<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Scheduling mixed-architecture processes in tightly coupled FPGA-CPU reconfigurable computers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Hamilton, BK; Inggs, M; So, HKH</td>
</tr>
<tr>
<td><strong>Citation</strong></td>
<td>The IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM 2014), Boston, MA., 11-13 May 2014. In Conference Proceedings, 2014, p. 240</td>
</tr>
<tr>
<td><strong>Issued Date</strong></td>
<td>2014</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td><a href="http://hdl.handle.net/10722/201237">http://hdl.handle.net/10722/201237</a></td>
</tr>
<tr>
<td><strong>Rights</strong></td>
<td>This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.; ©2014 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.</td>
</tr>
</tbody>
</table>
Scheduling Mixed-Architecture Processes in Tightly Coupled FPGA-CPU Reconfigurable Computers

Brandon Kyle Hamilton, Michael Inggs
Department of Electrical Engineering
University of Cape Town
Cape Town, South Africa
{brandon.hamilton, michael.inggs}@uct.ac.za

Hayden Kwok-Hay So
Department of Electrical & Electronic Engineering
University of Hong Kong
Hong Kong
hso@eee.hku.hk

Abstract—The design and implementation of a multitasking run-time system on a tightly coupled FPGA-CPU platform is presented. Using a mix of CPU and FPGA programmable logic for computing, user applications are executed as mixed-architecture processes from the perspective of the OS. Context switching mechanisms with hybrid scheduling containing both blocking and preemption support were implemented to support concurrent execution of multiple mixed-architecture processes, and evaluated under a synthetic workload.

I. SYSTEM OVERVIEW

This work presents the preliminary design and implementation of a mixed-architecture application scheduling mechanism targeted toward a reconfigurable processor based on a unified machine model called MURAC. The system was implemented as a Linux kernel module and was validated on the Xilinx Zynq platform.

The Multiple Runtime Architecture Computer (MURAC) is a unified machine model providing an abstraction for utilizing different compute architectures during runtime. Computation utilizing programmable logic (PL) is modeled as an architectural branch from the main control flow of the application. It eliminates the implicit forking of a user process when a PL accelerator is used, and helps to reduce the amount of complex intra-process synchronization between processor and PL.

A mixed-architecture application therefore contains standard machine instructions as well as PL configurations directly embedded at the desired point of execution in the application binary. During runtime, the machine automatically configures the PL from the embedded configuration as part of the normal program control flow. The next CPU instruction resumes execution upon completion of the PL portion. The main process scheduler must therefore handle concurrent processes executing in both CPU and PL at the same time.

II. SCHEDULING MIXED-ARCHITECTURE PROCESSES

The Linux kernel software scheduler (CFS) handles CPU resource allocation for all executing processes as normal. During the execution time of a mixed-architecture process, the PL resource allocation mechanism will be invoked if the application is currently in the PL execution state or an architectural branch is encountered due to an embedded PL configuration. If the PL is available it will be configured and the process will then yield on the CPU or, depending on the strategy: (a) block and wait, yielding to reduce its virtual runtime until the PL becomes available, or (b) preempt, performing a PL context switch. A PL context switch is implemented using read-back and reconfiguration of the PL region. This context switch is non-interruptible, and will influence the virtual runtime of the active process. Due to the difference between the latency of PL context switch, the allocated execution time before being preempted on the CPU, an application running on the PL will only be PL preemptable after a minimum execution time to avoid thrashing.

III. IMPLEMENTATION RESULTS

A synthetic workload application is used to simulate variable length execution time programmable logic algorithms, consisting of a simple context thread of software code with an embedded long-running PL design. On the Zynq XC7Z020 processor, the embedded partial bitstream has a size of 54944 words, leading to a hardware context switch (read-back and reconfiguration) time of 45ms, approximately 3 orders of magnitude slower than the kernel software context switch overhead. Due to this high hardware context switch overhead, the blocking strategy (minimizing context switching) works well for applications with shorter-running programmable logic sections, especially if the PL execution time granularity between processes is similar. However, if the PL context switch latency is much smaller than the overall PL execution time and the minimum execution time allowed to the process, preemption will produce better overall system performance in a multitasking environment.

IV. CONCLUSION

We have designed and implemented scheduling mechanisms in a tightly coupled CPU-FPGA system where applications may execute in a mixed-architecture environment. The blocking and preemptive scheduling strategies were examined. Preliminary results showed that blocking is a good baseline strategy given the long context switch time for PL processes. Future research will examine the detailed interaction between the scheduler and system application mix.