Instrumenting Linear Algebra Energy Consumption via On-chip Energy Counters

James Demmel
Andrew Gearhart

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-168
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-168.html

June 23, 2012

Instrumenting linear algebra energy consumption via on-chip energy counters

James Demmel
UC Berkeley
demmel@cs.berkeley.edu

Andrew Gearhart
UC Berkeley
agearh@cs.berkeley.edu

Abstract

Despite the move toward chip multiprocessing in the mid-2000s, the problem of machine energy consumption is still a prevalent and growing problem within the computing sector. To evaluate energy consumption at the application level, researchers were previously limited to specialized external instrumentation, modeling or simulation. Thankfully, new microprocessor designs have now exposed a handful of hardware counters that are able to measure energy directly, avoiding many limitations of previous techniques. This work details the capability of these counters, supports their accuracy via other measurement hardware, and explores the energy consumption of common dense and sparse linear algebra routines for various problem sizes and core frequencies.

Keywords: on-chip counters, energy and power measurement, linear algebra

1 Introduction

A number of years ago, researchers noted [24, 5, 16] that the industry reliance on deeper pipelines and higher clock frequencies for increased performance was inherently limited by the polynomial relationship between power and frequency/voltage. This problem was regarded as the ”power wall”, and motivated the transition toward multicore processor designs. By increasing the number of cores on die, peak floating point performance scales nearly linearly with increasing core count. This allows for continued scaling of peak performance, but does not address the growing gap between memory capability and floating point capability. Furthermore, large data centers and embedded devices are still bound by the available power envelope. Due to these reasons, managing application power and energy still represent areas of critical research interest. In particular, two problems are prevalent:

- **Improving energy efficiency of an application's execution to reduce the overall energy consumption of a datacenter or portable device.** From the datacenter perspective, a significant amount of expense is required to provide energy for compute nodes and cooling [17]. By increasing the energy efficiency of algorithms, both costs can be reduced. On the other hand, energy efficiency serves to increase the battery life of handheld devices.
- **Managing power consumption within a power-constrained environment.** Datacenter power is limited by the building infrastructure and processor workloads must be accurately characterized so as to provide insight into the thermal requirements of a system.

Toward addressing these above problems, accurate and efficient methods of measuring machine energy consumption are required to aid researchers and allow for feedback to software. Previous methods for measuring power and energy used external instrumentation and counters [15, 3, 20, 19, 1], simulation [6, 14], modeling [18, 8, 28] or emulation [2]. While each of these techniques has advantages and disadvantages, researchers were unable to directly access energy information during actual application execution on most machines (with the exception of handheld and embedded devices). During the past year, Intel's successors the to Nehalem microarchitecture, Sandy Bridge, have emerged with on-chip counters that allow for CPU and DRAM energy to be queried entirely through software. The latter capability (measuring DRAM energy) represents a critical feature to researchers, as on-node memory traffic is expected to become the largest component of energy consumption for many algorithms [21, 12]. At time of writing, no on-chip energy counting capability has been made available to the user from other major manufacturers. Thus, this work will exclusively utilize Intel's Running Average Power Limit (RAPL) interfaces for data collection.

This work explores the new energy counting capabilities of Intel processors and uses these counters to examine the energy efficiency of various linear algebra algorithms and implementations. While these counters provide the ability to set power limits for various chip domains (probably useful for the above mentioned second problem), this work only reads energy state registers as opposed to actively setting limits. First, we demonstrate concordance between measurements of wall power and energy counter readings for compute and communication-intensive microbenchmarks (a cpu-intensive loop and STREAM [22], respectively). This accomplished, we demonstrate that energy counters allow for easy experimental verification of energy savings due to frequency scaling and optimized thread allocation to logical cores. Furthermore, we demonstrate that the energy required to compute sparse matrix-vector multiplication (SPMV) is strongly dependent on the machine's Last Level Cache (LLC) size, number of nonzero entries and the sparsity pattern of the problem. Interestingly, SPMV is able to draw more DRAM power than STREAM with randomized memory accesses and the DRAM power on the experimental machine represents a small fraction of wall power. Finally, we discuss implications for future research and conclude.

2 Intel's Running Average Power Limit (RAPL) interfaces

2.1 Overview

While exploring the current capabilities for on-chip energy counters, Intel model-specific (or machine-specific) registers (MSRs) are implemented within the x86 and x64-64 instruction sets as means for processes to access and modify parameters related to cpu execution [11]. A large number of these registers exist, and most are beyond the scope of this work. However, a handful of MSRs are allocated for platform specific power management within the Sandy Bridge and successor microarchitectures and allow access to energy measurement and enforcement of power limits. In particular, Intel refers to these registers as the Running Average Power Limit (RAPL) interfaces. According to Section 14.7 of [11], the RAPL interfaces ”expose multiple domains of power rationing within each processor socket” and power management for multiple sockets (”processor complexes”) must be handled individually. The RAPL interfaces currently define MSRs for access to 4 domains:

- **Package (PKG):** entire socket
- **Power Plane 0 (PP0):** cores
- **Power Plane 1 (PP1):** uncore*
- **DRAM:** sum of DIMM powers for the socket

The Intel documentation [11] states that client platforms have access to {PKG,PP0,PP1} while server platforms (code name Jaketown) may access {PKG,PP0,DRAM}. From the above domain definitions, one expects that $energy(PP0) + energy(PP1) = energy(PKG)$. The asterisk on the entry for PP1 above indicates that on client Sandy Bridge platforms PP1 measures the energy of the on-chip graphics processor, as opposed to the entire uncore. On these machines $energy(PP0) + energy(PP1) <= energy(PKG)$ and $energy(PKG) - (energy(PP0) + energy(PP1)) = energy(uncore)$. These equations have been confirmed by Intel representatives [25].

2.2 Capabilities

[11] lists the capabilities of the RAPL interfaces to monitor and control system power and energy:

- **Power Limit** - MSR interfaces to specify power limit and time window
- **Energy Status** - Power metering interface providing energy consumption information
- **Perf Status (certain machines)** - Interface providing information on the performance effect due to power limits.
- **Power Info (certain machines)** - Interface providing information on the range of parameters for a given domain, minimum power, maximum power, etc.
- **Policy (certain machines)** - 4-bit priority information which is a hint to hardware for the dividing power budget between PP0/PP1 within PKG.

These capabilities are available in various degrees to each RAPL domain available on a given machine. As mentioned earlier, results presented within this work only utilize the Energy Status component of the RAPL interfaces and thus access the MSRs in a read-only manner.

2.3 Access

To enable user level access to the RAPL interfaces on Linux, the ”msr” kernel module must be built and loaded. Once loaded, the module creates the /dev/cpu/*/msr character device for each logical processor. These character devices by default require superuser permissions to access, and for the purposes of this study were modified to allow user access. Once accessed as a normal file, a specific MSR can be read or written according to offsets defined within the asm/msr.h header file. Insight for the code utilized in this work was obtained via [26].

2.4 Limitations

Currently, the small set of RAPL interface MSRs are limited in functionality in that they represent metrics that are of socket scope. Thus, individual cores cannot be measured and PP0 represents the sum of all core energies. Similarly, DRAM and uncore energy data do not distinguish between various memory channels or uncore devices.

While the power limiting capabilities of the RAPL interface are not considered directly within this document, at time of writing the ability to set per-core power limits is not available. Users are limited to setting limits for the RAPL domains described above.

Table 1: Machines utilized in experiments

Machine	Model	Physical Cores	Max freq	Min freq	Memory	Hyperthreading?	Turbo Boost?
Sandy Bridge client	Core(TM) i7-2600	4	3.4Ghz	1.6Ghz	32Gb	Y	N
Sandy Bridge server	Core(TM) i7-3960X	6	3.3Ghz	1.2Ghz	16Gb	Y	N

3 Hardware and Experimental Setup

There were 2 machines used in this study, with key parameters as shown in Table 1. All parallel codes were run with the number of threads equal to the number of physical (as opposed to logical) cores and caches were warmed up by running the test function once prior to beginning data collection. Both machines in Table 1 were running Ubuntu Linux version 10.04 and code was compiled with the Intel compiler version 12.1.0. Tuned linear algebra routines were called from Intel's Math Kernel Library version 10.3. To reduce potential noise in data collection, the Turbo Boost [10] feature of these machines was disabled. The effect of turbo upon energy counter behavior is currently unclear, but certainly warrants future investigation.

Sandy Bridge machines have a single clock domain that encompasses all cores, the on-chip communication network and last-level cache. Thus, frequency scaling data is obtained by scaling all cores simultaneously to the same clock rate.

4 Microbenchmarks and relationship to wall power

Before presenting results related to linear algebra routines, it is useful to sanity-check the RAPL counters by comparing average energy consumption to the results measured by a wall power meter (Brand Electronics Model 21-1850-CI). These experiments were performed on the Sandy Bridge client platform running 4 threads (one thread per physical core). Observing the concordance between different instrumentation methods also allows for the relationship between chip power draw and total system power to be observed. Wall power traces were obtained at a granularity of 1 second, and were recorded for several seconds before and after the run period of the benchmark. Note that one should not expect Wall power to correlate exactly with chip package measurements as the wall power measurement also includes machine components such as DRAM DIMMs, fans and power supply losses.

4.1 STREAM benchmark

The STREAM benchmark [22] is a commonly utilized microbenchmark designed to measure sustained memory bandwidth on multicore nodes. It is an operation completely dominated by data operations, and as such should be less dependent upon core frequency for performance. Indeed, the runtime of STREAM at 1.6Ghz is similar to that at 3.4Ghz at about 10.1 seconds. In this work, STREAM's INIT test (which initializes an array to a constant value) was measured for the purposes of energy. We note that a significant amount of extra energy is required to run the benchmark at the higher core frequency. By dividing the energy counter data by the runtime, we can obtain a value for average power over the PKG domain. These values for core frequencies of 3.4 and 1.6Ghz are shown in Table 2, and differ by over 20W.

By correlating wall power data with timestamps, we can calculate the average wall power increase over idle during a data run. To calculate idle, we performed no work on the machine[1] and collected wall power data at 1 second intervals for 2 minutes at the target core frequency. Idle power was then set to be the average of these values and was approximately 76W at both core frequencies under consideration. Figure 1(a) shows annotated wall power traces for the Sandy Bridge client machine while running STREAM's INIT test. The meaning of the plot labels is as follows:

- START: test program start
- INIT(3.4): start of STREAM INIT test with core frequency 3.4Ghz
- END: end of INIT test

[1] No other users were present on the machine, and only standard system processes were running.

Table 2: Comparison of average wall and PKG powers for LOOP/STREAM

	Freq (Ghz)	Wall Total (W)	Idle	Wall-Idle	PKG
STREAM	3.4	125.57	76.48	49.09	42.23
	1.6	98.66	75.9	22.76	16.58
LOOP	3.4	113.75	76.48	37.27	41.65
	1.6	89.65	75.9	13.75	16.07

The program activity between START and INIT is dominated by memory allocation and initialization, and is thus excluded. We calculated the average total wall power based upon the values between the INIT and END labels, as is shown in the ”Wall Total (W)” column of Table 2. Once the average wall power for the entire machine was calculated, the idle power (”Idle” column of Table 2) was subtracted in an attempt to capture the dynamic power utilized during the benchmark run (”Wall-Idle” column of Table 2). For STREAM, the counter-derived average power (42.23W) for the processor die (PKG) differed from the wall power average (49.09W) by approximately 7W. The authors suspect that most of the discrepancy in this value is due to dynamic DRAM DIMM power, which is not measured via the PKG counter. The Sandy Bridge client machine has only 4 DDR3 DIMMS of DRAM, suggesting that the DIMM power contribution to wall power at high utilization will be relatively small (hence the difference of only a few Watts between the two averages). On servers with many DIMMs, one would expect the DRAM power to be a significant portion of overall machine power.

4.2 Cpu-intensive loop

While STREAM attempts to saturate memory bandwidth, Algorithm 1 stresses the integer computation capability of the processor.

Algorithm 1 LOOP benchmark

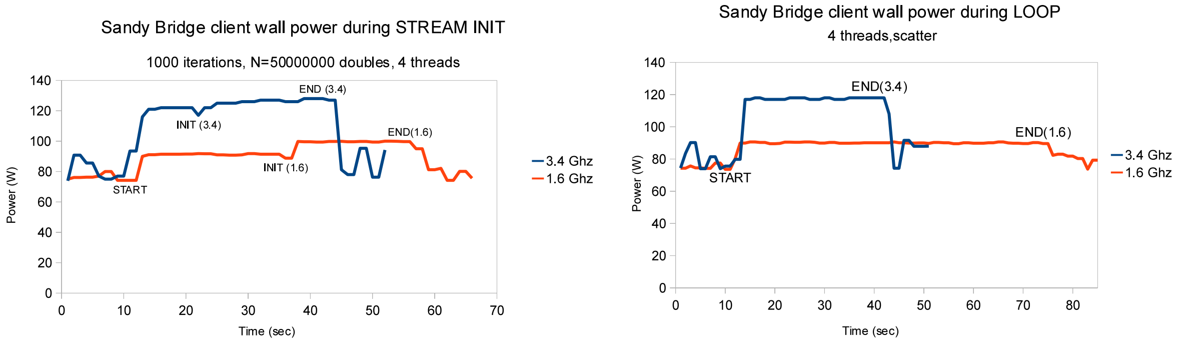
```
Require: INPUT x, outer, inner
#pragma omp parallel for
for $i = 0$ to $outer$ do
    for $j = 0$ to $inner$ do
        $x = XOR(x, x + (i\&j))$
    end for
end for
return x
```

This micro benchmark is based upon the code utilized by Buchert in his work on processor performance emulation [7], but with the addition of an OpenMP [4] pragma for parallelism. Similarly to STREAM, a significant amount of additional energy is required to run at the highest clock frequency (see Table 2). In the case of LOOP there is little initialization overhead for the benchmark, so the START label on the wall power trace (Figure 1(b)) indicates the beginning of the actual test (equivalent to the INIT label in the case of STREAM). To calculate average total wall power, the values between START and END were considered. In this situation, we see in Table 2 that the average power obtained via the wall power meter (37.27W) is several Watts less than that obtained by the energy counter for the die. The reason for this effect is currently unknown, as one would expect the wall power to be strictly higher than the on-die counter average.

5 Dense linear algebra

Computation and Communication-bound algorithms

As an initial step toward analyzing more practical algorithms, we consider tuned implementations of dense matrix-matrix (GEMM) and matrix-vector multiplication (GEMV) via Intel’s Math Kernel Library (MKL) 10.3. These cases are useful in that matrix-matrix multiplication is a common example of a compute bound $O(n^3)$ algorithm, while



(a) STREAM INIT

(b) LOOP

Figure 1: Wall power traces for microbenchmarks

Table 3: Comparison of average wall and PKG powers for DGEMM/DGEMV

	Freq (Ghz)	Wall Total (W)	Idle	Wall-Idle	PKG
DGEMM	3.4Ghz	155.17	76.48	78.68	81.63
	1.6Ghz	100.4	75.9	24.51	29
DGEMV	3.4Ghz	126.31	76.48	49.83	49.08
	1.6Ghz	101.6	75.9	25.7	22.48

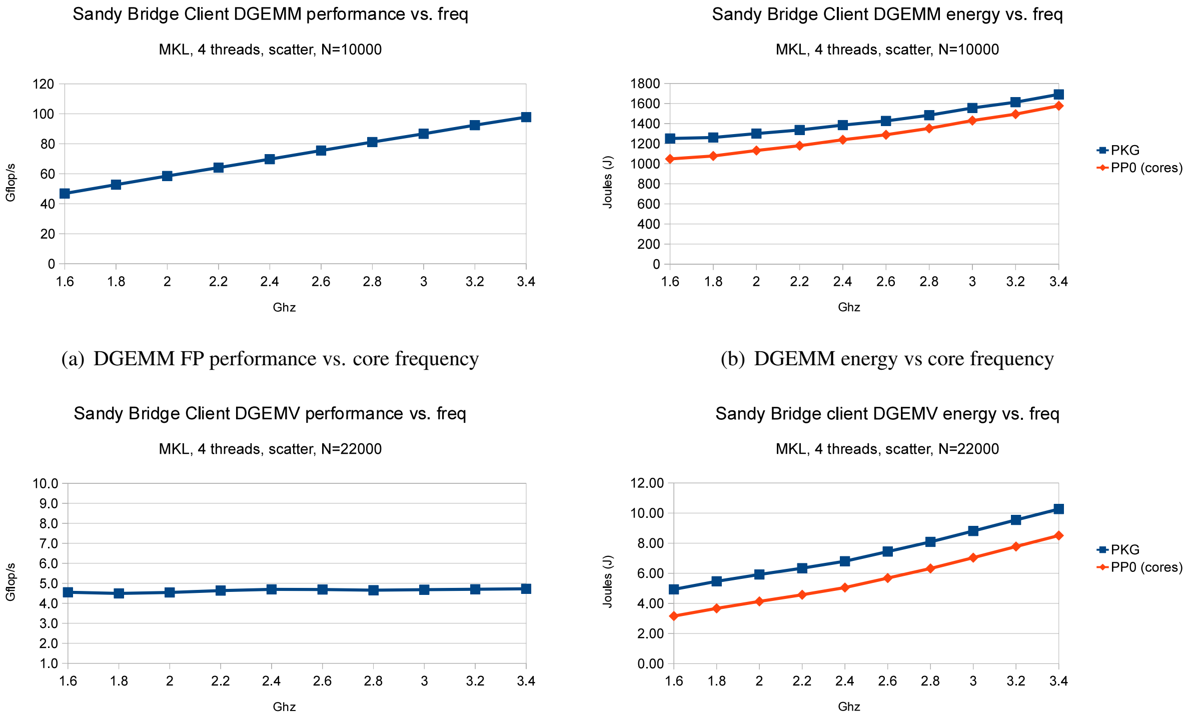
the matrix-vector multiplication runtime is dominated by the cost of moving data into fast memory. As such, one would expect the performance of GEMM to scale almost linearly and the performance of GEMV to stagnate with core frequency. The balance between compute and communication-boundedness of an algorithm is dependent on machine, algorithm and implementation factors, and is a key consideration when attempting to improve code performance. The interplay between these factors is elaborated upon with the Roofline model, presented in [29].

In the extremely computationally dominated GEMM, the typical strategy is to finish the problem as fast as possible and then allow the processor to do other useful work. This strategy is commonly known as Race to Halt (RtH). While our results do show a strong relationship between core frequency and floating point performance for GEMM (Figure 2(a)), it is still more energy efficient to complete the problem at a lower clock frequency (Figure 2(b)). On the other hand, the communication-dominated GEMV algorithm does not improve in floating point performance with higher clock rate as available DRAM bandwidth limits the amount of computation that can be performed rapidly (Figure 2(c)). Higher clock frequencies are even less efficient in this case (Figure 2(d)), making execution at a slower rate a clear choice to obtain efficiency without reducing runtime.

To correlate the on-chip energy counters, we also considered data from a wall power meter. The same idle power results were used as with the LOOP and STREAM benchmarks, and wall power traces (Figure 3) were annotated in a similar fashion to STREAM. Table 3 shows that the counter and wall meter average power meters only differ by several Watts. Interestingly, the compute-bound DGEMM differs more at 1.6Ghz while the communication-bound DGEMV values are nearly exactly identical at 3.4Ghz. This undermines the hypothesis of memory power consumption being a significant contributor to accuracy differences, and will have to be investigated further on a machine with a DRAM counter and a large number of DRAM DIMMs (so that system memory is a significant fraction of total wall power).

5.1 Saving energy via logical cores

In addition to the RAPL interfaces, the Sandy Bridge machines utilized for this study are capable of simultaneous multithreading (SMT). SMT allows multiple threads to utilize on-core hardware so as to improve resource efficiency in certain cases. Intel's implementation of SMT is called Hyper-Threading Technology [9], and manifests itself by allowing two threads to run on each physical core. For these machines, we can choose at runtime between two types of thread layouts (affinity schemes): scatter and compact. As shown in Figure 4, a scatter layout favors placing threads on independent physical cores while the compact layout tends to place two threads per physical core if possible.



(a) DGEMM FP performance vs. core frequency

(b) DGEMM energy vs core frequency

(c) DGEMV FP performance vs. core frequency

(d) DGEMV energy vs. core frequency

Figure 2: Energy consumption and floating point performance for DGEMM/DGEMV

In practice, SMT typically works well when both threads on the core are dominated by communication and can yield the core to each other often. In this situation, SMT of two threads on one physical core can approach the performance of two physical cores. Figure 5 shows the effect of thread affinity upon performance and energy for the MKL 10.3 GEMM and GEMV. This figure does not include energy data from the PP0 domain, as results were quite similar to PKG.

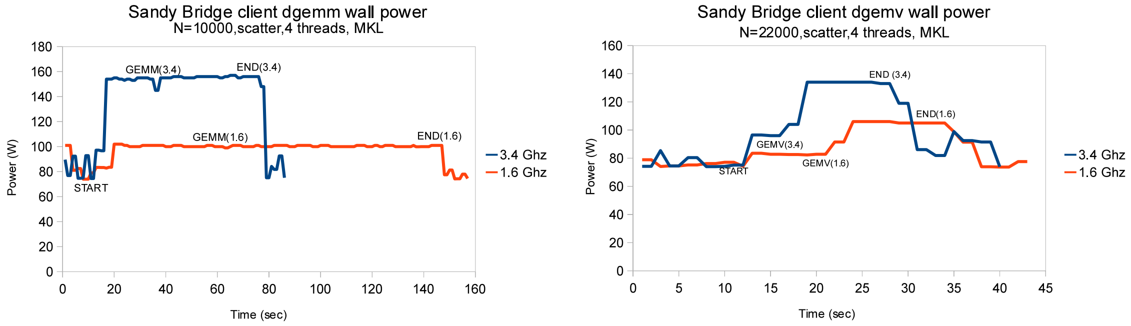
In the case of GEMM, computation dominates and utilizing a compact scheme results in threads competing for floating-point unit resources. This significantly reduces floating point performance over a scatter layout, and as such requires more energy to complete (as the code requires more execution time before the machine is able to idle). These results are shown in Subfigures 5(a) and 5(b). On the other hand, the communication-bound GEMV achieves a slightly higher level of floating point performance for higher core frequencies with a compact layout (Figure 5(c)). The compact layout also utilizes less energy in this case (Figure 5(d)), resulting in a lower energy expenditure for slightly higher performance.

In general, however, the tradeoff between thread affinity scheme, energy consumption and performance is a complex function of hardware configuration, algorithm choice and implementation efficiency. In the presented case of GEMV on a Sandy Bridge client platform, two physical cores are able to saturate the entire available memory bandwidth. As such, performance of the matrix-vector multiplication does not reduce when using two cores. On other machines, all further cores or multiple sockets would be required to achieve further bandwidth for an algorithm so constrained.

6 Sparse linear algebra

6.1 Background

In the previous section, we assume that each entry of the input matrices are nonzero entries. In many situations, the formulation of a scientific problem results in input matrices that have very few nonzero entries relative to the



(a) DGEMM

(b) DGEMV

Figure 3: Wall power traces for DGEMM/DGEMV

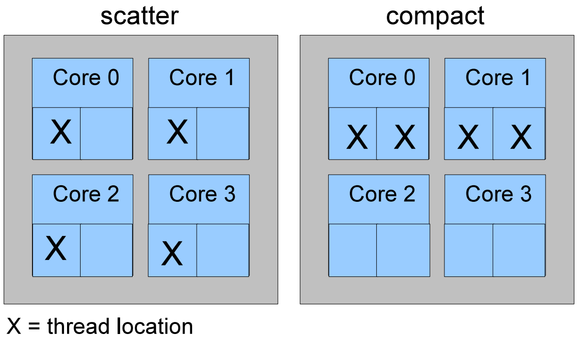


Figure 4: Assigning 4 threads to a 4 core CPU with SMT

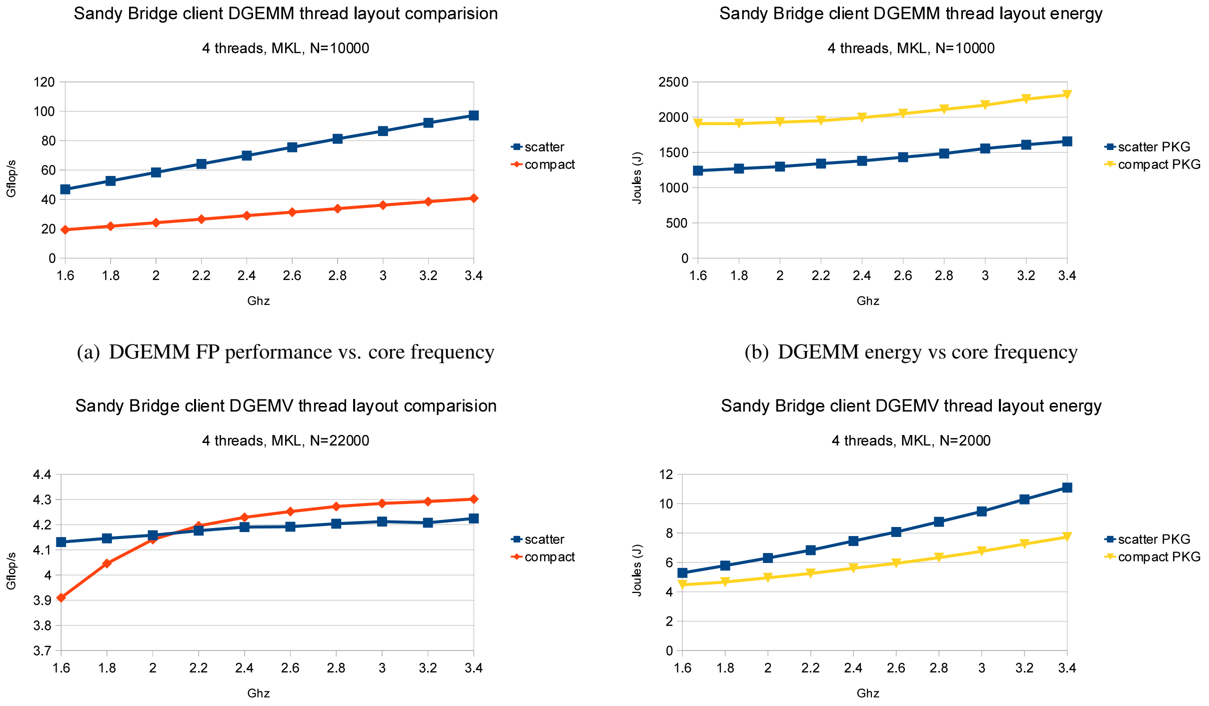
dimensions of the matrix. Such matrices are called ”sparse”, and allow algorithms to calculate the correct answer with many fewer floating point operations than if the matrices were considered to be dense. Furthermore, many problems require iterated solvers that perform repeated sparse matrix-vector multiplication (SPMV) operations in hope of converging to a numerical solution. In order to take advantage of the fewer flops required to calculate a given SPMV, the sparse matrix must be stored in a data structure that allows the algorithm to reference the location of nonzeros efficiently. While many sparse matrix storage schemes have been developed (see [27]), the experiments performed for this work exclusively utilized matrices stored in Compressed Sparse Row (CSR) format. Figure 6 outlines the CSR storage scheme.

While CSR avoids representing zero values, and compresses n^2 values to $2NNZ + n + 1$ where NNZ is the number of non-zero entries, the format adds a level of memory indirection when attempting to compute a matrix-vector multiplication. This indirection can be seen in the access of the b vector within Algorithm 2. As such, the memory access pattern of CSR SPMV is somewhat dependent on the pattern of nonzeros within the input matrix. This often limits load/store locality and results in a significant performance penalty due to bad caching behavior.

To generate SPMV data for this report, we utilized several hundred sparse matrices with in-memory data structure sizes ranging from several hundred bytes to several gigabytes. Matrices were stored in the Matrix Market format [23] and downloaded from the University of Florida sparse matrix collection [13]. All sparse problems used in this study were symmetric.

6.2 Observations

Figure 7 show MKL DSPMV energy and floating point performance data for the Sandy Bridge client and Sandy Bridge server machines. In these plots, a single point on the x-axis represents a single sparse problem. Problems were sorted by the number of nonzero values, so that matrices to left side of the x-axis are very small (several hundred bytes) while matrices to the right of the x-axis are large (several Gigabytes). The sample set was biased toward small matrices,



(a) DGEMM FP performance vs. core frequency

(b) DGEMM energy vs core frequency

(c) DGEMV FP performance vs. core frequency

(d) DGEMV energy vs. core frequency

Figure 5: DGEMM/DGEMV thread assignment comparison

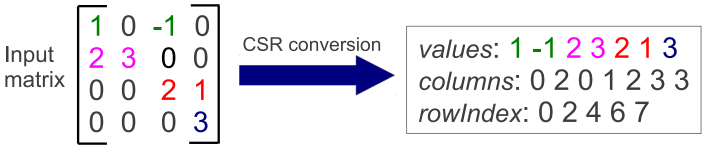


Figure 6: Converting a sparse matrix to CSR format

so the size increase along the x-axis is nonlinear. On each plot in Figure 7, the percent change between PKG energy between the maximum and minimum clock frequencies of the machine is plotted in blue. Similarly, percent change in GFLOP/S between the two frequencies is plotted in orange. On both machines, a distinctive transition can be seen toward the center of each plot where floating point performance and energy becomes considerably more variable. On each machine, this transition appears to occur when the in-memory size of the problem is 1-2Mb larger than the size of the on-chip L3 cache[2]. If the problem fits into cache, moving to a lower clock frequency results in a relatively constant tradeoff between performance degradation and energy improvement. On the other hand, problems larger than cache show a significant amount of variability even between similarly-sized problems. On the client machine, a large number of matrices show a favorable energy/gflop/s tradeoff with energy savings outweighing performance penalty at the lower frequency. On the other hand, the server data clearly shows that one will almost always lose more floating point performance than energy by running at the lower frequency.

In an attempt to explore the performance variability mentioned above, we run the MKL DSPMV on a band matrix (stored in CSR format) of size N=50000 on the Sandy Bridge server platform (Figure 8). Along the x-axis, we scale the bandwidth of the matrix from completely diagonal (no diagonals) to a bandwidth of 200. This results in a series of matrices that range in size from approximately 1 Mb to 1Gb. Again, we notice the transition as the problem falls out of L3 (toward the left side of the x-axis in this case, as opposed to the center) and note that the performance variability has been nearly eliminated. This supports the hypothesis that sparsity pattern reflects approximately 10-20% of the

[2] L3 sizes: Sandy Bridge client 8 Mb and Sandy Bridge server 15 Mb.

Algorithm 2 Naive sparse matrix-vector multiplication in CSR format

```
Require: matrix A in CSR form: A = {values,columns,rowIndex}, vectors b and c
for $i = 0$ to $n$ do
    $c_i = 0$
    $currIndex = rowIndex[i]$,$nextIndex = rowIndex[i + 1]$
    for $j = 0$ to $nextIndex - currIndex$ do
        $offset = currIndex + j$
        $c_i+ = values[offset] * b[columns[offset]]$
    end for
    $c[i] = c_i$
end for
```

floating point performance and energy consumption difference between the two targeted frequencies. Future work may the consider the frequency tuning space represented by this variation for large problems.

Finally, Figure 9 displays data collected from the DRAM energy counter on the Sandy Bridge server machine and compares the average DRAM power consumption at 1.2Ghz (orange) and at 3.3Ghz (blue). This machine was installed with 4 DDR3 non-ECC DIMMS running on 4 channels at 1600Mhz. Memory clock frequency was not varied in the course of this experiment. Unsurprisingly, the counter data shows a significant increase in power once the problem falls out of the L3 cache and memory is accessed more often. When the problem fits in L3 the DRAM appears to sit in a low-power state and only consumes a fraction of a Watt. On the other hand, larger problems result in a power draw of about 5.5W at 1.2 Ghz and over 8W at 3.3Ghz.

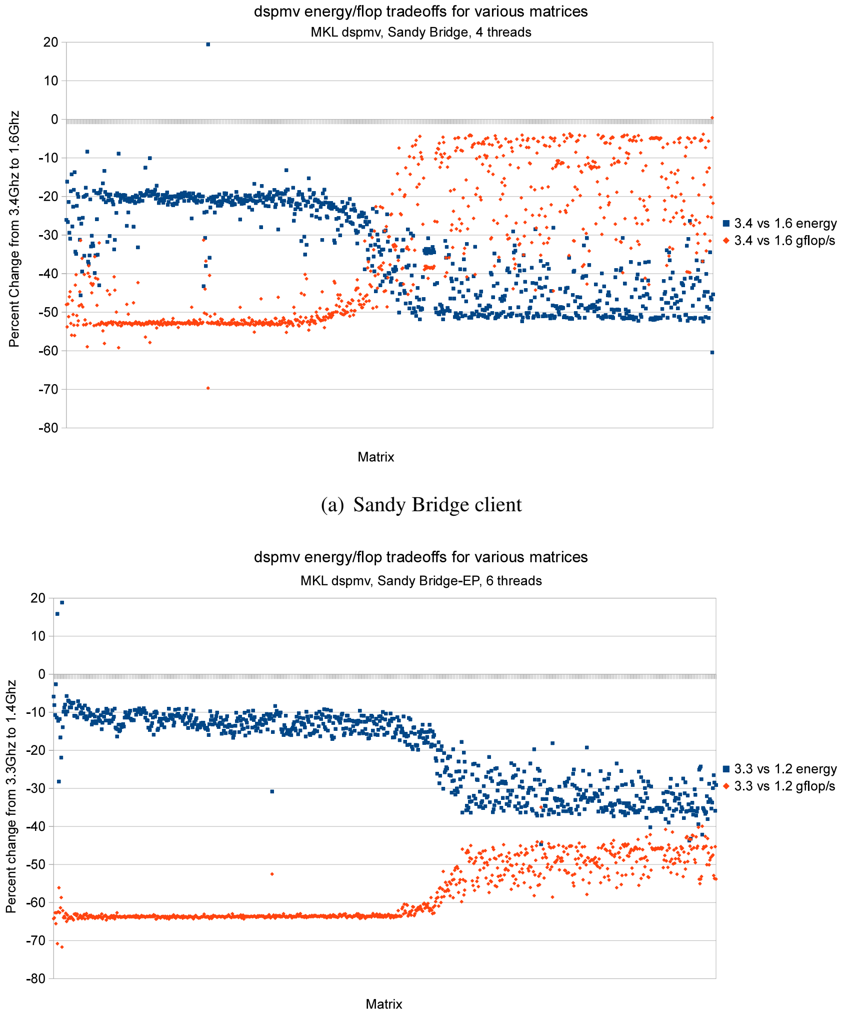
It is to be noted that the reported power consumption for the DRAM is not an extremely significant fraction of the wall power. Thus, we were unable to utilize the wall power meter to confirm the results reported in Figure 9 due to system noise varying the wall power by a handful of Watts. We hope to explore the accuracy of the DRAM energy counter in greater detail on a machine with a larger number of installed DIMMs in the near future.

7 Conclusions

This work provides an initial exploration of Intel's on-chip energy counters, and attempts to show some evidence for accuracy based upon measurements from a wall power meter. Furthermore, the work notes energy and floating point performance trends between common communication and compute-bound linear algebra routines and also tests a large number of sparse matrix problems. It was found that (perhaps unsurprisingly) floating point performance saturates at a low core frequency for communication-bound algorithms, resulting in a significant energy motivation to run these codes as the minimal possible frequency. This benefit even extended to using a lower number of physical cores on the chip in combination with low frequency.

Regarding the sparse matrix-vector multiplication data, we found that energy efficiency and floating point performance varies significantly (10-20%) with sparsity pattern once the problem is larger than the last level cache. This variation is nearly eliminated in problems that fit in cache (except for extremely small problems on the order of hundreds of bytes). Finally, we presented initial results obtained from the DRAM energy counter on the Sandy Bridge server platform. We found that for problems that fit in cache, the DRAM consumes a very small amount of average power (.5W). On the other hand, larger problems showed a large amount of variation in DRAM power and also a significant power difference between the minimum and maximum core frequencies (about 5.5W vs. 8.2W). We could not verify the DRAM counter data with a wall power meter due to the small amount of memory power consumption on this machine.

In the future, we hope to further evaluate the accuracy of the on-chip energy counters with various algorithms and under differing levels of load. Also, we wish to verify the DRAM counter results on a machine where the dynamic DRAM power is a significant portion of total wall power. Several of the results presented in this work suggest an auto tuning space for core frequency and sleep states. We hope to explore this in the near future, with perhaps an application to heterogeneous processing environments.



(a) Sandy Bridge client

(b) Sandy Bridge Server

Figure 7: MKL DSPMV performance and energy tradeoffs

References

[1] Frank Bellosa. The Benefits of Event Driven Energy Accounting in Power-Sensitive Systems. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, EW 9, pages 37–42, New York, NY, USA, 2000. ACM.

[2] Abhishek Bhattacharjee, Gilberto Contreras, and Margaret Martonosi. Full-system chip multiprocessor power evaluations using FPGA-based emulation. In *Proceedings of the 13th international symposium on Low power electronics and design*, ISLPED '08, pages 335–340, New York, NY, USA, 2008. ACM.

[3] W. Lloyd Bircher and Lizy K. John. Analysis of Dynamic Power Management on Multi-core Processors. In *Proceedings of the 22nd annual international conference on Supercomputing*, ICS '08, pages 327–338, New York, NY, USA, 2008. ACM.

[4] OpenMP Architecture Review Board. OpenMP Specifications. `http://openmp.org/wp/openmp-specifications/`.

[5] Shekhar Borkar. Getting Gigascale Chips: Challenges and Opportunities in Continuing Moore's Law. *ACM Queue*, 1(7):26–33, 2003.

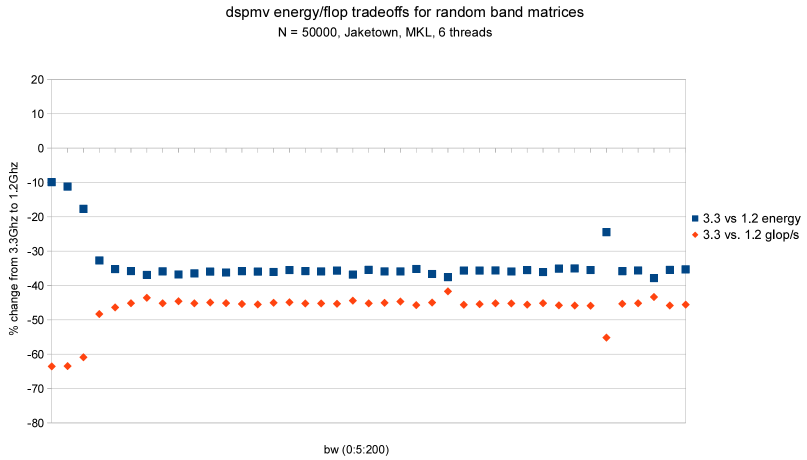


Figure 8: Sandy-Bridge Server energy/performance change for MKL DSPMV

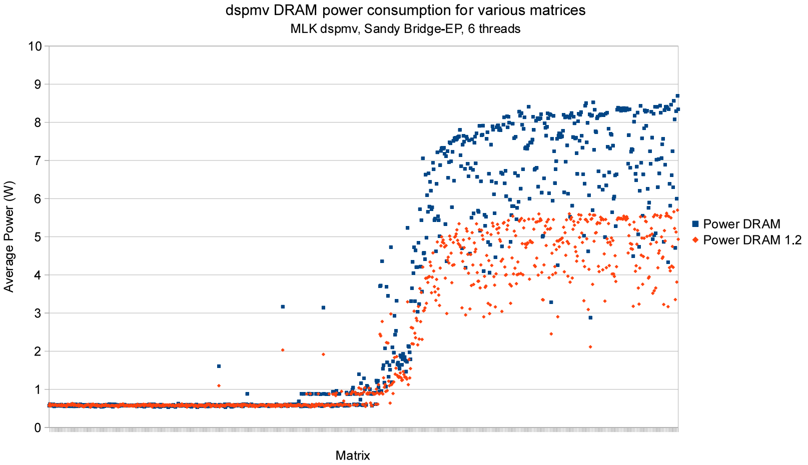


Figure 9: Sandy-Bridge Server memory power consumption for MKL DSPMV

[6] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A Framework for Architectural-level Power Analysis and Optimizations. *SIGARCH Comput. Archit. News*, 28(2):83–94, May 2000.

[7] Tomasz Buchert. Methods for Emulation of Multi-core CPU Performance. Master's thesis, Poznan University of Technology, Poland, 2010.

[8] J.A. Butts and G.S. Sohi. A Static Power Model for Architects. In *Microarchitecture, 2000. MICRO-33. Proceedings. 33rd Annual IEEE/ACM International Symposium on*, pages 191 –201, 2000.

[9] Intel Corporation. Intel Hyper-Threading Technology. `http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html`.

[10] Intel Corporation. Intel Turbo Boost Technology: On-Demand Processor Performance. `http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html`.

[11] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual. 2011.

[12] William Dally. Power Efficient Supercomputing. Accelerator-based Computing and Manycore Workshop, 2009.

[13] Tim Davis. The University of Florida Sparse Matrix Collection. `http://www.cise.ufl.edu/research/sparse/matrices/`.

[14] Noel Eisley, Vassos Soteriou, and Li-Shiuan Peh. High-level Power Analysis for Multi-core Chips. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, CASES '06, pages 389–400, New York, NY, USA, 2006. ACM.

[15] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and K.W. Cameron. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. *Parallel and Distributed Systems, IEEE Transactions on*, 21(5):658 –671, may 2010.

[16] Ed Grochowski and Murali Annavaram. Energy Per Instruction Trends in Intel Microprocessors. *Computer*, 2006.

[17] Urs Hoelzle and Luiz Andre Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[18] Sunpyo Hong and Hyesoon Kim. An Integrated GPU Power and Performance Model. *SIGARCH Comput. Archit. News*, 38(3):280–289, June 2010.

[19] Canturk Isci and Margaret Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 93–, Washington, DC, USA, 2003. IEEE Computer Society.

[20] S. Kamil, J. Shalf, and E. Strohmaier. Power efficiency in High Performance Computing. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –8, april 2008.

[21] Peter Kogge, Dan Campbell, Jon Hiller, Mark Richards, and Allan Snavely. ExaScale Computing Study : Technology Challenges in Achieving Exascale Systems. *Government PROcurement*, page 278, 2008.

[22] John D. McCalpin. STREAM: Sustainable Memory Bandwidth in High Performance Computers. Technical report, University of Virginia, Charlottesville, Virginia, 1991-2007. A continually updated technical report. http://www.cs.virginia.edu/stream/.

[23] National Institute of Standards and Technology. Matrix Market Exchange Formats. `http://math.nist.gov/MatrixMarket/formats.html`.

[24] Fred J. Pollack. New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies. *Microarchitecture, IEEE/ACM International Symposium on*, 0:2, 1999.

[25] Mark Rowland. Email correspondence, 2012.

[26] Zhang Rui. introduce intel_rapl driver. Posted on LWN.net, 2011.

[27] Youcef Saad. SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations - Version 2, 1994.

[28] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware Microarchitecture: Modeling and Implementation. *ACM Trans. Archit. Code Optim.*, 1(1):94–125, March 2004.

[29] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM*, 52(4):65–76, April 2009.