Preliminary Results of Autotuning GEMM Kernels for the NVIDIA Kepler Architecture – GeForce GTX 680

– LAPACK Working Note 267

Jakub Kurzak[1]
Piotr Luszczek[1]
Stanimire Tomov[1]
Jack Dongarra[1,2,3]

[1]*Electrical Engineering and Computer Science, University of Tennessee*
[2]*Computer Science and Mathematics Division, Oak Ridge National Laboratory*
[3]*School of Mathematics & School of Computer Science, University of Manchester*

Abstract

Kepler is the newest GPU architecture from NVIDIA, and the GTX 680 is the first commercially available graphics card based on that architecture. Matrix multiplication is a canonical computational kernel, and often the main target of initial optimization efforts for a new chip. This article presents preliminary results of automatically tuning matrix multiplication kernels for the Kepler architecture using the GTX 680 card.

1 Introduction

Accelerators are gradually making their way into the world of *High Performance Computing* (HPC). Before graphics cards entered the arena, the IBM Cell processor made a brief appearance in HPC (from 2006 to 2009), culminating in the deployment of the Roadrunner supercomputer at the Los Alamos National Laboratory, which secured the first place on the TOP500 list in June 2008, as the first machine to cross the performance of one PetaFLOPS. However, development of the Cell architecture was canceled by IBM in November 2009 and, in the meantime, graphics cards have been gradually turning towards general programmability and steadily gaining ground in HPC. The introduction of the *Compute Unified Device Architecture* (CUDA) by NVIDIA in November 2006 was a significant milestone towards adoption of GPUs for general purpose computing. Another one was the introduction of the GF100 (Fermi) architecture in April 2010, with an array of essential numerical computing features. These developments culminated in the deployment of the Tianhe-1A supercomputer at the National Supercomputer Center in Tianjin,

China, which secured the first place on the TOP500 list in November 2010, with the performance of 2.5 PetaFLOPS. The next major development on the horizon is the deployment of the Titan supercomputer at the Oak Ridge National Laboratory at the end of this year (2012). Titan is expected to reach unprecedented performance by leveraging the power of the new NVIDIA's GK104 (Kepler) architecture.

Matrix multiplication is a classic computational workload, a common performance benchmark and a usual target of early optimization efforts for every new architecture. The kernel is also frequently referred to as `GEMM`, following the *Basic Linear Algebra Subroutines* (BLAS) standard [1]. Matrix multiplication can be trivially defined with three nested loops, and most of the time, its performance optimization relies on classic techniques, such as loop tiling and loop unrolling. Despite that, optimizing matrix multiplication remains a challenge and a common target of academic and commercial software efforts. Not without significance is the fact that the `GEMM` routine is critical to the performance of the *High Performance LINPACK Benchmark* (HPL) [2], the official benchmark of the TOP500 list. At the same time, the value of fast matrix multiplication goes beyond benchmarking. Performance of many indispensable software packages is closely tied to the performance of `GEMM`, especially packages for solving problems in dense linear algebra, such as LAPACK [3] and ScaLAPACK [4] and packages for solving problems in sparse linear algebra using direct methods, such as MUMPS [5] and SuperLU [6].

2 Motivation

This article is a follow-up to a previous paper by the same authors, where the autotuning framework was introduced called *Automatic Stencil TuneR for Accelerators* (ASTRA) [7]. ASTRA's prototype was developed on the Fermi architecture and used to produce fast BLAS-compliant `GEMM` kernels, covering all precisions (single, double / real, complex) and all cases of transposed inputs (A: *Trans, NoTrans, ConjTrans*; B: *Trans, NoTrans, ConjTrans*). At the time of ASTRA's conception, the Fermi architecture has already been available for more than a year, and `GEMM` kernels in CUBLAS and MAGMA were already delivering very good performance. As a result, ASTRA produced only marginal performance improvements in most cases, with the most noticeable effect on the double-complex (`ZGEMM`) kernels, where the performance was bumped from ∼300 GigaFLOPS to ∼340 GigaFLOPS (ca. 13%). However, the true motivation for ASTRA was the capability to quickly deploy fast `GEMM` kernels, as well as other BLAS kernels, when a new architecture becomes available. Therefore, the availability of the Kepler architecture creates the first true opportunity for validation of the ASTRA methodology.

3 Target Hardware

The GF100 (Fermi) was the first graphics architecture with a complete set of essential features for scientific and engineering computing. The most important feature, from the standpoint of numerical computing, is double precision performance on a par with single precision performance. Double precision operations consume twice the storage of single precision operations (two 32-bit registers per element) and execute at half the throughput of single precision operations (16 operations per multiprocessor per cycle), which is the desired behavior. The *Fused Multiply-Add* (FMA) operation is available, and offers extra precision over the *Multiply-Add* (MADD) operation. Also, the floating-point hardware supports *denormalized* numbers and all four IEEE 754-2008 rounding modes (nearest, zero, positive infinity, negative infinity). Finally, the memory system provides *Error Correction Code* (ECC) protection against bit flips by radiation, which makes it suitable for large installations with thousands of cards, servicing long production runs. The full-featured Fermi chips are available in the Tesla line of products (Tesla 2050, 2070, 2090). At the same time, Fermi chips stripped down from some of those features are available in the GeForce 400 line of cards for video games (GTX 460, 480, 580). For the most part, the gaming cards offer a much lower double precision performance.

The GK104 (Kepler) is the newest GPU architecture from NVIDIA. As a successor of Fermi, Kepler is meant to preserve the array of essential HPC characteristics. As usually, the first products to enter the market are the gaming cards of the GeForce 600 series, specifically the GTX 680. Although the chip is not identical to the one that will eventually target the HPC market, its availability creates an exciting opportunity to evaluate the new architecture. It also provides a unique chance to validate an autotuning system, where the adaptation to a new architecture is the *Holy Grale*.

The list of most important changes from the Fermi architecture to the Kepler architecture can be built by consulting the *"NVIDIA GF100"* whitepaper [8], the *"Fermi"* whitepaper [9], the *"NVIDIA GeForce GTX 680"* whitepaper [10] and the *"NVIDIA CUDA C Programming Guide, Version 4.2"* (Appendix F) [11]. Table 1 presents a rough comparison of selected features. Kepler is designed to run with clock rates ∼35% lower than the maximum clock rate of Fermi, but contains three times the number of CUDA cores, which results in doubling of the peak single precision floating point performance. At the same time, Kepler consumes ∼20% less energy (judging by *thermal design power*), which means that Kepler is almost 2.5 times more energy efficient, in terms of peak performance per Watt. The number of multiprocessors in Kepler is lower (8 instead of 16), but they are have much more CUDA cores (192 instead of 32). They contain twice the number of registers and

the same amount of shared memory. L2 cache is slightly smaller (512 KB instead of 768 KB) and slightly faster (512 bytes per clock instead of 384 bytes per clock). The peak memory bandwidth is the same.

Table 1: Brief summary of architectural changes from Fermi to Kepler.

	GF100(Fermi)	GK104(Kepler)
MULTIPROCESSORS	16	8
CUDA CORES PER MULTIPROCESSOR	32	192
CUDA CORES TOTAL	512	1536
CLOCK RATE	1544 MHz	1006 MHz
SINGLE PRECISION FLOATING POINT PEAK	1581 GFLOPS	3090 GFLOPS
REGISTERS PER MULTIPROCESSOR	32 K	64 K
SHARED MEMORY + L1 PER MULTIPROCESSOR	64 K	64 K
L2 CACHE SIZE	768 KB	512 KB
L2 CACHE BANDWIDTH	384 B per clock	512 B per clock
MEMORY BANDWIDTH PEAK	192.4 GBPS	192.4 GBPS
TRANSISTORS COUNT	3.0 B	3.54 B
THERMAL DESIGN POWER	244 W	195 W

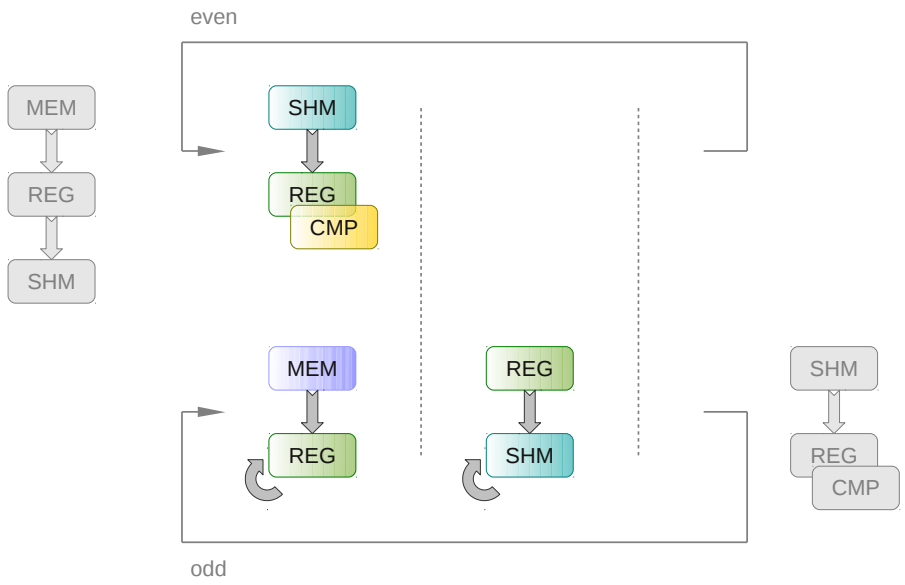
Interestingly, the number of transistors in Kepler (3.0 billion) is only slightly larger than in Fermi (3.5 billion). Despite that, the total number of CUDA cores tripled. Within the multiprocessor, the number of CUDA cores increased sixfold, while the number of registers doubled and the combined size of shared memory and L1 cache remained the same. Combined with the fact that the bandwidth of the device memory remained the same, this slightly shifts the balance of the architecture towards computational intensity.

4 Kernel Stencil

The kernel of interest here is the `GEMM` operation, as defined by the BLAS standard, expressed by the general formula $C = alpha\, C + beta\, A \times B$, where A, B and C are matrices of sizes $m \times k$, $k \times n$, and $m \times n$ respectively, in one of the four BLAS precisions (S, C, D, Z), and either of the matrices A and B can be transposed. The CUDA *stencil* for the kernel solidified on the Fermi architecture. Figure 1 shows the general structure of the kernel's pipelined loop. The loop's prologue and epilogue is marked with faded boxes, and the loop's steady state is marked with colored boxes. In this kernel, the data always passes through shared memory, what relieves

the stress on the device memory and allows for efficient handling of transpositions. In the past, solutions were presented, where only one input passed through shared memory. This strategy worked well for older generation of NVIDIA cards [12], as well as recent generation of AMD cards [13]. Nothing indicates, tough, that bypassing shared memory can deliver good performance on the Kepler card. At the same time, it severely complicates the handling of transposed inputs, therefore passing both inputs through shared memory remains the solution of choice here.

Figure 1: The structure of the `GEMM` kernel pipelined loop.



The principles of operation are as follows. First, the prologue loads the first tile of A and B to shared memory. The data is loaded to registers and deposited in shared memory. Then the code enters the steady-state loop. The loop has two stages, separated by `__syncthreads()` calls (barriers). In the first one, the data in shared memory is loaded to registers and used for calculations. At the same time, new tiles of A and B are being fetched. In the second step, the newly fetched tiles are dropped to shared memory, which is than consumed as processing transitions back to step one. The code follows the classic scheme of double-buffering, where computation can be overlaid with data fetches. Here, two sets of registers are used to concurrently issue data reads from memory to one set of registers, while computing the solution using another set of registers. On the other hand, only one

piece of shared memory is used, forcing `__syncthreads()` between step one and two. This is because shared memory is a scarce resource and increasing shared memory usage decreases occupancy deteriorating the performance.

The kernel is expressed as a single, heavily parametrized, source file in CUDA. Specifically, all blocking sizes are parametrized. This includes tiling for shared memory and shape of the thread block. Also parametrized is the precision (single / double, real / complex), and so are transpositions of the input matrices (*Trans, NoTrans, ConjTrans*). Altogether, the code can be compiled to 78 cases of different precisions and transpositions, and an endless number of cases with different tilings.

5 Prunning Settings

The objective of pruning is to make the number of tested kernels manageable. Specifically, the process is not supposed to identify the kernels that will run well, but rather discard the kernels that are certain to not run well. The pruning engine applies straightforward heuristics to eliminate kernels, which are very unlikely to produce high performance. First of all, the kernel cannot exceed hardware capabilities, such as the number of available threads, the number of available registers, and the size of the shared memory. Kernels which do, are immediately discarded. Second, it is checked if tiling matches the shaped of the thread block, i.e., if the thread block can be shaped such that tile dimensions are divisible by thread block dimensions. It is also checked if the number of threads in a block is divisible by the warp size. Finally, three heuristic constraints are applied to further narrow the focus of the search:

- **minimum occupancy:** minimum number of threads per multiprocessor,
- **minimum register reuse:** number of FMAs per load in the innermost loop,
- **minimum number of thread blocks per multiprocessor**.

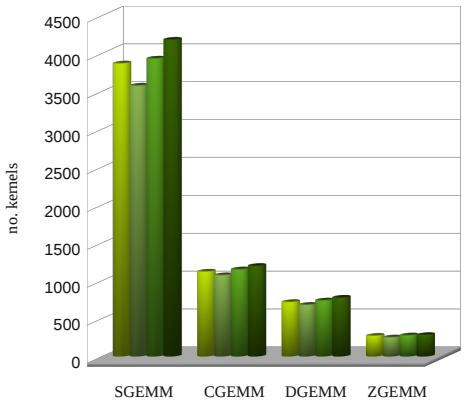
Currently, the choice of values for the heuristic parameters is arbitrary and more of an art than a science. For the most part, they are set according to the developer's intuition and in such a way that the number of kernels passing through the selection is manageable. Thousands of kernels can be benchmarked on a single card within a few hours. In the future, this arbitrary selection can be replaced by a more systematic process using, e.g. machine learning techniques. Another solution is to relax all constraints to the maximum and perform the search in parallel, using potentially hundreds of cards.

Figure 2 shows the selection of the heuristic pruning parameters and the number of kernels passing through the selection. For the most part, setting the occupancy to 1024 (half of the hardware maximum) and the register reuse to 2.0,

produced the desired number of kernels for single precision real and complex arithmetic. The register reuse had to be relaxed to 1.0 for double-real kernels and the occupancy had to be relaxed to 768 for double-complex kernels. The number of thread blocks per multiprocessor did not play a role here.

Figure 2: Numbers of tested kernels.

	min. occupancy	min. reuse	min. blocks	no. kernels
SGEMM $A \times B$	1024	2.0	1	3864
SGEMM $A \times B^T$	1024	2.0	1	3571
SGEMM $A^T \times B^T$	1024	2.0	1	3929
SGEMM $A^T \times B$	1024	2.0	1	4178
CGEMM $A \times B$	1024	2.0	1	1115
CGEMM $A \times B^T$	1024	2.0	1	1068
CGEMM $A^T \times B^T$	1024	2.0	1	1145
CGEMM $A^T \times B$	1024	2.0	1	1191
DGEMM $A \times B$	1024	1.0	1	713
DGEMM $A \times B^T$	1024	1.0	1	676
DGEMM $A^T \times B^T$	1024	1.0	1	735
DGEMM $A^T \times B$	1024	1.0	1	771
ZGEMM $A \times B$	768	1.0	1	267
ZGEMM $A \times B^T$	768	1.0	1	246
ZGEMM $A^T \times B^T$	768	1.0	1	273
ZGEMM $A^T \times B$	768	1.0	1	279



The much smaller number of kernels produced for double precision comes from the fact that, as the size of the data element becomes larger, the possibilities for arranging the data and the threads become much more constrained. On the other hand, the incentives for tuning double precision are much smaller, as the double precision runs much slower, and therefore it is much easier to achieve its peak. Plus, tuning double precision for the GTX 680 card is not the true objective here, since this is not the card which will eventually target the HPC market.

6 Results

All experiments were done on a EVGA GTX 680 card with 1 GHz clock and 2 GB of memory. CUBLAS library form SDK 4.2.6 (release candidate) was used, as it was producing the fastest performance numbers (faster than SDK 4.1). On the other hand, NVCC compiler from SDK 4.1.28 (`-arch sm_20`) was used to compile ASTRA kernels, as it was producing faster kernels than the NVCC compiler from SDK 4.2.6 (`-arch sm_30`). The autotuning sweep was done for matrices A, B and C of size 4096×4096, and all presented performance numbers are for that size.

Figures 3 and 4 show the performance for single precision, in real and complex arithmetic respectively. Execution rates are shown along with the best tiling sizes for ASTRA. ASTRA delivers performance comparable to CUBLAS for `SGEMM`

Figure 3: SGEMM performance (bright: CUBLAS, dark: ASTRA).

SGEMM	CUBLAS perf.	ASTRA perf.	ASTRA tiling*			
$A \times B$	1101	1125	128×224×32	32×32	64×16	32×32
$A \times B^T$	1254	1151	64×64×6	16×8	64×2	64×2
$A^T \times B^T$	896	1097	224×128×32	32×32	32×32	64×16
$A^T \times B$	1230	1146	192×160×32	32×32	32×32	32×32

* shared memory tiling (the first triplet), followed by the shape of the thread block for: computing C, reading A and reading B.

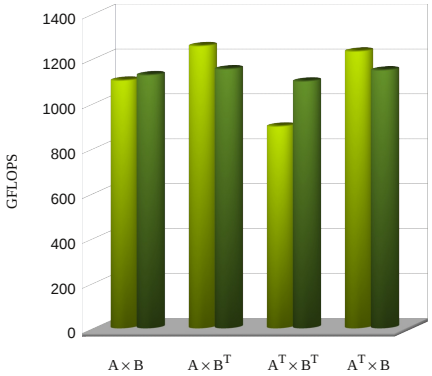
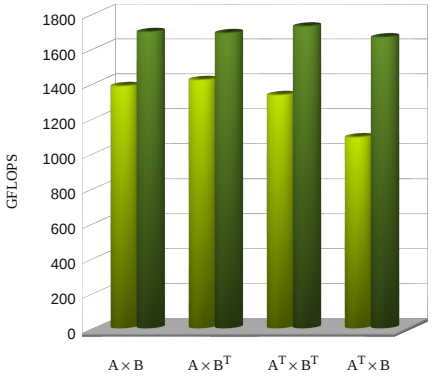


Figure 4: CGEMM performance (bright: CUBLAS, dark: ASTRA).

CGEMM	CUBLAS perf.	ASTRA perf.	ASTRA tiling			
$A \times B$	1383	1693	48×72×12	16×18	48×6	4×72
$A \times B^T$	1419	1687	48×72×12	12×24	24×12	72×4
$A^T \times B^T$	1332	1725	48×72×12	16×18	12×24	72×4
$A^T \times B$	1090	1663	64×96×16	16×32	16×32	16×32



and substantially higher than CUBLAS for CGEMM. At the same time, ASTRA's performance is oblivious to the different transposition cases, while CUBLAS performance is more affected by input transpositions. CUBLAS performance problems come most likely from a suboptimal choice of tiling, which can be quickly fixed by adopting ASTRA's tiling sizes. On the other hand, ASTRA's performance problems may reside in the compiler back-end and not be easily fixable at the PTX level. The fact that ASTRA's performance changes substantially depending on the version of the NVCC compiler seem to suggest that this is the case.

Double precision is penalized on the Kepler card for video games, so it is not very insightful to tune double precision, but since there is no extra effort in autotunig double precision kernels, they are presented here for completeness. Figures 5

Figure 5: `DGEMM` performance (bright: CUBLAS, dark: ASTRA).

DGEMM	CUBLAS perf.	ASTRA perf.	ASTRA tiling			
$A \times B$	123	131	32×64×8	8×32	32×8	4×64
$A \times B^T$	120	137	32×64×16	16×32	32×16	64×8
$A^T \times B^T$	124	128	16×32×8	8×16	8×16	32×4
$A^T \times B$	121	131	32×64×8	8×32	8×32	8×32

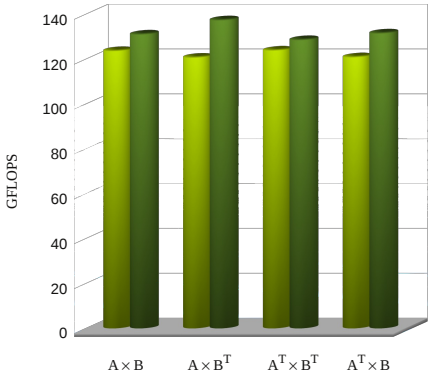
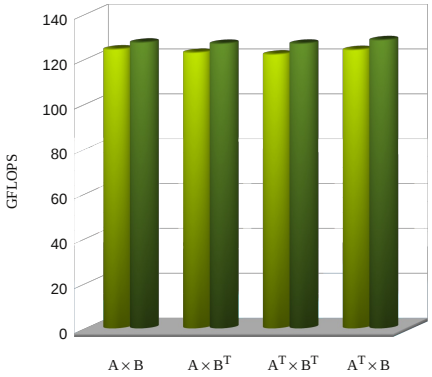


Figure 6: `ZGEMM` performance (bright: CUBLAS, dark: ASTRA).

ZGEMM	CUBLAS perf.	ASTRA perf.	ASTRA tiling			
$A \times B$	124	137	16×32×16	8×32	16×16	8×32
$A \times B^T$	123	127	16×32×16	8×32	16×16	32×8
$A^T \times B^T$	122	127	16×32×16	8×32	16×16	32×8
$A^T \times B$	124	128	16×16×8	8×16	8×16	8×16



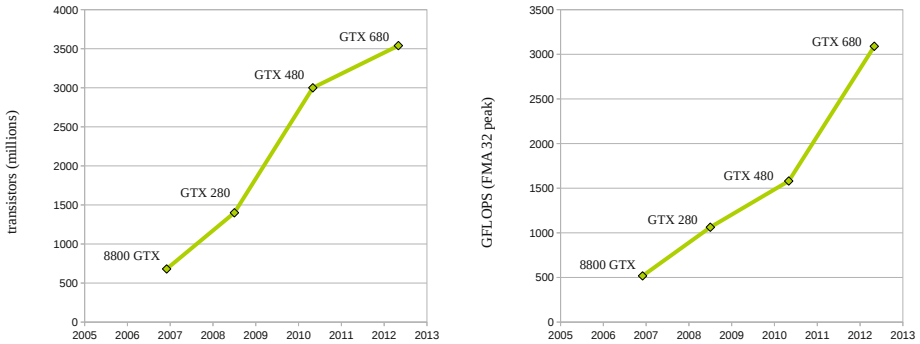
and 6 show the performance for double precision, in real and complex arithmetic respectively. Here, achieving the floating point peak is relatively easy, since the much slower floating point units come nowhere near saturation of the memory bandwidth. Here ASTRA is only marginally faster than CUBLAS, but nevertheless never slower. One can observe that this miserable performance for a GPU would be a very respectable performance for a standard multicore CPU (x86 and alike).

7 Summary

The authors would like to believe that ASTRA could play a similar role for GPUs as ATLAS plays for CPUs, i.e., allow for relatively fast deployment of BLAS (and hopefully other types of routines) for new GPU architectures. Similarly to ATLAS, it is not ASTRA's goal to seek optimizations at the lowest hardware level, but instead provide good baseline performance, upon which vendor libraries can improve, by applying low-level optimizations (instruction scheduling, etc.) The Kepler experiments prove ASTRA's capability to play such role.

A Hardware Trends

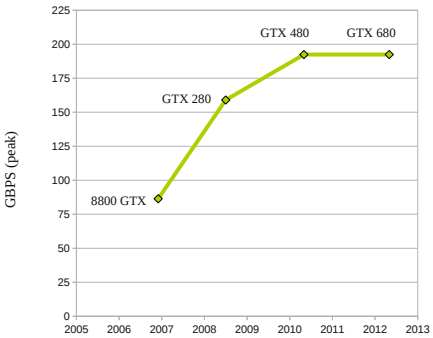
Figure 7: Transistors trend (left) and GLOPS trend (right).



In the age of dramatic shifts in the hardware world, it is very popular to plot changes from one generation of chips to another. Since we have not come across trends that would include the Kepler chip, we take the liberty of presenting such trends here (Figures 7 and 8).

The Kepler GPU made a phenomenal jump in the "raw" floating point capabilities by doubling the peak floating point performance of the Fermi GPU. This changes the previous linear trend to an exponential trend. At the same,

Figure 8: Memory bandwidth trend.



time the Kepler GPU discontinued the exponential growth in the number of transistors, and gave the trend the shape of an *S curve*. And finally, the bandwidth trend shows that the memory bandwidth completely reached its plateau.

References

[1] Basic Linear Algebra Technical Forum. *Basic Linear Algebra Technical Forum Standard*, August 2001. http://www.netlib.org/blas/blast-forum/blas-report.pdf.

[2] J. J. Dongarra, P. Luszczek, and A. Petitet. The LINPACK benchmark: Past, present and future. *Concurrency Computat.: Pract. Exper.*, 15(9):803–820, 2003. DOI: 10.1002/cpe.728.

[3] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. W. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, 1992. http://www.netlib.org/lapack/lug/.

[4] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA, 1997. http://www.netlib.org/scalapack/slug/.

[5] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods in Appl. Mech. Eng.*, 184(2-4):501–520, 2000. DOI: 10.1016/S0045-7825(99)00242-X.

[6] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM. J. Matrix Anal. & Appl.*, 20(3):720–755, 1999. DOI: 10.1137/S0895479895291765.

[7] J. Kurzak, S. Tomov, and J. Dongarra. LAPACK working note 245: Autotuning GEMMs for Fermi. Technical Report UT-CS-11-671, Electrical Engineering and Computer Science Department, University of Tennessee, 2011. http://www.netlib.org/lapack/lawnspdf/lawn245.pdf (accepted to IEEE TPDS).

[8] NVIDIA Corporation. *NVIDIA GF100: Worlds Fastest GPU Delivering Great Gaming Performance with True Geometric Realism, Version 1.5*, 2010. http://www.nvidia.com/object/IO_89569.html.

[9] NVIDIA Corporation. *NVIDIAs Next Generation CUDA Compute Architecture: Fermi, Version 1.1*, 2009. http://www.nvidia.com/object/IO_89570.html.

[10] NVIDIA Corporation. *NVIDIA GeForce GTX 680: The fastest, most efficient GPU ever built, Version 1.0*, 2012. http://www.geforce.com/Active/en_US/en_US/pdf/GeForce-GTX-680-Whitepaper-FINAL.pdf.

[11] NVIDIA Corporation. *NVIDIA CUDA C Programming Guide, Version 4.2*, 2012. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf.

[12] V. Volkov and J. W. Demmel. Benchmarking GPUs to tune dense linear algebra. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC'08*, Austin, TX, November 15-21 2008. IEEE Press. DOI: 10.1145/1413370.1413402.

[13] N. Nakasato. A fast GEMM implementation on a Cypress GPU. In *1st International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems, PMBS'10 (held as part of SC'10)*, New Orleans, LA, November 13-19 2010. http://www.dcs.warwick.ac.uk/~sdh/pmbs10/pmbs10/Workshop_Programme_files/fastgemm.pdf.

ACKNOWLEDGEMENT

The authors are grateful to Wayne McKinney and Peter Takacs for very useful discussions. The Advanced Light Source is supported by the Director, Office of Science, Office of Basic Energy Sciences, Material Science Division, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 at Lawrence Berkeley National Laboratory.

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor The Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or The Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or The Regents of the University of California.