Power-Aware Parallel 3-D Finite Element Mesh Refinement Performance Modeling and Analysis on CUDA/MPI Multi-core and GPU Architectures

Da Qi Ren[1,2], Reiji Suda[1,2] and Dennis D. Giannacopulos[3]
[1]Department of Computer Science, the University of Tokyo, Tokyo, 1130033, Japan, [2]JST, CREST, Japan
[3]Department of Electrical and Computer Engineering, McGill University, Montreal, H3A 2A7, Canada

Email: dennis.giannacopoulos@mcgill.ca

Abstract —The power dissipation of parallel 3-D mesh refinement is highly dependent on the underlying algorithm and the power-consuming features of the processing elements (PEs). This contribution presents a practical methodology for modeling and analyzing the power performance of parallel 3-D FEM mesh refinement on CUDA/MPI architectures based on detailed software prototypes and power parameters in order to predict the power functionality and runtime behavior of the algorithm, optimize the program design and thus achieve the best power efficiency. We propose approaches for GPU parallelization, dynamic CPU frequency scaling and dynamic load scheduling among PEs. The performance improvements of our designs are demonstrated and the results are validated on a real multi-core and GPU cluster.

I. INTRODUCTION

The General Purpose Graphics Processing Unit (GPGPU) provides new, evolving solutions in High Performance Computing (HPC) by its massively parallel processing architecture. However, the energy usage of GPU has been continually increasing and high performance GPUs may become the largest power consumers in a multiprocessing system. A CUDA Processing Element (PE) is a hardware unit comprised of a CPU and GPU that executes streams of CUDA kernel instructions; several such PEs can be bus-connected where each PE acts as a building block. Multi-core CPUs and GPUs provide cooperative architectures in which both Single Instruction Multiple Data (SIMD) and Single Program Multiple Data (SPMD) programming models can co-exist and complement each other. MPI works as the data distributing mechanism between the GPU nodes and CUDA as the computing engine. The CUDA/MPI model is becoming an important choice in various HPC applications, however much less research has been carried out to improve the power performance with such heterogeneous parallel programming paradigms. Towards power efficient computation on CPU-GPU multi-processing computers, we investigate software methodologies to optimize the power utilization through algorithm design and programming technique.

Many algorithm level design methods have been studied based on CPU platforms. A hardware-software approach called “thrifty barrier” is introduced in [1] to save energy in parallel applications that exhibit barrier synchronization imbalance. In [2], a runtime system named “Adagio” is introduced for complex scientific applications by combining lessons learned from static energy-reducing CPU scheduling. The fact that CPU and GPU computing elements are involved inside one PE rather than only CPU does not change the nature of power optimization problems, however the power saving techniques of CPU machines have to be modified according to the requirements of GPGPU architectures. The idea of our algorithm design framework for saving HPC power by software approach is illustrated in Fig.1, and we provide brief introductions to the following modules in the figure:

A. PE Power Feature Determination

An algorithm restricts the behavior of a C/CUDA program starting from high-level code to executables that run on a set of computer components including multi-core CPU, GPU and memories. The energy approximation is the summation of the products of each component power and its execution time [3], [4]. A large-scale SIMD program drives a processor running same operations repeatedly in a streaming way. When the processor’s frequency and temperature are invariables, and the number of executions in one time unit is fixed, the power can be modeled as a constant value. For a particular SIMD task, the computation of energy consumption is only dependent on the execution time. These power-consuming features are true for each of the components in a CUDA PE. An overall power model can be built up for the entire multiprocessing platform based on them.

B. PE Computation Capability

Computation capability of a parallel processing component, i.e., CPU and GPU is determined by its micro-architecture, programming language and characteristics of the computation performed on it [3], [4].

C. Algorithm and Code Optimization Strategies

The design of an algorithm has impacts on the amount of computer resources and power consumption required for a

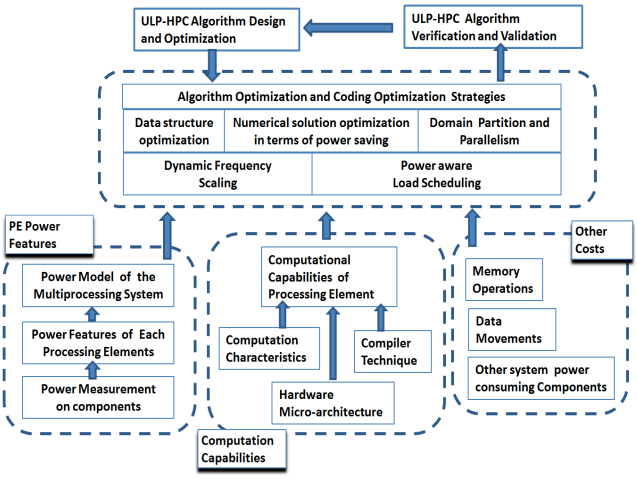

Fig.1. The framework for power-aware algorithm optimization.

given computing problem. In our power-aware design, algorithms and coding strategies are chosen based on the power model and computation capabilities of the target multiprocessing platform. Therefore, performance-tuning approaches such as domain partitioning, load parallelization, dynamic frequency scaling and workload scheduling are considered in order to reach the best overall power efficiency.

D. Verification and Validation

Performance improvement needs to be validated by comparing the results with the original design purpose, then according to the improvement satisfaction to decide the necessity of further refinement until the required power performance is reached.

Based on above framework, the methodology introduced in this paper imports hardware power parameters to software algorithm study, then estimates power consumption with CUDA/MPI program analysis. One of the advantages is that it allows obtaining design characteristic values at the early programming stage, thus benefiting programmers by providing necessary environment information for choosing the best power-efficient alternative.

II. RESULTS

We have implemented three 3-D tetrahedral mesh refinement programs to solve a resonant cavity problem with different power aware design approaches each of which will be elaborated in the long version paper: scale-down CPU frequency; GPU device parallelization with and without dynamic load-balancing. Performance and power efficiency improvement by each program have been validated through examining the measurement results on real CUDA/MPI platform when 603,979,776 tetrahedral elements are produced, as illustrated in Figs. 2-4. The CUDA PEs used in this work comprise an Intel QX9650 CPU [5] and NVIDIA 8800GTS/512 GPU. The GPU has 16 MPs (multiprocessors) and each MP has 8 SPs (streaming processors) [6].

In Fig. 2, we demonstrate the power efficiency enhancement by CPU frequency scaling approach where one CPU and one GPU have been used. When the CPU runs at 3GHz (left), the average CPU load power is 59.0W higher than that at 2GHz (right). GPU and memory power do not change when the CPU frequency scales. The computation time when the CPU frequency is 2GHz is increased by 0.16 seconds because the speed of serial part of the code that is used for building up graphical files is decreased. In total, using this approach has saved 9% of the overall energy consumption.

In addition to above CPU frequency scaling approach, GPU device parallelization has been implemented with and without load balancing functions, the measurement results are shown in Fig.3 and Fig.4, respectively. In Fig. 3, the parallel computation speedup is 5.3% (CPU on 2GHz) and 8.9% (CPU on 3GHz) comparing with single GPU program in Fig.2. The overall energy consumption is increased 39% (CPU on 2GHz) and 27% (CPU on 3GHz) because one additional GPU is involved in the computation, which brings additional power cost. In Fig. 4, the new parallel computation speedup is 52.6% (CPU on 2GHz) and 55.1% (CPU on 3GHz); and the overall

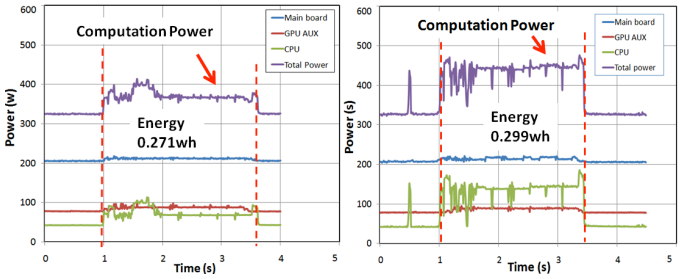

Fig.2. The resulting power charts of frequency scheduling approach on 3-D mesh refinement: (left) CPU runs on 2GHz; (right) CPU runs on 3GHz.

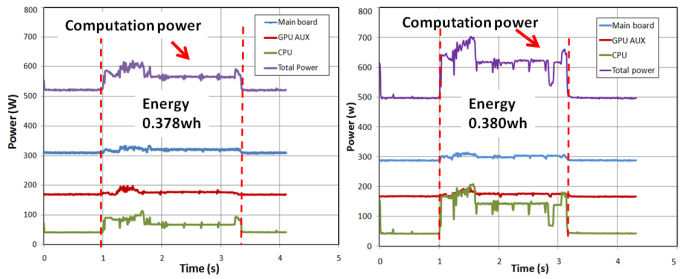

Fig.3. The resulting power charts of parallel GPU on 3-D mesh refinement: (left) CPU runs on 2GHz; (right) CPU runs on 3GHz.

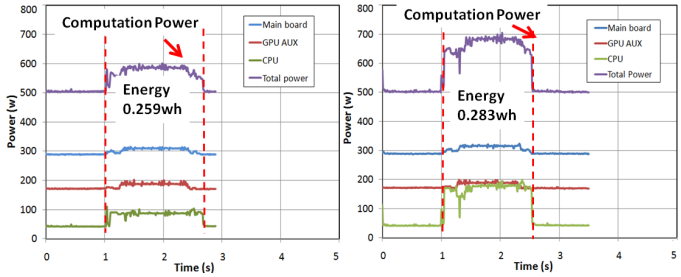

Fig.4. The resulting power charts of parallel GPU with enhanced random polling dynamic load balancing on 3-D mesh refinement: (lift) CPU runs on 2GHz; (right) CPU runs on 3GHz.

energy consumption is decreased 4.43% (CPU on 2GHz) and 5.35% (CPU on 3GHz) comparing with those of single GPU program in Fig.2. This demonstrates that the load balancing function is extremely important for GPU parallelization that will significantly enhance the computation performance and thus save the energy consumptions even when there is one additional power consuming hardware involved

Full details of the tetrahedral mesh refinement used and the power-aware algorithm design method inside one PE and among the PEs of a cluster will be provided in the long version paper. Also, the program architecture of CUDA/MPI and the power feature abstraction from the measurement of SIMD operations will be elaborated.

III. REFERENCES

[1] S. Ravi, *et al.*, *Proceedings of the 16th International Conference on VLSI Design*, pp. 431439, New Delhi, India, Jan 2003.
[2] B. Arts, *et al.*, *Proceedings of PATMOS* 03, pp.197-207, 2003.
[3] D. Q. Ren, R. Suda, *Proceedings of CSE'09*, pp. 424-429, 2009.
[4] D. Q. Ren, R. Suda, *Proceedings* of PPAM 2009, 2009.
[5] Intel, Intel® 64 and IA-32 Architectures Software Developer's Manuals, June 2010.
[6] NVIDIA, CUDA Programming Guide, Version 2.3.1, Aug 2009.