On Energy Nonproportionality of CPUs and GPUs

Ravi Reddy Manumachu and Alexey Lastovetsky
School of Computer Science
University College Dublin
Dublin, Ireland
Email: {ravi.manumachu,alexey.lastovetsky}@ucd.ie

2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) | 978-1-6654-9747-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/IPDPSW55747.2022.00015

***Abstract*—Energy proportionality (EP) means designing a system that consumes energy proportional to the amount of work it performs. For an EP system, optimizing an application for performance also optimizes the application for total energy. Energy-proportional multicore CPUs and graphics processing units (GPUs) are fundamental to addressing the grand technological challenge of energy efficiency in Information and Communications Technology. In this work, we formally propose strong and weak notions of EP for modern microprocessors.**

Multicore CPUs were experimentally found to violate both strong and weak EP. This work presents the first attempt at a theoretical analysis to explain the behaviour. GPUs are carefully designed with on-chip resources primarily dedicated to achieving high arithmetic throughput rather than caching and flow control. Consequently, the mainstream view is that GPUs exhibit strong and weak EP. However, GPUs were experimentally found to violate strong EP. In this work, we experimentally study the weak EP of an Nvidia K40c GPU and an Nvidia P100 PCIe GPU using a specially designed matrix multiplication application. We show that both the GPUs also breach weak EP, which presents an opportunity for bi-objective optimization of the application for dynamic energy and performance. By analyzing the Pareto fronts of dynamic energy and performance for a wide range of workloads, the maximum dynamic energy savings are up to 18% while tolerating a performance degradation of 7% for Nvidia K40c GPU and (50%,11%) respectively, for Nvidia P100 PCIe GPU.

***Index Terms*—Energy Proportionality, Multicore CPU, GPU, Bi-objective Optimization, Energy, Performance, 2D FFT, Matrix Multiplication**

I. Introduction

Energy efficiency in Information and Communications Technology (ICT) is now a grand technological challenge and the top design constraint in all computing settings (mobile, desktop, server, supercomputer, and data centre) [1],[2]. Energy-proportional computing is considered fundamental to addressing this challenge. The high-level definition of energy proportionality (EP) is to design a system (such as a microprocessor) that consumes energy proportional to the amount of work it performs.

Chandrakasan et al. [3] is an influential work in proposing energy efficiency in digital system design and paving the path towards energy-proportional computing. Barroso and Hölzle [4] propose that designing energy-proportional servers should be the primary design goal of component and system designers since a server's energy consumption has an enormous direct impact on the data centre's infrastructure cost. Since this proposal, EP has been analyzed extensively in server CPUs [5], [6], [7], [8], networks [9], and storage [10], [11].

We now illustrate strong and weak notions of energy proportionality. Before we do this, relevant terminology for power and energy consumption in computing follows. There are two types of power consumption in a component executing an application: dynamic power (P_d) and static or idle power (P_s). Dynamic power consumption happens due to the switching activity in the component's circuits. Static power consumption happens when the component is not active or doing work. The total energy consumption (E_T) during an application execution is the sum of dynamic and static energy consumptions. The static energy consumption (E_s) is the idle power of the platform (without application execution) multiplied by the application's execution time (t). The dynamic energy consumption (E_d) is the total energy consumed by the platform during the application execution minus the static energy consumption.

A. Strong Energy Proportionality

Mathematically speaking, a strong interpretation of EP based on its high-level definition signifies that $E_d = c \times W$ for an EP system where c is a constant and W is the work performed. Therefore, it implies that dynamic energy consumption increases linearly with work.

However, Khaleghzadeh et al. [12] establish through extensive experiments using two data-parallel applications on a modern Intel multicore CPU and two Nvidia GPUs that strong EP is breached significantly for these processors. We summarize the experimental setup and the findings here. The processors are a dual-socket Intel Haswell multicore CPU containing 24 physical cores with 64 GB main memory, an Nvidia K40c GPU and an Nvidia P100 PCIe GPU (specifications shown in Table I). The 2D-FFT application employed in the experiments computes 2D discrete Fourier Transform of a complex signal matrix of size $N \times N$. It employs Intel MKL FFT routines for the CPU and CUFFT routines for Nvidia GPUs. The application on multicore CPUs is a multithreaded parallel application that divides the workload equally between the threads and cores. There are no communications involved

This publication has emanated from research conducted with the financial support of Sustainable Energy Authority of Ireland (SEAI) under Grant Number 21/RDD/664. This publication has emanated from research supported in part by a research grant from Science Foundation Ireland and the Sustainable Energy Authority of Ireland under the SFI Frontiers for the Future Programme 20/FFP-P/8683.

978-1-6654-9747-3/22/$31.00 ©2022 IEEE
DOI 10.1109/IPDPSW55747.2022.00015

TABLE I: Specifications of the Intel Haswell multicore CPU, a Nvidia K40c, and a Nvidia P100 PCI-E GPU.

Intel Haswell E5-2670V3	
No. of cores per socket	12
Socket(s)	2
CPU MHz	1200.402
L1d cache, L1i cache	32 KB, 32 KB
L2 cache, L3 cache	256 KB, 30720 KB
Total main memory	64 GB DDR4
(Intel MKL, OpenBLAS) versions	(2020.0.4, 0.2.19)
NVIDIA K40c	
No. of CUDA cores (Base clock)	2880 (745 MHz)
Total board memory	12 GB GDDR5 SDRAM
L2 cache size	1536 KB
Thermal design power (TDP)	235 W
(CUDA, nvcc) versions	(7.5, 7.5.17)
NVIDIA P100 PCIe	
No. of CUDA cores (Base clock)	3584 (1328 MHz)
Total board memory	12 GB CoWoS HBM2
L2 cache size	4096 KB
Thermal design power (TDP)	250 W
(CUDA, nvcc) versions	(10.1, 10.1.243)

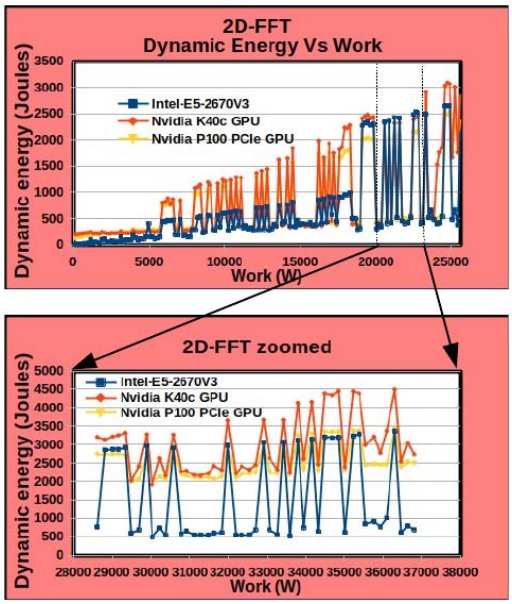

Fig. 1: The dynamic energy consumption (E_d) versus the work (W) for the FFT application computing 2D discrete Fourier Transform of a dense complex signal matrix of size $N \times N$.

between the threads. The amount of work (W) performed is $5.0 \times N^2 \times \log_2(N)$. N ranges from 125 to 44000.

Figure 1 shows the dynamic energy consumption (E_d)versus the work (W). *For each data point reported in this work, the application is run repeatedly until the sample mean lies in the 95% confidence interval, and a precision of 0.025 (2.5%) is achieved. For this purpose, Student's t-test is used assuming that the individual observations are independent and their population follows the normal distribution. The validity of these assumptions is verified using Pearson's chi-squared test.* The dynamic energy consumption is determined using the system-level power measurements provided by the WattsUp Pro power meter, which is the most accurate mainstream method [13].

The graph shows that for all three processors, the dynamic energy is a complex non-linear function of work performed, and therefore strong EP does not hold for them. The research work [12], however, does not provide a theoretical analysis of the experimentally observed energy nonproportionality.

B. Weak Energy Proportionality

We look at the simple EP model specified and investigated in [4], [14], [15], [5], [6] to define weak energy proportionality. Research works [4], [5] specify that an ideal EP system would consume dynamic power proportional to its utilization level. Fan et al. [14] show that the dynamic power is nearly linear against CPU utilization for a dual-core Intel Xeon processor. Rivoire et al. [15] demonstrate that the dynamic power is linear for CPU utilization up to 500% (5 cores) and then levels off for an 8-core Intel Xeon processor. Ryckbosch et al. [5] analyze dynamic power versus CPU utilization graphs for the SPECpower_ssj2008 benchmark for about 210 servers from 20 vendors. They find that some servers exhibit a linear relationship.

Mathematically speaking, the simple EP model states that the dynamic power consumption (P_d) is linearly proportional to utilization (U), $P_d = a \times U$, for an EP system. The execution time of an application (t) by definition is inversely proportional to utilization, $t = \frac{b}{U}$. The dynamic energy and total energy consumptions are then equal to the following:

$$E_d = P_d \times t = a \times U \times \frac{b}{U} = a \times b = k$$
$$E_T = E_d + E_s = k + P_s \times t$$

Therefore, optimizing an application for *performance* will optimize the application for *total energy* since k and P_s are constants. We define *weak energy proportionality* based on the practical implication of the simple EP model, which is that the dynamic energy consumption is a constant for all the application configurations solving the same workload.

To isolate the impact of the system architecture on energy proportionality from the impact of the application software, the design of the application testing the energy proportionality of the system should follow some constraints. The application must be a load-balanced multithreaded parallel application where all the application configurations run one thread per core and distribute the workload equally between threads. Ideally, there should be no communications or synchronization between the threads. In this case, the variations in utilizations of cores for different configurations can be attributed entirely to the complexity of the system architecture (mainly due to contention for shared resources) rather than to the parallel algorithm or unequal distribution of workload.

To summarize, we define *strong EP* to mean that dynamic energy consumption increases linearly with work, whereas *weak EP* to imply that dynamic energy consumption is a constant for all the application configurations solving the same workload (or performing an equal amount of work), given that different configurations equally distribute the workload

between the parallel threads resulting in equal utilization of identical abstract processors running the threads.

Khokhriakhov et al. [8] discover that weak EP does not hold for modern multicore processors. They study weak EP of four modern multicore processors by carefully designing four highly optimized multithreaded data-parallel applications and by analyzing the functional relationship between dynamic energy and execution time for different application configurations solving the same workload. They show that optimizing for performance alone may significantly increase dynamic energy consumption and optimizing for dynamic energy alone – in considerable performance degradation. They propose a qualitative dynamic energy model that demonstrates that energy nonproportionality is due to data translation lookaside buffer's disproportionately energy expensive activity (dTLB). However, the authors [8] do not provide a theoretical analysis of the experimentally observed energy nonproportionality of their multicore CPU platforms.

In this paper, we bridge the gap with the first attempt at a theoretical analysis of the energy nonproportionality of multicore CPUs based on CPU utilization. We first show how the EP picture has changed from a single-core era when systems obeyed the ideal EP model to the complex multicore era where non-functional EP behaviour is observed. We then consider the simplest case of two homogeneous cores executing different application configurations solving the same workload. We show that dynamic energy increases in all situations when there are differences in utilizations of the cores, thereby contravening the simple EP model.

GPUs are designed carefully with on-chip resources (for example, the large number of small CUDA cores) primarily dedicated to data computations rather than caching and flow control to provide high arithmetic throughput. Therefore, a GPU is inherently less heterogeneous than a multicore CPU with less complex dynamic power management, and consequently, one would expect that GPUs exhibit strong and weak EP. However, the Figure 1 shows that GPUs breach strong EP [12].

In this work, we experimentally study the weak EP of the two Nvidia GPUs shown in Table I. To elucidate the weak EP of a GPU, we specially design the matrix multiplication application (detailed in Section IV). It computes a given number of matrix products of two dense square matrices of size, N=18432, on an Nvidia P100 PCIe GPU (Table I). The selected matrix size is among the several workloads exhibiting energy nonproportionality and is used only for illustrative purposes. There are three decision variables employed in the application; the per-block shared memory dimension employed during one matrix product call, BS, the size of a group of device matrix product codes repeated textually one after the other, G, and the number of runs of a group, R.

Figure 2 illustrates the energy nonproportionality of Nvidia P100 PCIe. The data points in the graph represent different configurations (BS, G, R) of the application solving the same problem (N). The dynamic energy and execution time are measured only for the CUDA kernel invocations. The dynamic energy consumption is determined using the system-level power measurements provided by the WattsUp Pro power meter.

The figure shows two distinct regions. The top right plot shows a region of energy nonproportionality where dynamic energy increases monotonically with the execution time. This region contains application configurations employing per-block shared memory size, BS, ranging from 1 to 20. So, optimizing the application for performance in this region optimizes it for dynamic energy. The bottom left plot shows the region of energy nonproportionality containing data points for application configurations employing per-block shared memory dimension, BS, ranging from 21 to 32. There is an opportunity here for bi-objective optimization of the application for dynamic energy and performance. The Pareto front resulting from the dynamic energy and performance trade-off analysis contains two points. A 2.5% performance degradation (from the performance-optimal solution) gives 12.5% dynamic energy savings. If we consider the region of nonproportionality for application configurations with BS less than or equal to 30, one can obtain 24% dynamic energy savings while allowing a performance degradation of only 8%.

To summarize, this work presents the first attempt at a theoretical analysis of the violation of weak EP of multicore CPUs. We then experimentally study the weak EP of an Nvidia K40c GPU and an Nvidia P100 PCIe GPU using a matrix multiplication application specially designed for energy proportionality analysis. We show that both the GPUs violate weak EP, which presents an opportunity for bi-objective optimization of the application for dynamic energy and performance. For a given workload, we determine the Pareto front using the dynamic energies and execution times determined for all the application configurations solving the workload. Based on a wide range of workloads, the observed average and the maximum points in local Pareto fronts are 4 and 5 for the Nvidia K40c GPU. For this GPU, the global Pareto front consists of only one point, signifying that the optimal solution for performance is optimal for dynamic energy. For the Nvidia P100 PCIe GPU, the observed average and the maximum number of points in the global Pareto front are 2 and 3. The maximum dynamic energy savings can be up to 18% while tolerating a performance degradation of 7% for Nvidia K40c GPU and (50%,11%) respectively, for Nvidia P100 PCIe GPU.

In summary, the main contributions of this work are:

- Formalization of strong and weak notions of energy proportionality of modern microprocessors;
- The first attempt at a theoretical analysis of the violation of the weak energy proportionality of multicore CPUs based on CPU utilization;
- The first experimental study demonstrating the violation of the weak energy proportionality of two Nvidia GPUs, an Nvidia K40c GPU and an Nvidia P100 PCIe GPU.

We organize the rest of this paper as follows. We start with survey of related works in section II. We present the theoretical analysis of energy nonproportionality of multicore CPUs in III. We describe the matrix multiplication application used for

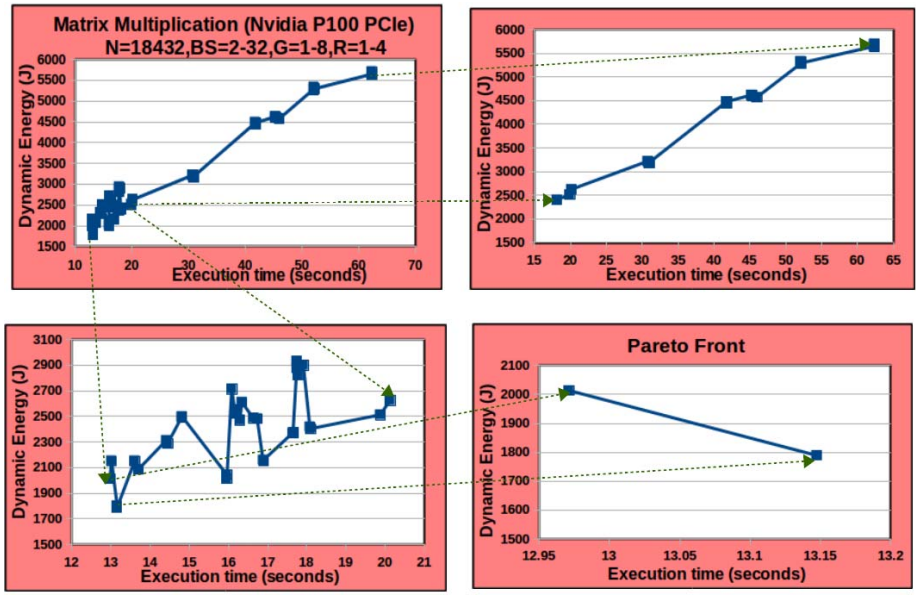

Fig. 2: EP plots for Nvidia P100 PCIe GPU executing different configurations of the matrix multiplication application that multiplies two dense square matrices of size, N=18432. Each data point in the graph represents an application configuration (BS, G, R) solving the same workload (N). The top left plot graphs the dynamic energies and execution times for all the application configurations. The top right plot shows the region of energy nonproportionality where optimizing for performance alone optimizes the application for dynamic energy. The bottom left plot zooms in to the region of energy nonproportionality where bi-objective optimization for dynamic energy and performance results in the global Pareto front (shown in the bottom right plot). The Pareto fronts displayed in this work are discrete and contain the solid square points. The points are connected by lines for visualization purposes only.

analysis of weak EP for Nvidia GPUs in IV. Then, we present our experimental results and discussion in section V. Finally, we conclude the paper in section VI.

II. Related Work

We start this section with a review of notable research works in bi-objective optimization for energy and performance followed by research works on energy proportionality (EP) of the CPU, network, and storage devices.

A. Bi-Objective Optimization on High Performance Computing Platforms

There are two principal categories of solution methods for optimizing high-performance computing (HPC) platforms for performance and energy applications. The first category of system-level solution methods aims to optimize the performance and energy of the executing environment of the applications. The dominant decision variable in this category is Dynamic Voltage and Frequency Scaling (DVFS). DVFS reduces the dynamic power consumed by a processor by throttling its clock frequency. The methods proposed in [16],[17],[18] optimize for performance under a energy budget or optimize for energy under an execution time constraint. The methods proposed in [19],[20],[21] solve bi-objective optimization for performance and energy with no time constraint or energy budget.

The second category of application-level solution methods [22],[23],[24],[25],[26],[12] use application-level decision variables and models. The most popular decision variables include the loop tile size, workload distribution, number of processors, and number of threads. Reddy et al. [25], [26] study bi-objective optimization of data-parallel applications for performance and energy on homogeneous clusters multicore CPUs employing only one decision variable, the workload distribution. They propose an efficient solution method. The method accepts the number of available processors, the discrete function of the processor's energy consumption against the workload size, and the processor's performance against the workload size. It outputs a Pareto-optimal set of workload distributions. Khaleghzadeh et al. [12] propose exact solution methods solving bi-objective optimization problem for hybrid data-parallel applications on heterogeneous computing platforms for performance and energy.

Tarplee et al. [27] consider optimizing two conflicting objectives, the make-span and total energy consumption of all nodes in an HPC platform. The decision variable is task mapping. Aba et al. [28] present an approximation algorithm to minimize both make-span and the total energy consumption of parallel applications running on heterogeneous platforms. The decision variable is task scheduling.

B. Energy Proportionality

Guerra et al. [10] study the potential energy savings by an energy-proportional storage architecture where energy usage is proportional to the utilization of storage units. They study the impact of storage energy-saving techniques (intelligent data placement, disk low energy modes, compression) on the EP of the storage. Harder et al. [11] study the EP of disk-based buffer algorithms and find no noticeable power variation

when the system state changes from idle to full utilization. They conclude that the processor component is not energy-proportional during the execution of the database workload. Abts et al. [9] propose ways to design a data center network whose power consumption is proportional to the volume of data traffic flowing through it.

Ryckbosch et al. [5] propose a metric to quantify the EP of a server's CPU. They define it as one minus the area between the CPU's actual power consumption curve and the ideal power consumption curve divided by the area under the ideal curve. Wong and Annavaram [6] propose metrics that they believe accurately quantifies EP. They show using the metrics that EP improvements are not uniform across various server utilization levels. Lo et al. [29] present a server power management solution that adjusts power in a fine-grained manner based on server request latency statistics to achieve the EP objective. Subramaniam et al. [7] investigate the possibility to achieve EP for enterprise-class server workloads by power management using Intel Running Average Power Limit interfaces (RAPL). Hsu and Poole [30] examine a range of metrics for quantifying EP. Sen and Wood [31] extend the original definition ([4]) for reconfigurable processors.

All the works reviewed above study EP based on the functional relationship between power (or energy) consumption of a system and its utilization.

Khokhriakhov et al. [8] demonstrate that the practical implication of EP does not hold for modern multicore processors using a novel application-level bi-objective optimization method for energy and performance on a single multicore processor. The method uses two decision variables, the number of identical multithreaded kernels (threadgroups) and the number of threads in each threadgroup. A given workload is partitioned equally between the threadgroups.

In this work, we present a theoretical analysis of the energy nonproportionality of multicore CPUs. We experimentally analyze the EP of an Nvidia K40c GPU and an Nvidia P100 PCIe GPU by analyzing the functional relationship between dynamic energy and execution time of a matrix multiplication application.

III. Theoretical Analysis of Energy Nonproportionality of Multicore CPUs

This section presents our first contribution: the first simple theoretical analysis of the energy nonproportionality (violation of weak EP) that was experimentally observed for multicore CPUs [8].

We start with an experimental analysis of the relationship between performance and average CPU utilization and dynamic energy and average CPU utilization for a modern multicore CPU processor. For the multicore CPU processor, the average CPU utilization is the average of the utilizations of the individual cores. The analysis aims to understand how the EP picture has changed from a single-core era when systems obeyed the ideal EP model to the complex multicore era and to allow us to explain the observed behaviour theoretically.

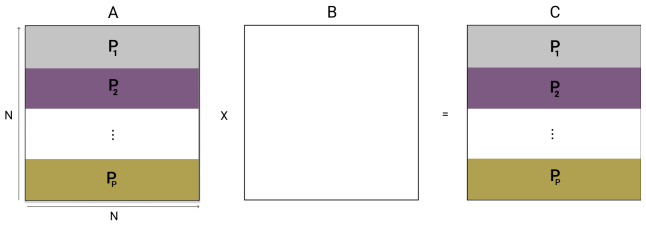

Fig. 3: Decomposition of the dense matrices, A, B, and C, in the parallel matrix multiplication application. The application is executed using p threadgroups where each group contains an equal number of threads. Matrices A and C are horizontally partitioned equally among the threadgroups. Matrix B is shared among the threadgroups. Each thread is bound to a separate core. There are no communications involved between the threads. Therefore, the workload is equally distributed between the threads and hence cores.

For the experiments, we employ a dual-socket Haswell processor with hyperthreading enabled (Table I) and two parallel matrix multiplication applications based on Intel MKL and OpenBLAS DGEMM routines. The multithreaded parallel matrix multiplication application is carefully designed to follow the guidelines set by the definition of weak EP. It is a load-balanced application that distributes the workload equally between the threads and the cores. In addition, there are no communications or synchronization between the threads.

For the theoretical analysis, we inspect how the dynamic energy consumption varies for the simplest case of two homogeneous cores executing different application configurations that solve the same workload.

A. Parallel Matrix Multiplication Application

The parallel matrix multiplication application computes the matrix product of two dense square matrices A and B of size $N \times N$. The application is executed using p threadgroups, $\{P_1, ..., P_p\}$. The matrices A and C are partitioned horizontally such that each threadgroup is assigned $\frac{N}{p}$ of the rows of B and C as shown in the Figure 3. Matrix B is shared among the threadgroups. Each threadgroup has the same number of threads, t. Each thread is bound to a separate core. There are no communications involved between the threads. Therefore, each thread solves an equal amount of workload $(N/(p \times t))$. Each threadgroup P_i computes its horizontal partition C_{P_i} using the matrix product, $C_{P_i} = \alpha \times A_{P_i} \times B + \beta \times C_{P_i}$. The application's performance is calculated as $(2.0 \times N^3)/t$, where t is the application's execution time. The application is described in detail in [8].

B. Analysis of Energy Nonproportionality

The Figure 4 shows the dynamic power consumption versus the average CPU utilization and the performance (GFLOPs) versus the utilization for the matrix size $N = 17408$. The data points are obtained by employing a strict statistical methodology. They represent different application configurations (type of partitioning, number of thread groups, number of threads per group). The behaviour is observed for a wide range of matrix sizes. The selected matrix size is a representative example. The dynamic power consumption equals the dynamic

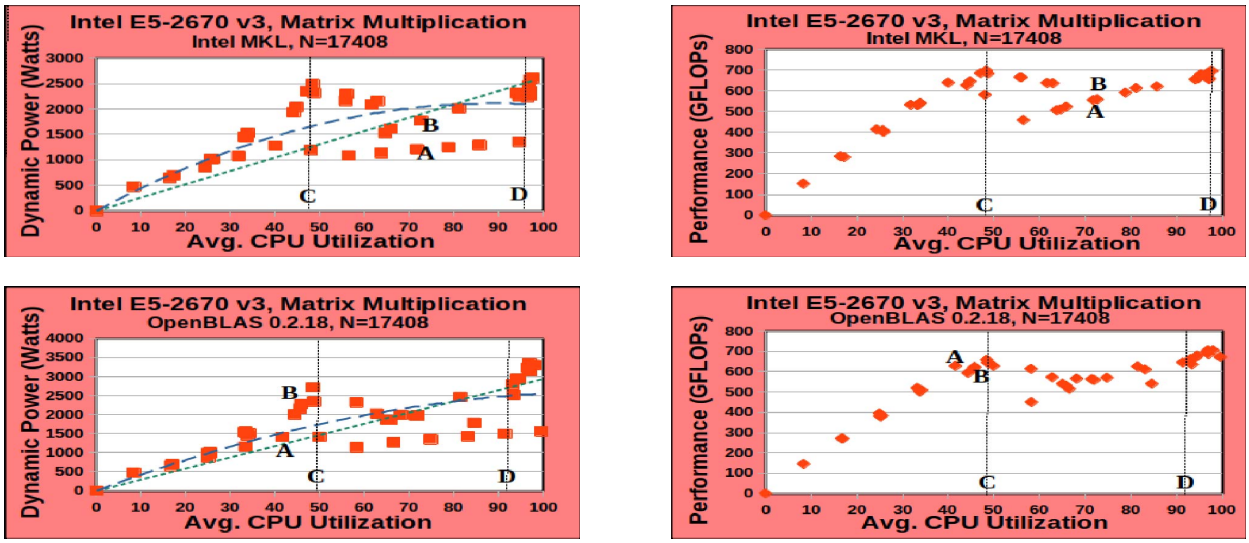

Fig. 4: Dynamic power versus average CPU Utilization and performance versus average CPU Utilization for the Intel MKL and OpenBLAS DGEMM applications multiplying two dense square matrices of size $N \times N$ on an Intel E5-2670 V3 dual-socket processor with hyperthreading where N=17408. The average CPU utilization is the average of the utilizations of the 48 logical cores. The data points in the graph represent different application configurations (type of matrix partitioning, number of thread groups, number of threads per group) solving the same matrix size. The green and blue lines show the linear and polynomial concave trend lines reported in the literature. Points A and B represent the case when there is a small change in utilization of one or more cores. Points on lines C and D have the same average CPU utilization.

energy consumption divided by the application's execution time.

The average CPU utilization is obtained using the /proc/stat interface. It is the average of the utilizations of the individual cores. The first "cpu" line in /proc/stat file aggregates the numbers in all of the other "cpuN" lines, one line per core. Since the multicore CPU processor has 48 logical cores, there are 49 lines in total. The numbers in each line identify the amount of time the CPU has spent performing different kinds of work.

The performance exhibits a typical relationship with the average CPU utilization. It is linear until the peak performance of 700 GLOPs before plateauing, suggesting that utilizing the CPU further does not yield better performance. The flattening of the performance is due to the memory activity of the threads hitting the peak memory bandwidth of the system.

However, the dynamic power consumption starts as a linear function for low utilization before exhibiting a nonfunctional relationship. For example, points with about 50% utilization have different dynamic powers and performances in the graphs for Intel MKL application (Figure 4). Similarly for points close to 65% utilization in the graphs for OpenBLAS application. This is abnormal behavior differing from the observed linearity following the simple EP model ([14], [15], [5]) or the deviations from the linearity (but still following a functional relationship) observed in [6], [30]. The green line and blue line show the linear and polynomial concave trend lines that were reported in ([14], [15], [5], [6], [30]).

However, the experimentally observed differences in utilizations of individual cores (even for the same average CPU utilization) for different application configurations executing the same workload are normal rather than abnormal for modern multicore CPUs due to the inherent complexity of the hardware architecture.

Therefore, while the simple EP model is accurate for one microprocessor component, it does not accurately capture the complex inner workings of a multicore CPU. A modern multicore processor is complex with independently powered heterogeneous components, expensive and inexpensive in terms of power consumption, and exhibits different functional utilizations due to inherent energy-efficient hardware techniques such as clock and power gating, dynamic voltage and frequency scaling, and dynamic power management.

To illustrate this complexity, we will consider the simplest case where a microprocessor comprises two homogeneous cores, C_1 and C_2, connected by a single power supply and that follow the simple EP dynamic power model, $P_1 = a \times U$ and $P_2 = a \times U$ [4], [14], [15], [5]. For example, the cores on the same socket executing a multithreaded parallel application where threads are bound to separate cores and do not interact with each other. For the execution time, we will use a simple model, $t = \frac{b}{U}$. The constants a and b will be the same for all the application configurations solving the same workload.

Consider the dynamic energy consumption of the application configuration that utilizes both C_1 and C_2 equally. The average utilization is U.

$$\begin{aligned} E_{d1,1} = E_{d2,1} &= a \times U \times \max(\frac{b}{U}, \frac{b}{U}) = a \times b \\ E_1 &= E_{d1,1} + E_{d2,1} = 2 \times a \times b \end{aligned} \quad (1)$$

where $E_{d1,1}$ and $E_{d2,1}$ are the dynamic power consumptions of C_1 and C_2.

Consider an application configuration that increases only the utilization of C_1 by ΔU, then

$$\begin{aligned} E_{d1,2} &= a \times (U + \Delta U) \times \max(\frac{b}{U + \Delta U}, \frac{b}{U}) \\ &= a \times b \times \frac{U + \Delta U}{U} \\ E_{d2,2} &= a \times U \times \max(\frac{b}{U + \Delta U}, \frac{b}{U}) \\ &= a \times b \\ E_2 &= E_{d1,2} + E_{d2,2} \\ &= a \times b \times \frac{U + \Delta U}{U} + a \times b \\ &> 2 \times a \times b > E_1 \end{aligned} \tag{2}$$

Therefore, this application configuration increases dynamic energy without improving performance. For example, points A and B depict this case in Figure 4 for the two applications. For the application configuration that decreases only the utilization of C_1 by ΔU without changing the utilization of C_2, the dynamic energy increases and the performance decreases.

Now consider an application configuration that increases only the utilization of C_1 by ΔU and decreases the utilization of C_2 by ΔU. The average utilization is U. For example, the points on the lines C and D in Figure 4 exemplify this case.

$$\begin{aligned} E_{d1,3} &= a \times (U + \Delta U) \times \max(\frac{b}{U + \Delta U}, \frac{b}{U - \Delta U}) \\ &= a \times b \times \frac{U + \Delta U}{U - \Delta U} \\ E_{d2,3} &= a \times (U - \Delta U) \times \max(\frac{b}{U + \Delta U}, \frac{b}{U - \Delta U}) \\ &= a \times b \\ E_3 &= E_{d1,3} + E_{d2,3} \\ &= a \times b \times (1 + \frac{U + \Delta U}{U - \Delta U}) \\ &> a \times b \times (1 + \frac{U + \Delta U}{U}) > E_2 \\ &> 2 \times a \times b > E_1 \end{aligned} \tag{3}$$

Therefore, this application configuration increases dynamic energy while decreasing the performance. One can indeed explain the non-functional behaviour exhibited by points C and D (Figure 4) using the distribution of the individual core utilizations (that follow the simple EP model).

Hence, we prove energy nonproportionality for the simplest case of two homogeneous components executing different application configurations solving the same workload. In our future work, we will investigate energy nonproportionality in-depth for a real-life microprocessor with more than two homogeneous components and considering their realistic functional dynamic power models.

IV. Matrix Multiplication Application for Energy Proportionality Analysis of GPUs

This section presents our second contribution: the design and development of a matrix multiplication application used to analyze the weak EP of two Nvidia GPUs experimentally.

There were two design goals. The first design goal is to select an optimized CUDA code with one or more application-level decision variables. Having one or more decision variables with a high positive correlation with performance and dynamic energy will provide performance and dynamic energy tradeoffs (for a given problem size) and allow effective analysis of EP. We choose the blocked matrix multiplication supplied in the CUDA programming guide that allows the user to specify the per-block shared memory dimension. The shared memory is faster than global memory and minimizes the number of global memory accesses from a CUDA block. The CUBLAS DGEMM library routine is not selected since it lacks application-level tuning variables.

The second design goal is that the application should allow us to select CUPTI performance events and metrics [32] that satisfy the practical implications of the theory of energy predictive models of computing [33]. Briefly, the theory of energy predictive models of computing is a formalism containing properties of energy predictive models employing performance events as model variables. The properties are manifestations of the fundamental physical law of energy conservation. They capture the essence of single application runs and characterize the serial execution of two applications.

The practical implications of the theory include an *additivity* property for the selection of model variables that allow constructing accurate and reliable linear energy predictive models. The property is based on an intuitive and simple rule that if a model variable is employed in a linear energy predictive model, its count for a *compound* application should be equal to the sum of its counts for the executions of the base applications forming the compound application. A *compound application* is defined as the serial execution of two applications, which we call the *base* applications.

Therefore, our goal is to design a compound CUDA application that employs two base applications that can be executed one after the other on the GPU. CUPTI event counts are obtained for the two base applications separately. The CUPTI event count for the compound application is then obtained, and the additivity error is recorded [33]. Finally, the most additive CUPTI events are employed in constructing a qualitative linear dynamic energy model to accurately and reliably analyze EP by demonstrating the processor components with energy nonproportional activity.

A compound application executes two device matrix multiplication kernels one after the other on the GPU. There are two approaches for the serial execution of two device kernels. The first approach is to call the kernels one after another. The second approach is to call a kernel containing the two device matrix product codes repeated textually one after another. This grouping of codes provides an additional decision variable. The number of runs of a group forms the third decision variable.

The matrix multiplication application is illustrated in Figure 5. It computes $G \times R$ matrix products, $C = A \times B$, of two dense square matrices A and B of size $N \times N$. The application employs three decision variables. First, BS is the

per-block shared memory dimension employed during one matrix product call. The size of the shared memory used in one matrix product is $2 \times BS \times BS$. Second, G represents the size of a group of device matrix product codes repeated textually one after the other. For example, $G = 4$ means a group of four device matrix product codes repeated textually one after another. Finally, R is the number of runs of a group.

Lines 1-21 show the computations involved in a matrix product. Each thread block computes one square sub-matrix $Csub$ of C. Each thread within the block computes one element of $Csub$. $Csub$ is the product of a rectangular sub-matrix of A of dimensions, (N, BS) and a rectangular sub-matrix of B of dimensions, (BS, N). The two rectangular sub-matrices are divided into square matrices of size, $BS \times BS$. $Csub$ is the sum of the sub-products of these square matrices. Each sub-product is computed by loading the two corresponding square matrices from global memory to shared memory (Line 10). Each thread accumulates the result of each product into a register and writes the result to global memory (Line 20). The per-block shared memory dimension BS is passed as a template parameter.

Lines 22-35 show the eight groups of device matrix product codes. Group1 (*dgemmG1*) contains one device matrix product code. Group2 (*dgemmG2*) contains two device matrix product codes. Similarly, until Group 8. Lines 36-72 show the $G \times R$ matrix product invocations for block sizes BS varying from 1 to 32. The library function call, __syncthreads(), is used only to synchronize threads within a block. The blocks of threads are executed simultaneously by the GPU multiprocessors and there do not communicate with each other.

For a given matrix size N, the application is executed for all the possible combinations (BS, G, R). Due to the limited size of the per-block shared memory, only certain (G, R) combinations are permissible for a given BS. The dynamic energy and execution time is obtained for each valid combination. The resulting execution times and dynamic energies are used to analyze energy proportionality and determine the Pareto front.

V. Experimental Results

We study the weak energy proportionality of an Nvidia K40c GPU and an Nvidia P100 PCIe GPU whose specifications are given in Table I using the matrix multiplication application described above (Section 5). The dynamic energy and execution time shown in the graphs are for the CUDA kernel invocations only. The execution time of a CUDA kernel is measured using the *cudaEventCreate*, *cudaEventRecord*, *cudaEventSynchronize*, *cudaEventElapsedTime*, and *cudaEventDestroy* call sequence.

Each GPU resides in a separate node. The node hosting a GPU has one WattsUp Pro power meter, which sits between the wall A/C outlets and the input power sockets of the node. The power meter provides the total power consumption of the node. It has a data cable connected to one USB port of the server. A script written in Perl collects the data from the power meter using the serial USB interface. The execution of the

```
1  template <int BS> __device__ void dgemmG1(
2          double *C, double *A, double *B, int N) {
3      int bx = blockIdx.x; int by = blockIdx.y;
4      int tx = threadIdx.x; int ty = threadIdx.y;
5      int aBegin = N * BS * by; int aEnd   = aBegin + N - 1;
6      int aStep  = BS; int bBegin = BS * bx;
7      int bStep  = BS * N; double Csub = 0;
8      for (int a = aBegin, b = bBegin; a <= aEnd;
9           a += aStep, b += bStep) {
10         __shared__ double As[BS][BS], Bs[BS][BS];
11         As[ty][tx] = A[a+N*ty+tx];
12         Bs[ty][tx] = B[b+N*ty+tx];
13         __syncthreads();
14 #pragma unroll
15         for (int k = 0; k < BS; ++k)
16             Csub += As[ty][k] * Bs[k][tx];
17         __syncthreads();
18     }
19     C[N*BS*by + BS*bx + N*ty + tx] += Csub;
20 }
21 template <int BS> __device__ void dgemmG2(
22     double *C, double *A, double *B, int N) {
23      /* dgemmG1 matrix multiplication code here... */
24      __syncthreads();
25      /* dgemmG1 matrix multiplication code here... */
26      __syncthreads();
27 }
28 /* dgemmG3, ..., dgemmG7 */
29 template <int BS> __device__ void dgemmG8(
30     double *C, double *A, double *B, int N) {
31     /* dgemmG1 code here... */
32     __syncthreads();
33     /* dgemmG1, __syncthreads() repeated 7 more times */
34 }
35 __global__ void dgemm1(double *C, double *A, double *B,
36             const int N, const int G, const int R) {
37     for (int run = 0; run < R; run++)
38         if (G == 1)
39             dgemmG1<1>(C, A, B, N);
40         /* dgemmG2<1>(...);
41             ...,
42             dgemmG7<1>(...); */
43         if (G == 8)
44             dgemmG8<1>(C, A, B, N);
45 }
46 /* dgemm2, ..., dgemm7 */
47 __global__ void dgemm8(double *C, double *A, double *B,
48             const int N, const int G, const int R) {
49     for (int run = 0; run < R; run++)
50         if (G == 1)
51             dgemmG1<8>(C, A, B, N);
52         /* dgemmG2<8>, dgemmG3<8>, ..., dgemmG7<8> here */
53         if (G == 8)
54             dgemmG8<8>(C, A, B, N);
55 }
56 /* dgemm9, ..., dgemm31 */
57 __global__ void dgemm32(double *C, double *A, double *B,
58             const int N, const int G, const int R) {
59     for (int run = 0; run < R; run++)
60         if (G == 1)
61             dgemmG1<32>(C, A, B, N);
62         if (G == 2)
63             dgemmG2<32>(C, A, B, N);
64 }
```

Fig. 5: The matrix multiplication application employed to analyze the energy proportionality of our GPUs. Routine $dgemmx(C, A, B, N, G, R), x \in \{2, \ldots, 32\}$ computes $G \times R$ matrix products, $C = A \times B$, of two dense square matrices A and B of size $N \times N$ where each product employs per-block shared memory dimension, x. BS is the per-block shared memory dimension. G is the size of a group of repeated device matrix product codes. R is the number of runs of a group.

script is non-intrusive and consumes insignificant power. An automated tool, HCLWATTSUP [34], is used to determine the dynamic and total energy consumptions. HCLWATTSUP has no extra overhead and, therefore, does not influence the energy

consumption of the kernel. Several precautions are taken in computing energy measurements to eliminate the potential disturbance due to components such as SSDs and fans.

The application is executed repeatedly to obtain an experimental data point until the sample mean lies in the 95% confidence interval, and a precision of 0.025 (2.5%) is achieved. For this purpose, Student's t-test is used assuming that the individual observations are independent and their population follows the normal distribution. Finally, we verify the validity of these assumptions using Pearson's chi-squared test.

A. Energy-Performance Interplay Using Only G

This section demonstrates the interplay between the dynamic energy and performance of the matrix multiplication employing only the decision variable G. The other application-level configuration parameters, (N, BS, R), are fixed.

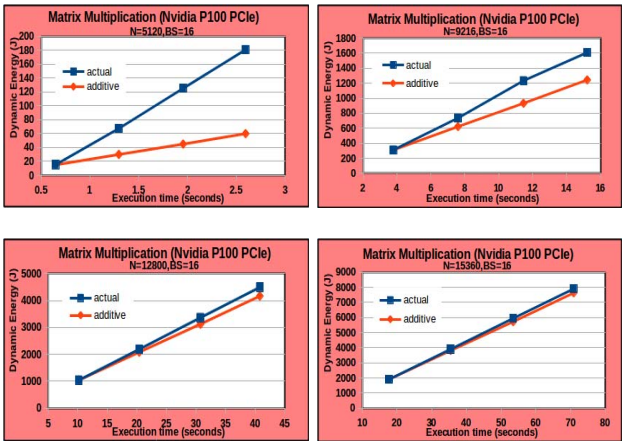

Fig. 6: Energy expensive activity on Nvidia P100 PCIe shown by non-additivity of dynamic energy as G is increased from 1 to 4 for the matrix multiplication application (section IV). For the additive graphs in the plots (shown in red), the dynamic energy is $G \times E_{g1}$ where E_{g1} is the dynamic energy for $G = 1$. The dynamic energies are highly non-additive for N=5120. The non-additivity keeps decreasing before becoming zero for matrix sizes exceeding N=15360.

Figure 6 shows the relationship between dynamic energy and execution time for Nvidia P100 GPU as the variable G is increased. For the additive graphs in the plots (shown in red), the dynamic energy is $G \times E_{g1}$ where E_{g1} is the dynamic energy for $G = 1$. The execution times are observed to be additive.

The dynamic energies start highly non-additive and become additive for matrix sizes exceeding 15360. Similar behaviour is observed for Nvidia K40c GPU where the dynamic energies start highly non-additive and become additive for matrix sizes exceeding 10240. The non-additivity of the dynamic energy in the Figure 6 is due to an energy-expensive component consuming constant dynamic power consumption of 58 W. If we include this dynamic power in the static power consumption, then the resulting dynamic energy consumption becomes additive. We will investigate if this behaviour is application-specific in our future work.

B. Energy Proportionality and Pareto Front

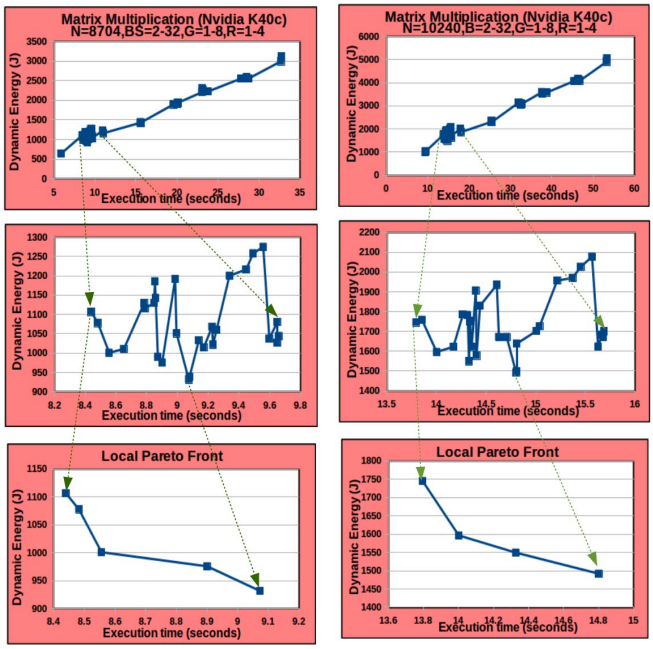

Fig. 7: Energy nonproportionality of Nvidia K40c obtained by executing the matrix multiplication application (section IV) for two matrix sizes, N=8704 (left column) and N=10240 (right column). The top plot graphs the dynamic energies and execution times for all application configurations. The middle plot shows the energy non-proportionality region where bi-objective optimization for dynamic energy and performance results in a local Pareto front (shown in the bottom plot). The Pareto fronts are discrete and contain the solid square points. The points are connected by lines for visualization purposes only.

This section analyzes the interplay between dynamic energy and performance by studying the graphs containing data points for all the application configurations for a specific matrix size N. For some matrix sizes, the *global Pareto front* contains only one point signifying that the performance-optimal solution is also optimal for dynamic energy. We show a *local Pareto front* for such matrix sizes, which contains solutions that are less optimal than the solutions in the global Pareto front. There are two reasons for doing this. First, determining a global Pareto front by exhaustively obtaining the data points for all the application configurations can be expensive and may not be feasible in dynamic environments with time constraints. Second, local Pareto fronts contain regions of high energy nonproportionality that provide many diverse trade-off solutions (application configurations) for dynamic energy and performance.

Figure 7 shows the energy nonproportionality and local Pareto fronts for the Nvidia K40c GPU for matrix sizes, N=8704 and N=10240. The global Pareto front consists of only one point for the matrix sizes used in our experiments, signifying that the performance-optimal solution is optimal for dynamic energy also. We observe that a maximum of 18% dynamic energy saving can be obtained while tolerating a

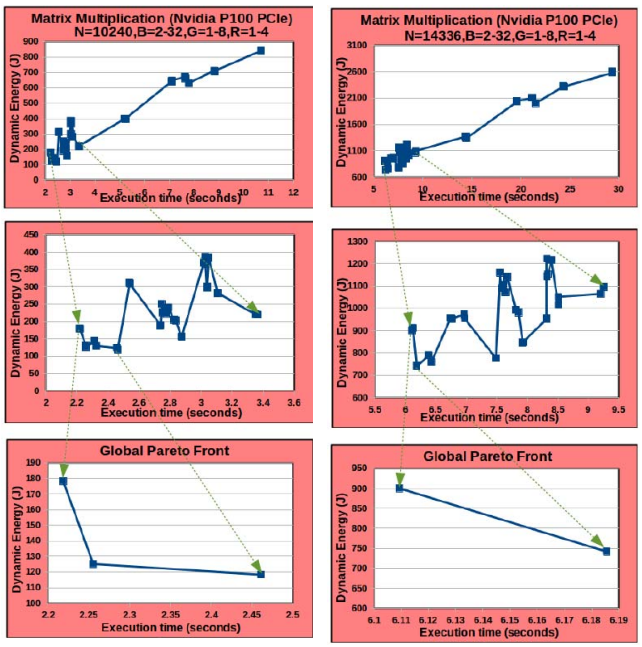

Fig. 8: Energy nonproportionality of Nvidia P100 PCIe obtained by executing the matrix multiplication application (section IV) for two matrix sizes, N=10240 (left column) and N=14336 (right column). The energy nonproportionality (middle plots) results in a global Pareto front shown in the bottom plots. For N=10240, there are three points in the global Pareto front where allowing 11% performance degradation (from the performance-optimal solution) provides 50% dynamic energy saving.

7% performance penalty. The observed average and maximum points in the local Pareto fronts are four and five.

Figure 8 shows the energy nonproportionality and the global Pareto front for the Nvidia P100 PCIe GPU for matrix sizes, N=10240 and N=14336. For the range of matrix sizes employed in our experiments, we find that allowing a maximum performance degradation of 11% (from the performance-optimal solution) can provide a maximum of 50% dynamic energy saving. The observed average and maximum points in the global Pareto fronts are two and three.

C. Discussion

For the Nvidia K40c GPU, the global Pareto front contains only one point for the matrix sizes employed in our experiments. The value of BS for this configuration is 32, the maximum allowed by the application. However, regions of high energy nonproportionality exist that contain local Pareto fronts with diverse dynamic energy and performance tradeoffs. For the Nvidia P100 PCIe GPU, the average and the maximum number of points in the global Pareto front are 2 and 3. It suggests that the GPU architectures are becoming more complex and, therefore, present an excellent opportunity for the application programmer to pursue global bi-objective optimization using application-level decision variables.

One approach to explain the energy nonproportionality is to study the relationship between dynamic energy and performance by using a dynamic energy model employing performance monitoring counters (PMCs) and resource utilization as model variables and the application's execution time. Khokhriakhov et al. [8] propose a qualitative dynamic energy model based on variables reflecting TLB activity (the duration of page walk) and average CPU utilization. The model variables are selected using the theory of energy of computing [33] and a high positive correlation with dynamic energy to analyze the energy nonproportionality on their experimental platforms. The model demonstrates that the energy nonproportionality is due to the disproportionately energy-expensive data translation lookaside buffer (dTLB) activity.

CUPTI events and metrics are provided for performance profiling and are employed in dynamic energy predictive models [35], [36], [37]. However, we observed many key events and metrics overflow for large matrix sizes ($N > 2048$) and reported inaccurate counts. Therefore, the CUPTI library is inadequate to analyze the energy nonproportionality of the GPUs. We will explore in our future work methods and tools to explain the discovered energy nonproportionality of the GPUs.

VI. Conclusion

Energy-proportional computing is fundamental to addressing the grand technological challenge of energy efficiency in Information and Communications Technology. Therefore, the energy proportionality (EP) of the dominant processors, multicore CPUs and GPUs, is a crucial design goal for architects. In this work, we presented formal definitions of strong and weak notions of EP. The strong notion of EP signifies designing a system that consumes dynamic energy proportional to the amount of work it performs. On the other hand, the weak notion of EP implies that the dynamic energy consumption is a constant for all the application configurations solving the same workload while requiring careful design of the parallel application to ensure that the variations of the core utilizations are due to the complexity of the hardware architecture and not the parallel algorithm.

Unfortunately, multicore CPUs are found to violate both strong and weak EP [8]. In this work, we presented the first attempt at a theoretical analysis of the weak energy nonproportionality of multicore CPUs. Specifically, we considered the simplest case of two homogeneous cores executing different application configurations solving the same workload. We showed that dynamic energy increases in all situations when there are differences between the utilizations of the cores, thereby contravening the simple EP model.

On the other hand, a GPU is designed carefully and is less heterogeneous than a multicore CPU, where a more significant part of its on-chip resources are dedicated to data computations rather than caching and flow control. Consequently, the mainstream view is that GPUs exhibit strong and weak EP. However, GPUs were found to breach strong EP ([12]).

In this work, we studied the weak EP of an Nvidia K40c GPU and an Nvidia P100 PCIe GPU by employing a specially designed matrix multiplication application. Both the GPUs were shown to exhibit energy nonproportionality. While energy nonproportionality is undesirable, it presents an

opportunity for bi-objective optimization of the application for dynamic energy and performance. By analyzing the Pareto fronts of dynamic energy and performance for a wide range of workloads, we find that the maximum dynamic energy savings can be up to 18% while tolerating a performance degradation of 7% for Nvidia K40c GPU and (50%,11%) respectively, for Nvidia P100 PCIe GPU.

REFERENCES

[1] R. Lucas and et al., "DOE advanced scientific computing advisory subcommittee (ASCAC) report: Top ten exascale research challenges," 2 2014. [Online]. Available: https://www.osti.gov/biblio/1222713

[2] A. Andrae, "New perspectives on internet electricity use in 2030," *Engineering and Applied Science Letters*, vol. 3, pp. 19–31, 06 2020.

[3] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, 1992.

[4] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, no. 12, pp. 33–37, 2007.

[5] F. Ryckbosch, S. Polfliet, and L. Eeckhout, "Trends in server energy proportionality," *Computer*, vol. 44, no. 09, pp. 69–72, sep 2011.

[6] D. Wong and M. Annavaram, "KnightShift: Scaling the energy proportionality wall through server-level heterogeneity," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 119–130.

[7] B. Subramaniam and W.-c. Feng, "Towards energy-proportional computing for enterprise-class server workloads," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '13. ACM, 2013, pp. 15–26.

[8] S. Khokhriakov, R. R. Manumachu, and A. Lastovetsky, "Multicore processor computing is not energy proportional: An opportunity for bi-objective optimization for energy and performance," *Applied Energy*, vol. 268, p. 114957, 2020.

[9] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," *SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 338–347, Jun. 2010.

[10] J. Guerra, W. Belluomini, J. Glider, K. Gupta, and H. Pucha, "Energy proportionality for storage: Impact and feasibility," *SIGOPS Operating Systems Review*, vol. 44, no. 1, pp. 35–39, Mar. 2010.

[11] T. Härder, V. Hudlet, Y. Ou, and D. Schall, "Energy efficiency is not enough, energy proportionality is needed!" in *Database Systems for Advanced Applications*, J. Xu, G. Yu, S. Zhou, and R. Unland, Eds. Springer, 2011, pp. 226–239.

[12] H. Khaleghzadeh, M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, "Bi-objective optimization of data-parallel applications on heterogeneous HPC platforms for performance and energy through workload distribution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 543–560, 2021.

[13] M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, "A comparative study of methods for measurement of energy of computing," *Energies*, vol. 12, no. 11, 2019.

[14] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 13–23.

[15] S. Rivoire, P. Ranganathan, and C. Kozyrakis, "A comparison of high-level full-system power models," in *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, ser. HotPower'08. USA: USENIX Association, 2008, p. 3.

[16] L. Yu, Z. Zhou, S. Wallace, M. E. Papka, and Z. Lan, "Quantitative modeling of power performance tradeoffs on extreme scale systems," *Journal of Parallel and Distributed Computing*, vol. 84, pp. 1–14, 2015.

[17] N. Gholkar, F. Mueller, and B. Rountree, "Power tuning HPC jobs on power-constrained systems," in *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*. ACM, 2016, pp. 179–191.

[18] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, "Bounding energy consumption in large-scale MPI programs," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, Nov 2007, pp. 1–9.

[19] Y. Kessaci, N. Melab, and E.-G. Talbi, "A pareto-based metaheuristic for scheduling HPC applications on a geographically distributed cloud federation," *Cluster Computing*, vol. 16, no. 3, pp. 451–468, Sep. 2013.

[20] J. J. Durillo, V. Nae, and R. Prodan, "Multi-objective energy-efficient workflow scheduling using list-based heuristics," *Future Generation Computer Systems*, vol. 36, pp. 221–236, 2014.

[21] J. Kołodziej, S. U. Khan, L. Wang, and A. Y. Zomaya, "Energy efficient genetic-based schedulers in computational grids," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 4, pp. 809–829, Mar. 2015.

[22] J. Lang and G. Rünger, "An execution time and energy model for an energy-aware execution of a conjugate gradient method with CPU/GPU collaboration," *Journal of Parallel and Distributed Computing*, vol. 74, no. 9, pp. 2884–2897, 2014.

[23] A. Chakrabarti, S. Parthasarathy, and C. Stewart, "A pareto framework for data analytics on heterogeneous systems: Implications for green energy usage and performance," in *Proceedings of the 46th International Conference on Parallel Processing*. IEEE, 2017, pp. 533–542.

[24] A. Lastovetsky and R. Reddy, "New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1119–1133, April 2017.

[25] R. R. Manumachu and A. Lastovetsky, "Bi-objective optimization of data-parallel applications on homogeneous multicore clusters for performance and energy," *IEEE Transactions on Computers*, vol. 67, no. 2, pp. 160–177, 2018.

[26] R. Reddy Manumachu and A. L. Lastovetsky, "Design of self-adaptable data parallel applications on multicore clusters automatically optimized for performance and energy through load distribution," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 4, p. e4958, 2019.

[27] K. M. Tarplee, R. Friese, A. A. Maciejewski, H. J. Siegel, and E. K. Chong, "Energy and makespan tradeoffs in heterogeneous computing systems using efficient linear programming techniques," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 6, pp. 1633–1646, 2016.

[28] M. A. Aba, L. Zaourar, and A. Munier, "Approximation algorithm for scheduling a chain of tasks on heterogeneous systems," in *Proceedings of the European Conference on Parallel Processing*. Springer, 2017, pp. 353–365.

[29] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proceedings of the 41st ACM/IEEE International Symposium on Computer Architecture*, 2014, pp. 301–312.

[30] C.-H. Hsu and S. W. Poole, "Measuring server energy proportionality," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '15. ACM, 2015, pp. 235–240.

[31] R. Sen and D. A. Wood, "Energy-proportional computing: A new definition," *Computer*, vol. 50, no. 8, pp. 26–33, 2017.

[32] NVIDIA Corporation, "CUDA profiling tools interface (CUPTI) - 1.0," 2021. [Online]. Available: https://developer.nvidia.com/cupti-1_0

[33] A. Shahid, M. Fahad, R. R. Manumachu, and A. Lastovetsky, "Energy predictive models of computing: Theory, practical implications and experimental analysis on multicore processors," *IEEE Access*, vol. 9, pp. 63 149–63 172, 2021.

[34] M. Fahad and R. R. Manumachu, *HCLWattsUp: energy API using system-level physical power measurements provided by power meters*. Heterogeneous Computing Laboratory, University College Dublin, April 2021. [Online]. Available: https://csgitlab.ucd.ie/manumachu/hclwattsup

[35] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," in *Proceedings of the International Conference on Green Computing*, 2010, pp. 115–122.

[36] Y. Zhang, Y. Hu, B. Li, and L. Peng, "Performance and power analysis of ATI GPU: A statistical approach," in *Proceedings of the IEEE Sixth International Conference on Networking, Architecture, and Storage*, ser. NAS '11. IEEE Computer Society, 2011, pp. 149–158.

[37] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A simplified and accurate model of power-performance efficiency on emergent GPU architectures," in *Proceedings of the 27th IEEE International Symposium on Parallel and Distributed Processing*, 2013, pp. 673–686.