A Fine-Grained Pipelined Implementation of LU Decomposition on SIMD Processors

Kai Zhang, Shuming Chen, Wei Liu, Xi Ning

A Fine-Grained Pipelined Implementation of LU Decomposition on SIMD Processors

ZHANG Kai, CHEN ShuMing[a)], LIU Wei, NING Xi

School of Computer, National University of Defense Technology
#109, Deya Road, Changsha, 410073, China
a) smchen@nudt.edu.cn

Abstract. The LU decomposition is a widely used method to solve the dense linear algebra in many scientific computation applications. In recent years, the single instruction multiple data (SIMD) technology has been a popular method to accelerate the LU decomposition. However, the pipeline parallelism and memory bandwidth utilization are low when the LU decomposition mapped onto SIMD processors. This paper proposes a fine-grained pipelined implementation of LU decomposition on SIMD processors. The fine-grained algorithm well utilizes data dependences of the native algorithm to explore the fine-grained parallelism among all the computation resources. By transforming the non-coalesced memory access to coalesced version, the proposed algorithm can achieve the high pipeline parallelism and the high efficient memory access. Experimental results show that the proposed technology can achieve a speedup of 1.04x to 1.82x over the native algorithm and can achieve about 89% of the peak performance on the SIMD processor.

1 Introduction

The solving of dense linear algebra with large scale is the key problem of many scientific computation applications. The famous LINPACK benchmark [1] [2], which is used to test the performance of super computers, runs the computation of a set of linear equations. There are two methods for linear algebra systems, which are direct methods and iterative methods. Usually, the direct method is very suited to dense matrices [3]. The typical direct method is solving the linear algebra with the help of LU decomposition, which makes efficient implementations, either purely in software or with hardware assistance, is therefore desirable.

Libraries for linear algebra [4] [5] with optimized subroutines have been highly tuned to run on a variety of platforms. The GPUs based system is a popular method. On the CPU/GPU hybrid architecture, the performance can be high up to 1000 GFLOPs for single precision as well as 500 GFLOPs (1 GFLOPS = 10^9 Floating Point Operations/second) for double precision. However, the memory hierarchy of GPUs is complex. Michael [6] tailored their implementation to the GPUs inverted memory hierarchy and multi-level parallelism hierarchy. They proposed a model that can accurately predict the performance of LU decomposition. Leandro [7] evaluated different types of memory access and their impact on the execution time of the

algorithm. They showed that coalesced access to the memory leads to 15 times faster than the non-coalesced version. The task scheduling between CPUs and GPUs is also important. Optimized task scheduling can effectively reduce the computation [8-12], which is corresponding to the idea on general purpose processors [13]. Reconfigurable platforms always focus on designing high performance float-point computation units and the scalable architecture [14-16]. However, the high cost of reconfigurable hardware leads that they are not widely used.

The previous methods pay much attention on the optimization among system nodes. And the optimization on each unique node lacks of architecture transparence. The common idea of various platforms is that try to exploit sufficient parallel resources, which makes the single instruction multiple data (SIMD) technology been a popular method to accelerate the LU decomposition. This paper extends the work to optimize the LU decomposition on each node chip implemented with the modern SIMD architecture. When the LU decomposition is mapped onto SIMD processors, the native data dependences of the algorithm lead to the low pipeline parallelism and too much memory access. And the large amount of non-coalesced memory access existed in the algorithm results in much more memory access time.

To address these issues, a fine-grained pipelined implementation of LU decomposition is designed to increase the performance on SIMD processors. The proposed algorithm well utilizes data dependences to explore the fine-grained pipeline among all the computation resources. The pipeline parallelism is highly increased on the SIMD processor. The memory access operations are reduced by immediately reusing the temporary result. By using external loop unrolling method, the non-coalesced memory accesses are smartly transformed into coalesced version, which effectively reduces most of the memory access time.

The rest of the paper is organized as follows: in Section 2, we show a basic SIMD architecture which is taken as our optimization platform. Section 3 analyzes the basic LU decomposition on SIMD processors. In Section 4, we describe the proposed fine-grained pipelined algorithm. The experimental results are presented in Section 5. Finally, Section 6 concludes the paper.

2 SIMD architecture

The SIMD technology can operate on multiple data in parallel by the control of a single instruction. Thanks to the simple control logic and the low cost hardware, it can provide high performance at low power consumption and is well applied to the design of processors, such as GPUs, AnySP [17], Cell [18] and VT [19]. Based on the above modern state-of-the-art SIMD processors, we setup a basic framework for SIMD processors. The basic SIMD processor is shown in Fig. 1 and is chosen as our platform to optimize the LU decomposition.

As is depicted in Fig. 1, a SIMD processor generally consists of a Scalar Unit (SU) and multiple Vector Processing Elements (VPEs). The SU is used to execute the communication and scalar computation. Multiple VPEs are used to accelerate the parallel computation. The number of VPEs is referred as SIMD width. Each VPE usually contains functional units such as ALU, Multiply-Accumulate (MAC) and

Load/Store (LS). The adder tree performs the reduction add operation among VPEs. The shuffle unit can rearrange the vector data among all VPEs. The LS controls the access to the on-chip vector memory (VM) which consists of multiple parallel banks. A coalesced memory access can increase the bandwidth efficiency and therefore the performance of memory-bound kernels. The shared-registers (SR) can be accessed by the SU and all VPEs.

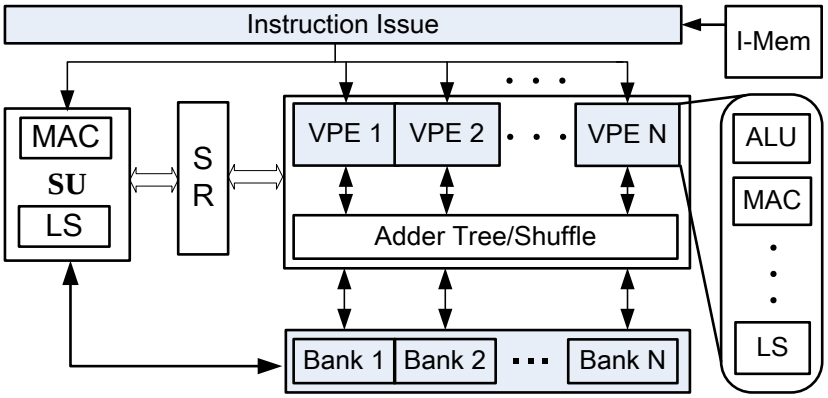

Fig. 1. The basic SIMD architecture.

3 LU Decomposition

For solving a set of linear equations of the form

$$Ax = b,$$

where x and b are $n\times1$ column vectors. The LU decomposition is used to factor a $n\times n$ matrix A into a $n\times n$ lower triangular matrix L (the diagonal entries are all 1) and a $n\times n$ triangular matrix U. Such that,

$$LUx = b.$$

Figure 2 shows the procedure of the basic LU decomposition of a matrix. The computation amount of the algorithm is $O(n^3)$. And the communication amount is $O(n^2)$. The matrix A is replaced by matrices L and U after the decomposition. The loop can be divided into two segments, which are separated by dotted lines in Fig. 2. The first segment is used to update the k^{th} column, where the computation amount is $O(n^2)$. The second segment is used to update the right lower sub-matrix, where the computation amount of the algorithm is $O(n^3)$. As shown in Fig. 2, this algorithm has high data dependences. The updated k^{th} column of A is used to update the right lower sub-matrix. The matrix updating segment can only be executed after all the iterations of k^{th} column updating computation. As the matrix for decomposition is always large, the updated k^{th} column cannot all reside in local registers and must be written back to memory. Although it is time cost, the updated k^{th} column must be read from memory when performing the matrix updating segment, which causes large amount of memory access operations.

```
for k = 1 to n-1
    * * * * * * * * segment 1 * * * * * * * *
   for i = k + 1 to n
      A[i,k] = A[i,k]/A[k,k];
   end for
    * * * * * * * *  segment 2 * * * * * * * *
   for j = k + 1 to n
      for i = k + 1 to n
         A[i,j] = A[i,j] − A[i,k] × A[k,j];
      end for
   end for
    * * * * * * * * * * * * * * * * * * * * *
end for
```

Fig. 2. The basic LU decomposition algorithm.

In the segment of updating the k^{th} column, $A[k,k]$ is shared among all VPEs to perform the computation in parallel. The updated results cannot totally reside in registers, thus they must be stored into memory. We assume that the SIMD width is N. Totally N iterations can be executed in parallel in the segment of updating the right lower matrix, where N elements of the k^{th} column must be loaded into registers from memory. As we can see, there are three apparent deficiencies in the above algorithm. Firstly, the high data dependences lead to the low pipeline parallelism. The two segments are totally executed in sequence. Secondly, all the temporary results, which are possible to reside in registers for reusing, are written back to the memory. Thirdly, the column data access, which causes non-coalesced access to parallel memory banks, is low efficient. To address this issue, we propose a fine-grained pipelined LU decomposition algorithm. The detail of this algorithm is explained in the next section.

4 Fine-grained Pipelined LU Decomposition

To achieve the high pipeline parallelism and the high efficient memory access, a direct way is reducing data dependences and transforming the non-coalesced access to coalesced version. To reduce the amount of memory access, we must try to keep the temporary results, which will be reused, in local registers. Thus, we must make a detailed analysis on the program. We analyze data dependences among the two task segments of the program code. Fig. 3 shows data dependences of the basic LU decomposition algorithm.

The loop unrolling technique is a traditional method for loop accelerating. For nested loop optimization, the universal method is internal loop unrolling with software pipelining. However, this method causes the non-coalesced memory access to k^{th} and j^{th} columns of matrix A. As shown in Fig. 3, the updated $A[i,k]$ is used to update $A[i,j]$ (j=k+1 to n) which is the i^{th} row of matrix A. As the updating of the i^{th} row is executed in the external loop of the second segment, we can unroll the external loop instead of the internal loop. The unrolled loop is dispatched to all VPEs. Each VPE execute an iteration of the external loop of segment 2. Then all VPEs access the

i^{th} row of matrix A in parallel. The non-coalesced memory access is smartly replaced by the coalesced memory access.

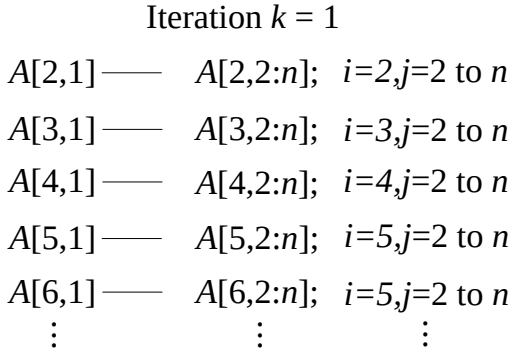

Fig. 3. Data dependences in the LU decomposition.

The SIMD execution of the i^{th} row updating only requires the updated $A[i,k]$ but not the whole updated k^{th} column. We can start the updating of i^{th} row as soon as the $A[i,k]$ is updated. As the $A[i,k]$ is a scalar element, we can employ the SU to compute the $A[i,k]$. The updated $A[i,k]$ is fast shared with all VPEs by using the SR without writing back to memory. The shuffle unit can broadcast $A[i,k]$ to all VPEs. Then all VPEs immediately execute the updating of i^{th} row in SIMD way after receiving $A[i,k]$.

We can see that, the above method introduces a fine-grained pipelined algorithm for LU decomposition. The updating of sub-matrix is divided into a series updating processes for each matrix row. The updating of k^{th} column is divided into a series updating processes for each column element. As soon as each column element is updated, it is immediately used to update the corresponding matrix row. Compared with the basic algorithm, the proposed method provides more fine-grained parallelism for the software pipeline.

The proposed software pipeline greatly corresponds to the data dependences. By software method assistance, the fine-grained pipelined LU decomposition can be easily implemented. Thus a high pipeline parallelism can be achieved. The proposed fine-grained pipelined algorithm has the following advantages: efficiently utilization of all the computation resource on the modern SIMD processor, reducing the memory access amount and transforming the non-coalesced memory access to coalesced version. The problems of low pipeline parallelism, too much memory access and low efficient memory bandwidth utilization are well solved.

Fig. 4 shows the data flow of the proposed algorithm on the SIMD architecture. As shown in Fig. 4, the k^{th} column updating task is executed on SU. The right lower sub-matrix updating task is executed on all VPEs in SIMD way. To transform the non-coalesced memory access to coalesced version, the external loop of the sub-matrix updating segment is unrolled to be executed on all VPEs in parallel. And each VPE performs the internal loop. The SU and SIMD tasks are executed in parallel to achieve the high pipeline parallelism. The software pipelining technology is used to keep the native data dependence of the algorithm.

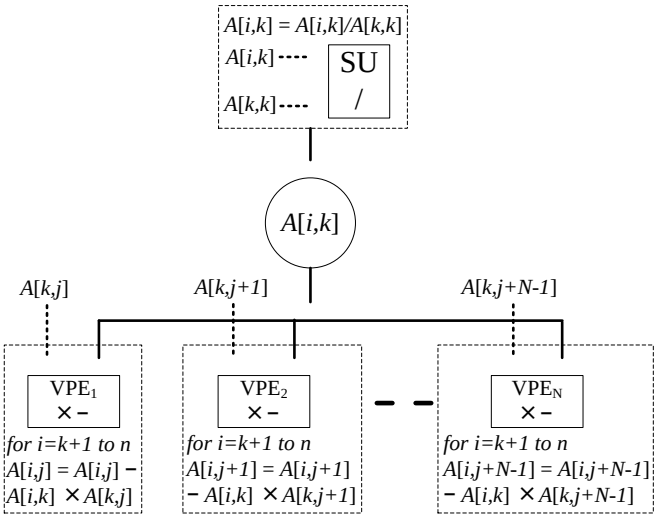

Fig. 4. The data flow of the proposed algorithm.

Fig. 5 shows the code example of the proposed fine-grained pipeline algorithm. The software pipelining is not shown to simplify the description. As shown in Fig.5, the SU task and SIMD task can be executed in parallel. The SU task is executed in the scalar unit to update the k^{th} column. As soon as each element of the k^{th} column is updated, it is written to the SR. Then the shuffle unit broadcasts the updated element to all VPEs. Each VPE employs this element to update a matrix row. All VPEs updates the sub-matrix in parallel.

```
for k = 1 to n-1
   Initial SU task
   for i = k + 1 to n
      A[i,k] = A[i,k]/A[k,k];
   end for
   Initial SIMD task
   for j = k +1 to n
      broadcast A[i,k] to all VPEs;
      for i = k + 1 to n
         A[i,j] = A[i,j] – A[i,k] × A[k,j];
      end for
      for i = k + 1 to n
         A[i,j+1] = A[i,j+1] – A[i,k] × A[k,j+1];
      end for
            ⋮
      for i = k + 1 to n
         A[i,j+N-1] = A[i,j+N-1] – A[i,k] × A[k,j+N-1];
      end for
      j = j +N;
   end for
end for
```

Fig. 5. The proposed fine-grained pipelined LU decomposition algorithm.

5 Experiment Results

We have implemented a cycle-accurate simulator FT-Matrix-Sim based on our previous single-core simulator [20] for the SIMD architecture as shown in Fig. 1. In FT-Matrix-Sim, the VM is organized as scratchpad memory and simulated with a cycle-driven system. There are 16 VPEs, each having a 32-entry register file. Each VPE contains four function units: ALU, two L/S and MAC. The MAC unit supports one float-point operation each cycle. The SU has the same units with each VPE. The VM has 16 banks. Manually optimized assemble code is used as the input of the simulator.

The architecture established by the simulator has also been implemented in Verilog hardware design language. The RTL implementation has been synthesized with Synopsys Design Complier under TSMC 65 nm technology in order to obtain the operating speeds and the cycle accurate performance simulation. The clock frequency of the synthesized circuit can be up to 500 MHz.

Fig.6 shows the performance of the proposed fine-grained pipelined LU decomposition algorithm. As can be seen, the speedup of the fine-grained algorithm over the basic algorithm can be up to 1.04x to 1.82x. The average speedup is 1.15x. The performance of the fine-grained algorithm can be up to 15.98 GFLOPS, where GFLOPS is a popular metrix for numerical computations. The peak performance of the synthesized circuit is 18 GFLOPS. Thus, we achieve 88.78% of peak performance on the SIMD processor by using the proposed fine-grained algorithm. The delectable result is that we can obtain better performance with the increasing of matrix size.

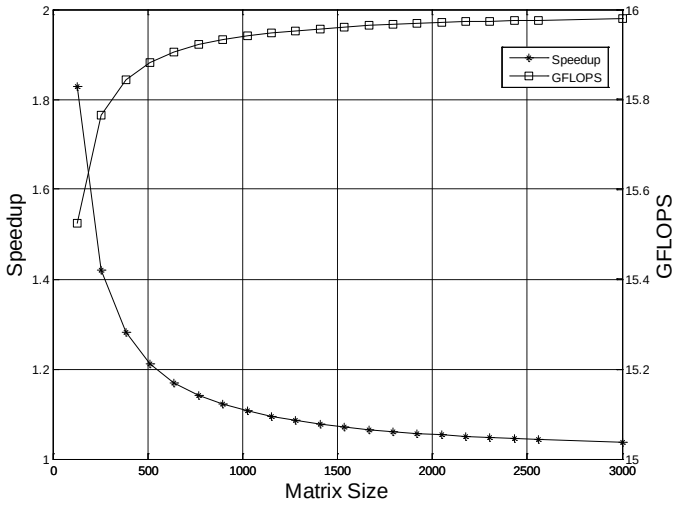

Fig. 6. The performance of proposed algorithm.

The performance improvement resulting from the fine-grained algorithm is due to the following two factors: the high pipeline parallelism and the high efficient memory

access. The fine-grained method greatly corresponds to the native data dependences of the algorithm. It well explores the fine-grained parallelism between the SU and multiple VPEs, which can effectively increase the pipeline parallelism and reduce the memory access amount. The external loop unrolling method of the fine-grained pipelined algorithm transforms all non-coalesced memory access to coalesced version, which is the high efficient memory access way. The coalesced method can reduce most of the memory access time. Table I shows the reduced memory access time of the fine-grained algorithm compared with the basic version. As can be seen, the proposed algorithm can reduce the memory access time by 90% to 96%.

The main contribution of the proposed algorithm is reducing the communication amount and hiding the computation time of updating the k^{th} column. Both complexities of the above two tasks are $O(n^2)$. However, the total computation amount of the algorithm is $O(n^3)$. So the proportion of the performance increments brought by the proposed method goes smaller when the matrix size goes larger. But the absolute performance of the proposed algorithm is always high.

Table 1. Reduced memory access time.

Size	Reduced (%)	Size	Reduced (%)	Size	Reduced (%)
128	90.15%	1024	94.85%	1920	95.85%
256	91.62%	1152	95.01%	2048	95.92%
384	92.65%	1280	95.23%	2176	95.95%
512	93.41%	1408	95.39%	2304	95.96%
640	93.81%	1536	95.53%	2432	95.97%
768	94.19%	1664	95.64%	2560	95.99%
896	94.54%	1792	95.74%	3000	95.99%

6 Conclusions

This paper proposed a fine-grained pipelined LU decomposition implementation on SIMD processors. The main performance improvement is achieved by the high pipeline parallelism and the high efficient memory access. The new algorithm well corresponds to data dependences of the native algorithm and effectively transforms the non-coalesced memory access to coalesced version, which increases the pipeline

parallelism, reduces the memory access amount and increases the efficient of memory access. To illustrate our ideas, we implemented a cycle-accurate simulator and a RTL design. The RTL design was synthesized under TSMC 65 nm technology and achieved a frequency of 500 MHz. With the proposed algorithm, we can obtain a performance speedup about 1.04x to 1.82x over the basic LU decomposition algorithm. Most of the peak performance on the SIMD processor is well utilized by the proposed technology.

Acknowledgement

This work is supported by the National Natural Science Foundation of China (No.61070036) and HPC Foundation of NUDT.

References

1. Dongarra, Jack J., Piotr Luszczek, and Antoine Petitet: The LINPACK benchmark: past, present and future. Concurrency and Computation: practice and experience 15.9 (2003): 803-820.
2. LINPACK. http://www.netlib.org/linpack.
3. Michailidis, Panagiotis D., and Konstantinos G. Margaritis: Implementing parallel LU factorization with pipelining on a multicore using OpenMP. Computational Science and Engineering (CSE), 2010 IEEE 13th International Conference on. IEEE, 2010.
4. Dongarra, Jack J., and David W. Walker: Software libraries for linear algebra computations on high performance computers. SIAM review 37.2 (1995): 151-180.
5. Whaley, R. Clint, and Jack J. Dongarra: Automatically tuned linear algebra software. Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM). IEEE Computer Society, 1998.
6. Anderson, Michael J., David Sheffield, and Kurt Keutzer: A predictive model for solving small linear algebra problems in gpu registers. Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International. IEEE, 2012.
7. Cupertino, Leandro F., et al: LU Decomposition on GPUs: The Impact of Memory Access. Computer Architecture and High Performance Computing Workshops (SBAC-PADW), 2010 22nd International Symposium on. IEEE, 2010.
8. Galoppo, Nico, et al: LU-GPU: Efficient algorithms for solving dense linear systems on graphics hardware. Proceedings of the 2005 ACM/IEEE conference on Supercomputing. IEEE Computer Society, 2005.
9. Lifflander, Jonathan, et al: Dynamic Scheduling for Work Agglomeration on Heterogeneous Clusters. Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International. IEEE, 2012.
10. Donfack, Simplice, et al: Hybrid static/dynamic scheduling for already optimized dense matrix factorization. Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International. IEEE, 2012.
11. Lifflander, Jonathan, et al: Mapping dense lu factorization on multicore supercomputer nodes. Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International. IEEE, 2012.
12. Venetis, Ioannis E., and Guang R. Gao: Mapping the LU decomposition on a many-core architecture: challenges and solutions. Proceedings of the 6th ACM conference on Computing frontiers. ACM, 2009.

13. Grigori, Laura, James W. Demmel, and Hua Xiang: Communication avoiding Gaussian elimination. Proceedings of the 2008 ACM/IEEE conference on Supercomputing. IEEE Press, 2008.
14. Jaiswal, Manish Kumar, and Nitin Chandrachoodan: Fpga-based high-performance and scalable block lu decomposition architecture. Computers, IEEE Transactions on 61.1 (2012): 60-72.
15. Zhuo, Ling, and Viktor K. Prasanna: High-performance and parameterized matrix factorization on FPGAs. Field Programmable Logic and Applications, 2006. FPL'06. International Conference on. IEEE, 2006.
16. Zhuo, Ling, and Viktor K. Prasanna: High-performance designs for linear algebra operations on reconfigurable hardware. Computers, IEEE Transactions on 57.8 (2008): 1057-1071.
17. Mark Woh, Sangwon Seo, Scott Mahlke, Trevor Mudge, Chaitali Chakrabarti and Krisztian Flautner: AnySP: Anytime Anywhere Anyway Signal Processing. In Proceedings of the 31st Annual International Symposium on Computer Architecture(ISCA'09), Austin, Texas, June 20–24, 2009.
18. Brian Flachs, Shigehiro Asano, Sang H.Dhong, et al: The Microarchitecture of the Synergistic Processor for a Cell Processor. IEEE Journal of Solid-State Circuits, Vol. 41, NO. 1, Jan.2006.
19. Krashinsky, Ronny, et al: The vector-thread architecture. Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on. IEEE, 2004.
20. Chen, Shu-Ming, et al: YHFT-QDSP: High-performance heterogeneous multi-core DSP. Journal of Computer Science and Technology 25.2 (2010): 214-224.