Heterogeneity Impact on MPSoC Platforms Performance

Nikolina Frid, Danko Ivošević and Vlado Sruk
University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia
nikolina.frid@fer.hr, danko.ivosevic@fer.hr, vlado.sruk@fer.hr

Abstract – Embedded systems have become an integral part of High Performance Computing (HPC) due to their appealing energy and resource consumption characteristics. Required performance goals can be achieved only by deploying the application on a heterogeneous platform. The established approach of designing a custom made FPGA architecture platform targeting particular application is challenged by novel multiprocessor heterogeneous platforms built using existing off-the-shelf embedded CPUs. The challenge is to exploit all the available parallelism and heterogeneity to design a time and cost efficient, but also a reusable solution which will meet the performance goals. In this paper the heterogeneity of parallel SoC system is evaluated through different processor cores and memory configurations usage examination. The design space exploration undertaken relies on several hypotheses concerning design concepts relations. Since system operating frequency is crucial, but not the only design performance success parameter, the importance of operating processor data path is emphasized as it conducts the application mapping approach. We show that, depending on the different heterogeneous elements configurations used within some multicore SoC solution, the suitability of particular processor cores usage varies on application type, but with achieved performance comparable to application-specific custom generated hardware.

Keywords: Embedded Computer Systems, Heterogeneous Computing, FPGA, High Level Synthesis, Design Space Exploration, Software partitioning and mapping,

I. INTRODUCTION

Broad and versatile application of embedded computer systems has marked the last decade of their usage. Due to the wide variety of applications these systems are used for, their design can pose many challenges. The design requirements for embedded systems involve meeting desired performance goals while observing numerous other constraints ranging from development cost and time, energy efficiency, final product cost to reliability concerns and more [1][2]. Choosing the optimal architecture platform for a particular application often proves to be the biggest design issue. In order to satisfy the above enlisted design requirements, often a heterogeneous multiprocessor platform must be used. Still the design will not benefit from the platform heterogeneity unless the software application is properly tuned to exploit available hardware potential. Common approach is to choose a certain platform based on one or more of the following criteria: price, availability, earlier experience or legacy status, and then manually optimize the application software for the target platform. The alternative is the Design Space Exploration (DSE), a process to find out near-optimal system architectures for a given application, considering the design constraints and objectives. It involves task partitioning and mapping, architecture and processing element selection, and performance estimation before a hardware prototype is built [3]. The process starts with application specified using a model of computation, description of architecture instance and a set of constraints for the overall system. Those inputs are fed to mapping engine which performs partitioning and mapping of software to hardware. Finally, a performance analysis is conducted to determine whether the output design meets given constraints. During DSE process, the platform can be built either entirely out-of-the-shelf components (embedded CPUs, RAM memories etc.) or custom made FPGA components can be used to build the part or even an entire architecture platform targeting particular application. High level synthesis (HLS) [4] denotes a set of methodologies that generate custom hardware units from high level specification in automated manner. The idea is to help designers to build and verify custom piece of hardware in short time by writing algorithmic specification. Usually, the specification language is C or some of its derivate such as C++ or SystemC, and the output is synthesizable RTL level (VHDL, Verilog) circuit description. FPGA is the target implementation device as the most popular prototyping platform with good software support for logic synthesis process. It is suitable for generating dedicated high performance components that will be used as coprocessors for design functionalities selected for acceleration, and during last decade a number of academic and commercial tools appeared in design community and on the market.

II. RELATED WORK

Lately, many research groups have tried to optimize and automate parts or the entire DSE process. The main two points of focus have been: (1) code analysis and partitioning and (2) task to processor mapping. Early work in this area embraced the approach to take the application, divide it into tasks on a very high level and try to come to best near-optimal (NP-hard problems) task-to-processor mapping scheme in a reasonable
amount of time [5][6]. The scheduling algorithms used were modified versions of integer linear programming. More recent work has focused on optimizing both the application and the mapping scheme, while automating the most part of this process. The most successful at the time were the Daedalus tool [8], developed at University of Amsterdam and MAPS tool [7] developed at the research group at RWTH Aachen University. While MAPS features a more advanced compiler and focuses on adapting the application to the given platform, Daedalus provides better support for platform customization and is able to produce custom cores in FPGA, which MAPS is not. However, both tools rely on virtual platforms for performance estimation, which are not cycle-accurate by definition and do not reflect the implicit processor data path features. They are also not able to work with a loosely defined platform definition i.e. choose a certain number from a pool of components.

III. PROBLEM DEFINITION & HYPOTHESIS
As stated above, DSE application-to-processor mapping process currently relies on virtual platform simulations for performance estimation or manual intervention of the system designer based on professional experience. However, virtual platform simulations do not take inherent features of a certain platform, like inner data path architecture, into account and thus do not correctly estimate the time for a certain task to execute on an element of a platform. Consequently, any approach relying on this type of performance estimation can potentially fail to exploit the full potential of heterogeneous platforms.

The goal of this research is to explore the extent of impact of platform heterogeneity on mapping efficiency regarding performance goals. The rest of this paper will try to give an answer to whether it is possible to partition the application and map it to heterogeneous platform to achieve performance results comparable to a custom hardware platform built using fully automated HLS approach. The custom hardware platform has been chosen as a reference point because currently it is the most common architecture platform type used for critical tasks that require significant speedup.

At the beginning of this research three hypothesis have been established.

**Hypothesis 1:** Inherent features of a certain platform architecture significantly affect the performance of execution of specific types of operations. The time needed for a processor to execute a certain operation does not only depend on the operating frequency, but also on the inner data path architecture, which can indeed have a major impact on the execution duration. Consequently, the suitability of two processors with a somewhat similar architecture (e.g. RISC type) can be proved between different types of operations, assuming the same external memory environment.

**Hypothesis 2:** In order to be able to determine which processor is more suitable for a certain task (in broader sense then operation) it is enough to examine the performance of “basic tests” i.e. execution of operations typical for a certain task (e.g multiplication and trigonometry vs. bitwise operations) in a given memory environment.

**Hypothesis 3:** By examining the results of processor performance for “basic tests” in a given memory environment and applying efficient task-level parallelization of the application, it is possible to achieve an overall performance comparable to custom built hardware platform using HLS method. It is assumed that basic tests were chosen to accurately represent the prevailing types of operations in each task in the application of interest.

In the rest of this paper the experiments conducted in order to prove these three hypothesis are described.

IV. EXPERIMENTAL SETUP & RESULTS
In first part of this section the evaluation platform and test bench are described in detail. Later the results of conducted experiments are presented and discussed.

**A. Evaluation platform**
All experiments were conducted on the ZC706 evaluation board which provides a hardware environment for developing and evaluating designs targeting the Zynq®-7000 XC7Z045-2FFG900C AP SoC. The ZC706 evaluation board provides features common to many embedded processing systems, including DDR3 SODIMM and component memory (1GB), a four-lane PCI Express® interface, an Ethernet PHY, general purpose I/O, and two UART interfaces.

Two types of RISC processors were used:

1. **The ARM Cortex-A9 processor**, a popular general purpose choice for low-power or thermally constrained, cost-sensitive devices. The processor is a mature option having been introduced in 2008, and remains a very popular choice in smartphones, digital TV, and both consumer and enterprise applications enabling the Internet of Things. Zynq system integrates two ARM cores.

2. **MicroBlaze™**, a 32-bit RISC Harvard architecture soft processor core with highly flexible architecture, plus a rich instruction set optimized for embedded applications, delivers the exact processing system needed at the lowest system cost possible. This industry-leader in FPGA-based soft processors features advanced architecture options like AXI or PLB interface, Memory Management Unit (MMU), instruction and data-side cache, configurable pipeline depth, Floating-Point unit (FPU), and much more.
B. Test bench

According to the previously stated hypothesis three different test cases have been prepared.

Two test cases were adopted from MiBench [9] and represent “basic” test cases which aim to determine the behaviour of each type of processor in performing different types of computation.

1. CubicTB test solves the third-degree polynomial equation, and serves to measure performance in performing typical mathematical operations (addition, multiplication, basic trigonometry).

2. SqrtTB finds the square root of an integer with all the calculations being the base-two analogue of the square root algorithm learned in grammar school. Bit-wise operations are the prevailing type in this test case with absolutely no multiplications or divisions.

The above two test cases have been executed on twelve different architecture configurations. A single core of the ARM processor was used in the following configuration: operating frequency @ 667 MHz, 32 KB L1 cache and 512 KB L2 cache with two different memory mapping schemes:

- both instructions and data stored in local DDR3 SDRAM memory operating at 533 MHz (local),
- instructions stored in local memory and data was stored in shared DDR3 SDRAM operating at 200 MHz (mixed).

MicroBlaze processor was used in three different architecture configurations: (1) 3-stage pipeline @ 100 MHz, (2) 5-stage pipeline with hardware multiplier and barrel shifter @ 100 MHz, and (3) 5-stage pipeline with hardware multiplier and barrel shifter @ 200 MHz. The 200 MHz clock frequency is the maximum achievable frequency on PL side of Zynq system and thus is the reason why both shared DDR and MicroBlaze are not able to operate at higher speed. As was the case for the ARM processor, all three configurations of MicroBlaze have been tested with three different memory mapping schemes:

- both the instructions and data stored in local BRAM memory (local),
- instructions stored in local BRAM memory, and data in shared DDR memory operating at 200 MHz (mixed),
- both instructions and data stored in shared DDR memory operating at 200 MHz (shared).

The performance results are reported in TABLE I. and visually compared in Figure 1 and 2.

The results clearly show that the ARM processor greatly outperforms MicroBlaze for CubicTB in all memory settings. On the other hand, SqrtTB results show that

<table>
<thead>
<tr>
<th>Target processor</th>
<th>Memory configuration</th>
<th>Execution time [s]</th>
<th>CubicTB</th>
<th>SqrtTB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM 1</td>
<td>Local</td>
<td>4.74E-06</td>
<td>1.69E-06</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mixed</td>
<td>6.15E-05</td>
<td>2.58E-04</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shared</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>MB100 2</td>
<td>Local</td>
<td>2.21E-03</td>
<td>2.74E-05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mixed</td>
<td>5.20E-03</td>
<td>1.67E-04</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shared</td>
<td>7.94E-02</td>
<td>8.02E-04</td>
<td></td>
</tr>
<tr>
<td>MB100+ 3</td>
<td>Local</td>
<td>5.77E-04</td>
<td>1.54E-05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mixed</td>
<td>2.83E-03</td>
<td>1.61E-04</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shared</td>
<td>1.85E-02</td>
<td>4.08E-04</td>
<td></td>
</tr>
<tr>
<td>MB200+ 4</td>
<td>Local</td>
<td>2.56E-04</td>
<td>8.74E-06</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mixed</td>
<td>1.20E-03</td>
<td>9.27E-05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shared</td>
<td>1.06E-02</td>
<td>2.31E-04</td>
<td></td>
</tr>
</tbody>
</table>

TABLE I. CubicTB and SqrtTB Execution Times

| 1 | ARM® Cortex™-A9 CPU at 667 MHz |
| 2 | XILINX® MicroBlaze™ CPU at 100 MHz |
| 3 | XILINX® MicroBlaze™ CPU at 100 MHz with HW multiplier and barrel shifter |
| 4 | XILINX® MicroBlaze™ CPU at 200 MHz with HW multiplier and barrel shifter |

© 2015 MIPRO
MicroBlaze delivers much better performance than ARM for bitwise operations when accessing shared memory is required. Additionally, modifying MicroBlaze by adding two more stages in pipeline, hardware multiplier and a barrel shifter delivers up to four time better performance at the same clock speed.

As for the memory configuration, it can be clearly stated that the best performance is always achieved when everything is stored locally. However, this is often not feasible, especially in multicore systems when processor need to exchange data and thus the mixed memory configuration results should be considered in further discussion.

At this point it can be deduced that ARM would be a better choice for tasks consisting of mathematically more complex operations, while MicroBlaze would be a much better choice for simple and bitwise operations.

For the third test case JPEG [10] compression algorithm was chosen since it consists of several tasks with different prevailing types of operation which should be a good test of previously deduced assumption about processor affinity.

The JPEG compression algorithm has 6 consequential steps with YUV 3-component image model given as input. After the initial extraction of 8x8 pixel block from the original image, every pixel is shifted by -128 so that the whole range of DCT is used. The shift is followed by a DCT transformation which broadens the pixel data range to 12 bits, and scales back to 8 bits per pixel when quantization is performed. Next, data is arranged in a zig-zag manner so that the coefficients with higher value come first which is important for the efficiency of the following step – the Huffman entropy coding. Finally, coded data is formatted and written to a .jpeg file according to Standard. The JPEG compression flow is shown in Figure 3.

In the original standard DCT requires 64x64 passes through the pixel block and its core includes 3 multiplications per each step thus taking up more time than all other parts of the JPEG compression together. Through exploration of the specific properties of DCT, special algorithms were devised that help achieve significant speedup. In this paper the AAN algorithm [11] for scaled 1D DCT (8 points) was chosen for implementation and testing. Single pass of this algorithm requires 5 multiplication, 29 additions and 16 2nd complements. Two passes are needed for transformation of one block of pixels. Since it is completely obvious that MB is no match for ARM for original DCT, this algorithm was chosen as a more interesting alternative which could prove to be suitable enough for MicroBlaze to be able to compete in performance with ARM.

From the previous twelve architecture configurations only three that have previously shown best performance results were chosen:

1. ARM with local memory access,
2. ARM with mixed memory configuration,
3. MicroBlaze at 200 MHz with 5-stage pipeline, hardware multiplier and barrel shifter and mixed memory configuration.

It must be noted that, as previously stated, the first configuration where ARM communicates only with local memory is not a viable solution for any multicore system but it has been put here for comparison purposes. The obtained performance results for a picture size of 40 blocks are presented in TABLE II. As expected from the previous test cases, MicroBlaze outperformed the ARM processor in mixed memory mode in shift, Huffman coding and create_image tasks in which simple and bitwise operations prevailed. These results give strong evidence in favour of the first and the second hypothesis.

**C. Case Study – JPEG performance optimization**

Each of the previous test cases was executed on a single processor in order to obtain performance results for each task and determine the affinity of each type of processor to a certain type of tasks. The ultimate goal of these research is to show how to most efficiently use available platform resources based on processor affinity to certain tasks in order to achieve performance comparable to

<table>
<thead>
<tr>
<th>Target configuration</th>
<th>Create_blocks</th>
<th>Shift</th>
<th>DCT</th>
<th>Zig-zag</th>
<th>Huffman coding</th>
<th>Create_image</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM local</td>
<td>3.42E-04</td>
<td>2.72E-04</td>
<td>1.85E-03</td>
<td>2.13E-04</td>
<td>4.31E-03</td>
<td>1.28E-04</td>
<td>7.11E-03</td>
</tr>
<tr>
<td>ARM mixed</td>
<td>2.12E-02</td>
<td>1.90E-02</td>
<td>9.28E-02</td>
<td>1.91E-02</td>
<td>7.55E-01</td>
<td>3.26E-02</td>
<td>9.40E-01</td>
</tr>
<tr>
<td>MB200+ mixed</td>
<td>2.02E-02</td>
<td>1.85E-02</td>
<td>9.41E-02</td>
<td>1.85E-02</td>
<td>6.70E-02</td>
<td>3.25E-03</td>
<td>2.22E-01</td>
</tr>
</tbody>
</table>
custom made FPGA architecture platform targeting this particular application - JPEG.

For this purpose, a custom made architecture, which would serve as a reference point, has been designed on Zynq platform. It consists of a single core ARM processor and custom made hardware designed using HLS method with the source code previously used and implemented in FPGA. The ARM processor performs the first and the last task in JPEG flow (receiving/sending data from/to memory and pipelining it to FPGA) while the other tasks are completely executed in hardware. In order to ensure equal testing environment, mixed mode memory was used. Three levels of optimization of DCT function were applied and resulted in three solutions: memory was used. Three levels of optimization of DCT order to ensure equal testing environment, mixed mode memory was used. Three levels of optimization of DCT function were applied and resulted in three solutions: three.

The results are presented in TABLE III. and, as expected, the third solution achieves best performance results. After obtaining performance results from a custom made architecture, the next step was to devise an application partitioning and mapping scheme on the same Zynq platform, which would deliver results comparable to the previous ones. Zynq platform provides two ARM A9 cores with the surrounding programmable logic which can be used to instantiate a number of soft-core MicroBlaze processors. In this research the maximum possible number of soft processors was targeted. Due to routing and timing limitations the maximum possible number of MicroBlaze instances was ten. By observing the timings obtained for each type of processors in previous experiments for a mixed memory environment, using a combination of LPP[13] and LPT[12] mapping and scheduling heuristics for the same picture size as in the previous cases the following average execution time has been achieved: 9.18E-02. It is important to point out that it is only 1.7 time slower than the best time achieved on custom hardware platform and provides strong arguments in favour of the third hypothesis.

Resource utilization for multicore system is compared with resource utilization in custom hardware system in TABLE IV. The percentage of utilized resources is rather similar, and while custom hardware platform uses more DSP resources, the multicore platform uses more BRAM resources. However, it is important to mention that while custom hardware platform can be used only for the application for which it was designed, while a multicore platform is much more versatile and same configuration (with a different mapping scheme) can be used for a various set of applications thus improving the reusability of the solution.

V. DISCUSSION & FUTURE WORK

The results presented in the previous chapter provide strong evidence in favour of all three hypothesis established at the beginning of this research. The key concept of this approach is the possibility to gain insight into platform architecture specifics by using a basic set of tests and use that knowledge to develop an efficient mapping scheme for a more complex application. What is more, this approach, unlike expert knowledge, can be more easily automatized and integrated into a mapping tool targeting heterogeneous embedded multiprocessor platforms. It is important to note that the custom made hardware cores are often irreplaceable for accelerating parts of the application, especially in hard real-time systems, and in the future research devising a heuristic which will determine when a part of the application must be “moved” to hardware should be considered. Certainly, many more types of architectures and applications need to be studied in order to develop a viable mapping heuristic. Finally, much more emphasis needs to be put on reliability and energy consumption parameters to be able to provide solutions which will fully keep the pace with current demands for embedded computer systems.

TABLE III. JPEG ON CUSTOM HARDWARE - EXECUTION TIMES

<table>
<thead>
<tr>
<th>Target configuration</th>
<th>Create_blocks*</th>
<th>Shift</th>
<th>DCT</th>
<th>Zig-zag</th>
<th>Huffman coding</th>
<th>Create_image*</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>solution1</td>
<td>2.12E-02</td>
<td>5.08E-05</td>
<td>1.73E-03</td>
<td>2.16E-05</td>
<td>2.17E-04</td>
<td>3.26E-02</td>
<td>5.59E-02</td>
</tr>
<tr>
<td>solution2</td>
<td>2.12E-02</td>
<td>5.08E-05</td>
<td>1.62E-03</td>
<td>2.16E-05</td>
<td>2.17E-04</td>
<td>3.26E-02</td>
<td>5.57E-02</td>
</tr>
<tr>
<td>solution3</td>
<td>2.12E-02</td>
<td>5.08E-05</td>
<td>2.32E-05</td>
<td>2.16E-05</td>
<td>2.17E-04</td>
<td>3.26E-02</td>
<td>5.41E-02</td>
</tr>
</tbody>
</table>

*These tasks were executed on ARM

TABLE IV. RESOURCE UTILIZATION

<table>
<thead>
<tr>
<th>Target configuration</th>
<th>Resource utilization [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LUT</td>
</tr>
<tr>
<td>config1</td>
<td>16</td>
</tr>
<tr>
<td>config2</td>
<td>20</td>
</tr>
</tbody>
</table>

* multicore platform
** custom hardware platform
REFERENCES


