Spatial and Temporal Cache Sharing Analysis in Tasks

Germán Ceballos, David Black-Schaffer

Uppsala University, Sweden
firstname.lastname@it.uu.se

Abstract

Understanding performance of large scale multicore systems is crucial for getting faster execution times and optimize workload efficiency, but it is becoming harder due to the increased complexity of hardware architectures. Cache sharing is a key component for performance in modern architectures, and it has been the focus of performance analysis tools and techniques in recent years. At the same time, new programming models have been introduced to aid the programmer dealing with the complexity of large scale systems, simplifying the coding process and making applications more scalable regardless of resource sharing. Task-based runtime systems are one example of this that became popular recently. In this work we develop models to tackle performance analysis of shared resources in the task-based context, and for that we study cache sharing both in temporal and spatial ways. In temporal cache sharing, the effect of data reused over time by the tasks executed is modeled to predict different scenarios resulting in a tool called StatTask. In spatial cache sharing, the effect of tasks fighting for the cache at a given point in time through their execution is quantified and used to model their behavior on arbitrary cache sizes. Finally, we explain how these tools set up a unique and solid platform to improve runtime systems schedulers, maximizing performance of execution of large-scale task-based applications.

Keywords Task-based runtime systems, cache sharing, performance analysis, NESUS

I. Introduction

Maximizing applications performance on the multi-cores era is hard due to sharing resources, such as the caches, as it can have a negative or positive impact on the total execution time. To deal with this, newest programming models simplify the coding process of large scale parallel applications. Task based programming is one example of this, where the code is dis-aggregated in small units of code (independent functions) called tasks, and a runtime system determines their execution order and placement. The task based approach is simpler to reason for the programmer while it is also a good approach for performance as it can adapt the scheduling to the effective resource sharing. However, it is a very different dynamic of execution, making harder to understand performance of these systems due to the lack of models and tools.

In this paper we look at two key types of cache sharing (both *temporal* and *spatial*, in a task based context. An application might reuse data brought to the cache in the past, meaning that the cache is being shared in a *temporal* way. On the other hand, two applications might contend for the cache at the same moment in time, fighting to install and keep data in it, meaning that the cache is being shared in a *spatial* notion.

To do so, we develop efficient modeling techniques to predict performance with the goal of improving runtime scheduling decisions based on task sensitivity to hardware resource sharing, maximizing performance of large scale parallel applications.

To achieve this we first developed StatTask, a fast and efficient method to predict cache miss ratios for any arbitrary schedule from information sampled from a single execution. This method addresses temporal cache sharing between tasks: how sensitive tasks are to inter-task data reuse over time. An example can be seen in Figure 1 for tasks A, B and C. Tasks A

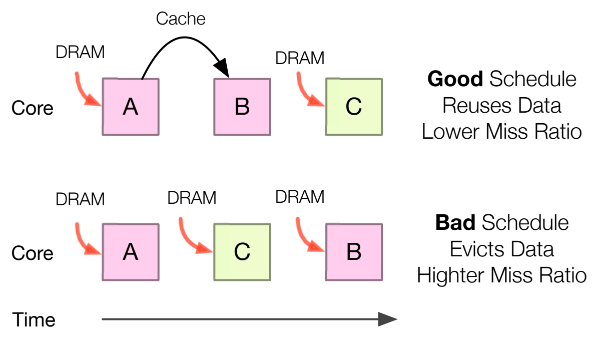

Figure 1: Temporal Locality in Task Based Systems.

and B share data, and B might reuse it from the cache However, executing tasks C in between could evict this shared data, causing data to be fetched from memory and increasing the execution time.

Second, we developed a method for predicting performance of co-running applications combining both statistical cache models and performance models for regular applications. Previous works did not take into account parallelism in the memory hierarchy in combination with statistical cache models, which is a key factor for performance.

Later, we extended this method to address tasks spatial resource sharing: how the memory hierarchy is shared at a given moment in time during execution. An example is display in Figure 2, where tasks A1 and B1 executing in parallel will bring data at the same time to the caches with different ratios. Since they fight for the cache, both tasks will end up with smaller cache portions impacting on their performance. However, if tasks would have been co-executed with tasks sharing data (respectively A2 and B2) sharing could have reduced their misses.

The method we present is able to predict quickly, accurately and with low-overhead, how multiple tasks running in parallel will compete for the caches.

Third, we explain how our models for temporal and spatial cache sharing can be combined improve schedulers of task-based runtime systems by giving them feedback.

II. Related work

There are three categories of related work: existing profiling tools that identify bottlenecks of task-based applications, task-scheduling optimization techniques, and finally techniques to analyze and understand data locality properties of applications.

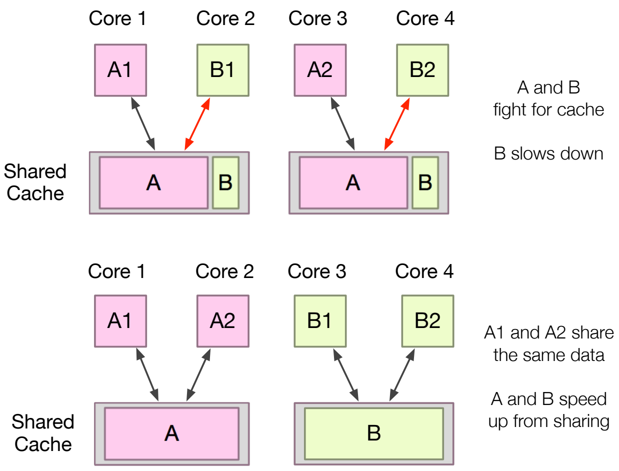

Figure 2: Spatial Locality in Task Based Systems.

Many tools exist to profile scheduling and load-balancing of tasks. Ding et. al. [8] presented a generic and accessible tool for task monitoring, independent of any program or library and able to acquire rich information with very low overhead, targeting load balancing and scheduling problems unrelated to data reuse. Lorenz et. al developed [16], a library for identifying performance problems inherent to tasking with OpenMP through direct instrumentation. Schmidl et. al. [17] surveyed different techniques to analyze data delivered by instrumentation of task-based programs in order to integrate parallel performance modeling to the automation of load-balancing. Ghosh et. al. [14] have proposed OpenMP extensions to support dependence-based synchronization, Brinkmann et al. presented a graphical debugging tool for task parallel programming that works with most of the production frameworks. Weng and Chapman [19] looked at the task graph for OpenMP applications to optimize load balance.

In the second category, work has been done on improving scheduling strategies. The standard work-stealing approach was carefully analyzed by Blumofe and Leiserson in [5] and [1]. Strategies accounting for the tasks types were presented by Wimmer et. al. [20]. Adaptive cut-off scheduling to take advantage of data locality and reduce the runtime overhead were considered in [9]. Recently, important work on cache-aware task stealing was carried out in [7] by Chen et. al. Qian Cao et. al.

[6] proposed a hybrid scheduling policy for heterogeneous multicores using breadth-first over the available task-pool.

None of these approaches for task-based profiling have incorporated a general method for understanding the data reuse implications of the tasks and schedules. In this category, characterization of data reuse has been done theoretically in [12] by Frigo. Practically, this can be done through instrumentation based techniques as presented by Aamer et. al. in [15] and Weidendorfer in [18].

Statistical cache modeling, first introduced in [2], is another widely used way to characterize data locality. This work has been extended to other cache replacement policies by Eklov in [11], and to support thread-based or multicore shared caches in [4, 3, 10].

III. Thesis Idea

Our main contribution is the development models that address the prediction of temporal and spatial cache sharing for arbitrary cache sizes for task-based runtime systems. These model preserve fundamental properties to be used in conjunction with runtime schedulers for better scheduling: both models are fast and low-overhead, portable (easy to implement across different runtimes) and architectural-independent (working seamlessly with different architectures).

III.1 Temporal resource sharing

For task-based programs, data is initially brought into the cache by a task, and if it is reused, this reuse can come from either the same task (private reuse) or by a subsequent one (shared reuse). Other tasks that execute between tasks with shared data also bring new data into the cache that may evict the shared data, turning reuses from the cache into a cache misses, and hurting performance.

Thus, we classify memory accesses in three types, depending on where they come in the memory hierarchy: First accesses to a particular memory location must be brought from DRAM, for example cold cache misses, and therefore we will call them *DRAM Accesses*. Second, memory accesses to addresses previously loaded by another task, and which we will call *shared reuses*. This reuses will be able to bring data from the cache if it is large enough to hold the data sets of the sharing tasks and the data is not evicted by other tasks before the shared reuse. Finally, memory accesses to addresses previously loaded by the same task, called *private reuses*. This type of accesses will bring data from the cache if it is large enough to hold the entire task's data set while it is executing. With this classification, we are able to improve statistical cache models to support memory access information per-task.

A key property of statistical cache models is that are able to sample a memory access stream from an application during execution, build a profile depending on a *distance notion* that determines how close/far the data reuses happened, and use statistical inference to predict cache miss ratios for different cache sizes very quickly. However, if these methods are used on task-based applications, the profile would be built based on information collected from the execution of a particular schedule. Since changing the tasks' schedule can affect observed data reuses, predictions for cache misses given by these models would be wrong.

StatTask extends existing statistical cache models collecting extra information during the memory profiling stage. Memory access samples are taken for a particular task schedule and then classified on a task basis. Later, multiple profiles are built for different schedules, adapting what would happen to the distances in the reuses on each of those cases. With these new profiles, statistical inference is used to get cache miss ratios for the new schedules, predicting the correct scenarios. This enables accurate prediction of cache behavior for arbitrary schedules of tasks and cache sizes.

III.2 Spatial resource sharing

When analyzing cache behavior in multi-program workloads, previous statistical cache models did not treat memory level parallelism, which now became crucial in latest architectures. In modern multicore processors, a last level cache miss might queue a new request in the memory controller's queue, which might be handled in parallel with a previous miss. Thus two consecutive misses are likely to overlap, hiding the latency for the second miss compared to the case of treating them sequentially having a drastic improvement in performance.

The number of parallel misses treated on average throughout execution can be measured and is often known as memory level parallelism (*MLP*). Our second contribution is a technique that combines statistical cache modeling with a modern performance model, adding support for memory level parallelism, that is able to predict a breakdown of performance (measured in CPI) of co-running applications.

To do so, applications memory accesses are sampled with binary instrumentation, running on isolation. Later, a statistical cache model called StatCC is used to predict cache miss ratios of co-running application for arbitrary cache sizes, assuming an initial performance. Later, a realistic and advanced performance model called Interval model [13] is used to calculate the number of cycles spent on memory per-application when co-running. The interval model is based on the abstraction that the execution time is driven by long-latency events, such as long latency loads and branch misses. However, the number of cycles calculated can change the ratio in which each application miss in the cache. Thus, StatCC is used iteratively, predicting new miss-ratios and recomputing the number of cycles spent on memory with the interval model as a fixed-point iterative solver.

This method needs to be adapted for the task-based context. To do that, it is necessary to add the same support for identifying tasks as in Section III.2 generating a per-task profile. In addition, the MLP modeling has to be done on a per-task basis as well. Our method runs on a pair profiles tasks sequences profiles and applies the technique described above to estimate the CPI of both sequences of co-running tasks.

IV. Conclusion and future work

Multicore architectures have the potential for high performance on parallel applications, but they are hard to optimize for due to the complexities of resource sharing. In this work we have presented two contributions to understand cache sharing in a task-based context based on the analysis of memory access samples. First, we presented StatTask, an efficient statistical cache model that predicts cache miss ratios for arbitrary task schedules, addressing the temporal cache sharing problem. Second, we introduced a new method that quickly predicts the effect of simultaneous cache sharing on the tasks performance, addressing the spatial cache sharing issue. Both of our methods use the same low-overhead, sampled input information, and can be easily combined to enable performance modeling of arbitrary task schedules. With these new capabilities we will be able to to develop more intelligent task scheduling policies that take into account the effects of temporal and spatial cache sharing, and thereby enable task-based programs to automatically adapt to the complexities of modern multicore resource sharing.

Acknowledgments

The work presented in this paper has been partially supported by EU under the COST programme Action IC1305,‘Network for Sustainable Ultrascale Computing (NESUS)’, and by the Swedish Research Council, carried out within the Linnaeus centre of excellence UPMARC, Uppsala Programming for Multicore Architectures Research Center.

References

[1] U. Acar, G. Blelloch, and R. Blumofe. The data locality of work stealing. *Theory of Computing Systems*, 35(3):321–347, 2002.

[2] E. Berg and E. Hagersten. Statcache: A probabilistic approach to efficient and accurate data locality analysis. *Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, 2004.

[3] E. Berg and E. Hagersten. Fast data-locality profiling of native execution. *SIGMETRICS Perform. Eval. Rev.*, 33(1):169–180, June 2005.

[4] E. Berg, H. Zeffer, and E. Hagersten. A statistical multiprocessor cache model. In *Performance Analysis of Systems and Software, 2006 IEEE International Symposium on*, pages 89–99, March 2006.

[5] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720–748, Sept. 1999.

[6] Q. Cao and M. Zuo. A scheduling strategy supporting OpenMP task on heterogeneous multicore. In *26th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IPDPS 2012, Shanghai, China, May 21-25, 2012*, pages 2077–2084, 2012.

[7] Q. Chen, M. Guo, and Z. Huang. Cats: Cache aware task-stealing based on online profiling in multi-socket multi-core architectures. In *Proceedings of the 26th ACM International Conference on Supercomputing*, ICS ’12, pages 163–172, New York, NY, USA, 2012. ACM.

[8] Y. Ding, K. Hu, and Z. Zhao. Performance monitoring and analysis of task-based OpenMP. 2013.

[9] A. Duran, J. Corbalan, and E. Ayguade. An adaptive cut-off for task parallelism. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–11, Nov 2008.

[10] D. Eklov, D. Black-Schaffer, and E. Hagersten.

Statcc: A statistical cache contention model. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10, pages 551–552, New York, NY, USA, 2010. ACM.

[11] D. Eklöv and E. Hagersten. Statstack : Efficient modeling of LRU caches. In *Proc. International Symposium on Performance Analysis of Systems and Software : ISPASS 2010*, pages 55–65. IEEE, 2010.

[12] M. Frigo and V. Strumpen. The cache complexity of multithreaded cache oblivious algorithms. *Theory of Computing Systems*, 45(2):203–233, 2009.

[13] D. Genbrugge, S. Eyerman, and L. Eeckhout. Interval simulation: Raising the level of abstraction in architectural simulation. In *In High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–12. IEEE, 2010.

[14] P. Ghosh, Y. Yan, D. Eachempati, and B. M. Chapman. A prototype implementation of OpenMP task dependency support. In *OpenMP in the Era of Low Power Devices and Accelerators - 9th International Workshop on OpenMP, IWOMP 2013, Canberra, ACT, Australia, September 16-18, 2013. Proceedings*, pages 128–140, 2013.

[15] A. Jaleel, R. S. Cohn, C. keung Luk, and B. Jacob. Cmp$im: A pin-based on-the-fly multi-core cache simulator.

[16] D. Lorenz, P. Philippen, D. Schmidl, and F. Wolf. Profiling of OpenMP tasks with Score-P. In *41st International Conference on Parallel Processing Workshops, ICPPW 2012, Pittsburgh, PA, USA, September 10-13, 2012*, pages 444–453, 2012.

[17] D. Schmidl, P. Philippen, D. Lorenz, C. Rössel, M. Geimer, D. an Mey, B. Mohr, and F. Wolf. Performance analysis techniques for task-based OpenMP applications. In *OpenMP in a Heterogeneous World - 8th International Workshop on OpenMP, IWOMP 2012, Rome, Italy, June 11-13, 2012. Proceedings*, pages 196–209, 2012.

[18] J. Weidendorfer, M. Kowarschik, and C. Trinitis. A tool suite for simulation based analysis of memory access behavior. In *In Proceedings of International Conference on Computational Science*, pages 440–447. Springer, 2004.

[19] T. Weng and B. Chapman. Towards optimisation of openmp codes for synchronisation and data reuse. *Int. J. High Perform. Comput. Netw.*, 1(1-3):43–54, Aug. 2004.

[20] M. Wimmer, D. Cederman, J. L. Träff, and P. Tsigas. Work-stealing with configurable scheduling strategies. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '13, pages 315–316, New York, NY, USA, 2013. ACM.