Optimizing Matrix Multiplication on Heterogeneous Reconfigurable Systems

Ling Zhuo and Viktor K. Prasanna

Ming Hsieh Department of Electrical Engineering,
University of Southern California,
Los Angeles, USA
E-mail: {*lzhuo, prasanna*}*@usc.edu*

With the rapid advances in technology, FPGAs have become an attractive option for acceleration of scientific applications. In particular, reconfigurable computing systems have been built which combine FPGAs and general-purpose processors to achieve high performance. Previous work assumes the nodes in such systems are homogeneous, containing both processors and FPGAs. However, in reality, the nodes can be heterogeneous, based on either FPGAs, processors, or both. In this paper, we model these heterogeneous reconfigurable systems using various parameters, including the computing capacities of the nodes, the size of memory, the memory bandwidth, and the network bandwidth. Based on the model, we propose a design for matrix multiplication that fully utilizes the computing capacity of a system and adapts to various heterogeneous settings. To illustrate our ideas, the proposed design is implemented on Cray XD1. Heterogeneous nodes are generated by using only the FPGAs or the processors in some nodes. Experimental results show that our design achieves up to 80% of the total computing capacity of the system and more than 90% of the performance predicted by the model.

1 Introduction

Field-Programmable Gate Arrays (FPGAs) are a form of reconfigurable hardware. They offer the design flexibility of software, but with time performance closer to Application Specific Integrated Circuits (ASICs). With the increasing computing power of FPGAs, high-end systems have been built that employ them as application-specific accelerators. Such systems include SRC 6 and 7[1], Cray XD1 and XT3[2], and SGI RASC[3]. 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[4,5].

Reconfigurable computing systems contain multiple forms of heterogeneity. For example, the nodes of these systems can be based on either processors, FPGAs, or both. Also, due to its reconfigurability, the computing capacity of the FPGA in a node varies based on the designs implemented on it. Moreover, the nodes have access to multiple levels of memory with various sizes and various bandwidths. To fully utilize the available computing capacity of the system, all these heterogeneity need to be considered.

In this paper, we propose a model to facilitate workload allocation and load balancing in heterogeneous reconfigurable systems. Each node of the system is characterized by various parameters, including its computing capacity, the available memory size, and memory bandwidth. The model also considers the time for transferring data from the main memory of the processors to the FPGAs and the network communication costs of executing an application over heterogeneous nodes.

Using the proposed model, we develop a design for floating-point matrix multiplication. The input matrices are partitioned into blocks and the block multiplications are distributed among the nodes. Multiple levels of memory available to the FPGAs are employed to reduce memory transfer time. In addition, the design scales across multiple heterogeneous nodes by overlapping the network communications with computations.

To illustrate our ideas, we implemented the proposed design on 6 nodes of Cray XD1 [6]. The nodes of XD1 contain both AMD Opteron processors and Xilinx Virtex-II Pro FPGAs. In our experiments, we chose to use only the FPGAs, or only the processors, or both to generate heterogeneous nodes. Experimental results show that in various heterogeneous settings, our design achieves up to 80% of the total computing capacity of the system. In addition, our design achieves more than 90% of the performance predicted by the model.

The rest of the paper is organized as follows. Section 2 introduces related work and presents several representative reconfigurable computing systems. Section 3 proposes our model. Section 4 proposes a design for matrix multiplication based on our model. Section 5 presents the experimental results. Section 6 concludes the paper.

2 Related Work and Background

Heterogeneous systems have been studied extensively [7,8]. Such systems contain processors with various computing capacities which are interconnected into a single unified system. Task scheduling algorithms have been proposed to minimize the execution time of an application[7,8]. In these systems, it is assumed that all tasks can be executed on any processors. However, this may not be true for reconfigurable computing systems because not all tasks are suitable for hardware implementation.

Heterogeneous distributed embedded systems combine processors, ASICs and FPGAs which are interconnected by communication links. Hardware/software co-synthesis techniques have been proposed to perform allocation, scheduling, and performance estimation[9,10]. However, such techniques cannot be applied to heterogeneous reconfigurable systems straightforwardly due to the complex memory hierarchy and multiple nodes in these systems.

Many reconfigurable computing systems have become available. One representative system is Cray XD1[2]. The basic architectural unit of XD1 is a compute blade, which contains two AMD 2.2 GHz processors and one Xilinx Virtex-II Pro XC2VP50. Each FPGA has access to four banks of QDR II SRAM, and to the DRAM memory of the processors. In SRC 7[1], the basic architectural unit contains one Intel microprocessor and one reconfigurable logic resource called MAP processor. One MAP processor consists of two FPGAs and one FPGA-based controller. Each FPGA has access to six banks of SRAM memory. The FPGA controller facilitates communication and memory sharing between the processor and the FPGAs. SGI has also proposed Reconfigurable Application Specific Computing (RASC) technology, which provides hardware acceleration to SGI Altix servers[3]. In an SGI RASC RC1000 blade, two Xilinx Virtex-4 FPGAs are connected to 80 GB SRAM memory. Each blade is directly connected to the shared global memory in the system through the SGI NUMAlink4 interconnect.

Hybrid designs have been proposed for reconfigurable computing systems that utilize both the processors and the FPGAs[4,11]. In some work, the computationally intensive part of a molecular dynamics simulation is executed on the FPGAs, while the remaining part runs on the processors[4]. In our prior work, the workload of several linear algebra applications is partitioned between the processors and the FPGAs[11] so that they are both effectively utilized. However, all these work assumes that the nodes are homogeneous and does not address the heterogeneity in the systems. The authors of[12] investigated scheduling algorithms for heterogeneous reconfigurable systems. However, that work only utilizes a single node while our work considers multiple nodes.

3 Model for Reconfigurable Computing Systems

Reconfigurable computing systems can be seen as distributed systems with multiple nodes connected by an interconnect network. The nodes are based on either general-purpose processors, FPGAs, or both. Each node has its own local memory, and may have access to the memory of a remote node. Suppose the system contains p nodes, including p_1 P(Processor)-nodes, p_2 F(FPGA)-nodes, and p_3 H(Hybrid)-nodes. The architectural model of such a system is shown in Figure 1. The nodes are denoted as $N_0, N_1, \ldots, N_{p-1}$. "GPP" in the figure stands for "general-purpose processor".

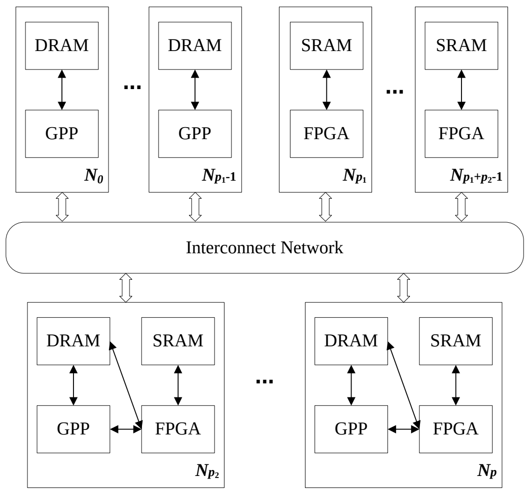

Figure 1. Architectural model of heterogeneous reconfigurable computing systems

In our model, we use C_i to refer to the computing capacity of N_i ($i = 0, \ldots, p-1$). As we target scientific computing applications, the computing capacity of a node equals its floating-point sustained performance for a given application. If N_i is a P-node, C_i is obtained by executing a program for the application. If N_i is an F-node, C_i is determined by the FPGA-based design for the application. Suppose the design performs O number of floating-point operations in each clock cycle and its clock speed is F. Thus, $C_i = O \times F$. For an H-node, C_i depends not only on the processor-based program and the FPGA-based design, but also on the workload partitioning and coordination between the processor and the FPGA.

Each node in the system has its own memory hierarchy. As we focus on FPGA-based designs, our model considers only the storage capacity and memory bandwidth available to the FPGAs. Designs on F-nodes have access to multiple levels of memory. The first level is the on-chip memory of the FPGA, usually Block RAMs (BRAMs). The second level is off-chip but on-board memory, which is usually SRAM. The FPGAs also has access to the DRAM memory of the adjacent processors.

In our model, the FPGA-based design reads the source data from the DRAM memory and stores the intermediate results in the SRAM memory. Thus, two important parameters are the size of the SRAM memory and the bandwidth between the FPGA and the DRAM memory. These two parameters are denoted as S and bw_d, respectively. Note that our model does not consider memory access latency because data are streamed into and out of the FPGAs for the applications considered. Therefore, the memory access latency is incurred only once for each task and is negligible.

We assume that the nodes are interconnected by a low latency, high bandwidth network. The network bandwidth between any two node is denoted as bw_n. In the system, only the processors in the P-nodes and the H-nodes can access the network directly. An F-node can only communicate with other nodes through the DRAM memory of a P-node or an H-node. The bandwidth of such access is denoted as bw_f, which is bounded by $\min\{bw_n, bw_d\}$. In this case, we say the F-node is "*related*" to the P-node or H-node. Without loss of generality, we assume that each F-node is related to only one P-node or H-node.

In our model, an F-node reads data from the DRAM memory of the adjacent processor. Because of spatial parallelism, the design on FPGA can continue to execute while the data are streaming in. However, because the processor moves the data from the DRAM to the FPGA, the computations on the processor cannot continue until the data transfer is complete. Also, the computations on the processor have to be interrupted while the processor receives the data sent to the F-node from network. Note that if multiple threads run on a processor, the memory transfer and the network communications can also be overlapped with the computations. However, in this case, additional overheads such as the cost of context switch and thread synchronization arise. As we are concerned with providing a simple yet effective model, we assume the processors are single-threaded.

4 Design for Matrix Multiplication

Consider computing $\mathbf{C} = \mathbf{A} \times \mathbf{B}$, where $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are $n \times n$ matrices. Each element of the matrices is a floating-point word.

4.1 Workload Allocation

To distribute the workload among the nodes in the system, we partition $\mathbf{B}$ into row stripes of size $k \times n$. The value of k depends on the FPGA-based matrix multiplier, which will be explained in Section 4.2. The tasks in the application are identified as $(n \times n) \times (n \times k)$ block multiplications. Each task contains n^2k floating-point multiplications and n^2k floating-point additions. The tasks contain few data dependencies so that they can be executed on any node. Suppose x_i tasks are assigned to N_i. N_i performs an $(n \times n) \times (n \times kx_i)$ matrix multiplication and is in charge of computing kx_i columns of $\mathbf{C}$. For load balance, we have $\frac{x_0}{C_0} \approx \frac{x_1}{C_1} \approx \cdots \approx \frac{x_{p-1}}{C_{p-1}}$.

We next adjust the workload allocation by overlapping memory transfer and network communications with the computations, as discussed in Section 3. To do so, we partition matrix $\mathbf{A}$ into column stripes of size $n \times k$. Thus, we overlap the multiplication of one stripe of $\mathbf{A}$ and one stripe of $\mathbf{B}$ with the memory transfer and network transfer of the subsequent stripes.

Suppose N_u is an F-node, and N_v is the P-node or H-node related to N_u. During the computations, N_v gets one stripe of $\mathbf{A}$ and one stripe of $\mathbf{B}$. The $\mathbf{A}$ stripe and $x_u \times k \times k$ words of the $\mathbf{B}$ stripe are then sent to N_u. Using these two stripes, N_v and N_u performs $2nk^2x_v$ and $2nk^2x_u$ floating-point operations, respectively. If N_v is a P-node, we have

$$\frac{2nk^2x_v}{C_v} + \frac{2nk}{bw_n} + \frac{nk + k^2x_u}{bw_f} \approx \frac{2nk^2x_u}{C_u} \tag{1}$$

If N_v is an H-node, the task allocated to it is partitioned between its processor and its FPGA. A simple partitioning method is to assign n_p rows of $\mathbf{A}$ to the processor and n_f rows to the FPGA. $n_p + n_f = n$ and $\frac{n_p}{n_f} \approx \frac{C_{vp}}{C_{vf}}$, where C_{vp} and C_{vf} are the computing capacities of the processor and the FPGA within N_v, respectively. Thus, we have

$$\frac{2n_pk^2x_v}{C_{vp}} + \frac{2nk}{bw_n} + \frac{n_fk + k^2x_v}{bw_d} + \frac{nk + k^2x_u}{bw_f} \approx \frac{2n_fk^2x_v}{C_{vf}} \approx \frac{2nk^2x_u}{C_u} \tag{2}$$

If no F-node is related to N_v, we have:

$$\frac{2n_pk^2x_v}{C_{vp}} + \frac{2nk}{bw_n} + \frac{n_fk + k^2x_v}{bw_d} \approx \frac{2n_fk^2x_v}{C_{vf}} \tag{3}$$

4.2 Proposed Design

In our design, matrices $\mathbf{A}$ and $\mathbf{B}$ are distributed among the nodes initially. In particular, $\frac{n}{p_1+p_3}$ columns of $\mathbf{A}$ and $\frac{n}{p_1+p_3}$ rows of $\mathbf{B}$ are stored in the DRAM memory of N_i, where N_i is either a P-node or H-node. During the computations, matrix $\mathbf{A}$ is read in column-major order, and matrix $\mathbf{B}$ is read in row-major order. The stripes are further partitioned into $k \times k$ submatrices. For each row stripe of $\mathbf{B}$, N_i node stores x_i such submatrices into its DRAM memory. When a submatrix in $\mathbf{A}$ is transferred to a node, it is multiplied with all the stored submatrices of $\mathbf{B}$ whose row indices are the same as its column index. The final results of $\mathbf{C}$ are transferred back to N_0, and are written back to the DRAM memory of N_0.

Each FPGA employs our FPGA-based design for matrix multiplication[13]. This design has k PEs. Each PE performs one floating-point multiplication and one floating-point addition during each clock cycle. The BRAM memory on the FPGA device serves as the internal storage of the PEs. Using an internal storage of size $\Theta(k^2)$ words, the design achieves the optimal latency of $\Theta(k^2)$ for $k \times k$ matrix multiplication[13]. This matrix multiplier utilizes the SRAM memory to minimize the amount of data exchanged between the DRAM memory and the FPGA.

As F-node N_u calculates kx_u columns of $\mathbf{C}$, it needs a storage of nkx_u words for the intermediate results of $\mathbf{C}$. In particular, $nkx_u \leq S$, where S is the size of the SRAM memory of the FPGA. On the other hand, the FPGA in H-node N_v is assigned n_{vf} rows of $\mathbf{A}$ and kx_v columns of $\mathbf{B}$. It needs to store $n_{vf}kx_v$ words of intermediate results of $\mathbf{C}$. Thus, we have $n_{vf}kx_v \leq S$.

When the matrix size is large, block matrix multiplication is performed. Matrices $\mathbf{A}$ and $\mathbf{B}$ are partitioned into blocks of size $b \times b$. The value of b is determined using $bkx_u \leq S$ and $b_{vf}kx_v \leq S$.

5 Experimental Results

To illustrate our ideas, we implemented our design on 6 nodes of XD1. In each node, we used at most one AMD 2.2 GHz processors and one Xilinx Virtex-II Pro XC2VP50. The FPGA-based designs were described in VHDL and were implemented using Xilinx ISE 7.1i[14]. Our own 64-bit floating-point adders and multipliers that comply with IEEE-754 standard were employed[15]. A C program was executed on the processor, and was in charge of file operations, memory transfer and network communications.

5.1 Detailed Model for XD1

Based on the proposed model in Section 3, we developed a detailed model for XD1 and used it for our design. In XD1, $bw_n = 2$ GB/s. As the FPGA-based design gets one word from the DRAM memory in each clock cycle, $bw_d = 1.04$ GB/s. On each node, 8 MB of SRAM memory is allocated to store the intermediate results of $\mathbf{C}$, hence $S = 2^{23}$ bytes. To satisfy the memory requirement in Section 4.2, we set $n = 3072$.

When our FPGA-based matrix multiplier[13] is implemented on one FPGA in XD1, at most 8 PEs can be configured. Each PE performs two floating-point operations in each clock cycle and our design achieves a clock speed of 130 MHz. Thus, the computing capacity of an F-node is $16 \times 130 \times 10^6 = 2.08$ GFLOPS. To obtain the sustained performance of the processor for matrix multiplication, *dgemm* subroutine in AMD Core Math Library (ACML)[16] was executed. Using this subroutine, the computing capacity of a P-node is approximately 3.9 GFLOPS. According to Equation 3, an H-node achieves a sustained performance of 5.2 GFLOPS for matrix multiplication.

5.2 Workload Allocation

In our experiments, we chose to use only the FPGAs or the processors in some nodes to achieve heterogeneity.

In Setting 1, all the nodes are H-nodes. As the nodes are homogeneous, the number of tasks assigned to N_i, x_i, equals $\frac{n}{6k}$, ($i = 0, 1, \ldots, 5$). Using Equation 3, we partition the workload among the processor and the FPGA so that $n_{if} = 1280$ and $n_{ip} = 1792$.

In Setting 2, there are three P-nodes and three F-nodes. Each F-node is related to a P-node. The source matrices are sent to the P-nodes first. Each P-node then sends the data to the related F-node. The computations on the P-nodes do not continue until these network communications are complete. The task allocation is determined according to Equation 1. If N_i is a P-node, $x_i = 204$; if it is an F-node, $x_i = 180$.

In Setting 3, there are two P-nodes, two F-nodes and two H-nodes. The source matrices are sent to the P-nodes and the H-nodes first. P-nodes then start their computations while each H-node sends the data to the F-node related to it. After that, the processor of each H-node transfers part of the data to the FPGA within the node. We perform the partitioning according to Equation 2. For a P-node, $x_i = 54$; for an F-node, $x_i = 46$. If N_i is an H-node, $x_i = 92$, $n_{if} = 1440$, and $n_{ip} = 1632$.

5.3 Performance Analysis

Figure 2 shows the floating-point performance achieved by our design for the three settings defined above. It also shows the sum of the computing capacities of the nodes in each set-

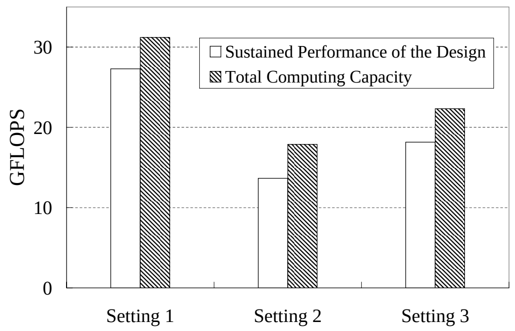

Figure 2. Performance of the proposed design under various heterogeneity

ting, which is denoted as C_{sum}. $C_{sum} = \sum_{i=0}^{5} C_i$. In Setting 1, our design overlaps more than 90% of the memory transfer time and more than 80% of the network communication time with the computations. It achieves more than 85% of C_{sum}. In Setting 3, because of the related F-nodes, the processors in the H-nodes spend more time on network communications. Thus, the design achieves 80% of C_{sum}. In Setting 2, the P-nodes cannot perform computations when they are receiving data and sending data to the F-nodes. Thus, only 70% of the network communication time is overlapped with the computations. In this case, our design achieves about 75% of C_{sum}.

Figure 2 shows that our design can adapt to various heterogeneous settings. When the total computing capacity in the system increases, the sustained performance of our design also increases. In XD1, our design achieves up to 27 GFLOPS for double-precision floating-point matrix multiplication.

We also used the proposed model for performance prediction. To do so, we use the same system parameters and the same workload allocation as in the experiments. However, we assume all the communication costs and memory transfer time are overlapped with the computations on the FPGAs. As our design efficiently hides the memory transfer and network communication costs, it achieves more than 90% of the predicted performance in all three settings. The figure is not shown here due to page limitation.

6 Conclusion

In this paper, we proposed a model for reconfigurable computing systems with heterogeneous nodes. The model describes a system using various parameters, including the computing capacity of the nodes, the size of the memory available to the nodes, the available memory bandwidth, and the network bandwidth among the nodes. Matrix multiplication was used as an example and was implemented on Cray XD1. Experimental results show that our design for matrix multiplication can adapt to various heterogeneous settings, and its performance increases with the total computing capacity in the system. The proposed model can also provide a fairly accurate prediction for our design. In the future, we plan to develop detailed models for more reconfigurable computing systems, such as Cray XT3 and SGI Altix350. We also plan to use the model for more complex applications.

Acknowledgments

This work is supported by the United States National Science Foundation under grant No. CCR-0311823, in part by No. ACI-0305763 and CNS-0613376. NSF equipment grant CRI-0454407 and equipment grant from Xilinx Inc. are gratefully acknowledged.

References

1. SRC Computers, Inc. `http://www.srccomp.com/`.
2. Cray Inc. `http://www.cray.com/`.
3. Silicon Graphics, Inc. `http://www.sgi.com/`.
4. R. Scrofano, M. Gokhale, F. Trouw, and V. K. Prasanna. A Hardware/Software Approach to Molecular Dynamics on Reconfigurable Computers. In *Proc. of the 14th IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2006.
5. L. Zhuo and V.K. Prasanna. Scalable Hybrid Designs for Linear Algebra on Reconfigurable Computing Systems. In *Proc. of the 12th International Conference on Parallel and Distributed Systems (ICPADS)*, July 2006.
6. Cray Inc. Cray XD1™. `http://www.cray.com/products/xd1/`.
7. B. Hamidzadeh, D.J. Lilja, and Y. Atif. Dynamic Scheduling Techniques for Heterogeneous Computing Systems. *Concurrency: Practice & Experience*, 7(7), October 1995.
8. H. Siegel and S. Ali. Techniques for Mapping Tasks to Machines in Heterogeneous Computing Systems. *Journal of Systems Architecture*, 46(8), June 2000.
9. H. Oh and S. Ha. A Hardware-Software Cosynthesis Technique Based on Heterogeneous Multiprocessor Scheduling. In *Proc. of the 7th International Workshop on Hardware/Software Codesign*, May 1999.
10. B.P. Dave, G. Lakshminarayana, and N.K. Jha. COSYN: Hardware-Software Co-Synthesis of Heterogeneous Embedded Systems. *IEEE Transactions on Very Large Scale Integration Systems*, 7(1), March 1999.
11. L. Zhuo and V.K. Prasanna. Hardware/Software Co-Design on Reconfigurable Computing Systems. In *Proc. of the 21st IEEE International Parallel & Distributed Processing Symposium*, March 2007.
12. Proshanta Saha and Tarek El-Ghazawi. Applications of Heterogeneous Computing in Hardware/Software Co-Scheduling. In *Proc. of 2007 ACS/IEEE International Conference on Computer Systems and Applications*, May 2007.
13. L. Zhuo and V.K. Prasanna. Scalable and Modular Algorithms for Floating-Point Matrix Multiplication on FPGAs. In *Proc. of the 18th International Parallel and Distributed Processing Symposium*, April 2004.
14. Xilinx Incorporated. `http://www.xilinx.com`.
15. G. Govindu, R. Scrofano, and V.K. Prasanna. A Library of Parameterizable Floating-Point Cores for FPGAs and Their Application to Scientific Computing. In *Proc. of International Conference on Engineering Reconfigurable Systems and Algorithms*, June 2005.
16. AMD Core Math Library. `http://developer.amd.com/acml.aspx`.