Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Two-level main memory co-design: Multi-threaded algorithmic primitives, analysis, and simulation[☆]

Michael A. Bender[a], Jonathan W. Berry[b,*], Simon D. Hammond[b], K. Scott Hemmert[b], Samuel McCauley[a], Branden Moore[b], Benjamin Moseley[c], Cynthia A. Phillips[b], David Resnick[b], Arun Rodrigues[b]

[a] *Stony Brook University, Stony Brook, NY 11794-2424, USA*

[b] *Center for Computing Research, Sandia National Laboratories, Albuquerque, NM 87185, USA*

[c] *Washington University in St. Louis, St. Louis, MO 63130, USA*

HIGHLIGHTS

- Emerging architectures feature high-bandwidth scratchpad memory that is not cache.
- We adapt a classical external-memory model to this context.
- We design and analyze algorithms for sorting and k-means using this model.
- We validate our predictions with simulation and/or a custom machine.

ARTICLE INFO

Article history:
Received 29 October 2015
Received in revised form
2 September 2016
Accepted 3 December 2016
Available online 3 January 2017

Keywords:
Two-level memory
High-bandwidth memory
Sorting
k-means clustering

ABSTRACT

A challenge in computer architecture is that processors often cannot be fed data from DRAM as fast as CPUs can consume it. Therefore, many applications are memory-bandwidth bound. With this motivation and the realization that traditional architectures (with all DRAM reachable only via bus) are insufficient to feed groups of modern processing units, vendors have introduced a variety of non-DDR 3D memory technologies (Hybrid Memory Cube (HMC),Wide I/O 2, High Bandwidth Memory (HBM)). These offer higher bandwidth and lower power by stacking DRAM chips on the processor or nearby on a silicon interposer. We will call these solutions "near-memory," and if user-addressable, "scratchpad." High-performance systems on the market now offer two levels of main memory: near-memory on package and traditional DRAM further away. In the near term we expect the latencies near-memory and DRAM to be similar. Thus, it is natural to think of near-memory as another module on the DRAM level of the memory hierarchy. Vendors are expected to offer modes in which the near memory is used as cache, but we believe that this will be inefficient.

In this paper, we explore the design space for a user-controlled multi-level main memory. Our work identifies situations in which rewriting application kernels can provide significant performance gains when using near-memory. We present algorithms designed for two-level main memory, using divide-and-conquer to partition computations and streaming to exploit data locality. We consider algorithms for the fundamental application of sorting and for the data analysis kernel *k*-means. Our algorithms asymptotically reduce memory-block transfers under certain architectural parameter settings. We use and extend Sandia National Laboratories' SST simulation capability to demonstrate the relationship between increased bandwidth and improved algorithmic performance. Memory access counts from simulations corroborate predicted performance improvements for our sorting algorithm. In contrast, the *k*-means algorithm is generally CPU bound and does not improve when using near-memory except under extreme conditions. These conditions require large instances that rule out SST simulation, but we demonstrate improvements by running on a customized machine with high and low bandwidth memory. These case studies in co-design serve as positive and cautionary templates, respectively, for the major task

[☆] This was selected as a best Paper from International Parallel and Distributed Processing Symposium (IPDPS) 2015.

[*] Corresponding author.

E-mail addresses: bender@cs.stonybrook.edu (M.A. Bender), jberry@sandia.gov (J.W. Berry), sdhammo@sandia.gov (S.D. Hammond), kshemme@sandia.gov (K. Scott Hemmert), smccauley@cs.stonybrook.edu (S. McCauley), bjmoor@sandia.gov (B. Moore), moseleyb85@gmail.com (B. Moseley), caphill@sandia.gov (C.A. Phillips), drresni@sandia.gov (D. Resnick), afrodni@sandia.gov (A. Rodrigues).

http://dx.doi.org/10.1016/j.jpdc.2016.12.009

of optimizing the computational kernels of many fundamental applications for two-level main memory systems.

1. Introduction

Recently vendors have proposed a new approach to improve memory performance by increasing the bandwidth between cache and memory [26,27]. The approach is to bond memory directly to the processor chip or to place it nearby on a silicon interposer. By placing memory close to the processor, there can be a higher number of connections between the memory and caches, enabling higher bandwidth than current technologies. While the term *scratchpad* is overloaded within the computer architecture field, we use it throughout this paper to describe a high-bandwidth, local memory that can be used as a temporary storage location. Note that this term also refers to high speed internal memory used for temporary calculations [41,8] which is a different technology than that discussed in this paper.

The scratchpad cannot replace DRAM entirely. Due to the physical and cost constraints of adding the memory directly to the chip, the scratchpad cannot be as large as DRAM, although it will be much larger than cache, having gigabytes of storage capacity. Since the scratchpad is smaller than main memory, it does not fully replace main memory, but instead augments the existing memory hierarchy.

The scratchpad has other limitations besides its size. First, the scratchpad does not significantly improve upon the latency of DRAM. Therefore, the scratchpad is not designed to accelerate memory-latency-bound applications, but rather memory-bandwidth-bound ones. Second, adding a scratchpad does not improve the bandwidth between DRAM and cache. Thus, the scratchpad will not accelerate a computation that consists of a single scan of a large chunk of data that resides in DRAM.

This gives rise to a new multi-level memory hierarchy where two of the components – the DRAM and the scratchpad– work in parallel. We view the DRAM and scratchpad to be on the same *level* of the hierarchy because their access times are similar. There is a tradeoff between the memories: the scratchpad has limited space and the DRAM has limited bandwidth.

Since the scratchpad does not have its own level in the cache hierarchy, when a record is evicted from the cache there is an algorithmic decision whether to place the record in the scratchpad or directly into the DRAM. In the currently-proposed architecture designs, this decision is user-controlled.

Under the assumption that the memory is user-controlled, an algorithm must coordinate memory accesses from main memory and the scratchpad. Ideally the faster bandwidth of the scratchpad can be leveraged to alleviate the bandwidth bottleneck in applications. Unfortunately, known algorithmics do not directly apply to the proposed two-level main-memory architecture. The question looms, can an algorithm use the higher bandwidth scratchpad memory to improve performance? Unless this question is answered positively, the proposed architecture, with its additional complexity and manufacturing cost, is not viable.

Our interest in this problem comes from *Trinity* [25], the latest NNSA (National Nuclear Security Administration) supercomputer architecture. This supercomputer uses the Knight's Landing processor from Intel with Micron memory. This processor chip uses such a two-level memory [27]. The mission for the Center of Excellence for Application Transition [26], a collaboration between NNSA Labs (Sandia, Los Alamos, and Lawrence Livermore), Intel, and Cray, is to ensure applications can use these new architectures.

1.1. Results

In this paper we introduce an algorithmic model of the scratchpad architecture, generalizing existing sequential and parallel external-memory models [1,4]. (See Section 9 for a brief background on the architectural design and motivation for the scratchpad.) We introduce theoretically optimal scratchpad-optimized, sequential and parallel sorting algorithms. We report on hardware simulations varying the relative-bandwidth parameter. These experiments suggest that the scratchpad will improve running times for sorting on actual scratchpad-enhanced hardware for a sufficient number of cores and sufficiently large bandwidth improvement. We also study the data analysis technique k-means clustering for contrast.

Theoretical contributions

We give an algorithmic model of the scratchpad, which generalizes existing sequential and parallel external-memory models [1,4]. In our generalization, we allow two different block sizes, B and ρB ($\rho > 1$) to model the bandwidths of DRAM and the larger bandwidth of the scratchpad. Specifically, $\rho > 1$ is the relative increase in bandwidth of the scratchpad in comparison to DRAM.

We give scratchpad-aware algorithms for two basic problems: sorting and k-means clustering. In both cases, we obtain asymptotic reductions in memory block transfers compared to conventional algorithms. However, we find that our sorting algorithms are more practical than our k-means algorithms. Our experience in designing and experimenting with these algorithms has led us to enumerate several basic properties of successful scratchpad-aware algorithms. We present these in Section 1.1.

We exhibit, under reasonable architectural assumptions of the scratchpad, sorting algorithms that can achieve speedups proportional to ρ over the state-of-the-art sorting algorithms that use only far DRAM. We begin by introducing a sequential algorithm for sorting, and then generalize to the multiprocessor (single node) setting. We complement this result by giving a matching lower bound in our theoretical model. Our algorithms supply theoretical evidence that the proposed architecture can indeed speed up fundamental applications.

Algorithm engineering contributions

For an application to benefit from a scratchpad-enhanced memory architecture, good data locality is necessary, but it is not sufficient. For example, consider a program that scans a large array once. This program has great data locality. However, running this program on a memory system with a scratchpad will not help. Moving the data to the scratchpad before computation is effectively just an extra copy.

The following definitions help describe the conditions under which a scratchpad-enhanced memory can help. A computation is *memory bound* if the limiting factor in its running time is the time to feed data to the processors. Adding more compute power does not improve runtime in any significant way, but delivering data faster will. A process is *compute bound* if the limiting factor is the