Enhancing Scheduling through Monitoring and Prediction Techniques

Antoni Navarro Muñoz*†, Vicenç Beltran Querol*, Eduard Ayguadé Parra*†
*Barcelona Supercomputing Center, Barcelona, Spain
†Universitat Politècnica de Catalunya, Barcelona, Spain
E-mail: {antoni.navarro, vbeltran, eduard.ayguade}@bsc.es

Keywords—High-Performance Computing, OmpSs-2, Scheduling, Monitoring, Predictions, Cost

I. Extended Abstract

Modern applications become larger and more complex with each passing day. To name a few, weather forecasting or particle simulations are examples of how applications may have significant differences in features, constraints, and limitations.

Most runtimes supply users with functionalities to tune their executions. However, many aspects have to be taken into consideration when optimizing applications. Input sizes, recursive depths, system workloads, or the underlying architecture onto which apps are running, are just a few. Users often try different configurations until they stumble upon one which seems to yield the most performance. This proves to be nonportable, as a slight change in any of the aspects mentioned before might yield undesirable negative effects in performance.

In this work, our primary goal is to add several monitoring modules to runtimes. These modules introduce precise information about the units of work these libraries must schedule. The extension of these libraries allows for accurate real-time predictions for present and future executions. Such predictions can be used to obtain better scheduling of future units of work automatically and, therefore, improve the overall performance of executions or the utilization of resources. All this, while being unnoticed by users, thus giving more power to the runtimes.

Through the evaluation provided, we demonstrate the precision of our predictions and how they can be used to optimize resource utilization among others. We integrate all the extensions mentioned above on an already existing runtime maintaining the vision of the integration being capable of any similar runtime or library.

A. Monitoring Techniques

For the purpose of improving scheduling techniques adaptively and automatically, we propose a monitoring infrastructure. The primary objective of the infrastructure is to gather metrics and use them in real time. Our approach consists of an API that couples with an existent runtime. To exemplify this, we use OmpSs-2, the second generation of OmpSs [1], a task-based programming model. More specifically we integrate this infrastructure mentioned above in Nanos6 [2], a runtime library that implements OmpSs-2.

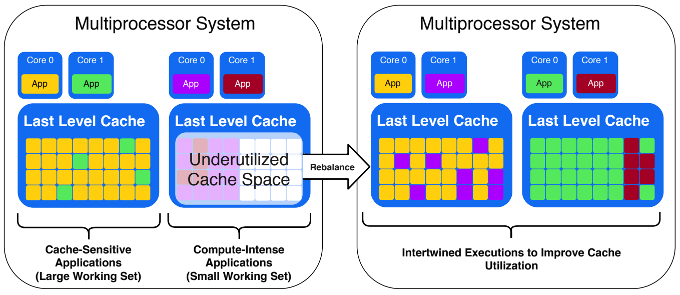
Multiprocessor System
Core 0
Core 1
App
Last Level Cache
Underutilized Cache Space
Rebalance
Cache-Sensitive Applications (Large Working Set)
Compute-Intense Applications (Small Working Set)
Multiprocessor System
Intertwined Executions to Improve Cache Utilization

Fig. 1. Rebalancing applications across processors for optimal resource utilization

With this infrastructure, scenarios that are optimizable come to surface. One of these is to detect when resources can be exploited more efficiently. To serve as an example, depending on the internal static scheduling policies of a runtime, resources such as CPUs can be underutilized. Another scenario includes detecting when units of work can be scheduled more efficiently to improve execution time and overall performance.

Our primary goal is to detect these scenarios. Figure 1 exemplifies this. In this figure, we can see two multiprocessor systems, each with a last level cache and two cores. In each of these cores is running an application. On the first pair of cores we observe cache-intensive applications. On the second pair, compute-intensive ones. As shown, the applications on the left utilize the whole last level cache. This can lead to inefficient usage of the last level cache, as both applications will fight over the resources. On the other hand, the ones on the right underutilize the last level cache. This scenario can be optimized by rebalancing the workloads. Compute-intensive applications can be interleaved with cache-intensive ones. This is shown in the right part of the figure, where applications are mixed up. Rebalancing in these scenarios is bound to improve cache utilization.

Two APIs form our monitoring infrastructure. The first monitors timing metrics for elements such as tasks (units of work), threads, and CPUs. The latter monitors hardware events for the same parts. Both are generic APIs, and independent from each other. Next we show how researchers may benefit from using our infrastructure by creating smarter scheduling policies or mechanisms that take advantage of the collected metrics and predictions.

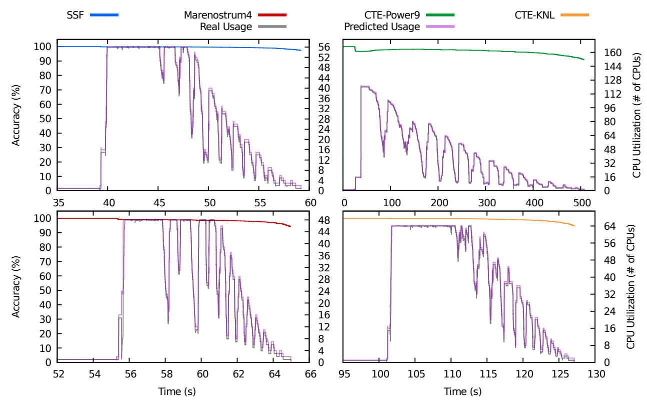

Fig. 2. Accuracy of the CPU predictor across different architectures

B. Current Contributions

So far, our contributions have targeted both of the previously mentioned scenarios. In our first contribution [3] we created a mechanism that automatically detects when an excessive amount of parallelism is being generated in recursive applications. Upon detecting such an event, the mechanism adapts the execution through the use of timing metrics and predictions. Through these predictions, it is capable of automatically ceasing the generation of recursive tasks. Instead, these are inlined in parent tasks.

In other recent contributions, we created a predictor that uses real-time data from the same monitoring infrastructure. This predictor is able to infer the amount of CPU needed to execute the current workload of the system. Next, we present the evaluation of the predictor.

C. Evaluation & Results

Our predictor was evaluated using a set of six benchmarks with varying features and granularities. Moreover, it was tested in different architectures, as our approaches are application and architecture independent. Some of the architectures tested are IBM's Power9 8335-GTG processors, Intel's Xeon Phi Knights Landing processors and Intel's Xeon Phi E5-2690v4 processors.

In figure 2 we demonstrate the effectiveness of our predictor in the Cholesky factorization application. We showcase four figures, each with the real CPU usage, the predicted CPU Usage and the overall accuracy of the prediction at each timestep. The four figures represent the accuracy of the predictor in the four architectures we tested.

Our results show that in all the architectures that were tested, and for any application or parameter, our predictor was able to precisely predict the CPU usage for the most part of all executions. This, as shown in that same figure, is not limited to sudden drops or peaks of workload. These last scenarios are tackled by using timing predictions and the size of internal queues.

D. Future Work

After demonstrating the accuracy of our timing predictions in previous works, we aim to assess the accuracy of our hardware-event based predictions.

Ensuring a consistent and precise infrastructure for metrics will lead to creating other mechanisms to ensure efficient executions – this, taking into account both resource usage and application performance. Thus, our roadmap includes detecting scenarios like those shown in figure 1 and introducing countermeasures to tackle them.

E. Conclusion

In our research, we have surfaced the need for adaptive scheduling policies that are independent of application and architecture features. We have also proven our predictions to be both accurate and useful for task-based programming models. In this study, we have focused on two of our previous works. The first targets application performance. The second resource efficiency optimization.

By having established a consistent ground for our research with a monitoring infrastructure, we are confident that our next contributions will enable researchers to realize the potential of adaptive techniques in their programming models and runtimes.

II. Acknowledgments

Part of this work has been published in [3]. Other parts have been introduced in [4], and are expected to be extended and published in the future.

References

[1] Alejandro Duran, Eduard Ayguadé, Rosa M Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, and Judit Planas. Ompss: a proposal for programming heterogeneous multi-core architectures. *Parallel processing letters*, 21 (02):173–193, 2011.

[2] Barcelona Supercomputing Center Programming Models Group. The nanos6 runtime repository, 2018. URL `https://github.com/bsc-pm/nanos6`. Accessed: 29-12-2018.

[3] Antoni Navarro, Sergi Mateo, Josep Maria Perez, Vicenç Beltran, and Eduard Ayguadé. Adaptive and architecture-independent task granularity for recursive applications. In Bronis R. de Supinski, Stephen L. Olivier, Christian Terboven, Barbara M. Chapman, and Matthias S. Müller, editors, *Scaling OpenMP for Exascale Performance and Portability*, pages 169–182, Cham, 2017. Springer International Publishing. ISBN 978-3-319-65578-9.

[4] Antoni Navarro, Vicenç Beltran, and Eduard Ayguadé. *Enhanced scheduling techniques through lightweight monitoring for OmpSs-2*. Master's thesis, Universitat Politècnica de Catalunya (UPC), 2019.

Antoni Navarro received his BSc degree in Computer Engineering and his MSc degree in the Master in Innovation and Research in Informatics with High-Performance Computing specialization from Universitat Politècnica de Catalunya (UPC), Barcelona, in 2016 and 2018 respectively. Since 2016 he has been in the Programming Models group of the Computer Science Department of Barcelona Supercomputing Center (BSC-CNS). In 2018, he started as a Ph.D. student at the department of Computer Architecture of Universitat Politècnica de Catalunya (UPC), Spain.