Towards the Reproduction of Selected Dynamic Loop Scheduling Experiments Using SimGrid-SimDag

Ali Mohammed, Ahmed Eleliemy, and Florina M. Ciorba
Department of Mathematics and Computer Science
University of Basel, Switzerland
Email: {ali.mohammed, ahmed.eleliemy, florina.ciorba}@unibas.ch

***Abstract*— Modern computing architectures exhibit increasing parallelism. Therefore, dynamic loop scheduling (DLS) plays an increasing role in the performance optimization of parallel applications executing on the modern computing architectures. In the previous decades, there was a large body of research concerning DLS techniques. Reproduction of the DLS experiments is significant for ensuring the trustworthiness of the DLS techniques implementation in modern scheduling tools or within new scientific applications. The results of executing the implemented DLS techniques are expected to be in agreement with the results reported in earlier work. The present work is a step towards the reproduction of the experiments that introduced the well-known DLS technique named factoring (FAC). Studying scheduling techniques via simulation is favorable compared to native execution to have control over all the factors that may affect the performance. The use of simulation in this work is essential for the reproduction of the scheduling experiments performed on computing systems that no longer exist. This work shows that the self scheduling technique with matrix multiplication kernel has a significantly poorer performance on the modern system considered in this study than on the past system.**

1. Introduction

Scheduling is the assignment in space and over time of parallel tasks. Dynamic loop scheduling (DLS) is the assignment of loop iterations (as tasks) to idle and requesting processing elements during execution. Loops are considered a rich source for parallelism in scientific applications. Loop iterations may have variable execution times due to problem, algorithmic, or systemic characteristics. In parallel execution, variable iterations' execution times causes uneven processor finishing times and degrade the performance. DLS can achieve a load balanced parallel execution by assigning loop iteration(s) to available processors requesting computational work. DLS has gained an increasing importance as modern systems exhibit increased parallelism.

One of the well-known and successful DLS techniques is *factoring*, introduced by Hummel et al. [1] in 1992. Therein, the authors compared the performance of the IBM Research Parallel Processor Prototype (RP3) system [2] for the execution of three computational kernels: matrix multiplication, adjoint convolution, and Gauss-Jordan elimination using four scheduling techniques: static chunking (STATIC), self scheduling (SS) [3], guided self scheduling [4] (GSS), and factoring (FAC). The RP3 was considered the state-of-the-art shared memory parallel computing machine in 1985 that was designed to accommodate up to 512 processors. Reproducing the same experiments on a modern manycore system would help explore the performance of the DLS techniques on these systems and the difference in their behavior from past systems to modern systems.

Reproducibility is a key aspect of scientific research [5]. Reproducing scientific experiments helps confirm the experiments and establish that conclusions drawn from these experiments are of a scientific relevance [6]. As such, in the present work, selected scheduling experiments with two computational kernels from [1] are reproduced using simulation, as well as executed natively. The reproduction of the scheduling experiments from [1] is used as a means to verify the implementation in SimGrid [7] of the DLS techniques presented in [1], and in particular in its SimDag interface (SimGrid-SD).

The present work makes the following contributions: (1) Uses reproduction to verify the implementation into SimGrid-SD of the four scheduling algorithms from [1]. (2) Explores the impact of using modern manycore computing system on the conclusions drawn from past experiments. (3) Validates the developed SimGrid-SD-based simulation of two computational kernels from [1] using the four aforementioned non-adaptive dynamic loop scheduling techniques.

The remainder of the paper is structured as follows: In Section 2, the background of the used DLS techniques and the used simulation toolkit are reviewed. Certain studies related to this work are discussed in Section 2. The proposed approach for the use of reproduction as a means to verify and validate the simulation of the DLS techniques with SimGrid-SD is described in Section 3. The results of the reproduction of the selected experiments natively and in simulation are presented in Section 4. The conclusion and insights into future work are outlined in Section 5.

2. Background and Related Work

In this section, a background on scheduling and reproducibility, as well as a brief overview of the

SimGrid simulation toolkit is given. Certain studies related to this work are presented afterward.

Scheduling. Four loop scheduling techniques are considered in this work: one static (STATIC) and three dynamic (SS, GSS, FAC). They are briefly described next. Using STATIC, each processor is assigned exactly one chunk of size equal to the total number of iterations divided by the number of processing elements. STATIC has low scheduling overhead. SS [3] can be seen as the other scheduling extreme. Using SS, a processor obtains a new iteration whenever it becomes available. SS is characterized by its great ability to balance the load between the processors and by its high scheduling overhead. Other scheduling techniques offer a good compromise between the load balancing ability and the scheduling overhead, such as GSS and FAC. GSS [4] assigns a chunk of iterations to an available processor that is equal to the number of the remaining unscheduled iterations divided by the number of the processors. GSS has the characteristic of assigning a very large chunk to the first available processor and this may lead to load imbalance. FAC [1] is designed to balance the execution of loop iterations with variable execution times. It assigns chunks of iterations to available processors in batches, therefore, reducing the scheduling overhead. The number of the iterations in a chunk depends on the number of remaining iterations and the coefficient of variation of the iteration execution times.

Reproducibility. Reproduction is understood as defined in [5]: revisiting a certain scientific problem (in the present case: the performance of DLS techniques) without the original artifacts or the possibility to execute artifacts on the same computing system.

SimGrid. SimGrid [7] is a scientific simulation library, developed to study the behavior of large-scale distributed systems such as the Grid, the Cloud, and peer-to-peer (P2P) systems. SimGrid provides three different interfaces: MSG, SimDag, and SMPI, for different simulation purposes. SimDag interface was chosen in this work as it provides different types of tasks and the ability to specify dependencies between tasks. Therefore, it facilitates a precise representation of the scheduling context.

Related work. Two closely related studies that use reproduction to confirm the implementation of certain DLS techniques using simulation are [8] and [9]. Therein, the reproducibility of the use of DLS in earlier simulation-based experiments was studied. The reproduction was performed by implementing the DLS techniques in SimGrid-MSG.

The present work extends and complements previous work by implementing the DLS techniques using the SimGrid-SD interface. The focus of this work is the use of reproduction as a means of verification of the DLS techniques' implementation in SimGrid-SD. Moreover, the reproduction approach taken in this work is the first to confirm that the implementation of the DLS techniques in simulation (SimGrid-SD) is in agreement with their native execution performance on modern manycore architectures (e.g., Intel's KNL).

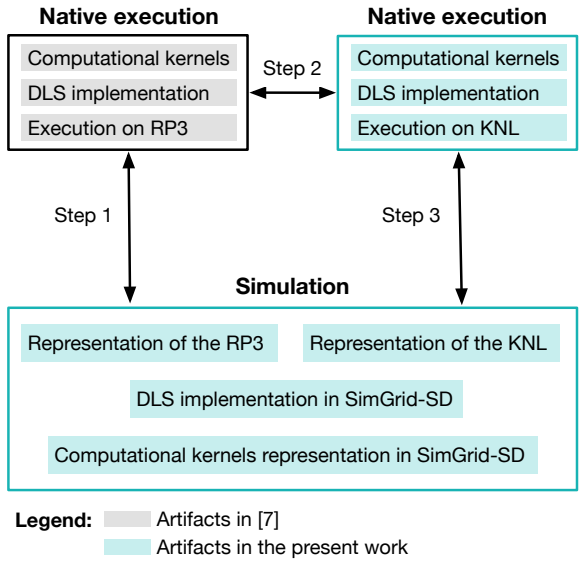

Figure 1. Reproduction of DLS techniques' behavior in SimGrid-SD.

3. Proposed Methodology

In this work, selected scheduling experiments from [1] are reproduced using SimGrid-SD simulation as the computing system used in the original paper is not available. The results of the simulation of the selected experiments from the present are compared to the results obtained in the original paper to verify the implementation of the DLS techniques in the simulation. As shown in Figure 1, the same experiments are reproduced on a modern manycore system to explore the impact of using a modern shared memory system on the conclusions from the original paper. The developed simulation is modified to represent the execution on the KNL instead of the RP3. The results of the modified simulation are compared to the results of the scheduling experiments from the present on the KNL to validate the simulation of the scheduling experiments under study.

Computational kernels. Two computational kernels, matrix multiplication and adjoint convolution, have been chosen for reproduction in this work to represent loop iterations with different properties. Matrix multiplication iterations are equal in the number of floating point operations. In the case of adjoint convolution, the inner loop is triangular. Therefore, the number of floating point operations per loop iteration of the adjoint convolution kernel is decreasing/increases as the loop execution progresses depending on the direction of the outer loop.

The two outermost loops of the matrix multiplication are coalesced, and the resulting loop is parallelized. For the adjoint convolution kernel, it consists of two nested loops, where the outer loop is parallelized. The pseudocode for the two computational kernels and their parallelization are available online[1], along with the developed simulation codes and results obtained from simulation and native execution.

1. https://drive.switch.ch/index.php/s/NbEu0nlR70jQqCE

TABLE 1. SELECTED DLS EXPERIMENTS FOR REPRODUCTION

Computational kernels	Matrix size	Scheduling technique	Computing system
Matrix multiplication	300×300	STATIC	
Adjoint convolution (increasing and decreasing task sizes)	75×75	SS GSS FAC	RP3 KNL

A master-worker execution model is assumed in implementing the scheduling experiments reproduced from [1] as there is no information on how the kernels were executed or how the data is distributed in the original paper. The master process is dedicated to the calculation of chunks and the assignment of tasks to requesting workers. The selected experiments for reproduction are summarized in Table 1.

Computing systems. Scheduling experiments included in the original paper [1] were executed on the RP3 [2] system. The RP3 was built by IBM for research purposes. The system used in the experiments consisted of 64 processors connected by an Omega network. Each processor has 32 KB cache and a local memory. Nonlocal memory is accessible over the network. The system was configured to run a version of Mach operating system, where multiprogramming is eliminated to reduce variability in the execution time between runs.

The selected scheduling experiments are reproduced on the state-of-the-art shared memory system KNL standalone processor version 7210. The KNL processor has 64 cores and 96 GB main memory[2]. The KNL runs CentOS operating system version 3.10.0, and the application code is compiled using the GNU compiler version 6.3.0 with $-O3$ $-mavx512f$ $-mavx512cd$ $-mavx512pf$ optimization flags for the KNL architecture.

SimGrid-SD simulation. A simulator is developed to simulate the execution of the two computational kernels with the four aforementioned DLS techniques using SimGrid-SD. Each iteration of the inner loop of the two computational kernels is represented as a sequential computational task in SimGrid-SD. Work request and work assignment are represented as end-to-end communication tasks in SimGrid-SD. A sequential computational task is executed on the master upon each work request, to represent the scheduling overhead to calculate chunks. Computational kernels do not execute at the peak speed of the processors in the native execution due to several factors, such as cache misses, branches, and delays in the execution pipeline. Therefore, applications execute at a much slower rate than the processor's nominal speed, resulting in a longer execution time in native execution than in simulation. To close the gap between native execution results and simulated results, the native execution time of each task on the KNL is measured. The amount of floating point operations (FLOP) that would result in each iteration execution time is inserted in the simulator to represent the iterations of the computational kernels. In the case of the simulation of the execution on the RP3, fitting parameters are used to close the gap between simulator results and results in the original publication.

To provide the SimGrid simulation engine with the specifications of the simulated system, an XML file called the platform file is needed. Each processor in the RP3 system is represented as a host in the SimGrid platform file used in the reproduction experiments. All hosts (processors) are interconnected by creating a communication link between every host and all others. The values used to represent the RP3 system are: processor speed 1.562 MFLOP/s, network bandwidth 50 Mbit/s, and latency 2 us. In the case of the KNL representation, each core is represented as a host in the platform file. The values used to describe the processor speed, network bandwidth, and network latency of the KNL system used in these experiments are 41.6 GFLOP/s, 100 Gbit/s, and 100 ns, respectively.

4. Evaluation

The scheduling experiments reproduced on the KNL are repeated 200 times, and the execution cost is measured and reported. The confidence interval of the measurements is calculated, and experiments are repeated until the sample average lies in the interval of 5% error with the confidence level of 95%. The performance of the adjoint convolution kernel with the four scheduling techniques is close to the corresponding performance in the original paper. In the case of the matrix multiplication kernel performance results (shown in Figure 2), the behavior of the SS technique is different from the results in the original paper, as the scheduling overhead dominates the performance and results in a longer execution time than other DLS techniques.

Native and simulation results are compared to verify the implementation of the DLS techniques and to validate

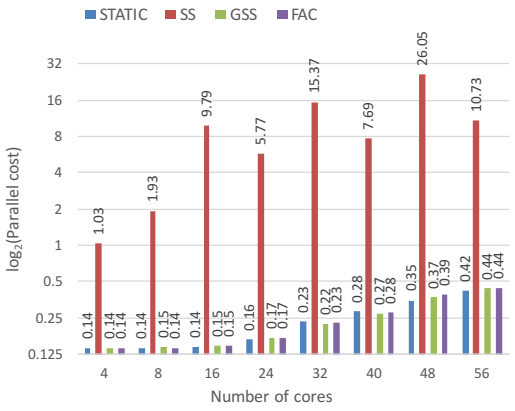

Figure 2. Native execution performance (as parallel cost) of the matrix multiplication kernel on KNL using the four scheduling techniques considered in this study. Parallel cost = parallel program execution time $\times$ number of workers.

2. Special features of the KNL, such as the multichannel DRAM (MCDRAM) and the cluster on die modes are not used in this study, to avoid having heterogeneity in the system or anomalies in the results. The MCDRAM is configured in the flat mode and the processor is booted in all-to-all cluster mode.

the simulation of the native performance. The relative percentage difference between the native and the simulated execution time is calculated as

$$\text{relative percentage difference} = \left(\frac{\text{simulated time}}{\text{native execution time}} - 1\right) \times 100\%.$$

The average of absolute relative differences for the verification experiments is 7.44% (comparison between native RP3 results from [1] and SimGrid-SD simulation results from the present work). The minimum and maximum relative absolute differences are 0.49% and 30.94%, respectively, in all the verification experiments. For the validation experiments (comparison between native KNL results and SimGrid-SD simulation results), the average of absolute relative difference is 9.99%. The minimum and the maximum relative differences are -0.23% and -44.99%, respectively.

The developed simulator can be configured to generate parallel execution traces to be visualized with Vampir [10]. In Figure 3, for instance, the traces of executing the adjoint convolution kernel with GSS scheduling using 16 workers are produced for the native execution and the simulation. The coefficient of variation (c.o.v.) of the cores finishing times in the native execution and simulation is calculated as a measure of the load imbalance. The c.o.v. in the native execution is 0.404 and in the simulation is 0.407. The comparison shows that the traces of native and simulated executions are visually similar and the c.o.v. of the cores finishing times are in the same order of magnitude.

5. Conclusion and Future Work

Reproduction of the selected scheduling experiments showed that conclusions drawn from results of the original paper might not hold on modern parallel shared memory architectures. For the matrix multiplication kernel, the gap between the performance of FAC and the performance of SS executing on the KNL architecture is much wider compared to the performance gap on the RP3 system. Given the significant advancement in the computation capabilities of modern architectures, the cost of synchronization is relatively large compared to the cost of computation. Consequently, for an application with loop iterations with equal sizes executing on modern computing architecture, SS may have a poorer performance than expected. Therefore, to select the most suitable DLS technique, the granularity of the computational work within each loop iteration needs to be considered, as scheduling and synchronization overheads can degrade the application performance. The DLS techniques can be implemented using a centralized execution model, such as the master-worker model employed in this work. The DLS techniques can also be implemented using a distributed execution model; and this is part of future work.

Acknowledgment

This work is in part supported by the Swiss National Science Foundation in the context of the Multi-level Scheduling in Large Scale High Performance Computers (MLS) grant, number 169123.

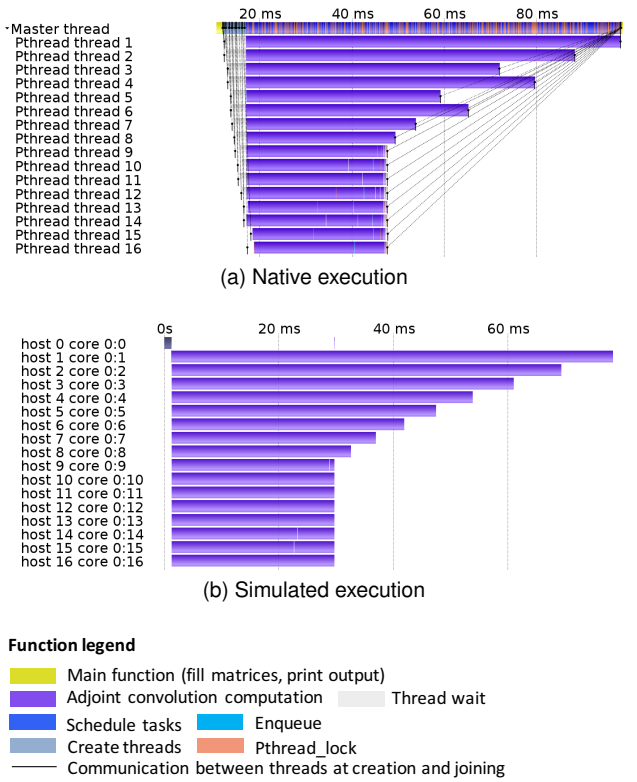

Figure 3. Traces obtained from execution (top) (with execution time of 0.082 s) and its corresponding simulation (bottom) (with simulated execution time of 0.079 s) of the GSS with adjoint convolution kernel of decreasing task sizes using 16 worker threads.

References

[1] S. F. Hummel, E. Schonberg, and L. E. Flynn, "Factoring: A method for scheduling parallel loops," *Communications of the ACM*, vol. 35, no. 8, pp. 90–101, 1992.

[2] G. Pfister, W. Brantley, D. George, S. Harvey, W. Kleinfelder, K. McAuliffe, E. Melton, V. Norton, and J. Weiss, "The IBM research parallel processor prototype (RP3): Introduction and architecture," in *International Conference on Parallel Processing*, August 1985, pp. 764–772.

[3] P. Tang and P.-C. Yew, "Processor self-scheduling for multiple-nested parallel loops," in *International Conference on Parallel Processing*, vol. 86, 1986, pp. 528–535.

[4] C. D. Polychronopoulos and D. J. Kuck, "Guided self-scheduling: A practical scheduling scheme for parallel supercomputers," *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1425–1439, 1987.

[5] ACM. Artifact review and badging. [Online]. Available: https://www.acm.org/publications/policies/artifact-review-badging

[6] S. Hunold and J. L. Träff, "On the state and importance of reproducible experimental research in parallel computing," *Computing Research Repository*, vol. abs/1308.3648, 2013. [Online]. Available: http://arxiv.org/abs/1308.3648

[7] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, 2014.

[8] F. Hoffeins, F. M. Ciorba, and I. Banicescu, "Examining the Reproducibility of Using Dynamic Loop Scheduling Techniques in Scientific Applications," in *Proceedings of the 4th International Workshop on Reproducibility in Parallel Computing (REPPAR) of the 31st IEEE International Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW 2017)*, Orlando, USA, May 2017.

[9] F. Hoffeins, F. M. Ciorba, and I. Banicescu, "Towards the Reproducibility of Using Dynamic Loop Scheduling Techniques in Scientific Applications," in *Proceedings of 16th International Symposium on Parallel and Distributed Computing (ISDPC)*, July 2017, p. 8.

[10] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, and W. E. Nagel, "The Vampir performance analysis tool-set," in *Proceedings of the 2nd International Workshop on Parallel Tools for High Performance Computing*, July 2008, pp. 139–155.