Matrix Computations on Heterogeneous Reconfigurable Systems [*]

Ling Zhuo, Qingbo Wang, Viktor K. Prasanna
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089-2562
{lzhuo, qingbow, prasanna}@usc.edu

Abstract

Reconfigurable computing systems have been built that combine FPGAs and general-purpose processors to achieve high performance. The nodes in these systems can have different compute capacities based on the processors and FPGAs within them. In this paper, we study the algorithm design on heterogeneous reconfigurable systems for two key linear algebra kernels: matrix multiplication and LU decomposition. Our designs exploit the heterogeneous nodes and utilize up to 80% of the compute capacity of the system.

1. Introduction

With the ever-increasing compute power of FPGAs, high-end reconfigurable computing systems that employ FPGAs as application-specific accelerators have been built, such as SRC 6 and 7 [3], Cray XD1 and XT5h [2], among others. These systems contain multiple nodes that are connected through an interconnect network. Each node is based on either FPGAs, general-purpose processors, or both. It has been shown that such systems can achieve higher performance than systems with processors only [4].

However, the heterogeneity of reconfigurable computing systems has been rarely investigated. These systems are heterogeneous in that the nodes have different computing capacities by containing different numbers of processors and FPGAs. Secondly, the memory accessible to different nodes in the system have different sizes and bandwidths. Finally, when an application is partitioned into tasks, not all the tasks can be executed on all the nodes.

In this paper, we propose high-performance designs for floating-point matrix multiplication and LU decomposition. Both the designs exploit the heterogeneous nodes, overlap the communication overheads with the computations, and achieve load balancing. We have implemented our designs on a Cray XD1 [2]. Experimental results show that for both kernels, our designs achieve up to 80% of the total compute capacity of the system in various heterogeneous settings.

2 Designs for Matrix Computations

In algorithm design on heterogeneous reconfigurable systems, we address various issues. First, the tasks in a given application are identified based on the data flow; the compute load and I/O requirements of each task are analyzed. Secondly, the tasks are assigned to the nodes. For load balancing, the workload assigned to a node is proportional to its compute capacity. Thirdly, workload assignment is further adjusted so that the computations on the FPGAs can be overlapped with network communications and memory accesses.

2.1 Matrix Multiplication

Consider computing $E = A \times B$, where A, B and E are $n \times n$ matrices partitioned into blocks of size $b \times b$. In our design, the columns of matrix A are partitioned into multiple column stripes. Each stripe consists of $\frac{n}{b}$ blocks. Similarly, the rows of B are partitioned into multiple row stripes. The tasks are then identified as computing one column stripe of E.

We denote the p nodes in the system as N_0, N_1, ..., N_{p-1}. Suppose x_i tasks are assigned to N_i. To maintain load balance in the system, the number of tasks assigned to N_i should be proportional to its compute capacity. If N_i contains an FPGA, x_i is adjusted so that network communications are overlapped with the computations on the FPGA.

In our design, matrices A and B are initially stored in the DRAM memory of N_0. During the computations, one column stripe of matrix A and one row stripe of matrix B are sent by N_0 to all the other nodes in one network transaction. For each row stripe of B, node N_i stores x_i blocks into its

[*]Supported by the United States National Science Foundation under grant No. CCR-0702784. Equipment grant from Xilinx Inc. is gratefully acknowledged.

DRAM memory (SRAM memory if N_i is an FPGA). When a block in A is transferred to a node, it is multiplied with all the stored blocks of B whose row indices are the same as its column index. The final results of E are transferred back to the DRAM memory of N_0.

2.2 LU Decomposition

We follow the block algorithm in [1] for LU decomposition of large matrices. The input matrix A^0 is partitioned into $b \times b$ blocks, which are denoted as A^0_{gh} $(0 \leq g, h \leq \frac{n}{b} - 1)$. We identify five types of tasks in block LU decomposition: *opLU*, *opL*, *opU* are basically LU decomposition for $b \times b$ blocks; *opMM* and *opMS* are block multiplication and block subtraction for $b \times b$ blocks. *opMM* operations need the outputs of *opL* and *opU* operations, and *opMS* operations need the outputs of *opMM* operations.

In our design, N_i initially stores x_i columns of the $b \times b$ blocks in A^0 $(0 \leq i \leq p-1)$. The tasks are assigned to the nodes, but only *opMM* operations are executed on the FPGAs. For load balance, x_i is proportional to N_i's compute capacity for matrix multiplication. In addition, the columns are assigned to the nodes interleavingly because the total number of tasks in one iteration keeps decreasing. The algorithm is illustrated in Figure 1.

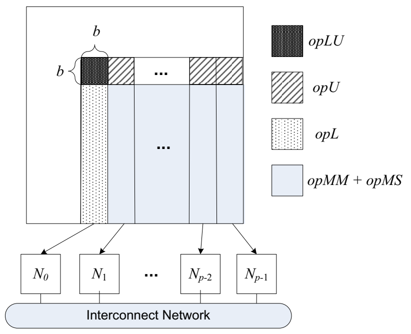

Figure 1. Algorithm Illustration for Block LU Decomposition

In iteration k $(0 \leq k \leq \frac{n}{b} - 1)$, there are $(\frac{n}{b} - k)$ phases. In phase 0, *opLU* operation is performed on block A^k_{kk} by $N_{t'}$, where the block is stored. The resulting U^{k+1}_{kk} is then transferred to all the other nodes. When N_i $(0 \leq i \leq p-1, i \neq t')$ performs *opU* operations on its stored blocks, $N_{t'}$ performs $(x_{t'} - 1)$ *opU* operations and one *opL* operation. In the following phase, the result of the *opL* operation is transferred to the other nodes for *opMM* operations. When N_i $(0 \leq i \leq p-1,\ i \neq t')$ performs *opMM* and *opMS* operations, $N_{t'}$ performs $(x_{t'} - 1)$ *opMM*, $(x_{t'} - 1)$ *opMS* operations, and one *opL*. Again, the result of the *opL* operation is transferred to the other nodes. This continues until all the phases are completed.

3 Experimental Results

To illustrate our ideas, we implemented our designs on one chassis of XD1 [2]. The chassis contains 6 compute blades. Each blade contains two AMD 2.2 GHz processors and one Xilinx Virtex-II Pro XC2VP50.

In our experiments, we chose to use only the FPGAs or the processors in some nodes of XD1 to simulate heterogeneous nodes. In Setting 1, all nodes are H(Hybrid)-nodes and each node contains one processor and one FPGA; in Setting 2, there are one H-node, one P(Processor)-node and for F(FPGA)-nodes; in Setting 3, there are two P-nodes, two H-nodes, and two F-nodes.

For matrix multiplication, our design achieves up to 85% of the total compute capacity of the system in all the three settings. At least 70% of the network communication time is overlapped with the computations. For LU decomposition, our design achieves lower percentage (up to 80%) of the total compute capacity because operations *opLU*, *opL* and *opU* are executed on the processors only. In the worst case (Setting 2), 60% of the network communications are overlapped with the computations. For both applications, when the total compute capacity of the system increases, the sustained performance of our design increases accordingly.

4 Conclusion

Reconfigurable computing systems contain various degrees of heterogeneity. We have proposed designs on such systems for two linear algebra kernels: matrix multiplication and block LU decomposition. Our designs were implemented on one of the representative systems, Cray XD1, and are shown to efficiently exploit the architectural potential of the system. In the future, we will study a generic architecture model for heterogeneous reconfigurable systems, which can provide more guidance to algorithm design and optimizations.

References

[1] J. Choi, J. Dongarra, L. Ostrouchov, A. Petitet, D. W. Walker, and R. C. Whaley. Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines. *Scientific Programming*, 5(3):173–184, Fall 1996.
[2] Cray Inc. `http://www.cray.com/`.
[3] SRC Computers, Inc. `http://www.srccomp.com/`.
[4] K. Underwood, K. Hemmert, and C. Ulmer. Architectures and APIs: Assessing Requirements for Delivering FPGA Performance to Applications. In *Proc. of the ACM/IEEE SuperComputing 2006 Conference (SC—06)*, Florida, USA, November 2006.