Performance Comparison of Cholesky Decomposition on GPUs and FPGAs

Depeng Yang, Junqing Sun, JunKu Lee, Getao Liang, David D. Jenkins, Gregory D. Peterson, and Husheng Li
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville, TN, 37996

***Abstract*—Cholesky decomposition has been widely utilized for positive symmetric matrix factorization in solving least square problems. Various parallel accelerators including GPUs and FPGAs have been explored to improve performance. In this paper, Cholesky decomposition is implemented on both FPGAs and GPUs by designing a dedicated architecture for FPGAs and exploiting massively parallel computation for GPUs. Performance of the Cholesky decomposition on GPUs, CPUs, FPGAs, and hybrid systems are compared in both single and double precision. Results show that the FPGA implementation has the highest efficiency with respect to clock cycles compared with our pure GPU implementation, a hybrid system with MAGMA, and a CPU with LAPACK. The GPU implementation is better than other implementations using MAGMA and LAPACK library for small matrices, and the hybrid system with MAGMA is the best for larger matrices.**

I. Introduction

Cholesky decomposition is often adopted for positive symmetrical matrix decomposition in solving least square problems, such as signal processing [1], machine learning and data regression [2]. However, Cholesky decomposition is computational expensive with $O(N^3)$ complexity. Modern parallel accelerators such as FPGAs and GPUs can be utilized for accelerating Cholesky decomposition.

In this paper, Cholesky decomposition is implemented on both FPGAs and GPUs for computation acceleration. On FPGAs, a dedicated accelerator is designed with only one scalable triangular linear equation solver with 64 Processing Elements (PEs). On GPUs, the massively parallel computation is well organized and optimized for best performance by rearranging the computation procedure and managing memories. Moreover, computation accelerations on the GPU, CPU, FPGA, and hybrid system are compared from many perspectives, such as the programming language, flexibility, memory volume, memory bandwidth, and computational precision. Performance of implementations on the GPU, FPGA, CPU, and hybrid system for Cholesky decomposition are also compared.

II. Cholesky Decomposition

The Cholesky decomposition factors a positive symmetric matrix to the product of a lower triangular matrix and its transpose, of which two general ways are given by:

$$A = U^T U \quad (1)$$

$$A = LDL^T \quad (2)$$

where A is a positive symmetric matrix. U is the upper triangular matrix. L is the unit lower triangular matrix with units along the diagonal line. D is the diagonal matrix in which all elements are zeros except the diagonal elements.

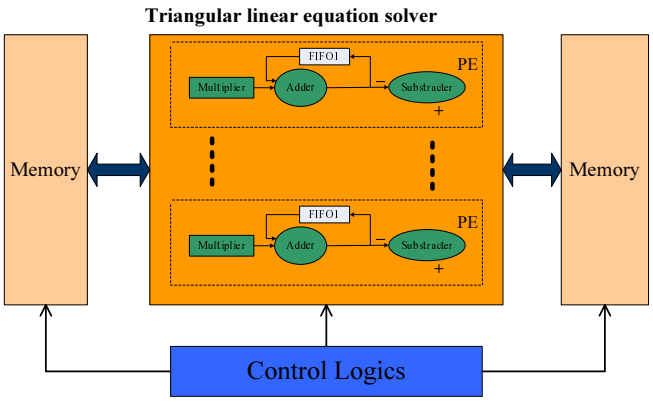

Fig. 1: Triangular linear equation solver for FPGA Cholesky decomposition

Cholesky decomposition has $O(\frac{1}{3}N^3)$ complexity with heavy inner data dependency. Introducing a diagonal matrix as shown in Eq. (2) in Cholesky decomposition has many advantages, such as avoiding square roots and alleviating data dependency [4]. In this paper, we design and implement the LDL^T Cholesky decomposition in Eq. (2) on both FPGAs and GPUs for computation acceleration.

III. Implementation

A. FPGA implementation

We design a dedicated hardware architecture for Cholesky decomposition on FPGAs. For FPGA implementation, the traditional Cholesky decomposition in Eq. (1) results in a long latency square root operation, which makes it hard to design an efficient pipelined PE. Therefore we realize an LDL^T Cholesky decomposition on FPGAs. To achieve good performance, we only implement one triangular solver for the FPGA. Fig. 1 shows the implemented Cholesky decomposition on FPGAs utilizing the triangular linear equation solver [4]. Control logic, memory and the communication bus can be optimized for the best performance. For small matrices, the memory bandwidth is not an issue. The number of the pipelined PEs determines the performance. We implement 64 PEs for the LDL^T Cholesky decomposition. More details can be found in [4].

B. GPU implementation

We also implement an LDL^T Cholesky decomposition on GPUs. In contrast to FPGAs, a GPU has a fixed SIMD-like hardware architecture for general computation. To achieve good performance, massively parallel computation should be primarily used and carefully orchestrated to provide enough threads to keep the functional units busy while managing memory to hide latencies. Instead of the iterative Cholesky decomposition procedure which is utilized for FPGA implementation, a "right-looking" algorithm is adopted [6] for LDL^T Cholesky decomposition on GPUs. Our LDL^T "right-looking" blocked algorithm is similar to LL^T "right-looking" algorithm [6], but it is modified by removing square root operations and re-organizing the computation procedure to improve performance. The computation starts with the top-left sub-matrix. Following that, the strip of bottom-left sub-matrices is calculated. Next, the rest of the elements are updated by taking advantage of matrix-matrix multiplication routines on GPUs. More details are in [6] [3]. We optimize the GPU implementation for good performance. Shared memory is fully utilized. Memory access patterns are optimized for coalesced memory reads and writes. Banking conflicts in shared memory are also avoided. All these efforts benefit the final performance.

C. Performance comparison of FPGA and GPU implementations

Both FPGAs and GPUs can be utilized as accelerators to speedup Cholesky decomposition. Generally, GPU has a fixed SIMD hardware architecture, which can provide massively parallel execution resources and high memory bandwidth. FPGAs provide basic logic units, function blocks, lookup tables, and routing resources which are highly customizable for fine grained parallelism. One expects that FPGAs can provide the best performance, flexibility, and low overhead because the hardware architecture can be fully customized for the specific application. Enabling configurable hardware has costs in both size and performance. Moreover, FPGA programmers must design the dedicated architecture and consider all hardware details, while GPUs tend to be easier to programs and have less hardware controllability.

The programming language for GPUs is CUDA [5] or OpenCL [10], an extension of C and an associated API for general purpose applications. By using CUDA, a GPU programmer does not need to know many hardware details. In CUDA, computation tasks are performed by thousands of threads in 3-dimensions, which are further organized as thread blocks and grids. CUDA provides a friendly interface for programmers by hiding hardware architecture details. However, for the best performance, the CUDA program for Cholesky decomposition acceleration still needs to be tuned and optimized according to the characteristics of the application and GPU hardware limits, such as arithmetic operation order, memory access pattern and communication.

For FPGAs, we utilize VHDL as the hardware description language. FPGA programmers have full control on the low level logic circuits, programmable resources and hardware architecture. IP CoreGen [] provides plenty of reusable and optimized IP cores, such as arithmetic units for scientific computations, which can save lots of designers' time. However, the dedicated hardware architecture for a particular application is still needed to be specially designed for the best performance. All hardware details, such as pipeline depth, latency, throughput, and memory bandwidth should be fully investigated and considered for FPGA designers.

Another way for accelerating Cholesky decomposition is to utilize heterogeneous GPUs and CPUs by using the Magma library [7]. The symmetric matrix is first partitioned into blocks. Massively parallel tasks like matrix multiplication are performed on GPUs and other serial tasks with heavy inner data dependence such as triangular solvers are executed on CPUs. The data communication cost between the host CPU and GPU is covered using data streaming functions.

Both GPUs and FPGAs have good potential to accelerate computations by exploiting the parallelism. GPUs generally have more memory resources than FPGAs. The fixed memory architecture, such as cache, global memory, and shared memory, along with their associated bandwidths, may slow down the performance for the Cholesky decomposition. For example, for the NVIDIA GeForce GTX480 GPU [9], the peak performance is 1.35Tflops/s in single precision with a 1400MHz frequency. However, due to memory bandwidth this peak performance is very difficult to achieve for Cholesky decomposition. For FPGAs, a dedicated triangular linear equation solver is designed by using several pipelined PEs. The dedicated scalable architecture can be fully customizable and optimized, but on-chip memory is very limited. For example, the newer Xilinx XC6VSX475T in Virtex 6 family FPGA has only 38,304 KB total memory which can be fully customized, so the limited memory resource is not suitable for the large matrix operations. The memory communication bus can be fully customized for Cholesky decomposition in order to achieve the best performance. GPUs are more suitable for large size matrix operation while FPGAs are the best for small size matrix due to limited but customizable on chip hardware resources.

GPUs can support single and double floating precision. When computation is associated with some operations, such as division, inversion, and square root, results from GPUs lose some precision. Due to limited hardware resources, the peak performance for double precision is worse than that for single precision. For instance, the GTX480 GPU has double precision performance only one half of the single precision performance [9], and older GPUs have one eighth the performance of single precision (or no support for double precision at all). On FPGAs, IP CoreGen provides IEEE 754 arithmetic units for both signal and double precision. Moreover, the precision of those arithmetic units in the PE can be fully customized by adjusting mantissa and exponential bits. Lower precision arithmetic units on FPGAs require significantly less hardware resources than the high precision units, leading to higher frequency and performance for FPGAs.

IV. Results

The Cholesky decomposition is implemented on the latest representative commercial products of GPUs and FPGAs. The GPU is an NVIDIA GTX480 card running at 1400MHz. The CPU is a 2.67GHz Intel Quad Core i7 with 12 GB RAM. The FPGA is XC6VSX475T in the Virtex 6 family. We utilize VHDL and IP CoreGen from Xilinx ISE 11.4 for providing IEEE 754 standard floating point arithmetic units for the PEs. We also use software compilers OpenCL version 3.0 and gcc version 4.3.3 for GPU and CPU.

Fig.2 and Fig.3 compare the computation cycles of Cholesky decomposition implemented on GPUs, FPGAs, CPUs using LAPACK [8], and a hybrid system using MAGMA [7] in both signal and double precisions. For the FPGA implementation, we implement 64 pipelined PEs for the Cholesky decomposition. The FPGA clock cycles are calculated based on the operations needed for a certain size matrix decomposition. The maximum achievable frequency on FPGAs is estimated as 180MHz based on the ISE tool. For the GPU and CPU implementation, we obtain the computation cycles through the division of the measured execution time and frequency clocks (1400MHz). Obviously, the number of cycles using FPGA, GPU, and CPU all increased with the matrix size. The FPGA implementation needs the minimum number of cycles because the hardware architecture is fully customized and optimized for the Cholesky decomposition. For single and double precision, our GPU implementation needs much fewer cycles than LAPACK and MAGMA, indicating more efficiency and less execution time. When the matrix size is larger than 512×512, that performance gap decreases.

Table 1 shows the performance comparison for Cholesky decomposition. The FPGA implementation is slow due to its low frequency (180MHz). Our GPU implementation is much better than LAPACK, and it is still better than MAGMA for small matrices.

TABLE I: Performance Comparison

Matrix	LAPACK GFLOP/s	MAGMA GFlop/s	GPU GFlop/s	FPGA GFlop/s
512 (SP)	19.49	22.21	58.40	19.23
512 (DP)	11.99	20.52	57.49	19.23
768 (SP)	29.53	38.53	81.87	20.38
768 (DP)	18.12	36.97	54.02	20.38
1024 (SP)	36.07	57.01	67.96	21.0
1024 (DP)	22.06	49.60	42.42	21.0
2048 (SP)	65.66	117.49	96.15	–
2048 (DP)	32.21	87.78	52.74	–

V. Conclusion

Cholesky decomposition has been implemented on both FPGAs and GPUs. For the best performance, a dedicated architecture is implemented on FPGAs and parallelism is rearranged for GPUs acceleration. Results show that FPGA needs the minimum number of cycles,whereas the GPU provides better runtime performance, particularly for larger matrices.

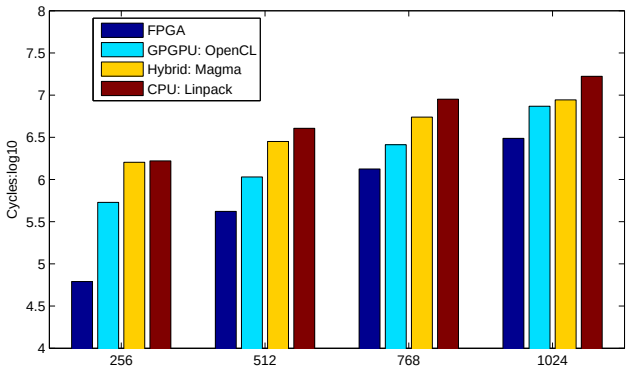

Fig. 2: Performance comparison for single precision

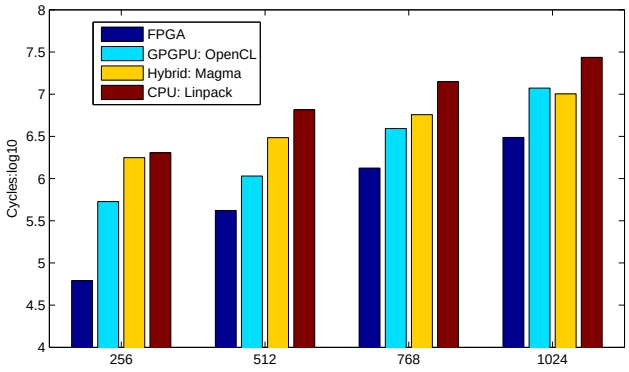

Fig. 3: Performance comparison for double precision

References

[1] D. Yang, H. Li, and G.D. Peterson, "Space-time Turbo Bayesian Compressed Sensing for UWB system," *IEEE Information Conference on Communications (ICC) 2010*, Kapton, South Africa, 2010.

[2] D. Yang, H. Li, G. D. Peterson and A. E. Fathy, "UWB Signal Acquisition in Locationing Systems: Compressed Sensing and Turbo Signal Reconstruction," *Conference on Information Sciences and Systems (CISS)*, Baltimore, MD, Mar., 2009.

[3] D. Yang, G. Tao, D. Jenkins, G. D. Peterson and A. E. Fathy, "High performance relevance vector machine on GPGPUs," *Symposium on Application Accelerators in High Performance Computing (SAAHPC)*, Knoxville, TN, July, 2010.

[4] D. Yang, G. D. Peterson, H. Li and J. Sun, "An FPGA Implementation for Solving Least Square Problem," *The 17th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, California, April., 2009.

[5] NVIDIA CUDA programming guide. Website. 2009. http://www.nvidia.com/object/cuda_home.html#

[6] Cholesky factorization algorithm. [Online] http://userweb.cs.utexas.edu/~plapack/icpp98/node2.html

[7] Matrix Algebra on GPU and Multicore Architectures (MAGMA) user guide. [online] http://icl.cs.utk.edu/magma/

[8] Linear Algebra PACKage (LAPACK), Version 3.2.1 [online] http://www.netlib.org/lapack/

[9] NVIDIA Tesla GTX480 computing processor. [Online] http://www.nvidia.com/object/product_geforce_GTX480_us.html

[10] OpenCL programming guide. [Online] http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_OpenCL_ProgrammingGuide.pdf