sOMP: Simulating OpenMP Task-Based Applications with NUMA Effects

Idriss Daoudi, Philippe Virouleau, Thierry Gautier, Samuel Thibault, Olivier Aumage

sOMP: simulating OpenMP task-based applications with NUMA effects

Idriss Daoudi[1,2][0000−0003−2425−8359], Philippe Virouleau[1,2], Thierry Gautier[2], Samuel Thibault[1], and Olivier Aumage[1]

[1] INRIA, LaBRI, Université de Bordeaux, IPB, CNRS, Bordeaux, France
[2] LIP, ENS-Lyon, UCBL-Lyon 1, INRIA, Lyon, France

Abstract. Anticipating the behavior of applications, studying, and designing algorithms are some of the most important purposes for the performance and correction studies about simulations and applications relating to intensive computing. Often studies that evaluate performance on a single-node of a simulation don't consider Non-Uniform Memory Access (NUMA) as having a critical effect. This work focuses on accurately predicting the performance of task-based OpenMP applications from traces collected through the OMPT interface. We first introduce TiKKi, a tool that records a rich high-level representation of the execution trace of a real OpenMP application. With this trace, an accurate prediction of the execution time is modeled from the architecture of the machine and sOMP, a SimGrid-based simulator for task-based applications with data dependencies. These predictions are improved when the model takes into account memory transfers. We show that good precision (10% relative error on average) can be obtained for various grains and on different numbers of cores inside different shared-memory architectures.

Keywords: OpenMP tasks · NUMA architecture · Performance modeling · Simulation

1 Introduction

Simulation tools are of significant interest in the field of application development. They allow, among other things, to understand whether an application has been designed efficiently, and to test limits and sensitivity of hardware characteristics for components such as CPUs and memory buses. They can be an important predictive tool for evaluating existing and non-existing systems in procurements.

OpenMP is probably the most commonly used programming language for shared-memory paradigms in HPC applications. On these architectures, the increasing number of cores leads to the need for a complex memory hierarchy, which implies Non-Uniform Memory Access (NUMA) timings from each core to each memory location. To benefit the most from such a platform, it is thus not enough that several blocks of operations are made to execute in parallel on different cores. It is also essential that these blocks of operations are executed on CPU cores close to the memory node in which the data they access is located.

The placement of data, therefore, has a primary effect on the performance of the application. This makes simulating task-based applications on shared memory architectures a challenging endeavor, since these different NUMA-related effects must be captured accurately to perform a reliable simulation.

This work targets OpenMP applications composed of tasks with data dependencies, such as dense linear algebra routines (Cholesky, QR...). Task-based applications are indeed increasingly common, but overheads in runtime systems implementations may limit the applicability of the task model [12]. It is fundamentally important to be able to exhibit the precise performance of an OpenMP task-based application without artifacts from the runtime implementation: in addition to performance profiling [7,11], simulation is a way to achieve this goal. In previous works [25,24], we explored the simulation of task-based scheduling on heterogeneous architectures, which is now used as a reliable tool for scheduling experiments [1]. This work differs in that it targets the complications of task-based OpenMP programs that use architectures with large core counts.

To retrieve the information necessary to replay and predict an application's performance, we developed a tool called TiKKi[1] on top of the OMPT API, that records all profiling events required to construct a task graph. We then created a simulator named sOMP[2], using the SimGrid framework, to address the problem of predicting the performance of a task-based parallel application on shared memory architectures. sOMP finely models the platform to simulate data transfer contentions accurately, and takes data locality into account to predict the execution time from various scenarios of data placements. Although SimGrid is designed to simulate distributed memory architectures, we adapted the possibilities offered by this tool to execute tasks on simulated shared-memory NUMA platforms. To summarize, this paper presents the following contributions:

1. We introduce TiKKi an OMPT-based tracing tool to extract the high-level information necessary for the simulation;
2. We introduce a modeling of a NUMA machine using the SimGrid framework;
3. We develop sOMP, an implementation of the simulator that leverages the S4U API tool from SimGrid;
4. We propose a model to refine the simulations that uses the link parameters of the simulated platform to model the effects of contention and data access;
5. We show a small relative error of the simulation for various architectures (Intel and AMD) while taking into account the locality effects of the data. The simulation itself is found to be much faster than real executions.

2 Related work

Many simulators have been designed for predicting performance in a variety of contexts, in order to analyze application behavior. Several simulators have been developed to study the performance of MPI applications on simulated platforms,

[1] `https://gitlab.inria.fr/openmp/tikki/-/wikis/home`
[2] `https://gitlab.inria.fr/idaoudi/omps/-/wikis/home`

such as BigSim [30], xSim [10], the trace-driven Dimemas tool [13], or MERPSYS [6] for performance and energy consumption simulations. Some others are oriented towards cloud simulation like CloudSim [4] or GreenCloud [18].

Other studies are oriented towards the simulation on specific architectures, such as the work by Aversa and al. [2] for hybrid MPI/OpenMP applications on SMP, and task-based applications simulations on multicore processors [22,24,15,26]. All these studies present approaches with reliable precision, but, as with Simany [16], no particular memory model is implemented.

Many efforts have been made to study the performance of task-based applications, whether with modeling NUMA accesses on large compute nodes [8,14], or with accelerators [25]. Some studies have a similar approach to our work, whether in the technical sense, like using SimGrid's components for the simulation of parallel loops with various dynamic loop scheduling techniques [20], or in the modeling sense, such as simNUMA [19] on multicore machines (achieving around 30% precision error on LU algorithm) or HLSMN [23] (without considering task dependencies). But to our knowledge, no currently available simulator allows the prediction of task-based OpenMP applications performances on NUMA architectures, while taking into account data locality effects. To build our simulator, it was necessary to develop new tools and models that employ an extraction process of OMPT traces, but also manage task dependencies, data locality, and memory access effects.

3 sOMP: simulating task-based OpenMP applications

Since our goal is to build a simulator for existing OpenMP task-based applications on multicore NUMA architectures, we will use two tools. First, the TiKKi tool leverages the OMPT API [9] to record events of a running OpenMP application. Secondly, the generated traces are then processed by the sOMP tool to perform an offline simulation on top of the SimGrid [5] generic engine. For this work, sOMP extends SimGrid with the modeling of NUMA architectures.

To perform the simulations, tasks and their dependencies are re-computed by collecting information contained in the post mortem execution trace generated by TiKKi. We then introduce a communications-based model to take into account NUMA effects, to produce improved simulations.

3.1 TiKKi: tracing with OMPT

OMPT [9] is the OpenMP API for performance tools integrated in OpenMP since its revision 5.0 [3]. OMPT allows developers to instrument tools with trace-based methodologies.

The libKOMP [29] OpenMP runtime has an embedded trace and monitoring tool, based on the work of de Kergommeaux et al. [17]. The tool, called TiKKi, was developed using the initial OMPT API [9] available in an older version of the LLVM OpenMP runtime with extensions. We have updated it to match the OMPT specification of the current standard [3]. TiKKi captures all

events required to construct the program's task graph, and records them to a file. It also enriches the recording with performance information. For instance, task attributes may contain locality information [29], and hardware performance counters may be registered, in addition to time, within specific events (task creation, task termination...). Hence, TiKKi can generate several output forms of execution traces: task graph as a `.dot` file, Gantt chart as an R script, or a specific file format for the simulations performed by sOMP. In the current OpenMP standard, it is impossible to recover the information about data size: we are doing this explicitly for the moment, but the standard could be improved to expose this information, the implementation is usually easy.

The structure of the execution trace is a sequence of parallel regions, where the events of each task are recorded. When TiKKi processes the trace, it generates a sequence of sOMP input files, one per parallel region. Each of these can then be simulated as a separate task graph.

3.2 Modeling of NUMA architectures with SimGrid

SimGrid [5]. The specific objective of SimGrid is to facilitate research in the field of programming and running parallel applications on distributed computing platforms, from a simple network running in a workstation to the computing grids. It provides the basic functionalities for the simulation of heterogeneous distributed applications in distributed environments.

The operating principle is as follows: an *actor*, *i.e.* an independent stream of execution in a distributed application, can perform several *activities*, such as computations or communications, on a *host*, representing some physical resource with computing and networking capabilities. Several *actors* can communicate, and all classical synchronization mechanisms such as barriers, semaphores, mutexes, and conditional variables are provided.

From a platform description point of view, SimGrid provides the building blocks for a detailed description of each element of a distributed system, such as the computing *hosts* mentioned above, *routers*, *links*..., but also the routing on the platform, *i.e.* which path is taken by communications between two hosts. These elements have arguments that allow configuration and tuning of the platform in order to simulate different scenarios.

NUMA architecture modeling. While SimGrid is initially designed for simulating applications running on distributed architectures, we divert its use to simulate NUMA platforms.

The approach to model these architectures is as follows. The CPU cores are considered to be computing units interconnected by a network of links. Cores are thus grouped into NUMA nodes and sockets according to the actual architecture topology and these groups are interconnected with links to ensure access to the memory.

The model we consider in this work does not take into account the memory topology exhaustively. The addition of even more architectural components

would result in better precision, but also contribute to increasing the complexity of the problem and the simulation time. We had to make a compromise between accuracy and cost of the simulation: we could simulate more architectural elements, and that would be precise and expensive, or simulate very few elements which would be inexpensive but imprecise.

Notably, we do not model the L1/L2 cache, because all data sizes considered in this work exceed L1/L2 cache sizes, and their behavior will thus be caught already well enough when measuring task execution time without contention. Therefore, we model a NUMA architecture using elements sketched in Figure 1a, and we employ the concepts defined by SimGrid to model these components.

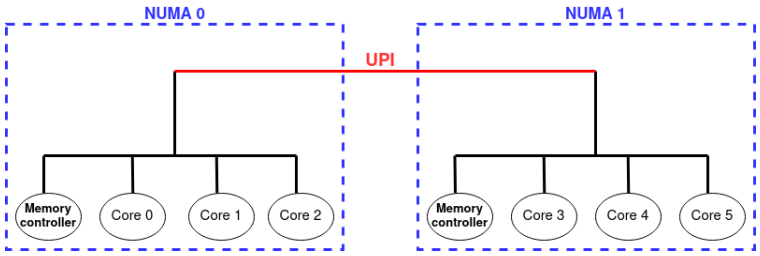

Fig. 1: NUMA machine modeling

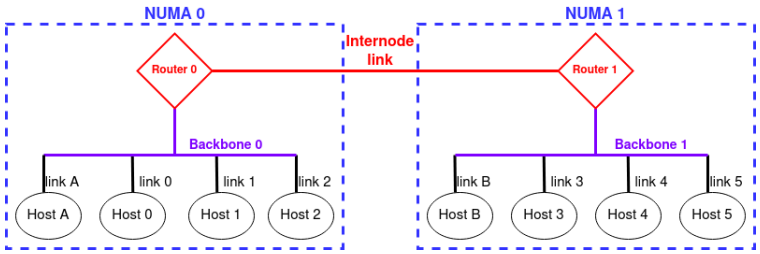

Fig. 2: Model using SimGrid components

Modeling with SimGrid SimGrid offers the possibility of describing a platform with the XML format. Any platform must contain basic essential elements such as *hosts*, *links*, *routers*, etc.. SimGrid requires the explicit declaration of the routes and links between these components in order to simulate communications between *hosts*.

We map the cores of a processor with SimGrid *hosts*. These represent a computing resource on which *actors* can run. From there, as depicted in Figure 2, we can model a NUMA node by a group of *hosts*, each having a *link* to a *backbone*-type link. The latter makes it possible to model the intra-node contention and connect the group of *hosts* to a *router*, which allows communications with other NUMA nodes. Regarding the memory controller, we chose to model it with a "fake" *host* (memory controller host) that does not perform any computation: this component only receives communications which simulate accesses to the ma-

chine's memory. Every *link* and *route* is referenced by an ID and can be tuned with parameters such as latency and bandwidth, allowing them to match the real machine's characteristics. We will discuss the tuning of those parameters in Section 5.3.

In the end, sOMP provides SimGrid with an assembly of simulated components (*hosts*, *links*, *backbones*, *routers*, *routes*) which mimics the actual architecture topology: for instance *routes* between *routers* represent real UPI/Infinity Fabric links. The properties of these components (notably the bandwidth) are then set to the values obtained on the native system. This allows, with a simple architecture description, to model different Intel/AMD platforms and obtain accurately simulated behavior as described in Section 5.3. The SimGrid network model used in this work is the LV08 default model.

3.3 Task-based applications simulation

Here we use two components to model a task-based application: first, only the task computational time is modeled, then the memory access costs are taken into account.

Task execution simulation. At runtime, we assume that a task mainly executes arithmetic instructions interleaved with memory instructions (typically load/store instructions). Let's assume that the execution time of a task t_i is decomposed into (ie, we neglect interactions between memory accesses and computations):

$$Time(t_i) = T_{Computations}(t_i) + T_{Memory}(t_i) \quad (1)$$

where $T_{Computations}(t_i)$ represents the time spent in the sequential execution of the task t_i with data local to the core executing the task. The term $T_{Memory}(t_i)$ represents the penalty due to a remote memory access on a NUMA architecture, which depends on the data location on the machine as well as the core that initiates the access. We consider that $T_{Computations}(t_i)$ is the execution time that we collect from a sequential execution, which thus does not suffer from NUMA effects. This time can also be collected from regression-based models [24].

As a first step, the sOMP model considers that $T_{Memory}(t_i) = 0$. Hence, we only simulate tasks computation time without any consideration for memory access and data locality. Such a model is well adapted for computation-bound applications.

Communications-based model. When the application is more memory-bound and is executed on NUMA architectures, the time to perform memory accesses should be taken into account. Our model considers the set of memory accesses made by a task, groups them by task operands (e.g. matrix tiles), and takes into account the machine topology and the capacity of links between components.

The grouping allows matching with SimGrid's programming model which is oriented towards distributed memory platforms: we model the task memory accesses with data transfers for the task operands, i.e., as SimGrid *communications*.

Since application tasks usually access to the content of all operands in an interleaved pattern, we make these communications concurrent and let SimGrid account for contention on the simulated links.

$T_{Memory}(t_i)$, the communications time of the task operands, which allows to improve the simulation accuracy, can then be written as:

$$T_{Memory}(t_i) = \max_{j=0}^{n-1} T_{Comm}(a_{i,j}) \tag{2}$$

where n is the number of memory accesses, $a_{i,j}$ is the j-th operand of task t_i and $T_{Comm}(a_{i,j})$ the time to transfer $a_{i,j}$ depending on its location and the core performing task t_i.

Moreover, the memory access modes (read, write, or read-write) allows us to take into account the cost of each communication differently: for read-write type operations, we double the communication time since these are composed of two distinguished transfers of the same tile.

To summarize, we express memory accesses to task operands as sets of concurrent SimGrid communications. SimGrid can then take into account the concurrency between the various communications of all tasks executing at the same time on the platform, with respect to the network characteristics, as depicted in Figure 2. This allows us to model the actual concurrency observed in real platforms [21]. SimGrid can thus determine for each communication how its duration $T_{Comm}(a_{i,j})$ gets affected by contention. These are then gathered by equation (2) into $T_{Memory}(t_i)$ which influences the simulated execution time according to equation (1). All of this is driven by the machine model and the defined latency and bandwidth values of the intra and inter-node links (obtaining those values will be discussed in section 5.2).

4 Implementation

To simulate the execution of task-based applications on the architecture model presented in Section 3.2, we need to develop a scheduling algorithm to manage task dispatching, execution, and dependencies on SimGrid *hosts*, with support of memory accesses for the communications-based model.

4.1 sOMP architecture

Since we exploit a trace file from a sequential execution of the application, we first need a parser to extract all the useful pieces of information contained in the generated file with the TiKKi tool in .rec format (from GNU Recutils). This file gives details on the executed tasks and provides their name, submission order, dependencies, logical CPU number and memory node on which the task was executed, submission/start/end time, the nature of memory transfers performed by the task, the data on which these operations happened, and their size.

After parsing the trace file, we proceed with inserting tasks in a submission queue (FIFO) that the sOMP scheduler handles. The scheduler submits

the tasks for execution by the simulated cores (hosts). The scheduler's task submission works according to two constraints: tasks must be ready, i.e., all their dependencies have been satisfied, and *hosts* workers must be idle, i.e., they are not currently executing another task. We use a centralized task queue for now which is similar to the one performed by a typical OpenMP runtime. Other scheduling policies can be tested in the future to try to improve the application performances relating to that field. We do not use the SimDAG (deprecated) and disk support of Simgrid since they do not allow us to finely control data transfers and interactions on the memory bus.

On each simulated core, a SimGrid actor (called *worker*) picks tasks one by one for simulation. The worker first simulates the memory accesses of the task: for every operand access, it triggers a message with the corresponding size (in bytes). The worker then waits for the completion of the transfer of all messages, which will increase SimGrid's internal clock, taking into account the latency and bandwidth of the traversed links and the contention induced on those links by concurrent accesses. The worker then simulates the task's execution by advancing the internal clock of SimGrid by a time equal to the task's real execution time, obtained from the TiKKi trace. Once the execution of a task is completed, the worker is responsible for activating the submission of the successors of the finished task to the scheduler, if all their dependencies have been satisfied.

4.2 Managing data locality

In the communications-based model, we store the NUMA node number on which each data allocation and initialization task was executed, and thus the NUMA node on which the data was effectively allocated. Since the other (computation) tasks will need to access those pieces of data, their NUMA locations are crucial to properly model the accesses.

When modeling the access to an operand with a communication, we not only define a payload size corresponding to the size of the operand, but also specify the source and recipient of the communication. This corresponds to modeling data accesses according to their location: the communication is performed between the memory controller *host* of the NUMA node where the operand was effectively allocated, and the core *host* that executes the task. Notably, if the core is in the NUMA node where storage is assigned, the communication will take place only on the local backbone, thus modeling the reduced contention.

5 Evaluation

The KASTORS [27] benchmark suite has been designed to evaluate the implementation of the OpenMP dependent task paradigm, introduced as part of the OpenMP 4.0 specifications. It includes several benchmarks. The experiments presented here are based on the PLASMA subset of the KASTORS benchmark suite, which provides three matrix factorization algorithms (Cholesky, LU, QR) extracted from the PLASMA library [28]. Experiments with the KASTORS benchmarks were performed on two machines:

- dual-socket **Intel Xeon Gold 6240**, 24.75MB L3 cache, 36 cores, **CascadeLake** microarchitecture with 1 NUMA node per socket, and 18 cores per NUMA node;
- dual-socket **AMD EPYC 7452**, 128MB L3 cache, 64 cores, **AMD Infinity** architecture with 4 NUMA nodes per socket, and 8 cores per NUMA node.

5.1 Methodology

In order to evaluate our simulator performance, we carry out various tests with the KASTORS benchmarks on the machines presented above. We choose different matrix sizes and different tile sizes in order to observe the accuracy of our simulator when confronted with a variety of scenarios.

To measure the reliability of the simulations by comparing simulation time (T_{sim}) with real execution time (T_{native}), we do not consider the absolute values of the metric, but set a metric that defines the precision error of sOMP compared to native executions: $PrecisionError = (T_{native} - T_{sim})/T_{native}$. Therefore, when the precision error is positive, it means that we "under-simulate" the actual execution time, in other terms our prediction is optimistic. A negative precision error means that we "over-simulate", hence a pessimistic prediction.

5.2 Latency and bandwidth measurements

To model a NUMA machine, providing the link's latency and bandwidth corresponding to the real values in the architecture is essential. As stated before, we consider that all of our memory transfers only involve the L3 cache and the DRAM, since all the representative tile sizes exceed the conventional sizes of the L1 and L2 caches (respectively around 64Kb and 256Kb). Therefore, we have set data sizes at least equal to the size of L2 cache in our experiments. To carry out our measurements, we used two benchmarks: BenchIT combined with x86membench and Intel Memory Latency Checker v3.8 to confirm the results.

The latency and bandwidth measurements inside a NUMA node for a data size just beyond the size of the L2 cache are attributed to the intra-node links, while measurements with data sizes just beyond the size of the L3 cache are attributed to the backbone. For inter-node links, we performed tests to measure values corresponding to the UPI/Infinity Fabric links latencies/bandwidths for Intel/AMD simulations: 147ns/221ns, and 45GBps/70GBps.

5.3 Results

In order to evaluate the simulator, we carry out tests on dense linear algebra applications in different data size scenarios and check the sOMP precision error, both in the case with only task execution modeling, and with the addition of the communications-based model

Our first tests aim to verify the reliability of the simulator for several tile sizes. We compare a real execution time to the simulated time for a matrix

with a size of 16384×16384 and different tile sizes (512×512, 768×768, and 1024×1024) on the machines presented in Section 5.2. We perform tests using a single core up to using all cores on a node. As presented in Figure 3(left), on the Intel architecture the variation in the number of cores influences the accuracy of the simulator regardless of tile size.

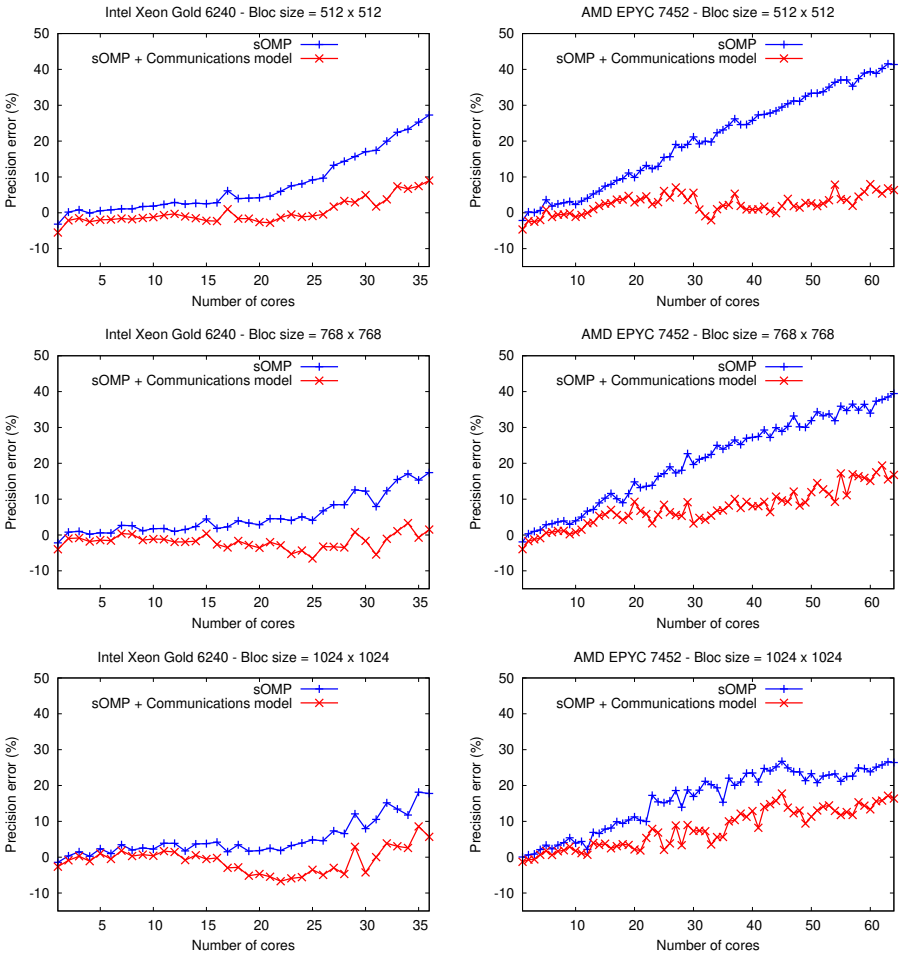

Fig. 3: sOMP simulator accuracy for the Cholesky algorithm using three different bloc sizes on the two architectures, for the same matrix size 16384 x 16384.

On the Intel architecture, we achieve $\pm 5\%$ error of precision on the Cholesky algorithm when running on a single socket (up to 18 cores) and using the task execution model only. Additionally using the second socket contributes to an increase in precision error, especially when approaching full machine usage.

However, we observe that the communications model introduced compensates for the loss of precision, notably when the majority of the cores of the two sockets are used. This is highlighted in tests with AMD architecture (Figure 3(right)), where the communication model provides excellent precision compared to task execution alone, especially for fine-grain simulations —under 10% error up to 32 cores and less than 20% at full node in all configurations—. The difference in precision between the two machines is due to the nature of each architecture: Intel with a single NUMA domain per socket generates less memory effects than the four NUMA domains socket AMD.

The simulator's behavior, when coupled with the communications model, allows us to confirm the reliability of the developed NUMA modeling, and also, observe the impact of memory-related effects on the execution of the application when disabling communications. In algorithms where tasks are handling fewer data, sOMP default model allows us to obtain better accuracy, as depicted in Figure 4 for the LU algorithm. The tile size is fixed (768 x 768) and two matrices of 8192 and 16384 are simulated. For the task execution model the error of precision remains lower than 15model based on communications, it is possible to efficiently improve the simulations even for a large number of cores with an error contained in the interval [-5%, 5%] regardless of the problem size. Therefore, we can achieve better overall precision on the LU algorithm compared to other simulators such as simNUMA (30% average error).

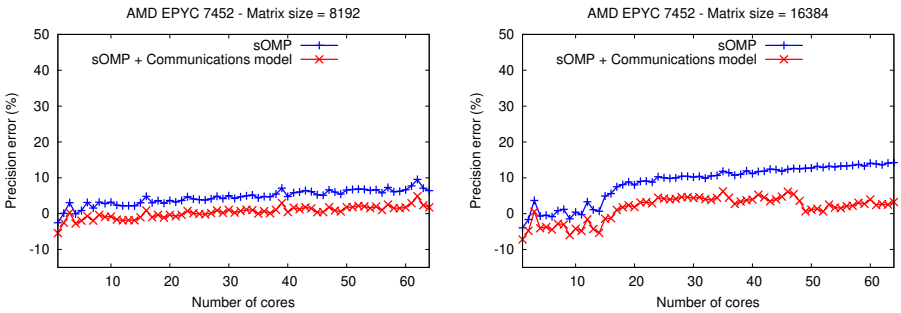

Fig. 4: sOMP simulator accuracy for the LU algorithm using two different matrix sizes and a tile size of 768 x 768 on AMD EPYC 7452

For the QR algorithm, the precision is influenced widely by the size of the problem: in Figure 5, we fix the tile size (768×768) and vary the size of the matrix. In the first case (matrix size of 8192×8192 and a tile size of 768×768), the error remains below 10% on average. However, for the matrix size 16384×16384 (four times bigger) the precision error grows linearly from -5% to about 37%, with an average of 21%. We also observe that the communications model contributes less to improving accuracy compared with tests for the Cholesky algorithm. This is related to the nature of the task graph of the QR algorithm, which is slightly

different: first, the arithmetic intensity of the kernels is more significant, so data accesses have less impact on the execution time. Next, QR kernels handle more data per task, some of which are temporary, generating significant cache-related effects that are not supported by the current version of the simulator and will be addressed in further work.

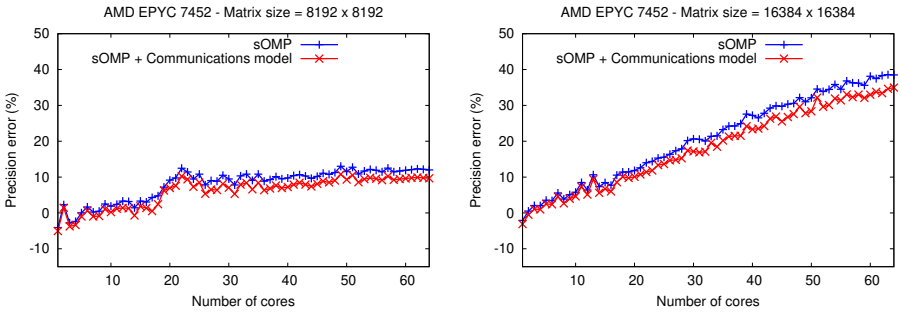

Fig. 5: sOMP simulator accuracy for the QR algorithm using two different matrix sizes and a tile size of 768 x 768 on AMD EPYC 7452

Finally, simulation with sOMP provides considerable time savings compared to actual execution. The simulation time is primarily linked to the number of tasks and the number of cores that emit communications at the same time. At the full scale (all cores) and fine grain blocking (matrix size of 16384×16384 and tile size of 512×512) and with the default model, simulations are typically $30\times$ faster. With the communications model, they are typically only $5\times$ faster on the Intel system and $2.5\times$ faster on the AMD system which has twice as many cores emitting communications at the same time. The overhead created by the communications-based model highlights the concerns mentioned earlier in adding too many architectural elements. Furthermore, SimGrid uses only one core, so several simulations can be run simultaneously on a multicore laptop.

6 Conclusion

This work focuses on simulating OpenMP task-based applications on shared memory architectures. On such structures, taking memory effects into account is crucial to obtain accurate simulations of linear algebra applications. Modeling the execution time of the tasks only is not sufficient.

We introduced a model to simulate the effects of memory accesses by leveraging the communications features offered by the SimGrid framework. Although SimGrid is oriented towards simulations of distributed architectures, we showed that we could divert its use to model a shared-memory machine, and to build a simulator for linear algebra applications based on parallel tasks with data dependencies, offering a good trade-off between the cost and the accuracy of the simulations.

We showed that the communications model consistently reduces the precision error, regardless of the number of cores or the architectures. Within a processor, the simulator initially obtains an average relative error of around 15%; the communications model lowers this to less than 5% for the LU algorithm. Therefore, we have shown that it is necessary to consider memory access metrics in the architecture model to reduce precision errors.

Moreover, we observed that variations in the number of cores and granularity deeply impact simulation accuracy within a socket. Two effects are involved: concurrent memory access contention, and data movements between caches. Even if our machine model does not capture the detailed connectivity between the cores, we were able to simulate the contention delays accurately. However, we do not yet model data movements between caches, as depicted in results with the QR algorithm. Capturing this second effect is the subject of on-going work. We can also take into account more architecture components and simulate other applications such as SpMVM, BiCGStab... In the longer run, it will be useful to combine this work with simulations of MPI and GPUs to achieve the simulation of hybrid MPI/OpenMP applications on heterogeneous architectures.

Acknowledgments

This work is partially supported by the Hac Specis INRIA Project Lab. Experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see https://www.plafrim.fr/).

References

1. Agullo, E., Beaumont, O., Eyraud-Dubois, L., Kumar, S.: Are Static Schedules so Bad ? A Case Study on Cholesky Factorization. In: IPDPS'16. Proceedings of the 30th IEEE International Parallel & Distributed Processing Symposium, IPDPS'16, IEEE, Chicago, IL, United States (May 2016)
2. Aversa, R., Di Martino, B., Rak, M., Venticinque, S., Villano, U.: Performance prediction through simulation of a hybrid mpi/openmp application. Parallel Computing **31**(10), 1013 – 1033 (2005), openMP
3. Board, O.A.R.: Openmp application programming interface - version 5.0. `https://www.openmp.org` (2018)
4. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience **41**(1), 23–50 (2011)
5. Casanova, H.: Simgrid: a toolkit for the simulation of application scheduling. In: Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid. pp. 430–437 (2001)
6. Czarnul, P., Kuchta, J., Matuszek, M., Proficz, J., Rościszewski, P., Wójcik, M., Szymański, J.: Merpsys: An environment for simulation of parallel application execution on large scale hpc systems. Simulation Modelling Practice and Theory **77**, 124 – 140 (2017)

7. Daumen, A., Carribault, P., Trahay, F., Thomas, G.: ScalOMP: analyzing the Scalability of OpenMP applications. In: IWOMP 2019: 15th International Workshop on OpenMP. vol. Lecture Notes in Computer Science book series, volume 11718 ; Programming and Software Engineering book sub series, volume 11718, pp. 36–49. Springer, Auckland, New Zealand (Sep 2019)
8. Denoyelle, N., Goglin, B., Ilic, A., Jeannot, E., Sousa, L.: Modeling non-uniform memory access on large compute nodes with the cache-aware roofline model. IEEE Transactions on Parallel and Distributed Systems **30**(6), 1374–1389 (2019)
9. Eichenberger, A.E., Mellor-Crummey, J., Schulz, M., Wong, M., Copty, N., Dietrich, R., Liu, X., Loh, E., Lorenz, D.: Ompt: An openmp tools application programming interface for performance analysis. In: OpenMP in the Era of Low Power Devices and Accelerators. pp. 171–185. Springer, Berlin, Heidelberg (2013)
10. Engelmann, C.: 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 **30**, 59 – 65 (2014), special Issue on Extreme Scale Parallel Architectures and Systems, Cryptography in Cloud Computing and Recent Advances in Parallel and Distributed Systems, ICPADS 2012 Selected Papers
11. Feld, C., Convent, S., Hermanns, M.A., Protze, J., Geimer, M., Mohr, B.: Score-p and ompt: Navigating the perils of callback-driven parallel runtime introspection. In: OpenMP: Conquering the Full Hardware Spectrum. pp. 21–35. Springer, Cham (2019)
12. Gautier, T., Pérez, C., Richard, J.: On the Impact of OpenMP Task Granularity. In: IWOMP 2018 - 14th International Workshop on OpenMP for Evolving Architectures. pp. 205–221. Springer, Barcelone, Spain (Sep 2018)
13. Girona, S., Labarta, J.: Sensitivity of performance prediction of message passing programs. The Journal of Supercomputing **17** (2000)
14. Haugen, B.: Performance analysis and modeling of task-based runtimes. Ph.D. thesis (2016)
15. Haugen, B., Kurzak, J., YarKhan, A., Luszczek, P., Dongarra, J.: Parallel simulation of superscalar scheduling. In: 2014 43rd International Conference on Parallel Processing. pp. 121–130 (2014)
16. Heinrich, F.: Modeling, Prediction and Optimization of Energy Consumption of MPI Applications using SimGrid. Theses, Université Grenoble Alpes (May 2019)
17. de Kergommeaux, J.C., Guilloud, C., de Oliveira Stein, B.: Flexible performance debugging of parallel and distributed applications. In: Euro-Par 2003. Parallel Processing, 9th International Euro-Par Conference, Klagenfurt, Austria, August 26-29, 2003. Proceedings. Lecture Notes in Computer Science, vol. 2790, pp. 38–46. Springer (2003)
18. Kliazovich Dzmitry, Bouvry Pascal, K.S.U.: Greencloud: a packet-level simulator of energy-aware cloud computing data centers. The Journal of Supercomputing (2012)
19. Liu, Y., Zhu, Y., Li, X., Ni, Z., Liu, T., Chen, Y., Wu, J.: Simnuma: Simulating numa-architecture multiprocessor systems efficiently. In: 2013 International Conference on Parallel and Distributed Systems. pp. 341–348 (Dec 2013)
20. Mohammed, A., Eleliemy, A., Ciorba, F.M., Kasielke, F., Banicescu, I.: Experimental verification and analysis of dynamic loop scheduling in scientific applications. In: 2018 17th International Symposium on Parallel and Distributed Computing (ISPDC). pp. 141–148. IEEE (2018)
21. Porterfield, A., Fowler, R., Mandal, A., Lim, M.Y.: Empirical evaluation of multicore memory concurrency (2009)

22. Rico, A., Duran, A., Cabarcas, F., Etsion, Y., Ramirez, A., Valero, M.: Trace-driven simulation of multithreaded applications. In: (IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software. pp. 87–96 (2011)
23. Slimane, M., Sekhri, L.: Hlsmn: High level multicore numa simulator. Electrotehnica, Electronica, Automatica **65**(3) (2017)
24. Stanisic, L., Agullo, E., Buttari, A., Guermouche, A., Legrand, A., Lopez, F., Videau, B.: Fast and Accurate Simulation of Multithreaded Sparse Linear Algebra Solvers. In: The 21st IEEE International Conference on Parallel and Distributed Systems. Melbourne, Australia (Dec 2015)
25. Stanisic, L., Thibault, S., Legrand, A., Videau, B., Méhaut, J.F.: Faithful performance prediction of a dynamic task-based runtime system for heterogeneous multi-core architectures. Concurrency and Computation: Practice and Experience **27**(16), 4075–4090 (2015)
26. Tao, J., Schulz, M., Karl, W.: Simulation as a tool for optimizing memory accesses on numa machines. Performance Evaluation **60**(1-4), 31–50 (2005)
27. Virouleau, P., Brunet, P., Broquedis, F., Furmento, N., Thibault, S., Aumage, O., Gautier, T.: Evaluation of openmp dependent tasks with the kastors benchmark suite. In: International Workshop on OpenMP. pp. 16–29. Springer (2014)
28. Virouleau, P., Brunet, P., Broquedis, F., Furmento, N., Thibault, S., Aumage, O., Gautier, T.: Evaluation of openmp dependent tasks with the kastors benchmark suite. In: Using and Improving OpenMP for Devices, Tasks, and More. pp. 16–29. Springer, Cham (2014)
29. Virouleau, P., Roussel, A., Broquedis, F., Gautier, T., Rastello, F., Gratien, J.M.: Description, implementation and evaluation of an affinity clause for task directives. In: International Workshop on OpenMP. pp. 61–73. Springer (2016)
30. Zheng, G., Kakulapati, G., Kalé, L.V.: Bigsim: A parallel simulator for performance prediction of extremely large parallel machines. In: 18th International Parallel and Distributed Processing Symposium, 2004. Proceedings. p. 78. IEEE (2004)