Compositional Model of Coherence and NUMA Effects for Optimizing Thread and Data Placement

Hao Luo, Jacob Brock, Pengcheng Li, Chen Ding
Department of Computer Science
University of Rochester
{hluo, jbrock, pli, cding}@cs.rochester.edu

Chencheng Ye
College of Computer Science and Technology
Huazhong University of Science and Technology
ccye@hust.edu.cn

Abstract—On today’s multi-socket systems, the parallel performance is hampered by remote cache and memory access. There is much prior work on thread and data placement to curb remote access. However, the number of possible placements is large, and heuristic-based techniques only examine a fraction of the entire solution space.

This paper presents a compositional model to analyze the effect of thread and data placement choices. The model includes an analysis for cache coherence and (remote) memory access. It has the property of being compositional, meaning the performances of all the placements can be composed from the results of one profiling pass. Based on this model, this paper further introduces a prototype tool called Tapas to optimize parallel programs for non-uniform memory access (NUMA) platforms.

I. INTRODUCTION

Chip multiprocessors provide programmers with coherent, shared memory. Shared memory relieves programmers of the burden of communicating shared data between parallel tasks, especially when dynamic and complex data structures are used. Coherence is implemented in hardware following protocols that detect and enable data sharing.

On multi-socket systems, threads on different sockets may share data, and this sharing causes a thread on one socket to access the cache or the memory on another socket. Such an access is called a remote access. A remote cache access incurs one or more coherence messages, depending on the coherence protocol used by the machine. A remote memory access takes longer than a local (intra-socket) access. This effect is known as non-uniform memory access (NUMA).

The performance impact of coherence and NUMA access depends on thread and data placement. When we divide threads differently among sockets, we may reduce the data sharing between sockets, and hence the amount of coherence traffic. When we store a memory page on the home memory of the socket where the threads make the most access to the page, we minimize the remote access to the page.

The type of remote access depends on locality: whether the shared data reside in remote cache or not. If so, it is a coherence access; otherwise, it may be a remote or a local memory access, depending on data placement. Therefore, the best thread and data placement depends on both program locality and machine parameters, especially the cache size and the coherence protocol.

In this paper, we present a model to analyze the effects of these factors and their interaction. The most challenging factor is coherence locality, which includes the effects of a program’s data sharing and the machine’s coherence protocol.

There is extensive previous work to improve cache sharing among threads, including thread co-location [1], [2] and task scheduling [3]. Thread co-location swaps two threads at a time to gradually reduce the coherence miss count. It is incremental and makes one change each time. It builds a path of improvements. While the incremental solution is adaptive and has a low cost, it may take many steps, it may not converge to a stable solution, and the solution it has may not be the best. It also depends on the initial solution it uses.

We advocate an approach based on optimization instead of adaptation. Based on the run-time knowledge of thread data demands and interaction with the host machine, we try to derive the globally optimal solution.

Optimization must consider a broad range of possible solutions. Given the many factors of locality and machines mentioned earlier, the number of possible solutions is too large to test directly. Models are needed to evaluate the solutions that have not been tested. It is this observation that motivates us to design a model for fast evaluation.

This paper makes the following contributions:

- We propose a compositional model of the off-socket and inter-socket bandwidth to analyze performance of different thread and data placements.
- Based on this model, we introduce Tapas, a profiling based framework for optimizing thread and data placement.

II. LOCALITY MODEL

We model two performance factors of thread and data placement on NUMA architectures, the off-socket bandwidth demand and inter-socket bandwidth usage. For off-socket bandwidth demand, we extend the technique developed by Xiang et al. [4], which predicts the miss ratios of last level caches (LLCs). For inter-socket bandwidth usage, we analyze the number of coherence messages sent between sockets. We follow the coherence protocol to obtain the message count. The proposed approach is independent of which node the memory pages are allocated to; therefore, the obtained count information can be used to derive the inter-socket bandwidth usage for all data placements.
A. Off-socket Bandwidth Demand

The off-socket bandwidth demand depends on how threads are grouped on LLCs. The LLC miss ratios of sockets are predicted to quantify bandwidth demand. We choose miss ratio because it can be predicted efficiently with the higher order theory of locality (HOTL) described by Xiang et al. [4]. HOTL describes a locality metric, footprint, from which the miss ratio curve can be derived. The advantage of footprint over miss ratio is that footprint is composable, making it efficient in producing the LLC miss ratio curves for a large number of task-to-core mappings. However, the proposed technique is limited to the placement of sequential programs. We next discuss an extension to address this limitation, and propose a method to find the thread placement with minimal LLC miss ratios for parallel programs.

Calculated from a memory access stream, the footprint quantifies the average number of unique cache lines accessed in all time durations (as measured in cache accesses). In [4], Xiang et al. used footprint to quantify the locality of programs and the obtained per-program locality was used to guide scheduling.

We extend their approach to handle multi-threaded programs. Similar to [4], we measure footprint for each thread (denoted as \(FP_t\) for thread \(t\)) and try to obtain the footprint of any group of threads from composition. The challenge here is, depending on how much data are shared, the sum of multiple threads’ footprints does not accurately represent the footprint of the threads as a group. To tackle this problem, we further measure the footprint of the data shared between each thread pair, which we denote as \(SFP_{(u,v)}\) for pair \((u, v)\). The footprint of a two-thread group, \(u\) and \(v\), is thus \(FP_u + FP_v - SFP_{(u,v)}\). We approximate the footprint of a group, \(G\), as:

\[
\sum_{t \in G} FP_t - \sum_{u \in G, v \in G} SFP_{(u,v)}
\]

This computes the group’s footprint by summing its members’ footprints and subtracting the overlapped part. Here we assume that the majority of cache lines have no more than two sharers during a single lifetime in cache. Note that \(FP_t\) and \(SFP_{(u,v)}\) can be reused multiple times for different groups.

Once the footprint is obtained, we follow Xiang et al.’s procedure to predict the cache miss ratio [4]. To select thread placement, all the thread placements can be examined and their LLC miss ratios can be predicted. Due to the exponentially large number of possible placements, we follow a best-effort approach in practice.

B. Inter-socket Bandwidth Usage

The inter-socket bandwidth usage is modeled by an approximate number of the coherence messages sent between sockets. This is dependent on both thread and data placement. The program’s memory access stream is analyzed according to the coherence protocol on the interconnect. The analysis is decoupled from the way threads are grouped and which node the memory pages are allocated on. The analysis reads through the memory access stream and records statistics that are processed later to model inter-socket traffic.

Scanning through the memory access stream, our model simulates the cache coherence protocol. The simulation tracks two records: how many times a thread supplies a cache line requested by another thread, and how many times a cache line is written to or fetched from memory.

The first record gives the total bandwidth usage for transferring data between caches. The second gives the bandwidth for writing/fetching data to/from memory. After going through the stream once, these two statistics can be recorded completely in two tables. Note that the values in the tables are decoupled from any specific data placement, yet the bandwidth usages of all specific placements can be read. This property makes the analysis compositional.

III. COMPOSITIONAL MODEL BASED OPTIMIZATION

Tapas is a profiling guided framework for optimizing a parallel program’s thread and data placement. It is built on the model we presented above and fully exploits the model’s compositionality. The core idea of Tapas is as follows: instead of incrementally tuning performance using empirical heuristics, Tapas examines a large number of randomly chosen thread placements at once and select the best one. Because the strength of a compositional model is its efficiency in processing a large number of choices, this approach does not pose significantly more overhead than empirical tuning, but it greatly enlarges the scope of the analyzed placements.

Tapas consists of two major components: a runtime system for profiling, and a modeling backend for searching for the global optimum. The runtime system samples the memory accesses and records them on disk. After program execution, the modeling backend goes through the memory access stream to look for a thread placement that minimizes the off-socket bandwidth demand as described in Section II-A. Once the thread placements are evaluated, the backend examines data placements to optimize for inter-socket bandwidth usage.

IV. CONCLUSION

This paper describes a compositional model for analyzing thread and data placement on multi-socket systems and a prototype optimizer called Tapas. As future work, we hope to evaluate Tapas in detail and compare it with other relevant work [2]. This would help us better assess compositional models for their strengths and limitations.

REFERENCES


