Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Analysis of classic algorithms on highly-threaded many-core architectures

Lin Ma[a,b,*], Roger D. Chamberlain[a], Kunal Agrawal[a], Chen Tian[b], Ziang Hu[b]

[a] *Washington University in St. Louis, St.Louis, MO, USA*
[b] *Huawei America Research Center, Santa Clara, CA, USA*

HIGHLIGHTS

- Analyze effect of memory latency hiding by threads and determine the algorithm bound.
- A wide range of algorithms analyzed on a wide spectrum of architectures including both NVIDIA and AMD GPUs and XMT machines.
- Analysis is accurate compared with our and other researchers' experimental findings.
- Predict important, non-trivial, and previously unexplained trends and artifacts in empirical data.
- Verify the TMM model is effective at predicting effect of changing various parameters on diversified many-core machines.

ARTICLE INFO

Article history:
Received 29 February 2016
Received in revised form
8 January 2017
Accepted 6 February 2017
Available online xxxx

Keywords:
Threaded Many-core Memory (TMM) model
GPU
XMT
Algorithm analysis
Modeling
Simulation and evaluation techniques of HPC systems

ABSTRACT

The recently developed ***Threaded Many-core Memory (TMM)*** model provides a framework for analyzing algorithms for highly-threaded many-core machines such as GPUs and Cray supercomputers. In particular, it tries to capture the fact that these machines hide memory latencies via the use of a large number of threads and large memory bandwidth. The TMM model analysis contains two components: computational and memory complexity.

A model is only useful if it can explain and predict empirical data. In this work, we investigate the effectiveness of the TMM model. Under this model, we analyze algorithms for 5 classic problems— suffix tree/array for string matching, fast Fourier transform, merge sort, list ranking, and all-pairs shortest paths—on a variety of GPUs. We also analyze memory access, matrix multiply and a sequence alignment algorithm on a set of Cray XMT supercomputers, the latest NVIDIA and AMD GPUs. We compare the results of the analysis with the experimental findings of ours and other researchers who have implemented and measured the performance of these algorithms on a spectrum of diverse GPUs and Cray appliances. We find that the TMM model is able to predict important, non-trivial, and sometimes previously unexplained trends and artifacts in the experimental data.

1. Introduction

Highly-threaded many-core architectures such as Graphics Processing Units (GPUs) and Cray Extreme Multi-Threading (XMT) supercomputers are increasingly being exploited as powerful compute engines. For these architectures, a number of high-performance algorithms have been developed, such as sorting [1], hashing [2], dynamic programming [3], graph algorithms [4], matrix multiplication [5], and other classic algorithms [6]. Many performance studies have also been conducted [7–9] to understand the performance of applications on such machines.

We are interested in the theoretical asymptotic analysis of algorithms, since it allows us to compare the high-level performance characteristics of algorithms without worrying about low level implementation details of both the algorithm and the particular machine. PRAM [10] is the most common model for theoretical analysis of parallel algorithms. However, the PRAM model ignores some important characteristics of GPU-like highly-threaded machines. For example, PRAM assumes that all memory accesses take constant time; this assumption is violated by the complex memory subsystems of most modern machines and is especially not a reasonable assumption for GPUs and XMTs. Recently, a few models for asymptotic analysis of GPU algorithms

* Corresponding author at: Huawei America Research Center, Santa Clara, CA, USA.
E-mail address: lin.ma@wustl.edu (L. Ma).

http://dx.doi.org/10.1016/j.future.2017.02.007
0167-739X/© 2017 Elsevier B.V. All rights reserved.

have been proposed [11,12] that do try to take important characteristics of these machines into consideration.

In this paper, we focus on one of these models, namely, the ***Threaded Multi-core Memory (TMM)*** model [11]. In the TMM model, a program is analyzed in terms of both its computational complexity and its memory complexity. Computational complexity analysis is identical to the standard analysis used by parallel algorithms, in particular, by the PRAM model. However, the TMM model is meant to capture important characteristics of the memory subsystem of highly-threaded many-core machines, such as GPUs. Therefore, the analysis framework for memory complexity in the TMM model takes three important characteristics of these machines into consideration. (1) There is a large latency for accessing the largest and slowest memory on the machine, namely the global memory. (2) These machines have a large number of hardware managed threads and use fast context switching between these threads to hide this latency. (3) In addition to the fast context switching, these machines typically have a large memory bandwidth between the core and the slow memory; therefore, if the memory accesses are regular and predictable, then many memory operations can be *grouped* into one large bandwidth memory transfer. Therefore, in this model, the memory complexity of a program depends not only on how many memory accesses it contains, but also on both how many threads it can use and how effectively its memory accesses can be grouped. Typical examples of such machines include both NVIDIA and AMD/ATI GPUs and the Cray YarcData uRiKA system. We do not try to model the Intel Xeon Phi, because it has limited thread count, typically insufficient for latency hiding. In contrast, its approach to hide memory latency is primarily based on strided memory access patterns associated with vector computation.

In the TMM model, the total running time of an algorithm on P cores depends on both its computational complexity and memory complexity. The computational complexity is derived in terms of ***work*** T_1—the total amount of computation, or, in other words, its running time on 1 processor – and ***span*** T_∞[1]—the amount of computation on the critical path or, in other words, the running time on an infinite number of processors. The memory complexity is derived using a parameter M, which is the total number of memory transfers (either grouped or not) from slow memory. The total running time on P cores is then derived using both computational and memory complexity, as well as machine parameters such as the latency to slow memory, memory bandwidth to slow memory, and number of available hardware threads.

Asymptotic models are meant to capture important characteristics of the machine while ignoring low-level details; however, a model is only useful if it can predict and explain empirical data. Like all asymptotic models, the TMM model makes a few different categories of predictions: (1) For a given algorithm, it predicts the impact of increasing problem sizes on performance. (2) When we have multiple algorithms for the same problem, asymptotic models can help us compare them and understand relative performance at a high level. However, unlike the standard RAM or PRAM models, these predictions may depend on some machine parameters, such as the relationship between memory latency, the fast memory size, and the number of allowable threads on the machine.[2] In addition, unlike the PRAM model, the TMM model also allows us consider the effects of changing the parameters of the machine itself (such as reducing memory latency or changing the fast memory size) on the performance of an algorithm.

In this paper, we evaluate the effectiveness of the TMM model in terms of its ability to predict empirical performance of algorithms. We extend our analysis [13,14] of a number of classic algorithms with a wider range and more in-depth investigation under the TMM model. Specifically, we analyze suffix trees and suffix arrays for the problem of string matching, Fast Fourier Transform (FFT), merge sort, Wylie's list ranking, looped memory accesses, matrix multiplication, and sequence alignment. For all of these algorithms, we compare the analytical conclusions derived using the TMM model to the empirical observations of researchers who have previously implemented these algorithms, in order to investigate whether the TMM analysis correctly predicts the trends observed in the empirical data, and whether it can explain previously unexplained characteristics of the data. In addition, we also take a second look at an all-pairs shortest paths algorithm, which was previously analyzed in the original TMM paper [11], and perform additional experiments on two more latest GPU architectures, both to investigate if the TMM model applies to the newest GPU machines and to investigate the effect of changing machine parameters.

Our findings indicate the TMM model is effective at explaining many kinds of empirical observations, and its analysis framework appears to be well suited to understanding and predicting the high-level characteristics of the performance of algorithms on all these machines. In particular, one can compare the effect of computational and memory complexity on the runtime for various parameter settings, and explain the behavior of algorithms for varying problem sizes. In addition, the TMM model also predicts the effect of changing the machine parameters, such as memory latency, on the performance of algorithms. Note that the TMM model ignores constant factors; therefore, it can only explain high-level trends, not particular numerical values at which these trends may occur. For all the classic algorithms considered in this paper, the TMM analysis indicates that considering both computational and memory complexity is necessary to understand the performance of algorithms on highly-threaded many-core machines such as GPUs and XMTs.

2. Threaded many-core memory model

The TMM model [11] models highly-threaded many-core machines as consisting of a number of core groups (called streaming multiprocessors on NVIDIA GPUs, compute unit on AMD GPUs, and Threadstorm processors on Cray XMTs). Some of these machines like GPUs contain a number of cores and a fast local on-chip memory of size Z shared within a core group. Computation and access to fast memory takes unit time in the TMM model. Some machines like XMTs have only one core per Threadstorm processor as a core group.

A large slow global memory is shared by all the core groups and accessing the slow memory takes L time steps (L is the ***memory latency***). Data is transferred from slow to fast memory in ***chunks*** of maximum size C, also called the ***chunk size*** or ***memory access width***; this represents the large bandwidth between slow and fast memory.[3] These machines support a large number of hardware threads, much larger than the total number of cores P, and these threads are used to hide the memory latency. The hardware limit on the ***number of threads per core*** is represented by X; the total number of threads supported on the machine is therefore bounded by XP.

[1] Also called depth, time, or critical-path length in the literature. We reserve the term "time" for the running time on P processors, not an infinite number of processors.

[2] This is not very different from other models that also consider memory subsystem, such as DAM (see Section 10).

[3] The chunk can either be a cache line of hardware managed caches or an explicitly-managed combined read from multiple threads.