Cent. Eur. J. Comp. Sci. • 3(1) • 2013 • 1-16
DOI: 10.2478/s13537-013-0101-5

VERSITA

Central European Journal of **Computer Science**

Performance and power analysis for high performance computation benchmarks

Research Article

Joseph Issa*

Department of Computer Engineering, Santa Clara University, Santa Clara, CA 95053-0566, USA

Received 11 May 2012; accepted 23 January 2013

Abstract: Performance and power consumption analysis and characterization for computational benchmarks is important for processor designers and benchmark developers. In this paper, we characterize and analyze different High Performance Computing workloads. We analyze benchmarks characteristics and behavior on various processors and propose a performance estimation analytical model to predict performance for different processor micro-architecture parameters. Performance model is verified to predict performance within <5% error margin between estimated and measured data for different processors. We also propose a power estimation analytical model to estimate power consumption with low error deviation.

Keywords: performance and power analysis • high performance computation

1. Introduction

Computational benchmarks are important in measuring and ranking processor's performance and performance-per-watt. LINPACK is a software algorithm implemented to do double precision matrix multiplication; it mainly utilizes matrix-matrix multiply routine from the BLAS [14] (Basic Linear Algebra Subprograms) to perform basic vector and floating-point matrix multiplication. The performance metric for LINPACK [14] defined by the floating-point operations a processor can complete per second known as FLOPS. The LINPACK algorithm for matrix multiplication consists of two parts, multiplication, and addition of double-precision floating-point (64 bits). SGEMM [24] is a compute bound benchmark for matrix multiplication using single precision floating-point operation instead of double precision. MONTECARLO workload is a compute bound workload used by financial institutions to compute option pricing. All the three High Performance Computational (HPC) workloads used in this paper are open-source code. We start with analyzing SGEMM, MONTECARLO and LINPACK benchmarks and their performance sensitivity. We then present a performance prediction model to estimate performance score for different processor configurations. We combined both aspects of performance estimation in one model, the first part related to processor architecture metrics, and the second is the characteristics of

* *E-mail: jissa@scu.edu*

the benchmark with respect to performance. The performance analysis and estimation for LINPACK is based on AMD K8, K10 processors, and Intel Core2 processor. For SGEMM, the performance and power analysis and estimations is based on Nvidia's GTX480, GTX570, and GTX580, note that all Nvidia's GTX graphics cards used are none High Performance Compute (HPC) that means, they are only capable of doing single precision matrix multiplication. For MONTECARLO, we used Nvidia's GTX580 and Tesla 2050 graphics cards.
Since performance and compute power comes at the expense of power budget, we present a performance-per-watt analysis and estimation model for SGEMM and MONTECARLO based on Nvidia's architecture and LINPACK based on Intel architecture. The basic definition for performance-per-watt is a measure of the energy efficiency of processor architecture. It is defined as the rate of transactions or computations score for a benchmark that can delivered by a processor for every watt of power consumed. In this paper, we propose a performance-per-watt model in which we can model performance and power for SGEMM and LINPACK to understand the optimum operating point while achieving higher performance-per-watt. We also propose a prediction method, which predicts the performance-per-watt for SGEMM at higher core frequencies for different matrix sizes and higher number of cores for LINPACK. In the results section, we show that the model can predict performance, power and performance-per-watt with <5% error deviation between estimated and measured data. The remainder of the paper is organized as follows; we start with related work section, in which we compare our analysis and estimation model to different published papers. In Section 3, we present overview and analysis for both benchmarks, in Section 4 we derive the performance prediction model. In Section 5, we present performance estimations experimental results, in Sections 6 we derive and analyze performance-per-watt estimation models with experimental results, and conclude in Section 7.

2. Related work

In this paper, we propose an analytical model to predict performance for LINPACK, MONTECARLO and SGEMM benchmarks with error <5% between measured and estimated data. We also present a power prediction model for SGEMM based on Nvidia architecture. Several researchers have worked on predicting processor performance and power consumption on given benchmarks using different prediction methods including simulation trace-based methods. Our performance and power consumption prediction models identifies performance dependencies and bottlenecks for a given processor and workload. The model can be used to estimate performance and power for different processor settings (i.e. frequency, core shaders, Instructions-per-Cycle (IPC), efficiency, and execution time), as well as model workload specific variables affecting performance and power, such as matrix size.
Kamil et al. [13] presented different power measurements and estimation methods for different HPC benchmarks. The paper does not provide any performance-per-watt analysis and estimation method with respect to processor frequency and other benchmark metrics scaling for different HPC workloads.
Chou [6] presented a LINPACK estimation model based on computation power and message passing overhead for large clusters with error margin <5%. His method does not show any processor architecture dependencies and their impact on performance score. He presented matrix size scaling with respect to performance score, but does not show any scaling or sensitivity analysis for different processor metrics.
Bhatai et al. [4] presented a performance prediction model that combines both architecture and application-specific performance characteristics for HYCOM application. The error obtained is < 30% for several test cases. In this paper we propose a different approach than what is presented in [25] and we verified that the estimation method can estimate performance <5% for SGEMM and LINPACK benchmarks.
Jen [12] presented a performance prediction model for LINPACK based on predicting runtime using message-passing model. Our performance prediction model approach is different, in which we analyze processor hardware and workload characteristics dependencies affecting LINPACK and SGEMM performance.
Goel et al. [7] presented a per-core linear power model using sampled performance counter on single and multithreaded applications. Error deviation is <5% for all tested workloads.

3. Benchmark overview and analysis

3.1. LINPACK overview

LINPACK is one of the top HPCC (High Performance Computing Challenge) benchmarks. It is used to rank top HPC systems, higher the GFlops, higher the rank. It is also ranked as a top ranking benchmark among other HPC benchmarks such as Stream, FFT, and DGEMM. There is no common library for LINPACK; the code is optimized differently for different processors. In addition, the matrix size and configuration is different for different processor's core numbers and topologies. A processor-specific binary achieves better score compared to when the same code is executed on a different processor. In general, LINPACK is an algorithm to show hardware floating point efficiency rather than a workload efficiency. LINPACK workload is divided into two phases, the floating time phase and the synchronization time phase. The floating-point library used is BLAS and the sync library used is OpenMPI. We used a profiling tool to confirm that the floating-point phase is the dominant phase for LINPACK as shown in Figure 1.

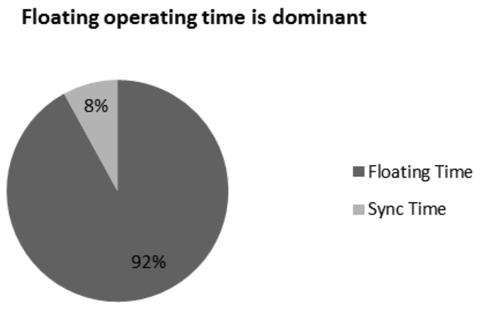

Figure 1. Floating time and Sync time profiling for LINPACK.

There are different factors that determine the GFLOPS score for LINPACK. In general, the definition for GFLOPS is the total floating points operations per time unit. Since LINPACK consists of two time phases, which are defined as the floating-point running time *T1* and sync phase running time *T2*, GFLOPS is then calculated based on both:

$$GFLOPS = \frac{G}{T1+T2} = \frac{G/T1}{1+T2/T1} = \frac{eff \times peak}{1+T2/T1}, \tag{1}$$

where G is total number of floating point operations from the LINPACK algorithm, which is determined as a function on N, where N is the matrix size. The relation between G and N is given by:

$$G = \frac{2}{3} \times N^3 + \frac{3}{2} \times N^2. \tag{2}$$

Floating phase efficiency *eff* is constant for certain processors, and *peak* is defined as the maximum GFLOPS score a given processor can achieve. In Figure 2, we show sync time ratio and GFLOPS from peak scaling with matrix size using Opteron AMD processor. At a certain point for matrix size (i.e., >30K), the sync time ratio and the GFLOPS from peak stays flat as shown in Figure 2.
Efficiency for floating point is defined in Eq. (3):

$$eff = \frac{FlOpPerCyc_{perform}}{FlOpPerCyc_{peak}}, \tag{3}$$

where the efficiency for the floating phase is constant and can be narrowed down to 92%-98% for most recent processors. It is defined as the ratio of floating-point operations performed per cycle in the floating-point phase divided by the maximum hardware floating operation per cycle. Floating phase ratio is defined by the code, and floating operation per cycle (peak) is determined by the processor architecture. In summary, the factors that determine GFLOPS are floating phase efficiency, peak, and sync time ratio. We did sensitivity analysis for AMD K8 [1] and Intel Core 2 processors for

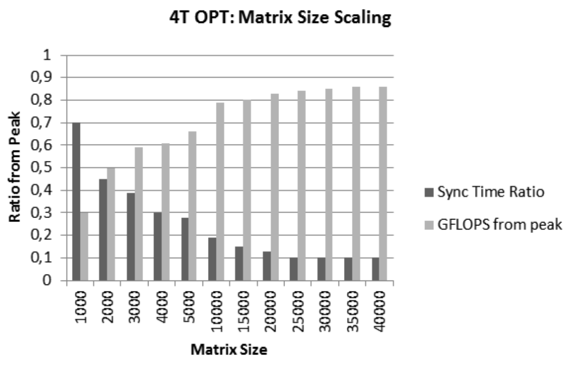

Figure 2. Matrix size scaling.

matrix size =40K. For AMD K8 processor, we conclude that floating phase is not sensitive to configurations outside the core for matrix size =40K.

All normalized results are ~1.0, which is the normalized factor relative to a baseline. We can conclude that there is no sensitivity with respect to core frequency, memory type, and threads for matrix size >40K. For small size matrices, we expect scaling to occur with respect to core frequency and number of threads.

The second phase or LINPACK is the sync phase. Different configuration can change the instruction mix for the sync phase. The sync-time ratio is small when the matrix size is large as shown before in Figure 2. The ratio of sync phase running time (*T*_1) to floating phase running time (*T*_2) is then defined as

$$T_1/T_2 = \frac{f_1 \times O(N^2)}{O(N^3)} = f_1 \times O(\frac{1}{N}), \tag{4}$$

where *f_1* is the sync parameter related to memory or link latency.

3.2. SGEMM overview

SGEMM is a single precision matrix multiplication benchmark, which uses the following equation for matrix multiplication, where A is an MxN matrix, B is an N∗K matrix and C is an M∗K matrix, and α and β are scalars:

$$c = \alpha * op(A) * op(B) + \beta * C. \tag{5}$$

SGEMM is a compute bound application, the task of computing the product of two matrices A and B can be divided among multiple thread blocks as follows: each thread block computes one sub-block sub-C for C, and each thread within the thread block computes several elements of sub-C as shown in Eq. (5). The performance outcome for SGEMM performance is the number of floating point operations measured in GFLOPS.

3.3. Cache optimization for matrix multiplication

The optimization of cache has the greatest potential for performance improvements. Cache blocking factor is the most popular for cache optimization which will reduce the amount of cache misses. In this section, we discuss optimization techniques for L1 and L2 caches to improve performance for SGEMM and LINPACK. We will not cover software and/or compiler optimization. A basic matrix multiplication procedure is as follows:

```
matrix_multiple (A,B,C,m)
   i= 1 to m
      for j= 1 to m
         for k= 1 to m
            C(i,j) = C(i,j)+ A(i,k)*B(k,j)
```

Matrix multiplication is done in blocks, so optimizing the block size will yield to optimized performance. If the two blocks we want to multiply can fit in L1 cache, this will avoid getting into cache misses which improves performance. An optimum L1 and L2 cache size is calculated as $2*(\text{blocksize})^2*\text{wordsize}$. Assume L1 cache size is 16K, we calculate block size to be 32, and for L2 size of 512 KB, the optimized block size is calculated at 181. The wordsize value used for double data type is 8 bytes.

3.4. MONTECARLO overview

Financial institutions use MONTECARLO a computational model, to determine efficient option pricing implementations using CUDA. For example, if the market stock price at the exercise date is greater than the strike price, a call option makes its holder profit; otherwise, there is no profit. MONTECARLO approach for option pricing mathematically estimates the call and put options by seller and buyer and generates the last number of random samples. It then calculates the derivative end-period prices corresponding to each sample and averages the generated prices. MONTECARLO is a compute bound benchmark, so computing one option per thread generally does not efficiently occupy the GPU, so the benchmark uses multiple threads per option. CUDA-enabled GPUs can efficiently and accurately formulate MONTECARLO option pricing for small and large path counts. In our performance estimation model, we search for performance in options/sec processed by the GPU.

4. Performance estimation model derivation

4.1. LINPACK performance estimation model

The variables required to calculate *GFLOPS* is defined as a function of *eff*, *peak,* and *ratio* that can be defined as

$$GFLOPS = \frac{eff \times peak}{1 + T2/T1} = eff \times peak \times \frac{T1}{T1 + T2} = eff \times peak \times (1 - ratio), \tag{6}$$

where *T1* is floating phase running time and *T2* sync phase running time, *ratio* is sync time running ratio, and *eff* is the floating phase efficiency which is a constant for certain processors, and *peak* is calculated as

$$peak = frequency \times number\, of\, cores \times flops\, per\, cycle\,. \tag{7}$$

We derived the equation for *ratio* as a function of number of cores and we plotted sync time ratio at different number of threads. In general, for Intel processors, each core represents two hardware threads. GPUs are effective at performing fat calculation due to their pipeline, which allows for example up to 320 computations at the same time on a single GPU. While the CPU can do parallel computations but this is limited to the number for cores available.

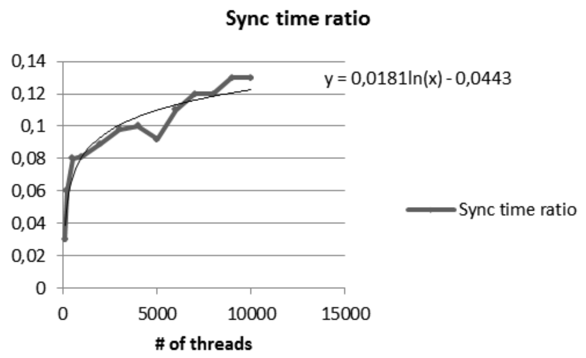

Figure 3. Sync-time ratio vs. # of Threads.

For low number of threads, the sync-time ratio is ∼4%, for clusters with 10000 cores and it can be narrowed to ∼10%. We used the LN() curve-fitting to narrow the sync time ratio, as shown in Figure 3, to obtain the following equation:

$$ratio = 0.0181 \times LN(CoreNum) - 0.0443\,. \tag{8}$$

4.2. SGEMM performance estimation model

The next benchmark we use for modeling performance is SGEMM; we used CUDA profiling tool from Nvidia to collect specific GPU related performance metric for SGEMM. The first step to model SGEMM application specific behavior is to understand how *warp#* changes when the size of matrix increases. The *warp#* can be curve fitted as a function of matrix size N with respect to loop count as shown in Figure 4 to obtain the following equation:

$$Warp\# = 0.0009 * N^2 + 6 * 10^{-14} * N - 9 * 10^{-11}\,, \tag{9}$$

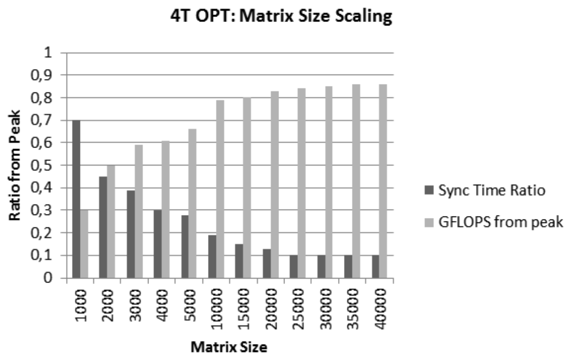

Figure 4. Matrix size *N* vs. Loop Count for SGEMM.

where N is the matrix size. The *loop_per_warp_thread* and *instruction_per_warp_thread* data is collected using the CUDA profiling tool. We also curve-fitted both variables as a function of matrix size N:

$$loop\_per\_warp\_thread = 0.062 * N\,, \tag{10}$$

$$Instruction\_per - warp\_thread = 834 * loop\_per\_warp\_thread\ +\ 253\,, \tag{11}$$

Total Flops is calculated as a function of matrix size N:

$$Total\_FLOPS = 2 * N^3\,. \tag{12}$$

The final step for the estimation model is to calculate the GFLOPS/sec per unit time. The *total_time* variable can be derived as

$$Total\_time = \frac{Total\_cycle}{Frequency}\,, \tag{13}$$

where T*otal-cycle* is

$$Total\_cycle = \frac{Path\_length}{\frac{Core\#}{(Efficiency*IPC)}}\,, \tag{14}$$

IPC∼1.0 and efficiency is in the range of ∼0.92-0.98 where

$$Efficiency = \frac{Operations\_per\_cycle\_perform}{Operations\_per\_cycle\_peak}, \tag{15}$$

$$Path\_length = Instruction\_per\_warp\_thread \ * warp\# \tag{16}$$

and

$$GFLOPS/\sec = Total\_GFLOPS/Total\_time. \tag{17}$$

We can determine that upper bound SGEMM performance runs in the order of N^3. We can also conclude from the final estimation equation for SGEMM that its performance is directly proportional to frequency, efficiency, IPC, number of core shaders, and inversely proportional to *warp#* and total time.

4.3. MONTECARLO performance estimation model

In this section, we derive a set of equations for the MONTECARLO benchmark performance model. We collected most data using the CUDA profiling tool provided by Nvidia. These equations are based on generic GPU architecture but are also specific to benchmark behavior. The number of warps is generally the total number of threads divided by 32. Applying this to the benchmark, we get the following:

$$warp\# = \frac{M * 256}{32}, \tag{18}$$

where *warp#* is the total number of threads divided by 32, M is the number of options, *loop_per_warp_thread* is defined as

$$loop\_per\_warp\_thread = \frac{N}{1024} \ , \tag{19}$$

where N is the simulation path. Using the CUDA profiling tool, we analyzed the logic loops through instructions issued per warp as shown in Figure 5.

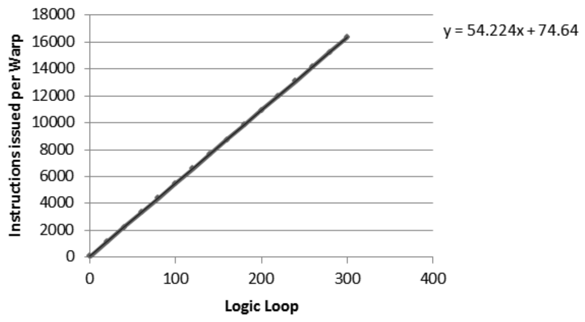

Figure 5. Logic Loop vs. Instructions issued per Warp.

$$Instructions\_per\_warp\_thread = \ 54.4 * loop\_per\_warp\_thread + 74.64 \tag{20}$$

and

$$Options/\sec = \frac{M}{Total\_time}, \tag{21}$$

M is defined as the number of options, *SM#* is number of core shaders, frequency is the GPU core frequency, IPC ~1.0, Efficiency ~r0.92, and N is the simulation path. From these equations, we see that the MONTECARLO benchmark performance is directly proportional to number of core shaders, IPC, efficiency, and core frequency, and is inversely proportional to simulation path N, *warp#* and total time.

5. Experimental results

5.1. LINPACK experimental results

In this section, we show that the performance estimation model for LINPACK can estimate performance with minimum error between estimated and measured GFLOPS. For this experiment, we used two AMD K8 [1] and K10 [6] systems with different core frequencies and number of threads so we can verify that the model can estimate and scale properly with respect to core frequency and number of cores. In all experiments, the error deviation between estimated and measured was <2% as shown in Figures 6–9.

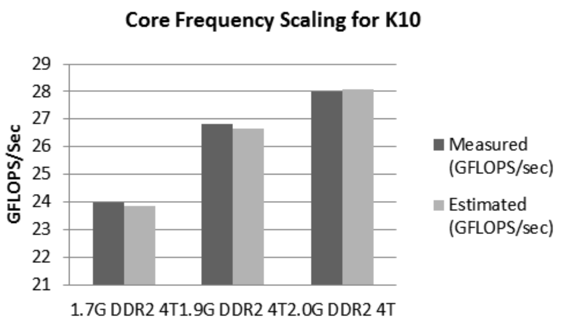

Figure 6. K10-Core Frequency scaling.

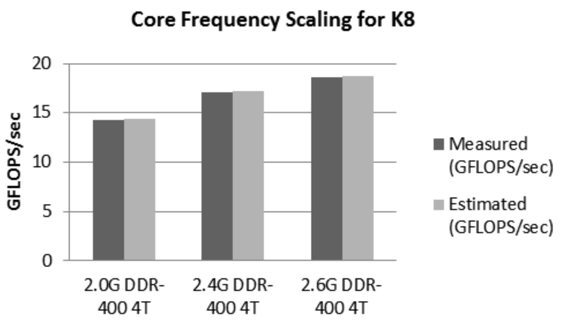

Figure 7. K8-Core Frequency scaling.

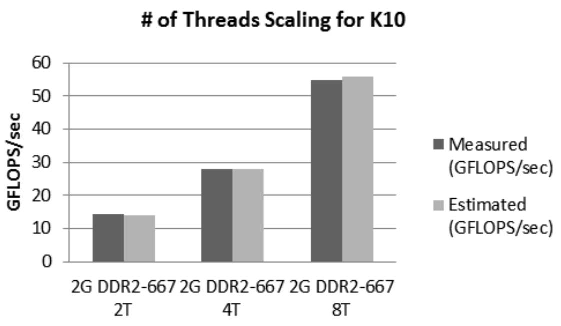

Figure 8. K10 -Thread scaling.

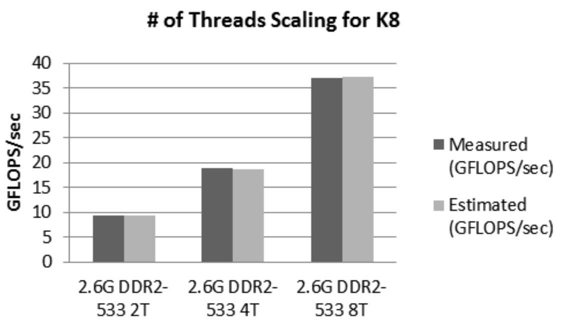

Figure 9. K8-Thread scaling.

Applying the same model for Intel architecture processor, using the same model derived for LINPACK, we show that error deviation between measured and estimated data is <2%, again this is using the same performance estimation model derived for LINPACK for different processor architecture. The only equation we change for Intel Architecture is the ratio, which is derived as:

$$ratio = 0.0072 \times LN(CoreNum) + 0.0436\,. \tag{22}$$

From Figures 10-12, we show that using the same LINPACK estimation model for Intel architecture processor, we can estimate LINPACK score with error deviation <2% for different core frequencies, number of threads and memory type.

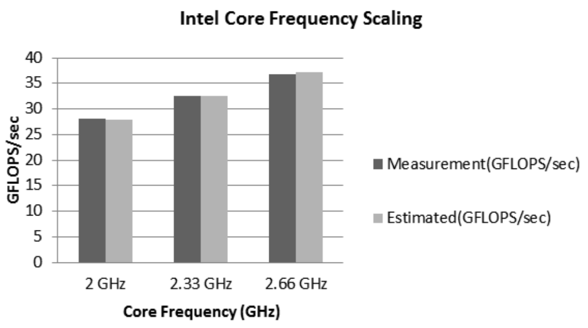

Figure 10. Core Frequency Scaling for LINPACK estimated and measured scores.

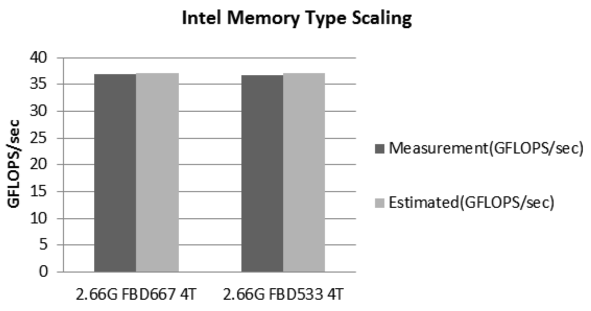

Figure 11. Memory type estimation and measured score for LINPACK.

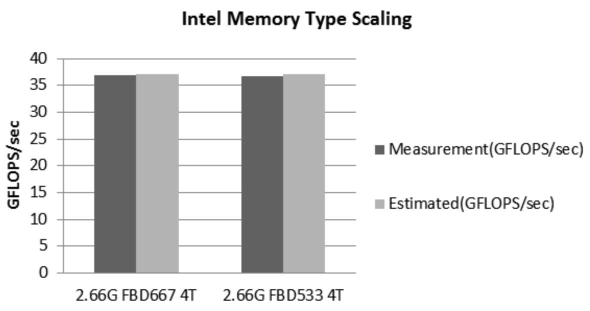

Figure 12. LINPACK Measured vs. Estimated for different IA processors.

5.2. SGEMM experimental results

For SGEMM, we show estimated data for SGEMM model against measured data. We compare the SGEMM model to actual measured data for same matrix size =8K using different Nvidia GTX graphics cards, and the error between measured and estimated performance for SGEMM was also <5% for all three cards as we scale with different number of core shaders while keeping core frequency fixed at 1126MHz as shown in Figure 13.

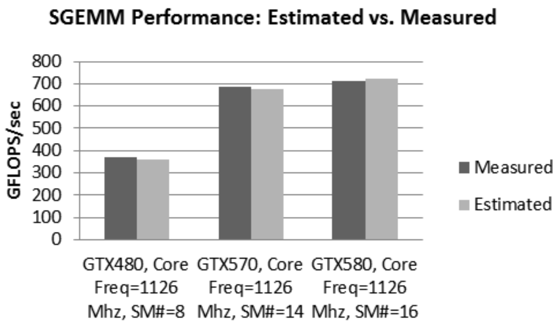

Figure 13. Measured vs. Estimated for SGEMM using different Nvidia GTX cards.

5.3. MONTECARLO performance experimental results

First, we verify MONTECARLO using the Nvidia Tesla 2050 [9] (CUDA cores=448, or $SM\#=448/32=14$) and NV GTX580 [15] (CUDA cores=512 or $SM\#=512/32=16$) graphics cards. The data results show an error of <5% between the estimated and measured data at different core frequencies and numbers of core shaders, shown in Figure 14. For both cards (Tesla 2050 and GTX580), the simulation path $N=256*1024$ and options number $M=2048$.

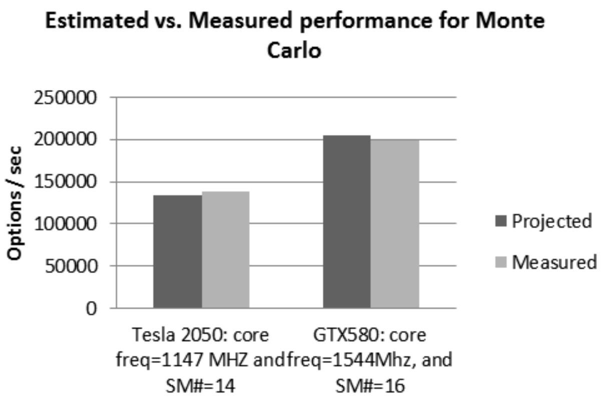

Figure 14. Estimated vs. Measured for Monte Carlo.

6. Performance per watt analysis and estimations

Any increase in computational performance should come at the expense of a power budget increase. It is important to understand the performance-per-watt for any benchmark running on a given processor. For the power analysis and estimation experiment, we use SGEMM benchmark running on different Nvidia GPUs. In general, power consumption for digital devices defined as

$$Power \;=\; k \times C \times V^2 \times f \,, \tag{23}$$

where k is application specific constant, C is the total switching capacitance of the processor, V is the input voltage which changes relative to frequency, and f is the core frequency. The dynamic power equation is a function of input voltage V and core frequency f, note the f and V are directly proportional. The power estimation equation is derived as a function of frequency fraction multiplied by difference in power:

$$Estimated\,Power \;=\; P_0 + (P_1 - P_0)\,\frac{f_1}{f} \,, \tag{24}$$

where *P1* is the measured execution power at frequency *f1* and *P0* is the non-scale power. We can write *P0* in terms of a second measurement *P2* at *f2*:

$$P_0 = \frac{P_2 f_2 - P_1 f_1}{f_2 - f_1} \,. \tag{25}$$

For this model to work, we have to setup a measured baseline, which consists of minimum two measured power data points at two different core frequencies. There is one assumption we need to take into account for establishing the measured baseline; the number of core shaders is fixed. For example, we cannot use the measured baseline for NV GTX570 with 448 core shaders to estimate power for NV GTX580 with 512 core shaders. Both the measured baseline and what we are estimating to must have the same number of core shaders. From Eq. (22) and Eq. (23), the power is a function of frequency, so if we need to change the number of core shaders, we need to establish a new measured power baseline for that number of core shaders and estimate for higher core frequencies from that measured baseline. The power measured is the total AC the system is using. We used AC power meter sampling rate of one power-reading sample per second making record keeping accurate and easy to analyze. We used Yokogawa WT210 digital calibrated power meter that measures total AC power usage of the system in watts every second while workload is running.

6.1. SGEMM Performance per watt

Before establishing a measured power baseline and estimating for power at higher frequencies, first we need to understand the sensitivity analysis for SGEMM with respect to core frequency and to matrix sizes. From SGEMM performance model, these two variables are the dominant factor for determining SGEMM performance. From Figure 15, we can conclude that the total AC power and performance score for SGEMM (GFLOPS/sec) do scale with respect to frequency and matrix size. However as matrix size increases, the return on performance start to diminish, which is expected. At small matrix sizes, the scaling for performance and power is larger compared to larger matrix sizes.
With the measured baseline, we can apply the power estimation regression method with respect to GPU core frequency for a given matrix size as shown in Figure 16.
To verify that the power curves in Figure 16 are within our expect error range of $<5\%$ between measured and estimated power, we selected two different estimated power curves for matrix sizes of 1000Bytes and 2000Bytes. Then we verify measured vs. estimate power for different core frequency range (550MHz- ~900MHz) using NV GTX480 as a baseline as shown in Figure 17.

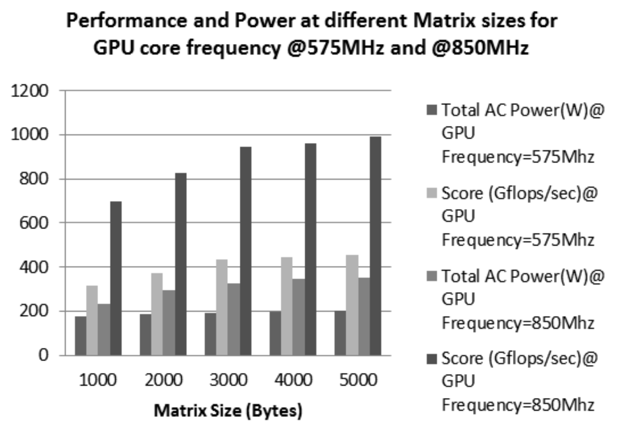

Figure 15. Measured SGEMM performance and power for GPU frequencies 575MHz and 850MHz at different matrix sizes using NV GTX480.

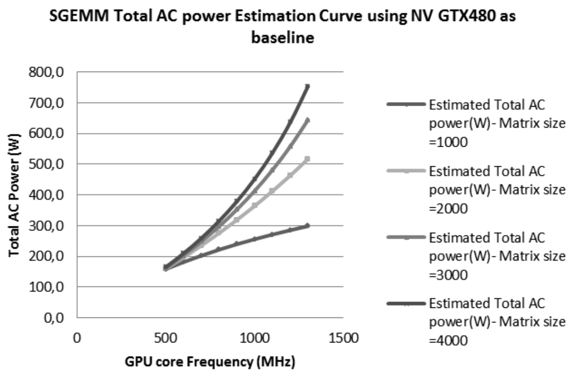

Figure 16. SGEMM Total AC power estimation curves for different matrix sizes at different GPU frequencies.

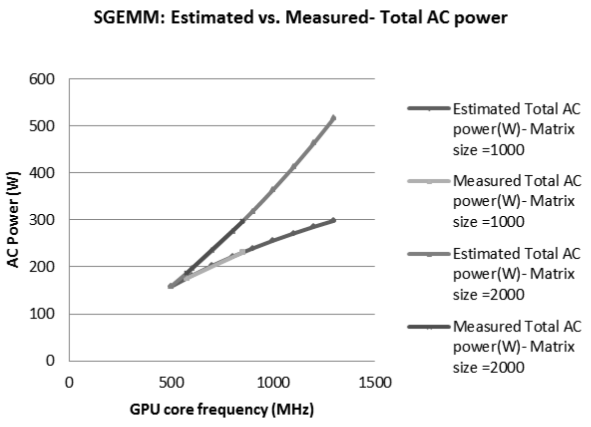

Figure 17. SGEMM estimated vs. measured total AC power for matrix size=1000Bytes and 2000Bytes using GTX480.

We conclude that power estimation compared to measured error is <2% for all tested cases within a given frequency range. We concluded that performance can achieve certain scaling at lower core frequencies, but as core frequency increases, the return on performance gain diminishes. In this section, we take one more step by combining both performance and power estimation models to estimate performance–per-watt for SGEMM, which is an important analysis for any workload. Performance per Watt is calculated by dividing performance by the average system AC power at a given core frequency. The objective is to achieve higher performance-per-watt as frequency increases, in other words, we do not want a workload to operate in a region were performance is flat and power is increasing as frequency increases resulting in a negative slope for performance-per-watt curve. In Figure 18, we show performance-per-watt estimation at higher GPU core frequencies for different SGEMM matrix sizes using NV GTX480 as a baseline.

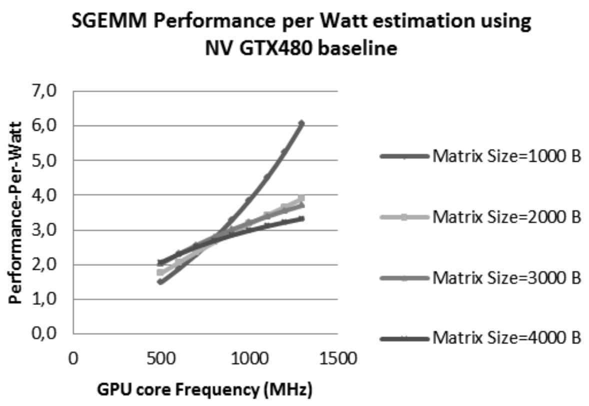

Figure 18. SGEMM estimated Performance-Per-Watt.

We noticed that, beyond core GPU frequency of ~800 MHz, for matrix size=1000Bytes, a much higher performance-per-watt is achieved as core frequency increases. This is because the matrix size is small and is not a limiting factor for performance. In addition, for this matrix size, power consumption does not increase much as frequency increases given its low computation needs. This explains why performance-per-watt is high for matrix size=1000Bytes. For other matrix sizes 2000Bytes, 3000Bytes, and 4000Bytes, beyond 800 MHz core frequency, the performance-per-watt starts to diminish because the matrix size will affect performance so it starts to level off with respect to core frequency while power is increasing as matrix size increases.

6.2. LINPACK Performance per watt

For LINPACK benchmark performance-per-watt analysis, we used Intel XEON dual processors @ 2.3 GHz each. We did this analysis based on number of cores and matrix size scaling, since we can change the number of cores in BIOS. The power we are measuring for this analysis is AC power associated with LINPACK benchmark. From measured data, we noticed that power does not scale very well with matrix sizes for LINPACK at different number of cores (4, 8 and 16) as shown in Figure 19.

Using the Xeon processor measured data; We verified power estimation data and compared it to measured data as shown in Figure 20.

We noticed that beyond 10 cores, the power (measured and estimated) increases exponentially. Now we combine both the power and performance estimation models for LINPACK to analyze the performance-per-watt while keeping the core frequency fixed at 2.3GHz, and changing the # of cores. We took the ratio performance/watt for a given number of cores as shown in Figure 21.

We noticed that as we increase the number of cores, we reach a peak for performance-per-watt at about 8 cores, any increase in number of cores beyond 10 cores, will result in a negative slope, in which power is increasing, while performance is not increasing proportionally which will result in a negative performance-per-watt slope. The optimum point to operate LINPACK benchmark is at the peak point of the curve which is roughly between 12 and 13 cores for all matrix sizes.

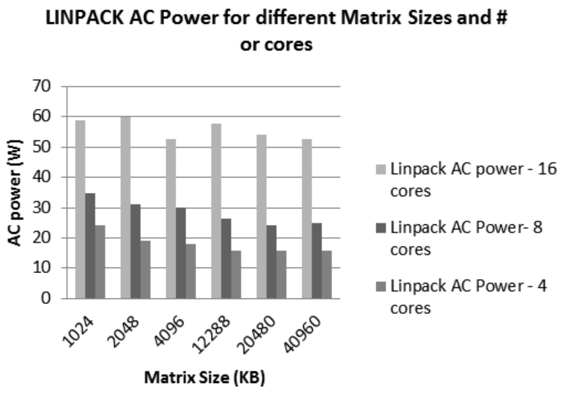

Figure 19. LINPACK Power for different matrix sizes and number of cores.

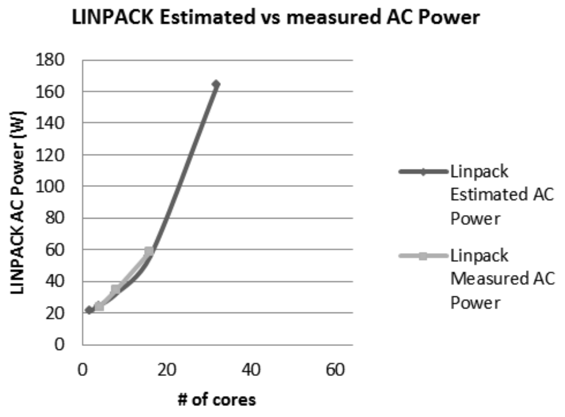

Figure 20. LINPACK Power, Measured vs. Estimated at different number of cores.

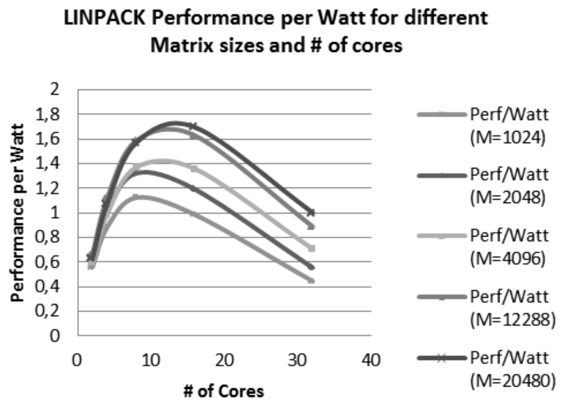

Figure 21. Performance per Watt for LINPACK at different matrix sizes.

7. Conclusion

In this paper, we analyzed LINPACK, SGEMM and MONTE CARLO benchmarks, which are combination of compute and memory bound workloads. We developed a performance estimation analytical model as a function of several processor architecture parameters with impact on performance. We verified the model by testing different processors configuration running a given benchmark and compared measured results with estimated results. We also derived a power estimation model for LINPACK to determine optimum operating point. All tested experiments for performance show deviation error between estimated and measured data of <5%. We used similar approach to develop an analytical performance and power estimation models for SGEMM. We proposed a performance-per-watt analytical estimation models for SGEMM and LINPACK, in which we can estimate performance and power to different processor configurations and application settings.
The performance-per-watt estimation model will enable researches to identify optimum operating point for these two benchmarks on different processor configurations. The performance and power estimation models presented in this paper are analytical based models that do not require trace-based simulation for a given workload. Our future research will focus on adding more HPC workloads and use the same modeling approach to estimate performance-per-watt for different processor configurations.

References

[1] AMD K10- http://www.xbitlabs.com/articles/cpu/display/amd-k10.html
[2] AMD Opteron-K8: http://www.cpu-world.com/CPUs/K8/index.html
[3] Baghsorkhi S., Delahaye M., Patel S., Gropp W., Hwu W., An adaptive performance modeling tool for GPU architectures, In: Proceedings of ACM PPOPP, 105-114, 2010
[4] Bhatai N., Alam S., Performance modeling of emerging HPC architectures, HPCMP Users Group Conference, June 2006
[5] BLAS: http://www.netlib.org/blas/
[6] Chau-Yi Chou, A semi-empirical model for maximal LINPACK performance predictions", 6th IEEE International Symposium on Cluster Computing and the Grid, 30 May 2006
[7] Goel et al., Portable, scalable, per-core power estimation for intelligent resource management, Int. Green Computing Conference, 2010
[8] Gustafon J., Todi R., Conventional Benchmarks as a sample of the Performance Spectrum, J. Super Comput., 13, 321-342, 1999
[9] Hennessy J.L., Patterson D.A, Computer Architecture: A Quantitative Approach (4th Ed., Morgan Kaufmann, 2007)
[10] http://www.nvidia.com/content/GTC/documents/SC09_Dongarra.pdf
[11] Isci et al., Live, runtime phase monitoring and prediction on real systems with application to dynamic power management, Int. Symposium on Microarchitecture, 2006
[12] Jens S., Performance Prediction on Benchmak Programs for Massively parallel Architectures, 10th Internaltion conference of High-Performance Computer (HPCS), June 1996
[13] Kamil S., Power efficiency for high performance computing, IEEE International Symposium on Parallel and Distributed processing, June 2008
[14] LINPACK: http://www.top500.org/project/linpack/
[15] Livny M., Basney J., Raman R., Tannenbaum T., Mechanisms for High Throughput Computing, SPEEDUP J., 1997
[16] Nvidia GTX460 http://www.nvidia.com/object/product-geforce-gtx-460-us.html
[17] Nvidia GTX570 http://www.nvidia.com/object/product-geforce-gtx-570-us.html
[18] Nvidia GTX580 http://www.nvidia.com/object/product-geforce-gtx-580-us.html
[19] Nvidia nTune utility http://www.nvidia.com/object/ntune_2.00.23.html
[20] Nvidia Tesla C2070 http://www.nvidia.com/object/personal-supercomputing.html
[21] Rafael Saavedra H., Smith A.J., Analysis of benchmark characteristics and benchmark performance prediction, ACM Transactions on Comput. Syst., 14, 1996

[22] Rohr D. et al., Multi-GPU DGEMM and High Performance LINPACK on Highly Energy-Efficient Clusters, IEEE Micro, September 2011

[23] Ryoo S., Rodrigues C.I, Baghsorkhi S.S., Stone S.S., Kirk D.B., Hwu W.W., Optimization Principles and Application Performance Evaluation of a Multithreaded GPU using CUDA, In: Proceedings of the 13th ACM SIGPLAN, 73C82, ACM Press, 2008

[24] SGEMM: http://keeneland.gatech.edu/software/sgemm_tutorial

[25] Snavely A. et al., A Framework for Application Performance Modeling and Prediction, ACM/IEEE Supercomputing Conference, 2002

[26] Volkov V., Demmel J.W., Benchmarking GPUs to Tune Dense Linear Algebra SC08, November 2008