Institute for Economic Forecasting, Romanian Academy
"Nicolae Titulescu" University from Bucharest, Romania
1st Workshop on Modelling and Economic Forecast
"New Trends in Modelling and Economic Forecast"
9-10 December 2011

CUDA BASED COMPUTATIONAL METHODS FOR MACROECONOMIC FORECASTS

Bogdan OANCEA[5],
Tudorel ANDREI[6],
Raluca DRAGOESCU[7]

Abstract: Parallel computing can offer an enormous advantage regarding the performance for very large applications in almost any field: economics, scientific computing, computer vision, databases, data mining. GPUs are high performance many-core processors that can obtain very high FLOP rates. Since the first idea of using GPU for general purpose computing, things have evolved and now there are several approaches to GPU programming: CUDA from NVIDIA and Stream from AMD. CUDA is now a popular programming model for general purpose computations on GPU for C/C++ programmers. In this paper we present an implementation of some iterative and direct linear systems solvers that use the CUDA programming model. Our CUDA library is used to solve macroeconometric models with forward-looking variables based on Newton method for nonlinear systems of equations. The most difficult step for Newton methods represents the resolution of a large linear system for each iteration. Our library implements LU factorization, Jacobi, Gauss-Seidel and non-stationary iterative methods (GMRES, BiCG, BiCGSTAB) using C-CUDA extension. We compare the performance of our CUDA implementation with classic programs written to be run on CPU. Our performance tests show speedups of approximately 80 times for single precision floating point and 40 times for double precision.

Keywords: parallel algorithms; macroeconometric models; rational expectations models; linear algebra; Krylov techniques;

JEL Classification: C01, C02, C53

1. INTRODUCTION

Advances in the computational power have a large influence on almost all fields of scientific computing. Although, during the last decade, microprocessors'

[5] "Nicolae Titulescu" University, Bucharest, Romania, email: oanceab@ie.ase.ro
[6] The Bucharest Academy of Economic Studies, Romania, email:andreitudorel@yahoo.com
[7] "Artifex" University, Bucharest, Romania, Email: ralucadragoescu@yahoo.com

performance has significantly increased and new architectures like multi-core processors has appeared, there are still problems that cannot be solved on a single desktop computer (Creel, 2008).

One of the fields that need a special attention is macroeconometric modeling. Macroeconometric models with forward-looking variables are a special class of models which involve very large systems of equations. The matrices resulting from these models could be so large that doesn't fit with the internal memory of a single desktop computer. For such models it is necessary to develop high performance parallel algorithms that can be run in parallel execution environments like parallel computers, clusters of workstations or grid environments.

A special kind of macroeconometric models are the rational expectations models (Fischer, 1992). These models contain variables that forecast the economic system state for the future periods t + 1, t + 2, ... , t + T, where T is the forecast time horizon. Depending on the size of the forecast time horizon, macroeconometric models with rational expectations could give raise to systems with tens or hundreds of thousands of equations.

For example, MULTIMOD model (Isard, P., 2000), (Masson, P, 1990) is a dynamic, annual forecast model designed by the International Monetary Fund that describes the economic behaviour of the whole world decomposed in 8 industrial regions and the rest of the countries. The model contains 466 equations. If we want to solve the model for a 30 years time horizon then we will have to solve a nonlinear system containing 13908 equations which is not a simple task nor for the most powerful workstations.

QPM (Quarterly Projection Model)(Armstrong, J, 1995) (Coletti, D., 1996) is a quarterly model developed by the Bank of Canada to obtain economic forecasts and as a research tool for the analysis of macroeconomic policies and economic equilibrium on long term. The QPM model has 329 nonlinear equations. The resolution of the model for a 30 years time horizon means to solve a system of 39480 equations.

FRB/US (Brayton, F., 1996, 1997) is a quarterly econometric model that describes the U.S. economy and has around 300 equations. An extension of this model is FRB/GLOBAL that describes the world economy using few thousands equations. The resolution of these models for a 20-30 years time horizon implies nonlinear systems with hundreds of thousands of equations.

Let's consider the general form of the nonlinear model with rational expectations:

$$h_i(y_t, y_{t-1}, \dots, y_{t-r}, y_{t+1|t-1}, \dots, y_{t+h|t-1}, z_t) = 0, \quad i = 1,\dots m$$

where $y_{t+j/t-1}$ is the expectation of y_{t+j} conditioned on the information available at the end of the period t-1 and z_t represents the exogenous and random variables. For consistent expectations, the forward expectations $y_{t+j/t-1}$ have to coincide with the next period's forecast when solving the model conditioned on the information available at the end of period t-1. These expectations are therefore linked in time and solving the model for each y_t conditioned on some start period 0 requires each

$y_{t+j|0}$ for j = 1,2, ... T-t and a final condition $y_{T+j|0}$, j = 1, 2 ..., h. Considering these equations for successive time periods a large nonlinear system of equations will result.

One of the first methods used to solve such models was the extended path algorithm proposed by Fair and Taylor (Fair, R.C., and J. B.Taylor 1983). They use Gauss-Seidel iterations to solve the model, period after period, for a given time horizon. The convergence of this method depends on the order of the equations. The endogenous forecast variables are considered as predetermined and then the model is solved period after period for a time horizon.

The solutions thus obtained represent the new values for the forecast variables. The process is repeated until the convergence is obtained. The advantage of this method is it's simplicity in implementation and the low storage requirements but this method has a main disadvantage: if the initial values for the endogenous variables are not "well" chosen, the convergence of the system is very poor or the system is not convergent at all.

An alternative method to solve the model is to build a system of equations written for successive periods t, t + 1, …, t + T¸ and to solve this system of nT nonlinear equations by one of the existing methods for nonlinear systems. Due to the large scale of the system, this method has been avoided in the past. Due to the recent advances in the parallel algorithms field it is now possible to solve such large scale systems with efficiency.

The Newton method applied to solve this model uses the following algorithm:

NEWTON Method
Given an initial solution y(0)
for k = 0,1,2, ... **until** convergence
Evaluate b(k) = - h(y(k),z)
Evaluate J(k) = ∂h(y(k),z)/∂y'
Solve J(k)s(k) = b(k)
y(k+1) = y(k) + s(k)

If the linear system $J(k)s(k) = b(k)$ is very large, the use of direct methods to determine the solution can be very expensive due to high memory requirements and computational cost.

This is a very good reason to develop high performance parallel algorithms as an attractive alternative to the classical serial algorithms. Another alternative to serial direct methods are the iterative methods which determine only an approximation of the solution, but this fact does not influence the convergence of the Newton method. These iterative algorithms can be parallelized too.

We will analyse high performance iterative and direct methods used to solve large linear systems that result by applying the Newton method, then we will describe an implementation of the parallel versions of such algorithms that we've developed using C-CUDA extension.

2. SERIAL ITERATIVE AND DIRECT METHODS FOR SOLVING LINEAR SYSTEMS

Stationary iterative methods such as Jacobi and Gauss-Seidel are well known and there are many textbooks that describe these methods [14]. For very large linear systems, the most appropriate iterative methods are the so-called Krylov techniques [22]. Contrary to stationary iterative methods such as Jacobi or Gauss-Seidel, Krylov techniques use information that changes from iteration to iteration. For a linear system $Ax = b$, Krylov methods compute the i^{th} iterate x(i) as :

$$x(i) = x(i-1) + d(i) \qquad i = 1, 2, \ldots$$

Operations involved to find the i^{th} update d(i) are only inner products, saxpy and matrix-vector products that has the complexity of $\Theta(n^2)$, so that Krylov methods are computational attractive comparing to the direct methods for linear systems.

Perhaps the best known of the Krylov' method is the conjugate gradient method. This method solves symmetric positive definite systems. The idea of the CG method is to update the iterates x(i) in a way to ensure the largest decrease of the objective function $\frac{1}{2}x'Ax - x'b$, while keeping the direction vectors d(i) A-orthogonal. This method can be implemented using only one matrix-vector multiplication per iteration. In exact arithmetic, the CG method gives the solution for at most n iterations. The complete description of the CG method can be found in (Golub, G. H, 1996).

Another Krylov method for general non symmetric systems is the Generalized Minimal Residuals (GMRES) introduced by (Saad, Y. 1996). The pseudo-code for GMRES is:

GMRES
Given an initial solution x(0) compute r = b – Ax(0)
$\rho = \|r\|_2$, v(1) = r/ ρ, β = ρ
for k = 1,2,... **until** convergence
  for j = 1,2, ... k,
    h(j,k) = (Av(k))'v(j)
  end
  $v(k+1) = Av(k) - \sum_{j=1}^{k} h(j,k)v(j)$
  $h(k+1,k) = \|v(k+1)\|_2$
  v(k+1,k) = v(k+1)/h(k+1,k)
endfor

The most difficult part of this algorithm is not to lose the orthogonality of the direction vectors v(j). To achieve this goal the GMRES method uses a Gram-Schmidt orthogonalization process. GMRES requires the storage and computation of an increasing amount of information, vectors v and matrix H. To overcome these

difficulties, the method can be restarted after a chosen number of iterations m. The current intermediate results are used as a new starting point.

Another Krylov method implemented by the authors is the BiConjugate Gradient method (Golub, G. H, 1996). BiCG uses a different approach based upon generating two mutually orthogonal sequences of residual vectors and A-orthogonal sequences of direction vectors. The updates for residuals and for the direction vectors are similar to those of the CG method, but are performed using A and its transpose. The disadvantage of the BiCG method is an erratic behaviour of the norm of the residuals and potential breakdowns. An improved version, called BiConjugate Gradient Stabilized BiCGSTAB, is presented below:

BiCGSTAB
Given an initial solution x(0) compute r = b – Ax(0)
$\rho_0 = 1$, ρ_1 = r(0)'r(0), $\alpha = 1$, $\acute{\omega} = 1$, p = 0, v = 0
for k = 1,2, ... **until** convergence
$\beta = (\rho_k/\ \rho_{k-1})(\alpha/\acute{\omega})$
p = r + β(p- ώv)
v = Ap
$\alpha = \rho_k/$(r(0)'v)
s = r – αv
t = As
ώ = (t's)(t't)
x(k) = x(k-1) + αp + ώs

For the BiCGSTAB method we need to compute 6 saxpy operations, 4 inner products and 2 matrix-vector products per iteration and to store matrix A and 7 vectors of size n. The computational complexity of the method is $\Theta(n^2)$ like the other Krylov methods.

The operation count per iteration cannot be used to directly compare the performance of BiCGSTAB with GMRES because GMRES converges in much less iterations than BiCGSTAB. We have implemented these iterative methods and run experiments to determine the possible advantages of them over the direct methods. The results of our experiments are presented in the next section.

The other alternative to solve a linear system $Ax = b$ is the direct method that consists in two steps:

- First, the matrix A is factorized, $A = LU$ where L is a lower triangular matrix with 1s on the main diagonal and U is an upper triangular matrix; in the case of symmetric positive definite matrices, we have $A = LL^t$.
- Second, we have to solve two linear systems with triangular matrices: $Ly = b$ and $Ux = y$.
- The standard LU factorization algorithm with partial pivoting is (Golub, G. H, 1996):

Right-looking LU factorization
for k =1:n-1 **do**
find ν with k≤ ν≤n such that $|A(\nu,k)| = \|A(k:n,k)\|_\infty$
A(k,k:n)↔A(ν, k:n)
p(k) = ν
if A(k,k) ≠ 0 **then**
A(k+1:n, k) = A(k+1:n,k)/A(k,k)
A(k+1:n,k+1:n) = A(k+1:n,k+1:n) - A(k+1:n, k) A(k, k+1:n)

The computational complexity of this algorithm is $\Theta(2n^3/2)$. After we obtain the matrix factors L and U we have to solve two triangular systems: $Ly = b$ and $Ux = y$. These systems are solved using forward and backward substitution that have a computational complexity of $\Theta(n^2)$, so the most important computational step is the matrix factorization. That's why we have to show a special attention to the algorithms for matrix factorization.

In practice, using actual computers with memory hierarchies, the above algorithm is not efficient because it uses only level 1 and level 2 BLAS operations (Lawson, C. L., et. al. 1979), (Dongarra, J.,1988). As it is well-known, level 3 BLAS operations (Dongarra, J.,1990) have a better efficiency than level 1 or level 2 operations. The standard way to change a level 2 BLAS operations into a level 3 BLAS operation is delayed updating. In the case of the LU factorization algorithm we will replace k rank-1 updates with a single rank-k update.

We present a block algorithm for LU factorization that uses level 3 BLAS operations. The $n \times n$ matrix A is partitioned as in Figure 1. The A_{00} block consists of the first b columns and rows of the matrix A.

A_{00}	A_{01}	=	L_{00}	0	*	U_{00}	U_{01}
A_{10}	A_{11}		L_{10}	L_{11}		0	U_{11}

Figure 1. Block LU factorization

We can derive the following equations starting from A=LU:

$$L_{00}U_{00} = A_{00} \quad (1)$$
$$L_{10}U_{00} = A_{10} \quad (2)$$
$$L_{00}U_{01} = A_{01} \quad (3)$$
$$L_{10}U_{01} + L_{11}U_{11} = A_{11} \quad (4)$$

Equations (1) and (2) perform the LU of the first b columns of the matrix A. Thus we obtain L_{00}, L_{10} and U_{00} and now we can solve the triangular system from equation (3) that gives U_{01}.

The problem of computing L_{11} and U_{11} reduces to compute the factorization of the submatrix $A_{11}' = A_{11} - L_{10}U_{01}$ that can be done using the same algorithm but with A_{11}' instead of A.

The block LU factorization algorithm can now be derived easily: suppose we have divided the matrix A in column blocks with b columns in each block. The complete block LU factorization algorithm is given below.

Block LU factorization
for k_b =1 to n-1 **step** b **do**
$b_f = \min(k_b + b - 1, n)$
{LU factorization of $A(k_b : n, k_b : b_f)$ with BLAS 2}
for $k = k_b$ **to** b_f **do**
find k such that $|A(k,i)| = \|A(i:n,i)\|_\infty$
if $i \neq k$ **then**
swap rows i and k
endif
A(i+1:n, i) = A(i+1:n, i)/A(i,i)
$A(i+1:n, i+1: b_f) = A(i+1:n, i+1: b_f) - A(i+1:n, i)\, A(i, i+1: b_f)$
endfor
{Let $\tilde{L}$ be unit lower triangular matrix $b \times b$ stored in $A(k_b : b_f, k_b : b_f)$}
Solve triangular systems $\tilde{L}\, Z = A(k_b : b_f, b_f + 1 : n)$
Update $A(k_b : b_f, b_f + 1 : n) \leftarrow Z$
{Delayed updating}
$A(b_f + 1 : n, b_f + 1 : n) = A(b_f + 1 : n, b_f + 1 : n) - A(b_f + 1 : n, k_b : b_f)\, A(k_b : b_f, b_f + 1 : n)$
endfor

The process of factorization is shown in Figure 2. The factorization of the current column block is done with the usual BLAS 2 operations and the active part of the matrix A will be updated with b rank-one updates simultaneously which in fact is a matrix-matrix multiplication (level 3 BLAS).

If $n >> b$ almost all floating point operations are done in the matrix-matrix multiplication operation.

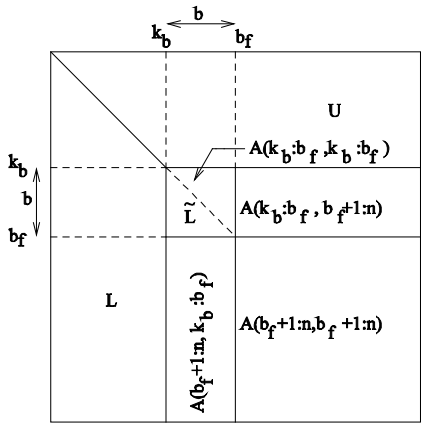

Figure 2. Block LU factorization with BLAS 3 operations

3. PARALLEL IMPLEMENTATION OF THE DIRECT AND ITERATIVE ALGORITHMS

In 2002 Mark Harris (Harris, Mark J., 2003) pointed out a new approach to obtain a high megaflop rate to the applications when he started to use GPUs (graphical processing unit) for non-graphics applications. Nowadays Graphics Processing Units contain high performance many-core processors capable of very high FLOP rates and data throughput being truly general-purpose parallel processors. Since the first idea of Mark Harris many applications were ported to use the GPU for compute intensive parts and they obtain speedups of few orders of magnitude comparing to equivalent implementations written for normal CPUs.

At this moment there are several models for GPU computing: CUDA (Compute Unified Device Architecture) developed by NVIDIA (NVIDIA, 2011), Stream developed by AMD (|AMD, 2008) and a new emerging standard, OpenCL (Khronos OpenCL Working Group, 2009) that tries to unify different GPU general computing API implementations providing a general framework for software development across heterogeneous platforms consisting of both CPUs and GPUs. We used the C CUDA extension to develop a library that implements iterative linear systems solvers.

We've used CUBLAS library in the implementation of the direct and iterative algorithms. Our library implements LU factorization as a direct method, Jacobi, Gauss-Seidel, CG, GMRES and BiCGSTAB iterative methods.

The general flow of the solver implemented in our library is:

- Allocate memory for matrices and vectors in the host memory;
- Initialize matrices and vectors in the host memory;
- Allocate memory for matrices and vectors in the device memory;
- Copy matrices from host memory to device memory;
- Define the device grid layout:

Number of blocks

Threads per block

- Execute the kernel on the device;
- Copy back the results from device memory to host memory;
- Memory clean up.

4. RESULTS

We've tested our direct and iterative solvers for both single precision and double precision floating point numbers. For our tests we used a computer with Intel Core2 Quad Q6600 procesor running at 2.4 Ghz, 4 GB of RAM and a NVIDIA GeForce GTX 280 graphics processing unit (GPU) with 240 cores running at 1296 MHz, 1GB of video memory and 141.7 GB/sec memory bandwith. The operating system used was Windows Vista 64 bit.

We compared the results obtained using the CUDA code with a single threaded C implementation run on CPU.

The CPU implementation of the direct and iterative algorithms used the optimized ATLAS (Whaley, R. C., 2001) library as a BLAS implementation. This gives better performances than a standard reference implementation of the BLAS.

Table 1 shows the speedup obtained by the C-CUDA implementation of the iterative solvers compared with the traditional CPU code for single precision floating point numbers and table 2 shows the speedup for double precision numbers.

From the results presented below one can see that GPU outperforms CPU for numerical computations.

Comparing the results for each method, it can be noticed that BiCGSTAB has better performances than the other methods.

For GMRES, in our experiments we restarted the method after 35 iterations. The tolerance for the solution was fixed at 10-4 for all methods.

For our experiments we have considered linear systems containing between 2000 and 20000 variables.

Table 3 shows the speedup of the CUDA implementation of the direct method for linear systems compared with a single threaded C implementation (the standard block-level implementation that can be found in LAPACK). We considered linear systems with 500 to 3500 equations.

Our performance results show the net advantage of GPU computing compared to the classical CPU code. The results also emphasize the advantage of the iterative solutions compared with the direct solution.

Another advantage of using CUDA programming model is that the code can be easier to read and support. The major drawback of CUDA is that it is only available for NVIDIA devices. A port of our library to OpenCL is intended for the future.

Table 1. Speed up for single precision FP

Matrix dimension	Speedup			
	Jacobi	Gauss-Seidel	GMRES(35)	BiCGSTAB
2000	67.4	69.3	78.3	82.2
4000	56.2	65.5	81.8	84.5
8000	68.3	67.4	80.1	81.9
12000	66.7	68.4	81.4	84.1
16000	71.1	69.2	79.3	86.0
20000	72.8	69.9	81.3	86.9

Table 2. Speed up for double precision FP

Matrix dimension	Speedup			
	Jacobi	Gauss-Seidel	GMRES(35)	BiCGSTAB
2000	35.2	36.1	39.6	41.7
4000	36.1	36.0	41.2	42.3
8000	29.1	35.2	41.6	43.6
12000	33.6	37.8	40.5	43.9
16000	32.3	35.9	42.8	44.0
20000	35.6	37.1	43.2	46.1

Table 3. The speedup of the direct method based on LU factorization

Matrix dimension	C-CUDA
500	8.99
1000	12.45
1500	11.41
2000	16.78
2500	16.23
3000	14.39
3500	15.92

5. CONCLUSIONS

We developed a C-CUDA library that implements the direct method with LU factorization and Jacobi, Gauss-Seidel and non-stationary iterative methods (GMRES, BiCGSTAB). The matrix-vector and matrix-matrix computations were done using CUBLAS routines. We compared the performance of our CUDA implementation with classic programs written to be run on CPU. Our performance tests show speedups of approximately 80 times for single precision floating point numbers and 40 times for

double precision for the iterative methods and about 10-15 for the direct method with double precision FP. These results show the immense potential of the GPGPU. In the future we intend to extend our direct and iterative solver library and to port it to OpenCL.

REFERENCES

1. AMD, ATI Stream Computing - Technical Overview. AMD, Tech. Rep. 2008
2. Anderson, E., Z. Bai, J. Demmel, J., Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. Mckenney, S. Ostrouchov, And D. Sorensen, LAPACK Users's Guide. SIAM, Philadelphia, 1992.
3. Armstrong, J., R. Black, D. Laxton, and D. Rose, "The Bank of Canada's New Quarterly Projection Model QPM. Part 2: A Robust Method for Simulating Forward-Looking Models", Technical Report No. 73, Ottawa: The Bank of Canada, 1995.
4. Black, R., D. Laxton, and R. Tetlow, "The Bank of Canada's New Quarterly Projection Model QPM. Part 1: The Steady-State Model", Technical Report No. 72, Ottawa: The Bank of Canada, November, 1994.
5. Brayton, F., and P. Tinsley, "A guide to FRB/US : A Macroeconomic Model of the United States", Technical Report, Finance and Economics Discussion Series, Federal Reserve Board, 1996.
6. Brayton, F., E. Mauskopf, D. Reifschneider, P. Tinsley, and J. Williams, "The Role of Expectations in the FRB/US Macroeconomic Model", Federal Reserve Bulletin, 1997.
7. Coletti, D., B. Hunt, D. Roseand, and R. Tetlow , "The Bank of Canada's New Quarterly Projection Model QPM. Part 3: The Dynamic Model", Technical Report No. 75, Ottawa: The Bank of Canada, 1996.
8. Creel, M., and W. L. Goffe, "Multi-core CPUs, Clusters, and Grid Computing: a Tutorial", Computational Economics, 32 (4), 353-382, 2008.
9. Dongarra, J., J. Du Croz, S. Hammarling, and I. Duff, "A set of level 3 basic linear Algebra subprograms", ACM Transactions on Mathematical Software, 16 (1), 1-17, 1990.
10. Dongarra, J., J. Du Croz, S. Hammarling, and R. Hanson, "An extended set of FORTRAN basic linear algebra subprograms", ACM Transactions on Mathematical Software, 14, (1), 1-17, 1988.
11. Doornik, J. A., D. F. Hendry, and N. Shephard, "Parallel Computation in Econometrics: A Simplified Approach" Chapter 15 in Handbook of Parallel Computing and Statistics, Chapman & Hall/CRC, 449-476, 2007.
12. Fair, R.C., and J. B.Taylor, "Solution and Maximum Likelihood Estimation of Dynamic Nonlinear Rational Expectations Models", Econometrica, 51(4), 1169-1185, 1983.
13. Fisher, P., Rational Expectations in Macroeconomic Models. Kluwer Academic Publishers, Dordrecht, 1992.

14. Golub, G. H., and C. F. Van Loan, Matrix Computations, Johns Hopkins Series in Mathematical Sciences, The Johns Hopkins University Press, 1996.
15. Harris, Mark J., William V. Baxter III, Thorsten Scheuermann, and Anselmo Lastra, "Simulation of Cloud Dynamics on Graphics Hardware." In Proceedings of the GGRAPH/Eurographics Workshop on Graphics Hardware 2003, pp. 92-101, 2003
16. Isard, P., "The Role of MULTIMOD in IMF's Policy Analysis", Technical Report IMF Policy Discussion Paper, International Monetary Fund, Washington DC, 2000.
17. Khronos OpenCL Working Group, The OpenCL Specification - Version 1.0. The Khronos Group, Tech. Rep. 2009.
18. Lawson, C. L., R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic linear algebra subprograms for Fortran usage", ACM Transactions on Mathematical Software, 5 (3), 308-323, 1979.
19. Levin, J., and R. Tryon, "Evaluating International Economic Policy with the Federal Reserves Global Model", Federal Reserve Bulletin, 1997.
20. Masson, P., S. Symanski, and G. Meredith, "MULTIMOD Mark II: A Revised and Extended Model", Technical Report, Occasional paper 71, International Monetary Fund, Washington DC, 1990.
21. NVIDIA, CUDA C Programming Guide, Version 4.0, 2011.
22. Saad, Y. Iterative Methods for Sparse Linear Systems, PWS Publishing Company, 1996.
23. Whaley, R. C., A. Petitet, and J. Dongarra, "Automated Empirical Optimization of Software and the ATLAS project", Parallel Computing, 27(1-2), 3-35, 2001.