PROPHET: Goal-Oriented Provisioning for Highly Tunable Multicore Processors in Cloud Computing

Dong Hyuk Woo Hsien-Hsin S. Lee

School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA 30332
dhwoo@ece.gatech.edu leehs@gatech.edu

Abstract

In this article, we propose PROPHET, a goal-oriented provisioning infrastructure based on execution history profile gathered from the cloud of distributed heterogeneous computing environment. It can autonomously tune the efficiency of a data center or satisfy the end-users' need when running network-based applications. With more tunable features provided by future multicore and many-core processors, we envision that PROPHET can be easily integrated into today's network infrastructure to provide value-added service to many of us.

Categories and Subject Descriptors C.1.2 [*Processor Architectures*]: Multiple Data Stream Architectures (Multiprocessors); D.4.1 [*Operating Systems*]: Process Management—scheduling

General Terms Design, Performance

Keywords Multicore, Heterogeneous environment, Provisioning

1. Introduction

As multicore and future many-core processors emerge with more complex optimization features, how to achieve optimal efficiency in terms of energy, power, and performance when running parallel workloads poses a new challenge to both large-scale system administrators and networked machines used by individuals. Unlike traditional high performance computing systems which were often designed with a slew of homogeneous processing elements, a modern computing environment such as data centers or distributed nodes in a computing cloud will consist of a variety of multicore or many-core processors with different underlying microarchitectures. For example, Intel's Core i7 processor (code-named Nehalem) is significantly different from previous x86 multicore processors. In addition to its hyperthreaded cores integrated with an on-die memory controller, it also supports more advanced power management schemes. When deploying these highly tunable systems into an execution environment with older machines, such heterogeneity makes it impractical for software developers to determine the optimal runtime parameters such as the optimal number of threads to spawn, the usefulness of hyperthreading, and the optimal voltage/frequency level, at static development time. Moreover, the lifetime of software deployed often span across several architectural generations. Thus, one set of optimized runtime parameters for a given software on a particular machine does not necessarily result into the most efficient execution for another multicore processor configuration.

Furthermore, workloads based on users' input can also vary dynamically, leading to optimization scenarios that are too complex to understand even for one particular software. For example, user-created content in Web 2.0 era such as online video clips can have a wide variability, such as video length, resolution, and audio sampling rate, which is very difficult to predict at the time of software development. Last but not least, these computer users may have different computing requirements or preference. For example, business owners who host server farms may desire to optimize their system performance for a given fixed cooling capacity to maximize performance per watt. On the other hand, laptop users may want to protract the runtime of a particular software (e.g., an online game and a YouTube video decoder) for a given fixed battery power supply, which, in essence, exploits performance per joule.

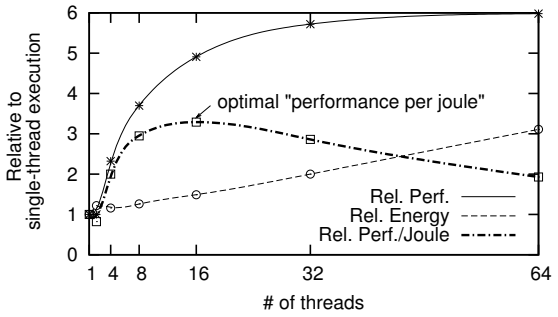

Figure 1. Relative Performance, Energy, and Performance per Joule of Bodytrack

To motivate, Figure 1 shows the trends of *performance*, *energy*, and *performance per joule* by increasing the number of threads from 1 to 64 when running bodytrack, a benchmark program from PARSEC. Even though the overall performance is improved with more active threads thus more energy consumed as expected, the non-linear performance improvement suggests an optimal "performance per joule" when the number of concurrent, active threads reaches 16. In other words, one can optimize the energy efficiency of a system by finding the ideal number of threads to execute. Unfortunately, such optimality can be difficult to predict in a complex, networked heterogeneous computing environment.

2. PROPHET Infrastructure

To tackle this problem, we propose an infrastructure called PROvisioning Processing in a Heterogeneous EnvironmenT, or PROPHET, to perform provisioning and management for future networked heterogeneous many-core platforms. In PROPHET, a group of many-core processors over the Internet cloud will share application profile data of each processing platform. A processor node that runs an application previously executed by other nodes can refer to the global history profile gathered and analyzed by a server. Based on

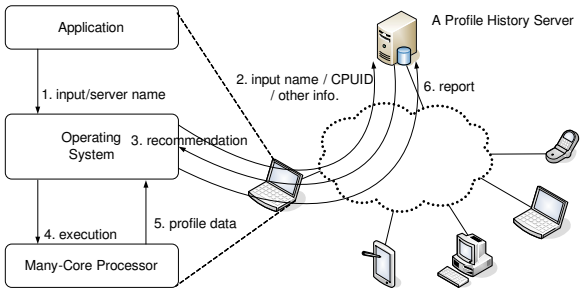

Figure 2. PROPHET Infrastructure

which, a provision can be made to perform tuning to achieve the execution objective, e.g., optimal performance per joule. For example, a user watching a video clip from YouTube.com on a particular computing machine model can obtain an optimally tuned setting by consulting input from other users who had watched the same clip on the same machine model. Another example is online 3D gameplay. Each gamer's machine can be autonomously tuned to maximize its performance per joule using settings suggested and analyzed by the central game server based on input of millions of others who had played the same scenario. Given these types of applications are already network-connected, the social cost to provide this additional service will be modest.

Figure 2 shows the PROPHET infrastructure along with layers of software/hardware stack of each individual node. To obtain optimization guidance in PROPHET, each processing node including its multicore or many-core processor and the OS running atop will communicate with a *Profile History Server* (PHS) on the cloud to gather recommended configurations such as the most energy-efficient number of threads to be spawned, whether to use hyper-threading, and such. The server maintains a history of prior runs from other nodes, and provides such recommendation to its client equipped with the same or closest configuration. This server may be provided by a content provider such as YouTube.com, by an application provider such as Adobe, by an OS vendor, or even by an individual user that runs the server in a peer-to-peer manner. This server information should be set manually by a user, by an OS vendor, or by an application provider.

Figure 2 also demonstrates a process's interaction with an OS, a processor and a PHS. Upon launching a new application (e.g., Adobe flash player to decode a YouTube clip), an OS retrieves the name of an input set (e.g., the clip's URL) and the name of a profile history server (e.g., YouTube.com). Then, the OS transfers the following information to the PHS including the name of the input set, the CPUID of the processor running the OS (or a laptop made model), and information such as the number of available cores and their CPU times, or other optimization objectives, etc. Then, the PHS responds with a recommended configuration to the requesting OS. According to the feedback from the PHS, the OS will then make a decision with respect to how to provision the system resources prior to the application's execution. On the other hand, after the process is completed, the OS can read profile data from performance counters of the processor. This data may include conventional performance counters as well as power consumption information measured by an on-chip ammeter, which is already integrated in some commercial products (McNairy and Bhatia 2005). Then, the OS reports the profile data back to the PHS, and the PHS updates its database and may perform offline analysis to benefit future requests made by similar client configurations.

3. Challenges and Directions

There are several research challenges to be addressed and an ample design space to be explored in our PROPHET infrastructure. To make PROPHET feasible, a list of basic studies regarding configuration/workload mining and system provisioning are listed as follows.

- **Configuration Mining and Recommendation**. The decision of a recommended provision can be made using a simple binary search method or a complex offline analysis. The analysis can be fulfilled by an analytical model, a machine learning algorithm, or even a microarchitecture-level simulation for popular machines and workloads.
- **Input Workload Mining**. For certain applications, each end-user may have a unique input workload such as personal pictures. Although the contents of such inputs are unique, they have identical characteristics for the PHS to make provisioning decisions. Instead of using an exact signature of input workload, the PHS may extract common key characteristics as the indices for its database.
- **Granularity of Provision**. PROPHET can supply its provision to the requesting system upon each application's launch. Or, a new provision can be re-issued whenever a new thread is spawned when running a parallelized workload.
- **Provisioning Parameters**
  - **The number of threads**: Given an objective (e.g., maximizing performance per joule), PROPHET will recommend the optimal number of threads to be spawned.
  - **Dynamic voltage, frequency scaling (DVFS)**: PROPHET can provision Per-Core DVFS (Kim *et al.* 2008) to exploit the execution slack for saving power for certain applications. However, DVFS may affect user satisfaction and require end-users' input (Shye *et al.* 2008).
  - **Cores to schedule threads**: Depending on the underlying architecture, the overall performance or energy efficiency may highly depend on the core affinity of the execution. For example, in a tiled architecture, the distance between cores can affect the overall performance. We envision PROPHET will be capable of provisioning resource allocation such as designating certain cores for executing the same group of threads to minimize communication penalty or cache pollution.
  - **Chip-level scheduling and thread partitioning**: Given the future single processing node is likely to be a heterogeneous multicore itself, e.g., an Intel multicore integrated with a Larrabee GPGPU, the PHS in PROPHET can also provide provision command for workload partitioning and execution among these heterogeneous cores. This requires more in-depth research to obtain better understandings with regard to how can PROPHET guide such complex systems to unleash their full potential.

References

W. Kim, M. Gupta, G. Wei, and D. Brooks. System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators. In *Proceedings of Int'l Symp. on High Performance Computer Architecture*, 2008.

C. McNairy and R. Bhatia. Montecito: A Dual-Core, Dual-Thread Itanium Processor. *IEEE MICRO*, pages 10–20, 2005.

A. Shye, B. Ozisikyilmaz, A. Mallik, G. Memik, P.A. Dinda, R.P. Dick, and A.N. Choudhary. Learning and Leveraging the Relationship between Architecture-Level Measurements and Individual User Satisfaction. In *Proceedings of Int'l Symp. on Computer Architecture*, 2008.