International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2011)
Rio de Janeiro, RJ, Brazil, May 8-12, 2011, on CD-ROM, Latin American Section (LAS) / American Nuclear Society (ANS)
ISBN 978-85-63688-00-2

COMPUTATION OF LARGE COVARIANCE MATRICES BY SAMMY ON GRAPHICAL PROCESSING UNITS AND MULTICORE CPUs

G. Arbanas, M.E. Dunn, and D. Wiarda
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6171, U.S.A.
arbanasg@ornl.gov; dunnme@ornl.gov; wiardada@ornl.gov

ABSTRACT

Computational power of Graphical Processing Units and multicore CPUs was harnessed by the nuclear data evaluation code SAMMY to speed up computations of large Resonance Parameter Covariance Matrices (RPCMs). This was accomplished by linking SAMMY to vendor-optimized implementations of the matrix-matrix multiplication subroutine of the Basic Linear Algebra Library to compute the most time-consuming step. The ^{235}U RPCM computed previously using a triple-nested loop was re-computed using the NVIDIA implementation of the subroutine on a single Tesla Fermi Graphical Processing Unit, and also using the Intel's Math Kernel Library implementation on two different multicore CPU systems. A multiplication of two matrices of dimensions 16,000$\times$20,000 that had previously taken days, took approximately one minute on the GPU. Comparable performance was achieved on a dual six-core CPU system. The magnitude of the speed-up suggests that these, or similar, combinations of hardware and libraries may be useful for large matrix operations in SAMMY. Uniform interfaces of standard linear algebra libraries make them a promising candidate for a programming framework of a new generation of SAMMY for the emerging heterogeneous computing platforms.

Key Words: GPU, resonance, covariance, matrix, multiply, SAMMY

1. INTRODUCTION

Computer code SAMMY [1] is frequently used for evaluation of experimental data in the resolved and unresolved resonance energy regions. SAMMY employs a generalized least-squares algorithm for fitting R-matrix resonance parameters to data, while accounting for experimental conditions. Uncertainties in the experimental data, consisting of systematic uncertainties on measurement-related quantities (such as normalizations, backgrounds, time-of-flight, sample thickness, and others) are incorporated in the fitting procedure. The result of SAMMY evaluation is a set of R-matrix resonance parameters and a corresponding Resonance Parameter Covariance Matrix (RPCM) that provide the best generalized least-squares fit to experiments and an accurate representation of the uncertainties of those experiments, respectively.

Computation of RPCMs for the largest sets of resonance parameters and data may take a long time when executed serially on a single CPU. For example, a retroactive computation of a RPCM for the 15,965 resonance parameters of ^{235}U originally took four weeks [3]. A performance analysis indicated that the vast majority of the time was consumed by a matrix-matrix multiplication, programmed as a triple-nested Fortran DO-loop with a nesting order out of harmony with the column-major ordering of Fortran. It is estimated that a corrected nesting order would have shortened the execution time down to a few days. Although this would constitute a several-fold improvement, better performance was needed. A need for better performance motivated this exploration of vendor-optimized implementations of the standard matrix-matrix multiplication routines of the Basic Linear Algebra Library (BLAS).

2. HARDWARE AND SOFTWARE

Computations on the Tesla Fermi GPU were performed using the NVIDIA CUBLAS 3.2, while the computations on the two multicore systems, labeled Dell and Mac, respectively, were performed using the Intel Math Kernel Library (MKL) distributed with the Intel ifort Fortran compiler. The GPU was installed on the Dell, which also served as a stand-alone multicore CPU system. The Mac listed in the table served as another multicore CPU workstation. Some technical details of the hardware and software used in this work are listed in Table I.

Table I: The hardware and software combinations used

Name	GPU	Dell	MacPro
Model	Tesla Fermi C2050	Xeon 5160	Xeon “Westmere”
# CPU	n/a	2	2
Total Cores	448	4	12
GHz	1.15	3.0	2.66
BLAS	CUBLAS 3.2	MKL	MKL
ifort -v	11.1	11.1	12.0.0
O.S.	RHEL 5	RHEL 5	OS X 10.6.4
RAM (GB)	2.1	28	32

The CUBLAS implementations transparently parallelized matrix-matrix multiplication across the 448 cores of the NVIDIA Tesla Fermi C2050 GPU, while the MKL implementation parallelizes it over all cores available on a given system. This suggests that the common BLAS library interface provides a straightforward and potentially long-term path towards transparent parallelization of computationally intensive large-matrix operations, without significant changes to the source code.

3. RETROACTIVE COVARIANCE MATRIX APPROACH

The retroactive covariance method is used to post-evaluate uncertainties by computing a RPCM for an existing set of evaluated resonance parameters. This method was used in [3] to compute a retroactive RPCM for resonance parameters of ^{235}U evaluated by [4]. The most time-consuming matrix multiplication appears in the right-hand side of the expression for the retroactive RPCM, labeled M':

$$M'^{-1} = G^T (V^{-1} G), \tag{1}$$

where G is the sensitivity matrix and V is a data covariance matrix. The dimensions of G are $d \times p$, where d is the number of data points, and p is the number of resonance parameters for which the retroactive RPCM is wanted. The right-hand side is computed for each data set, and therefore d varies among the sets, while the number of resonance parameters is held fixed at $p = 15,965$ for all data sets. The data covariance matrix V is of dimensions $d \times d$, and it happens to be diagonal for all data sets considered here. The inverse of a diagonal V is computed easily and so is $(V^{-1}G)$ of dimensions $d \times p$. The expression above can then be schematically written as

$$C = A^T \cdot B \tag{2}$$

where $C \equiv M'^{-1}$, $A \equiv G$, and $B \equiv (V^{-1}G)$ are the matrix labels used in the conventional definition of the BLAS matrix-matrix multiplication subroutine DGEMM. Since the number of parameters p in this work is held fixed, the computational cost of multiplying the above two matrices will be determined by the number of data points d in each set. Computation of the matrix inverse of M' is performed by SAMMY as before, since its execution time is relatively short.

4. NUMERICAL METHOD

The largest ^{235}U data set contains $d \approx 20,000$ data points, and it requires ≈ 8 GB of memory to multiply the two matrices in double-precision arithmetic. While the 28 and 32 GB RAM available on the Dell and Mac systems easily accommodates such multiplication by a single call to the Intel's MKL DGEMM subroutine, the 2.1 GB of memory[1] available on the NVIDIA Tesla Fermi C2050 GPU is insufficient to perform the same matrix multiplication in a single call to CUBLAS_DGEMM. To accommodate the computation within the 2.1 GB of the GPU memory, the two large matrices being multiplied were broken into four sub-matrices of nearly equal sizes; which sub-matrices could then be multiplied by a single call to CUBLAS_DGEMM due to their smaller size. The original multiplication

$$C = A^T \cdot B \tag{3}$$

was thus broken up into eight multiplications of smaller sub-matrices

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11}^T & a_{21}^T \\ a_{12}^T & a_{22}^T \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}^T b_{11} + a_{21}^T b_{21} & a_{11}^T b_{12} + a_{21}^T b_{22} \\ a_{12}^T b_{11} + a_{22}^T b_{21} & a_{12}^T b_{12} + a_{22}^T b_{22} \end{pmatrix}, \tag{4}$$

from which the complete product matrix C was then constructed in the RAM of the (Dell) host system. The addition of the products in the right-hand side of Eq. (4) was achieved by a feature

[1]This is the value of the maximum memory pitch reported by the deviceQuery program of the CUDA SDK.

of DGEMM whereby the prior contents of C can be added to the product, namely

$$C \to \alpha A^T \cdot B + \beta C. \tag{5}$$

A complete product matrix C is then computed by initiating $C \to 0$, setting $\alpha = \beta = 1$, and then calling DGEMM twice in succession for each of C's four sub-matrices. For the c_{11} sub-matrix, the two successive DGEMM calls are

$$\begin{aligned} c_{11} &\to a_{11}^T \cdot b_{11} + c_{11} \\ c_{11} &\to a_{21}^T \cdot b_{21} + c_{11}, \end{aligned} \tag{6}$$

while the analogous calls for the remaining three sub-matrices of C could be read off the right-hand side of Eq. (4). The result of the first DGEMM call $c_{11} = a_{11}^T \cdot b_{11}$ is thus added to $a_{21}^T \cdot b_{21}$ without any additional programming. The sub-matrices of A, B, and C are passed to the DGEMM subroutine via an array-slicing feature of Fortran 90. For example, the four sub-matrices of B are

$$\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} \texttt{B (1 : d/2 , 1 : p/2)} & \texttt{B (1 : d/2 , p/2 + 1 : p)} \\ \texttt{B (d/2 + 1 : d , 1 : p/2)} & \texttt{B (d/2 + 1 : d , p/2 + 1 : p)} \end{pmatrix}. \tag{7}$$

The method above was used for data sets with $7,000 < d < 20,000$. (For even larger data sets, this method could be repeated recursively until sub-matrices are sufficiently small to be multiplied in a single CUBLAS_DGEMM call [5, 6].)

For data sets with $3,000 < d < 7,000$, the matrices were broken into *two* sub-matrices instead of the four above, where the division was made along the p-dimension. The separation into two sub-matrices leads to four matrix-matrix multiplications, analogous to the eight above. For matrices smaller than $d < 3,000$ division into sub-matrices was not necessary. The GPU performance is optimal when the most GPU cores are utilized. This can be seen in the increasing performance with increasing d in each of the three ranges in Fig. 1. It also explains a discontinuous loss of performance as d crosses 3,000 and 7,000, where the number of sub-matrices doubles. This makes the sub-matrices smaller and therefore underutilizes the GPU's full capacity, causing a loss of performance.

5. RESULTS

A measure of performance of a matrix-matrix multiplication is established by dividing the number of floating point operations by the elapsed time. A multiplication of two matrices of dimensions $(d \times p)^T \cdot (d \times p)$, where d is the number of data points and $p = 15,965$ is the number of parameters, incurs $(2p^2d)$ operations. The measure of performance in Flops is then

$$\text{Flops} = \frac{2p^2 d}{\Delta t}, \tag{8}$$

where Δt is the difference in times returned by the `cpu_time()` before and after the multiplication. For ^{235}U data sets, Eq. (8) is plotted in Fig. 1. The figure shows the ranges in which the matrices are multiplied whole, or are broken into two, or four, smaller sub-matrices, to make their multiplication fit into a single CUBLAS_DGEMM call on the GPU. The performance of the CUBLAS_DGEMM subroutine in Fig. 1 indicates that SAMMY performance may approach

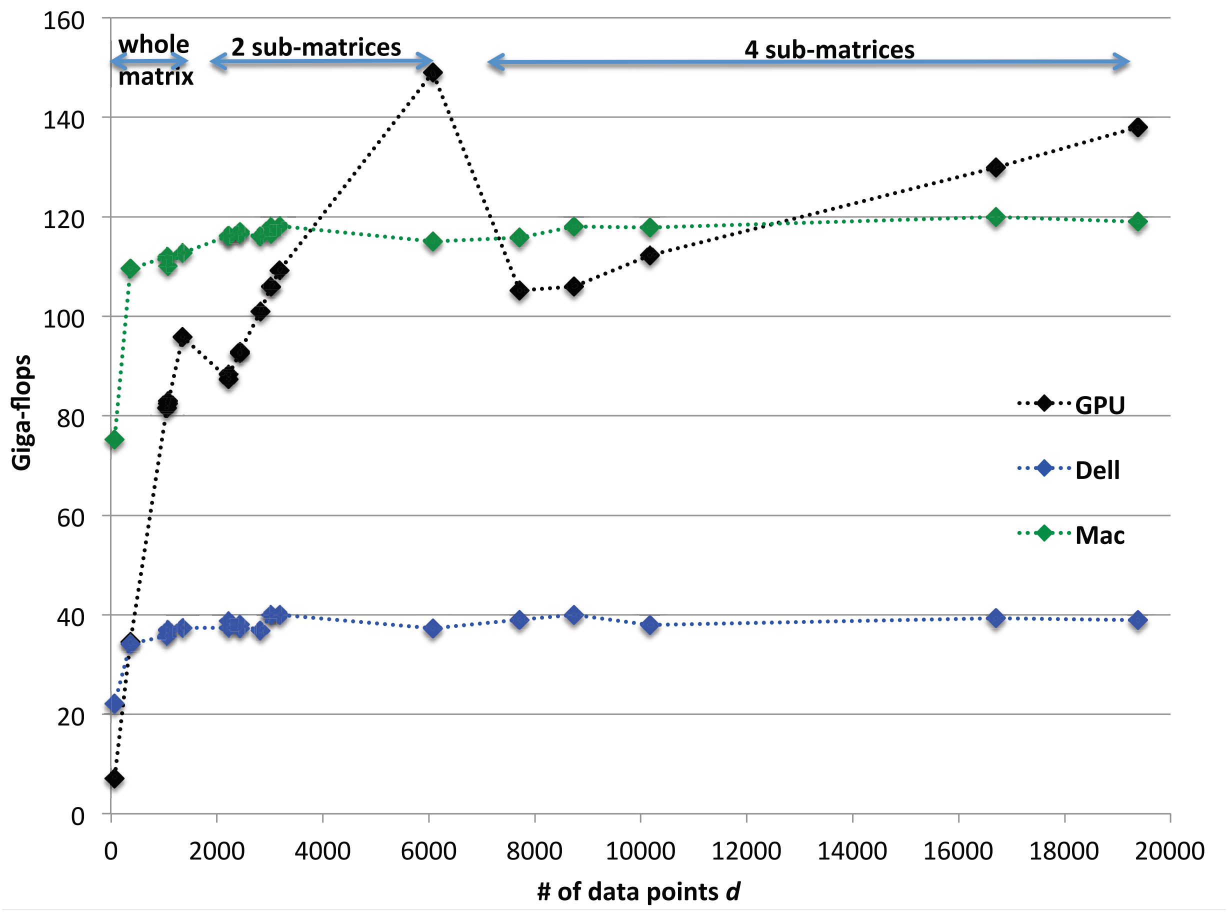

Figure 1: Performance of CUBLAS_DGEMM and DGEMM in SAMMY for all ^{235}U data sets. In contrast, the original triple-nested DO-loop multiplication of the largest matrices performed at ≈ 0.025 GFlops.

approximately 150 Gflops for the largest matrices considered, out of the maximum 240 Gflops achieved by a simple benchmarking program[2]. The 90 Gflops of the performance not utilized by SAMMY may be attributable to the overhead of a full-fledged SAMMY run, because this load is not taken into account by a simple matrix-matrix benchmarking program.

6. CONCLUSIONS AND OUTLOOK

Utilization of vendor-optimized double-precision matrix-matrix multiplication BLAS subroutine DGEMM in SAMMY has shortened the most time-consuming step in the computation of the ^{235}U retroactive RPCM from days down to one minute. NVIDIA's implementation of CUBLAS_DGEMM was used on the Tesla Fermi C2050 GPU and Intel MKL parallel implementation of DGEMM was used on the multicore CPU systems.

[2]A simple program for benchmarking large matrix-matrix multiplication achieved ≈240 Gflops on the GPU, which is 20% below the performance reported in [7].

Since the R-matrix formalism programmed in SAMMY can to a large degree be expressed in terms of linear-algebra identities (i.e., matrix and vector identities), the performance improvements described in this paper could be an impetus for optimizing other time-consuming matrix operations, by using BLAS or LAPACK routines. This could enable efficient evaluations of very large data sets and parameter sets that would otherwise be prohibitively time-consuming. Furthermore, a standardized interface of these libraries may provide a sound basis for a framework of the next generation of SAMMY. Such a framework would shield the SAMMY developers from the details of implementations, making it easier to design, and maintain its functionality on multiple platforms.

7. ACKNOWLEDGMENTS

The authors would like to thank K. Barker for installation of the NVIDIA Tesla Fermi C2050, and Dr. S. Goluoglu for hosting the GPU in his workstation.

REFERENCES

1. N.M. Larson. *Updated Users Guide for SAMMY: Multilevel R-Matrix Fits to Neutron Data Using Bayes Equations*. ORNL/TM-9179/R6, Oak Ridge National Laboratory, Oak Ridge, Tenn. (2003).
2. J.J. Dongarra, J. Du Croz, I.S. Duff, and S. Hammarling. Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Soft.* **16**, 18–28 (1990).
3. G. Arbanas, L.C. Leal, N.M. Larson, and H. Derrien. Retroactive Covariance Matrix for 235U in the Resolved-Resonance Region. *Proc. of American Nuclear Society Topical Meeting on Reactor Physics: Advances in Nuclear Analysis and Simulation, PHYSOR 2006*, paper C015, Vancouver, BC, Canada, September 10–14, 2006.
4. L.C. Leal, H. Derrien, N.M. Larson, and R.Q. Wright. *R-Matrix Analysis of ^{235}U Neutron Transmission and Cross Sections in the Energy Range 0 to 2.25 keV*. ORNL/TM-13516, Oak Ridge National Laboratory, Oak Ridge, Tenn. (1997).
5. J.A. Gunnels, F.G. Gustavson, K. Pingali, and K. Yotov. Is Cache-Oblivious DGEMM Viable? *Applied Parallel Computing. State of the Art in Scientific Computing, Lecture Notes in Computer Science*, **4699** (2007).
6. I. Girotto and Y. Yang. ICHECs GPU research: porting of scientific application on NVIDIA GPU. http://www.nvidia.com/content/GTC/posters/2010/I15-ICHECs-GPU-Research-Porting-of-Scientific-Application-on-NVIDIA-GPU.pdf (2010).
7. S. Tomov, G. Bosilca, and C. Augonnet. Accelerating Linear Algebra on Heterogeneous Architectures of Multicore and GPUs using MAGMA and DPLASMA and StarPU Schedulers. http://icl.cs.utk.edu/projectsfiles/magma/pubs/MAGMA_Tomov.pdf (2010).