

Timing Issues of Operating Mode Switch in High Performance Reconfigurable Architectures^{*}

Rama Sangireddy, Huesung Kim, and Arun K. Somani

Dependable Computing & Networking Laboratory
Department of Electrical and Computer Engineering
Iowa State University, Ames, IA 50011, USA
{sangired,huesung,arun}@iastate.edu

Abstract. The concept of a reconfigurable coprocessor controlled by the general purpose processor, with the coprocessor acting as a specialized functional unit, has evolved to accelerate applications requiring higher computing power. The idea of Adaptive Balanced Computing (ABC) architecture has evolved, where a module of Reconfigurable Functional Cache (RFC) is configured with a selective core function in the application whenever a higher computing resources are required. Initial results have proved that the ABC architecture provides with speedups ranging from 1.04x to 5.0x depending on the application and the speedups in the range of 2.61x to 27.4x are observed for the core functions. This paper further explores the issues of management of RFC, where the impact of various schemes for configuration of core function into the RFC module is studied. This paper also gives a detailed analysis on the performance of ABC architecture for various configuration schemes, including the study of the effect of the percentage of the core function in an entire application over the management of RFC modules.

1 Introduction

The technology of reconfigurable hardware, to implement a computation intensive function in a single unit whenever required, has evolved to accelerate the execution of selective functions and these reconfigurable chips are used as coprocessors in tandem with the general purpose processor. To accelerate structured and regular computations, such as DSP and multimedia applications, FPGA-like reconfigurable logic that is specialized for these computations has been developed and integrated into on-chip microprocessors [1–4].

Current processor designs often devote a largest fraction (up to 80%) of on-chip transistors to caches. However, many workloads, like media processor applications, do not fully utilize the large cache resources due to streaming nature and lack of temporal locality for the data. From these observations, an idea of a different kind of computing machine - Adaptive Balanced Computing (ABC) architecture, has evolved. ABC uses a dynamic configuration of a part of on-chip cache memory to convert it into a specialized computing unit. A reconfigurable functional cache (RFC) operates as a conventional cache memory or a specialized computing unit [7]. This paper explores two methodologies for the configuration

^{*} The research reported in this paper is partially funded by the grants Carver's Trust and Nicholas Professorship from Iowa State University, and grant No. CCR9900601 from NSF.

of the RFC module. In the first scheme, the RFC is configured with the core function at the beginning of the multimedia application and this would result in a reduced cache capacity for the entire application. In the second scheme, the RFC is configured with the core function only when the core computation is necessary and the RFC module is released when not required to be used as a normal cache memory by the other computations within the multimedia application. With various cache mapping organizations, we study the overall impact on the performance of selected benchmarks, such as multimedia and DSP applications. The results prove that performance of each scheme varies depending on the structure of the application used.

The rest of the paper is organized as follows. In Section II we present an overview and the preliminary results obtained in building an ABC microprocessor. In Section III we discuss the main goal of this paper by proposing various RFC configuration schemes. Section IV presents a comprehensive discussion on the impact of various parameters, related to the microarchitecture as well as the application, on the overall execution time. In Section V, we discuss the results obtained from applying various RFC configuration schemes. Finally, Section VI concludes the discussion.

2 Performance of ABC Architecture

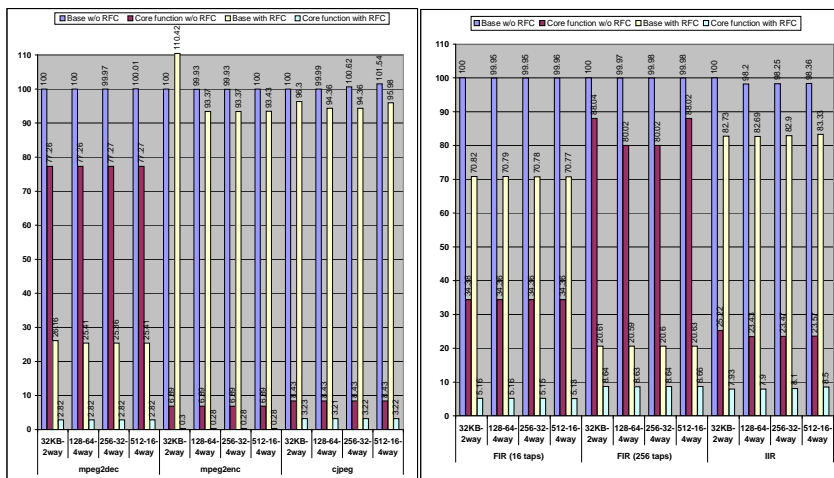


Fig. 1. Normalized execution cycles in base processor without RFC and ABC processor with RFC, with varying cache organizations.

The organization of a reconfigurable cache module (RFC) had been extensively discussed in [7]. To analyze the performance of the ABC architecture, the comparison is made between the number of cycles taken to execute each application with varying cache parameters. The performance, in terms of the total number of execution cycles normalized to the execution cycles in the base processor with 32KB 2-way set associative cache for each of the benchmarks, mpeg2dec, mpeg2enc, cjpeg, FIR-16taps, FIR-256taps, and IIR, is shown in Figure 1.

The performance improvement can be obtained from the normalized total cycles for the execution of the overall function and also the core function. The specialized computing units configured in the RFCs improve the performance of each core function significantly. The most important factors for speed-up are the reduced number of instructions and the acceleration of computation with a specialized unit. Hence, it is observed that the overall speed-up is largely proportional to the frequency of the core function calls in the entire application. However, the initial simulation results show that the RFC is not a good idea to implement in a low associative cache in certain cases where it dramatically reduces the cache capacity. For example, a performance degradation is observed in the execution of mpeg2encode application in the ABC processor with 32KB 2-way set associative cache. This is due to the fact that in a 2-way set associative cache, when one of the cache modules is used as an RFC, the cache memory capacity is reduced to 50% and also the data cache acts as direct-mapped and hence causes the performance degradation. This motivated us to look further into this issue in the design of the ABC architecture, and hence we set-out to explore various methodologies of the management of RFC between the functional unit and the cache memory modes.

3 RFC Configuration Schemes

In the multimedia applications, it has been observed that the applications like mpeg2decode, mpeg2encode, FIR and other filters are highly iterative and hence multiple instances of the core functions would be occurring during the processing of the entire application. For example, while running mpeg2decode applications it has been observed that the DCT/IDCT function is called 15 times more as compared to while running mpeg2encode application [9]. Further, it is noted that the core functions like DCT/IDCT occupy a varying degree of percentage of entire application.

The other factor that has a predominant effect on the utilization of the RFC for the execution of the core function would be the configuration overhead. It is the time, in number of cycles, taken to load the configuration data into the target hardware. The configuration overhead would be greater as more number of instances of RFC configuration occur. The configuration data to program a cache into a function unit may be either available in an on-chip cache or an off-chip memory. Load time for the configuration data in the latter case will be larger than in the former case. In the ABC architecture, the time to configure the RFC, from the normal cache memory mode to a functional unit mode, depends on the number of cycles to write words into a cache. On the other hand, an RFC operating as a functional unit can also be partially configured at run-time using write operations to the cache. When a partial reconfiguration occurs, the function unit must wait for the reconfiguration to complete before feeding the input data. Since the computation data (input and output data) and the configuration data (contents of LUTs) for an RFC unit share the global lines for data buses, we cannot perform both the computation and the partial reconfiguration simultaneously. It is possible to perform both the operations simultaneously, if separate data buses for computation and configuration are

provided, which would dramatically increase the cost of the microarchitecture design.

Thus it becomes imperative for us to study and devise various RFC management schemes for an efficient utilization of the computing and memory resources to achieve a better performance from the integration of the RFCs into a conventional microprocessor.

3.1 Scheme1: One-time configuration of RFC

In the current design [8] of the ABC architecture, the configuration of the core function into the RFC module is performed a priori, i.e., at the beginning of the application so that the configuration of the same core function at multiple instances can be avoided. However, due to this method, the computations other than the core function and those computed by the main processor, are denied the usage of all the cache modules.

This scheme would prove to be beneficial when the core function is dominant in its execution over the entire application as in the case of DCT/IDCT function in the mpeg2decode. However, the scheme has some limitations for adoption, as it can be observed that it will have a negative impact when the percentage of core function is significantly small over the entire application as in the case of DCT/IDCT function in mpeg2encode application. Also, the scheme would not be better for adoption, when the configuration overhead, the total number of cycles taken to configure the core function, would prove to be significantly smaller and can be offset by the maximum utilization of the cache memory resources.

3.2 Scheme2: Continual configuration of RFC

To overcome the limitations of the continuous denial of usage of full cache memory capacity for computations other than the core function, the RFC module can be configured with the core function at every instance of its occurrence and then be released to function as normal cache memory for the benefit of other computations. Thus the RFC module has to follow a pattern of continuous switching of modes, between the normal cache memory mode and the functional unit mode.

As it was originally anticipated and as argued in the case of the first scheme, the proposed continual reconfiguration scheme also has its gains and limitations. This scheme will have a negative impact when the percentage of core function execution time is significantly large in the entire application as in the case DCT/IDCT function in the mpeg2decode. Besides, the scheme would prove to be limiting when the number of instances of the occurrence of the core function is high as the configuration overhead would be significantly large.

4 Analysis of Execution Time

In this Section, we present our arguments to justify the need of an analytical study of the two RFC configuration schemes described above. To adopt the RFC based ABC architecture for processing various multimedia applications, one of the above schemes has to be incorporated in the design. In view of the gains and limitations of the proposed RFC configuration schemes, and keeping in view of the nature of the multimedia applications where the percentage of core function in the entire application and the number of instances of occurrence of core function vary, we develop a mathematical model for the analysis of the

performance in terms of total execution time, to determine which scheme proves to be beneficial under varying circumstances.

The architecture and application parameters, that will have significant impact on the overall execution time, have to be considered for the trade-off analysis among various schemes. The parameters that need to be considered are, the number of cycles required for configuration of RFC (C_p), the number of instances of occurrence of core function in the application (N), the fraction of time the core function is processed over the entire application (P), and the cache blocking factor (ϕ). Note that $(1-P)$ represents the fraction of time for the computations other than the core function. Also, since we design the architecture for the faster execution of applications by accelerating the core function, we assume that there always exists a portion of the core function that can be mapped into the RFC. Hence the assumption that $P \neq 0$, holds true for the entire discussion.

Table 1. Architecture Design Parameters.

C_p	Cycle time for RFC configuration when data is fetched from off-chip memory.
P	Fraction of core function over the entire application.
N	Number of instances of core function over the entire application.
ϕ	Cache blocking factor.
X_A	Execution time of core function in the base processor.
X_B	Execution time of core function in the RFC functional unit.
S	Speed-up of core function = $(\frac{X_A}{X_B})$

The total execution time (X_{org}) of an application in the base processor without RFC can be represented by the expression:

$$X_{org} = X_A \left(\frac{1-P}{P} \right) + X_A = \frac{X_A}{P} \quad (1)$$

Similarly, the total execution time (X_{rfc}) of the application in the RFC integrated ABC processor can be represented by:

$$X_{rfc} = X_A \left(\frac{1-P}{P} \right) \phi + C'_p n_c + X_B \quad (2)$$

The $X_A \left(\frac{1-P}{P} \right)$ accounts for the execution time of the computations other than the core function. When the continual RFC configuration scheme is employed, the cache blocking factor (ϕ) ≈ 1 , i.e., all the cache modules are available for utilization as cache memory during the execution of computations other than the core function. The ϕ is assumed to be approximately equal to 1 instead of being exactly equal to 1 in the continual configuration scheme, in the process of RFC being switched between various modes, the main processor will be experiencing cache misses, which might be hit under normal cache operation. When the one-time RFC configuration scheme is employed, the cache blocking factor (ϕ) > 1 , as

one of the cache modules is reserved for the functional unit and hence execution of the computations other than core function would take more time due to the rise in cache miss rate and the reduced cache capacity.

The $C'_p n_c$ accounts for the RFC configuration overhead over the entire application. When the one-time RFC configuration scheme is integrated into the design, the configuration overhead will be significantly smaller, as $n_c = 1$. Also, $C'_p = C_p$ since we assume that during the first time configuration of RFC, all the configuration data would be a miss in the cache and hence need to be fetched from the off-chip memory. However, when the continual RFC configuration scheme is employed, the configuration overhead forms a considerable portion of the total application execution time, as $n_c = N$. However the bright spot in this case is that the simulation has proved that $C'_p \ll C_p$, since the subsequent instances of configuration of RFC with the same function as the preceding instance configuration will not take as many cycles as the first instance of configuration. This is due to the fact that, between the adjacent calls of the core function when the other computations are being executed, a few data blocks in the RFC module get replaced and hence during the subsequent configuration, only the corrupted configuration data is fetched from the off-chip memory.

Table 2. Variation in parameters for various RFC configuration schemes.

	One-time configuration	Continual configuration
Cache blocking	$\phi > 1$	$\phi \approx 1$
Configuration instances	$n_c = 1$	$n_c = N$
Configuration cycles	$C'_p = C_p$	$C'_p \ll C_p$

The summary of the variation of parameters with respect to each of the schemes employed is as shown in Table 2. The parameters, ϕ in the case of one-time RFC configuration scheme, and C'_p in the case of continual RFC configuration scheme are the two factors that affect the total execution time predominantly. It is not possible to exactly quantify these two factors, as both ϕ and C'_p are entirely dependent on the amount of data cache accesses, the cache organization and the cache miss rate. However, fortunately we can define the upper bounds for each of the parameters with the goal of obtaining a better performance from the RFC based ABC microprocessor.

4.1 Execution time analysis with one-time RFC configuration

When the design of ABC architecture is incorporated with the one-time RFC re-configuration scheme, the expression for X_{rfc} will be modified, after substituting relevant parameters from Table 2, as given by the following expression:

$$X_{rfc} = X_A \left(\frac{1-P}{P} \right) \phi + C_p + X_B \quad (3)$$

We are aware that, in the worst case, the configuration time (C_p) can be in the order of thousands of cycles while, X_B would be in the order of millions of cycles and hence C_p can be conveniently neglected in the expression. Now, for a gain in the performance, the execution time of application in the RFC based

ABC processor should be less than the execution time of application in the base processor. Hence, solving $X_{rfc} < X_{org}$ yields

$$X_A \left(\frac{1-P}{P} \right) \phi + X_B < \frac{X_A}{P} \quad (4)$$

from which we can obtain the upper bound for the cache blocking factor as:

$$\phi < \left[\frac{1 - \left(\frac{P}{S} \right)}{1 - P} \right] \quad (5)$$

From the above expression, it is obvious that the lower bound on ϕ is 1, as $\frac{X_B}{X_A} \leq 1$ under all circumstances, i.e., the speed-up obtained for the core function by computing in the RFC would be at least 1.

The variation of the cache blocking factor with respect to the fraction of the core function (P) and the speed-up of the core function (S) is shown in Figure 2. It can be observed that, for applications with a larger fraction of the core function, the upper bound of the cache blocking factor is large which signifies that the cache blocking factor (ϕ) can be raised up to its upper bound without degradation in the overall performance when the one-time RFC configuration scheme is implemented. Similarly, when the percentage of the core function is small, the upper bound of ϕ is small and hence the value of ϕ needs to be maintained within that tight bound to avoid a degradation in the performance of the ABC processor. Besides, it can be observed that for a fixed portion of the core function, the upper bound of the ϕ remains almost constant irrespective of a large variation in the speed-up of the core function.

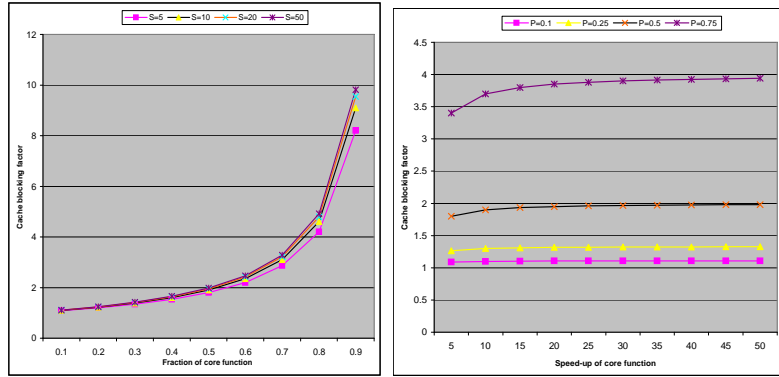


Fig. 2. Variation of cache blocking factor with (a) fraction of core function (b) speed-up in core function.

4.2 Execution time analysis with continual RFC configuration

When the design of ABC architecture is integrated with the continual RFC re-configuration scheme, the expression for X_{rfc} will be modified, again substituting the relevant parameters from Table 2, as shown:

$$X_{rfc} = X_A \left(\frac{1-P}{P} \right) + C'_p N + X_B \quad (6)$$

As argued in the earlier subsection, for the implementation of continual RFC configuration scheme, it is assumed that the cache blocking factor (ϕ) = 1 for practical purposes, though in the ideal scenario, $\phi \approx 1$. Now, for a gain in the performance, the execution time of application in the RFC based ABC processor should be less than the execution time of application in the base processor. Hence, solving $X_{rfc} < X_{org}$ yields

$$X_A \left(\frac{1-P}{P} \right) + C'_p N + X_B < \frac{X_A}{P} \quad (7)$$

from which the upper bound for the average configuration time (C'_p) can be obtained as:

$$C'_p < \left[\frac{X_A - X_B}{N} \right] \quad (8)$$

The upper bound on the average configuration time (C'_p) signifies that the total configuration overhead in the application should not exceed the difference in the execution time of the core function in the base processor and the RFC. This inference synchronizes with the generalized condition for the RFC based ABC processor to perform better than the base processor.

4.3 Effect of number of core function instances (N)

From the expression for X_{rfc} , it can be observed that, in general, the execution time $\propto N$, i.e., more the number of instances of the core function in the application, more will be the execution time due to the higher configuration overhead. However, for a considerably smaller number of N, the continual RFC configuration scheme would prove to be beneficial while the effect of a larger N can be offset by employing the one-time RFC configuration scheme. For example, the simulation study shows that the DCT/IDCT function has been called 129600 times in the mpeg2decode application, while it has been called 8,448 times in mpeg2encode application. Hence, while running the mpeg2decode application, the one-time RFC configuration need to be employed while the continual configuration scheme proves to be better while running the mpeg2encode application, as discussed in the results section.

4.4 Effect of percentage of core function (P)

From the expression for X_{rfc} , it can be observed that the execution time $\propto \left(\frac{1-P}{P} \right)$ and hence, more the percentage of the core function in the application, higher will be the speed-up of the overall application. This phenomenon is typical of the characteristic of Amdahl's Law, where the speed-up of the overall application is proportional to the portion of the application being accelerated. It can be deduced that when the portion of core function is smaller over the entire application, then it will not make a significant impact even when it is accelerated using an RFC. The simulation results, as shown in Section III, proves this principle. From Figure 1, it can be observed that the speed-up of around 4X is obtained in the execution of mpeg2decode application in RFC integrated ABC processor while the maximum speedup obtained in the execution of mpeg2encode application in RFC integrated ABC processor is only 1.07X and it can be seen that

the DCT/IDCT computation occupies a portion of 77.27% in the mpeg2decode application, while it occupies a portion of only 6.89% in the mpeg2encode application.

4.5 Effect of cache blocking factor (ϕ)

When all the cache modules are not available during the application execution time, size of the cache reduces along with reduction in the cache associativity, which causes the cache miss rate to increase. Subsequently, the execution time of the application increases. Hence, from the expression for X_{rfc} , it is obvious that the execution time $\propto \phi$. However, when the portion of the computations other than the core function in the overall application is significantly smaller, it will not be appropriate to employ the continual RFC configuration scheme, as the gain obtained due to the availability of full cache capacity for smaller execution time would be offset by the configuration overhead.

5 Results and Analysis

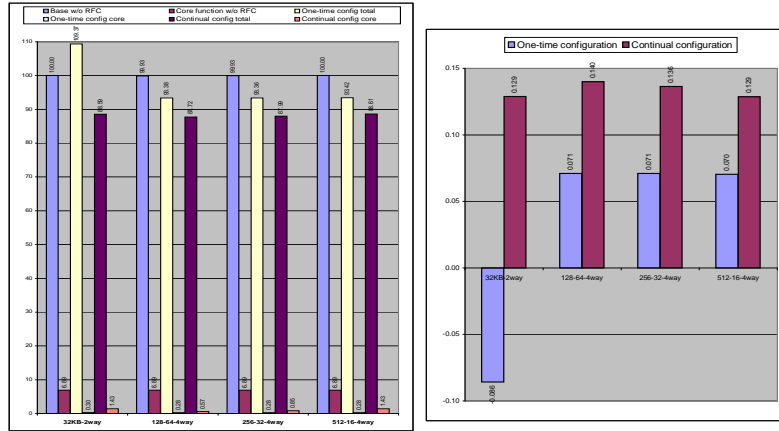


Fig. 3. (a) Normalized execution cycles in the base processor without RFC, and the ABC processor with different RFC configuration schemes (b) Relative performance improvement in two RFC configuration

From Figure3(a), it can be observed that the performance of ABC processor with continual RFC configuration scheme is better than the base processor without the RFC and also the ABC processor with one-time RFC configuration scheme, for mpeg2encode application. This is due to the fact that the percentage of the core function over the entire application is significantly smaller and also the number of instances of the core function is not big enough to generate a considerable configuration overhead. Note that the portion of core function indicated in the continual RFC configuration scheme includes both, the RFC configuration overhead and the computation time for the core function. Another interesting observation is that when the core function is called for configuration in RFC N times, and the configuration time is (C_p) cycles for full loading of

the RFC module with the configuration data from the memory, the total configuration overhead is found to be only 32% of the expected overhead $N*(C_p)$ cycles.

For the above reasons, the continual RFC configuration scheme performs better even in the 32KB 2-way cache while the one-time RFC configuration scheme results in a performance degradation, as shown in Figure 3(b). Also, it can be observed that there is an improvement in the performance of the continual RFC configuration scheme, with the reduction in the number of blocks in the RFC cache module. As the number of blocks in a cache module increases, the configuration time for RFC module increases and subsequently configuration overhead is significant, thus having a negative impact on the overall performance.

6 Conclusions

Reconfigurable Functional Cache (RFC) accelerates the computations using a specialized computing unit with minimal modification and overhead in area/time domains in the cache and microarchitecture. In continuance of our effort to build an efficient ABC architecture with improved performance, various RFC configuration schemes have been studied. The impact of various architectural parameters and the factors governing the structure of an application over the execution time of an application has been extensively studied. With the help of the study undertaken, the design of ABC microprocessor can be incorporated with the dynamic decision capability, so that appropriate RFC configuration scheme is chosen dynamically for running a particular application.

References

1. J. R. Hauser and J. Wawrzynek, "Garp: a MIPS processor with a reconfigurable coprocessor", *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, April 1997, pp. 12-21.
2. Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee, "CHIMAERA: a high-performance architecture with a tightly-coupled reconfigurable functional unit", *Proc. 27th International Symposium on Computer Architecture*, 2000, pp. 225-235.
3. A. DeHon, "DPGA-coupled microprocessors: commodity ICs for the early 21st Century", *Proc. IEEE Workshop on FPGAs for Custom Computing Machines*, 1994, pp. 31-39.
4. R. Razdan and M. D. Smith, "A High-Performance Microarchitecture With Hardware-Programmable Functional Units", *Proc. 27th Annual International Symposium on Microarchitecture, MICRO-27*, 1994, pp. 172-180.
5. P. Ranganathan, S. Adve, N. P. Jouppi, "Reconfigurable caches and their application to media processing" *Proc. 27th International Symposium on Computer Architecture*, 2000, pp. 214-224.
6. Huesung Kim, A. K. Somani, and A. Tyagi, "A Reconfigurable Multi-function Computing Cache Architecture", *Proc. FPGA 2000*, February 2000, pp. 85-94.
7. Huesung Kim, A. K. Somani, and A. Tyagi, "A reconfigurable multifunction computing cache architecture", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume: 9, Issue: 4, August 2001, pp. 509-523.
8. Huesung Kim, "Towards Adaptive Balanced Computing (ABC) Using Reconfigurable Functional Caches (RFCs)", Ph. D. Dissertation, Dept. of Electrical and Computer Engineering, Iowa State University, July 2001.
9. Chunho Lee, M. Potkonjak and W. H. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems", *Proc. Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture*, 1997, pp. 330-335.
10. Texas Instruments, "TMS320C6000 benchmarks", 2000, available on <http://www.ti.com/sc/docs/products/dsp/c6000/62bench.htm>
11. Doug Burger and Todd M. Austin, "The SimpleScalar Tool Set, Version 2.0", Computer Sciences Department Technical report # 1342, University of Wisconsin-Madison, June 1997.