Automatic System Architecture Synthesis for FPGA-based Reconfigurable Computers

Colin Yu Lin, Ngai Wong, Hayden Kwok-Hay So

Department of Electrical and Electronic Engineering, University of Hong Kong
Pok Fu Lam Road, Hong Kong
{linyu, nwong, hso}@eee.hku.hk

I. Introduction and Motivation

As reconfigurable computing and Field Programmable Gate Array (FGPA) technology advance, high-performance applications are choosing FPGA-based reconfigurable computers to accelerate their computations. Reconfigurable hardware has demonstrated remarkable speedups on computation-intensive applications [1]. However, the performance of reconfigurable hardware is very sensitive to the quality of the application implementation. Significant challenges remain in development of systematic scheduling and mapping techniques for computing architectures [2].

In [2], a computing model and mapping algorithms are developed to exploit features of reconfigurable hardware for computer vision applications. This model may not be applicable to general applications. In some implementations, for example in [3], carefully hand-optimized architecture design for the target computation is developed to achieve high system performance. But it is difficult to apply such technique generally to all applications. In [4], Herbordt *et al.* present design techniques to achieve significant speedups on reconfigurable architecture without expending exorbitant developement effort. However, as pointed out by the authors, neither low-level methods related to logic design and synthesis in electronic design automation nor high-level methods such as scheduling are covered in their research.

In this research work, we are going to develop an automatic method of system architecture synthesis on FPGA-based reconfigurable computers for general high performance applications. This architecture synthesis problem is a hardware/software codesign problem, which makes balancing decisions carefully in implementation of high-performance applications on FPGA-based reconfigurable computers, both in tools and in architecture.

Currently we have addressed the application operation scheduling problem on a target reconfigurable architecture using a time-indexed integer linear programming (ILP) formulation based on the well-studied resource-constrained project scheduling problem. Although the time-indexed ILP formulation captures all aspects of our scheduling problem, solving such ILP is, in general, computationally intractable for even moderately sized problems. As a result, we have developed a variation of the Graham's list scheduling algorithm that gives approximation to the optimal solution. Using this scheduling model and list scheduling algorithm, we are progressing to implement system architecture in physical computational medium.

II. Previous Achievements

In order to limit our scope, we make certain assumptions about the target reconfigurable system. In our scheduling model, only one FPGA and two off-chip memory devices are considered. One off-chip memory is used for input data while the other one used for output. At any one point in time, the maximum number of words that the FPGA can read from memory and/or write to memory is the bandwidth **B**. All data in the input matrices have the size of a word. We assume **P** processing elements (PEs) are implemented. The PEs may be reconfigured to perform add, multiply or multiply-accumulate in each step. There are **M** words of on-chip memory that act as local buffers. A PE may access any memory location and all PEs may access the memory at the same time. Furthermore, we assume that I/O operations and the PEs may operate in parallel.

The operation scheduling problem can be formulated as minimizing system latency and and I/O operations, subject to the a) input data, b) operation, c) input-operation data dependency, d) operation-operation data dependency, and e) on-chip memory constrains.

In order to obtain a feasible and near-optimal approximation to our scheduling problem, a list scheduling algorithm is used. The main procedure is fairly straightforward. For each control step, $p \leq \mathbf{P}$ operations and $b \leq \mathbf{B}$ data input with highest priorities in the ready queues are scheduled to take place. Once the operations are scheduled, priorities of nodes with the same successor of this node are updated. The effectiveness of our algorithm therefore lies with the way data and compute operations are selected in each step. This selection process depends on three factors: (i) the state of an operation; (ii) the priority of an operation; and (iii) the available system resource.

In theory, multiplying two n-by-n matrices requires n^3 multiply accumulate operations. In [3], the hand-optimized systolic array architecture achieved a near-optimal latency of $n^3 + n$ when the available on-chip memory capacity $M \geq n^2 + 2n$. Multiplications are scheduled automatically using our list scheduling and the results are shown in Table I.

For large n, our list scheduling algorithm can successfully schedule operations with latency and memory requirements

TABLE I

MATRIX-MATRIX MULTIPLICATIONS USING OUR LIST SCHEDULING ALGORITHM (**L**), THE SYSTOLIC ARRAY IMPLEMENTATION (**S**) [3], AND THEORETICAL VALUES (**T**). THE OVERHEADS OF OUR SOLUTION WHEN COMPARED TO THE SYSTOLIC ARRAY ARE LISTED UNDER THE COLUMN **O(%)**. $\mathbf{P} = 40, \mathbf{B} = 2, \mathbf{M} = 50000$.

ranks	latency			
	L	**S**	**T**	**O(%)**
40	1904	1640	1600	16.10
60	5854	5460	5400	7.22
80	13404	12880	12800	4.07
100	25754	25100	25000	2.61
120	44104	43320	43200	1.81
140	69654	68740	68600	1.33
160	103604	102560	102400	1.02
180	147154	145980	145800	0.80
200	201504	200200	200000	0.65

ranks	memory required		
	L	**S**	**O(%)**
40	1681	1680	0.06
60	3721	3720	0.03
80	6561	6560	0.02
100	10200	10200	0.00
120	14640	14640	0.00
140	19880	19880	0.00
160	25920	25920	0.00
180	32760	32760	0.00
200	40400	40400	0.00

close to the hand-optimized systolic array implementation. Although it doesn't work well for small n, the result is encouraging as no manual intervention was needed to obtain such near-optimal result.

Detail explanation and more results of the scheduling model and list scheduling algorithm can be found in [5].

III. CURRENT WORK

Although the list scheduling algorithm worked well in theory, second-order effects, such as the extra latency associted with the on-chip communication network, will inevitably offset the performance of the implemented design. As a result, we are currecntly working on an extended scheduling algorithms that takes into accout low-level implementation constraints, such as the exact organization of PEs with respect to the on-chip memory. Begining with a study of the use of a systolic on-chip architecture, we are progressively extending the work towards a flexible and fully automated architectural compilation framework.

As shown in Fig. 1, we proposed a systolic alike on-chip architecture. PEs are lined in a column and only connected to their neighbor(s) by two data-paths, one from up to down and the other from bottom to top. Only the PE on the top can access off-chip memory for I/O operation. In the implementation of each PE, a dedicated DSP block is used, and it can

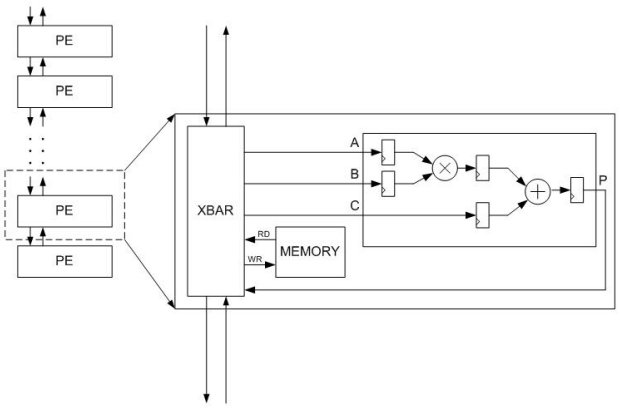

Fig. 1. Proposed Architecture

perform add, multiply or multiply-accumulate in each step. Also, private memory is implemented in each PE. Finally, a crossbar block is used to select data from data-paths to DSP block and private memory, and to determin data to pass to its neighbor from DSP block, memory or data-path.

IV. CONCLUSION

The goal of this PhD project is to develop an automatic method of system architecture synthesis for general high-performance applications on FPGA-based reconfigurable computers. Through our previous research, we have built a theoretical model targeting the scheduling problem with first-order hardware constraints. And a list scheduling algorithm is developed to achieve near-optimal performances. Currently, we are working on the low-level implementation. A systolic architecture is used, and the list scheduling algorithm will be extended to take into account constraints deriving from exact hardware architecture.

REFERENCES

[1] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung, "Reconfigurable computing: architectures and design methods," in *IEE Proceedings - Computers and Digital Techniques*, 2005, pp. 193–207.

[2] K. Bondalapati and V. K. Prasanna, "Reconfigurable computing systems," *Proc. IEEE*, vol. 90, no. 7, pp. 1201–1217, 2002.

[3] J.-W. Jang, S. B. Choi, and V. K. Prasanna, "Energy- and time-efficient matrix multiplication on fpgas," *IEEE Trans. VLSI Syst.*, vol. 13, no. 11, pp. 1305–1319, 2005.

[4] M. C. Herbordt, T. V. Court, Y. Gu, B. Sukhwani, A. Conti, J. Model, and D. DiSabello, "Achieving high performance with FPGA-based computing," *IEEE Computer*, vol. 40, no. 3, pp. 50–57, 2007.

[5] C. Y. Lin, N. Wong, and H. K.-H. So, "Operation scheduling for FPGA-based reconfigurable computers," in *Proc. 19th International Conference on Field Programmable Logic and Applications (FPL)*, Prague, Czech Republic, Aug. 2009, pp. 481–484.