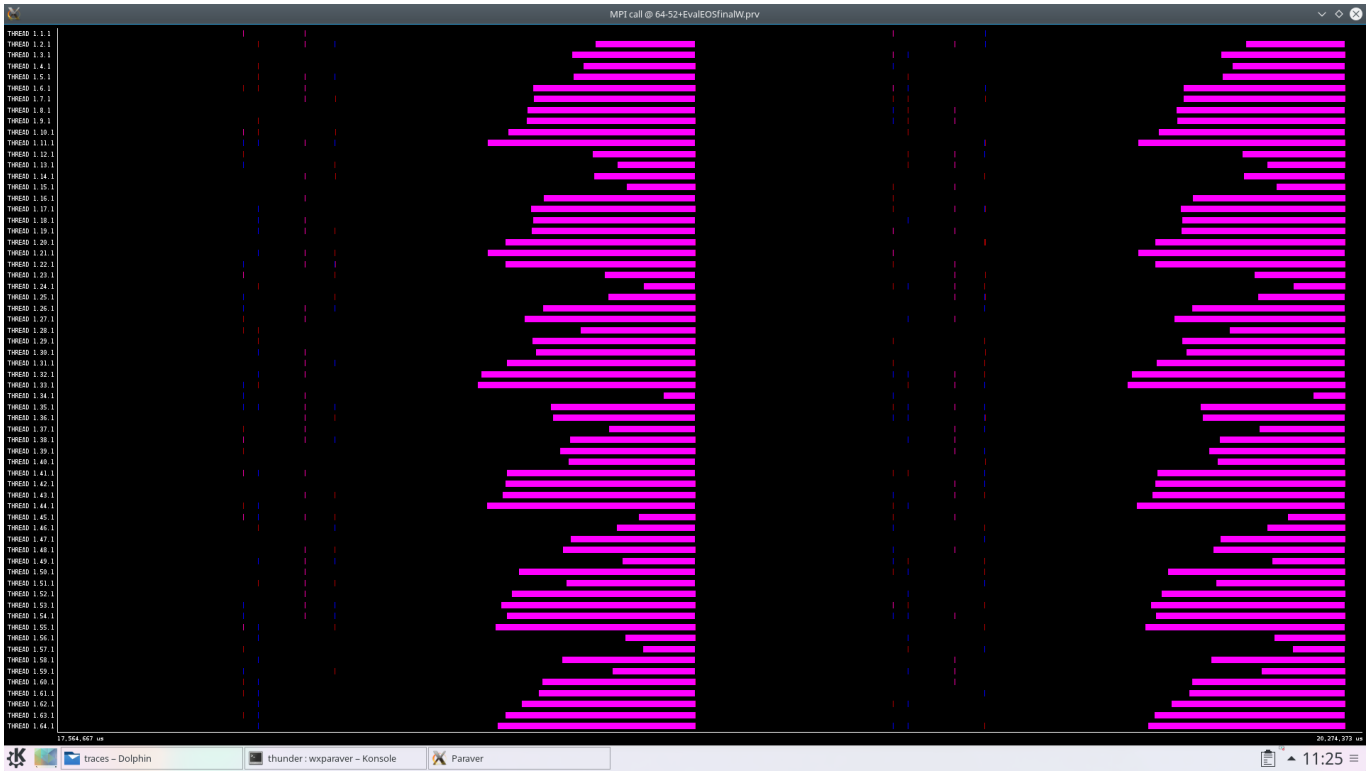

Figure 4: Trace of a Lulesh skeleton simulation with Dimemas online using Eiger models.

4 Experimentation

This section presents the results of the experiments that validate Dimemas online. All of them have been done using a skeleton for Lulesh, which is explained in Subsection 4.1. We have focused on the accuracy and the scalability of the simulations; these metrics are described in Subsection 4.2. The results are shown in Subsections 4.3, 4.4 and 4.5.

4.1 Lulesh Skeleton Application

Lulesh stands for Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics [**?**]. This application has as inputs:

- `-s SIZE` Creates a working set of a total of $SIZE^3 \times numprocs$ elements to be distributed uniformly among the processes.
- `-i MAXITERATIONS` Is a limit to the number of iterations to execute.

Also, the number of processes must be the cube of an integer. The skeleton of the Lulesh application has been calibrated in the following points:

- Total number of elements: 210^3 and 420^3.
- Number of processes: 1, 8, 64 and 216.
- Iterations per execution: 25.

All the executions for the calibration have been run in the Mont Blanc project prototype ThunderX [**?**]. The models provided by Eiger were all adopted except the one provided by the function EvalEOSforElemens since it was not accurate ($R < 0.65$). Thus, it was modeled by linear regression of the data set, in which every point has a weight inversely proportional to the time it consumes.

4.2 Performance Metrics

Three figures of merit are considered in our experiments:

- Relative *accuracy* of the simulation predictions. We will compare the *predicted time* given by the simulator, that is, the time the complete execution of the application would take in the target machine.
- *Host time*, which is the time employed by the simulator to perform a simulation.
- *Memory usage*, which is the maximum amount of memory used performing a simulation.

All these figures of merit are considered and compared using the simulator SST/macro. Thus, simulations of the Lulesh skeleton with SST/macro (labelled as SST/macro) and Dimemas online (labelled as Dimemas online) are compared.

All the experiments are carried out considering strong scaling, thus the number of elements to compute (the workload of the application) is fixed to 1050^3 while the number of processes varies. The number of processes ranges from $216 = 6^3$ to $15,625 = 25^3$, since the application needs a cube as the number of processes. For all the executions, the number of iterations is fixed to 5.

However, as it will be seen, SST/macro is not able to perform simulations over $3,375 = 15^3$ processes due to memory restrictions. As it will be shown in subsection 4.5, Dimemas online is able to perform simulations well above $15,000$ processes.

4.3 Time predictions of Dimemas online

In Figure 5 predicted time of Lulesh with 1050^3 elements varying the number of processes is shown. The same skeleton calibrated in the ThunderX cluster is simulated with SST/macro and Dimemas online. As it can be seen, the predicted time for both simulators is practically the same. That is, for this skeleton application, both simulators provide the same predicted times. This would suggest that, if the application does not stress the communications, then an analytical simulator such as Dimemas would be enough to obtain accurate time predictions. Also, it seems that the most part of the accumulated error comes from the calibration of the skeleton, which includes the obtention of the CPU models.

It is also interesting to note that the ThunderX cluster in which the calibration of the skeleton has been carried out has 4 nodes, each one with 96 cores. However, with Dimemas online it has been possible to simulate a machine with 1000 nodes, providing a total of 96k cores.

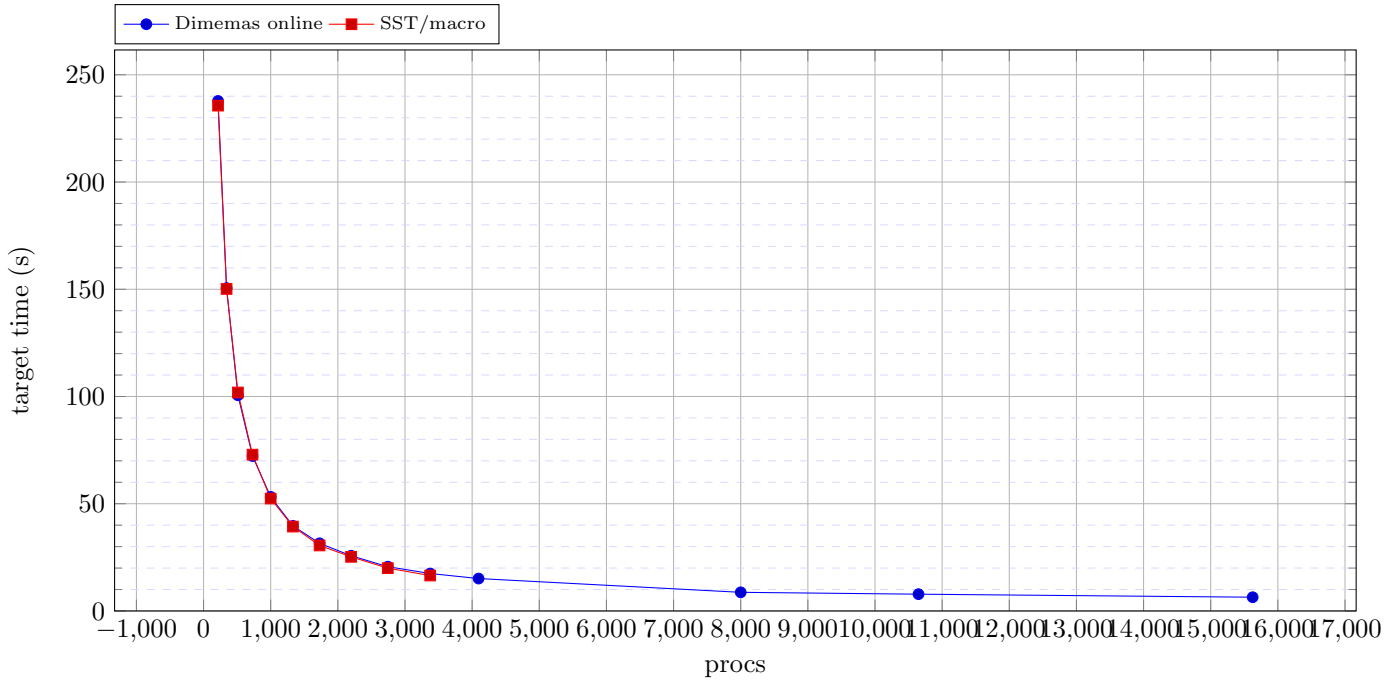

Figure 5: Predicted time of Lulesh simulation with 1050^3 elements and ThunderX as target machine.

4.4 Host time and Memory Usage

Figure 6 shows the host Time of Lulesh with 1050^3 elements varying the number of processes in the ThunderX cluster, simulated with SST/macro and Dimemas online. Clearly, SST/macro is faster than Dimemas online. In this particular case, the host time for Dimemas online is between 2× and 2.5× slower than SST/macro. However, the absolute values of the execution time in both cases are very reasonable for this kind of simulations, in the order of a few minutes. Therefore, the improvements provided by Dimemas online can overcome in many cases the inconvenience of the increased response time.

Dimemas online is somewhat slower than Dimemas, when the latter simulates the traces generated by the former. This should not be surprising because of two reasons. As explained in Section 3, Dimemas online is executing more code, including the implementation of MPI calls and the fragments of the application comprised in the skeleton. The second reason is that threads are continuously waiting for the result of others, resulting in time regions where the kernel of Dimemas is idle.

Figure 7 presents the host memory of Lulesh with 1050^3 elements varying the number of processes in the ThunderX cluster, for both simulators SST/macro and Dimemas online. As it can be observed, the use of memory by SST/macro suffers sudden changes and is its main limitation against Dimemas online. This abrupt changes make it impossible to determine *a priori* if a specific simulation is possible or not.

On the contrary, Dimemas online makes a more smooth and regular use of memory, which allows the estimation of the simulation capacity regarding the memory of the host. Furthermore, Dimemas online requires less host memory to carry out the same simulations. This means that, with the same host node configuration, Dimemas online is able to simulate a larger number of processes than SST/macro. In particular, it suggests that much bigger simulations could be done, which will be explored in Subsection 4.5.

4.5 Scalability of Dimemas Online

In this subsection we explore the feasibility of making larger simulations with Dimemas online. In this case, simulations have been performed in Mare Nostrum 4, where available memory is $96GB$. Table 1 shows host time, host memory and predicted time of Lulesh with 1050^3 elements varying the number of processes in the ThunderX cluster simulated with Dimemas online. Experimental results show that Dimemas is capable of simulating up to 90k MPI processes in a single node of the Mare Nostrum 4.

5 Conclusions

This paper has considered the use of application skeletons to simulate large systems. At least for the application considered in the paper, the skeleton application of Lulesh emulates correctly the application behaviour. Also, this

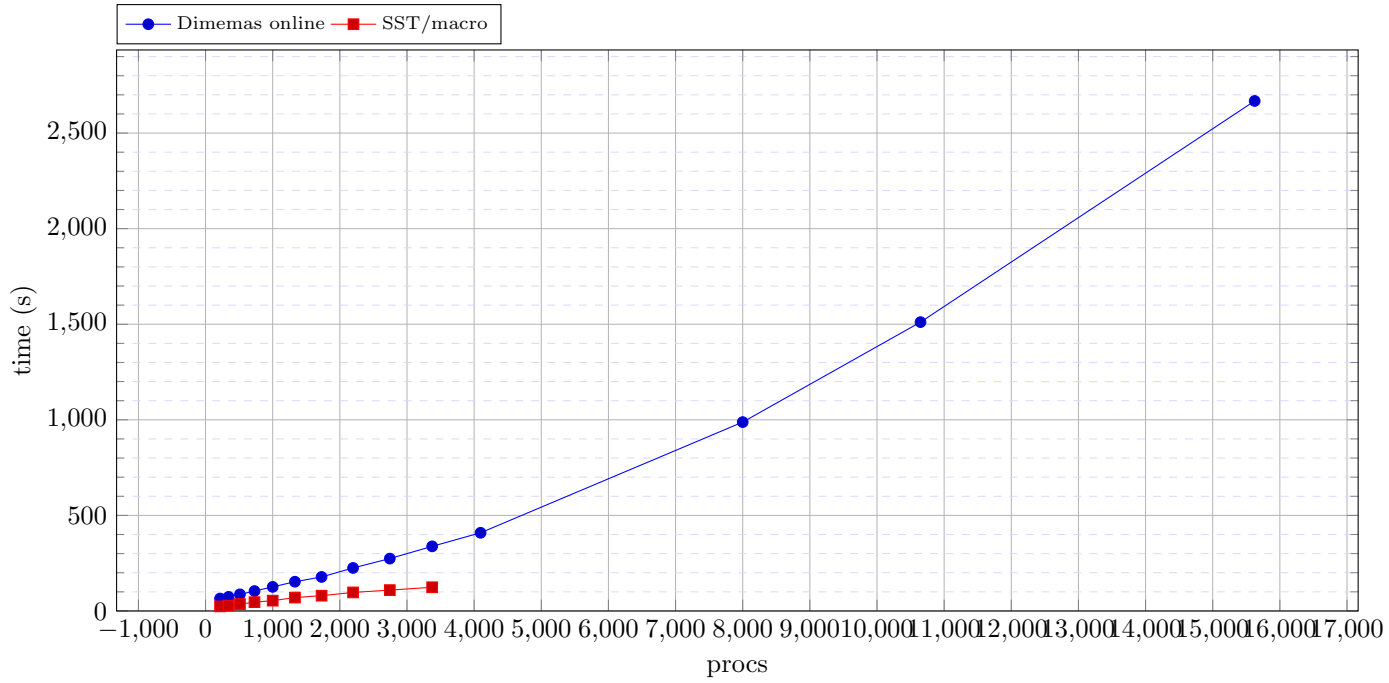

Figure 6: Host time of Lulesh simulation with 1050^3 elements and ThunderX as target machine.

Simulated Procs	**Host Time** (s)	**Host Memory** (GB)	**Predicted Time** (s)
15,625	2,729	25.9	6.399
32,768	9,640	33.8	5.979
64,000	38,396	41.8	6.819
91,125	85,910	47.9	7.822

Table 1: Host time, host memory and predicted time of Lulesh with 1050^3 elements in the ThunderX cluster simulated with Dimemas online in Mare Nostrum 4.

simulation methodology significantly reduces the response time and improves its scalability. Dimemas online has been developed, as an extension of the Dimemas simulator, in such a way that it allows the execution of skeletons and the obtention of traces from them. For example, there it is possible to generate a trace for $15,625$ processes, which for five iterations in Lulesh, has a size of 1.6GB. It is possible to convert this trace to Dimemas format and perform a simulation. As it has been shown, Dimemas online is slower than SST/macro, but it is still within a reasonable range considering the size of the simulation. On the contrary, Dimemas online uses much less memory, improving the simulation scalability, which has allowed to perform simulations up to 90k MPI processes. Moreover, the predictions made by both simulators are practically the same, when it is possible to make the comparison. This suggests that an analytical simulator is enough to make simulations of huge systems, provided that the skeleton application is well-calibrated.

Acknowledgments

The authors would like to thank Judit Jimenez, Marc Josep Fabrego and Jesus Labarta for their helpful support and to Esteban Stafford for his proofreading labor. This work has been partially supported under grants the Spanish Ministry of Science, Innovation and Universities under contract TIN2016-76635-C2-2-R (AEI/FEDER, UE), the Mont-Blanc project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671697 and Juan de la Cierva–Formación contract (FJCI-2017-31643) by the Ministerio de Ciencia, Innovación y Universidades of Spain.

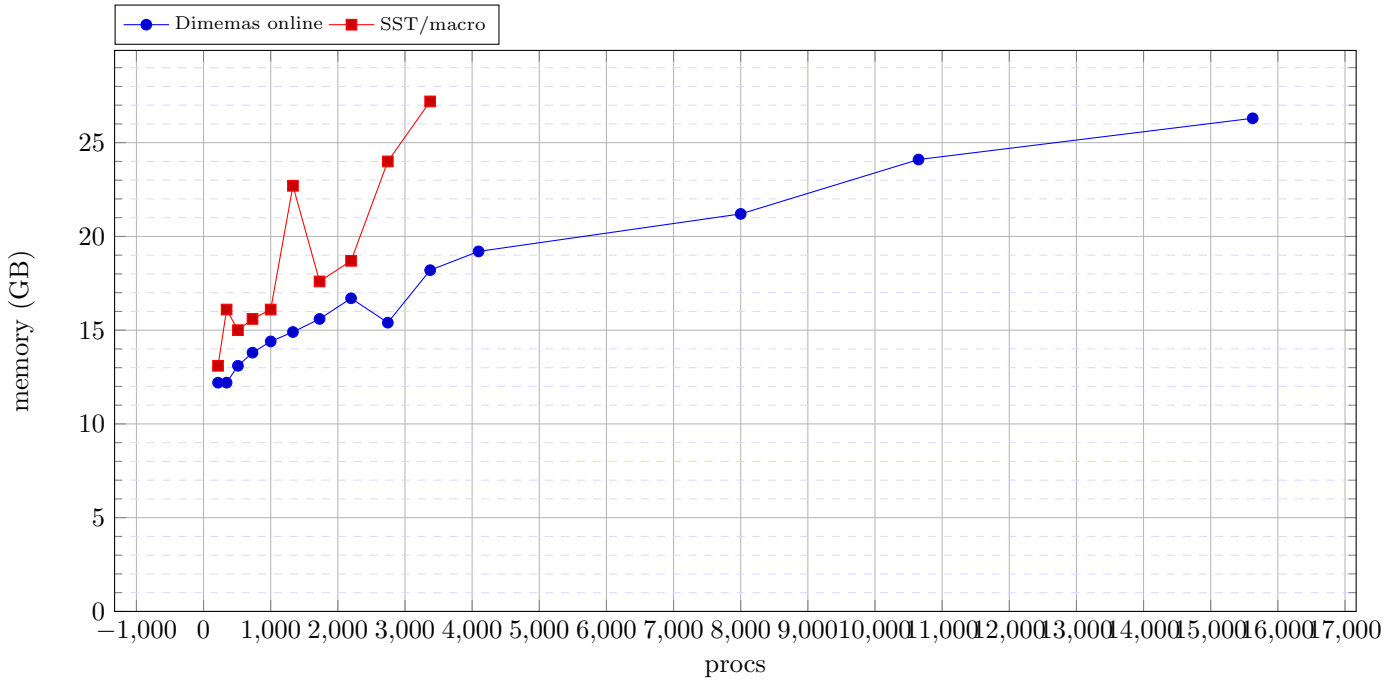

Figure 7: Host memory of Lulesh simulation with 1050^3 elements and ThunderX as target machine.