Low Cost Task Migration Initiation in a Heterogeneous MP-SoC [*]

V. Nollet, P. Avasare, J-Y. Mignolet, D. Verkest[†]
IMEC, Kapeldreef 75, 3001 Leuven, Belgium
[†]also Professor at Vrije Universiteit Brussel and at Katholieke Universiteit Leuven
{nollet, avasare}@imec.be

Abstract

Run-time task migration in a heterogeneous multiprocessor System-on-Chip (MP-SoC) is a challenge that requires cooperation between the task and the operating system. In task migration, minimization of the overhead during normal task execution (i.e when not migrating) and the minimization of the migration reaction time are important. We introduce a novel technique that reuses the processor's debug registers in order to minimize the overhead during normal execution. This paper explains our task migration proof-of-concept setup and compares it to the state-of-the art. By reusing existing hardware and software functionality our approach reduces the run time overhead.

1. Introduction

Today's state-of-the-art MP-SoC platforms contain multiple heterogeneous processing elements (PEs). An operating system (OS) is responsible for allocating the computing resources in an optimal way. In order to react to varying run-time conditions, the OS requires heterogeneous task migration capabilities. Run-time task migration can be defined as the relocation of an executing task from its current location, the source PE, to a new location, the destination PE. This allows the OS to e.g. maximize energy savings of dynamic voltage and frequency scaling techniques. It also enables thermal chip management by moving tasks away from *hot PEs*. The architectural differences between the source PE and destination PE are masked by capturing and transferring the *logical task state* (i.e. PE-independent state), shown by Figure 1.

In order to relocate a task, the operating system notifies the task by means of a *migration request* signal $^{(1)}$. Whenever that signaled task reaches a migration point (M), it checks if there is a pending migration request. In case of such a request, all the relevant state information of that migration point is transferred to the operating system $^{(2)}$. Consequently, the OS will instantiate the same task on a different processing element. The new task instantiation will be initialized using the state information previously captured by the operating system $^{(3)}$. Finally, the task resumes execution at the corresponding migration point.

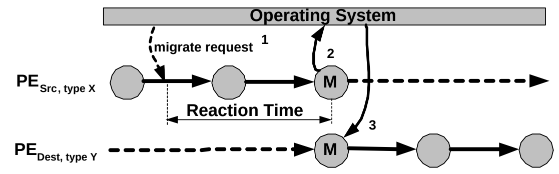

Figure 1. Task migration sequence.

One of the main issues in heterogeneous task migration is the overhead incurred by checking for a pending migration request during normal execution (i.e. when there is **no** pending migration request). Especially since a task requires frequent migration points in order to reduce the *reaction time*. The reaction time (Figure 1) is the time elapsed between selecting a task for migration and the selected task reaching the next migration point. In order to minimize the checking overhead during normal execution, further denoted as migration initiation, we propose a novel technique targeted at embedded systems, that uses the debug registers present on most modern PEs.

The rest of the paper is organized as follows. Section 2 briefly discusses the related work. Section 3 gives an overview of our proof-of-concept task migration initiation implementation. Section 4 presents our conclusions.

2. Assessment of Existing Techniques

Since the 1980s, several multicomputer task migration mechanisms have been developed. Two distinctive migration initiation approaches can be identified. The first approach is based on polling [3, 4]. It introduces initiation polling points into the execution code at the location of

* This work was partly funded by Xilinx Labs, Xilinx Inc. R&D group.

the migration point. The amount of poll points and their placement in the code is critical for performance: too many poll points introduce a large run-time overhead, while not enough poll points increase the reaction time. The amount of work required for the OS to enable the migration request can be as little as setting a global variable. The second approach is based on dynamic modification of code. In this case, the execution code is altered at run-time to introduce the migration-initiation code after receiving a migration request. The Tui System [2] stops the concerning task and places a breakpoint instruction at every migration point. Then the task continues until a breakpoint trap occurs. In order to avoid overwriting other instructions when inserting these breakpoint instructions, extra space needs to be reserved. This can be done using dummy instructions, which introduce some performance overhead during normal execution. Prashanth et al. [1] introduce a technique that detects the fragment of code containing the next migration point and places migration initiation instructions in a copy of that code. This technique does not require any placeholder instructions. The amount of work to enable a migration point (for the second approach) is quite large in contrast to the first approach, which prolongs the reaction time.

3. Hardware Supported Migration Initiation

Most contemporary microprocessors (PowerPC, ARM, i386, etc.) contain a set of debug registers. The PowerPC 405 [5], present in the Xilinx VirtexIIPro FPGA, contains four 32-bit Instruction Address Compare (IAC) registers. Whenever the program counter (PC) register reaches a value present in one of the activated IAC registers, an exception is generated.

However, this mechanism can be reused as a poll-free migration initiation technique that does not require any copying or insertion of code to enable migration points. The setup of our proof-of-concept implementation is illustrated by Figure 2. After starting the task (1), a migration handler is registered with the operating system (2). This handler will be responsible for collecting the logical task state after the task reaches a migration point. In addition, all migration point addresses (mp) are registered with the OS (3). Then, the task starts executing. In the absence of a migration request, there is no run-time overhead (4). The OS maintains the migration point addresses in a task-specific data structure. With every context switch to another task the OS updates the IAC registers. When the resource manager decides to migrate a task, it activates the IAC registers (i.e. simply setting some register flags) (5)(6). As soon as the task reaches the instruction on a migration point address (mp), a hardware interrupt is generated (7), which activates the migration signal handler (8). After gathering the logical task state, the task is ready for migration to the destination PE.

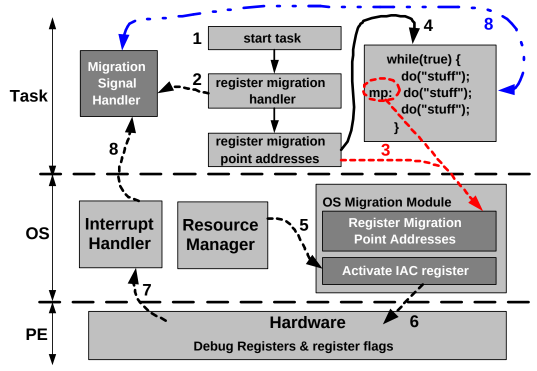

Figure 2. Migrating by using debug registers.

This technique not only incurs zero overhead during normal execution (just like the poll-free solutions), it also requires just a minimal amount of OS effort in order to issue a migration request (just like the polling solutions). A drawback of this technique is the limited amount of debug registers. This means that, currently, a task cannot have more migration points than the number of available debug registers. Eventually, this drawback can be overcome by selecting the right subset of migration point addresses during the OS migration request. Another drawback is the potential interference with debuggers and the lack of compiler support for setting migration points.

4. Conclusion

This papers details a novel task migration initiation technique for heterogeneous MP-SoC, based on the reuse of the debug registers present in most modern microprocessors. We show that our technique incurs no overhead during normal execution and minimizes the work of the operating system, which reduces the migration reaction time.

References

[1] Prashanth P. Bungale, Swaroop Sridhar, Vinay Krishnamurthy, ”An Approach to Heterogeneous Process State Capture/Recovery to Achieve Minimum Performance Overhead During Normal Execution”, Proc. IPDPS , April 2003.

[2] Peter Smith, Norman C. Hutchinson, ”Heterogeneous Process Migration: The Tui System”, Report TR-96-04, Univ. British Columbia, 1996.

[3] Adam J. Ferrari, Stephen J. Chapin, Andrew S. Grimshaw, ”Process Introspection: A Heterogeneous Checkpoint/Restart Mechanism Based on Automatic Code Modification”, Report CS-97-05, Univ. of Virginia, 1997.

[4] H. Jiang, V. Chaudary, ”Compile/run-time support for thread migration”, Proc. IPDPS, p58-66, April 2002.

[5] http://www.xilinx.com/bvdocs/userguides/ppc_ref_guide.pdf