# ReSim, a Trace-Driven, Reconfigurable ILP Processor Simulator

Sotiria Fytraki †‡, Dionisios Pnevmatikatos†,1

† Department of Electronic and Computer Engineering,
Technical University of Crete,
Chania, GR 73100, Greece
pnevmati@mhl.tuc.gr

‡ Parallel Systems Architecture Lab (PARSA)
Ecole Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
sotiria.fytraki@epfl.ch

*Abstract* — **Modern processors are becoming more complex and as features and application size increase, their evaluation is becoming more time-consuming. To date, design space exploration relies on extensive use of software simulation that when highly accurate is slow.**

**In this paper we propose ReSim, a parameterizable ILP processor simulation acceleration engine based on reconfigurable hardware. We describe ReSim's trace-driven microarchitecture that allows us to simulate the operation of a complex ILP processor in a cycle serial fashion, aiming to simplify implementation complexity and to boost operating frequency. Being trace driven, ReSim can simulate timing in an almost ISA independent fashion, and supports all SimpleScalar ISAs, i.e. PISA, Alpha, etc.**

**We implemented ReSim for the latest Xilinx devices. In our experiments with a 4-way superscalar processor ReSim achieves a simulation throughput of up to 28MIPS, and offers more than a factor of 5x improvement over the best reported ILP processor hardware simulators.**

## I. Introduction

Modern processors are becoming more complex and as features and application size increase, their evaluation is becoming more time-consuming. Chip-multi-processors exacerbate this problem requiring the simultaneous simulation of tens or hundreds of cores, an intensive task no matter how simple the individual cores are. To accelerate the design process and increase their understanding in properties of modern processors, architects and systems designers rely on simulation tools - traditionally in software. Important characteristics of simulation models are flexibility and detail and of course performance, i.e. simulation speed. However these are conflicting goals and it is difficult to achieve all these characteristics simultaneously in one simulator.

Simulators can be categorized according to several attributes. Functional simulators are faster and just execute a program and collect simple statistics such as number of instructions, etc, but do not offer any timing information. On the other hand, timing simulators implement detailed models of processor's microarchitecture and their execution speed is much slower compared to that of functional simulators: on a 2.4GHz Xeon workstation SimpleScalar out-of-order timing simulator can simulate a set of SPEC CPU2000 benchmarks [13] at a speed of about 300K instructions per second.

In this paper we focus on improving the speed of ILP processor simulation. We propose *ReSim,* a cycle-accurate, trace-driven timing simulation engine, aimed to accelerate timing-simulation of a high-performance, modern out-of-order processors supporting speculative execution. We structure ReSim in a manner similar to SimpleScalar, and actually can use the functional simulator of the SimpleScalar tool set for trace generation. Since ReSim is trace-driven, it does not execute instructions, it merely simulates their timing. This way it requires significantly fewer resources, and executes faster. To achieve good operating frequency and facilitate future extensibility, we pipeline our simulator implementation, defining a *major* cycle that corresponds to the simulated cycle, and *minor* cycles in which ReSim itself is pipelined. The simulated processor (micro-) architectural semantics are enforced only between major cycles. ReSim for a 4-way out-of-order simulated processor fits easily in a small Xilinx FPGA device, and sustains a simulation throughput of about 23 MIPS, outperforming by more than a factor of 5 all reported, state-of-the-art hardware simulators.

ReSim can be used with traces that are prepared off-line (for example for bulk simulations with varying design parameters), or can be used in combination with a fast *functional* software simulator to efficiently add the timing information on the fly, much like the FAST approach [3].

In the following two sections we first discuss related works, and then we present the ReSim simulator overview. Section 4 analyzes ReSim's pipeline, and Section 5 presents the implementation and evaluation results. Finally, in Section 6 we offer our conclusions.

## II. Related Work

There are a number of available simulators widely used in academia and industry that can be categorised as software, hardware and hybrid.

Software simulators offer the best flexibility at the cost of low simulation speeds. SimpleScalar [2] is a set of software simulators supporting a range from functional to detailed, timing simulation of an out-of-order, superscalar processor for several ISAs. SimpleScalar's code can be configured via predefined parameters or by adding handcrafted code.

Mambo [1] is a full-system software simulator which models PowerPC based systems. It provides a range of simulation models ranging from fast, functional simulation up to a detailed timing simulator modeling the PowerPC processor micro-architectural state. Mambo provides also a cycle-approximate model which uses probabilistic measurements to improve simulation speed.

[1] Dionisios Pnevmatikatos is also with FORTH-ICS.

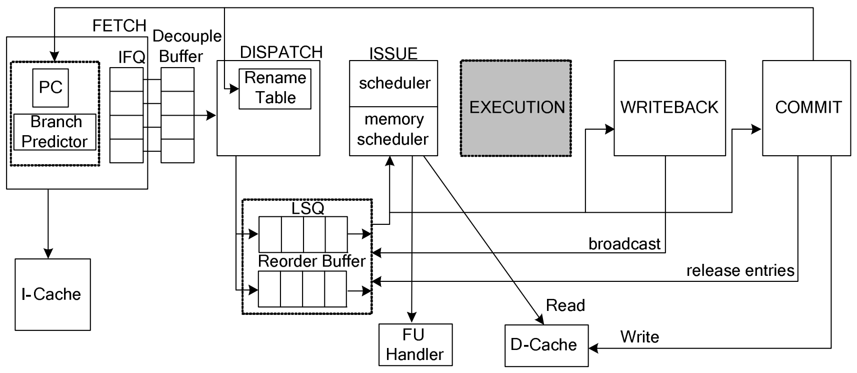

**Figure 1** ReSim Block Diagram.

Asim [6] is a cycle-accurate, functional/predictive partitioned simulator that runs the x86 ISA and has been used within DEC/Compaq/Intel. Its performance is very low in cycle-accurate mode, between 1KHz and 10KHz, as reported in [3]. PTLsim [12] is the only publicly available tool supporting cycle-accurate modeling of real x86 microarchitectures. GEMS [7] is a set of modules that work with Virtutech Simics [10] that enable detailed simulation of Sparc uni- or multi-processor systems.

The same simulation functionality can be provided by hardware, usually reconfigurable so as to adapt to changing needs and parameters. The RAMP project and FPGA platform [11] aims to accelerate performance models for large multiprocessor and multicore systems.

FAST [3] is a hardware-based, cycle-accurate computer system simulator, modelling SoCs, embedded systems and standard desktop/server computer systems, which is partitioned into a functional and a predictive (timing) model. It uses a DRC platform, boots unmodified Linux and runs SPECINT2000 benchmarks at an average rate of about 1.2 MIPS (or about 2,79 Muops) and a top performance rate exceeding 3 MIPS for some benchmarks.

HASim [5] is a recent Intel project intending to produce a hardware version of Asim. It combines both, timing and functional model, in hardware. However, it implements a simple ISA instead of a realistic one and no implementation results have been reported. A-ports [8] extend this approach with generic interfaces designed to optimize both the flexibility and the amount of parallelism within the simulator, and with an instruction set that is a subset of MIPS achieve a simulation rate of 4.7 MIPS for an out-of-order processor on a Virtex 2Pro device.

Hybrid approaches are also possible: Suh *et. al.* [9] proposed a hardware/software co-simulation method based on an off-the-shelf Pentium-III system that communicates with an FPGA via the Front-Side-Bus. A simple memory function (mem_access_latency) from SimpleScalar has been ported to the FPGA, as a proof of concept. Similarly, ProtoFlex [4] is an FPGA-accelerated, hardware-software simulator, intended to be used in large-scale multiprocessor research. It models a uniprocessor UltraSPARC III workstation with a simple 5-stage pipeline having up to 16MIPS maximum simulation speed. The processor frequently exercised behaviors are implemented in FPGA, while the remaining processor behaviors are support by embedded software simulation.

## III. ReSim Overview

ReSim aims to accelerate timing-simulation of high-performance, modern out-of-order processors supporting speculative execution. The simulated architecture is based on reservation stations and supports all the related features, such as branch prediction, load/store queues, etc, as shown in Figure 1. *ReSim* is designed to be parametarizable. We achieve this goal in two ways: coding ReSim in parametarizable VHDL, and supporting automatic generation of VHDL code whenever possible. An example is the Branch Predictor that includes a Direction Predictor, Branch Target Buffer (BTB) and a Return Address Stack (RAS). We use a script to produce VHDL code for the desired Branch Predictor according to the user parameters that include: the RAS size, the number of entries and associativity of the BTB, etc.

The main simulated stages are as follows. *Fetch* is the simulator's front end, fetching instructions from the trace until a control flow bubble is encountered or Instruction Fetch Queue (IFQ) is full. It performs target resolution of control flow instructions and checks for misfetches (i.e. when a control flow instruction is predicted taken but the predicted target PC is incorrect). On misfetch PC is set to the next sequential address, a misfetch delayed penalty is imposed. During Fetch Instruction Cache is also accessed. *Dispatch* allocates Load/Store Queue (LSQ) and Reorder Buffer (RB) entries, and accesses the Rename Table. The *Issue* stage examines the ready instructions and schedules them if there are available functional units. Load operations marked as ready by Lsq_refresh are issued and a read port is allocated if their value has not been forwarded in the LSQ. Issue also schedules a Writeback event. Loads can be issued only after their effective address has been calculated, and there are no unresolved memory dependencies. These checks are performed by *Lsq_refresh*. *Writeback* selects the oldest completed instruction(s) and broadcasts their results and wakes up all their dependent instructions. Finally, *Commit* commits the oldest RB entry releasing Store Operations to memory, if a memory write port is available, and updates the Branch Predictor in case of branch. Since ReSim is trace-driven, it only keeps track of instruction

| minor-cycles | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Fetch | DPL | F1 | F1 F2 | F1 F2 | F1 F2 | F2 | | | | | |
| Dispatch | | D | D | D | D | | | | | | |
| Issue | | | | | | | IS | IS CA | IS CA | IS CA | CA |
| Lsq_refresh | | | | | | LSQ | | | | | |
| Writeback | Wb1 | Wb1 Wb2 | Wb1 Wb2 | Wb1 Wb2 | Wb2 | | | | | | |
| Commit | | | | | | Cmt1 | Cmt1 Cmt2 | Cmt1 Cmt2 | Cmt1 Cmt2 | Cmt2 | |

**Figure 2** ReSim simple pipeline splits one simulated (major) cycle into several minor cycles. Shown for a 4-wide processor, the processing of instructions happens sequentially within the simulated pipeline stage, but overlapped with the other pipeline stages. In a Major-cycle Writeback and Lsq_refresh execution cycles precede Issue execution cycles. DPL and CA stand for Decouple and Cache Access respectively.

timing. Hence, issuing and executing instructions is combined, and the EX stage is shown grey in Figure 1.

While the simulated structure is close to many commercial processors, ReSim can be adapted to other pipelines. The closer the simulated processor to the "reference" pipeline, the easier it is to map it to the existing ReSim infrastructure.

## IV. ReSim Internal Pipeline

ReSim's pipeline is the result of analysis of the interactions between the various pipeline stages execution having as a primary goal to maximize the execution overlap, and maintain the simulated architecture semantics. We have structured ReSim at two levels: first is the simulated architecture, where all the ISA and micro-architectural state must be maintained while the second corresponds to the internal pipeline of ReSim.

Our initial attempt was to provide a truly parallel implementation in order to simultaneously simulate the N-instructions executed in the simulated processor. Such an approach requires the parallel/wide access to all internal structures: IFQ, RF, RB, rename table, etc and increase in sequential circuit delay. We have actually implemented the 4-wide parallel Fetch stage and found that besides the four-fold increase in cost, the unit was also 22% slower than fetching a single instruction. Other, more complicated structures would be even slower, while others are possibly prohibitively expensive in FPGA technology that does not support memory blocks with more than two access ports.

These observations lead us to adopt a serial execution model in ReSim. We define as a *Major* cycle the simulated processor cycle; this is the boundary were we enforce the simulated processor's (micro-) architectural semantics. We split each major cycle into several *Minor* cycles that are used to pipeline the internal ReSim's operation. We went one step further and. pipelined long operations in multiple operations (Fetch, Writeback and Commit) to achieve better operating frequency. The following subsections describe the exact operation of ReSims internal pipeline, starting with an

| minor-cycles | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Fetch | DPL | F1 | F1 F2 | F1 F2 | F1 F2 | F2 | | |
| Dispatch | | D | D | D | D | | | |
| Issue | | IS | IS CA | IS CA | IS CA | CA | | |
| Lsq_refresh | LSQ | | | | | | | |
| Writeback | | | | Wb1 | Wb1 Wb2 | Wb1 Wb2 | Wb1 Wb2 | Wb2 |
| Commit | | Cmt1 | Cmt1 Cmt2 | Cmt1 Cmt2 | Cmt1 Cmt2 | Cmt2 | | |

**Figure 3** ReSim efficient pipeline for a 4-wide processor including L1 D-Cache. A major cycle for a N-wide processor is N+4 minor-cycles.

initial simple solution, and improving it to achieve better processing throughput.

### *A. Simple Serial Execution*

In this section we describe how the internal pipeline of ReSim is organized. In serial execution model, if fetch bandwidth is N, ReSim's Issue stage will be executed N times in N minor-cycles during one major-cycle. This stands for all stages apart from Lsq_refresh, which is executed once per Major-cycle. The amount of different stages overlapped execution is constrained by both the architectural relationships as well as by implementation constraints. In general, datapath stage dependence decoupling occurs naturally; for example, Fetch and Dispatch communicate through the Decouple Buffer that allows their overlapped execution. However, this is not the case for the control. In particular, instructions waken up by their producer may be issued during the same simulated cycle. Hence at the circuit level there is dependence between writeback of one instruction to the issuing of its dependent one(s).

Figure 2 shows a simple approach to the internal pipeline for a 4-wide out-of-order superscalar processor. Within a major cycle, the stage chain involves the interdependence between writeback of the previous instruction, the LSQ update, and the scheduling of newly ready instructions in Issue. In the depicted pipeline, first Writeback is performed that broadcasts and wakes up all the dependent instructions. Then Lsq_refresh updates the LSQ to indicate possible new resolved memory dependencies. Then Issue can proceed with the scheduling of ready instructions. Outside this dependence chain, a separate minor-cycle is needed for load instructions in order to access the memory hierarchy (D-Cache). We have split Issue in two steps independently of instruction type, in order to have fixed latency major-cycle. D-Cache is also accessed when store instructions are committed. The dependence chain determines the duration of a major cycle in terms of minor cycles which in this case is 2*N+ 3, or 11 for our 4-wide example.

| Minor-cycles | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Fetch | DPL | F1 | F1 F2 | F1 F2 | F1 F2 | F2 | |
| Dispatch | D | D | D | D | | | |
| Issue | IS | IS | IS CA | IS CA | CA | | |
| Lsq_refresh | LSQ | | | | | | |
| Writeback | | Wb1 | bubble Wb2 | Wb1 bubble | Wb1 Wb2 | Wb1 Wb2 | Wb2 |
| Commit | Cmt1 | Cmt1 Cmt2 | Cmt1 Cmt2 | Cmt1 Cmt2 | Cmt2 | | |

**Figure 4** Optimized ReSim pipeline organization for a 4-wide out-of-order processor. Lsq_refresh and first Issue are executed during the same minor-cycle. The first Issue does not consider load Instructions and consequently it is not followed by a Cache Access. Latency for an N-wide processor is N+3 cycles.

### B. *Improved Serial Execution*

The description of the simple pipeline in the previous Section offers insight in ReSim key issues and implementation details. To improve performance we reorganized ReSim internal pipeline in a more efficient way, aiming at reducing in major-cycle latency. The key idea is again that ReSim does not actually execute instructions but merely simulates their execution. To improve performance we have to overlap the Issue and Writeback stages. This is allowed as long as the results of an executed instruction are broadcast and their dependencies updated before the beginning of a cycle. This is the well-known technique of pipelined control where the scheduling of the broadcast bus happens (one cycle) before the actual instruction completion, i.e. we have to perform Writeback in ReSim *one cycle before* its actual execution in the simulated pipeline. When the operation latency is greater than 1 this is straightforward.

For operation latency equal to one the consequence is that Issue and Writeback may have to occur in the same major cycle. To ensure correct timing, the Issue minor-cycle is performed *before the* Writeback minor-cycle during a major-cycle. We use a flag to prevent Commit from considering such instructions within the same major cycle – despite the fact that the instructions may be marked completed. The necessary bookkeeping tasks are performed during the last minor-cycle of simulated cycle so that their effects will be visible to Lsq_refresh at the beginning of the next major-cycle. A cache access occurs before writeback to determine whether there is a hit, or the writeback must be postponed. Taking into account all the issues analyzed above, the major-cycle latency is N+4 minor-cycles and the pipeline is shown in Figure 3 for a 4-wide out-of-order processor.

In the general scenario discussed in this Section, up to N load/stores can be issued/committed every major-cycle, as per the processor configuration. Typical N-wide processor will provide less than N memory ports. Based on this observation, we have further optimized ReSim internal pipeline. Without affecting the overall timing results, we disallow the issue and execution of a load instruction in the first slot of the major cycle, and hence we allow the execution of Lsq_refresh and of the first Issue to be performed in parallel. The optimized pipeline shown in Figure 4, has a major-cycle of N+3 cycles for a N-wide processor with the restriction that the simulated processor has up to N-1 memory ports.

## V. RESIM IMPLEMENTATION AND EVALUATION

Next we describe important implementation details and evaluate ReSim in terms of performance and cost.

### A. *Mis-speculation Handling*

ReSim's input trace consists of a record for each dynamic instruction in a pre-decoded format. Three formats are used: Branch (B), Memory (M) and Other (O), each with its own fields and length. Since the trace format is decoded and generic, ReSim supports all ISAs that can be described by it. The trace is produced by a modified (SimpleScalar) functional simulator, and all formats include a Tag Bit field used for mis-speculation handling.

To produce a trace that includes incorrect path instructions and simulate the effects of mis-speculation we use a functional simulator which includes branch predictor (sim-bpred). More specifically, our trace generation code inserts in the trace a number of incorrectly fetched instructions called *wrong path block* after each mis-predicted branch instruction. These instructions are tagged as mis-speculated, e.g. Tag Field is set to one. Upon reaching the mis-prediction point, ReSim will fetch the instructions from the wrong path and model their effects in instruction processing, caches, etc. Tagged instructions that have not been fetched by the branch resolution point at Commit, are discarded. A very conservative assumption for the wrong path block size is equal to Reorder Buffer size plus IFQ size.

### B. *Collecting Statistics*

During simulation, ReSim collects various statistics that are similar to the ones found in sim-outorder timing simulator of SimpleScalar Tool Set. To avoid overflow problems we use 64-bits registers for statistics. We collect general information such as Total Number of instructions, memory operation, branches, cache hits etc. ReSim also gathers statistics about IFQ, Reorder Buffer and LSQ and collects detailed information about branches. Additional statistics can be easily added to ReSim's code.

### C. *Performance Evaluation*

ReSim implementation is based on serial execution model of Figure 3 for a 4-way superscalar processor with 16 Reorder Buffer entries and 8 LSQ entries, four ALUs, one Multiplier and one Divider with one, three and ten cycle latency respectively. Misfetch and misspeculation penalty are parameters, set to three in the current implementation. The Branch Predictor is fully parametric and various

**Table 1.** In the left portion, ReSim's Simulation Performance with perfect memory on Virtex 4 (xc4vlx40) and Virtex 5 (xc5vlx50t) devices for SPECINT CPU2000 (input=train). Major-cycle latency is N+3 minor-cycles. In the right portion, ReSim's Simulation Performance for the same FPGA devices and a 2-issue processor with 32K L1 Instruction and Data Cache, with associativity of 8 and block size 64 bytes,. ReSim Major-cycle latency is N+4 minor- cycles. In the last column, FAST [3] Simulation Performance is reported in for perfect branch prediction, scaled from x86 MIPS into simulated Muops per second.

| SPEC Program | Perfect Memory System | | 32KByte L1 Cache | | |
|---|---|---|---|---|---|
| | ReSim 4 issue, 2-lev BP Sim. Speed (MIPS) | | ReSim 2 issue , perfect BP Sim. Speed (MIPS) | | FAST 2 issue, perfect BP Sim. Speed (MuOps) |
| | Virtex 4 | Virtex 5 | Virtex 4 | Virtex 5 | Virtex 4 |
| gzip | 23.26 | 29.07 | 20.44 | 25.55 | 2.95 |
| bzip2 | 27.55 | 34.44 | 18.53 | 23.16 | 3.51 |
| parser | 19.94 | 24.92 | 16.70 | 20.88 | 2.82 |
| vortex | 23.57 | 29.46 | 16.83 | 21.04 | 2.19 |
| vpr | 20.38 | 25.48 | 19.16 | 23.95 | 2.48 |
| **Average** | **22.94** | **28.67** | **18.33** | **22.92** | **2.79** |

configurations can be produced according to a full set of user parameters as explained in Section III. The Branch Predictor included in ReSim contains a Return Address Stack with 16 entries, a direct-mapped BTB with 512 entries. The Branch History Table size, History Register length and PHT are 4, 8 and 4096 respectively. We evaluate ReSim with two memory system configurations: (i) perfect memory system, and (ii) 32KByte, two-way associative level-1 instruction and data caches.

We implemented ReSim in a Virtex 4 (xc4vlx40) and a Virtex 5 (xc5vlx50t) devices using Xilinx ISE 9.1i. The operating frequency we achieved for minor-cycles is 84 and 105 MHz for Virtex 4 and Virtex 5 devices respectively.

In the left portion of Table 1, we show the simulation throughput of ReSim for a 4-issue out-of-order processor with perfect memory and a two-level branch predictor using five of SPECINT CPU2000 programs. The pipeline latency is N+3=7 cycles in this configuration. The average simulation throughput is 22.94 and 28.67 MIPS for Virtex 4 and Virtex 5 respectively.

**Table 2** Architectural Simulator Performance

| Simulator | ISA | Speed (MIPS) |
|---|---|---|
| PTLSim | x86-64 | 0.27 |
| sim-outorder | *PISA* | 0.30 |
| GEMS | Sparc | 0.07 |
| FAST | x86, gshare BP | 1.2 |
| FAST | x86, perfect BP | 2.79 |
| A-Ports | MIPS subset, 4-wide | 4.70 |
| ReSim | PISA, 2-wide, perfect BP, Virtex5 | 22.92 |
| ReSim | PISA, 4-wide, 2-lev BP, Virtex5 | 28.67 |

In the right portion of Table 1, we attempt a fair comparison with FAST [3]. We use instruction and data L1 caches of 32Kbytes, associativity of 8 and block size of 64 bytes, same as the L1 cache configuration of [3], and we simulate a 2-issue out-of-order processor with perfect branch predictor. The pipeline latency is N+4=6 cycles, in this configuration, and the average simulation throughput is 18.33 MIPS and 22.92 MIPS for Virtex 4 and Virtex 5 devices respectively. ReSim best performance is over 20 MIPS and 25 MIPS for the benchmark bzip2 for Virtex 4 and Virtex 5 FPGA technology respectively. We scale the reported x86 MIPS simulation speed results into simulated Muops per second, and find that for a common implementation technology, ReSim improves the simulation speed 6,57 times over FAST. However, these results do not include the generation/transfer time for the trace that may slowdown the simulation speed, while FAST is a complete system simulator that includes these overheads.

A direct comparison with A-ports [8] is not possible due to different implementation technologies. A-ports implement a subset of the MIPS instruction set and [8] reports a simulation speed of 4.7 MIPS on a Virtex2Pro device for an MIPS R10K 4-way superscalar, out-of-order issue processor. It is unclear whether they include a cache or a memory system. Based on the published information, we estimate the ReSim's speedup roughly at a factor of 5x compared to A-ports.

In Table 2, we present simulation speed of four simulators as reported in [3], A-Ports, and two ReSim configurations using Virtex 5 FPGA technology. FAST and A-ports outperform software simulators by orders of magnitude. Furthermore, ReSim outperforms FAST and A-ports, the best reported ILP processor hardware simulators by at least a factor of 5.

**Table 3** Resim Throughput Statistics

| SPEC | bits /Instr. | Simulation Throughput (MIPS) | Trace Throughput (MByte/sec) |
|---|---|---|---|
| gzip | 41.74 | 26.37 | 137.56 |
| bzip2 | 41.16 | 29.43 | 151.39 |
| parser | 43.66 | 22.83 | 124.58 |
| vortex | 47.14 | 24.47 | 144.20 |
| vpr | 43.52 | 24.44 | 132.94 |
| **Average** | **43.44** | **25.51** | **138.13** |

**Table 4** Area Cost on a Virtex 4 (xc4vlx40) device

| FPGA resources | Stage-Structures Area (%) of Total Design | | | | | | | | | | | | Total Area Cost |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | fetch | disp | issue | lsq | wb | Cmt | RT | RB | LSQ | BP | D-C | I-C | |
| Slices | 25 | 9 | 5 | 14 | 3 | 2 | 3 | 13 | 6 | 2 | 17 | 1 | 12273 |
| 4-input LUTs | 23 | 5 | 7 | 19 | 4 | 2 | 4 | 14 | 4 | 2 | 15 | 1 | 17175 |
| BRAMs | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 71 | 0 | 29 | 7 |

A performance concern for ReSim is the input trace throughput demands. We gathered statistics in our experiments and summarize our findings in Table 3 for the five SPECINT CPU2000. We have used ReSim having a perfect memory system in Virtex 4 technology. The table lists the average number of trace bits per instruction for the five benchmarks and ReSim's simulation throughput *including* mis-speculated instructions. This corresponds to the total trace instruction demands and also gives an indication of the cost due to mispredictions which is about 10%. In the last column, we list the required ReSim input trace throughput in MBytes per second. While this throughput (1.1Gbps) exceeds the available bandwidth of regular Gigabitr Ethernet network, tightly coupled CPU-FPGA systems –such as the DRC board- are available and use busses that offer substantially higher I/O bandwidth.

Finally, ReSim area cost is analyzed in Table 4. We present the area cost of the entire design and of each individual stage as a percentage of the total cost. We used Block RAMs only in the Branch Predictor, and used distributed RAMs that are more efficient for other structures such as Rename Table. In Table 4, Fetch and Dispatch area costs include the IFQ and decouple buffer respectively. The total area reported in this table does not include instruction and data caches (I-C and D-C). Since we do not store the actual data, we need to provide only the hit/miss indication and simulate the access latency, so the actual cache requirements are in the range of 1000 slices plus a few memory blocks for the tags. The area cost of 4-wide FAST configuration implemented in Virtex 4 FPGA technology is 29230 Slices and 172 BRAMs, which is 2.4 times and 24 times larger than ReSim corresponding Area Cost.

## VI. Conclusions – Future Work

In this paper we presented ReSim, trace-driven simulator geared to accelerate timing simulation of a high-performance, modern out-of-order ILP processors supporting speculative execution. We present the micro-architecture and ReSim's internal pipeline, the key to achieve good simulation performance. Our approach is simple to extend and many blocks are parametric. We are investigating the creation of a software tool that would automatically produce custom ReSim versions according to user parameters.

The performance of ReSim exceeds by more than a factor of 5 that of the best reported ILP processor simulation speed. ReSim is also very small and fits within about 10K Xilinx FPGA slices and uses just a few memory blocks. Therefore it is possible to fit multiple ReSim instances in a single FPGA and simulate multi-core systems. We are evaluating the modifications and extensions that need to be made to ReSim in order to support multi-core simulation. We also investigate ways to produce the trace on the fly directly from a functional simulator, an idea proposed in [3].

## References

[1] P. Bohrer, J. Peterson, M. Elnozahy, R. Rajamony, A. Gheith, R. Rockhold, C. Lefurgy, H. Shafi, T. Nakra, R. Simpson, E. Speight, K. Sudeep, E. Van Hensbergen, and L. Zhang. "Mambo: A Full System Simulator for the PowerPC Architecture," *SIGMETRICS Perform. Eval. Rev.*, 31(4):8–12, 2004.

[2] D. Burger and T. Austin. "The SimpleScalar Tool Set Version 2.0," Technical Report 1342, Computer Sciences Department, University of Wisconsin–Madison, May 1997.

[3] D. Chiou, D. Sunwoo, J. Kim, N. Patil, W. Reinhart, D. Johnson, Z. Xu, "The FAST Methodology for High-Speed SoC/Computer Simulation". In *Proceedings of the 2007 International Conference on Computer-Aided Design,* pp. 295-302, November 2007.

[4] E. Chung, E. Nurvitadhi, J. C. Hoe, B. Falsafi and K. Mai. "Virtualized Full-System Emulation of Multiprocessors using FPGAs," *2nd Workshop on Architectural Research Prototyping*, June 2007.

[5] N. Dave, M. Pellauer, Arvind, and J. Emer, "Implementing a Functional/Timing Partitioned Microprocessor Simulator with an FPGA," in *Proceedings of the Workshop on Architecture Research using FPGA Platforms*, Feb. 2006.

[6] J. Emer, P. Ahuja, E. Borch, A. Klauser, C.-K. Luk, S. Manne, S. S. Mukherjee, H. Patil, S. Wallace, N. Binkert, R. Espasa, and T. Juan, "Asim: A performance model framework," *Computer*, vol. 35, no. 2, pp. 68–76, 2002.

[7] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, D. Wood, "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset," *Computer Architecture News (CAN)*, Sept. 2005.

[8] M. Pellauer, M. Vijayaraghavan, M. Adler, Arvind, J. Emer: "A-Ports: An Efficient Abstraction for Cycle-Accurate Performance Models on FPGAs", *FPGA,* 2008.

[9] T. Suh, H.-H. S. Lee, S.-L. Lu, and J. Shen, "Initial Observations of Hardware/Software Co-Simulation using FPGA in Architectural Research," in *Proceedings of the Workshop on Architecture Research using FPGA Platforms, held at HPCA-12*, Feb. 2006.

[10] Virtutech Simics. http://www.virtutech.com/

[11] J. Wawrzynek, D. Patterson, M. Oskin, L. Shin-Lien C. Kozyrakis, J.C. Hoe, D. Chiou, K. Asanovic, "RAMP: Research Accelerator for Multiple Processors," *IEEE Micro*, vol.27, no.2, pp.46-57, March-April 2007.

[12] M. Yourst. PTLSim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator. *ISPASS*, Jan. 2007.

[13] http://www.spec.org/cpu2000/