New Dimensions for Multiprocessor Architectures: On Demand Heterogeneity, Infrastructure and Performance through Reconfigurability – The RAMPSoC Approach

Diana Göhringer\textsuperscript{1}, Michael Hübner\textsuperscript{2}, Thomas Perschke\textsuperscript{1}, Jürgen Becker\textsuperscript{2}

\textit{FGAN-FOM, Germany}\textsuperscript{1}
\textit{ITIV, Universität Karlsruhe (TH), Germany}\textsuperscript{2}
\{dgoehringer, perschke\}@fom.fgan.de\textsuperscript{1}
\{huebner, becker\}@itiv.uni-karlsruhe.de\textsuperscript{2}

Abstract

Multiprocessor hardware architectures enable to distribute tasks of an application to several microprocessors, in order to exploit parallelism for accelerating the performance of computation. Especially for the application domain of image data processing, where computation performance is a crucial factor to keep the real-time requirements, this approach is a promising solution for the assembly of high sophisticated algorithms e.g. for object tracking. Changing requirements and the necessary implementation of the tasks in terms of modified algorithms, precision and communication needs to be handled by software and hardware adaptation in state of the art architectures. Field Programmable Gate Arrays (FPGAs) enable to exploit the adaptation of hardware cores and the software running on embedded microprocessor cores on an integrated multiprocessor system.

Keywords: MPSoC, Reconfigurable Hardware, Image Processing

1. Introduction

The usage of Multi-Processor System-on-Chip (MPSoC) for accelerating performance intensive applications is an upcoming trend in actual chip technology. The approach of distributing tasks of an application to several processor devices is a well established method for increasing computational performance in computer science. The partitioning algorithms and methodologies are able to detect inherent parallelism in a dataflow-graph (DFG) and a mapping tool is able to distribute the tasks to a certain processing unit. The gap in this procedure is that the tasks can only be optimized to a certain extent to a given (multi) processor’s architecture. A certain granularity or requirement of a task cannot be further optimized. Exactly this is the problem whereof static actual multi-processor hardware architecture suffers from. Changing applications and requirements do not perform an optimized work balance of the processors and the communication infrastructure. A promising approach for bridging this gap is to exploit run-time reconfigurable hardware. The requirements of an application’s task can be offered by a flexible hardware which adapts in terms of computational performance and communication bandwidth. The top-down approach, in terms of application partitioning and task distribution, and bottom-up in terms of hardware adaptation through run-time reconfiguration leads to the “meet in the middle” solution. The paper is organized as follows: In section 2 and its subsections new dimensions for reconfigurable MPSoC-Systems that lead to the RAMPSoC approach are described. Section 3 describes the application scenario. The integration and results are presented in section 4. Finally the paper is closed by presenting the conclusions and future work in section 5.

2. New Dimensions for Reconfigurable MPSoC-Systems – The RAMPSoC Approach

Exploiting run-time reconfigurable hardware represents a promising solution to cope the disadvantages of the top-down designflow of state of the art MPSoC-Systems. Usage of run-time reconfigurable hardware offers a new degree of freedom, as now the in section 1 described top-down designflow, in terms of application partitioning and mapping, is extended by a bottom-up approach enabling hardware adaptation during design-time as well as during run-time. This leads to a “meet in the middle” designflow solution for RAMPSoC, which is illustrated in Figure 1. RAMPSoC uses the bottom-up approach during design-time to generate an initial RAMPSoC system that is optimized for the actual application requirements. Furthermore, the bottom-up designflow is used during run-time to adapt the hardware architecture on-demand to the time variant application requirements. Through this novel approach an optimized distribution of computing tasks onto the adaptive hardware can be ensured and with this, constraints for performance and area as well as for power consumption can be achieved more efficiently. These new degrees of freedom can be used beneficial to achieve high performance by taking advantage of the multiple forms of existent concurrency that is inherently included in most image processing applications. Forms of concurrency are e.g. data parallelism, task parallelism and instruction level parallelism. Especially image processing applications are
candidates including these kinds of parallelism. Task parallelism is exploited, by mapping different tasks to different processors or groups of processors. For example one processor group searches for one object in the input image, while the other group searches for a different object in the image. Data parallelism is very strong in image processing as an input image can be separated into independent tiles.

```
Application
Top-Down
RAMPSoC
Design time:
Tool supported
Partitioning
Bottom-Up
Run-Time:
HWsupported
Adaptivity
dyn. part.
Reconf.
```

**Figure 1. “Meet in the middle” Designflow approach for RAMPSoC**

These tiles can be mapped onto different processors. Each of these processors executes the same algorithm on its subset of data. Also instruction level parallelism lends itself very well to RAMPSoC. In [1], basic details of the proposed approach can be found.

### 2.1. Three Hierarchy-Levels of RAMPSoC

For classification reasons of the new degrees of freedom for MPSoCs, the RAMPSoC can be divided into three different levels of hierarchy, as shown in Figure 2. The top level of hierarchy is the MPSoC-level, followed by the communication-level and finally the bottom level of hierarchy is the processor-level. The run-time adaptation of the hardware as well as the software toolchain can be applied to the different levels of hierarchies, which will be explained more in detail in the following subsections.

#### 2.1.1. Processor-Level of Hierarchy

In the processor-level, each processor can be seen as a RISP (Reconfigurable Instruction Set Processor) [2]. RISPs are more flexible than ASIPs (Application Specific Instruction Set Processor), as their instruction set can be adapted on-demand. Also they are faster and more power-efficient than General Purpose Processors, because they consist of a processor with an attached reconfigurable block, in which the complex instructions are realised. Furthermore they are optimized during design-time as well as during run-time for the actual tasks by reconfiguring the reconfigurable block. Run-time adaptation in the processor-level of hierarchy is used to modify the instruction set of the processor or to exchange the contents of the data or the instruction memory.

#### 2.1.2. Communication-Level of Hierarchy

In the communication-level the task interconnection will be physically established using one or several connection structures such as NoC [3], Bus or Point-to-Point. Hereby it is important, that these structures support run-time adaptivity.

#### 2.1.3. MPSoC-Level of Hierarchy

The MPSoC-Level of hierarchy is the system level of the proposed architecture. This abstraction level allows for a coarse partitioning of the tasks within an application and consideration of quality of service parameters like e.g. performance requirements or power consumption restriction. The partitioning of the tasks to the different processors is done with a preliminary decision of which type of processor is to be chosen. Code compilation and the selection of accelerators will be done in the processor-level of hierarchy. Furthermore the task interconnection will be physically established in the communication-level of hierarchy but is prepared in the MPSoC-Level by the extraction of the communication dependencies from the task graph.

#### 2.2. I/O Virtualization and System Hardware Structure

The hardware architecture of the system at one point in time can be seen in Figure 3. RAMPSoC represents a modular, heterogeneous run-time adaptive MPSoC. The run-time adaptation is applied to the 3 different levels of hierarchies using the 2D dynamic partial reconfiguration feature of the Xilinx FPGAs [4] in order to:

- Exchange of the application code by reconfiguring the local memory of the processor. (Processor-Level)
- Exchange of the instruction set by reconfiguring the accelerator. (Processor-Level)
- Modify the communication infrastructure by adapting the connections between the processors or the routing algorithms. (Communication-Level)
- Exchange whole processors to reduce the power consumption, increase the performance. (MPSoC-Level)
The example RAMPSoC system structure in Figure 3 shows a heterogeneous MPSoC consisting of 1 Finite State Machine (FSM) combined with a hardware function and 3 processors that are connected with each other over a switch-based NoC. Two different types of processors are used. Some of them are directly connected to one or more accelerators to increase the computation performance of complex instructions by using instruction level parallelism. The FSM together with the Hardware Function can be seen as an alternative to a processor with an attached accelerator. The communication of the RAMPSoC with its environment, such as a camera or a monitor, is done over the Virtual-I/O component. This component belongs to the communication-level and supports task parallelism by forwarding the incoming data to the corresponding processors. Furthermore it supports data parallelism by splitting the incoming data into parts, which are then forwarded to the right processor. This feature exploitation is described in section 4 where an incoming image is divided into several overlapping tiles. Afterwards these tiles are forwarded to the respective processors. Furthermore the results of the micro-processors are collected and sent to the environment, which is a monitor in this example.

Figure 3. RAMPSoC system at one point in time

To provide these features, the Virtual-I/O component consists of several registers in which the values for splitting the input data and for combining the results are stored. These values can be updated during run-time by overriding the registers either directly or by using dynamic and partial reconfiguration. Furthermore this component can be exchanged completely by using dynamic and partial reconfiguration. This has to be done, for example if the number of processors within the system changes during run-time. To prevent the loss of incoming data, the data is stored in an input buffer.

2.3. Toolchain

For exploiting the advantages of such a heterogeneous run-time adaptive MPSoC and to make the porting of algorithms onto it easier, a well thought out toolchain is required. This toolchain is prospect of future research, but its basic principles will be outlined in this subsection. Also the toolchain follows the principles of the 3 different hierarchy levels. Starting on the MPSoC-Level, high performance computing algorithms, such as complex image processing algorithms, are often designed using tools such as Matlab (The MathWorks). Within these high level tools, C-code generators such as Matlab Realtime Workshop can be used for an automated C-code generation. For the generated C-code a task graph analysis has to be done in order to partition it to the different processors. The partition can be done in a first step by hand using pragmas and should be done in a second step automatically by using a partitioning tool. This task graph partitioning has impact on the communication-level as it defines the required communication infrastructures. In the last step the C-code fragments are mapped onto the different processor cores. Now in the processor-level these portions of C-code have to be partitioned into code that has to run in software and code that should be implemented in hardware. This partitioning is done by profiling the code and finding a trade-off for spatial and temporal partitioning as well as a trade-off for software and hardware partitioning (see also [5]). To transfer the dedicated functionality into hardware either existing IP-templates can be used, the required hardware code can be written by hand, which could be very time consuming depending on the complexity of the code, or existing C-to-HDL-code generators like e.g. ImpulseC can be used for automatic HDL-code generation. As the used processors within RAMPSoC are all state of the art microprocessors, they already come with a C-compiler. This compiler is then used to map the resulting C-code onto the processor.

3. Application Scenario

Usually tasks which require real-time image processing are object recognition and tracking from an incoming video stream with frame rates between 25 and 50 Hz. A Hot-Spot filter is an image processing algorithm, which searches inside an image for point like objects. It searches them by comparing the center pixel with its surrounding shells. Here 5 shells are used. As shown in equation (1), if the value of the center pixel $M$ is bigger than the minimum of the maxima of its surrounding shells $S1$ to $S5$, it is identified as a Hot-Spot. In case of a found Hot-Spot, the center pixel is assigned the difference between its original value and the minimum of the maxima of its surrounding shells.

$$M > \min \{ \max (S1), \max (S2), \max (S3), \max (S4), \max (S5) \} \quad (1)$$

4. System integration and Results

For the system integration an Alpha-Data ADM-XRC-4FX board with a Xilinx Virtex-4 FX100 was used. Three
MPSoC systems each using a different number of processors (1, 2, and 4) were implemented to execute the above described image processing algorithm. The system with 4 processors is shown in Figure 4. The PCI interface is used to transfer the image data to and from the FPGA board. For the processor cores the MicroBlaze ($\mu$Blaze, $\mu$B) [6] soft processor from Xilinx is used. In all systems an On-Chip Peripheral Bus (OPB) with two slave peripherals (OPB-Timer and UART) was added to one of the processors. The timer measures the execution time of the application, which is sent to the host PC over the UART interface.

The systems with 2 and 4 $\mu$Blazes use the Virtual-I/O component to exploit the data parallelism, which is provided by the above mentioned image processing algorithm. To exploit parallelism, the image is partitioned into overlapping tiles, which are sent to the different processors. For both algorithms the used input image has a size of 56x56 pixels. An efficient partition of the image is crucial for the achievable speedup. An inefficient partition can lead to the case that some processors are already in an idle status, while the others still have to calculate the results. Figure 5 shows a view of the used FPGA-Resources after Place and Route for the system with 4 $\mu$Blazes and the Virtual-I/O component. The area and timing results for all 3 MPSoC systems using a clock rate of 50 MHz are shown in Table 1.

A linear speedup very close to the expected values is measured. The small variation from the theoretically expected values is caused due to a suboptimal partitioning of the image data, because the image was partitioned on the coarse-grained basis of pixel rows. If the image partitioning would be more fine-grained, better results could be achieved. The impact of the Virtual-I/O component is negligible. Also it can be seen that to fulfill the real-time requirements of the application, using data from a video camera with a frame rate of 25 Hz, we need 4 $\mu$Blazes. Future investigation is to analyse implementations with fewer processors and to extend these processors with hardware accelerators.

**Table 1. Results for the Hot-Spot**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Hot-Spot (5 shells)</th>
</tr>
</thead>
<tbody>
<tr>
<td>System architecture</td>
<td>1 $\mu$B</td>
</tr>
<tr>
<td>Slices/BRAMs</td>
<td>2134/12</td>
</tr>
<tr>
<td>Exec. Time in ms</td>
<td>86,32</td>
</tr>
<tr>
<td>Speed up</td>
<td>1,00</td>
</tr>
</tbody>
</table>

5. Conclusions and Outlook

This paper presents the RAMPSoC approach which reveals new degrees of freedom like on-demand heterogeneity, infrastructure and performance through the “meet in the middle” design-space and -flow. It is obvious, that the effort for designing multi-purpose MPSoCs cannot come to a solution without facing runtime adaptivity which is provided by reconfigurable hardware. The RAMPSoC approach implies exactly this feature and additionally the benefit of a well established program model. Furthermore, clearly classified levels of hierarchy allow for the development of an efficient toolchain which will enable the acceptance of this novel architecture as well as from industry as from academics.

6. References


