Standardizing the Performance Assessment of Reconfigurable Processor Architectures

Lesley Shannon and Paul Chow
Department of Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada, M5S 3G4
{lesley, pc} @eecg.toronto.edu

Abstract

The field of reconfigurable computing is relatively new and unestablished, lacking some of the infrastructure that exists for parallel and general purpose processors. If new reconfigurable computing architectures are to be allotted the same recognition as the established fields of computing, it is necessary to improve the rigour with which new architectures are tested. Presently there is no defined standard, making baseline comparisons between different architectures difficult, if not impossible. Furthermore, designers are unable to measure the progress made in each new architecture.

This paper proposes a new abstraction for benchmark specification that permits a measure of reconfigurable architecture performance relative to a "standardized" algorithm. The objective is to enable designers of different architectures to program their processors with different languages yet still have a method of comparing their architecture's performance to that of a different research group.

1. Introduction

Comparing and evaluating the performance of reconfigurable processors is a difficult task. For this research field to progress in a more meaningful and scientific fashion, there needs to be a method of measuring progress between different reconfigurable architectures as well as with respect to traditional computing technologies.

This paper proposes the Reconfigurable Architecture TEsting Suite, or RATES, which defines a standard for describing and using benchmarks for reconfigurable architectures. Although creating a standard set of benchmarks seems to be an obvious solution, there are many issues to be addressed if this is to be realized. The first is that there is no *standard* input language for reconfigurable processors. While general purpose processors have numerous compilers designed to generate executables from common higher level languages such as C and Java, reconfigurable processors do not have a common language for algorithmic abstraction.

Many of these architectures are not suitable targets for typical compilers as they use hardware models. Today's compiler technology is inadequate for effectively compiling Higher Level Languages (HLLs) for a hardware fabric. More specifically, the technology does not exist to enable a compiler to retarget a common source code description to different types of reconfigurable platforms. This is partly caused by the fact that the designers have chosen different computational models for their processors, and thus adopted whatever language model enables a reasonable mapping to their architecture. RATES addresses this problem by abstracting the benchmark problem definition from the source code.

Processors such as Garp [1, 2], Chimaera [3, 4], NAPA [5], and OneChip [6, 7] model a general purpose processor with a reconfigurable function unit (RFU). They all support the description of an algorithm in a higher level language (HLL) such as C. However, Garp and Chimaera can compile C source code directly to the RFU where as NAPA and OneChip require the user to annotate the code for RFU operations. The Reconfigurable Communications Processor (RCP) from Chameleon Systems [8] has a similar architecture to those described above but uses a C compiler for the general purpose processor portions of the application and a Hardware Description Language (HDL) compiler for the RFU sections.

DISC [9], RaPiD [10, 11], and PipeRench [12] are reconfigurable fabrics that must be programmed with their own specialized languages. These processors allow the parallelization of data intensive applications and may be coupled with a RISC processor to increase their application space.

There is also the need to consider that the actual architecture of the processor is modifiable. Moreover, many of the newer reconfigurable processor architectures, such as the ones mentioned above, are at least partially dynamically reconfigurable. Benchmark suites are generally compiled to run on a specified unchanging architecture. While the flexibility of reconfigurable processors provides the user with greater power, it is exceedingly difficult to abstract this flexibility into a higher level language.

A second problem arises from the fact that there aren't any standard benchmarks for these processors. The problem of mapping an algorithm onto hardware is not trivial and remains the focus of a significant area of research. Generally, it is desirable to create the best mapping of an algorithm to the hardware to ameliorate performance. This may lead researchers to choose different algorithms when trying to implement the same benchmark. However, if a researcher does not use the same algorithm to implement the same benchmark there is some question as to the usefulness of the results for interproject comparisons.

Another consideration when choosing benchmarks are the input and output data sets. The size and content of the data sets often affect the execution time of the program as well as the segments of a program that are exercised. As there is presently no standardized suite of benchmarks, researchers who have chosen the same applications to verify their architectures have not necessarily used the same data sets. By structuring the benchmarks in RATES to have defined input and output sets, as well as a defined algorithm, the necessary commonality is created for enabling comparisons among different architectures.

The final problem involves the consideration of the types of tests that should be included in a reconfigurable benchmark suite. Most projects use numerous kernels to test their reconfigurable fabric and a few applications to measure the overall performance of the processor. When an architecture is first being developed, it is important to have kernels that can be used to verify the architecture. Furthermore, the repetitive throughput can be used to determine if the architecture has any inherent weaknesses early on in the design process. However, kernels do not provide a complete measure of a processor's performance. As the gate density of reconfigurable fabrics increases, it is important to choose applications that are more representative of the capabilities of the processor to evaluate how they implement the complete solution.

Applications are a combination of control and data manipulation and how a reconfigurable processor handles control intensive aspects of a program will significantly affect its overall performance. As is the case with all new computer architectures, the desire is to improve the overall application performance. Therefore, it is necessary to show overall speedup of complete applications. To do this, benchmarks should expose weaknesses in the overall design, such as overheads in the host interfaces, memory, and configuration. Ideally, RATES will contain a balance of kernels and applications representing both control and data intensive programs. However, input from the research community is essential to ensure that the suite addresses the important design issues.

As can be seen from the issues outlined in the above discussion, defining a standard benchmark suite for reconfigurable architectures is not a trivial task, so a methodology and set of specifications for describing benchmarks has also been proposed. The objective is for RATES to be a suite of benchmarks that is usable by any processor designer, independent of the underlying architecture. By standardizing the benchmarks, the comparisons among different architectures can be facilitated. This ensures a more scientific methodology for measuring performance and highlights the benefits and challenges of reconfigurable architecture design.

The remainder of this paper is structured as follows: Section 2 provides a summary of the previous benchmark work done for reconfigurable architectures. Section 3 describes the typical benchmark structure presently used and proposes a new structure better suited to reconfigurable processors. Section 4 presents the benchmarks that have been chosen for the suite thus far along with examples of the format used to specify them. In Section 5, a discussion of how these benchmarks were implemented on the OneChip simulator is presented. A summary of the benefits of this new scheme along with suggestions as to how the suite may evolve to better suit the reconfigurable community is given in Section 6.

2. Background

The main focus of this section is to discuss what benchmarks have been previously used to measure the performance of reconfigurable architectures. First, the history of the well known SPEC [13] benchmarks is summarized. The benchmarking framework for Network Processors presented by the Network Processing Forum (NPF) [14] and the Adaptive Computing Systems Benchmarking [15] from Honeywell are also discussed to provide a point of comparison.

SPEC was established in 1988 to develop and maintain a standardized set of benchmarks that are applicable to the "newest generation of high-performance computers" [13]. SPEC began providing source code for benchmark applications, which ensured that the same task was performed on all platforms using the same algorithm. By providing source code for real applications, SPEC made it easier for designers to obtain a better measure of their architecture's performance. Furthermore, the benchmarks are continually updated to reflect the ever increasing computational power of processors. The benchmarks are designed to include applications that are representative of those currently running on processors.

The present SPEC benchmarks are not executable on all reconfigurable processors because not all designers have created C compilers for their processors. Therefore, it is not possible to compare the execution of the same task on both general and reconfigurable processors using SPEC.

The Network Processing Forum Benchmarking Work Group [16] wrote a white paper [17] presenting a new format for network processor benchmarks. The challenge of describing a set of benchmarks for network processors arises from the fact that, by definition, they include a variety of different architectures and physical implementations. This is the primary cause of the problems presently faced by the reconfigurable community. To address this problem, the Network Processing Forum Benchmarking Work Group [14] has proposed that the benchmark descriptions not be specified as source code in a high-level language. Instead, they have chosen to use a functional description to identify "test configurations, variables, and metrics relevant to benchmark suites" [17]. This allows users to evaluate the processor based on its performance with respect to inputs and outputs, enabling designers to have greater freedom in their design choices. Furthermore, they have chosen to specify benchmarks at three different levels of granularity: *system level*, *application level*, and *task level*.

S. Kumar et al. [21] at Honeywell Technology Center developed a set of stressmarks to test specific properties as opposed to functional benchmarks, which focus on measuring the required execution time of an application. The characteristics of interest include versatility, capacity, interface, scalability, timing sensitivity and CAD performance. They also chose a variety of applications that are of interest to the reconfigurable community including the Fast Fourier Transform (FFT), image compression, and boolean satisfiability (SAT). This is believed to be the first work attempting to address the issue of the need for benchmarks for reconfigurable processors, however, they are only available to a closed community[1]. This means that while everyone could choose to use these applications as the standards for a suite of benchmarks, there is no guarantee that everyone would choose the same algorithms for implementation.

Over the years, designers have chosen numerous applications and kernels to verify the functionality and measure the performance of their architectures. Table 1 contains a summary of the more common applications chosen, the benchmark suites to which they have been attributed, and the number of different projects that used them. Similarly, Table 2 summarizes the kernels used. For the purpose of this discussion, kernels are defined to be basic tasks, whereas applications address complex problems in multiple steps. Ideally, applications should be created from multiple different kernels available in the benchmark suite as this would help to reduce design time.

Application Name	Benchmark Suite	Number of Projects
ADPCM	Mediabench	3 [4, 18, 7]
JPEG	Mediabench, Unknown	3 [18, 7, 12]
MPEG2	Mediabench	3 [4, 18, 7]
PEGWIT	Mediabench	3 [4, 18, 7]
Compress	SpecInt92, SpecInt95	2 [19, 3, 18]
Eqntott	SpecInt92	2 [19, 3]
G721	Mediabench	2 [4, 18]
li	SpecInt92, SpecInt95	2 [19, 18]
Conway's Game of Life	——	1 [3]
gcc	SpecInt92	1 [19]
Comp	Honeywell	1 [4]
Decomp	Honeywell	1 [4]
DNA and Protein Sequence Analysis	——	1 [20]
EPIC	Mediabench	1 [18]
espresso	SpecInt92	1 [19]
go	SpecInt95	1 [18]
GSM	Mediabench	1 [18]
ijpeg	SpecInt95	1 [18]
PEGWIT key	Mediabench	1 [4]
perl	SpecInt95	1 [18]
Edge Detector	——	1 [20]
sc	SpecInt92	1 [19]
vortex	SpecInt95	1 [18]

Table 1. Summary of Applications Commonly used on Reconfigurable Architectures.

Most designers chose applications from the University of California's MediaBench suite [23]. Table 1 lists ADPCM, JPEG, MPEG2, PEGWIT, G721, and EPIC, treating the combination of the encoder and decoder as a single application. While most designers ran both the encoder and decoder for these applications, only the MPEG2 encoder was run on the Chimaera [4] processor. The other exception not clearly specified in the table is the source of the JPEG algorithm used by the PipeRench [12] designers, denoted by the "Unknown" entry in the Benchmark Suite column. The rest of the applications were generally selected from either the SpecInt92 and SpecInt95 benchmark suites or the Honeywell suite [15], though a few were individual applications. In contrast, the choice of kernels used to measure architecture performance is apparently quite arbitrary, as only one of them is from an actual benchmark suite.

As can be seen from these two tables, most of the bench-

[1] We were refused permission to access the DARPA website.

Kernel Name	Benchmark Suite	Number of Projects
FIR	——	3 [11, 12, 22]
DCT-2D	——	2 [11, 12]
ATR		1 [12]
2D Convolution	——	1 [11]
Convolutional Encoder	——	1 [22]
Cordic	Honeywell	1 [12]
Data Encyption Standard (DES)	——	1 [1]
DCT	——	1 [12]
Dither	——	1 [1]
IDEA	——	1 [12]
IIR	——	1 [22]
Image Mean Filter	——	1 [9]
FFT	——	1 [22]
FSM	——	1 [22]
Full Motion Estimation	——	1 [11]
keyword and dictionary search	——	1 [20]
Matrix Multiply	——	1 [11]
Nqueens	——	1 [12]
Over	——	1 [12]
PGP	——	1 [12]
PopCount	——	1 [12]
Sort	——	1 [1]
Vector Add	——	1 [22]
Vector Dot Product	——	1 [22]
Vector Maximum	——	1 [22]
Walsh-Hadamard	——	1 [5]

Table 2. Summary of Kernels Commonly used on Reconfigurable Architectures.

marks are only used by one project. However, the benchmarks chosen for evaluating the processor's performance are generally data intensive. Therefore, while the choices are numerous, it should be possible to choose a select group of kernels and applications to represent the different problem areas that designers have been targeting with reconfigurable architectures.

3. Benchmark Structures

This section opens with a discussion of the typical structure used for testing a general purpose processor architecture and the inadequacy of this model for testing a reconfigurable processor. It follows with a proposal for a new structure for testing these architectures, explaining how this new format will help to address the problems discussed in

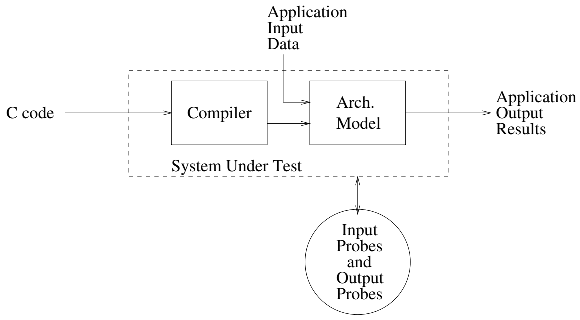

Figure 1. The traditional format for testing processor architectures that use a C code compiler.

previous sections. Conceptually, this suite is similar to the Honeywell and NPF suites.

NPF has so far only specified a format for the hardware information that must be provided along with the benchmark results and selected potential applications for the suite. They have not outlined a standardized format for describing applications, whereas this proposal includes a set of *rules* that must be followed when specifying a benchmark to be included in RATES.

Honeywell's suite also uses a benchmark specification format independent of language implementation similar to RATES. However, the emphasis of their *stressmarks* is to evaluate specific characteristics of an architecture and its corresponding infrastructure. The Honeywell benchmarks do not include a method for comparing processor performance equivalent to the manner that SPEC is used to compare general-purpose processors. From this perspective, the goals of the Honeywell suite and RATES are complementary.

RATES is a set of functional benchmarks, is totally independent from the architecture and language, and usable on any processing platform be it general purpose or reconfigurable. It requires standard algorithms to allow comparisons amongst architectures but allows custom algorithms to highlight specific features.

3.1. Typical Structure

Traditionally, processor performance is measured by compiling C source code for a set of accepted benchmarks to the new architecture. The executable is then run on the architecture and different measurements are taken to determine the overall performance of the architecture. This is illustrated in Figure 1. The dotted box defines the system being tested. As shown in the figure, the test includes both the compiler and the hardware model.

Note that the terms *application input data* and *application output results* are used to delineate the inputs and outputs for the actual application being executed. In contrast, the *probe inputs* and *probe outputs* are used to measure the performance and resource usage of the processor.

Also, it is important to realize that both the application input data and source code are provided. Furthermore, the application output results are defined to guarantee that there are no errors in the compilation process or the processor's execution. This means that all performance measures are dependent on how the code is compiled for the architecture and how the processor executes this low-level machine code.

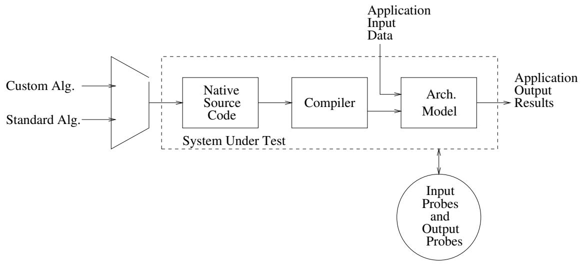

Figure 2. A new format for testing reconfigurable architectures.

3.2. Proposed Structure

To accommodate the lack of source code standards for reconfigurable processors, we must move to a higher level of abstraction. The first step is to describe an application in terms of an algorithmic representation, instead of just providing the source code for a benchmark. This form of description is necessary as it is the only level of "input" that can be common to all architectures.

It is important that a *standard* algorithm be provided for each application so that there is a baseline for comparison between implementations of the same algorithm on different reconfigurable architectures. Some reconfigurable architectures may have special features that are well suited to improving the performance of select benchmarks. In this case, it should be permitted to generate results using a *custom* algorithm tailored to that architecture. These results can then be compared with those of the standard algorithm. This illustrates the improvement that the custom algorithm provides over the standard algorithm implemented on this and various other processors. It is essential to note that the results for the standard algorithm are required to relate the results between custom and standard algorithms among all architectures measured using this methodology.

The second step is to develop specified input and output data sets for each application. As can be seen from Figure 1, a complete benchmark environment is specified by the source code, the application's input and output data sets, and the probe values that will be measured. Respecting these requirements, the new testing flow can be pictured as illustrated in Figure 2.

As can be seen from the figure, the designer is provided with an algorithm from which the native source code must be created. This source code will be recognized by the compiler, which is designed specifically for this system, and may be either an HLL or an HDL depending on the computational model chosen for the processor.

From the new structure outlined in Figure 2, it is apparent that the system being measured can accept one of two algorithm types. The first is the *Standard Algorithm* that is specified as part of the benchmark suite. The second is what is being termed as the *Custom Algorithm* that allows the use of specific architectural features as mentioned above.

For example, if one of the kernels in the suite is a Sort algorithm, it will be stated that the designer should use a specific Sort algorithm, such as quicksort. However, due to the nature of a reconfigurable architecture, a mergesort algorithm may result in a better performance than quicksort, or even some combination of these algorithms as was done by Garp [1]. To illustrate how the overall performance is improved due to a specific feature inherent to the architecture, the designer should implement the specified quicksort algorithm as well as their custom algorithm. Ultimately, these different algorithms implement the same function so the input and output data sets should remain unchanged.

Furthermore, it is important to note that the measures of the system's performance not only incorporate the compiler and the processor architecture, but also the native source code. This means that it is the designer's responsibility to create code that will implement the benchmark such that it has good executional behaviour.

Finally, to effectively assess the performance of the processor, it is important that complete applications are used to test the system. If the processor is tested using only kernels, certain portions of the architecture may not be exercised.

3.3. Guidelines for Benchmark Specification

Having presented a new format for benchmarks used in reconfigurable computing, it is important to give some guidelines. These define the necessary characteristics for specifying benchmarks that provide meaningful measures of system performance. It should also be noted that these rules are an initial attempt to structure the benchmark format and it may be necessary to append, amend, or expel specific rules as the the reconfigurable processing field matures. We propose the following rules as the initial set of guidelines, but expect that some forum for discussion will be necessary to fairly represent the needs of the reconfigurable community.

To ensure that the following discussion uses consistent

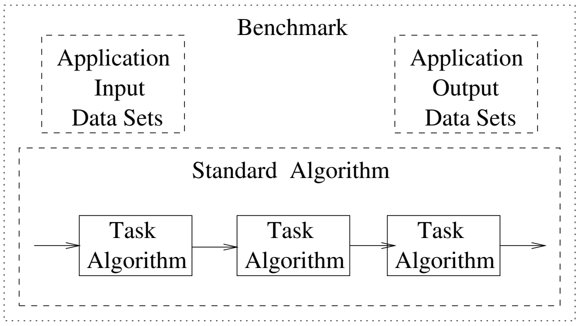

Figure 3. A pictorial hierarchy illustrating the suite's taxonomy.

terminology, the pertinent definitions are now given. Each benchmark contains application input and output data sets as well as a Standard Algorithm describing what functionality is to be implemented. This is the Standard Algorithm for the benchmark that may further be broken down into the individual tasks that must be performed. Each task is then described in terms of a Task Algorithm. This hierarchy is illustrated pictorially in Figure 3.

Rule 1: A sequential algorithm must be given as the Standard Algorithm for each benchmark, and C source code provided as an implementation example. Source code for the reconfigurable processor must be generated based on the Standard Algorithm and not the example source code. The designer's source code implements the same Standard Algorithm demonstrated in the example source code, however, it can be parallelized and pipelined so as to obtain the best possible performance on their architecture. The example code is provided as a reference point so that the actual functionality of the benchmark can be verified. A detailed description of the Standard Algorithm specification format is given in Section 4.1.

Rule 2: The specification of the Standard Algorithm must be at a functional level where the inputs and outputs to a task are detailed but the internal implementation is a black box. The objective is to describe to the user the basic components, or tasks, that should be used to implement the algorithm as illustrated in Figure 3. The tasks are described at a higher level to allow for flexibility of implementation, but a strict definition of the control flow between tasks is required to ensure that designers are executing the same Standard Algorithm.

This is sufficient information for a consistent algorithmic description. If the designer feels that the Standard Algorithm is too restrictive or does not fully utilize the processor's capabilities, the algorithm can be customized and the improvement in performance measured with respect to the standard.

Rule 3: The Standard Algorithm must always be implemented. Any modifications to this will be considered a separate Custom Algorithm. To make a reasonable comparison of performance between architectures, a common baseline is necessary. Not only must the problem be the same, but also the methodology used for its solution. When multiple different algorithms exist for solving the same problem, only one is chosen as the Standard Algorithm. Implementations of other algorithms are considered to be Custom Algorithms.

Rule 4: The input and output operands of a task must be specified in terms of type and bit width. To further restrict the interface between tasks, all input and output variable fields must be strictly specified, but the designer may choose the storage structures used to implement these variables. This ensures that everyone is working at the same minimal level of computational precision and that the application output results are consistent among processors, while not restricting the actual implementation. It should also be noted that internal temporary variables remain unspecified so as to allow even greater flexibility when mapping each task of execution to the processor.

Rule 5: Application Input and Output data sets must be supplied for reproducibility. All architectures should measure performance based on the same set of input data. This is also needed to provide a common basis for comparison. The output data sets allows for checking the correctness of the implementation.

Obviously, there is no expectation that the numerical values will be exactly the same for computationally intense applications, but the precision of the result is a measure of the correctness of the outputs. Therefore, the designer should stipulate the precision of the outputs along with the execution time. For example, if the output results from processor A are obtained in time t with 6 bits of precision whereas the results from processor B take time $2t$ to be generated but have 14 bits of precision, processor B could be considered a better architecture depending on the requirements.

Rule 6: The performance of the architecture as the size of the problem and/or the size of the reconfigurable fabric scales should be discussed. The intent of this rule is to provide some understanding of the behaviour of the architecture when the problem does not "fit" entirely on the fabric. While it is not a required measure, the results would significantly enhance the understanding of a processor's behaviour in these circumstances. All results must still show the performance of the Standard Algorithm.

Rule 7: All architectures must be evaluated with the same metrics. The process for obtaining a metric must also be clearly defined. Current metrics include:

1. *Execution Time.* Execution time and the output databit precision are required values for each benchmark. The definition of this metric is dependent on the type of benchmark. For streaming data applications, the ex-

ecution time may be measured as the throughput time for the data being processed. For applications with a definitive start and end point, the execution time is defined to be the time elapsed from when the program begins its load onto the processor until the final results are obtained.

2. *Configuration Time.* Another essential metric for a reconfigurable processor is the configuration time, which is to be reported as two components. The first is the time required for initialization of the processor fabric and the second is the overhead incurred when processor execution stalls so that the fabric can be reconfigured. For reconfigurable architectures, configuration time is an important consideration as the overhead can be quite significant if not done in parallel with program execution.

3. *Power Dissipation.* Power is a significant concern in reconfigurable architectures as there may be significant dynamic power dissipated when the processor is reconfigured. However, measuring power dissipation is a difficult exercise, so this metric is being discussed assuming that some researchers will focus on power issues in their architectures. It is not assumed that every designer will attempt to quantify the power dissipation of their architecture. However, for the measure to be meaningful when reported, any power dissipated during the initial configuring of the processor should also be included in the measurement.

4. *Area.* The silicon area required to implement the processor may be another potential metric for use in some research. A primary consideration is that the size of a reconfigurable fabric will greatly affect the ability to parallelize and pipeline application execution. This metric should be reported as an estimate of the die area required to implement the architecture in a technology specified by the designer.

Rule 8: A well written implementation of the Standard Algorithm for execution on general purpose processors must be provided. It must be used for all performance comparisons between the reconfigurable architectures being studied and general purpose processors. Use of the Standard Algorithm again provides a common basis for comparison between the implementation on the reconfigurable processor and the general purpose processor. Requiring all performance measurements on general purpose processors to use the same code for the Standard Algorithm also provides a means for calibration between projects as it is unlikely that different projects will measure performance using the exact same general purpose processor configurations.

3.4. Summary

The above rules are an attempt to codify the methodology that should be followed when designing and using RATES to measure the performance of a reconfigurable architecture. The new benchmark format compensates for many of the challenges faced by designers. However, these guidelines are open for modification based on input from the research community. Furthermore, it is assumed that it will be necessary to update these rules as further technological advances are made. Until then, it is important that some form of standardized testing be accepted by the community to introduce more rigour to the design process.

4. Current Benchmarks

This section describes the benchmarks that have been initially selected for RATES and why they were chosen. However, before describing these choices, a presentation of the format for specifying the Standard Algorithm of a benchmark is given.

4.1. Benchmark Suite Format

Although the suite will eventually contain numerous kernels and applications, it seemed important that some discussion should be involved in choosing the applications so that they fairly represent the interests of the community. Furthermore, the choice of kernels should be based on the modules of functionality in each application to enable code reuse. The final consideration will be to ensure that both control and data intensive benchmarks are fairly represented.

Having chosen an application or kernel for inclusion in RATES, the remainder of the benchmark specification consists of:

- the Standard Algorithm Specification
- the Application Input and Output Data Files
- the C code Algorithm Description
- the C code efficient implementation.
- the README file

The README file will provide the user with a description of how to execute the C examples, the input and output file formats, and any other information specific to this benchmark. This would include any parameters not strictly specified in the Standard Algorithm Description. The C code Algorithm Description file is simply an implementation modelling the Standard Algorithm Description. However, this code may have very poor run time performance. In these

2D-DCT: Kernel
General Description: The Discrete Cosine Transform (DCT) takes an input signal and outputs its spectral content.
Data Types (0)
Execution Metric: Throughput (DCTs/sec)
Inputs (1)
time_d: array of 64 signed values: 16 bits
Outputs (1)
freq_d: array of 64 signed values: 16 bits
Task Descriptions:
Task 1: 2D-DCT
Inputs (1)
time_d: array of 64 signed values: 16 bits
Outputs (1)
freq_d: array of 64 signed values: 16 bits
Algorithm: The mathematical expression for a 2D-DCT is:

$$Y(u,v) = \frac{1}{4} * k(u) * k(v) * \sum_{m=0}^{7} \sum_{n=0}^{7} [x(m,n) * \cos[\frac{u\pi}{8} * (m + \frac{1}{2})] * \cos[\frac{v\pi}{8} * (n + \frac{1}{2})]],$$

where:

$$k(j) = \frac{1}{\sqrt{2}}, j \neq 0 \quad \text{else} \, k(0) = 1$$

Flowchart:
N/A (only one task)

Figure 4. The DCT benchmark specification.

situations, more efficient C code must also be supplied so that comparisons between the new reconfigurable architecture and a general purpose processor architecture are unbiased. The most important component of the benchmark specification is, however, the Standard Algorithm specification described and illustrated in the following sections.

A description of the different fields in the general Standard Algorithm specification format is now given using the 2-dimensional Discrete Cosine Transform (2D-DCT) in Figure 4 as an example. First, the General Description field is used to provide an overview of the Standard Algorithm for the benchmark. Data Types may be used for creating structures to simplify the specification but they need not be used in the actual implementation of the algorithm. When a data type is declared, each field name and size must be specified. Since an Execution Metric must be used with every benchmark, the type is specified as either *throughput* or *total time* for the designer along with the appropriate units of measure. Application Inputs and Outputs are described in terms of their name, structure and size.

Next, the Task Descriptions are given in a format similar to the headings previously described. The necessary inputs and outputs are specified for each task but others may be added as required. The Task Algorithm is then stated, detailing what objective is to be accomplished. Finally, having described all the individual tasks, a flowchart illustrating the control flow between tasks is provided.

For the general format for benchmark specification and further information on RATES, please see: *http://www.eecg.utoronto.ca/~lesley/benchmarks/rates/*. Benchmarks, input and output data sets, and sample C code can be downloaded from there. The following sections describe the initial benchmarks chosen for the suite and provide sample Standard Algorithm specifications of each one.

4.2. 2D-DCT

The 2D-DCT kernel was chosen due to its popular usage as a benchmark for previous projects as shown in Table 1. However, it is also a task for many data intensive media applications that are likely candidates for inclusion in RATES. The specification for the benchmark is given in Figure 4.

The Standard Algorithm has only one task that was specified using a mathematical equation. No data types were needed for this specification and since it needs only one task to complete the Standard Algorithm specification, a flowchart was unnecessary.

4.3. Conway's Game of Life

John Conway's Game of Life was chosen as an easily implementable control intensive application. This solitaire game is played on a large grid where each cell is initially said to be *alive* or *dead* according to a seed configuration. Based on a defined set of rules, each cell is born, dies, or survives in the next generation. Previously used by designers of the Chimaera processor [3], it offers an intensive decision kernel, ease of scalability, and different models for implementation of custom algorithms. Depending on the size of the grid and the reconfigurable fabric, it may be possible to implement a parallel design where the the entire grid is represented on-chip. The alternative is for each cell to be evolved separately, requiring a significant number of memory swaps. Figure 5 provides the benchmark specification.

Obviously, the application has many more tasks and a more complex control structure than that of the 2D-DCT kernel. Furthermore, the data type *life_t* was declared with fields *pres_state*, *next_state*, and *neigh_cnt* to simplify the description of *reset_grid* (Task 1). The values for M and N are unnecessary to the algorithm and would thus be provided in the README file. Finally, Tasks 3 through 5 are not included due to spatial constraints, but the flowchart illustrating the control flow through the tasks is still included.

4.4. Benchmark Specification Issues

The major issue encountered when specifying the above benchmarks was the difficulty of concisely expressing a control flow problem. While data intensive applications, such as the 2D-DCT, will have mathematical expressions that can compactly represent the required computations to

John Conway's Game of Life: Application

General Description: A solitaire game where the initial *grid_seed* is evolved *life_span* times based on a predefined set of rules for the birth, death, and survival of each cell.

Data Types (1)

life_t: pres_state: 1 bit, next_state: 1 bit, neigh_cnt: 4 bits

Execution Metric: Total Time (sec)

Inputs (2)

life_span: 32 bit value

grid_seed: M by N grid; 1 bit values

Outputs (1)

life_grid: M by N grid of life_t; 6 bit values

Task Descriptions:

Task 1: reset_grid

Inputs (1)

life_grid: M by N grid of life_t; 6 bit values

Outputs (1)

life_grid: M by N grid of life_t; 6 bit values

Algorithm: Resets all of the fields in the *life_grid* struct to zero.

Task 2: init_grid

Inputs (2)

pres_state: M by N grid; 1 bit values

grid_seed: M by N grid; 1 bit values

Outputs (1)

pres_state: M by N grid; 1 bit values

Algorithm: Initializes the *life_grid* struct's *pres_state* field with the *grid_seed* values.

.
.
.

Task 6: set_pres_to_next_state

Inputs (2)

pres_state: M by N grid; 1 bit values

next_state: M by N grid; 1 bit values

Outputs (1)

pres_state: M by N; 1 bit values

Algorithm: Sets the *pres_state* to the *next_state* and increments the generation count

Flowchart:

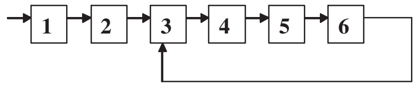

Figure 5. The Game of Life benchmark specification.

be performed, this is not true for control intensive applications. Although the Game of Life is a small application, it required numerous tasks to explain the values on which control decisions were made. Pragmatically, the specification is significantly shorter than the necessary C source code, but the challenge of compactly expressing control intensive applications is a concern.

5. Benchmark Implementation

Having introduced a set of benchmarks in the previous section, this section discusses the actual implementation of these benchmarks on a simulated reconfigurable architecture. The architecture of the Sim-OneChip simulator is based on the original OneChip described by Ralph Wittig [6], and consists of a RISC processor and a RFU. Jorge Carrillo [7] updated this architecture and developed the simulator using the SimpleScalar [24] simulator as its basis. Although there is no specific definition of the RFU fabric, the important test was to determine if it were possible to create these benchmarks based on the rules outlined in Section 3.3 and the benchmark specifications.

Both benchmarks were successfully implemented on Sim-OneChip based on the specifications in Figures 4 and 5. The final benchmark specification methodology is the result of an iterative process, where the Standard Algorithm was initially specified and then an attempt was made to write the source code based on the specification. Its format was then altered to reflect any necessary lacking information or restrictive requirements, and another attempt was made to code the algorithm. This process continued until the RATES guidelines outlined in Section 3.3 and the format illustrated in Figures 4 and 5 were formulated.

6. Conclusions

RATES has been proposed as a benchmark suite standard for reconfigurable architectures. A methodology and format for specifying benchmarks has been presented that enables designers of any type of processor architecture to use these benchmarks. This allows designers to be more rigourous in measuring their architecture's performance and to achieve greater scientific meaning when comparing their design to pre-existing ones.

The next phase of this project is to assess the importance of a format for specifying the performance results of a processor. This would likely include a summary of the processor hardware as is suggested by the NPF's benchmarking proposal [17]. There is also the need to have an open dialog to choose the remainder of the benchmarks for RATES. These will have to be described in the specified format and made publicly available. Finally, there will be a continuous need to evaluate the rules and specification format for the suite so that it can be updated when necessary.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council. The authors would like to thank Jason Anderson, Navid Azizi, and Tomasz Czajkowski for their many helpful comments and suggestions.

References

[1] J. Hauser and J. Wawrzynek. Garp: A MIPS processor with a reconfigurable coprocessor. In *Proceedings of the IEEE Symposium on Field–Programmable Custom Computing Machines*, pages 24–33, April 1997.

[2] T. J. Callahan, J. R. Hauser, and J. Wawrzynek. The Garp architecture and C compiler. In *Computer*, pages 62–69, April 2000.

[3] S. Hauck, T. W. Fry, M. Hosier, and J. P. Kao. The Chimaera reconfigurable functional unit. In *Proceedings of the IEEE Symposium on Field–Programmable Custom Computing Machines*, pages 87–96, April 1997.

[4] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee. CHIMAERA: A high–performance architecture with a tightly–coupled reconfigurable functional unit. In *Proceedings of the 27th International Symposium on Computer Architecture*, pages 225–235, June 2000.

[5] C. R. Rupp, M. Landguth, T. Garverick, E. Gomersall, H. Holt, J. A. Arnold, and M. Gokhale. The NAPA adaptive processing architecture. In *Proceedings of the IEEE Symposium on Field–Programmable Custom Computing Machines*, pages 28–37, April 1998.

[6] R. D. Wittig and P. Chow. OneChip: An FPGA processor with reconfigurable logic. In *Proceedings of the IEEE Symposium on Field–Programmable Custom Computing Machines*, pages 126–135, March 1996.

[7] J. E. Carrillo E. and P. Chow. The effect of reconfigurable units in superscalar processors. In *Proceedings of the Ninth ACM International Symposium on Field Programmable Gate Arrays*, pages 141–150, Monterey, CA, February 2001.

[8] Chameleon Systems Inc. What we do, 1997. http://www.chameleonsystems.com/what/whatWeDo.html.

[9] M. J. Wirthlin and B. L. Hutchings. A dynamic instruction set computer. In *Proceedings of the IEEE Symposium on Field–Programmable Custom Computing Machines*, pages 99–107, Napa, CA, April 1995.

[10] C. Ebeling, D. C. Cronquist, and P. Franklin. RaPiD: Reconfigurable pipelined datapath. In *The 6th International Workshop on Field-Programmable Logic and Applications*, pages 126–135, September 1996.

[11] D. Cronquist, P. Franklin, C. Fisher, M. Figueroa, and C. Ebeling. Architecture design of reconfigurable pipelined datapaths. In *Proceedings of the 20th Anniversary Conference on Advanced Research in VLSI*, pages 23–40, March 1999.

[12] S. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, and R. Laufer. PipeRench: A coprocessor for streaming multimedia acceleration. In *Proceedings of the 26th International Symposium on Computer Architecture*, pages 28–39, May 1999.

[13] Jeff Reilly. Spec discusses the history & reasoning behind spec95, September 1995. http://www.specbench.org/osg/cpu95/news/cpu95reason.html.

[14] Network Processing Forum. Home page: http://www.npforum.org.

[15] Honeywell Techonology Center. Adaptive computing system benchmarking, 1998. http://www.htc.honeywell.com/projects/acsbench/.

[16] Network Processing Forum Benchmarking Work Group. Home page: http://www.npforum.org/benchmarking/index.shtml.

[17] Prashant R. Chandra and Seow Yin Lim. Framework for benchmarking network processors, September 2000. http://www.npforum.org/techinfo/NPFFramework.pdf.

[18] M. Budiu. Application-specific hardware: Computing without CPUs. In *Fourth CMU Symposium on Computer Systems*, http://citeseer.nj.nec.com/497138.html, October 2001.

[19] R. Razdan and M. Smith. A high-performance microarchitecture with hardware–programmable functional units. In *Proceedings of the 27th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 172–180, November 1994.

[20] J. M. Arnold. The Splash 2 software environment. In D. A. Buell and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 88–93, Napa, CA, April 1993.

[21] S. Kumar, L. Pires, S. Ponnuswamy, C. Nanavati, J. Golusky, M. Vojta, S. Wadi, D. Pandalai, and H. Spaanenberg. A benchmark suite for evaluating configurable computing systems– status, reflections, and future directions. In *Proceedings of the 2000 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'00)*, pages 126–134, February 2000.

[22] P. Graham and B. Nelson. Reconfigurable processors for high-performance, embedded digital signal processing. In *Proceedings of the Ninth International Workshop on Field Programmable Logic and Applications*, pages 1–10, Villach, Austria, August 1999.

[23] Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communication systems. In *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture*, December 1997.

[24] D. Burger and T. M. Austin. The simplescalar tool set, version 2.0. *Technical Report 1342*, 1997. http://www.simplescalar.com/docs/users_guide_v2.pdf.