Optimizing the Performance and Energy of LU Decomposition on a Heterogeneous Multicore System with GPGPUs and DVFS

Brian Hunter*

University of Wisconsin - Madison
bdhunter@wisc.edu

7/31/2013

Abstract

The goal for computer applications is to run with maximal performance and minimal energy consumption. This is a critical issue in high performance computing where speed is increased at the expense of using large amounts of energy. For continued progress in this field, it is essential to develop methods which will allow for a low energy usage while still maintaining high performance. This study explores ways in which to lower the energy consumption of high intensity computations by experimenting with heterogeneous computation, frequency scaling, and dynamic scheduling on the LU decomposition application.

I. Introduction

In the field of high performance computing, a continuous growth of computer performance is desirable. Currently, the world's fastest supercomputer, Titan, located in the U.S., runs at 17.3 PetaFlops, or 17.3×10^{15} operations per second. Performance that breaks the ExaFlops barrier, or 10^{18} operations per second is a major goal to achieve in the next decade. However, pushing towards this goal has become difficult with finding ways to gain performance without using too much energy. The Titan supercomputer runs at 8.3 MegaWatts and scaling that figure to ExaFlops would be staggering. Recently, GPGPUs (General Purpose Graphics Processing Unit) have been explored as a means to obtain energy efficient computation, measured in GFlops (billions of operations per second) per watt. Along with the new general computing capability, new GPGPUs are also capable of dynamic voltage frequency scaling (DVFS), which allows them to run at various frequencies, allowing them to be scaled down when they are not heavily used. CPUs (Central Processing Unit) have also been equipped with this feature. The purpose of this research is to explore the benefits of using a heterogeneous approach to computing and implementing DVFS on both CPUs and GPUs to optimize the performance and energy of running an LU decomposition application.

All tests are run on a heterogeneous multicore system. There are sixteen Xeon SandyBridge E5-2670 cores, each of which has a default speed of 2.6 GHz, and the GPU used is a Tesla K20c, which has 4799.6 MB of local memory, a default memory speed of 2.6 GHz, and a core speed of 705 MHz. Each of Xeon cores is capable of running at 1.2 to 2.6 GHz, going up by 0.1 GHz, and 2.601 GHz. The K20c is capable of six frequencies: 2600x758, 2600x705, 2600x666, 2600x640, 2600x614, and

*Thank you to Professor Rong Ge of Marquette University Department of Mathematics, Statistics, and Computer Science for advising me during this project.

324x324 MHz, where the speeds are given by memory and core frequency, respectively.

Two linear algebra libraries are used for the tests, MAGMA and MORSE. All tests run solely on the CPU used the MORSE implementation of LU decomposition with 16 threads. Tests implementing both the CPU and GPU are run with MAGMA using 1 GPU and 1 thread.

II. Related Work

Accelerators such as GPUs are being explored as a means for energy efficient computing, especially in recent years. New libraries such as the MAGMA library have been implemented to run on heterogeneous systems for increased performance on linear algebra functions. These libraries contain methods to minimize the overhead for mapping data between the CPU and GPU and have had promising results in terms of speed-up.

A study by Qiang Liu and Wayne Luk[3] shows the potential for speed-up and energy efficiency using a GPU. They tested single precision matrix multiplication with a problem size of 20,480 and found that the a CPU+GPU combination gave 1.08 GFlops/Watt which is 24x more efficient than running only on the CPU which gave 0.06 GFlops/Watt. Furthermore, their GPU for the computation, an nVidia C2070 GPU has a throughput of 558.94 GFlops and power rating of 164.4 Watts, which gives the maximum efficiency of 3.47 GFlops/Watt. On the other hand, the CPU, a Xeon w3505, runs at 151 Watts for 12.83 GFlops as a maximum throughput; an efficiency of 0.085 GFlops/Watt. Again, the GPU has shown exemplary performance and energy efficiency. Their findings also suggest that GPUs have higher performance and more energy efficiency than another accelerator, FPGA (Field-Programmable Gate Array).

More can be done in for efficiency with using GPUs in terms of scheduling, to make sure that no computational resources are wasted. For this study, the strategy is to turn down components when not in use to lower power draw. Alternative approaches exist, a notable one being the strategy in [2], in the software *Qilin*. Instead of depending on the programmer to modify an application, as is the case for this paper, the research in [2] proposes an adaptive software to map work to open computational resources. The reason for adaptive software is that a hard-coded modification made by a programmer is only relevant to the system on which they are running and will not scale well to others. Depending on the application and system, the distribution of work that provides the best performance can be variable. By using adaptive mapping, which takes advantage of a heterogeneous system, the researchers demonstrate that the speed-up is greater than using only GPU or only GPU, and is typically also faster than a static distribution of the two. Using both CPUs and GPUs provides the best performance in their results, which is explored in this paper as well.

III. Methods

A comparison of a CPU-only approach and heterogeneous approach is done by running one iteration of an identical identical problem size for both over all range of CPU and GPU frequencies, and observing the performance and energy used. These figures also show which frequency gives the best performance for energy used. For the CPU-only runs, the power trace is analyzed to see the behavior of the different frequencies over the course of the runtime.

The heterogeneous method is further explored by running six iterations at four different problem sizes. For each problem size, the performance and energy is observed as well as the power trace to see the computational demand on GPU over time and how the different frequencies behave during runtime.

DVFS is used to dynamically schedule the runtime by modifying the MAGMA LU decomposition method to run the GPU at the maximum frequency during periods of heavy computation, scale down the GPU when not in use, and scale down the CPU cores not being used. The performance and energy of the modified run are compared the performance and energy to the standard method.

IV. Results and Discussion

The comparison between a CPU-only and heterogeneous approach is demonstrated on an 18,000 square matrix. Results of the heterogeneous run are taken by the fastest run over ten runs. There is more variation in using the CPU as there are background processes running. To accommodate for such variation, the results for the CPU-only run are averaged over six runs, with the minimum and maximum performance also recorded.

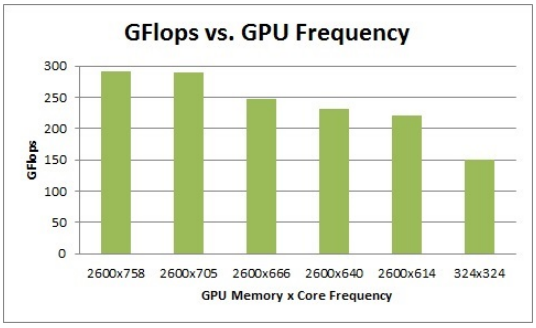

Figure 1: *Heterogeneous performance on an 18000 square matrix.*

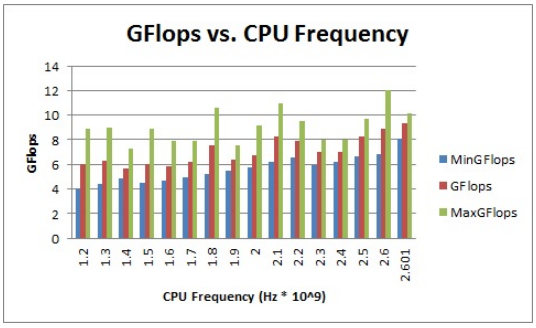

Figure 2: *CPU-only performance on an 18000 square matrix.*

Table 1: *Efficiency for GPU Frequencies*

Frequency MHz	Time Seconds	Energy Joules	Efficiency GFlops/Watt
324	25.73	10113.15	0.24
614	17.6	8785.79	0.28
640	16.77	8390.03	0.30
666	15.78	8390.19	0.30
705	13.4	5974.28	0.42
758	13.38	3519.76	0.54

It is clear to see that the heterogeneous method achieves a much higher performance than using only the CPU. The highest performance for CPU-only is 12.07 GFlops at 2.6 GHz,whereas the GPU obtained 290.57 GFlops at 758 MHz, a 24x performance increase. Going even further into the individual GPU frequencies, the computational efficiency increases as the cores are run faster. Better performance comes with increased efficiency, which is excellent for finding more efficient means of computing. These tests demonstrate the importance of a heterogeneous approach and that for a single iteration, using the highest GPU frequency is optimal.

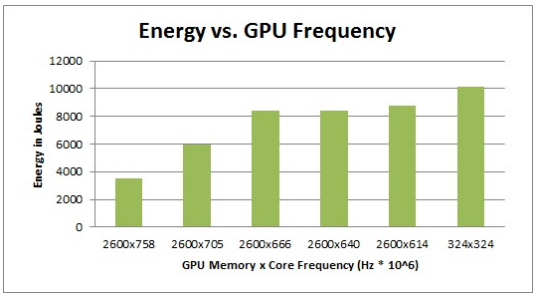

Figure 3: *Energy consumption of GPU frequencies for an 18000 square matrix.*

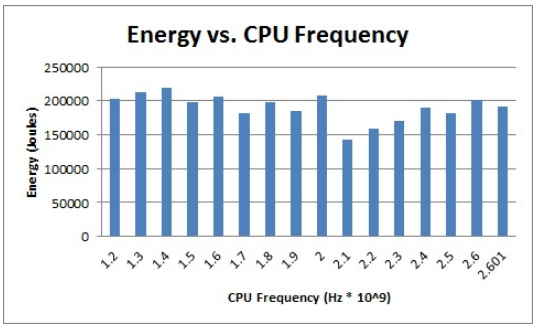

Figure 4: *Energy consumption of CPU frequencies for an 18000 square matrix.*

With a 24x shorter runtime, significant results are also seen in the energy consumed by each approach. The heterogeneous approach consistently used less than $\frac{1}{15}$th of the energy as the CPU-only variation. It is important to note that the primary reason for this saving in energy is due to the curtailed runtime, as there is less to time to consume energy.

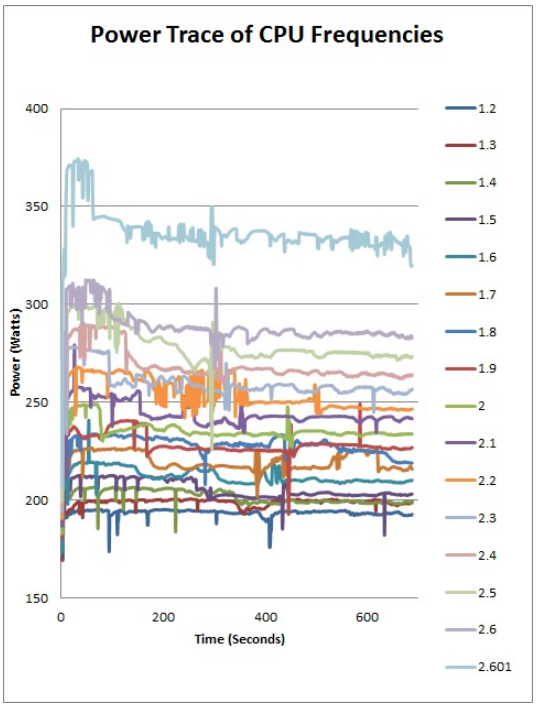

Figure 5: *Power trace of CPU frequencies for an 18000 square matrix.*

Looking at the power trace for the CPU-only approach, the trend is more power for higher frequencies. There is a notable increase in the power levels for 2.6 and 2.601 GHz. Referring to the performance, this increase in power is costly for marginally better performance. 2.6 GHz had the fastest speed over all runs for the CPU-only approach. This is important to keep in consideration for the dynamic scheduling strategy. Using 2.6 GHz for the cores in use will be more efficient than using 2.601 GHz.

With significant improvements in both speed and energy consumption, the heterogeneous approach is the best one to use. Further analysis of this approach is important to find the computation demands and optimal frequencies for different sizes. Four matrix sizes are tested: 18,000, 20,000, 22,000, and 24,480 and one test consists of six iterations of MAGMA's LU decomposition application.

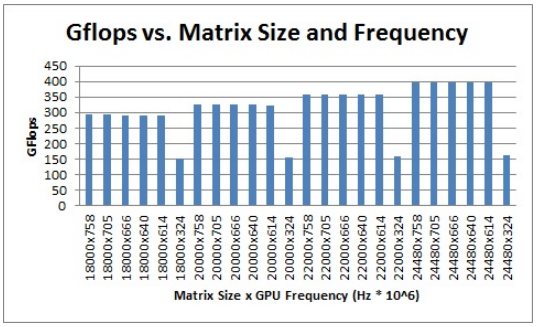

Figure 6: *Performance of heterogeneous approach for all test sizes and GPU frequencies.*

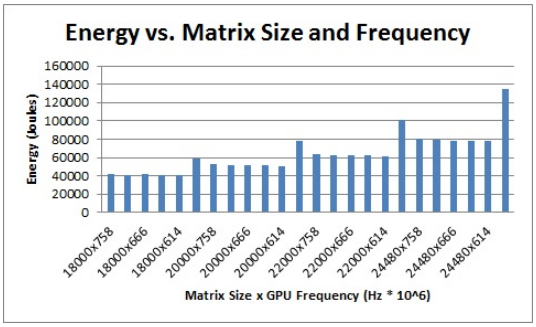

Figure 7: *Energy of heterogeneous approach for all test sizes and GPU frequencies.*

There is a similar trend for all four matrix sizes. The five fastest frequencies have almost identical performance and energy consumption, with 758 MHz having the slight lead in performance, and 614 MHz having the lowest energy usage. In all cases, running the GPU at 324 MHz has significantly less GFlops, less than half of the others for matrix sizes 20,000, 22,000, and 24,480. This also comes with a greater amount of energy needed to run at this frequency. Running the GPU at 758 MHz gives the best performance with only slightly more energy than the others, save 324 MHz. It is also

clear that a core speed of 324 MHz is not fast enough to efficiently handle the computational demand of LU decomposition. LU decomposition, using an iterative method will not require the GPU be doing work during the entirety of the run, thus, analyzing the power traces for each matrix size is valuable to find where the GPU can be scaled down, and if 324 MHz has a power draw much lower than the other frequencies.

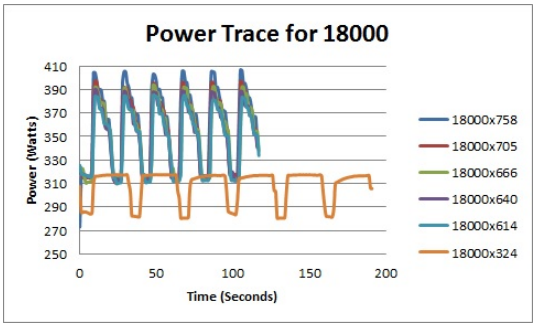

Figure 8: *Power trace for matrix size 18000.*

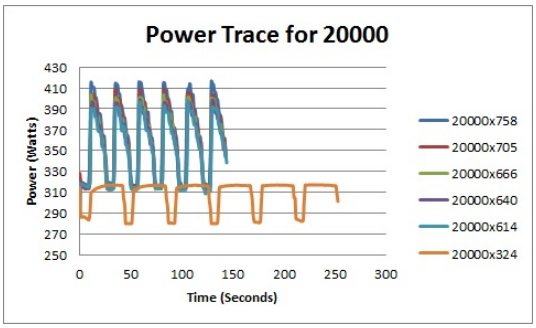

Figure 9: *Power trace for matrix size 20000.*

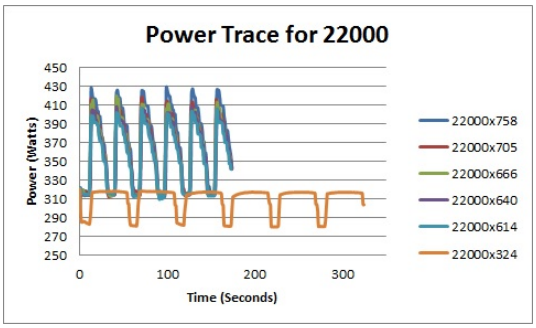

Figure 10: *Power trace for matrix size 22000.*

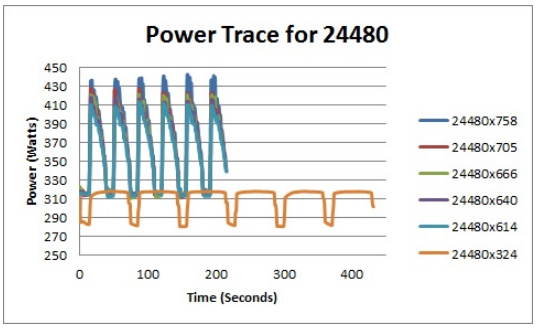

Figure 11: *Power trace for matrix size 24480.*

The above power trace figures show that the computational demands happen in waves, with periods of idle time between, which is expected of an iterative method. The five fastest frequencies (758, 705, 666, 640, and 614 MHz), are quite similar, which matches the data in figure 7. For the dynamic scheduling strategy, running the GPU at 758 MHz for the computationally intense periods is the best in terms of performance and efficiency. It costs little additional energy for added performance. Running the GPU at 324 MHz commands much less power than the other frequencies and it is worthwhile to scale down the GPU to this level when it is idle. Switching between these frequencies will give computational power when it is needed, and low power draw when possible.

With those observations, the dynamic scheduling strategy will consist of running 2 CPU cores at 2.6 GHz, one for the system to run on and one for MAGMA. All other CPU cores are scaled down to 1.2 GHz to save energy. The GPU will be switched to 758 MHz when it is being used, and is switched to 324 MHz at all other times. Doing this will take advantage of the previous results of Table 1, where 758 MHz is shown to be the most energy efficient for a single iteration. Scaling down during idle time will prevent the GPU from wasting energy during the times between each iteration where the CPU is initializing the matrices. In general, the goal is to have maximum performance for periods of heavy computation, and minimal energy for times where there is no computation to be done. This new method is tested for the

same cases and contrasted against static runs which use all CPU and GPU cores at 2.6 GHz and 758 MHz, respectively, for the entire run.

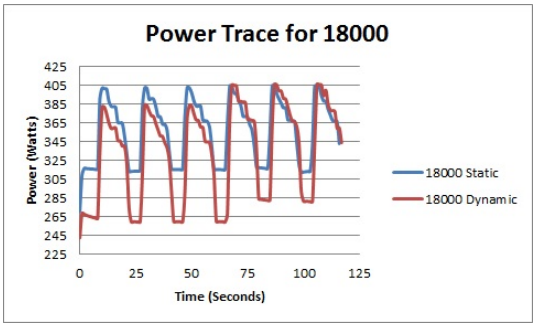

Figure 12: *Power trace for the schedule comparison at 18000.*

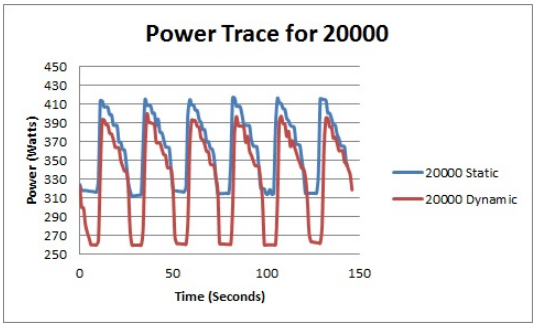

Figure 13: *Power trace for the schedule comparison at 18000.*

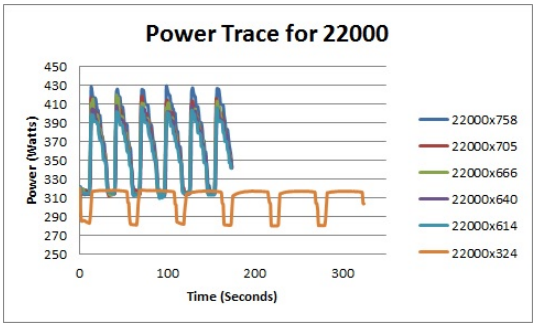

Figure 14: *Power trace for the schedule comparison at 22000.*

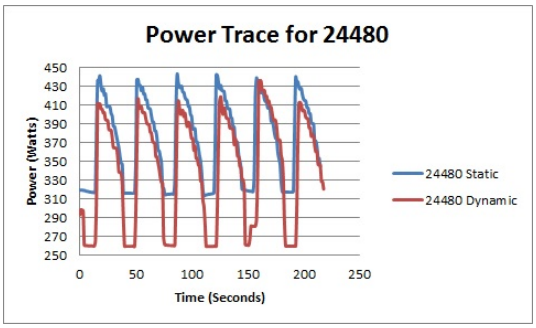

Figure 15: *Power trace for the schedule comparison at 24480.*

An analysis of the power traces show that scaling down unused computational resources is effective for conserving energy. These savings tend to get larger for the larger matrix sizes. LU decomposition not only is more efficient on the dynamic schedule, but also did not suffer performance loss.

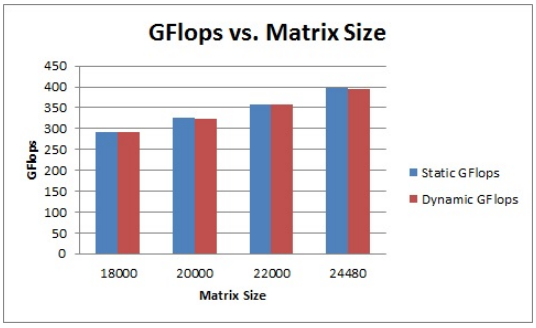

Figure 16: *Performance comparison of dynamic and static schedules.*

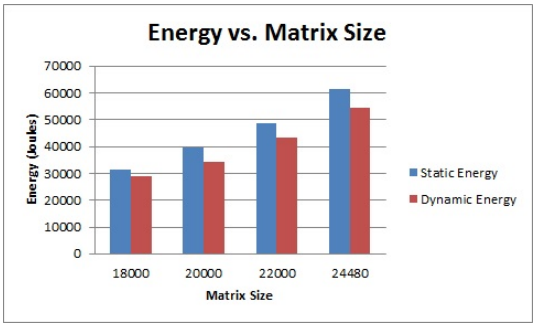

Figure 17: *Energy comparison of dynamic and static schedules.*

Table 2: *Energy Savings (Joules)*

Size	Static	Dynamic	Savings
18000	31349.14	29019.77	7.43%
20000	39441.71	34324.76	12.97%
22000	48749.26	43199.82	11.38%
24480	61268.85	54439.70	11.15%

Table 3: *Efficiency for the Schedules (GFlops/Watt)*

Size	Schedule	GFlops	Efficiency
18000	Static	292.92	0.74
20000	Static	326.03	0.81
22000	Static	357.18	0.87
24480	Static	397.66	0.96
18000	Dynamic	291.72	0.80
20000	Dynamic	324.66	0.93
22000	Dynamic	357.6	0.99
24480	Dynamic	393.78	1.08

At each matrix size, the performance of both the static and dynamic schedules are within 1% of each other, well within a natural variation of runtimes. Thus, in each case, the energy saved, over 11% for the largest three sizes, is free energy. From the table it is clear that the dynamic schedule is the more efficient means of running. For the 20,0000, 22,000, and 24,480, the dynamic schedule gains 0.12 GFlops/Watt, which is at minimum a 12.5% efficiency gain. The dynamic schedule is more energy efficient without having to sacrifice any other beneficial runtime qualities.

V. Conclusion

From the significant performance increase and energy efficiency found using GPUs for LU decomposition and the findings of [1], [2], and [3], there is strong evidence that GPUs are an excellent solution for achieving high, energy efficient performance. The heterogeneous approach is able to obtain over 20x the GFlops and less than $\frac{1}{15}$th the energy of a CPU-only approach. For optimization, using a GPU is essential for this application. The development of DVFS is also a useful tool in obtaining a lower energy consumption. It allows to take advantage of the times for which each frequency on the computing resources is the most efficient. During heavy workloads, a dynamic schedule can use the highest frequency to obtain the most performance and energy efficiency, and can switch to lower speeds during light or empty workloads to save, rather than waste, energy. For the larger matrix sizes tested, there is about an 11% savings in the amount of energy used. All energy conserved came without cost to performance, which is critical to the goal of this research. A majority of these savings come from scaling down the GPU when not in use, which is no surprise considering this is where the majority of the work is done. High performance computing can have continued progress in speed by using GPUs as a mainstream computing resource and by using DVFS to schedule all resources at high speeds only when working and low speeds when idle to achieve substantial energy savings.

References

[1] Dong, T (14 Nov. 2012). "MAGMA: A New Generation of Linear Algebra Library for GPU and Multicore Architectures." Innovative Computing Laboratory

[2] Kim, Hyesoon, Chi-Keung Luk, and Sunpyo Hong. (2009). "Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping" Atlanta, GA: Georgia Institute of Technology, Tech. no. TR-2009-001.

[3] Liu, Qiang, and Wayne Luk. (2012). "Heterogeneous Systems for Energy Efficient Scientific Computing." *International Conference on Reconfigurable Computing*, Hong Kong, China. N.p.: Springer, 64-75. EPICS.

[4] Simon, Horst. (2013). *Why We Need Exascale and Why We Won't Get There by 2020* Proc. of Optical Interconnects Conference, Santa Fe, New Mexico. Berkeley, CA: Berkeley Lab, 1-27.