

HARDWARE/SOFTWARE PROCESS MIGRATION AND RTL SIMULATION

Aric D. Blumer[†], Cameron D. Patterson

Bradley Department of Electrical and Computer Engineering
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
email: aric@vt.edu, cdp@vt.edu

ABSTRACT

This paper describes an execution cache that uses process migration between hardware and software contexts by way of run-time reconfiguration (RTR) of Field Programmable Gate Arrays (FPGAs). The feasibility of such a system is demonstrated using existing FPGAs by accelerating a cycle-based simulation of a Register Transfer Level (RTL) design description. Through the use of a common instruction set, each simulation process may be run in a software Virtual Machine (VM) or in a hardware Real Machine (RM). The implementation provides data for an empirical model used to examine the behavior of unimplemented parts of the system.

1. INTRODUCTION AND OVERVIEW

Traditional computing hardware is static, but Field Programmable Gate Arrays (FPGAs) provide some hardware flexibility through reconfiguration. An FPGA configuration is normally static at run time, but it is possible to change a configuration while running through run-time reconfiguration (RTR). RTR opens new avenues to computing system flexibility such as the migration of processes between hardware and software. By instantiating and removing processors in the logic array at run time, users can establish an adaptable, parallel execution system. It can be tuned by removing unused logic, establishing communication routes on demand, and by compiling processes directly to hardware.

This flexibility allows us to exploit locality of reference in a similar manner as memory or disk caches. Spatial locality occurs when data items near an accessed item are also accessed. Temporal locality occurs when an accessed data item is accessed again in the near future. *Executive locality of reference* refers to locality among executing processes. *Executive temporal locality* occurs when an executing process is executed again, and it can be exploited by keeping the most active processes in parallel execution units. *Executive spatial locality* occurs when processes communicating with an executing process are executed, and it can be exploited

by keeping collaborating processes in the FPGA with direct communication links. As in data caches, inactive processes are migrated out, and active processes are migrated in.

Hardware acceleration of simulations uses dedicated hardware to emulate all or part of the design under test, and FPGAs serve that purpose well. These accelerators are utilized by first mapping the Register Transfer Level (RTL) code to an array of FPGAs, offloading the work from a single processor to gain a speedup. This mapping is a time-consuming process requiring a non-negligible delay before the simulation can commence. Resources are allocated at compile time with no possibility of adjustment during run time. Furthermore, hardware accelerators are costly, ranging from hundreds of thousands to millions of dollars [1]. Through exploiting executive locality of reference with migration, however, a speedup may be achieved with less hardware without an initial mapping to hardware. The remainder of this document describes the application of process migration to the acceleration of RTL simulations.

A parallel RTL simulator consists of concurrent processes, processors, and communication infrastructure. If the processes may reside in software or hardware contexts, they must be in a form that is portable. A Common Instruction Set (CIS) serves that purpose. Processes in CIS form can then be executed in a system composed of simple Virtual Machines (VMs) and Real Machines (RM). A set of processes begins running entirely in VMs, one per process. The use of VMs allows the system to begin execution immediately while maintaining portability. Processes consuming the most time can be migrated to the RMs or compiled to native code in VMs. Idle processes can be left in CIS form until they are active. An analysis of a process's CIS can also indicate what instructions the target RM must support.

The efficiency of an RTL simulator that exploits executive locality of reference depends fundamentally on identifying idle and active processes to keep the acceleration hardware fully utilized with busy processes. If all processes are active during the simulation, then the parallel execution of a portion of the processes results in only a modest speedup dictated by Amdahl's Law, and only a single, initial migra-

[†]This work was graciously funded through a Virginia Tech College of Engineering Fellowship and a Bradley Fellowship.

tion is necessary. If, however, they have phases of activity, migration achieves greater efficiency by keeping the parallel array fully utilized. One method of identifying RTL process activity is to monitor a process's inputs. When a process is executed, if its inputs have not changed from the previous invocation, the outputs will not change. Therefore, the process does not need to be executed until an input does change.

2. IMPLEMENTATION DETAILS

The purpose of implementing this migration system is to prove the concept using existing FPGAs and to measure system performance for empirical modeling. Further development of the system is required for it to be complete, but empirical modeling can be used to evaluate the behavior of unimplemented parts. The system currently lacks just-in-time VM compiling to native code and run-time routing between RMs, so the effect of implementing these is evaluated in Section 2.2.

The system is implemented on an XUPV2P board developed for the Xilinx University Program. It is populated with a Virtex-II Pro FPGA (XC2VP30) and 512 megabytes of DDR memory. Using the Xilinx EDK version 7.1i, the design was instantiated with a DDR memory controller, the On-chip Peripheral Bus (OPB) infrastructure, the Internal Configuration Access Port (ICAP), an ICAP controller, and an array of RMs. A VM-based cycle simulator runs on the left PowerPC, equipped with code to migrate the state of the VMs to and from the RMs. The internal structure of both the VMs and RMs is shown in Fig. 1. Migration transfers the state in program memory, data memory, and the register file between VMs and RMs using RTR.

The original RMs were fully pipelined using flip-flops for the register file, but Virtex-IIs do not provide a way to update flip-flop state *on a per-RM basis*. The GRESTORE ICAP command updates all flip-flops in the FPGA including those of critical infrastructure [2]. A solution is to use Look-Up Table (LUT) distributed RAM instead. LUT RAMs can be updated on a per-RM basis, but they cannot provide the single-in-double-out interface required by pipelining unless the memories are doubled. We elected instead to execute an instruction every two cycles in a non-pipelined fashion, and the absence of data hazards makes the RMs smaller.

There is another limitation of LUT RAMs, however. A frame is the minimum configuration unit in an FPGA, and Virtex-II frames span the entire height of the device [2, p. 337]. Some of the logic used to load the frames is used by LUT RAMs during normal operation. Hence, no LUT RAMs within a frame can be read or written while the frame is being configured, but infrastructure such as the DDR controller utilizes them. Merely reading the frames used by the DDR controller LUT RAMs during run time causes a malfunction. Protective floorplanning is required to ensure that the RMs do not share any frames with this infrastruc-

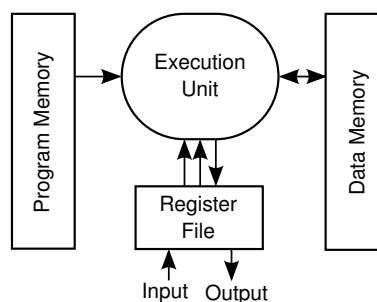


Fig. 1. VM and RM Block Diagram

ture. Each RM is constrained to occupy a 16x16 slice area, and they are floorplanned into columns with the remaining frames reserved for infrastructure. After protective floorplanning, 35 RMs fit with an 82% slice utilization.

2.1. The Simulator Application

The even-odd transposition (EOT) sort [3] is a good application to measure the performance of the simulator because it allows the number of RMs and the number of simulation cycles to be varied while still giving correct results. Each iteration of the EOT sort is a swapping—or transposition—of the numbers to place the greater number on the right. When executed in parallel, it requires a maximum of n simulation cycles (“simcycles”) to sort n numbers when there are $\frac{n}{2}$ iterations per simcycle. The sort was implemented in VHDL as an array of processes, each comparing its own output with its left or right neighbor’s output on alternating simcycles. The VHDL was translated manually into the CIS which is then assembled into executable VM/RM code. The CIS, in its current form, is like an assembly language. To make the RMs small, they have 16-bit instructions with eight 16-bit registers, some aliased to inputs and outputs. Two 32x16 LUT RAMs hold the data and code.

The PowerPC in the Virtex-II Pro operates at 300 MHz while the remaining logic operates at 100 MHz. The PowerPC executes the VMs and the simulation infrastructure with the “standalone” software package provided in Xilinx’s EDK. Inputs and outputs of the RMs are connected to the On-chip Peripheral Bus (OPB) through slave registers. Between simcycles, the simulator reads the outputs of the RMs and writes their inputs, a process called *software connectivity*. By contrast, *hardware connectivity* joins RM outputs to RM inputs directly in the FPGA. To measure the difference in overhead between the two types, a number of tests were run with hardware connectivity written into the VHDL code. A complete system would instead implement these connections at run time, and that is modeled later.

2.2. Results and Modeling

The FPGA holds 35 RMs, so the EOT sort was run with 35 processes. To evaluate pure parallel performance, the

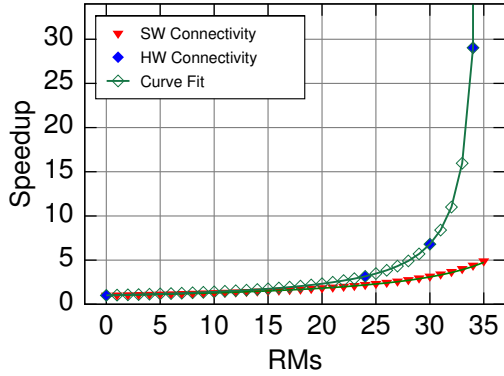


Fig. 2. Speedup of HW and SW connectivity (35 simcycles)

speedups with no migration were measured for both software and hardware connectivity. Fig. 2 shows the resulting plots. Because hardware connectivity is currently manual, its overhead was measured with 0, 24, 30, 34, and 35 RMs, and Marquardt-Levenberg curve fitting provides the other points. The speedup is the run time without using RMs (T_0) divided by the run time with RMs (T_r). Fig. 2 shows clearly that hardware connectivity is a necessary improvement. One result of executing code in VMs without native just-in-time compilation is a “super-linear” speedup because the RMs execute their processes faster than the VMs. Not shown in Fig. 2 due to scaling disparity is the speedup of 1005 when using hardware connectivity with 35 RMs. For illustration, consider a serial program that executes 10 tasks, each taking 1 s. If those tasks are parallelized on 10 processors, they all execute in 1 s for a speedup of 10. If, however, the tasks are executed on parallel processors that are ten times faster, they complete in 0.1 s for a speedup of 100.

Migration overheads are affected by reconfiguration time, and the Virtex-II architecture does not lend itself well to quick reconfiguration. In addition to the frames spanning the entire height, frame reads and writes require a “dummy” frame to be read or written, and the ICAP has only an 8-bit interface run at a lower clock rate [4, p. 4]. Multiple frames must be transferred per migration, and the unoptimized migration time per machine exceeded 220 ms. However, the full-height span of frames can be exploited. The RMs are floorplanned in columns of 8, except for the rightmost column which contains 11. For each RM, an RLOC constraint places the program, data, and register file memories into the same frames. Sharing frames among RMs also allows frame caching, which minimizes ICAP accesses by reading and writing a shared frame only once during a series of migrations. These optimizations reduce the average migration time to 6.0 ms, and the maximum for one RM is 18.9 ms.

The time required to execute the sorting simulation with software connectivity is $T(r) = C(t_o + rt_r + [P - r]t_v) + rMt_m$, where t_o , t_r , etc. are described in Table 1. This equation does not include t_R , the run-time routing time per

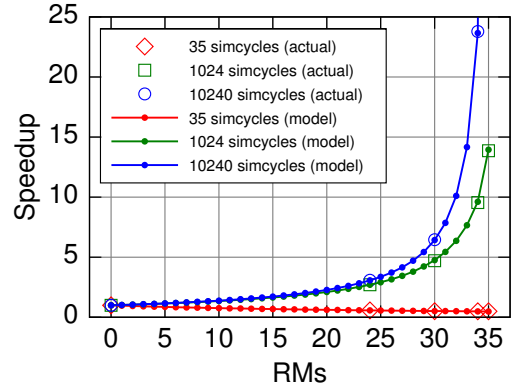


Fig. 3. HW connectivity and empirical model

Table 1. Modeling Parameters

Param.	Description	Value
C	Simcycles	<i>varies</i>
P	Processes	<i>varies</i>
r	Number of RMs	<i>varies</i>
M	Number of Migrations	<i>varies</i>
t_v	VM Execution Time	$83.13 \frac{\mu\text{s}}{\text{simcycle}}$
t_r	RM Execution Time	$17.03 \frac{\mu\text{s}}{\text{simcycle}}$
t_o	Simulator Overhead	$2.96 \frac{\mu\text{s}}{\text{simcycle}}$
t_m	Migration Overhead	$6.02 \frac{\text{ms}}{\text{migration}}$
t_R	Run-time Routing Overhead	<i>varies</i>

migration, which is used in the empirical model discussed later. The sort tests use $C = P = 35$, while r varies from 0 to 35. t_r includes any RM-related overhead (including the software connectivity accesses) plus any time spent waiting for the RMs to complete.

Speedup is defined as $S = T_0/T(r)$, which can be modeled as follows for hardware connectivity:

$$S = \begin{cases} \frac{T_0}{C(t_o + Pt_v)}, & \text{for } r = 0, \\ \frac{T_0}{C(t_o + t_r + [P - r]t_v) + rMt_m}, & \text{for } 0 < r < P, \\ \frac{T_0}{Ct_o + rMt_m}, & \text{for } r = P. \end{cases} \quad (1)$$

This piecewise equation differs from software connectivity because of the offloading of RM communication. In the first case, when zero RMs are in use, there are no RM overheads t_r and t_m . For the middle region, there is a single t_r for the single RM that has hardware connectivity on one side and software connectivity on the other. The remaining RMs have only hardware connectivity. For the final case, nothing is run in VMs, so there is only the simulator and migration overheads.

Since the EOT sort gives correct results for longer runs, Fig. 3 shows speedup plots for 35, 1024, and 10240 simcycles, including all migration overheads for one migration per RM ($M = 1$). Larger simcycle-to-migration ratios (SMRs) amortize the migration overhead over a longer period of

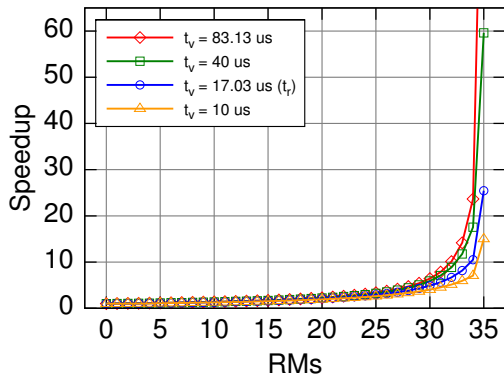


Fig. 4. Modeled effect of t_v on speedup (10k simcycles)

time, improving the speedup. For small SMRs (e.g., 35:1), the migration time exceeds the simulation run time, and the system exhibits a speedup less than one. Using these measured values with Equation 1, the Marquardt-Levenberg curve fitting algorithm solves for t_r , t_v , and t_m as shown in Table 1. t_o was calculated directly from the third case of Equation 1 when $M = 0$. These values are used for the subsequent graphs unless otherwise noted.

The following equation extends Equation 1 with $T_0 = T(0) = Ct_o + PCt_v$, and t_R to model the general speedup of this migration system:

$$S = \begin{cases} \frac{Ct_o + PCt_v}{C(t_o + [P-r]t_v) + rM(t_m + t_R)} = 1, & \text{for } r = 0, \\ \frac{Ct_o + PCt_v}{C(t_o + t_r + [P-r]t_v) + rM(t_m + t_R)}, & \text{for } 0 < r < P, \\ \frac{Ct_o + PCt_v}{Ct_o + rM(t_m + t_R)}, & \text{for } r = P, \end{cases} \quad (2)$$

Note that the first case is $T_0/T(0)$, which simplifies to 1 as expected. This model fits the 35-, 1024-, and 10240-simcycle plots of Fig. 3 to within 2.3%, 1.0%, and 0.61% error, respectively.

No complete system would run CIS code in VMs without compiling it to native code. The empirical model of Equation 2 is used to explore the effect of VM-related run times (t_v). Fig. 4 shows that speedups of 15 are possible even if $t_v < t_r$. Since t_v includes the time to execute a process and to update process inputs and outputs, it is unlikely to be less than t_r .

Finally, consider the theoretical effect of run-time routing for hardware connectivity, modeled with t_R . Fig. 5 shows that routing overhead could have a significant detrimental effect on the speedup, unless efficient ways of routing are developed. Route times are difficult to estimate, but one aspect of the system suggests that routing can be done quickly: Because RM inputs are updated between simcycles during t_o , the RM inputs can tolerate multi-cycle delays. Regardless, as with migration overhead t_m , the speedup can be improved if the simulation characteristics allow us to amortize t_R over a greater number of simcycles. As shown on the plots, an

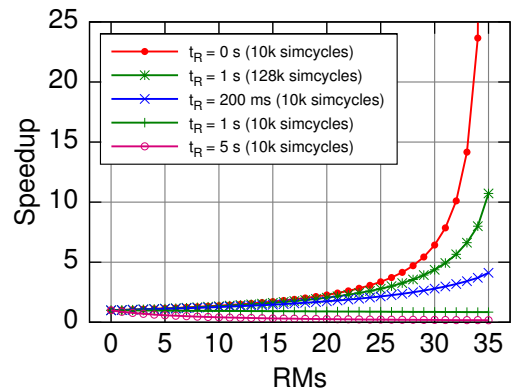


Fig. 5. Modeled effect of t_R on speedup ($t_m = 6.0$ ms)

SMR of 128k:1 for the $t_R = 1$ s plot improves the speedup from 0.85 to 10.7.

3. CONCLUSION

If expressed as simcycles per second, these results are less than a Pentium 4 or Opteron simulating at gigahertz clock frequencies, but an absolute performance comparison is not appropriate. Rather, the results show a relative speedup between a processor alone and a processor with a migration coprocessor, and one would expect similar speedups for faster hardware. Using DRC Computer Corporation's RPU110 [5], paired with an Opteron processor may result in a faster system compared to an Opteron alone. The RPU110 also utilizes the Virtex-4 family which has performance, density, and reconfiguration improvements over Virtex-II.

This paper introduced the concept of an execution cache using process migration, and demonstrated the migration of processes into a parallel execution system in a Virtex-II FPGA. The measured results of the proof-of-concept RTL simulator implementation allow us to model more general systems to explore the effects of overheads and execution speeds.

4. REFERENCES

- [1] G. D. Peterson, "Evaluating simulation acceleration techniques," in *Enabling Technology for Simulation Science V*. SPIE, 2001.
- [2] Xilinx, Inc., *Virtex-II Pro and Virtex-II Pro X FPGA User Guide*. Xilinx, Inc., March 2005.
- [3] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*, 2nd ed. Addison Wesley, 2003.
- [4] D. R. Curd, "Dynamic reconfiguration of RocketIO MGT attributes," <http://direct.xilinx.com/bvdocs/appnotes/xapp660.pdf>, February 2004.
- [5] DRC Computer Corporation, "RPU110: DRC reconfigurable processor unit," http://drccomputer.com/pdfs/DRC_RPU110_datasheet.pdf, 2006.