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

Da Qi Ren[1], Eric Bracken[1], Sergey Polstyanko[1], Nancy Lambert[1], Reiji Suda[2], and Dennis D. Giannacopulos[3]

[1]ANSYS Inc., Pittsburgh, PA 15219 USA
[2]Department of Computer Science, the University of Tokyo, Tokyo 1130033, Japan,
[3]Department of Electrical and Computer Engineering, McGill University, Montreal, QC H3A2A7, Canada

Software power performance tuning handles the critical design constraints of software running on hardware platforms composed of large numbers of power-hungry components. The power dissipation of a Single Program/Instruction Multiple Data (SPMD/SIMD) computation such as finite element method (FEM) mesh refinement is highly dependent on the underlying algorithm and the power-consuming features of hardware Processing Elements (PE). This contribution presents a practical methodology for modeling and analyzing the power performance of parallel 3-D FEM mesh refinement on CUDA/MPI architecture 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. In detail, we have proposed approaches for GPU parallelization, dynamic CPU frequency scaling and dynamic load scheduling among PEs. The performance improvement of our designs has been demonstrated and the results have been validated on a real multi-core and GPU cluster.

***Index Terms*—FEM mesh refinement, green computing, parallel algorithms, software engineering.**

I. Introduction

GENERAL-PURPOSE computing on graphics processing units (GPGPU) provides new, evolving solutions for High Performance Computing (HPC) on massive parallel processing architectures. However, the power usage of GPUs has been continually increasing, and they may become the largest power consumers in a modern multiprocessing system. A CUDA Processing Element (PE) is a hardware unit comprised of GPUs and CPUs that execute streams of CUDA kernel instructions; several such PEs can be bus-connected where each PE acts as a node. Multi-core processors and GPUs provide cooperative architectures in which both SIMD and SPMD programming models can co-exist and complement each other. The CUDA/MPI model is becoming an important choice in various HPC applications where MPI works as the data distributing mechanism and CUDA/GPU as the local computing engine, as shown in Fig. 1; however, much less research has been carried out to improve the power performance with integrated parallel programming paradigms. Towards power efficient HPC for electromagnetics, 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. 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 techniques for CPU machines have to be upgraded according to GPGPU architectures. A typical approach in [1] introduces an integrated power model for a GPU computer to predict execution times and calculate dynamic power events. By integrating an analytical timing model and an empirical power model, power consumption of GPGPU workloads is predicted. Higher-level models use more indirect and approximate design parameters, such as the algorithm level power model that we have introduced in [2]. The advantage of the approach is that instruction mixture information, pipelining structure and out of order processing information can be covered in the SIMD flows that are measured. In this paper we provide an algorithm design framework for saving SIMD computing energy by software approach as illustrated in Fig. 2:

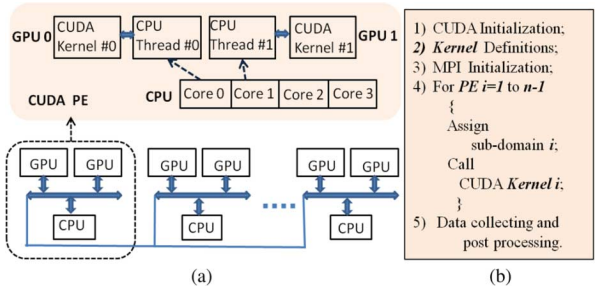

Fig. 1. CUDA/MPI Programming paradigm for parallel mesh refinement: (a) The architecture of a CUDA PE composed of Multicore and GPU components; (b) Algorithm for CUDA/MPI implementation.

1) *PE Power Feature Determination*
CUDA algorithms restrict the behaviors of low-level code executing on each component in a PE including core(s), GPU(s) and memories. A large-scale SIMD program drives a processor running the same operations repeatedly in a streaming way. When the processor's frequency and temperature are invariant, and the number of executions in one time unit is fixed, the power can be modeled as a constant value. The energy approximation is the summation of the products of each component power and its execution time [2]. An overall power model can be built up for the entire multiprocessing platform based on them.
2) *PE Computation Capability*
Computation capability of a parallel processing component, i.e., CPU and GPU is determined by

Manuscript received July 03, 2011; revised October 11, 2011; accepted November 15, 2011. Date of current version January 25, 2012. Corresponding author: D. Q. Ren (e-mail: daqi.ren@ansys.co).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMAG.2011.2177814

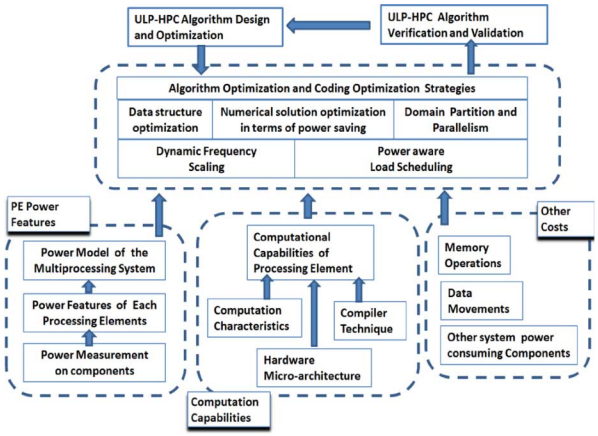

Fig. 2. The framework for power-aware algorithm optimization.

its micro-architecture, programming language and characteristics of the computation performed on it [3].

3) *Algorithm and Code Optimization Strategies*
Algorithms and coding strategies are chosen based on the power model and computation capability of each component. 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.

4) *Verification and Validation*
The performance improvement procedure usually consists of several incremental steps. The result of each step will be checked with original design objectives, then according to the improvement satisfaction to decide the necessity of further refinement until the required power performance is reached.

The methodology imports hardware power parameters to the 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. In this paper we have implemented 3D Hierarchical Tetrahedral and Octahedral (HTO) mesh refinement on a CUDA/MPI architecture. The above power-aware algorithm design method has been applied to it.

II. CUDA/MPI Hardware Architecture and Power Aware Design Approaches

The CUDA PEs used in this work include Intel QX9650 multicore and NVIDIA GeForce 8800GTS/512 GPUs. The GPU has 16 MPs (multi-processors), each MP has 8 SPs (streaming processors). GPU, multicore and main board (including memories and PCI bus on the main board) are the major power consuming components. The total power is defined in (1):

$$P_{system}(w) = \sum_{i=1}^{N} P^{i}_{CPU}(w^i) + \sum_{j=1}^{M} P^{j}_{GPU}(w^j) + P_{Mainboard} \tag{1}$$

where N and M represents the number of CPUs and GPUs involved in the computing, and w^i and w^j represent the workloads assigned to CPU_i and GPU_j, respectively [4]. The method to measure and evaluate the CUDA PE's power/energy consumptions has been introduced in our previous work in [2]. Here we provide the following methods in power aware algorithm design:

1) *Scale Down CPU Frequency*
During the execution of a CUDA kernel on GPU, its host CPU's time clocks are used for polling the running kernel and to give correct responses to the kernel calls. This requires that the CPU frequency has to be greater than or equal to the frequency of kernel calls in order not to decrease the GPU's computing speed. In general a CPU's selectable frequencies are all higher than a CUDA kernel calls' need. This makes it possible to scale down the CPU frequency without compromising the PE's performance [5]. This property will benefit designers to select a lower CPU frequency because the power will decrease accordingly. The minimum CPU frequency needs to be greater than or equal to the frequency of the PCI bus between the GPU and CPU.
The performance improvement procedure usually consists of several incremental steps. The result of each step will be checked with original design objectives, then according to the improvement satisfaction to decide the necessity of further refinement until the required power performance is reached.

2) *GPU Device Parallelization*
Using parallel GPUs is another practical way to enhance a CUDA PE's power efficiency. Multi-GPU devices sharing one CPU host will maximize the CPU's usage by multithreading, thus the CPU responds to multiple CUDA kernels on different GPU devices with different threads [6]. By multithreading, a CPU can work with two GPUs together to achieve speedup in kernel execution time. The total energy consumption will decrease compared with pairing one CPU and one GPU.

III. Power Aware Algorithm Design In 3-D HTO Mesh Refinement

In 3-D electromagnetic FEM, tetrahedral elements are commonly used to represent the geometric discretization of the problem. In HTO subdivision of a tetrahedron illustrated by Fig. 3, the refinement rule involves three steps: first, the tetrahedron is broken down into four scaled duplicate tetrahedra (one for each corner) and one octahedron (remainder) as shown in Fig. 3(a), left. Second, the resultant octahedron is then subdivided into six octahedral as given in Fig. 3(a), right [6]. The recursive application of the HTO refinement rules generates elements that belong to two congruence classes: one consisting of all generated tetrahedra, and one consisting of all generated octahedra. This refinement property is intentional, and it is useful for subsequent FEM computations [6]. HTO refinement can also be applied for creating efficient, interactive mesh hierarchies on volume visualization applications such as iso-surface extraction and direct volume rending [7]. An algorithm designed to implement HTO refinement on CUDA PEs uses only coarse meshblocks on CPU, since a coarse mesh can be efficiently updated and transmitted, to distribute the blocks to different GPU devices. According to the viewing parameters, refinement process is performed on GPUs to compute appropriate details. In each CUDA PE, the initial mesh blocks

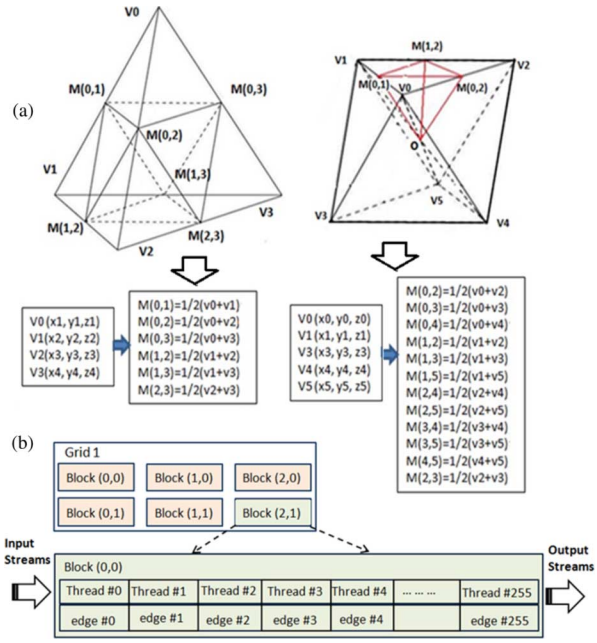

Fig. 3. Parallel HTO mesh refinement and mapping the computation: (a) Mesh refinement model: tetrahedron subdivision; primary octahedron subdivision and secondary octahedron subdivision; (b) mesh flow is executed by GPU threads.

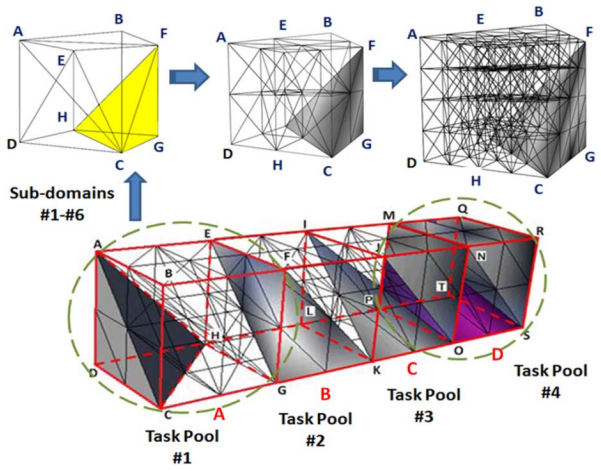

Fig. 4. Resonant cavity discretization in four smaller rectangular blocks (A-D); each of these blocks was subdivided into 6 tetrahedra. The resulting 24 tetrahedra mesh are the sub-domains to be assigned to CUDA PEs.

TABLE I
SUB-DOMAIN PARTITIONING AND TASK POOL STRUCTURE

Task Pools	Block A	Block B	Block C	Block D
Tasks	*ACDH, ABCH*	*EGHL, EFGL*	*IKLP, JFKP*	*MOPT, MFOT*
	ABEH, BCGH	*EFIL, FGKL*	*IFMP, JKOP*	*MFQT, NKST*
	BEGH, BEGF	*FIKL, FIKJ*	*JMOP, JMON*	*NQST, NQSR*

on main memory are not completely loaded into GPU memory all at one time, these geometries are restored in data flows and executed in a streaming way, as shown in Fig. 3(b).

The efficiency of power aware CUDA/MPI program design methods is investigated using the 3-D rectangular resonant cavity model illustrated in Fig. 4. The cavity was initially discretized into four smaller rectangular blocks (A-D); each of these blocks was subdivided into 6 tetrahedra. The resulting 24 tetrahedra mesh contains the sub-domains to be assigned to CUDAPEs in the parallel system. MPI is a platform-independent library that is used to manage inter-node communications. After receiving the mesh assignments, each CUDA PE operates independently. Based on the geometry properties of the initial problem and the number of GPUs devices, we define four task pools using stack structures. Each pool contains a 6 tetrahedra mesh located inside one smaller rectangular block (A-D), as shown in Table I. We design a CUDA kernel to compute the coordinates of new generation element vertices that include 6 add/divide operations for each tetrahedron and 12 add/divide operations for each octahedron, as shown in Fig. 4. Shared edges and faces between any two adjacent tetrahedral sub-domains are refined by the host CPU in order to save the inter CPU communication also decrease the amount of data transmission between parallel GPUs. The task pools can be used exclusively, race conditions are avoided and the possibility of bottlenecks when the number of CUDA PEs increases is decreased.

We apply a centralized dynamic load-balancing algorithm for workload scheduling because the initial mesh domains contain a small size of data. Note that even though the number of task assignments is same to each PE, the workload of each mesh may take a different amount of time because of the different viewing parameters. We design our algorithm to: locate all task pools on one master PE; when a slave PE finishes the tasks inits own task pool, it will poll the master PE to ask for new tasks. The master PE will then check the remaining tasks from other non-empty task pools in order to schedule new tasks for the polling PE. Note that the original random pooling dynamic balancing method specifies to split half of the remaining tasks for the polling PE. We formulate in this work that a target non-empty task pool transfers only one task at a time to the polling PE [6]. In the above 3-D FEM problem the total number of sub-domains is limited but the mesh refinement process in each sub-domain takes longer to complete.

Performance per watt is used in this work as the measure of energy efficiency for CUDA PEs. HTO mesh refinement is a SIMD program that performs the same operations repeatedly in a streaming way. The power feature can be described in f $(Mflops/watt)$ that measures the rate of computation that can be delivered by the PE for every watt of power consumed. When a multiprocessing system contains different PEs, i.e., $f_i \neq f_j$ (for PE $i, j \in n$) power aware algorithm design needs to employ a load scheduling protocol to achieve the best power performance. Assuming a work load W_i consumes power P_i $(watt)$ on PE_i whose power feature is f_i $(Mflops/watt)$, the W_i needs to satisfy with the following conditions in order to achieve the optimized energy consumption: Total energy

$$E = E_0 + E_1 + \ldots\ldots + E_{n-1} = \frac{W_0}{f_0} + \frac{W_1}{f_1} + \ldots + \frac{W_{n-1}}{f_{n-1}} \quad (2)$$

and W_i satisfies the minima of E:

$$\begin{cases} E'_{W0} = 0,\ E'_{W1} = 0, \ldots E'_{Wn-1} = 0 \\ W_0 + W_1 + \ldots\ldots W_{n-1} = W \end{cases} \quad (3)$$

IV. RESULTS

Three power aware algorithm designs introduced in Section II have been implemented in three different HTO mesh refinement implementations in solving the same resonant cavity problem.

TABLE II
MEASUREMENT RESULTS OF THREE POWER AWARE HTO MESH REFINEMENT PROGRAMS

	Single GPU		Multiple GPUs			
			Non-Load Balancing		Load Balancing	
CPU Freq.	2GHz	3GHz	2GHz	3GHz	2GHz	3GHz
Time(s)	2.59	2.43	2.31	2.18	1.66	1.53
Energy(wh)	0.271	0.299	0.378	0.380	0.259	0.283
Speedup	1	1.07	1.13	1.20	1.57	1.71
Energy Saving	0	-10.3%	-39.5%	-40.2%	4.4%	-4.4%

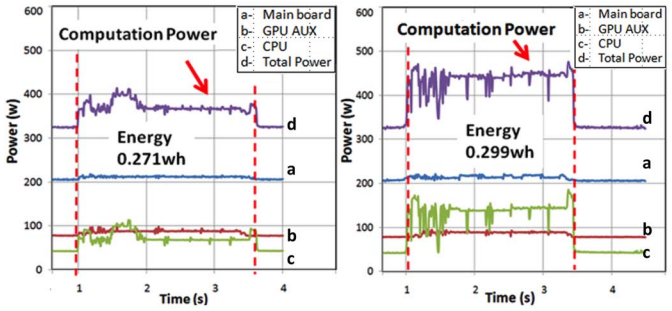

Fig. 5. The result power charts of frequency scheduling approach on 3-D HTO mesh refinement: (left) CPU runs at 2 GHz; (right) CPU runs at 3 GHz.

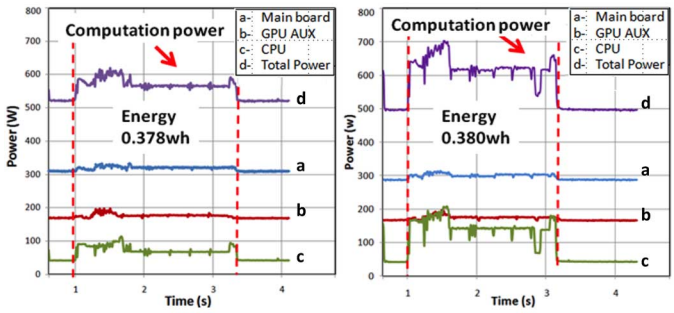

Fig. 6 The result power charts of parallel GPU on 3-D HTO mesh refinement: (left) CPU runs at 2 GHz; (right) CPU runs at 3 GHz.

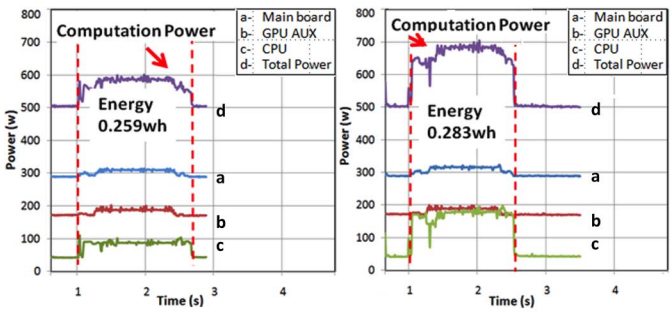

Fig. 7. The result power charts of parallel GPU with enhanced random polling dynamic load balancing on 3-D HTO mesh refinement: (left) CPU runs at 2 GHz; (right) CPU runs at 3 GHz.

The performance and energy efficiency improvement by each program has been validated through examining the measurement results on real CUDA/MPI platform when 603 million HTO tetrahedral elements are produced, as illustrated in Table II and Figs. 5–7.

In Fig. 5, one CPU and one GPU are used to demonstrate the difference in overall energy efficiency when the CPU frequency increases from 2 GHz (left) to 3 GHz (right). A computation speedup of 1.07 is obtained because as the frequency increase, the speed of the serial part of the program that is used for building up graphical files becomes faster. However because the CPU power increases with its frequency too, the overall energy consumption is increased 10.3%, as shown in Fig. 5.

In addition to the above CPU frequency scaling approach, parallel algorithms on multiple GPUs have been implemented with and without load balancing functions; their results are shown in Fig. 6 and Fig. 7, respectively. In Fig. 6 each GPU device works on its initial sub-domain individually without load balancing. The parallel computation speedup is 1.13 (CPU at 2 GHz) and 1.20 (CPU at 3 GHz) compared with the corresponding single GPU programs in Fig. 5 (left) and (right). The overall energy consumption is increased 39.5% (CPU at 2 GHz) and 40.2% (CPU at 3 GHz) because one additional GPU is involved in the computation, which brings additional power cost. In Fig. 7 an enhanced dynamic load balancing function has been implemented. The new parallel computation speedup is 1.57 (CPU at 2 GHz) and 1.71 (CPU at 3GHz, best speedup); and the overall energy consumption improvement is 4.4% (CPU at 2 GHz, best energy efficiency) and −4.4% (CPU at 3 GHz) compared with the corresponding value of single GPU program in Fig. 5 (left) and (right). This demonstrates that the load balancing function is extremely important for GPU parallelization and will significantly enhance the computation performance and thus reduce the energy consumption even when there is one additional power-consuming device involved.

V. CONCLUSION

A power-aware CUDA/MPI algorithm design framework has been introduced. Utilizing power parameters captured from the real system, modeling and evaluation for the power features of each PE in the target multi-core and GPU platform is formulated. Based on it, we have introduced GPU parallelization, CPU frequency scaling and power aware load scheduling methods to optimize the overall power consumption through tuning the number and usage of the CPU and GPU components. The power efficiency improvement of the designs has been validated by measuring the implemented program running on real systems.

REFERENCES

[1] S. Hong and H. Kim, "An integrated GPU power and performance model," presented at the Int. Symp. Computer Architecture, Saint-Malo, 2010.
[2] D. Q. Ren and R. Suda, "Power aware SIMD algorithm design on GPU and multicore architectures," in *Handbook of Energy-Aware and Green Computing*. London, U.K.: Chapman and Hall/CRC Press, 2012, 10: 1439850402.
[3] NVIDIA, CUDA Programming Guide ver. 2.3.1, 2009.
[4] J. M. Rabaey, *Digital Integrated Circuits*. Englewood Cliffs, NJ: Prentice Hall, 1996.
[5] Intel, Intel 64 and IA-32 Architectures Software Developer's Manuals, 2010.
[6] D. Q. Ren and D. D. Giannacopoulos, "Parallel mesh refinement for 3D finite element electromagnetics with tetrahedra: Strategies for optimizing system communication," *IEEE Trans. Magn.*, vol. 42, no. 4, pp. 1251–1254, 2006.
[7] G. Greiner and R. Grosso, "Hierarchical tetrahedral-octahedral subdivision for volume visualization," *Vis. Comput.*, vol. 16, pp. 357–369, Oct. 2000.