Towards Modelling Parallelism and Energy Performance of Multicore Systems

Bogdan Marius Tudor* and Yong Meng Teo
[*bogdan,teoym*]@*comp.nus.edu.sg*

***Abstract*—Multicore systems are increasingly adopted across many application domains. Consequently, understanding their performance is becoming an important issue for a growing number of users. However, performance analysis of parallel programs on multicore systems is still challenging, especially for large programs or applications developed in multiple programming languages. This paper proposes an analytical modelling approach for studying the parallelism and energy performance of shared-memory programs on multicore systems. The proposed model derives the speedup and speedup loss from data dependency and memory overhead in traditional UMA and NUMA multicore systems, and emerging platforms such as ARM multicores. Using only widely available inputs derived from the trace of the operating system run-queue and hardware events counters, the proposed model achieves high practicality and generality across many types of shared-memory programs running on different multicore platforms. Applications of the model include understanding achieved speedup and parallelism loss, and prediction of optimal core and memory configuration, where the optimality criteria is minimum execution time, minimum energy usage or a trade-off between these two.**

I. Introduction

Parallel systems with large number of cores now form the backbone of processing in a wide range of devices, including embedded systems, graphic processing units and large servers. Furthermore, technology trends consistently show that multicores are growing in at least two directions: widening of adoption and increase in core counts with each technology generation. However, the memory bandwidth that is shared among cores is increasing at a much slower rate, because of wire delays and power dissipation, among others. Thus, off-chip memory bandwidth available per core is not keeping pace with the growth in the number of cores. Furthermore the memory capacity available per dollar continues to grow according to Moore's Law, and therefore larger problem sizes become feasible to be executed on multicore systems. These two trends suggest that memory contention among cores is an increasing concern in exploiting multicore systems. Coupled with the hardware shift to multicore, software is undergoing an evolution. Traditional parallel programming techniques such as Fortran, C or C++ supported by POSIX threads and OpenMP are joined by new software systems such as Cilk, Fortress or X10 for shared-memory systems, CUDA and OpenCL for GPGPU, among many other parallel programming tools.

These changes in hardware and software systems pose new challenges in understanding the performance of multicore systems. Multicore brings a growing spectrum of parallel programming options such as programming models, languages, types of multicore systems, problem size, number of threads, number of cores, thread-to-core mapping and memory architecture, among others. This leads to significant challenges in understanding the performance loss associated with each choice. With the wide adoption of multicore systems, there is a growing need for performance analysis methods that are general enough to be applied across both software and hardware platforms.

Current performance analysis approaches can be compared based on three key design trade-offs: ease of use, intrusiveness of the method and accuracy of the results. Recently, a shift in analysis methods recognize that the performance of large parallel programs depends on a multi-dimensional space of options and configuration parameters, including programming models, number of threads and processor cores, problem size, memory architecture, thread-to-core placement among others. Therefore, the ease of applying the performance analysis methods across this parameter space is a crucial design criteria [3], [4]. Methods that rely on empirical data, such as regression based-approaches, neural networks and machine learning [3], [5] typically produce good accuracy but require significant modelling effort or large volume of training data. On the other hand, traditional methods for performance analysis include software instrumentation methods and trace-driven analysis [1], [2]. However, while they have good accuracy, these approaches are intrusive and often tailored to a particular programming language or platform. This leads to difficulty in generalizing them across programming languages and models or across different hardware platforms. Furthermore, analytical models are easy to apply, but often the simplifying assumptions about the hardware platform reduce their accuracy below practical usefulness. Finally, analytical models often do not use inputs which are easily available [6], [7]. Therefore there is a need for performance analysis methods that have minimal intrusiveness and can scale across different platforms.

The rest of this paper is organized as follows: Section II describes the objective, approach and contributions, then in section III we present an overview of the modelling approach. Section IV shows brief validation results and two applications of the model for understanding the performance and optimizing the execution of a shared-memory program. The paper is concluded in section V by presenting current status and further work.

II. Objective and Approach

The objective of our work is to develop an analytical modelling framework for understanding the parallelism and energy performance of shared-memory programs, across multiple programming languages and multicore architectures. Our model predicts the speedup and energy require-

ments of shared-memory programs, and the speedup loss due to data-dependency and memory contention among cores. Using predictions from our model, parallel program executions can be optimized by maximizing the speedup, minimizing energy requirements or achieving a trade-off between speedup and energy use.

Our approach is supported by a general parallelism model that splits the lifetime of a program into useful work, data-dependency and overhead induced by memory contention among cores. This general parallelism model comprises three sub-models: (i) model for predicting the data-dependency among threads, (ii) model of memory contention among cores for UMA and NUMA systems and (iii) model of energy use. To generalize the models across software and hardware platforms, we use two non-intrusive and widely-available sets of metrics reported by modern systems: the dynamic size of the operating system run-queue as the proxy for program parallelism, and hardware events counters. We target programs with large compute and memory requirements and therefore we do not address workloads where the impact of I/O, storage and network performance is crucial. Furthermore, we address multicore architectures with homogeneous cores.

Figure 1 presents the approach for applying the model. The target applications include HPC dwarfs, real-world

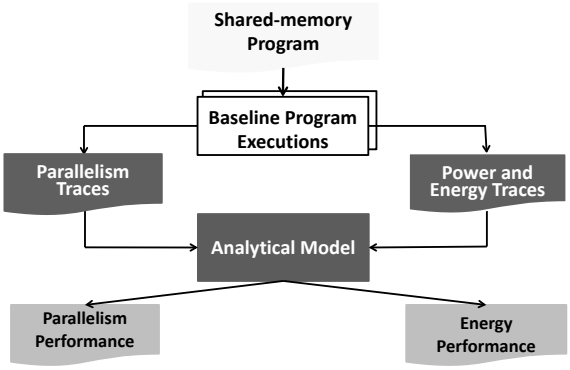

Figure 1. Approach for Predicting Parallelism and Energy Performance

parallel applications and emerging workloads such as mobile apps on a wide range of multicore architectures, such as commodity Intel/AMD systems and low-power ARM multicores. For a program and a target platform, we perform a small number of baseline executions. During these executions, which are conducted on a small number of cores, we collect parallelism traces (i.e. run-queue and memory contention traces), and power and energy traces. Using these as the inputs of our analytical models, we can predict parallelism and energy performance for the given program on different number of cores and memory configurations.

The contributions of our work can be summarized as follows:

1) Practical analytical models for understanding the performance of large parallel applications on modern architectural systems. The practicality of the models stems from their generality across programming languages/threading packages and architectures, and from new insights on the decrease of memory burstiness in multicore systems, for large problem sizes. These insights are based on extensive experiments on state-of-the-art multicore systems, with core counts up to 4 in ARM systems, and 48 in commodity UMA and NUMA systems [8], [9].
2) Using predictions of the model, we optimize execution of parallel programs by adjusting the number of cores or the core frequency in systems with dynamic frequency scaling. The optimality criteria ranges from fastest execution time to minimum energy usage, and trade-offs between these two [8].

III. Parallelism and Energy Performance Models

A. Model of Parallelism

Our general model of parallelism splits the lifetime of a shared-memory program into useful work and parallelism loss. The parallelism loss is further divided into data-dependency and memory overhead. Given a program that is partitioned into m threads and executes on n cores, $A(m,n)$ denotes the average number of active threads. However, due to overhead caused by memory contention, only a subset of the active threads will be performing useful work. We define $\omega(n)$, the memory contention factor, as the number of threads busy due to memory overhead to the number of threads busy due to useful work. With these notations, following the derivations described in [8], the speedup of a shared-memory program is:

$$S(m,n) = \frac{A(m,n)}{1+\omega(n)} \tag{1}$$

We consider the parallelism loss due to data dependency in the program, $D(m)$, to be the difference between the the available threads m, and the average number of active threads, given unbounded execution resources, $A(m,\infty)$:

$$D(m) = m - A(m,\infty) \tag{2}$$

The parallelism loss due to memory contention is:

$$R(n) = A(m,n)\frac{\omega(n)}{1+\omega(n)} \tag{3}$$

Equation 1 expresses the speedup of a program as a function of average number of threads $A(m,n)$ and memory contention factor $\omega(n)$. This general model is refined using the data dependency model for determining $D(m)$ and $A(m,n)$, and the memory contention model for determining $\omega(n)$ for UMA and NUMA systems.

B. Data Dependency Model

The objective of the data-dependency model is the predict $A(m,n)$ for an program with m threads, on an arbitrary number of cores, n. The approach is to perform a baseline execution of the program on b cores, where $m > b$. Because there are more threads than cores, some threads will queue for service in the operating system run-queue. The key idea behind our approach is that the parallelism of the program

at any time moment is the sum of the number of threads executing at that time moment and of the number of threads queueing in the run-queue.

During the baseline execution we trace the dynamic size of the run-queue and the number of active cores. We then compute the average number of active threads $A(m, n)$ as a weighted average of the parallelism of the program taking into account the load-balancing effect caused by over-subscription. After determining the average number of active cores, we use equation 2 to compute the parallelism loss due to data-dependency. We implemented this approach as a tool that samples the dynamic size of the run-queue using the `procfs` pseudo-filesystem which is widely-available on many operating systems.

C. Model of Memory Contention

The objective of the memory contention model is to predict the average number of threads busy due to memory contention among cores, for UMA and NUMA memory architectures. The modelling approach is to split the number of cycles incurred by a program on n cores into work cycles, stall cycles induced by memory contention and stall cycles unrelated to memory contention (i.e. pipeline hazards, branch stalls, latency of hitting the caches etc.). The work cycles are defined as cycles in which at least one integer or floating point operation is retired, while stall cycles are defined as cycles with no retired instructions.

Using extensive measurement analysis on state of the art multicore systems [8], [9], we conclude that for weak-scaling programs the last-level misses, number of cycles unrelated to memory contention and work cycles do not change significantly when n changes. Furthermore, we study the pattern of burstiness of the memory requests and conclude that large parallel programs do not exhibit bursty memory traffic [9]. Based on these insights, we propose a simple single-server queueing model for predicting the number of cycles incurred by a program on a single-socket system. We then extend this simple model for UMA and NUMA multiprocessors. The memory model requires two baseline runs for a single-socket system and two baseline runs plus another run for each NUMA latency in a multi-socket NUMA system.

D. Energy Model

The objective of the energy model is to predict the power and energy requirements of a program when executing on a target multicore architecture. The approach behind the energy model closely follows the memory contention model. We consider dynamic frequency scaling cores and fixed frequency memory. The total power drawn by the system when executing a program program is divided into idle power, power drawn by active cores and power drawn by memory. These three power values are measured during a set of baseline runs, using differential analysis between power consumption of idle system and power consumption when n cores are active, for a given core frequency. The number and configuration of the baseline runs is the same as for the memory model, plus a measurement of power consumption under idle load.

The energy used by the program is determined as the product of execution time, average number of active threads and power. The optimal performance configuration is then determined using bottleneck analysis: for low core counts or low core frequencies, the cores are the bottleneck, but for higher core counts or high core frequencies, the bottleneck shifts to the memory.

E. Limitations

There are three main sources of limitation in our model. When a program is composed of programming language tasks, such as OpenMP 3.0 tasks or Cilk tasks, the degree of parallelism of the program is affected by a run-time scheduler. In this case, oversubscribing the cores might result in a different degree of parallelism profile across different runs of the program. Secondly, if thread synchronize through busy waiting for significant periods of time, our model underestimates the data-dependency of the program. Third, the accuracy of the memory contention model decreases when the ratio of local to remote memory accesses differs among the threads.

IV. Model Application

We validated our model by comparing model predictions against measurements of speedup on three state-of-the art commodity multicore system with 8 cores (UMA), and 24 and 48 cores (NUMA). In general, applications with large memory contention and complex inter-thread synchronization, such as HPC programs, are more challenging to predict. Therefore, for validation and evaluation purposes, we included applications from the PARSEC 2.0 suite that use pthreads, and weak-scaling HPC dwarfs from NPB 3.3, using OpenMP. The average prediction error across more than two hundreds experiments with four problem sizes, and core counts ranging from 2 to 48 is 9% for UMA systems and 14% for NUMA systems. The accuracy of the model is strongly determined by the problem size, with best accuracy for larger problem sizes, due to large steady-state compute phases [8] and uniform memory access patterns [9].

We show an application of our model for understanding the performance of a program. We apply the model to predict the speedup and speedup loss as a factor of the number of active cores and problem size. We model the performance of OpenMP Fortran HPC dwarf SP, for up to eight cores. We perform this analysis for problem sizes small (W), medium (A), large (B) and very large (C). Figure 2 shows that program SP has a counter-intuitive behavior: *the speedup reduces as the problem size is increased*. Data-dependency reduces with the increase in problem size, because large problem size amortize the gaps caused by thread synchronization over longer execution time. However, memory contention increases with problem size, because larger working sets increasingly exceed the size of the caches

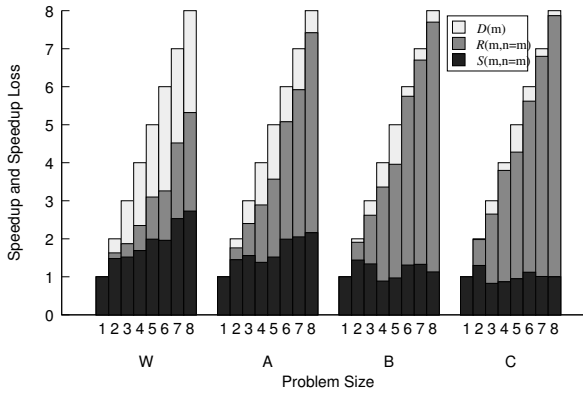

Figure 2. Understanding the performance of program SP for different number of cores and problem sizes

and cause intensive memory traffic. As a result, for large problem sizes B and C, the overall effect is that the achieved speedup does not exceed 1.9 even when using all eight cores.

Figure 3 shows an application of our model to optimize the execution of program SP. We apply our model to predict

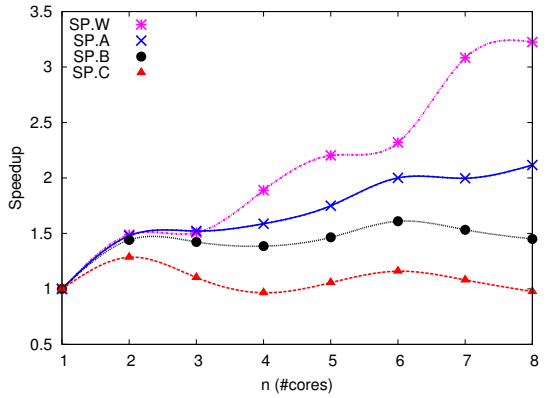

Figure 3. Measured speedup of program SP on different problem size

the number of cores that minimizes execution time for a dual-socket UMA system with total of eight cores. Using numerical differentiation, we apply numerical differentiation to determine the number of cores that maximizes equation 1 using inputs collected from baseline runs on 1, 2 and 5 cores. For program SP, we determined the optimal n that maximizes the speedup for problem sizes W to C: n_W = 12, n_A = 8, n_B = 2, n_C = 2. For problem size B and C, the optimal number of cores are much smaller than maximum number of cores of the machine. We compare the results against speedup measurement. The measurements are shown in figure 3 and they are a good match to the model prediction. This shows that for programs with large memory contention, predicting the number of cores that maximizes speedup is not straightforward, and allocating the entire number of cores of the machine can reduce speedup and increase resource utilization, thus increasing the energy use twofold. More details about the model applications are presented in [8].

V. Conclusions and Further Work

This paper describes a modelling approach for analyzing the parallelism and energy performance of shared-memory programs on multicore systems. The model determines the speedup, speedup loss due to data-dependency and memory overhead, and the energy usage of an application across different types of multicore systems. Our work can be applied to understand and optimize the execution of large shared-memory programs, for which traditional performance analysis methods relying on instrumentation are impractical.

Currently all components of the modelling framework have been evaluated on commodity UMA and NUMA multicore systems with up to 48 cores. Further work includes evaluating the model on emerging platforms such as state-of-the-art quad-core ARM systems, with the goal of optimizing energy usage through dynamically modifying core frequencies for programs with large memory requirements.

References

[1] M. Kumar, "Measuring Parallelism in Computation-Intensive Scientific/Engineering Applications," *IEEE Transactions on Computers*, 37(9):1088–1098, 1988

[2] D. Sehr and L. V. Kale, "Estimating the Inherent Parallelism in Prolog Programs," in *Proc. of 3rd International Conference on Fifth Generation Computer Systems*, pages 783–790, Tokyo, Japan, 1992

[3] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski and M. Schulz, "Prediction Models for Multi-dimensional Power-performance Optimization on Many Cores," in *Proc. of 17th International Conference on Parallel Architectures and Compilation Techniques*, pages 250–259, Toronto, Canada, 2008

[4] E. Thereska, B. Doebel, A. X. Zheng and P. Nobel, "Practical Performance Models for Complex, Popular Applications," in *Proc of 31st ACM International Conference on Measurement and Modeling of Computer Systems*, pages 1–12, New York, USA, 2010

[5] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski and M. Schulz, "A Regression-based Approach to Scalability Prediction," in *Proc. of 22nd Annual International Conference on Supercomputing*, pages 368–377, Island of Kos, Greece, 2008

[6] D. Eager, J. Zahorjan and E. Lazowska, "Speedup vs Efficiency in Parallel Systems," *IEEE Transactions on Computers*, 38(3):408–423, 1989

[7] K. Sevcik, "Characterizations of Parallelism in Applications and Their Use in Scheduling," in *Proc. of 10th ACM Conference on Measurement and Modeling of Computer Systems*, pages 171–180, Berkeley, USA, 1989

[8] B. M. Tudor and Y. M. Teo, A Practical Approach for Performance Analysis of Shared-Memory Programs, *Proc. of 25nd International Parallel & Distributed Processing Symposium*, Anchorage, USA, 2011

[9] B. M. Tudor, Y. M. Teo and S. See, Understanding Off-chip Memory Contention of Parallel Programs in Multicore Systems, *Proc. of 40th International Conference on Parallel Processing*, Taipei, Taiwan, 2011