

PAKCK: Performance and Power Analysis of Key Computational Kernels on CPUs and GPUs

Julia S. Mullen, Michael M. Wolf and Anna Klein
MIT Lincoln Laboratory
Lexington, MA 02420
Email: {jasm, michael.wolf,anna.klein }@ll.mit.edu

Abstract—Recent projections suggest that applications and architectures will need to attain 75 GFLOPS/W in order to support future DoD missions. Meeting this goal requires deeper understanding of kernel and application performance as a function of power and architecture. As part of the PAKCK study, a set of DoD application areas, including signal and image processing and big data/graph computation, were surveyed to identify performance critical kernels relevant to DoD missions. From that survey, we present the characterization of dense matrix-vector product, two dimensional FFTs, and sparse matrix-dense vector multiplication on the NVIDIA Fermi and Intel Sandy Bridge architectures.

We describe the methodology that was developed for characterizing power usage and performance on these architectures and present power usage and performance per Watt for all three kernels. Our results indicate that 75 GFLOPS/W is a very challenging target for these kernels, especially for the sparse kernels, whose performance was orders of magnitude lower than dense kernels.

I. INTRODUCTION

As sensor and data collection capabilities continue to improve on embedded systems, the amount of data for exploitation and analysis has continued to increase. Additionally, the complexity of algorithms necessary to process the data, often in real-time, has grown. Size, weight, and power (SWaP) constraints of DoD platforms is driving the need for computer architectures and processing systems that can support very high performance per Watt for a range of applications.

A common measure of power performance is the number of floating point operations computed per second per Watt or FLOPS/W. Current high-performance architectures, such as IBM Blue Gene, obtain over 2 GFLOPS/W but future DoD platforms need embedded systems that can support up to 50 GFLOPS/W for a range of applications. The DARPA Power Efficiency Revolution For Embedded Computing Technologies (PERFECT) program Broad Area Announcement addressed an ambitious power performance goal of 75 GFLOPS/W. To meet these constraints, revolutionary advances are necessary in power efficient architectures and applications. The Performance and Power Analysis of Key Computational Kernels

(PAKCK) study was funded by DARPA to gather performance data necessary to address both requirements.

As part of the PAKCK study, several key computational kernels from DoD applications were identified and characterized in terms of power usage and performance for several architectures. A key contribution of this paper is the methodology we developed for characterizing power usage and performance for kernels on the Intel Sandy Bridge and NVIDIA Fermi architectures. We note that the method is extensible to additional kernels and mini-applications. An additional contribution is the power usage and performance per Watt results for three performance critical kernels.

II. KEY COMPUTATIONAL KERNELS

A. Dense Kernels

Many DoD mission areas, particularly surveillance and reconnaissance, rely heavily on signal and image processing. The computational tasks associated with airborne surveillance focus on ingesting and processing streaming data in real time to create actionable products (e.g., images or detections). An analysis of canonical signal processing chains for several aerial surveillance modes: ground motion target indicator (GMTI), dismount motion target indicator (DMTI), and synthetic aperture radar (SAR) indicates that the fast Fourier transform (FFT) is the key computational kernel for this processing. In fact, FFT centric kernels account for approximately eighty percent of the processing. Less important to signal processing than the FFT but ubiquitous in computational science and engineering, the dense matrix-vector multiplication operation was the second kernel chosen for inclusion in this paper.

The trend in aerial surveillance is to process ever larger amounts of data on hardware capable of operating within the tight SWaP constraints of a small airborne platform such as an unmanned air vehicle (UAV) where the available power can be as low as tens of Watts. The particular challenge dense kernels pose to modern multicore architectures is that they spend significant time and energy moving data throughout the memory hierarchy. The 2D FFT, when distributed across processors requires an all-to-all communication between the FFT of the first and second dimension to create the contiguous data blocks necessary for computation. The matrix-vector kernel, on multiple processors, requires communication of the partial products prior to the reduction step. Historically, effort has been spent to optimize the runtime of these kernels,

*This work was sponsored by Defense Advanced Research Projects Agency (DARPA) under Air Force contract FA8721-05-C-0002. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government. This document is Approved for Public Release, Distribution Unlimited.

but energy is now equally important and the first step is to characterize kernel performance for energy and time.

B. Sparse Kernels

Sparse matrices and graphs are used throughout computational science and engineering fields and are becoming increasingly important in big graph analytics. This survey of sparse matrix applications was focused on those with sparse matrices resulting from graphs, in particular network graphs. Computations on very large network graphs are becoming an increasingly important area of interest for the DoD. Related problems of interest include the detection of threats in a social network, the detection of intruders in a computer network, and the identification of anomalous actors from multi-INT data. It is important to note that this choice of network graphs is particularly challenging for modern computer architectures.

There are at least three significant challenges for the parallel computation on these sparse matrices: irregularity of communication and computation, a lack of data locality, and the increased significance of communication latency in the computation. These parallel challenges also have relevance to serial computation with communication being generalized to included memory accesses. These challenges make it very difficult for modern architectures to realize peak performance, rendering caches (that enable peak performance for dense operations) useless.

This paper focuses on sparse matrix-dense vector multiplication (SpMV), a kernel of particular importance to sparse computation and to graph analytics related to network data. SpMV is the workhorse kernel of sparse iterative methods for solving linear systems and eigenvalue problems (e.g., CG, GMRES, and Krylov-Schur based eigensolvers) and often dominates the runtime of these methods. SpMV has been an essential kernel in the Signal Processing for Graphs work undertaken at MIT Lincoln Laboratory, where the computation of principal components (obtained through the use of eigensolvers) is a key step in finding anomalies in network data [1], [2]. For our experiments with these sparse kernels, synthetically generated R-MAT matrices [3] are used, which are typical of many graph applications derived from network data. R-MAT matrices approximate a power law degree distribution and have the advantage of being easy to scale across a range of problem sizes.

III. METHODOLOGY

A. Overview

For this study the metric of interest is performance per Watt, (GFLOPS/W). To characterize the kernels, the time and power were collected during kernel execution on the test platforms and the operation count is based on the theoretical value associated with the algorithmic expression rather than an actual count of operations performed. To maintain consistent operation counts across kernels applied to real and complex data, complex operations were decomposed into real operations for the purpose of operation count.

A number of performance tuning library tools exist to measure FLOPS, execution time, total instructions, memory usage and a variety of similar metrics. Many open source performance libraries use software estimates and virtually all open source libraries using hardware counters sit on top of the Performance API (PAPI) [4], [5]. Vendor specific libraries, which use hardware counters, are available for some platforms (e.g., vTune from Intel), but generally do not provide a means to capture power or energy usage. After reviewing a number of performance analysis tools, PAPI was chosen because the library: uses hardware counters to capture performance data, provides access to traditional performance data (e.g., runtime, FLOPS), and provides a means of accessing the energy and power counters for a few platforms.

Initial characterization experiments gathered performance data (FLOPS) via PAPI's preset command, PAPI_FP OPS. The results returned from this command were unreliable and appeared to return a count of floating point instructions rather than floating point operations. Lacking confidence in PAPI_FP OPS results, the approach was modified to capture the execution time via PAPI_get_real_nsec and compute performance using the theoretical operation counts for the kernels.

The development of techniques for obtaining accurate power measurements for computations at the application or kernel level is an active area of investigation. The tests reported here used PAPI-5's low level API to capture power or energy depending on supported hardware counters. To simplify the instrumented code, the PAPI initialization commands were placed in a header file, such as the one in Figure 1 for accessing NVML on the NVIDIA Fermi. The commands include the initialization of the library, determination and initialization of available components, in this case the NVML component, and creation of the EventSet containing the power and temperature events. The code in Figure 2 illustrates how PAPI was used to instrument a CUDA kernel in order to gather the runtime and power values while running the kernel. The commands and events are architecture specific and described in more detail in individual architecture sections.

```
// Header file
#include "papi.h"
char papiEvents[NUM_PAPI_EVENTS][BUFSIZ]=
{
    "Tesla_C2075:power",
    "Tesla_C2075:temperature",
};
int papiInit()
{
    int retval = PAPI_library_init( PAPI_VER_CURRENT );
    // check for the NVML component
    int numcmp = PAPI_num_components();
    for(cid=0; cid<numcmp; cid++)
    {
        if (strstr(cmpinfo->name,"nvml"))
            nvml_cid=cid;
    }
    // create the event set for power and temperature
    for(int i=0;i<NUM_PAPI_EVENTS;i++)
    {
        retval = PAPI_add_named_event( EventSet, papiEvents[i] );
    }
    return EventSet;
}
```

Fig. 1. PAPI Header Initialization of NVML Component

```

// instrumented Cuda code using NVML
int EventSet = papiInit();
long long values[NUM_PAPI_EVENTS],time1,time2;
PAPI_start(EventSet);
for (int trial=0; trial < numRuns; trial++)
{
    time1 = PAPI_get_real_nsec();
    runKernel
    time2 = PAPI_get_real_nsec();
}
PAPI_stop(EventSet, values);
double power = (double)values[0]*(0.001);
double elapsedTime = double(time2-time1)*(1.0e-09);

```

Fig. 2. PAPI-NVML Instrumentation of Cuda Code

B. NVIDIA Fermi

For our GPU study we chose the NVIDIA Fermi (Tesla C2075), because the power and temperature values are exposed through the NVIDIA Management Library (NVML). For this architecture the PAPI-NVML interface reports instantaneous power for the entire board, GPU and memory. The counter has milliwatt resolution and is sampled at approximately 60Hz [6]. This provides a stable, but low fidelity means of gauging power usage.

The experiments reported here used CUDA-5.0 with nvcc version 0.2.1221 and the instrumentation was placed around a call to the kernel as indicated in Figure 2. We note that the PAPI-5.1.0 library reads NVML power counters approximately every millisecond, so kernel execution time must be long enough to insure that the counter is capturing the power required to execute the kernel rather than reporting ambient power. We accomplished this by placing each kernel test inside a loop of five thousand trials to ensure sufficiently long sample times.

C. Intel Sandy Bridge

We chose the Intel Sandy Bridge as our CPU platform because in 2010, Intel exposed chip and core level energy measurements for the architecture via the Running Average Power Limit (RAPL) package [7]. The RAPL package is a software power model using hardware counters, temperature and leakage models to estimate energy use. The measurement values are exposed through Machine Specific Registers (MSRs) which can be polled, with appropriate software tools, to determine energy usage for a block of code. Using PAPI's low level API, the RAPL MSRs can be accessed and energy values (in nanoJoules) can be obtained for the [5]

- **Package**, the on-core energy of entire CPU package
- **PP0 Energy**, the total combined energy used by all cores
- **PP1 Energy**, the on-chip GPU (not active on all Sandy-Bridge chips)
- **DRAM**, the DRAM interface.

Note that the Package energy is for the chip and the memory, but not the DRAM memory. The DRAM is off-core and because the hardware data are not currently accessible via the low level PAPI API, it is not included in the results presented here. Based on the work of Rotem, et.al. [8], who validated the RAPL estimates against measured power data, we have confidence in the estimates obtained using PAPI and RAPL

to characterize power usage of the kernels. For the purposes of this study, the energy of the entire on-core CPU package provided the appropriate level of fidelity.

Characterization of the kernels on the CPU architecture using PAPI required instrumenting the code as described in the GPU methods section. The header file (Figure 1) used for the GPU was modified to search for the RAPL component rather than the NVML component and the kernel instrumentation structure was similar to that shown in Figure 2 with some subtle differences. When the EventSet is started with PAPI, the RAPL interface reads the hardware counter to get an initial value of energy, the execution of PAPI_Stop results in a second read of the hardware counter and PAPI stores the amount of energy used between the start and stop. Once PAPI has captured the total energy used, power is determined from the energy (nJ) and time in (nanoseconds).

For all of the kernel experiments, the PAPI timing instrumentation was placed directly around the kernel call, inside a loop iterating over a number of trials. The number of kernel trials varied with data size ranging from 100 for the smaller data sizes to one for the largest data size. This difference is directly connected to the length of time it took to execute the kernel, smaller array sizes executed so quickly that there was no time to sample the energy counters while larger kernels took so long that the energy samples overflowed the buffers. The PAPI-RAPL energy instrumentation was placed outside the loop and the reported energy and time results represent the average over the kernel trials. The architecture was composed of two eight core nodes and experiments were performed for $N_p = 1, 2, 4, 8, 16$ cores.

IV. RESULTS

A. Dense Matrix-Vector Multiplication

1) *NVIDIA Fermi*: The dense matrix-vector multiplication tests on the GPU used the CUBLAS [9] implementation of the GEneralized Matrix Vector multiplication kernel and included single (SGEMV) and double precision (DGEMV) versions. To approximate a general use case, the instrumented block included the transfer of the input and result vectors, the GEMV operation and the return of the result vector to the host CPU. A comparison of the single and double precision results for matrix sizes $N = 1024$ to $N = 8192$ are presented in Figure 3.

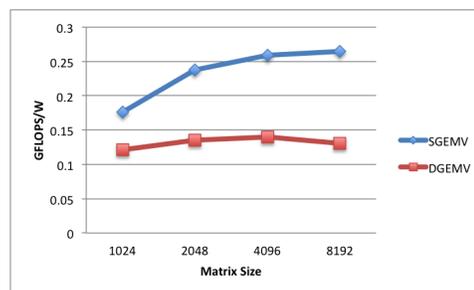


Fig. 3. Performance per Watt (GFLOPS/W) characterization for SGEMV vs. GEMV on NVIDIA Fermi.

The increase in performance with data size confirms that the GPU performs best when it has a large amount of work relative to the data movement. The DGEMV results begin to level off at $N = 4096$ because of the added data movement caused by the additional blocking required of larger data sizes. The power usage for the single and double precision kernel execution is almost identical. What differentiates the two kernels is the performance, where the single precision implementation is roughly 2x faster than the double precision version.

2) *Intel Sandy Bridge*: We characterized GEMV on the CPU with the double precision BLAS kernel DGEMV. Serial experiments used the FORTRAN reference BLAS kernel called from a C++ wrapper. The parallel experiments used pDGEMV (pBLAS), which uses both the BLAS and MPI libraries.

The performance results for all matrix sizes and processor counts are shown in Figure 4. For the most part, the performance increased as we increased the number of processors (at least up to 8 processors) For most of the parallel experiments, we see a slight decrease in the performance as the matrix size is increased. The sharp drops in performance for 8 and 16 processors seems to correspond to the problem no longer fitting into the L3 cache. The performance per Watt for all problem sizes and cores counts is shown in Figure 5. As expected the power increases with number of processors due to increased utilization of processor cores. There is also a slight increase in the power as the problem size is increased. From Figure 5 it is clear that up to a problem size of 8192, the increased performance due to increased parallelism more than compensates for the increase in power usage.

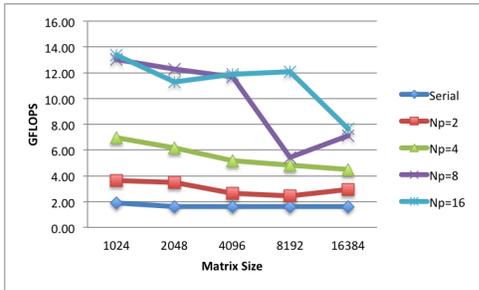


Fig. 4. Performance for DGEMV on Intel Sandy Bridge (GFLOPS). Results for matrix sizes $N = 1024$ to $N = 8192$ on processor counts $N_p=1$ to $N_p=16$.

A comparison of the DGEMV implementations on the CPU and GPU shows that while the GPU is not tuned for double precision calculation, it clearly outperforms all CPU implementations of the kernel.

B. 2D Fast Fourier Transform

We present the results for the 2D FFT on the NVIDIA Fermi and Intel Sandy Bridge architectures. For parallel 2D FFTs, it is important to note that there is a data reorganization step (“corner turn”) that is particularly challenging for this kernel.

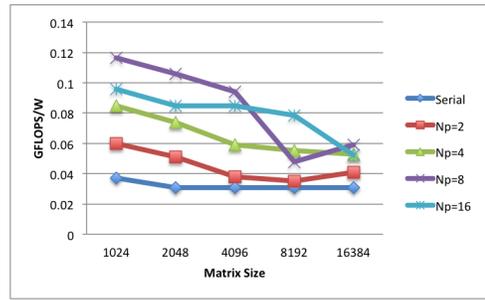


Fig. 5. Performance Per Watt for DGEMV on Intel Sandy Bridge (GFLOPS/W). Results for matrix sizes $N = 1024$ to $N = 8192$ on processor counts $N_p=1$ to $N_p=16$.

1) *NVIDIA Fermi*: We chose the 2D FFT in the CUFFT library (CUDA 5.0) as our reference kernel for the GPU characterization study. To closely match actual use cases, the FFT signal data was assumed to be streaming into the FFT and the results forwarded on to the next stage for further processing. Thus, the time instrumentation commands were placed directly around data movement and FFT calls, and the power instrumentation was outside of a loop of kernel execution trials. The choice of sizes for the FFT tests ranged from $N = 1024$ to $N = 8192$ and all FFTs were single precision complex to complex. The number of trials was five thousand for all tests. The performance results from the GPU implementation (orange line in Figure 6) indicate a steep rise in performance per watt as the problem size is increased from 1024 to 2048, followed by a moderate increase in performance with problem size. This significant increase in performance suggests that for $N > 2048$, the cost of moving the FFT signal data onto and off of the GPU was offset by the increase in computation required by the larger problem size.

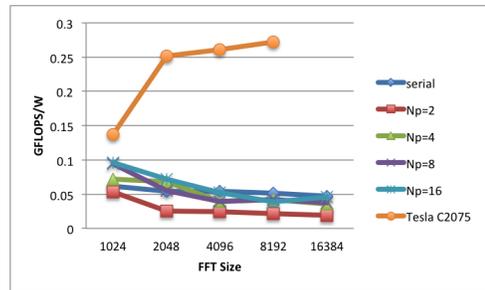


Fig. 6. Performance per Watt comparison between Intel Sandy Bridge and NVIDIA Fermi for 2D FFT (GFLOPS/W).

2) *Intel Sandy Bridge*: The 2D FFT tests on the CPU are based on the FFTW-3.3.2 2D double precision complex to complex transform. The first step in using FFTW is to create an FFT plan based on a set of parameters, the key parameters being, the FFT size and FFT type (e.g., complex to complex, real to complex). For the serial FFT, the library can use auto tuning to optimize the plan. For the parallel FFT, auto tuning is not available and a best plan is estimated based on the FFT parameters and the number of processors on which the FFT

is computed. The initialization of FFTW and the creation of the plan were not part of the instrumented code.

In addition to the estimated FFT plan, the parallel experiments used MPI to transfer data between processors in the corner turn step. The FFT was performed in place, and the data remapping in the corner turn step destroyed the ability to execute repeated FFT tests within a loop. While the serial experiments used multiple trials as described in III-C, for the parallel tests, the instrumentation was placed around the execution of the kernel and the reported time is the average time it took across all processors. The performance per Watt for all array sizes and processor counts is shown in Figure 7. Our experiments indicate that for a given core count the power is roughly constant for a given problem size increasing slightly for the problem sizes above 4096.

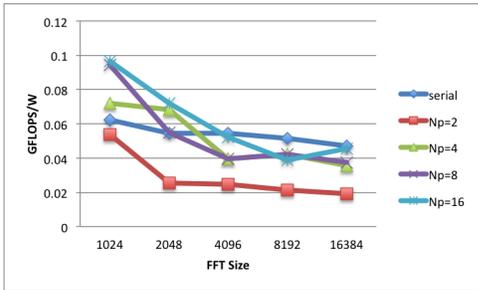


Fig. 7. Performance per Watt for 2D FFT on Intel Sandy Bridge.

We present the comparison of performance per Watt for the the 2D FFT on the CPU and GPU architectures in Figure 6. For both platforms the power stays relatively constant for a given problem size and number of processors. On the CPU we see performance degradation with with problem size that is not observed on the GPU yielding significantly higher performance per watt for the GPU implementations.

C. Sparse Matrix-Dense Vector Multiplication

1) *NVIDIA Fermi*: For our sparse matrix-dense vector multiplication (SpMV) numerical experiments on the NVIDIA Fermi GPU, we used the SpMV implementation found in the Cusp library [10]. For the sparse matrices, we used R-MAT matrices ($a = 0.5, b = c = 0.125, d = 0.25$) with an average of 8 nonzeros per row. We experimented with 8 different size matrices with the number of rows being powers of two in the range of 2^{15} to 2^{22} (scale 15 to 22 R-MAT matrices). We used a compressed sparse row format for the Cusp matrices, which is perhaps not well suited for this GPU architecture and stored the values of the matrix as double precision floating point numbers.

The green line in Figure 8 shows the power and performance results for this SpMV kernel for the 8 different problem sizes on the GPU. It is important to note that each data point is the average of 20 trials with 20 different R-MAT matrices for each problem size. This controlled the danger of generating one particularly poor or well performing R-MAT matrix. As the size of the problem increases, we see a decrease in the

GFLOPS/W power-performance metric. This is most likely due to the corresponding decrease in the data locality as the R-MAT matrix size increases, which results in decreased effective cache utilization. We theorize that the lack of data locality and irregularity in the data access patterns for these R-MAT matrices makes this kernel particularly challenging for the GPU. Bell and Garland showed that alternative sparse matrix storage formats to the compressed sparse row format can result in improved performance [11] for some classes of matrices (although they did not demonstrate this for R-MAT matrices). Thus, using an alternative storage format might improve the GPU results significantly.

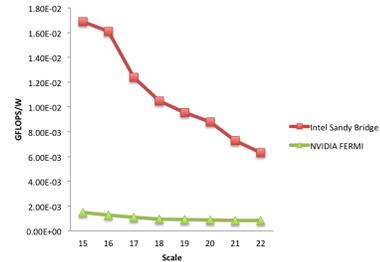


Fig. 8. Power and performance characterization results for SpMV on NVIDIA Fermi and Intel Sandy Bridge architectures. Maximum GFLOPS/W for scale=15 to scale=22 R-MAT matrices.

2) *Intel Sandy Bridge*: For our sparse matrix-dense vector multiplication (SpMV) numerical experiments on the Intel SandyBridge architecture, we used the SpMV implementation found in the Epetra package of the Trilinos software framework [12]. The Epetra implementation of SpMV uses a distributed memory (MPI based) approach to parallel computation. It is possible that a shared memory approach would result in slightly improved performance but for this kernel, we expect the improvement to be fairly negligible. We used the same R-MAT matrices as we did on the NVIDIA Fermi architecture. Epetra matrices use a compressed sparse row format, which should be well suited for the SpMV computation on this architecture, and store the values of the matrix as double precision floating point numbers. For the parallel experiments, the matrices are distributed in a one-dimensional row based fashion where each row is assigned to only one processor. The distribution of the rows is randomized to obtain good load-balance of the nonzeros across the processors (effectively balancing the computation). However, it is important to note that the random distribution of rows does not address the resulting communication of the SpMV operation. There are methods of determining distributions to minimize the communication in the SpMV operations [13] but determining good distributions for these power-law graph matrices is still an active area of research.

Figures 9 and 10 show the power and performance results for this SpMV kernel for the 8 different problem sizes and different numbers of processors on the Intel Sandy Bridge. Again, we averaged the results from 20 trials with 20 different R-MAT matrices for each problem size.

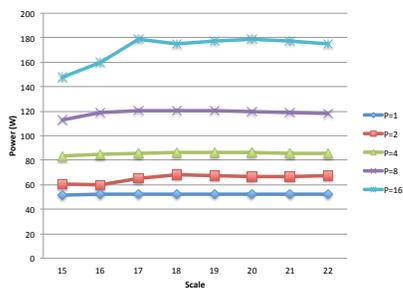


Fig. 9. Performance for SpMV on Intel Sandy Bridge (GFLOPS). Results for scale=15 to scale=22 R-MAT matrices.

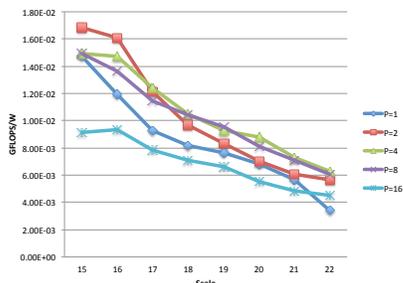


Fig. 10. Performance Per Watt for SpMV on Intel Sandy Bridge (GFLOPS/W). Results for scale=15 to scale=22 R-MAT matrices.

An increase in parallelism, results in an increase in power. Figure 10 shows the power-performance (GFLOPS/W) of the SpMV on the Intel Sandy Bridge architecture. As we increase the size of the problem, the performance actually decreases. This is most likely due to the corresponding decrease in the data locality as the R-MAT matrix size increases, which results in decreased effective cache utilization. However, we do see some improvement in the performance as we scale the number of processors up to 8. We do not see the linear performance improvement that we desire and the performance gets worse as we move from 8 to 16 processors, indicating that the communication costs are becoming increasingly significant as the number of processors increase. The “sweet spot” in the power-performance metric seems to be in the two to eight processor range. Figure 9 and Figure 10 summarize the power-performance metric, reporting the maximum value across the different number of processors for each problem size. Compared to the NVIDIA Fermi architecture, the Intel Sandy Bridge architecture performs significantly better for this sparse kernel.

V. CONCLUSIONS

We described a technique for capturing performance and power data on the Intel Sandy Bridge and NVIDIA Fermi architectures that is extensible to other kernels, mini-applications, and full applications. We applied our methodology to characterize the power and performance for three kernels of importance to DoD missions, dense matrix-vector product, two dimensional FFT and sparse matrix-vector multiplication.

Our results show the challenge of reaching the 75 GFLOPS/W PERFECT goal in the context of real architectures. The NVIDIA Fermi and Intel Sandy Bridge architectures failed to come within even 2 orders of magnitude of the 75 GFLOPS/W objective. As was expected, the sparse computational kernel SpMV is shown to be even more challenging, performing significantly worse the dense kernels and approximately four orders of magnitude below the 75 GFLOP/W target. It is clear that significant research breakthroughs need to be made to address these sparse problems. In the near future, too many important DoD problems such as big data analytics and multi-INT fusion that rely on sparse computation will need to be solved on SWaP constrained platforms for researchers to neglect this increasingly important area.

Moving forward, the methodology described here can be used in conjunction with additional kernels and mini-applications to further investigate the interplay between architecture and application with the goal of improving both.

ACKNOWLEDGMENT

The authors want to thank the DARPA MTO for support of the PAKCK study. We thank Nadya Bliss, Marc Burke, Karen Gettings, David Martinez and Albert Reuther for their contributions.

REFERENCES

- [1] B. A. Miller, N. Arcolano, M. S. Beard, J. Kepner, M. C. Schmidt, N. T. Bliss, and P. J. Wolfe, “A scalable signal processing architecture for massive graph analysis,” in *Proc. IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2012, pp. 5329–5332.
- [2] E. M. Rutledge, B. A. Miller, and M. S. Beard, “Benchmarking parallel eigen decomposition for residuals analysis of very large graphs,” in *Proc. IEEE High Performance Extreme Computing Conf.*, 2012.
- [3] D. Chakrabarti, Y. Zhan, and C. Faloutsos, “R-MAT: A recursive model for graph mining,” in *Proc. of the 4th SIAM Conference on Data Mining*, 2004, pp. 442–446.
- [4] [Online]. Available: <http://icl.cs.utk.edu/papi/>
- [5] V. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore, “Measuring energy and power with papi.” 1st International Conference on Power-Aware Systems and Architectures, September 2012.
- [6] K. K. Kasichayanula, “Power aware computing on gpus.” *Master’s Thesis, University of Tennessee*, 2012. [Online]. Available: http://trace.tennessee.edu/utk_gradthes/1170
- [7] H. Esmaeilzadeh, T. Cao, X. Yan, S. Blackburn, and K. McKinley, “Looking back and looking forward: Power, performance, and upheaval,” *Communications of the ACM*, vol. 55, no. 7, pp. 105 – 114, 2012.
- [8] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and W. Weissman, “Power-management architecture of the intel microarchitecture code-named sandy bridge,” *IEEE Micro*, vol. 32, no. 2, pp. 20 – 27, 2012.
- [9] [Online]. Available: <http://docs.nvidia.com/cuda/cublas/index.html>
- [10] N. Bell and M. Garland, “Cusp: Generic parallel algorithms for sparse matrix and graph computations,” 2012, version 0.3.0. [Online]. Available: <http://cusplibrary.github.com/>
- [11] —, “Efficient sparse matrix-vector multiplication on CUDA,” NVIDIA Corporation, NVIDIA Technical Report NVR-2008-004, Dec. 2008.
- [12] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, “An overview of the trilinos project,” *ACM Trans. Math. Softw.*, vol. 31, no. 3, pp. 397–423, 2005.
- [13] K. Devine, E. Boman, L. Riesen, U. Catalyurek, and C. Chevalier, “Getting started with zoltan: A short tutorial,” in *Proc. of 2009 Dagstuhl Seminar on Combinatorial Scientific Computing*, 2009, also available as Sandia National Labs Tech Report SAND2009-0578C.