Exploring Heterogeneous Task-Level Parallelism in a BMA Video Coding Application using System-Level Simulation

Carlos M. Betemps*†, Mateus S. de Melo*, Amir M. Rahmani‡¶, Antonio Miele§, Nikil Dutt¶, and Bruno Zatt*
*Federal University of Pelotas (UFPel), Postgraduate Program in Computing (PPGC), Video Technology Research Group (ViTech), Pelotas, RS, Brazil. Email: {cm.betemps, msdmelo, zatt}@inf.ufpel.edu.br
†Federal University of Pampa (UNIPAMPA), Campus Bagé, Bagé, RS, Brazil. Email: carlos.betemps@unipampa.edu.br
‡TU Wien, Vienna, Austria. Email: amirr1@uci.edu
§Politecnico di Milano (Polimi), Milano, Italy. Email: antonio.miele@polimi.it
¶University of California, Irvine (UCI), CA, USA. Email: dutt@uci.edu

Abstract—**High abstraction level models can be used within the system-level simulation to allow rapid evaluations of architectural aspects in early Design Space Exploration (DSE) and direct the development decisions. Further, early DSE is of paramount importance in the specification of future Embedded Systems (ES) and its evaluation for applications with high computing demands and energy restrictions. This paper presents the exploration of Heterogeneous Task-Level Parallelism (HTLP) in a Block-Matching Algorithm (BMA) video coding application. HTLP means the creation and execution of simultaneous threads of kernels defined for different types of Processing Elements (PE) – e.g., CPU and GPU – but all for an equal purpose. We employ a BMA implementation as a case study, and its characteristics are used to explore the HTLP – in particular, its kernels for data preparation, SAD (sum of absolute differences) criteria calculation, and SAD values grouping. For the exploration, a system-level simulation framework (`SAVE-htlp`) is augmented, being able to support the HTLP. In the performed experiments, `SAVE-htlp` simulates workload and architecture models and explores 22 settings varying the PE type employed during the tasks' execution and the number of concurrent threads for each kernel. Execution time, performance, energy, and power results show HTLP settings overcoming CPU-only ones as well as those with solely GPUs to process its tasks.**

Keywords—**Heterogeneous Task-level Parallelism, System-level Simulation, Block-Matching Algorithm, Embedded Systems.**

I. Introduction

A. Motivation and Context

Embedded Systems (ES) are highly diverse, ranging from small systems such as wearables in healthcare and intelligent sensors in smart spaces to distributed and complex systems such as automotive and aerospace control systems [1], [2]. Furthermore, embedded software features, size, and complexity are in constant growth [3]. The ES design stays in a competitive scenario with issues about time-to-market, new functionalities, and performance constraints [4]. The ES design involves basically three phases [5]: the **modeling** treats the concepts and refinement of the specifications producing hardware and software models; the **validation** aiming is to achieve a reasonable confidence level that the system will works as designed; and the **implementation** is the physical realization of the hardware and software.

Execution time and energy consumption are important issues in ES design since these systems are usually powered by batteries and have tight performance constraints. Current mobile System-on-Chips (SoCs) have large amounts of computing capability [6] that can provide high performance with low energy consumption. The single and multi-core eras are constrained by issues involving power, complexity, software parallelism, and scalability [7]. However, heterogeneous systems (e.g., SoCs) arise with advancement perspectives enabled by abundant data parallelism and power efficient GPUs [7], beyond the possibility of a better matching between the application' tasks and processing elements (PEs). SoCs combine CPU cores of different features, GPU cores, and perhaps other accelerators, with high bandwidth access to memory [7].

Usually, several application parts can be executed in parallel creating possibilities to examine the task-level parallelism (TLP). Multiprocessor systems allow exploring the TLP [8], including heterogeneous systems where threads from different kernel implementations (for distinct PE types), but all for the same purpose, can be created and executed in parallel.

Video sharing services such as YouTube and Netflix are widely used these days via mobile devices. The ES used on these devices need to provide video applications. Video coding (e.g., Motion Estimation) and computer vision (e.g., Object Tracking and Stereo Matching) domains normally employ Block-Matching Algorithm (BMA). This application type has some parallelization aspects making it suitable to be explored for HTLP. That is the cause we use a BMA implementation [9] as a case study.

DSE activities usually follow the Y-chart approach [10]. Each solution includes descriptions for architecture and workload, and also the mapping between both. Using execution on real hardware, simulation, or even estimates the solutions are evaluated based on metrics related to performance and power. According to the design constraints the architecture, workload, and mapping are adjusted aiming to find the best solutions.

Due to the complexity of the new product design, the competitiveness in the ES industry, and the existing gap in the co-design of hardware and software, more and more tasks of exploring alternative solutions are carried out at the system level [10], [11]. Modeling is used to deal with the ES complexity in design initiatives and to enhance the development of such systems [12]. Higher abstraction levels favor faster and more cost-effective approaches in the design space analysis [13], allowing early estimation based on partial and uncertain information that can guide the development process [3]. Simulation is a way to evaluate different solutions. It is a form of design analysis which aim is to bring a design properties view and test the configurations [14]. Modeling and simulation using high abstraction level have a key role in early design phases, allowing to catch the system behavior and its interactions usually requiring fewer development efforts [15].

Cycle and instruction accurate simulators cannot fully support the processing demand in DSE activities, given the volume of details about the system (possibly not available in early phases) which need to be simulated and also can require prohibitively simulation time [4], [16]. Thus, the use of high abstraction level is a necessity. Some works have used high abstraction level models and tools in the representation and simulation of the workload and architecture [4], [15]–[19]. However, it usually requires program level specifications (that reduce the abstraction level) and does not use HTLP suitably.

The ES diversity and its design complexity, the opportunities to employ heterogeneity in the applications' TLP, and the use of high abstraction level modeling/simulation reveal the benefits of early DSE. Further, early DSE is of paramount value in the spec of future systems (e.g., SoCs) and its evaluation for applications with computing and energy demands.

In this work, we use an enhanced version of the SAVE system-level simulator [17]. Its code structure is based on C++/SystemC classes using Transaction Level Modeling (TLM). SystemC is a system-level design language inserted in many industry flows that uses the C++ infrastructure and its object-oriented nature, being suitable to hardware/software co-simulation [20]. TLM separate the communication components from the computation ones using channels. It improves the simulation time, allowing explore and validate design solutions at a higher abstraction level [21].

B. Contributions and Goal

This work deals with the HTLP, which involves the creation and execution of heterogeneous parallel threads from kernels implemented for different types of PEs. For instance, in the experiments, we spawned heterogeneous threads for a specific goal that were executed concurrently on both CPUs and GPUs. Thus, the SAVE system-level simulation framework [17] is augmented intending to manage the HTLP and allowing threads to run implementations of each type of PE in parallel, raising the heterogeneity applied to the system. The SAVE structure has provided a suitable base for the performed extension. We call the enhanced framework as `SAVE-htlp`, where *htlp* stands for *h*eterogeneous *t*ask-*l*evel *p*arallelism. A pool of generic processing resources represents the system architecture while task graphs are used to model the applications. Thus, both models use high abstraction level.

In summary, during the experiments, we can observe that the better matching between the tasks' threads and the used PE types present enhanced performance and energy consumption. Application tasks have different properties and are suitable to use HTLP. System-Level simulation enables the rapid evaluation of the HTLP in the BMA case study application. We can state that the work involves phases of modeling and validation in the ES design [5], but using high abstraction level models in system-level simulations. Thus, the main goal of this work can be described as follows: *to evaluate the HTLP by exploring the different kernels characteristics of the BMA case study application by the use of system-level simulation.*

This work is organized as follows. Section II presents the related works. In Section III, the system-level simulation framework is described, including the original framework (SAVE) and the modified one (`SAVE-htlp`), as well as the simulation's input structure. Section IV presents the case study – a BMA application. Moreover, the application and architecture models are presented, including aspects of the models generation. The experimental setup is described in Section V, covering the configurations applied in the experiments, whereas Section VI discuss the results, presenting data about execution time, performance, energy, and power. Finally, Section VII concludes the discussion and presents future work.

II. Related Work

Some works have treated themes related to system-level modeling, simulation, and design space exploration (DSE). In [15], an approach called Sesame is presented that focuses on the multimedia applications to efficiently prune and explore the design space of target platform architectures. The applications are represented by the Kahn Process Network model of computation, generated automatically from C/C++ specification. A model language is used to specify the application model, the architecture model, and the mapping between them for co-simulation. The method needs code level specifications to define the system and its applications, reducing the abstraction level, as well as forcing earlier resources allocation.

The COMPLEX framework and methodology are described in [16] and [4], respectively. The work presented in [16] describes an approach of Platform-based Design that uses system-level time and power estimation to perform the DSE and detaches the use of UML/MARTE based models as a design input with the automatic generation of an executable SystemC model. In [4] is presented a DSE approach that includes Model Driven Engineering and Electronic System Level technologies. The framework capture a set of solutions based on UML/MARTE models and functional code. An executable, configurable, and high-performance model is automatically created. Then, SCoPE+ tool [4] allows the simulation of different solutions and generates performance estimates. Both works require the modeling of several system viewpoints, such as data, functional, communication&concurrency, platform, architecture, and verification views. Thus, many complex models are necessary to employ the approach at early design phases.

In [18] a DSE approach for Multimedia Embedded Systems is presented. The applications are represented as HSDF (Homogeneous Synchronous DataFlow) graphs, and the hardware platform as an ATG (Architecture Template Graph) directed graph. The mapping between applications and architecture resources is performed during the DSE. DEVS formalism is used in the modeling, analysis, and simulation, allowing the system evaluation to find optimal Pareto solutions. Genetic algorithms and stochastic simulation lead the DSE. The formalism in its usage demands developers with specific expertise.

Callou *et al.* [19] propose a methodology that aims to evaluate the energy consumption and execution time of embedded real-time applications in early design phases. From assembly code or C program, a Coloured Petri Net (CPN) model is generated to represent the applications. Estimates are generated through stochastic simulation, and a tool was developed for supporting automatic measurement on the hardware platform. Again, code level representations are used in the methodology.

In early development phases, the described works do not provide the option of creating simultaneous heterogeneous threads of same goal tasks even implemented for different types of PEs. Therefore, they do not take advantage of the different characteristics of the kernels that exist in the same application. Moreover, in the stated design step, the high-level description of the system and its workload are attractive. The simulation of these models allows evaluating the system behavior not requiring the creation of several artifacts.

III. System-Level Simulation

According to Gries [10], during a system-level simulation, the evaluation occurs at a high abstraction level. An interconnection of architectural blocks like as processors, memories, and buses represent the system. Coarse models such as interaction processes or even complete procedures describe the workload. This section describes the applied system-level simulation framework, including the original simulator, the simulator's input file structure, and the performed modifications to deal with HTLP – to obtain the enhanced simulator.

A. The Original Simulator – SAVE

Miele *et al.* [17] propose the SAVE system-level simulation framework implemented in SystemC and TLM to validate runtime resource management policies for Heterogeneous System Architectures (HSA) [6]. The framework [17] intends to deal with the runtime management to allocate system resources to applications, based on the PE's efficiency and to fulfill Service Level Agreement. The applications are modeled as task graphs that include information about tasks types and latencies, performance counters, threads, and others. Still, the architecture is modeled as a set of generic resources, describing performance and power models. The model's parameters can be extracted through simulation or execution in a real hardware, or even estimated by an experienced developer. SAVE simulator does not provide the capability of splitting same purpose tasks in parallel threads mapping them on different PE types. Hence, the original framework was modified, and we named this extended version as `SAVE-htlp`.

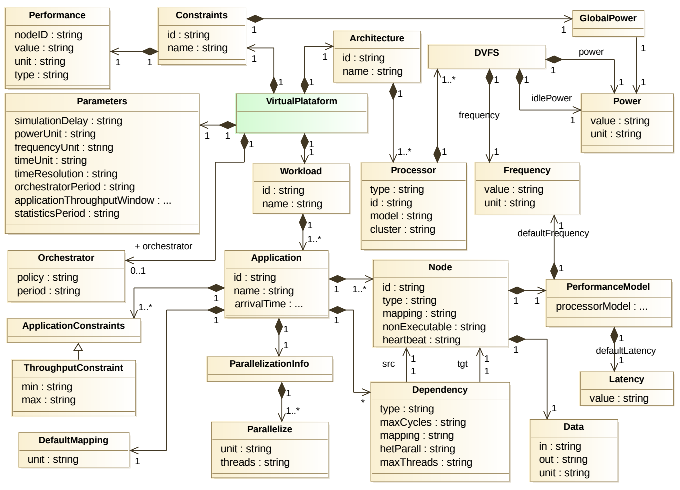


Fig. 1. `SAVE-htlp` XML file elements and their relationships.

B. The Framework's Input

An XML file provides both application and architecture models to the simulation framework. This file also contains the parameters values for each element of the architecture and applications. The Fig. 1 presents a UML class diagram [22] that conceptually illustrates the elements of an input XML file.

The XML file describes a VirtualPlatform, which consists of an Architecture, a Workload, and an Orchestrator, and have some properties such as simulation' Parameters and Constraints. The Architecture is composed of Processors, that are profiled by DVFS (Dynamic Voltage and Frequency Scaling) elements in terms of Power, IdlePower, and Frequency. A Workload includes all Applications to be executed. An Application is described by a task graph, which contains Nodes and Dependencies sets. A Node describes an application task and can be of two types: (*i*) *Main* tasks, which represent application portions that are always executed sequentially by the CPU; (*ii*) *Elaboration* tasks that represent *kernels* that can be executed by the CPU, GPU, or other PE type, and can be parallelized. Nodes contain a PerformanceModel and an input/output Data. Dependency represents an edge in the task graph. Edges can be of three types: (*i*) *forward* — indicates the normal application's advance to the next task; (*ii*) *backward* — represents a loop edge; and (*iii*) *branch* — indicates the advance to an elaboration task that can be parallelized, possibly including other elaboration tasks that can be executed in parallel on different PEs types. We here extended the original Dependency properties for HTLP execution: *hetParall* indicates that threads of the target task can be instantiated in parallel together with threads of other targets originated from the same task, and *maxThreads* defines

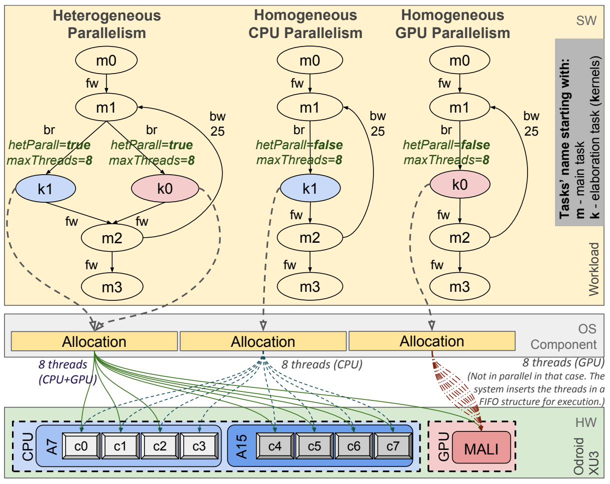

Heterogeneous Parallelism
Homogeneous CPU Parallelism
Homogeneous GPU Parallelism
SW
Tasks' name starting with:
m - main task
k - elaboration task (kernels)
m0
fw
m1
br
bw
25
hetParall=true
maxThreads=8
hetParall=false
k1
k0
m2
m3
Workload
Allocation
OS Component
8 threads (CPU+GPU)
8 threads (CPU)
8 threads (GPU)
(Not in parallel in that case. The system inserts the threads in a FIFO structure for execution.)
HW
CPU
A7
c0
c1
c2
c3
A15
c4
c5
c6
c7
GPU
MALI
Odroid XU3

Fig. 2. Heterogeneous Parallelism vs Homogeneous Parallelism.

the maximum number of (heterogeneous) parallel threads. To illustrate the HTLP concept, the Fig. 2 shows models for three sample applications and the Odroid-XU3 architecture [23]. On the up right of Fig. 2, two application models employ homogeneous parallelism using eight threads in CPU and GPU, respectively. On the up left, the model uses heterogeneous parallelism via the creation of eight threads to be executed concurrently in CPU and GPU. The attributes *hetParall* and *maxThreads* defines each case. Section IV further discuss details about the application and architecture models.

C. *The Modified Simulator* – `SAVE-htlp`

The modifications implemented in `SAVE-htlp`, compared to the original framework [17], are next described. `SAVE-htlp` allows that threads of different elaboration tasks (which can be parallelized and executed in different types of PE – described in Section III-B), achieved by the same previous task – i.e., sharing the same predecessor – can be initiated in parallel in order to increase the system heterogeneity. In addition, it has a controller to handle the join (in the next task) of the created threads from a set of elaboration tasks. The original framework allows threads that were created from only one elaboration task per time and application. Moreover, the number of threads to be created from an elaboration task set (in a TLP portion) can be explicitly defined and different for each TLP application part. Parameters in the application graph model allow it (parameters *hetParall* and *maxThreads* described in the DEPENDENCY class of Fig. 1 and in Fig. 2). `SAVE-htlp` is capable of dealing with nested loops by resetting the cycles counters. It aims to deal with more complex application models that can contain many nested loop structures. In `SAVE-htlp` the threads' latency is equal to the instantiated task latency. So, the advantage is provided by the possibility to create more threads while also executing fewer iterations.

IV. CASE STUDY APPLICATION AND ARCHITECTURE

This section describes the architecture model (based on the Odroid-XU3 platform) and a case study application (BMA – Block-Matching Algorithm). BMA is a mode of finding matching blocks in a digital video frames sequence. The BMA's goal is to get a block in the video frames representing the best match regarding a reference block. Domains like Video Coding (e.g., Motion Estimation) and Computer Vision (e.g., Object Tracking and Stereo Matching) apply such algorithms. BMA applications must use similarity criteria such as, e.g., the Sum of Absolute Differences (SAD). We employ a BMA implementation [9] as the base to the application model. Melo *et al.* [9] have used the SAD criteria in BMA implementations for GPU and FPGA. BMA has certain parallelization aspects making it suitable to be explored in HTLP.

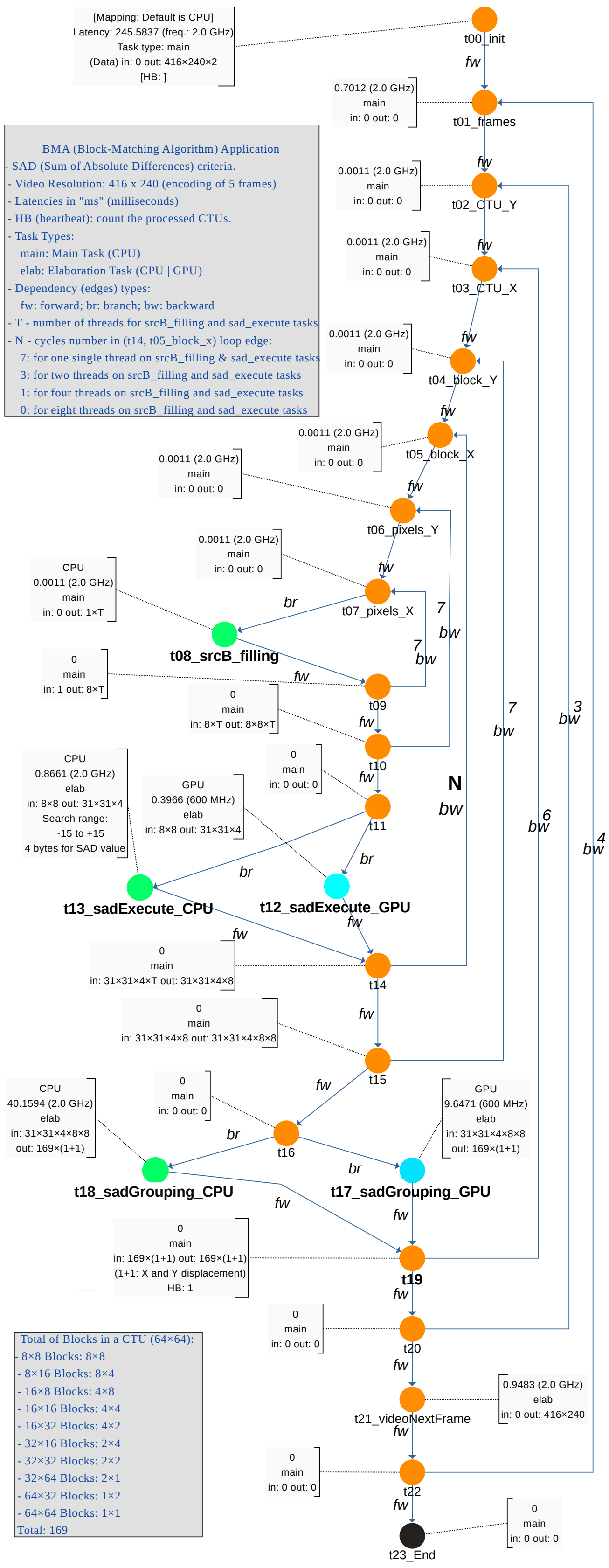
We manually build the control flow graph [24] of the BMA application code described in [9] (from its *main* function) and use it as the core application model. We have adapted this graph to the input format used in the `SAVE-htlp`. However, one can implement a way to generate it automatically from the source code. It is worth mentioning that one can generate the application model without the corresponding source code. The application model is presented in Fig. 3. Each node presents its mapping (CPU or GPU), latency, type, input/output data[1], and *heartbeat* update call [25] (when it is the case). The latter is on task *t19* (Fig. 3) where the application completes the processing of a Coding Tree Unit (CTU). A CTU is a processing unit of the High-Efficiency Video Coding (HEVC) video standard [26]. The dependencies are annotated with its type (*forward*, *backward*, or *branch*) and number of cycles in a loop (*backward* edges). We can observe that three points (elaboration tasks) in the application can be parallelized and heterogeneously executed:

- *srcB_filling*: data preparation for *sad_execute* task. According to the number of threads initiated for *sad_execute*, an equal one is triggered for this task.
- *sad_execute*: execution of SAD calculation. The number of threads for this task changes the cycles in the edge (*t14*, *t05_block_X*) — denoted by N in Fig. 3. As more threads started, fewer iterations will be needed.
- *sad_grouping*: grouping of SAD values — from 8x8 pixels size blocks into 64x64 pixels size CTU blocks.

We collect the parameters for performance and power models by profiling the BMA application [9] running on an Odroid-XU3 platform [23], featuring the Samsung Exynos 5422 Cortex-A15 quad-core and Cortex-A7 quad-core CPUs and a Mali-T628 MP6 GPU [23]. Thus, such a platform is the base of the case study architecture model.

We have adapted the BMA's application code [9] to log the tasks' latencies. Next, we executed it on the A15 cluster at 2.0 GHz frequency. For performance parameters extraction, the BMA application was executed 30 times coding five frames of

[1]The tasks latencies incorporate the memory operations delays. Thus, we have used zero values for all tasks input and output data parameters. Fig. 3 presents these values only for modeling purposes.



BMA (Block-Matching Algorithm) Application
- SAD (Sum of Absolute Differences) criteria.
- Video Resolution: 416 x 240 (encoding of 5 frames)
- Latencies in "ms" (milliseconds)
- HB (heartbeat): count the processed CTUs.
- Task Types:
  main: Main Task (CPU)
  elab: Elaboration Task (CPU | GPU)
- Dependency (edges) types:
  fw: forward; br: branch; bw: backward
- T - number of threads for srcB_filling and sad_execute tasks
- N - cycles number in (t14, t05_block_x) loop edge:
  7: for one single thread on srcB_filling & sad_execute tasks
  3: for two threads on srcB_filling and sad_execute tasks
  1: for four threads on srcB_filling and sad_execute tasks
  0: for eight threads on srcB_filling and sad_execute tasks

Total of Blocks in a CTU (64×64):
- 8×8 Blocks: 8×8
- 8×16 Blocks: 8×4
- 16×8 Blocks: 4×8
- 16×16 Blocks: 4×4
- 16×32 Blocks: 4×2
- 32×16 Blocks: 2×4
- 32×32 Blocks: 2×2
- 32×64 Blocks: 2×1
- 64×32 Blocks: 1×2
- 64×64 Blocks: 1×1
Total: 169

Fig. 3. BMA application model.

TABLE I
VALUES FOR THE PROCESSING ELEMENTS POWER MODELS

	A15		A7		Mali-T628	
Frequency (MHz)	IdlePower (W)	Power (W)	IdlePower (W)	Power (W)	IdlePower (W)	Power (W)
2000	0.1809	1.4111				
1900	0.1457	1.1632				
1800	0.1220	0.9937				
1700	0.1069	0.8686				
1600	0.0957	0.7558				
1500	0.0829	0.6665				
1400	0.0744	0.5933	0.0296	0.1335		
1300	0.0664	0.5301	0.0249	0.1179		
1200	0.0591	0.4690	0.0208	0.1014		
1100	0.0525	0.4134	0.0178	0.0882		
1000	0.0464	0.3656	0.0152	0.0761		
900	0.0408	0.3155	0.0128	0.0647		
800	0.0354	0.2713	0.0107	0.0545		
700	0.0330	0.2411	0.0089	0.0448		
600	0.0304	0.2102	0.0071	0.0356	0.0517	0.3805
500	0.0275	0.1781	0.0058	0.0284		
400	0.0242	0.1440	0.0049	0.0237		
300	0.0201	0.1036	0.0039	0.0176		
200	0.0155	0.0623	0.0029	0.0106		

TABLE II
EXPERIMENTS CONFIGURATIONS

#	ET1	threads	ET2	threads	ET3	threads	A15, A7, Mali
1	CPU	1	CPU	1	CPU	1	4, 4, 1
2	CPU	1	GPU	1	GPU	1	4, 4, 1
3	CPU	1	CPU|GPU	1	CPU|GPU	1	4, 4, 1
4	CPU	1	CPU|GPU	1	GPU	1	4, 4, 1
5	CPU	2	CPU+GPU	2	CPU	1	4, 4, 1
6	CPU	4	CPU+GPU	4	CPU	1	4, 4, 1
7	CPU	8	CPU+GPU	8	CPU	1	4, 4, 1
8	CPU	2	CPU+GPU	2	GPU	1	4, 4, 1
9	CPU	4	CPU+GPU	4	GPU	1	4, 4, 1
10	CPU	8	CPU+GPU	8	GPU	1	4, 4, 1
11	CPU	2	CPU	2	CPU	1	4, 4, 1
12	CPU	4	CPU	4	CPU	1	4, 4, 1
13	CPU	8	CPU	8	CPU	1	4, 4, 1
14	CPU	2	GPU	2	GPU	1	4, 4, 1
15	CPU	4	GPU	4	GPU	1	4, 4, 1
16	CPU	8	GPU	8	GPU	1	4, 4, 1
17	CPU	2	GPU	2	GPU	1	4, 4, 2
18	CPU	4	GPU	4	GPU	1	4, 4, 2
19	CPU	8	GPU	8	GPU	1	4, 4, 2
20	CPU	8	CPU+GPU	8	GPU	1	0, 8, 1
21	CPU	8	CPU+GPU	8	GPU	1	8, 0, 1
22	CPU	4	GPU	4	GPU	1	4, 4, 4

ET1: srcB_filling task; ET2: sad_execute task; ET3: sad_grouping task.
CPU|GPU: alternated CPU/GPU mapping; CPU+GPU: heterogeneous CPU/GPU mapping.

two video sequences from the Common Test Conditions (CTC) [27]: (*i*) Blowing Bubbles, 416x240 pixels, 50 fps; and (*ii*) Basketball Drill, 832x480 pixels, 50 fps. The average values were used to devise the PEs' performance model.

The power extraction was done using an interface to the virtual file system to log data from the Odroid-XU3' energy monitor sensors [23]. Threads for the application and energy reading were created and executed simultaneously. We have executed the readings every five milliseconds. The frequency range was 0.2 to 2.0 GHz for the A15 CPUs, and 0.2 to 1.4 GHz for the A7 CPUs (with 100MHz steps). For Mali, for both latency and power profiling, the frequency was 600 MHz. Tab. I presents the power models values for the PEs types, detailing each *frequency* and its respective *idle power* and *power*.

V. EXPERIMENTAL SETUP

Tab. II presents the settings, including the configuration *id* (#), the allocated PE type(s), the used number of threads, and the number of processors (column `A15, A7, Mali`) for each PE type available in the architecture for the *srcB_filling*, *sad_execute*, and *sad_grouping* tasks execution. Furthermore, we have used three `SAVE-htlp` resource mapping policies:

- `Base`: allocates all available CPUs (A15 and A7 clusters, in such order), according to the number of threads;

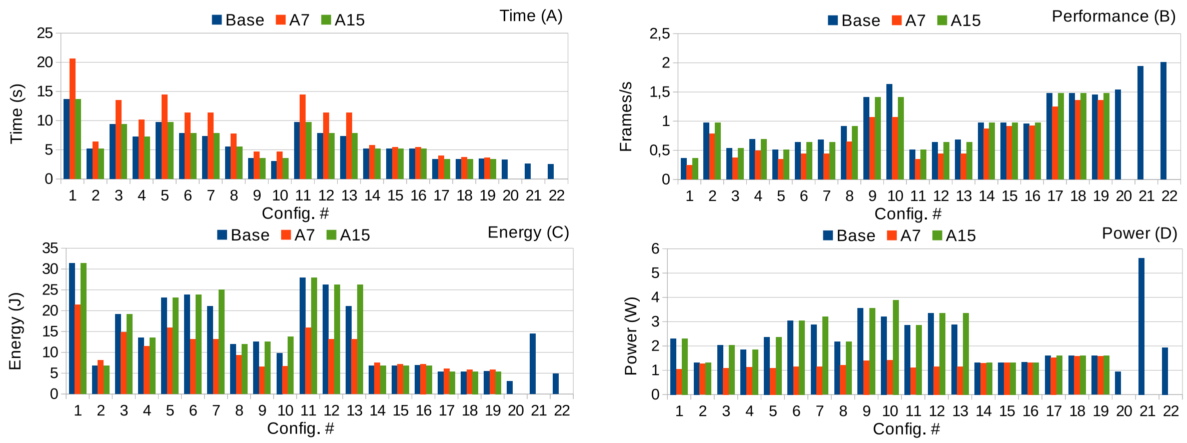


Fig. 4. (A) Execution time, (B) performance, (C) consumed energy, and (D) power per configuration and policy.

- `A7`: allocates only the A7 CPUs, according to the number of threads;
- `A15`: allocates only the A15 CPUs, according to the number of threads.

In the configurations #1 to #4 only one thread is triggered for each elaboration task. The PE type used in the *srcB_filling* task is always CPU since it has a nature of a memory transfer task. In all the configurations the task *sad_grouping* uses only one thread since it is a sequential procedure with abundant data parallelism. For the same task, the used PE type is CPU or GPU (exclusively) in each configuration. Exceptionally, setting #3 uses the CPU and GPU in an alternating way. The *sad_execute* task has configurations with only one type of PE (CPU or GPU) except for the settings #3 to #10. In #3 to #4 is used the alternated mapping between CPU and GPU. Settings #5 to #10 use the HTLP. According to the number of threads and PEs, threads execute in both types of PE (GPU and CPU). In configurations #17 to #19, the *sad_execute* is performed considering one additional GPU to deal with the *sad_execute* and *sad_grouping* kernels. The settings #1 to #19 were simulated with the three previously cited `SAVE-htlp` resource mapping policies (`Base`, `A7`, and `A15`). Settings #20 to #22 were executed only with the `Base` policy. In #20 and #21, we consider the use of only A7 and A15 CPUs, respectively. In #22, we extrapolate the number of GPUs to four in the kernels' execution – four threads in CPUs and GPUs for the *srcB_filling* and *sad_execute* tasks, respectively. In the simulations, execution time, performance, energy, and power metrics were extracted and presented in Section VI.

We perform the experiments in a notebook equipped with an Intel(R) Core(TM) i7-3612QM CPU (4 cores and 8 threads), 64bits, 2.10GHz frequency, 256KB L1 cache, 1MB L2 cache, 6MB L3 cache and 4GB main memory. The operating system version is Linux 4.4.0-130-generic #156-Ubuntu. We use the Linux *time* command (*user* mode) to log the simulation times.

VI. Results and Discussion

Using the described policies, Fig. 4 (A) presents the execution time of each configuration (Tab. II). Regarding the performance, Fig. 4 (B) presents the frames per second rates for the application in each setting and policy. Fig. 4 (C) exhibits the total energy consumed by each setting. In the same way, Fig. 4 (D) shows the data related to power.

Concerning the used `SAVE-htlp` policies, we can observe that `Base` and `A15` have similar behavior in all metrics. However, the `Base` policy achieves better results in time and performance. It is due to the use of more CPUs (A15 and A7) jointly. This scenario occurs when eight threads are triggered, and almost all CPUs are used in parallel with the GPU. `A7` policy presents the worst time and performance values, except in the settings #14 to #19, where the difference is not significant. The higher number of threads only in GPU (and one more GPU in #17 to #19) for the running of the *sad_execute* task cause it, but is limited by the number of GPUs. Regarding energy and power, `A7` policy shows lower values and yet little variations in #14 to #19 for time and performance. These values show the high energy efficiency and performance of GPUs. These findings indicates the GPU dependency in BMA application kernels (*sad_execute* and *sad_grouping*) to achieve better performances. We claim that the `Base` policy is suitable to performance-oriented scenarios since it can get an advantage of the total number of CPUs at lower energy costs.

Using the configuration #1 in `Base` policy as the baseline, Fig. 5 (A), (B), (C), and (D) presents an evaluation of the settings in the resource mapping policies (`Base`, `A7`, and `A15`) using the described metrics (Time, Performance, Energy, and Power). In `A7` and `A15` policies, we can observe the performance limitation regarding the use of four parallel threads at most. Therefore, this reinforces the previous claim about the `Base` policy. The following analysis considers only the `Base` policy unless otherwise noted.

Between the settings with an Odroid-XU3 compatible architecture (#1 to #16), we can observe the high performance when applying heterogeneous parallelism, i.e., a GPU processes the *sad_grouping* task, and CPU and GPU computes the *sad_execute* task threads (2, 4, or 8) – #8 to #10. These settings employ CPU and GPU to process the *sad_execute* task threads

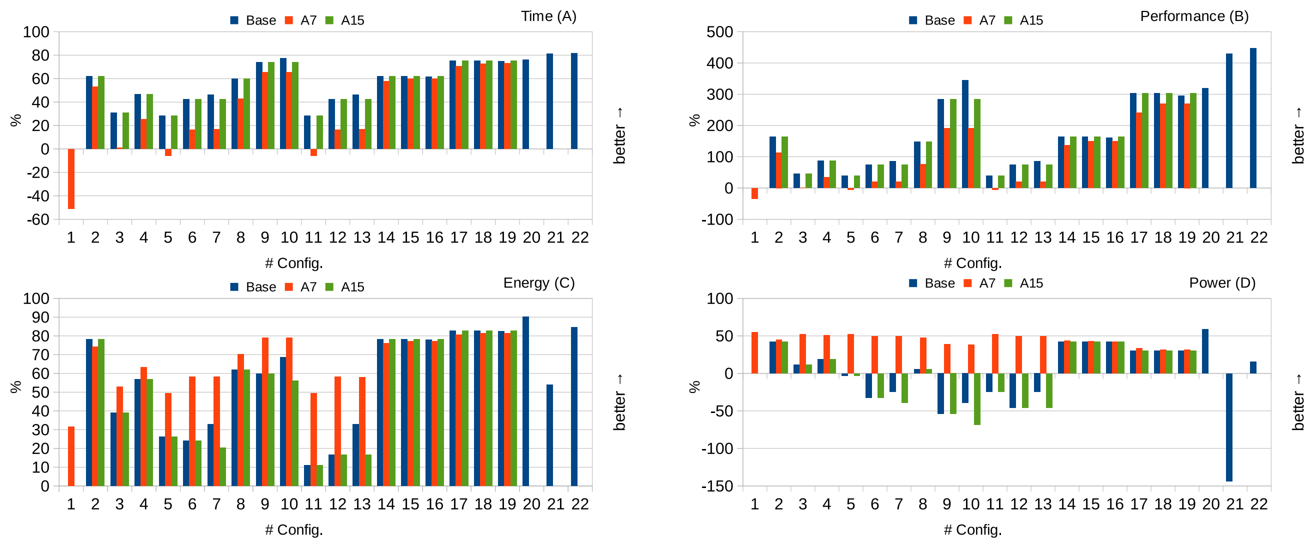


Fig. 5. Percentage gains for time (A), performance (B), energy (C), and power (D) for each configuration in the three resource policies (`Base`, `A7`, and `A15`) with #1 in the `Base` policy as baseline.

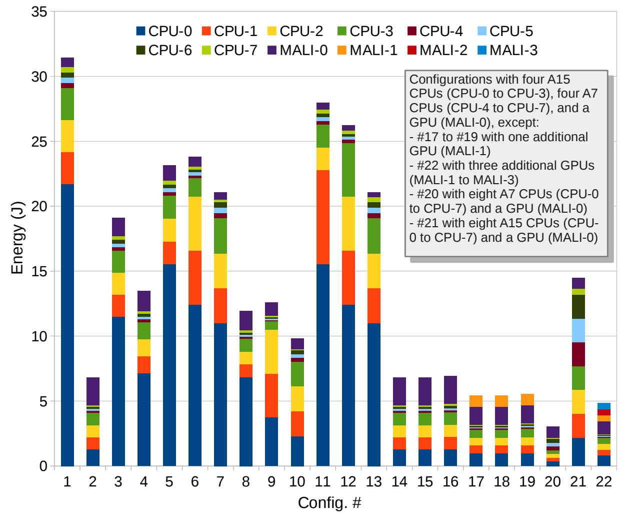


Fig. 6. Energy for each PE in the 22 configurations (in the `Base` policy).

and obtain advantage from use GPU to compute *sad_grouping* task threads. The CPU is not beneficial for the *sad_grouping* task due to their high latency and energy consumption.

Compared to the setting #1, #10 achieves gains circa 345% of performance (running in a 77% lower time), consuming approximately 68% less energy, and dissipating about 39% more power. Regarding performance, we consider #10 (in #1 to #16) the best configuration since it obtains the higher performance consuming nearly 49% more energy compared to setting #9 on `A7` policy (the lowest consumed energy in #1 to #16). About energy, we can emphasize the settings #14 to #19, which uses only GPU in the *sad_execute* and *sad_grouping* tasks. In these configurations, the consumed energy is similar in all policies while the performance is near to #10.

Only not Odroid-XU3 compliant settings have achieved similar or better performances compared to #10. Configurations #17 to #19 have one additional GPU and significantly increments the performance. In #20 and #21 was used only one type of CPU. Setting #20 uses eight A7 CPUs and one GPU to achieve a throughput similar to #10 consuming 31% of the energy. On the other hand, #21 uses eight A15 CPUs and one GPU to achieve the second higher performance consuming 47% more energy than #10. In #22, the four GPUs in the architecture has achieved the higher performance and the second lower energy consumption of the experiments. Compared to #1, #22 has achieved gains of 447% in performance, consuming 84% less energy, executing in an 81% lower time, and dissipating 15% less power.

Using the `Base` policy, Fig. 6 presents the consumed energy by each PE. In settings #1, #3, and #4 the high consumption of the CPU-0 is evident, caused by the use of only one thread. In #2 the use of solely GPU in the *sad_execute* and *sad_grouping* kernels show its energy efficiency. Configurations #5 to #10 employs the HTLP and affirms the necessity of GPU in the processing of the *sad_grouping* task since #8 to #10 present lower values for energy and time. Settings #11 to #13 shows that even with more threads, just CPUs do not achieve comparable performance and still consume more energy. Configurations #14 to #19 and #22 shows energy consumptions similar to the ones obtained using the `A7` policy, but with a top like performances. In #20 and #21, A7 CPUs and a GPU can achieve top performances, while A15 CPUs shows its energy cost to obtain high throughput.

Concerning the *sad_execute* task, during the experiments threads process each the same amount of data. However, the GPU thread finishes its work earlier, making the CPU threads represent a bottleneck, implying in the overall latency. Thus, for the data partition or the number of threads in CPUs and GPUs, we could use another strategy. An approach is to assign a not equal data size for the CPU and GPU threads or even

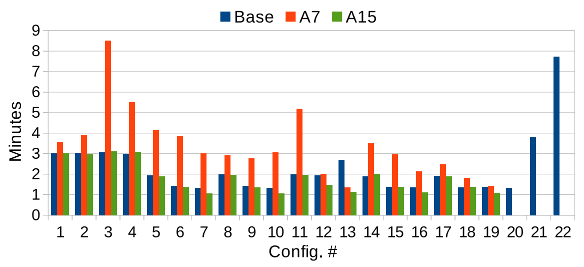


Fig. 7. Simulation times for each setting in the resource policies.

a different partition of the threads between CPUs and GPUs, intending to get a similar measure of time to each one complete its workload. Singh *et al.* [28] present promising strategies.

In the experiments, the simulation times varied from 1.07 to 8.51 minutes and the average was 2.47 minutes per configuration. Thus, one can evaluate many scenarios in a short period. Fig. 7 shows the simulation times for each setting in the resource policies (`Base`, `A7`, and `A15`).

VII. Conclusions and Future Work

This work has presented the exploration of HTLP through a case study. `SAVE-htlp` was adopted to employ application and architecture models through the rapid simulation of 22 different settings, using performance and power models based on values extracted from a real platform. Settings with HTLP have shown better results caused by the fitter matching between the tasks, its number of threads, and used PE types.

We stress the importance of knowing the application characteristics to obtain benefit from each TLP' portion. The BMA has three such parts: (*i*) *srcB_filling* task prepares data to be processed and can be parallelized only in CPU threads, (*ii*) *sad_execute* threads are numerous and are worth create them on GPUs and CPUs in a parallel way (HTLP), and (*iii*) *sad_grouping* have abundant data parallelism processed by the GPU sequentially. Compared to the baseline, the best setting with HTLP has executed in a 77% lower time and consumed 68% less energy.

Early DSE needs executable models. The presented models can be defined in development initiatives and simulated allowing to direct design decisions. Moreover, system-level simulation enables to explore many settings in a short period.

As future work, we plan to extend the `SAVE-htlp` framework in at least three aspects: (*i*) provides FPGA processing units, (*ii*) add resource policies to manage the HTLP and take advantage of it, and (*iii*) collect and provide memory usage details, like data transfers and contention.

References

[1] M. Voelter, D. Ratiu, B. Kolb, and B. Schaetz, "Mbeddr: Instantiating a language workbench in the embedded software domain," *Automated Software Engineering*, vol. 20, no. 3, pp. 339–390, 2013.

[2] F. Firouzi *et al.*, "Internet-of-things and big data for smarter healthcare: From device to architecture, applications and analytics," *Future Generation Computer Systems*, vol. 78, pp. 583 – 586, 2018.

[3] M. Hendriks, T. Basten, J. Verriet, M. Brassé, and L. Somers, "A blueprint for system-level performance modeling of software-intensive embedded systems," *International Journal on Software Tools for Technology Transfer*, vol. 18, no. 1, pp. 21–40, 2016.

[4] F. Herrera *et al.*, "The COMPLEX methodology for UML/MARTE Modeling and design space exploration of embedded systems," *Journal of Systems Architecture*, vol. 60, no. 1, pp. 55–78, 2014.

[5] G. De Micheli and R. K. Gupta, "Hardware/software co-design," *Proceedings of the IEEE*, vol. 85, no. 3, pp. 349–365, 1997.

[6] "HSA Foundation," http://www.hsafoundation.com/, 2018.

[7] P. Rogers, "Heterogeneous system architecture overview," in *2013 IEEE Hot Chips 25 Symp. (HCS)*, Aug 2013, pp. 1–41.

[8] D. P. Scarpazza, P. Raghavan, D. Novo, F. Catthoor, and D. Verkest, "Software simultaneous multi-threading, a technique to exploit task-level parallelism to improve instruction- and data-level parallelism," in *Integrated Circuit and Syst. Design. Power and Timing Modeling, Optimiz. and Simulation*. Springer, 2006, pp. 12–23.

[9] M. Melo *et al.*, "A parallel motion estimation solution for heterogeneous system on chip," in *Integrated Circuits and Systems Design (SBCCI), 2016 29th Symposium on*. IEEE, 2016, pp. 1–6.

[10] M. Gries, "Methods for evaluating and covering the design space during early design development," *Integration, the VLSI journal*, vol. 38, no. 2, pp. 131–183, 2004.

[11] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded system design: modeling, synthesis and verification*. Springer Science & Business Media, 2009.

[12] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, "Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice," *Software & Systems Modeling*, pp. 1–23, 2016.

[13] X. An, A. Gamatié, and E. Rutten, "High-level design space exploration for adaptive applications on multiprocessor systems-on-chip," *Journal of Systems Architecture*, vol. 61, no. 3–4, pp. 172–184, 2015.

[14] C. Ptolemaeus, *System design, modeling, and simulation: using Ptolemy II*. Ptolemy.org, Berkeley, 2014, vol. 1.

[15] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra, "A framework for system-level modeling and simulation of embedded systems architectures," *EURASIP Journal on Embedded Syst.*, no. 1, p. 82123, 2007.

[16] K. Grüttner *et al.*, "The complex reference framework for hw/sw co-design and power management supporting platform-based design-space exploration," *Microprocessors and Microsystems*, vol. 37, no. 8, pp. 966–980, 2013.

[17] A. Miele, G. C. Durelli, M. D. Santambrogio, and C. Bolchini, "A System-Level Simulation Framework for Evaluating Resource Management Policies for Heterogeneous System Architectures," in *Digital System Design, 2015 Euromicro Conf. on*. IEEE, 2015, pp. 637–644.

[18] B. Nogueira, P. Maciel, E. Tavares, R. M. A. Silva, and E. Andrade, "Multi-objective optimization of multimedia embedded systems using genetic algorithms and stochastic simulation," *Soft Computing*, pp. 1–18, 2016.

[19] G. Callou *et al.*, "Energy consumption and execution time estimation of embedded system applications," *Microprocessors and Microsystems*, vol. 35, no. 4, pp. 426–440, 2011.

[20] H. D. Patel and S. K. Shukla, *Ingredients for Successful System Level Design Methodology*. Springer, 2008.

[21] L. Cai and D. Gajski, "Transaction level modeling: An overview," in *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '03. New York, NY, USA: ACM, 2003, pp. 19–24.

[22] H. Gomaa, *Software modeling and design: UML, use cases, patterns, and software architectures*. Cambridge University Press, 2011.

[23] Hardkernel, "ODROID-XU3," http://www.hardkernel.com/main/products/prdt_info.php?g_code=g140448267127, 2018.

[24] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[25] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal, "Application heartbeats: A generic interface for specifying program performance and goals in autonomous computing environments," in *Proc. of the 7th Int. Conf. on Autonomic Computing*. New York, NY, USA: ACM, 2010, pp. 79–88.

[26] G. J. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Trans. on Circuits Syst. for Video Technol.*, vol. 22, no. 12, pp. 1649–1668, 2012.

[27] K. Sharman and K. Suehring, "Common test conditions," Joint Collaborative Team on Video Coding of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11. JCTVC-AC1100. Macao, China, Tech. Rep., 2017.

[28] A. K. Singh, A. Prakash, K. R. Basireddy, G. V. Merrett, and B. M. Al-Hashimi, "Energy-efficient run-time mapping and thread partitioning of concurrent opencl applications on cpu-gpu mpsocs," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, pp. 147:1–147:22, Sep. 2017.