Analytical and Speedup Models for Performance Evaluation of a Generic Reconfigurable Coprocessor (RC) Architecture

Syed S. Rizvi[1], Aasia Riasat[2], Muhammad S. Rashid[3]

Computer Science & Engineering Department, University of Bridgeport[1, 3], Bridgeport, CT
Department of Computer Science, Institute of Business Management[2], Karachi, Pakistan
{srizvi[1], muhammsi[3]}@bridgeport.edu, aasia.riasat@iobm.edu.pk[2]

Abstract

New analytical and the speedup models for evaluating the performance of a generic reconfigurable coprocessor (RC) system are presented. We present a generic performance model for the speedup of a generic RC system. We demonstrate how different parameters of speedup model can affect the performance of reconfigurable system (RS). In addition, we implement our pre-developed speedup model for a system that permits preloading functional blocks (FB) into the reconfigurable hardware (RH). The redevelopment of speedup model with the consideration of preloading demonstrates some interesting results that can be used to improve the performance of RH with a coprocessor. Our experiments show that the minimum and the maximum speedup mainly depend on the probabilities of miss and hit for the FB resides in the RH of coprocessor.

Index Terms—Field programmable gate array, reconfigurable coprocessors, reconfigurable hardware.

1. Introduction

Reconfigure systems have provided significant performance improvements by adapting to computations not well served with current processor architectures [5]. A programmable processor ceaselessly follow a three-phase implementation, where an instruction is first fetched from memory, after which it is decoded, then to be passed on to the final execution phase. The third phase of this implementation may require several clock cycles. This process is then repeated for the next instruction, and so on. An RC, on the other hand, can be regarded as non-iterative fetch phase. One of the main advantages of this approach is that the configuration string that fetches from memory requires no further interpretation and is directly used to configure the hardware. No further phases or iterations are needed as the processor is now configured for the task at hand [9]. The primary strength of a RC or functional unit is the ability to customize hardware for a specific program's requirements [5]. In order to efficiently use RC, one of the ways is to treat the reconfigurable logic not as a fixed resource but instead as a cache for the reconfigurable FB instructions. Those instructions that have recently been executed, or that we can otherwise predict might be needed soon, are kept in the reconfigurable logic. For instance, if an instruction is needed, it is brought into the FB by preloading it in or before the cycle in which it is required. In this way, the system uses partial run-time reconfiguration techniques to manage the reconfigurable logic. Since the reconfigurable logic is somewhat symmetric, a given instruction may be placed into the FB wherever there is space available. This way the RC gives quite significant speedup over the conventional core processor.

The goal of this paper is to compare the performance of a generic RC system with the performance of conventional core processors that use software implementation (SI). Further motivating our study is the large role of FB organization in the RH that may limit overall performance. We observe that most of the applications now a days demand not only sufficient quality of service (QoS) data transmission but also a

high-bandwidth host bus. If these demands persist, the conventional implementation of RC systems would limit the achievable performance.

2. Related work

Recently, computer architectures that connect a RC to a general-purpose processor have been proposed [1, 4]. The advantage of this approach is that the coprocessor can be reconfigured to improve the performance of a particular application. All of these proposed architectures use field programmable gate arrays (FPGAs) for the RH. The FPGA architecture, such as the small width of the programmable logic blocks and the programmable interconnection network, provides a great flexibility for the systems [7].By combining the computational density of the spatial computing paradigm and the flexibility of programmable platforms, a delicate tradeoff can be achieved. The GARP [3] architecture combines a reconfigurable array with a standard MIPS processor. The host processor executes the MIPS-II instruction set extended with instructions for the reconfigurable array of LUT-based two bit processing elements. Significant speedup for certain applications has been reported. MorphoSys [2] was targeted at applications with inherent data-parallelism, high regularity and high throughput requirements. The architecture comprises a RISC processor, a reconfigurable cell array and a high bandwidth memory interface. Each reconfigurable cell is a coarse grain unit with an ALU multiplier and a register file. Each of the above mentioned approaches have its benefits and limitations. RS integrates a high performance processor with a reconfigurable functional unit into the same chip (Garp [3], Remarc [8], PipeRench I-COP [4] etc.).

3. Proposed performance models for a generic RC system

Before going to present a performance model for the speedup of a RS, we discuss the main design methodology adopted behind the speedup model as well as some important assumptions common for both models present in this section. Model variables, along with their definition are listed in Table I. We start developing the first performance model for an ideal case where we assume that all the required FB are always present in the RH and no dependencies exist between the instructions etc. Based on the above assumptions, we can define the speedup as a ratio of execution time for a certain task using the SI to the execution time for the same task using the RH. The portion of enhancement for a complete application can be defined as a ratio of enhanced-time due to the RH to the normal time using the SI. This can be expressed as follows:

$$P_e = E_t / N_t \text{ where } 0 < P_e < 1 \tag{1}$$

Therefore, (1) clearly indicates that the resultant speedup for the system should be the reciprocal of the portion of enhancement and must be greater that one. This leads us to the following mathematical expression for the speedup of a system where the probability of miss almost reaches to zero.

$$Speedup = K = 1/P_e = N_t/E_t \text{ where } K > 1 \tag{2}$$

One common point that we can observe in both (1) and (2) is the fact that the enhanced-time should be less than the normal time processor takes to execute a certain amount of task.. Mathematically it can be expressed as follows:

$$E_t = N_t \left[(1-P_e) + (P_e/K) \right] \tag{3}$$

Substituting *K* from (2) into (3), yields,

$$E_t = N_t\left[(1-P_e)+P_e\Big/\left(\frac{N_t}{E_t}\right)\right] \tag{4}$$

After performing some simplification, we get

$$E_t/N_t = (E_t)^2+(N_t)^2-E_tN_t/(N_t)^2 \tag{5}$$

Since $K = N_t/E_t$, the final equation of the speedup can be written as:

$$Speedup = K = (N_t)^2/(E_t)^2-E_tN_t+(N_t)^2 \tag{6}$$

It should be noticed that (6) can not speedup the system more than the reciprocal of one minus the portion of enhancement (i.e. $K<1/1\text{-}P_e$) as shown in Fig.1. Fig.1 shows the comparison of the required execution time between the RH and the SI with respect to specific values of speedup. A normal-execution time for a system that does not support RH should equal to the sum of normal-execution time between the FB and the time it takes to execute a FB on the processor in software. Mathematically, this simple relationship can be expressed as:

$$N_T = T_N + T_B \tag{7}$$

TABLE I
System Parameters Definition

Parameter	Definition
E_T	Enhanced execution time for FB
T_C	FB call time
T_P	Reconfigurable programming time
P_E	Portion of enhancement
T_{RFB}	Execution time for a reconfigurable FB
T_{PFB}	Time to preload the FB into RH
T_{INI}	Time required to perform initialization
T_{CL}	Cleanup time
T_I	Preloading initiation time
T_{BC}	Time required to perform basic computation
T_{RD}	Time required to resolve dependencies
C	Constant value
NA	Ignored
P_d	Probability of dependency among N instructions
N	Given set of instructions
P_h	Probability of a hit
K	Speedup
P_m	Probability of a miss
N_T	Normal execution time for FB
T_N	Normal execution time between FB
T_B	Normal FB execution time

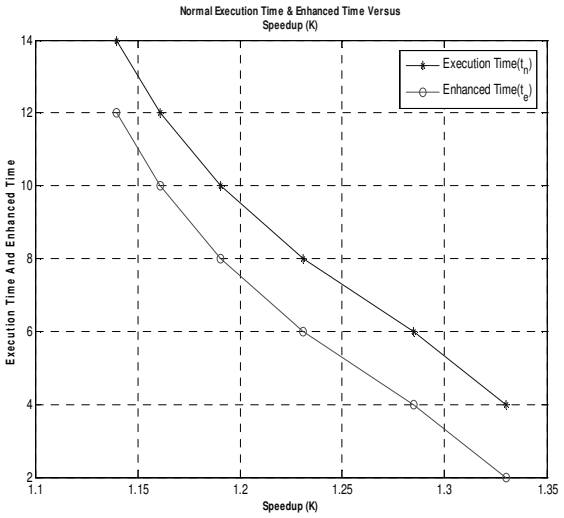

Fig. 1. Normal execution-time and enhanced-time versus the probability of hit

According to one of our assumptions, a miss in the RH yields a penalty of T_P cycles. This, therefore, implies that we should multiply the time required to call a FB in the RH with the probability of miss. The above assumptions lead us to the following mathematical expression:

$$E_T = T_N + T_C\left(1-P_h\right) + T_{RFB} + T_P \tag{8}$$

It can be seen in Fig. 2 that the best performance from the RH can be achieved with the probability of hit approaching its maximum value. On the other hand, the worst-case performance of the RH can also be noticed in Fig. 2 when the probability of miss approaching its maximum value. Once the probability of hit goes down by 50%, the SI outperforms the RC systems. The portion of enhancement and the speedup of a RH system for the real model can be expressed as follows:

$$P_e = \left[\left\{T_N + T_C(1-P_h) + T_P\right\} \Big/ \left(T_N + T_B\right)\right] \tag{9}$$

$$N_T \Big/ E_T = \left(T_N + T_B\right) \Big/ T_N + T_C(1-P_h) + T_P \tag{10}$$

In order to derive the expression for the speedup, we implement the concept of (3) on (9) and (10). This combine implementation results (11) as follows:

$$K = 1 \Big/ \left[1 - \left(T_C(1-P_h) + T_N + T_P \Big/ T_N + T_B\right) + \left(\frac{T_N + T_C(1-P_h) + T_P}{T_N + T_B} \div \frac{T_N + T_B}{T_N + T_P + T_C(1-P_h)}\right)\right] \tag{11}$$

After performing some simplification, we get

$$K = \left(T_N + T_B\right)^2 \Big/ \left\{\left(T_B - T_C(1-P_h) - T_P\right)\left(T_N + T_B\right)\right\} + \left(T_N + T_C(1-P_h) + T_P\right)^2 \tag{12}$$

It can be seen in Fig. 3 that how much faster an FB executes on average in the RH over a SI through a core processor. In addition, Fig. 3 demonstrates the change in the performance of a RH with respect to the changes in the probability of hit. The implementation of the preloading with the RH does not fully utilize the normal-execution time between the FBs. The preloading of the FB into the RH is limited to the partial use of normal-execution time of a processor, since an instruction can not be called until it is clear that an FB is really required. This leads us to the following sequential implementation of the preloading: (1) a signal generates that indicates that an FB is required, (2) an instruction executes that initiates the preloading of the required FB, and (3) the loading of the required FB into the RH starts that may complete while another call for the FB initiates. This sequence of implementation implies that the preloading of FB into the RH does not make scene for the first cycle. In order to develop a realistic speedup model, one should also include the startup latency in computing the execution time for the RH. As the applications run, different portion of the running application may need a FB from a RH being determined and preloaded ahead of the actual FB-call. Since we assume that we have startup latency, we do not need to consider the time required to initiate the preloading of FBs.

$$E_T = \left[\tfrac{1}{2}(T_N) + T_C\left(P_m\right) + \tfrac{1}{2}(T_P) + T_I + T_{RFB}\right] + \left(T_{PFB}\right)_C \tag{13}$$

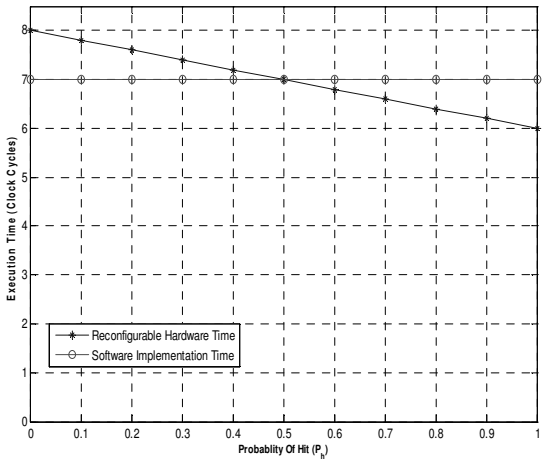

Fig. 2. RH and SI versus the probability of hit

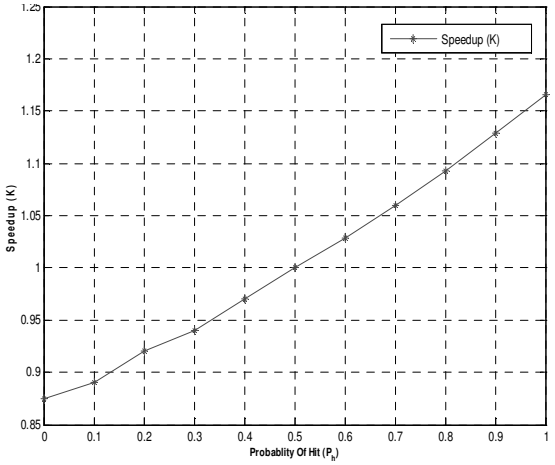

Fig. 3. Speedup for the RH versus the probability of hit

Equation (13) represents the enhanced-time for the worst case scenario where each system parameter may have non ideal values. Likewise, the enhanced-time for the best case can be expressed as:

$$E_T = \left[\tfrac{1}{2}(T_N)+1+T_{RFB}\right] + \left(T_{PFB}\right)_C \quad where \quad \left(T_C \to 1 \; and \; T_P \to 0, \; when \; P_m \to 0\right) \; and \; T_I \to NA \tag{14}$$

Equation (14) shows the enhanced-time computation for the best-case scenario where all system parameters may have ideal values. It should be noted that T_{RFB} in (13) represents a time required to execute a FB in the RH. This time is extremely short since we use dedicated hardware in a coprocessor that takes very short time compare to the SI. In addition, we should consider the fact that the RH does not have the first two phases of ordinary instruction execution through a conventional core processor. This fact, therefore, permits us to ignore the time required to execute an FB within a RH once a call to execute the required FB is initiated. Equation (13) can also be written as:

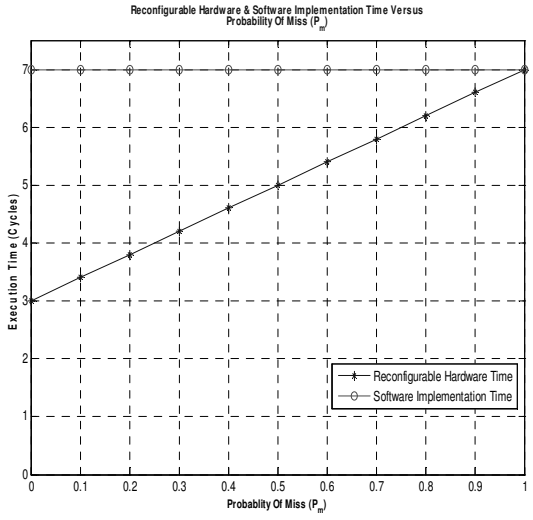

Fig. 4. RH and SI versus the probability of miss

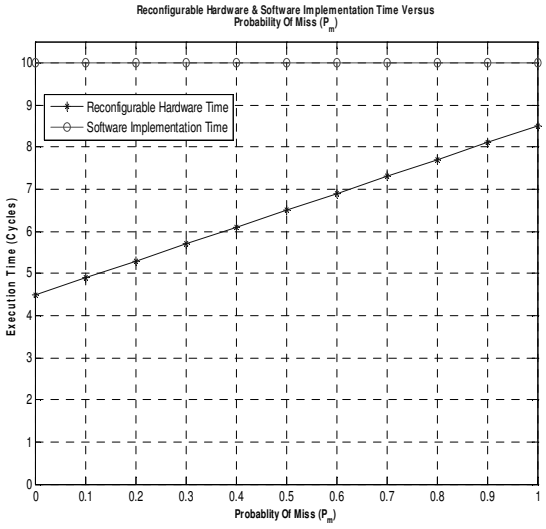

Fig. 5. RH and SI versus the probability of miss

$$E_T = \left[\tfrac{1}{2}(T_N) + T_C(P_m) + \tfrac{1}{2}(T_P) + T_I \right] + (T_{PFB})_C \quad (15)$$

Using (7) and (15) and, the mathematical expression for the speedup model can be expressed as follows: Since $K = 1/(1-E_T/N_T)+(E_T/N_T)^2$

$$K = \frac{1}{\left[1 - \frac{\left[\left\{\frac{1}{2}(T_N) + T_C(P_m) + \frac{1}{2}(T_P) + T_I + T_{RFB}\right\} + (T_{PFB})_C\right]}{(T_N + T_B)}\right] + \left[\frac{\left[\left\{\frac{1}{2}(T_N) + T_C(P_m) + \frac{1}{2}(T_P) + T_I + T_{RFB}\right\} + (T_{PFB})_C\right]}{(T_N + T_B)}\right]^2} \quad (16)$$

Fig. 4 demonstrates the performance of both RH and the SI with respect to a varying amount of P_m. The speedup model for preloading systems does not have any sever effects on the performance of the RH compare to its effects on the SI. This implies that an increase in T_N increases the total normal-execution time with a large magnitude compare to the RH. It can be seen in Fig. 5 that T_N increases the SI time by a large magnitude (approximately 43%) compare to an increase in the RH implementation.

4. Conclusion

We presented variety of simulation and numerical results that compare the performance of RH coprocessor system with the conventional general purpose processors. Furthermore, we presented the performance model of a system which allows preloading FB into the RH. We explored various system parameters and studied their impact on the system performance. Numerical results suggest that the RH coprocessors will be able to achieve significant speedup when implement with a good FB management system that can ensure the availably of required FB into the RH.

References

[1] Andre DeHon, “Reconfigurable Architectures for General-Purpose Computing,” *A.I. Technical Report,* No. 1586, Artificial Intelligence Laboratory, MIT, 1996.

[2] Hartej Singh, Ming-Hau Lee, Guangming Lu, Fadi J, Kurdahi, and Nader Bagherzadeh, ”MorphoSys: An Integrated Reconfigurable System for Data Parallel Computation-Intensive Applications,” *Transactions on Computers*, Vol. 49, Issue 5, pp. 465 – 481, May 2000.

[3] T. J. Callahan, J. R. Kouser, and J. Wawrzynek, “The GARP Architecture and C Compiler,” *IEEE Computer*, Vol. 33, Issue. 4, pp: 62 – 69, April 2000.

[4] Y. Chou, P. Pillai, H. Schmit, and J. P. Shen, “Pipe-Rench Implementation of the Instruction Path Coprocessor,” Proceedings. 33rd Annual IEEE/ACM International Symposium on *Micro-architecture*, pp.147-158, 2000.

[5] Scott Hauck, Thomas W. Fry, Matthew M. Hosler, and Jeffrey P. Kao, "The Chimaera Reconfigurable Functional Unit,” *IEEE Symposium on FPGAs for Custom Computing Machines*, Vol. 12, Issue. 2, pp. 206 – 217, Feb. 2004.

[6] J. R. Hauser and J. Wawrzynek, "GARP: A MIPS processor with a reconfigurable coprocessor," IEEE Workshop on FPGAs for Custom Computing Machines, pp. 12 – 21, April 16-18, 1997.

[7] T. Miyamori, and K. Olukotun, “A Quantitative Analysis of Reconfigurable Coprocessors for Multimedia Applications,” *IEEE Symposium on FPGAs for Custom Computing Machines,* pp. 2 – 11, Proceedings FCCM, 1998.

[8] Sanchez, E. et. al., [1999]. "Static and Dynamic Configurable Systems," *IEEE Trans. On Computers*, Vol. 48, Issue. 6, pp. 556-563, June 1999.

Author biographies

SYED S. RIZVI is a Ph.D. student of Computer Engineering at the University of Bridgeport. He received a B.S. in Computer Engineering from Sir Syed University of Engineering and Technology and an M.S. in Computer Engineering from Old Dominion University in 2001 and 2005 respectively. In the past, he has done research on bioinformatics projects where he investigated the use of Linux based cluster search engines for finding the desired proteins in input and outputs sequences from multiple databases. For last one year, his research focused primarily on the modeling and simulation of wide range parallel/distributed systems and the web based training applications. Syed Rizvi is the author of 15 scholarly publications in various areas. His current research focuses on the design, implementation and comparisons of algorithms in the areas of multiuser communications, multipath signals detection, multi-access interference estimation, computational complexity and combinatorial optimization of multiuser receivers, peer-to-peer networking, and reconfigurable coprocessor and FPGA based architectures.

AASIA RIASAT is an Associate Professor of Computer Science at Collage of Business Management (CBM) since May 2006. She received an M.S.C. in Computer Science from the University of Sindh, and an M.S in Computer Science from Old Dominion University in 2005. For last one year, she is working as one of the active members of the wireless and mobile communications (WMC) lab research group of University of Bridgeport, Bridgeport CT. In WMC research group, she is mainly responsible for simulation design for all the research work. Aasia Riasat is the author or co-author of 10 scholarly publications in various areas. Her research interests include modeling and simulation, web-based visualization, virtual reality, data compression, and algorithms optimization.

RASHID is a M.S. student of Computer Engineering at the University of Bridgeport. His research interest includes packet and circuit switching networks, high performance computing, and reconfigurable hardware.