Contract no. 248424
FP7 STREP Project

MADNESS

Methods for predictAble Design of heterogeNeous Embedded System with adaptivity and reliability Support

D6.6: Prototype integrating the system level design framework for adaptive MPSoC

Due Date of Deliverable	**31st December, 2012**
Completion Date of Deliverable	**31st December, 2012**
Start Date of Project	**1st January, 2010 - Duration 36 Months**
Lead partner for Deliverable	**UL**

Revision: v1.0

Project co-funded by the European Commission within the 7th Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	✓
PP	Restricted to other programme participants (including Commission Services)	
RE	Restricted to a group specified by the consortium (including Commission Services)	
CO	Confidential, only for members of the consortium (including Commission Services)	

Contents

1. Introduction

This document describes the prototype that was developed to integrate the outcome of WP5 and WP6 with the rest of the MADNESS framework. In particular, among all the techniques developed in WP5 and WP6, we choose the ones that suit better the considered application (H.264 decoder) and hardware platform features (Virtex-6 FPGA board).

The prototype can be logically divided into three components:

1. the hardware platform (see Section 2);
2. the middleware infrastructure, which allows dynamic run-time remapping of processes (Section 3);
3. the DSE techniques, which make remapping decisions at run-time, given a fault scenario (Section 4).

A complete version of the developed prototype, including hardware and software/middleware sources, is available online at **http://www.madnessproject.org/open-source/dl/adaptivity_prototype.tar.gz**.

2. Hardware platform

The prototype hardware platform is actually an instance of the MADNESS reference platform described in greater detail in Deliverables D3.1 and D6.1. The MADNESS reference platform is a distributed-memory tile-based template, in which tiles are interconnected through a Network-on-Chip (NoC). All the tiles in the prototype are homogeneous and comprise a MicroBlaze processor as CPU.

The communication network is built using an extended version of the the ×pipes-lite library of synthesizable components [1]. The topology can be completely arbitrary, since it includes a fabric of routers and links that can be almost entirely customized. Network access points are Network Interfaces (NI), that are in charge of constructing the packets on the basis of the communication transactions requested by the cores. NIs, placed at the interface between processing elements and the communication network, have been extended with support for message-passing communication model. For instance, a programmable message manager with DMA capabilities is integrated within the NI inside a module called Network Adapter.

As described in Deliverable D6.3, the platform template has been extended with newly developed hardware IPs that facilitate run-time management. For example, an interrupt on a certain tile can be triggered by sending a control message over the NoC. This is of great use in the implementation of a predictable process migration mechanism.

As depicted in the left part of Figure 2.1, the platform instance that was chosen for this prototype is a mesh of 2x3 tiles. In the first place, to choose the mesh size, we derived experimentally the size of the instruction and data memory required to execute at least 3 nodes of the test-case application (H.264 decoder), to provide an adequate level of flexibility to the remapping decisions. This memory requirement is 128 KB for each instruction and data memory of the tiles. Then, given this memory requirement and the on-board memory limitation of the Virtex-6 FPGA board, we derived the maximum number of tiles (6) which could fit into the FPGA.

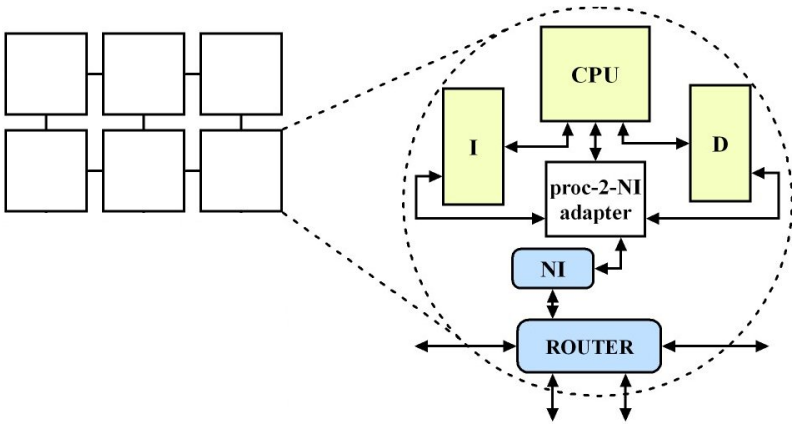

Figure 2.1: Prototype hardware platform topology (left) and internal tile structure (right).

3. Middleware layer

Each tile of the platform described in Section 2 is endowed with the software stack depicted in Figure 3.1. The *application level* resides at the top of the software stack. In MADNESS, applications are specified using the Polyhedral Process Networks (PPNs) model of computation.

At the bottom of the software stack, the *local operating system* provides basic functionalities such as process management (process creation/deletion, setting process priorities) and multitasking capabilities.

The *middleware level* of the software stack, highlighted in the left part of Figure 3.1, comprises the three main components listed below:

1. *PPN communication API:* it provides a set of primitives which allow the execution of applications modeled as PPNs on NoC-based MPSoC platforms. For further details, please refer to Deliverable D6.3.

2. *Process migration mechanism:* the migration mechanism described in detail in Deliverable D6.3 has been extended to emulate the behavior of the Task Migration Hardware; i.e., the steps which have to be performed on the faulty tile have been reduced to the minimum, such that they can be executed by a very simple piece of hardware external to the processor.

3. *Run-time manager:* it runs on a single tile and makes decisions, at run-time, concerning the adaptation of the system to changing resource availability. The remapping decisions are based on the DSE techniques described in Section 4.

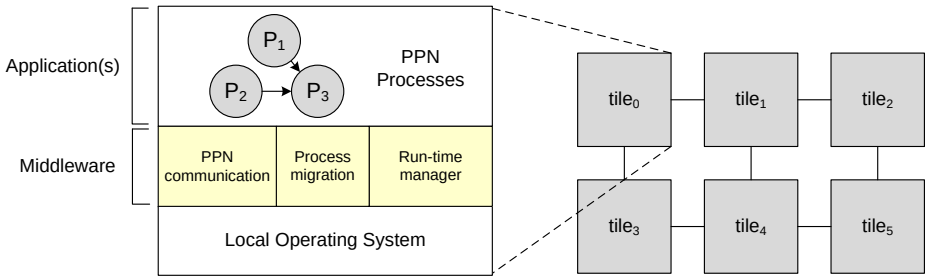

Figure 3.1: Software stack.

4. DSE-based run-time remapping policies

When one tile of the platform experiences a permanent fault, the Resource Manager (RM) has to make a remapping decision. In our prototype, the remapping decision is based on a set of alternative Pareto-optimal mappings, which are determined using DSE techniques. In the following two sections, we explain how this set of Pareto-optimal mappings is derived, and how the Resource Manager selects the next mapping among this set.

4.1 Generation of alternative application mappings

To explore the design space for optimum design points, within MADNESS we adopt a search engine that utilizes heuristic search techniques, such as multi-objective Genetic Algorithms (GAs). In particular, the SESAME framework has been extended in order to simulate the PPN application execution on a NoC communication infrastructure, according to the PPN communication API mentioned in Section 3.

In order to meet the requirement of our prototype, the design space search has to be driven to match the limitations of the actual system implementation. For instance, due to memory limitations, not every process can be mapped on every tile. A necessary condition for a process to be executed on a tile is that a *replica* of the code of that process has to be loaded in the instruction memory of the tile. For the prototype described in this document, we have chosen the H.264 decoder as a test application. Its topology is shown in Figure 4.1. The actual allocation of replicas of the H.264 processes is shown in Figure 4.2. The allocation of replicas restricts the mapping of a certain process only to a subset of tiles. For instance, process H_0 and H_1 can only be mapped to $tile_0$ because there are no replicas on the other tiles. By contrast, H_2 can be mapped on several tiles (0,1,3,4,5).

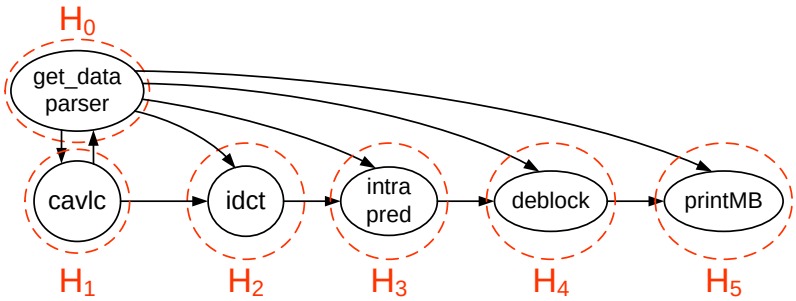

Figure 4.1: (Simplified) topology of the H.264 application.

4.2 Selection of application mapping at run-time

Once the set of alternative Pareto-optimal mappings has been derived, it can be stored in a simple data structure accessible by the Resource Manager. At run time, the Resource Manager is responsible to select the best remapping (from mapping M_i, before fault i, to mapping M_{i+1}, after fault i) when a tile experiences a permanent fault.

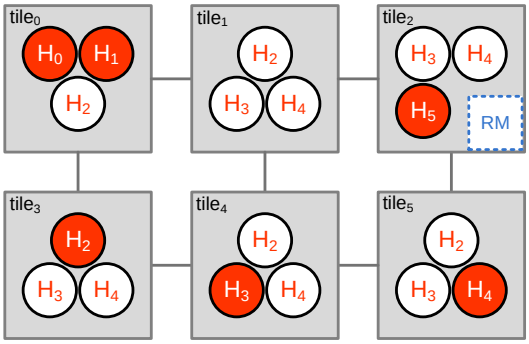

Figure 4.2: H.264 process replica allocation. The highlighted replicas correspond to the initial mapping.

The Resource Manager (RM in Figure 4.2) selects an appropriate mapping based on two criteria, which are listed below.

1. A mapping has to be feasible with the currently available platform resources. After a series of faults, only a subset of the alternative remappings will be feasible. In particular, only those mappings which do not involve faulty tiles will be feasible.

2. Among the subset of feasible mappings, the Resource Manager selects the mapping which has the best performance. If more mappings provide the same performance, the Resource Manager selects the one with the least *Hamming distance* from the previous mapping M_i. This choice is done in order to minimize the run-time cost of migration.

4.3 Example of process migration scenario

Given the initial mapping shown in Figure 4.2, this section describes an example of process migration scenario, which is depicted in Figure 4.3. In this scenario, $tile_3$ will be halted due to an emulated fault. The migration is performed in the following steps:

1. A *control thread* which runs on $tile_2$ waits until a certain amount of time has elapsed, then sends to $tile_3$ an interrupt-generating message, which emulates a permanent error.

2. The Task Migration Hardware on $tile_3$ is emulated by an interrupt handler; this handler is used to send the state of process H_2, which has to be migrated, to the Resource Manager (on $tile_2$).

3. The resource manager selects the remapping from the table provided by the DSE framework. In this example, process H_2 has to be moved from $tile_3$ to $tile_4$. The current state of H_2 is sent to $tile_4$.

4. H_2 restarts its execution on $tile_4$.

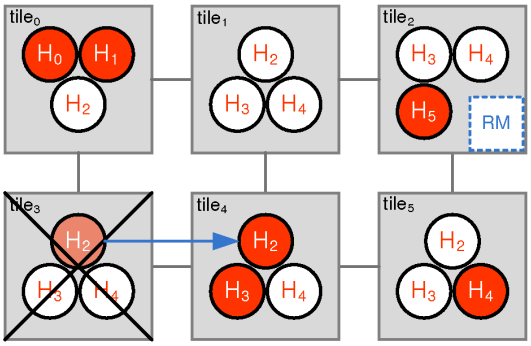

Figure 4.3: Example of process migration scenario on H.264.

Bibliography

[1] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini, "Xpipes: a Latency Insensitive Parameterized Network-on-Chip Architecture for Multi-Processor SoCs," in *Proc. of the 21st Int. Conf. on Computer Design*, ser. ICCD'03, Washington, DC, USA, 2003, pp. 536–.