ABSTRACT
Heterogeneous Multi-Processor Systems-on-Chip (MPSoC) exhibit increased design complexity due to numerous architectural parameters and hardware/software partitioning schemes. Automated Design Space Exploration (DSE) becomes an essential design procedure to discover optimized solutions in a reasonable time. For high-quality DSE, the accurate solution evaluation is a strong requirement. To this direction, High-Level Synthesis (HLS) can be used for the characterization of the design solutions. In this paper, we propose (a) a platform design methodology that exploits simulation-induced slacks generated by avoiding simulation re-initializations and exploits the gained time for HLS, and (b) a DSE tool-flow which takes into account multiple HW/SW partitioning schemes and intelligently schedules system evaluations. Experimental results show that the proposed methodology achieves 17% simulation improvements together with 77% higher accuracy, in comparison to a typical exploration approach.

Categories and Subject Descriptors
B.6.3 [Logic Design]: Design Aids—Automatic synthesis, Simulation; C.4 [Performance of Systems]: Measurement techniques, Modeling techniques

General Terms
Design, Performance, Experimentation

1. INTRODUCTION
The ubiquity of embedded systems with different requirements and specifications has resulted to an increased complexity, so that the designer has to investigate multiple design parameters. In addition, with the advent of heterogeneous Multi-Processor Systems on Chip (MPSoC) incorporating processing elements of different types (CPUs, ASICs, etc.), the designer has also to make decisions about hardware/software partitioning (HW/SW) partitioning, including the architectural parameters of the hardware components (e.g. CPU number, memory hierarchy, parallel processing elements for each HW kernel, etc), as well as the mapping of computationally intensive kernels onto hardware. The latter task is Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

PARMA-DITAM ’14, January 20 2014, Vienna, Austria
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2607-0/14/01...$15.00.
http://dx.doi.org/10.1145/2556863.2556864
mostly known as HW/SW codeign [14] and its importance relies on the fact that the hardware kernels will increase the application’s throughput, while in parallel the CPU(s) can execute other SW parts. This however leads to many different partially partitioned designs, thus to even higher design complexity.

To cope with the huge number of the different solutions, the use of Design Space Exploration (DSE) is considered an issue of great importance [8]. During exploration, a representation of the system under test, called Virtual Platform (VP), is simulated/evaluated using real-life datasets. For VP modelling, we utilize SystemC [4], a C++ extension which introduces hardware description language features. TLM2.0 extends SystemC and enables the transaction-level communication modelling among the VP components. The familiarity of both SystemC and TLM2.0 relies on the fact that, in contrast to C-based instruction-set simulators, they enable the designer to model any platform, with any hardware component (e.g. CPUs, custom accelerators etc) and most importantly in multiple abstraction levels. Finally, SystemC/TLM2.0 models (even with some constraints) are synthesizable by most modern High-Level Synthesis tools. As TLM2.0 itself does not provide particular CPU or memory models, VP usually acquires such components from a TLM-based library [3].

There are two conflicting requirements when deploying a VP: (a) metrics accuracy; and (b) simulation speed. Concerning accuracy, the use of cycle-accurate CPU models provides high-accuracy simulation [11] of the VP software. Additionally, for the annotation of VP, i.e. area-delay-power characterization of the VP components, High-Level Synthesis (HLS) [5] leads to increased annotation accuracy. The manual annotation of VP (as an alternative to HLS) is much slower, error-prone and does not address to optimizations performed by commercial HLS tools.

Concerning speed, however, both HLS and cycle-accurate simulation are two very time-consuming tasks. The HLS time depends on both the design size and the designer commands, while the simulation time slows down especially due to non-productive simulation phases, such as: (a) initialization of the platform components and the software, including the operating system; (b) Dataset parts which are out of scope; (c) VP warming-up, where the simulation might have to ignore a part of the dataset (e.g. a video frame) in order to obtain representative metrics, e.g. due to numerous cache misses occurring to that part of the dataset.

In this paper, we propose a DSE framework for Virtual Platforms which (a) bypasses the non-productive simulation parts, as much as possible, and (b) maximizes the simulation gains obtained by the simulation framework, while additionally considering HW/SW co-design application par-

1Manual annotation refers to the manual (i.e. without EDA tool) estimation of the area, power and latency metrics for a SystemC module, and their insertion into specific points of the source code.
tioning decisions. The main contribution of this paper, differentiated from our previous work [13], is the usage of an industrial-strength HLS tool, by exploiting the simulation-induced time improvements.

With the proposed methodology, a simulation speedup reaching a maximum of 17% is achieved. This speedup is invested in DSE accuracy by using HLS. In particular, HLS gives on average more than 77% higher annotation accuracy, in comparison to manually-annotated models.

The paper is organized as follows: Section 2 summarizes the work related to this paper. Section 3 gives a description of the proposed methodology. Section 4 provides a set of experimental results showing the effectiveness of this approach. Finally, section 5 ends up with the respective conclusions.

2. RELATED WORK

In order to minimize the non-productive simulation parts, our previous work [13] introduces a process-based reconfigurable SystemC module (PRM), which separates a SystemC model into two parts, the static and the dynamic one, each of which runs as different Unix process. This allows the bypassing of non-productive simulation phases. In this paper, we utilize the concept of PRM for simulation speedup, in order to leverage the use of HLS for SystemC modules annotation. Typically, HLS is used to the later design stages, in order to produce RTL [11]. However, HLS is also a very good approach for annotating virtual platforms in earlier design stages with realistic data, as it maximizes the annotation accuracy towards latency, area and power. Typical instance is SystemCoDesigner [9], where the result of HLS is also used for extracting the required design metrics for design space exploration. SystemCoDesigner has many similarities with the proposed approach. However, it does not provide a transparent way for supporting heterogeneous platforms, while DSE is not based on simulation evaluation.

Alternative ways for annotating a custom SystemC model involve the analysis and characterization of the primitive processing cores (represented as modules and/or threads of a module). A typical approach, followed in [10], is the separation of the source code into basic blocks, each of which matches to a delay. The total delay is calculated during execution. Similarly, other metrics can be estimated. Although such alternatives are acceptable in early design stages, they cannot be applied in a refined platform, as they do not address to optimization issues which are covered in HLS.

Finally, numerous HW/SW partitioning techniques have been proposed (e.g. [7]). However, only few of them take into consideration architectural parameters for each hardware-mapped kernel and for the entire system, while in most cases the integration with any simulator is not addressed.

3. PROPOSED METHODOLOGY

The reference VP for this paper is developed with the use of SoCLib\(^2\) [3] and represents an MPSoC consisting of \(N\) homogeneous MIPS32 CPUs and \(M\) heterogeneous DSP accelerators. Each accelerator implements a different computationally intensive kernel of the running algorithm.

In order to avoid the platform restarting, a simplified version of Process-based Reconfigurable SystemC Module (PRM) is deployed. According to Figure 1, this version of PRM comprises two Unix processes, for the static and the dynamic part respectively. In addition, a third process, called Simulation Control Server (SCS), implements the interface between the DSE engine and the VP. With this division the architectural parameters of VP are categorized to the Major and the Minor ones, which are related to the static and the dynamic part respectively. Similarly, a major modification is the change to at least one major parameter, while a minor modification involves only minor parameters. In this paper, the static part includes the set of CPUs, the memory hierarchy and the auxiliary peripherals (e.g. debuggers, terminals etc), while the dynamic part includes all the DSP accelerators. The modification type is chosen by SCS: Upon a major modification, the whole model is stopped, recompiled and restarted. In case of a minor modification, SCS (a) pauses the simulation, (b) stops and recompiles only the dynamic part, (c) restarts only the respective process of the dynamic part and (d) notifies the model to continue from the point it was paused.

The dynamic part is represented to the remaining system by the PRM wrapper, a SystemC module which performs the following tasks: (a) it forwards port data from the static to the dynamic part and vice versa, through a set of shared memory segments, (b) it detects the timing of the dynamic part, in order to forward it to the simulation time of the remaining model, (c) it safely "pauses" the simulation, upon receiving the "pause" command, by checking for a synchronization signal which notifies for a new result, avoiding possible data loss, and (d) it initializes the new dynamic part, when a "continue" command is received, without forwarding port data or model timing to the static part, so that the timing accuracy of the platform is not affected.

The methodology proposed in this paper is divided into two distinct parts, presented below:

1. Preparation Phase (Figure 2a): The preparation phase starts with an ANSI-C application profiling, so that the computationally intensive kernels are detected. Then, the application is fully partitioned to the SW and the HW part, while the middleware for the communication between SW and HW is inserted. Although the software is fully partitioned, partial partitioning is also supported through disabling some DSP cores. For such a case, the middleware includes control code which checks if a HW kernel is enabled. For the overall generation of VP, the appropriate SoCLib components (CPUs, cache models, etc) are instantiated into the VP source file. In addition, each DSP core is implemented as a SystemC module, which includes the ANSI C code of the respective kernel, methods for calling the HLS tool, as well as I/O ports, through which a TLM interface establishes the communication with the remaining system.

From the computationally intensive kernels found, the designer can decide about the different combinations of the kernels mapped onto hardware, denoted thereof as kernel combinations. In general for \(M\) kernels given, there will be \(2^M - 1\) combinations. Every single kernel, according to

\[ \text{Figure 1: The PRM simplified structure} \]
the profiling stage, matches to a significant part of the algorithm execution. In case that a single design space is formed by including all kernel combinations, there is the potential for exploring configurations in a biased manner, according to the local maximum or minimum found by the optimizer. This may lead to an unbalanced evaluation of the solution space, e.g. the optimizer is biasing evaluations for the most computationally intensive kernel. To avoid this undesired behaviour, after extracting the architectural parameters and their value ranges, separate design spaces are created, one for each kernel combination. At the end, the points of the individual design spaces are combined in order to extract a single system-wide pareto curve.

2. **DSE core flow (Figure 2b):** The proposed DSE tool-flow (Figure 2b) is based on Multicube [2] [12] and uses the NSGAII genetic optimizer [6]. Firstly, Multicube passes a XML representing the current point to the SCS through a simple client tool. SCS extracts the platform configuration from the XML input. In case of a major modification, SCS invokes the compilation of both the software and the VP. If a minor modification occurs, SCS recompiles only the dynamic part. The software compilation produces an executable, while the VP compilation produces binaries for the static and/or the dynamic part. Then, SCS starts up the static and/or the dynamic part and finally waits for the simulation metrics, which are forwarded to Multicube. In order to maximize the simulation time gains when using PRM, the solutions to be evaluated in each generation of NSGAII are properly re-ordered. In fact, the solutions are categorized in groups, each of which matches to specific major parameter values. Hence, the solutions of a single group have differences only to minor parameters. As the minor parameters match to the dynamic part of PRM, the transition to a solution of the same group does not impose the platform re-initialization.

During simulation, the DSP accelerators are characterized by calling HLS. As long as HLS is a slow task, a metrics cache mechanism is deployed. Hence, if a configuration has already passed through HLS, the cached metrics are used. Otherwise, HLS runs and its metrics are stored into metrics cache. In this paper, HLS is performed by CatapultC University Version of Calypto [1] in order to extract area and latency. Given that accurate power characterization of the HW kernels requires low level (post-RTL) simulation traces, we estimate power by translating the area as a number of NAND gates and analytically calculating the leakage and the dynamic energy per cycle. When the DSP core is idle, the total power is equal to leakage, otherwise dynamic power is also taken into account. Given the accelerator’s clock frequency, the total power is calculated during the actual VP simulation.

4. **EXPERIMENTAL RESULTS**

The experimental results of this section are obtained with the usage of two fully established multimedia algorithms: Motion Compensation of H264 decoder (video) and MPEG Layer-3 (MP3) decoder (audio). Motion Compensation uses the kernels "Chroma" and the "Luma", while MP3 spends most of its time on kernels "Antialias", "Hybrid" and "Sub-Band". Table 1 summarizes the architectural parameters of the hardware kernels used in each algorithm.

Table 2 shows the number of major and minor modifications, as well as the total number of simulations, for each application. In both applications, the number of major and minor modifications are similar, which also presents the significant number of bypassed re-initializations. Also, there is a small difference in the total number of simulations, because the respective DSEs are two different runs of NSGAII, thus slight different decisions are taken within each era.

Figure 3 shows the time distribution of DSE and HLS, for H264 and MP3 respectively, including the time spent on full and partial partitioning. For H264, a significant part of the overall time (more than 25%) is spent on HLS, mainly due to the complex control logic of Luma kernel, contrarily to MP3, the kernels of which majorly contain simpler arithmetic components. It is notable that, in both algorithms, most time improvements were succeeded in full partitioning, as the probability for minor modifications is increased. Especially for partial partitioning of H264, no minor modifications occurred. In total, the time gains when using PRM, in comparison with the traditional (no-PRM) scenario, reach 11% for H264 and 17% for MP3. In addition, in the case of MP3, these time improvements can cover the HLS total time, leaving an unused slack of 8%. On the contrary, in the case of H264, the DSE time improvements partially cover the HLS time, thus there is an overhead of 9%, which however is considered quite low.

For examining accuracy, the following comparisons are made by using as reference the DSE without PRM but with HLS (no-PRM,HLS): (a) For PRM accuracy, the comparison involves DSE using PRM and HLS (PRM,HLS). (b) For HLS accuracy, we run DSE without PRM and with manually annotated DSP cores like in [10] (no-PRM,no-HLS).
This paper has presented an exploration methodology trading simulation time for increased accuracy. We introduced the usage of process-based reconfigurable SystemC modules, to reduce simulation time together with HLS for improved evaluation accuracy. With two fully established use cases, the efficiency of the proposed methodology has been analyzed, showing simulation time improvements up to 17%, encouraging the use of HLS. We show that HLS can dramatically improve the metrics accuracy by up to 77%, while this improvement may become even higher in more complex DSP accelerators, where the problems of the manual annotation become more intense.

5. CONCLUSIONS

Table 1: DSE parameters for each kernel combination

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value Range</th>
<th>Parameter</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU frequency</td>
<td>200-1200 step: 200</td>
<td>CPU number</td>
<td>1-3</td>
</tr>
<tr>
<td>Instr. cache ways</td>
<td>1,2,4,8</td>
<td>Luma cores</td>
<td>1-16</td>
</tr>
<tr>
<td>Instr. cache sets</td>
<td>32 or 64</td>
<td>Luma Filters/core</td>
<td>4 or 8</td>
</tr>
<tr>
<td>Data cache ways</td>
<td>1,2,4,8</td>
<td>Chroma instances</td>
<td>1 or 2</td>
</tr>
<tr>
<td>Data cache sets</td>
<td>32 or 64</td>
<td>Chroma cores</td>
<td>1-4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chroma MACs/core</td>
<td>1,2,4,8</td>
</tr>
</tbody>
</table>

Table 2: Simulation runs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>No PRM</th>
<th>Using PRM</th>
</tr>
</thead>
<tbody>
<tr>
<td>H264</td>
<td>484</td>
<td>289</td>
</tr>
<tr>
<td>MP3</td>
<td>1062</td>
<td>492</td>
</tr>
</tbody>
</table>

For those comparisons, the metrics values (latency $t$, area $a$ and power $p$) of the pareto points are firstly normalized, as each metric has different value ranges. Then, the Manhattan distances between each pareto point of the DSE to be compared ($P^*(y)$) and all the pareto points of the reference DSE ($P^*(y)$) are calculated as follows:

$$M(i, j) = |P^*_r(i) - P^*_r(j)| + |P^*_a(i) - P^*_a(j)| + |P^*_p(i) - P^*_p(j)|$$

The following metric quantifies how close the respective pareto curves are, hence it is preferable to be as low as possible:

$$D(i) = \min_j M(i, j)$$

Table 3 presents some statistical measurements on $D(i)$ which show the accuracy of the proposed approach. According to this table, for H264, the average and median distances in case of (PRM,HLS) are respectively 66% and 89% shorter than in case of (no-PRM,no-HLS). For MP3, the respective average and median distances are 77% and 76% shorter. Thus, it is worth trying to reduce the simulation time for encouraging the use of HLS.

Table 3: Accuracy analysis, based on Manhattan distance $D(i)$ (Reference: no-PRM,HLS)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Average $D(i)$</th>
<th>Median $D(i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H264</td>
<td>MP3</td>
</tr>
<tr>
<td></td>
<td>no-PRM, no-HLS</td>
<td>PRM, HLS</td>
</tr>
<tr>
<td>average</td>
<td>0.116</td>
<td>0.0388</td>
</tr>
<tr>
<td>median</td>
<td>0.1044</td>
<td>0.0112</td>
</tr>
</tbody>
</table>

6. REFERENCES