4J-4

LU-factorization on Cypress GPU

Tomoya Sakai†, Kazuya Matsumoto†, Naohito Nakasato†, Stanislav G. Sedukhin†

†The University of Aizu

Abstract. We present performance result of LU - factorization on Cypress GPU architecture. Our current implementation of LU-factorization achieves 379 Gflop/s (67 % of the peak).

1 Introduction

LU-factorization is an important part in many practical problems, which are based on the solution of system of linear equations. It is well known that the most intensive part of LU-factorization is General Matrix Multiply (GEMM). To speed-up LU-factorization, it is important to accelerate GEMM.

Multicore architectures such as GPU specializes for computing. There are various works to speed-up GEMM on multicore architectures. Cypress GPU can compute 1600 fused multiply-add (FMA) operations per cycle in single precision and 320 FMA operations per cycle in double precision.

There are various works which speed-up LU-factorization by tuning GEMM on GPU. Nath et al [1] reported DGEMM and one-sided factorization on Fermi GPU. Their LU-factorization achieved 224 Gflop/s. Rohr et al [2] present their implementation of combined CPU/GPU DGEMM and LU-factorization on Cypress GPU. Their system used 24 cores CPU and AMD GPU achieved 563.2 Gflop/s for LU-factorization.

In this paper we presents LU-factorization on Cypress GPU. We implemented blocked right-looking algorithm. Our algorithm have four parts, panel factorization, solving triangular system of equations, swapping, and DGEMM. We ported solving triangular system of equations to GPU.

2 Implementation

In this section, we describe three different task distribution to port computation on GPU. We used blocked right-looking algorithm [3]. In our implementation the size of inital matrix is $n \times n$. Initial matrix is divided into $N \times N$ blocks where $N = n/b$ and b is the block size. It is stored as column-major order.

Pseudocode is described as Algorithm 1. This algorithm factorizes block $A_{K,K}$ at first on each iteration (line 2). This part includes pivoting. Trailing matrix of K-th column and K-th row is expressed as triangular system (lines 4 and 7). Each of them is computed by solving triangular system using $L_{K,K}$ and $U_{K,K}$. We call solving these triangular systems as L-block update and U-block update. Finally trailing matrix is updated (lines 9-13).

Algorithm 1 LU-factorization

```
1:  for K = 1 : N do
2:    panel factorization A_{K,K} = L_{K,K} × U_{K,K}
3:    for I = K + 1 : N do
4:      L_{I,K} = A_{I,K} × U^{-1}_{K,K}
5:    end for
6:    for j = k + 1 : N do
7:      U_{K,J} = L^{-1}_{K,K} × A_{K,J}
8:    end for
9:    for I = K + 1 : N do
10:     for J = K + 1 : N do
11:       A_{I,J} = A_{I,J} − L_{I,K} × U_{K,J}
12:     end for
13:   end for
14: end for
```

We made three different implementations depending on task distribution between CPU and GPU. For all implementations, panel factorization (line 2) is computed by DGETRF routine and updating trailing matrix (lines 9-12) is computed DGEMM routine. Differences in implementation are how we do L-block update, U-block update, and pivoting.

For the first implementation, L-block update is computed by DGETRF. It includes partial pivoting, which choose a pivot from $A_{k,k:N}$. U-block update is solved by DTRSM routine. For the second implementation, U-block update is computed on GPU. Before computation on GPU, inverse of $L_{K,K}$ is computed by TRTRI routine and then matrix multiplication is computed on GPU by DGEMM routine. For the third implementation, L- and U- block update is computed on GPU by using inverse of matrix. In this implementation, only one block matrix is factorized and pivot is chosen from $A_{K:K}$. This implementation is expected to be less stable than others. These differences are shown in Table 1.

	L-block update	U-block update	pivoting
(1)	DGETRF	DTRSM	$A_{K,K:N}$
(2)	DGETRF	DTRTRI+DGEMM	$A_{K,K:N}$
(3)	DTRTRI+DGEMM	DTRTRI+DGEMM	$A_{K,K}$

Table 1: Differences between three implementations

3 Performance Evaluation

In this section, we present performance evaluation of LU-factorization. We used Intel Core i7 920

(2.67GHz) as CPU. The peak performance for CPU is 42.72 Gflop/s. Radeon HD5870 is used as GPU. The peak performance for GPU is 544 Gflop/s.

BLAS routines except DGEMM are computed by GotoBlas2 version 1.13 [4]. We modified fast DGEMM kernel which computes $C \leftarrow C + A^T \times B$ on Cypress GPU implemented by Nakasato [5]. DGEMM kernel requires transposing matrix which is faster than using $C \leftarrow C + A \times B$ kernel. We chose $b = 1792$ as the block size in this research because it is optimal size for our current implementation of DGEMM [6]. The peak performance of DGEMM is 450 Gflop/s.

The performance, as the function of matrix size n, is shown in Fig 1. The maximum performance for three implementations is 262 Gflop/s (44% of the peak), 282 Gflop/s (48 % of the peak), and 379 Gflop/s (67% of the peak) respectively. The second and the third implementation require more operations than the first implementation but performance is better. A large speed-up was obtained by porting solving triangular systems on GPU.

The breakdown of run-time of four parts are shown in Fig 2. The effect of porting only U-matrix update is not so big. There is a big difference whether porting both L- and U- matrix update. In this research, the peak performance for CPU is quite lower than GPU. Therefore computation on CPU becomes bottleneck. By comparing our work with Rohr et al [2], we suppose the difference in performance comes from the peak performance of CPU.

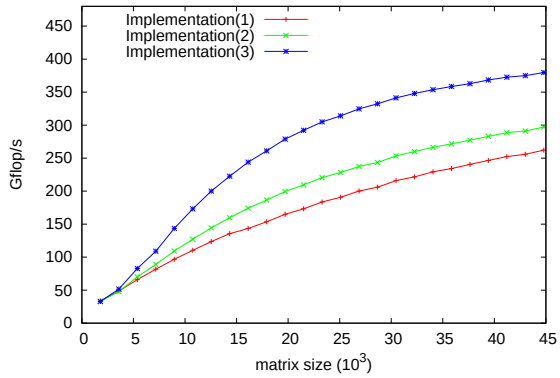

Figure 1: Performance of LU-factorization (see also Table 1)

4 Conclusion

We have presented three implementations of blocked right-looking algorithm. By porting both L- and U- matrix update to GPU we can speed-up LU-factorization because the peak performance of CPU is much lower than GPU in our system. However this implementation is expected to introduce problem in numerical stability.

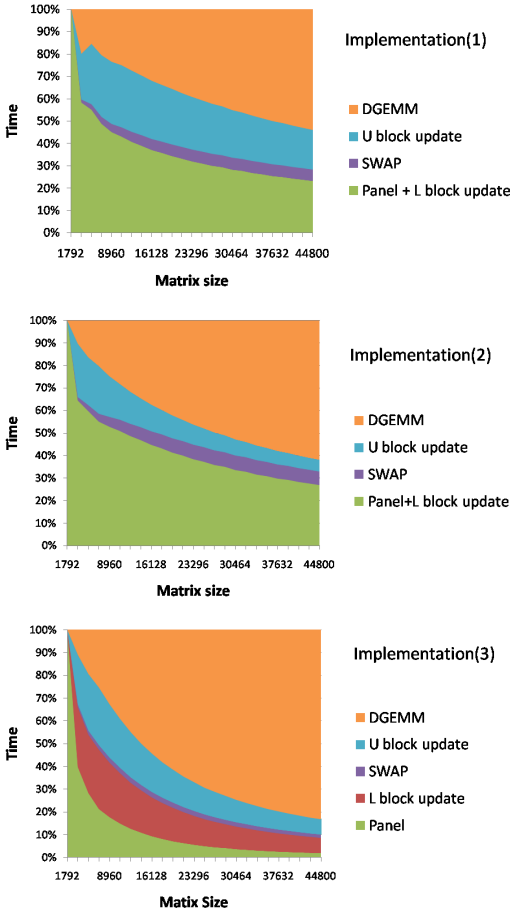

Figure 2: Breakdown of run-time (see also Table 1)

Future work includes futher optimization for LU-factorization and implementing other algorithms to solve system of linear equations such as QR factorization.

References

[1] Nath, R., Tomov, S. and Dongarra, J.: An Improved MAGMA GEMM For Fermi GPUs, *University of Tennessee, Tech. Rep UT-CS-10-655* (2010).

[2] Rohr, D., Kretz, M. and Bach, M.: Technical Report, CALDGEMM and HPL (2010).

[3] Dongarra, J.: *Numerical linear algebra for high-performance computers*, Society for Industrial Mathematics (1998).

[4] Goto, K.: . www.tacc.utexas.edu/resources/software/.

[5] Nakasato, N.: A Fast GEMM Implementation On a Cypress GPU, *PMBS 10* (2010).

[6] Matsumoto, K., Nakasato, N., Sakai, T., H, Y. and Sedukhin, S. G.: Multi-level Optimization in Matrix Multiplication for GPU-equipped Systems (submitted) (2011).