3D Optimisation of Software Application Mappings on Heterogeneous MPSoCs

Gereon Führ[1](✉), Ahmed Hallawa[1], Rainer Leupers[1], Gerd Ascheid[1], and Juan Fernando Eusse[2]

[1] RWTH Aachen University, Aachen, Germany
{fuehr,hallawa,leupers,ascheid}@ice.rwth-aachen.de
[2] Silexica GmbH, Cologne, Germany
eusse@silexica.com

Abstract. Increasing the efficiency of parallel software development is one of the key obstacles in taking advantage of heterogeneous multi-core architectures. Efficient and reliable compiler technology is required to identify the trade-off between multiple design goals at once. The most crucial objectives are application performance and processor power consumption. Including memory power into this multi-objective optimisation problem is of utmost importance. Therefore, this paper proposes the heuristic MORAM solving this three-dimensional Pareto front calculation. Furthermore, it is integrated into a commercially available framework to conduct a detailed evaluation and applicability study. MORAM is assessed with representative benchmarks on two different platforms and contrasted with a state-of-the-art evolutionary multi-objective algorithm. On average, MORAM produces 6% better Pareto fronts, while it is at least 18× faster.

Keywords: Power-performance trade-off · Mapping · Heterogeneous · MPSoCs · Multi-objective optimisation · Pareto

1 Introduction

For today's computational requirements of the embedded domain, heterogeneous Multi- and Many-Processor Systems-on-Chip (MPSoCs) provide the best trade-off for power, performance and cost requirements. However, developers writing applications for MPSoCs are forced to consider the increased hardware complexities all at once. Moreover, power management techniques, such as Dynamic Voltage and Frequency Scaling (DVFS), have to be set carefully to handle power budgets efficiently. Especially for high-performance embedded applications, memory power consumption has a share of up to 46% of the entire system power [11,16]. Hence, fast and accurate compiler technology has to ease software development and determine the trade-off between all requirements.

Consequently, it is not a coincidence that the simultaneous optimisation of different goals is the next evolution towards optimised software [6,12,13,15,18].

A. Brinkmann et al. (Eds.): ARCS 2020, LNCS 12155, pp. 56–68, 2020.
https://doi.org/10.1007/978-3-030-52794-5_5

This simultaneous optimisation is essentially a Multi-Objective Optimisation Problem (MOOP), which is known to be NP-hard [25]. The main research focus is still towards application performance and Processing Element (PE) power consumption, neglecting the impact of the memory to the entire power share. Therefore, this paper presents a solution for this three-dimensional MOOP. Compared to single-objective optimisation, an entire set of non-dominated optimal solutions is determined. This set is also known as Pareto front. These precomputed solutions can be used during execution of the application to select the most appropriate configuration dynamically [22,23].

In the literature, there are numerous multi-objective optimisation algorithms, e.g. particle swarm optimisation [31,32]. However, a widely utilised family of algorithms suitable for MOOPs in the context of MPSoC optimisation are Evolutionary Multi-Objective Algorithms (EMOAs) (Sect. 2). EMOAs are preferred as they find close-to-optimal solutions. They are general-purpose search strategies, population-based and inspired by biological evolution. However, EMOAs are performance sensitive to their hyperparameter setup. Also, the deterioration in solution quality is significant when the computational resources are restricted.

Much attention has also been paid to ensuring comprehensive tool flows. Examples are HOPES [14], DAEDALUS [29] or MAPS [19], and SLX [2].

As a consequence, this work proposes the novel heuristic *MORAM* designed to calculate three-dimensional Pareto fronts. The MOOP considers the objectives application performance, PE and memory power consumption for software application mapping on MPSoCs. The solution quality is comparable with a state-of-the-art EMOA implementation, while the computation is much faster. Further, MORAM is integrated into SLX to enable the evaluation of the applicability and quality. The case studies are based on representative benchmarks and two different hardware platforms: ODROID-XU3 [1], and an in-house Heterogeneous Many-core Virtual Platform (HeMVP).

2 Related Work

For single-objective problems, heuristics find satisfactory solutions in a short time frame by extracting and integrating the MPSoC platform and application features [27]. As the MOOP solution space is significantly larger, heuristic approaches are commonly not considered. To mention one example that addresses an MOOP, the authors of [10] present a heuristic which computes the Pareto front for two objectives: application performance and PE power. It achieves comparable solutions but with considerably faster speed than an EMOA.

More popular approaches for MOOPs are machine learning models, getting the best power-performance trade-off. The authors of [6] propose run-time optimisation of task mapping, voltage and frequency selection. A library is generated storing the Pareto optimal system configuration for minimum power and maximum performance. At run time, the most appropriate configuration is selected. In [13], a multinomial logistic regression classification is used to map a set of classifiers offline to Pareto optimal platform configurations. During run-time, these classifiers are invoked to select the most suitable configuration for the current system load. A later approach applies Deep Q-Learning to dynamically control the processor type, their number and frequency [12]. Similar to all these run-time methods, design-time training is necessary. With the heuristic approach of the paper at hand, there is no need to apply a machine learning model. Pareto optimal configurations are available immediately after the execution of the heuristic.

Software mapping optimisation based on EMOAs is presented, e.g. in [18]. Following the divide-and-conquer principle, a decomposition approach avoids handling the entire MOOP at once. The workload balancing for each processor, cluster and communication network is computed independently. A post-optimisation step captures the final Pareto front of this mapping problem. In [15], the two objectives application performance and memory energy consumption are optimised based on EMOAs. The latter comes closest to the approach proposed in this work. The heuristic of this paper optimises for performance, processor power and memory power consumption as a three-dimensional MOOP.

3 System Model

For the exploration and modelling of the mapping problem, SLX requires an application written in C for Process Networks and an MPSoC platform model as input. The tool offers heuristics to optimise for performance, PE power consumption, or solving the MOOP of both objectives [5]. A target-specific code generator translates the parallel code including the output of these heuristics into plain C code that is fed into the MPSoC compiler.

3.1 Application Model

The Kahn Process Network (KPN) model of computation is a well-known approach for modelling parallel behaviour [17]. Processes execute deterministically and sequentially, and communicate via unbounded point-to-point First-In First-Out (FIFO) channels. As a consequence, the mapping problem is reduced to the optimal distribution and mapping of the processes and selection of the appropriate power modes.

KPN applications are described as directed graphs, i.e. $A = \{\mathcal{Z}, \mathcal{C}\}$, where $\mathcal{Z}$ is the set of the application processes and $\mathcal{C}$ is the set of directed FIFO channels. Via the FIFO channel $c_{ij} \in \mathcal{C}$, a process $z_i \in \mathcal{Z}$ can communicate with process $z_j \in \mathcal{Z}$. For the implementation of KPN applications in ANSI-C, a small set of keywords form C for Process Networks (CPN). With this, processes and channels, and the required operations for accessing them are described. As unbounded FIFO channels cannot be realised, the minimum size is chosen that allows deadlock-free execution of the application.

In order to assess the application and generate timing and power information, an event-driven approach simulates a CPN application. The combination of static and dynamic profiling enables the computation of process and total execution time, as well as individual and collective power values. The dynamic profiling collects traces that contain the execution dependent behaviour of the application, such as write accesses to output channels. This timing simulation engine estimates the execution time, including communication within a 20% error margin [4].

3.2 MPSoC Model

The MPSoC platform model specifies, e.g., memory and communication architecture, and type and number of PEs. An MPSoC platform L is modelled as a directed graph: $L = \{R, E\}$, where R is a set of hardware resources present in the platform and E defines a set of connections. In this paper, R contains a set of all PEs $\mathcal{Q}$, all memories $\mathcal{M}$, and all caches $\mathcal{K}$, with $R = \mathcal{Q} \cup \mathcal{M} \cup \mathcal{K}$ and $(\mathcal{Q} \cap \mathcal{M}) \cup (\mathcal{Q} \cap \mathcal{K}) \cup (\mathcal{M} \cap \mathcal{K}) = \emptyset$. The set $M_q = \{m_1, m_2, ...\}$ denotes all memories reachable by $q \in \mathcal{Q}$.

A write Hardware Channel (HWC) is the path from q to a reachable $m \in M_q$ using connections $\{e_1, e_2, ...\} \in E$, crossing caches $\{k_1, k_2, ...\} \in \mathcal{K}$. For read HWCs, the direction is the opposite. FIFO channels are assigned to HWC after the processes $\mathcal{Z}$ are mapped to $\mathcal{Q}$. An HWC consists of a write HWC starting at the source PE q_i and a read HWC ending at sink PE q_j. Both write and read HWC use the same $m \in \{M_{q_i} \cap M_{q_j}\}$. Existing heuristics take care to choose a reachable m that is closest to both PE.

Power information is defined for each PE and memory. Hardware resources connected to the same power supply are part of a common voltage domain. Similarly, all resources connected to the same clock are part of the same frequency domain. The underlying power model consists of the basic CMOS power consumption parts, i.e. leakage power $P^{\mathrm{s}}_{f,i} = I \cdot V_f$ and dynamic power $P^{\mathrm{d}}_{f,i} = C \cdot f \cdot V_f^2$, where i indicates the hardware resource, I denotes the leakage current, V_f is the permitted minimum voltage for frequency f, and C is the switching capacitance and f the operating frequency. In [21], it is shown that with this model, power estimates for PEs including L1 caches are possible with about 9% error on average. The memory power model achieves average estimation errors of 15% (DRAM) and 11% (SRAM).

4 Multi-objective Optimisation

The objective functions are evaluated by simulating the task mapping and platform power configuration. The objective space has 3 values: the execution time of the application t^{e}, the average power consumed by the PEs $P^{\mathcal{Q}}$ and the average power consumed by the memories $P^{\mathcal{M}}$. The following reasoning explains why it is sufficient to not include memory allocation in the decision space. It only consists of the process mapping and the platform power configuration, i.e. the selection of voltage and frequency.

Due to the principles of KPN, source processes send their data to destination processes via the FIFO channels. This procedure requires writes and reads to the memory because the FIFO buffer is allocated there and contributes to increment the memory power consumption. By changing the process-to-PE mapping, the access behaviour to the underlying memory hierarchy is implicitly influenced.

For single shared main memory systems, the effectiveness of the caches can be exploited with an optimal mapping and thus the memory power reduced. In the case of distributed memories of different size or purposes, i.e. scratchpad or shared, it makes sense to allocate the FIFO buffer always to the most local memory (Sect. 3.2). If the source and destination process are assigned to PEs that do not share a scratchpad, the HWC is, e.g., routed via the main memory. The consequence is higher memory power consumption and slower application execution time.

4.1 Problem Definition

For the formal problem definition, the processes $\mathcal{Z}$ and available PEs $\mathcal{Q}$ are part of the inputs. The platform power configuration set $\mathcal{C}$ is also required and taken from the platform model. $\mathcal{C}$ contains the set of possible frequencies $\mathcal{F}$, the permitted minimum voltage V_f for a selected frequency f, the switching capacitance C and the leakage current I for every hardware resource. The CPN simulation engine computes t^{e}, $P^{\mathcal{Q}}$ and $P^{\mathcal{M}}$ according to Eq. 1, 2 and 3.

$$t^{\mathrm{e}} = \sum_{q\in\mathcal{Q}} \sum_{z\in\mathcal{Z}} \frac{M_{z,q}}{f_{k,q}} (t^{c}_{z,q} + t^{s}_{z,q}) \tag{1}$$

$$P^{\mathcal{Q}} = \sum_{q\in\mathcal{Q}} \sum_{z\in\mathcal{Z}} P^{\mathrm{d}}_{f,q} M_{z,q} + P^{\mathrm{s}}_{f,q} \tag{2}$$

$$P^{\mathcal{M}} = \sum_{m\in\mathcal{M}} P^{\mathrm{s}}_{m} + P^{\mathrm{d}}_{m} \cdot u_m \tag{3}$$

The number of cycles used by z scheduled on q are given with $t^{c}_{z,q}$. The simulated inter-process dependencies and concurrencies are considered with $t^{s}_{z,q}$, namely the latencies incurred by context switches and FIFO data communication, i.e. delays caused by the HWCs. The utilisation of the memory u_m is calculated using the HWC access activity trace, which is generated by the CPN simulation engine. $M_{z,q} = 1$ indicates that process z is mapped to PE q.

The resulting minimisation problem is given in Eq. 4, where (4b) and (4c) define that each process is mapped on exactly one PE.

$$\min \quad \mathbf{f} = \left(t^{\mathrm{e}}, P^{\mathcal{Q}}, P^{\mathcal{M}}\right) \tag{4a}$$

$$\text{s.t.} \quad \sum_{q\in\mathcal{Q}} M_{z,q} = 1,\ \forall z \in \mathcal{Z} \tag{4b}$$

$$M_{z,q} \in \{0,1\},\ \forall z \in \mathcal{Z},\ \forall q \in \mathcal{Q} \tag{4c}$$

4.2 Heuristic: MORAM

MORAM finds a Pareto front approximation for the objectives application execution time $\mathbf{t}^{\mathrm{e}}$, PE and memory power consumption, $\mathbf{P}^{\mathcal{Q}}$ and $\mathbf{P}^{\mathcal{M}}$. As invoking the CPN simulation engine is a major bottleneck, and due to the large search space, a pruning step is necessary. This procedure is taken from [10]. The authors prove that this reduces the search space effectively and application-independent. The final Pareto front is generated on the basis of this reduced exploration space. The heuristic contains further techniques reducing the amount of CPN simulation engine calls.

Algorithm 1 shows the entire pseudo code of MORAM with the input sets: PEs $\mathcal{Q}$, the processes $\mathcal{Z}$, and all platform power configurations $\mathcal{C}$. The heuristic outputs the final Pareto front in form of the objective value vectors $\mathbf{t}^{\mathrm{e}} \in \mathbb{R}^{|PP|}$, $\mathbf{P}^{\mathcal{Q}} \in \mathbb{R}^{|PP|}$, $\mathbf{P}^{\mathcal{M}} \in \mathbb{R}^{|PP|}$, with $|PP|$ being the number of the final Pareto points. Further, the corresponding platform power configurations $\mathbf{C} \in \mathbb{R}^{|PP| \times |\mathcal{Q}| \times 2}$ contain the selected frequency and voltage per PE and Pareto point. Also, the process mappings $\mathbf{M} \in \{0,1\}^{|PP| \times |\mathcal{Z}| \times |\mathcal{Q}|}$ are part of the output. In the following, the individual steps of MORAM are discussed.

PruneSearchSpace. First, a pre-pruning phase is necessary if $|\mathcal{C}|$ is very large to keep the run time of the entire pruning phase acceptable. To formalise this, a user-defined number N enables a uniform distributed random process, selecting N platform configurations, as shown in line 6–8. The uniform distribution ensures a representative selection of all possible $\mathcal{C}$. According to [10], a reasonable choice of N would be $N = 10^5$.

Second, a classification and selection procedure is performed based on two qualifiers: Total Nominal Power (TNP) and the Execution Time Indicator (ETI) (lines 9–11). The TNP value for a $c \in \mathcal{C}$ is computed as given in Eq. 5. The ETI reflects an execution time approximation, where the processes are assumed to have all input data available and are ready to execute. Also, the process-to-PE assignment is not done to remain mapping independent. The execution time is calculated for every PE type $\mathcal{Q}_{type} \subseteq \mathcal{Q}$. Hence, inter-process dependencies and concurrencies are not considered in Eq. 6.

$$P_{\mathrm{TNP}} = \sum_{q \in \mathcal{Q}} P^{\mathrm{d}}_{f,q} + P^{\mathrm{s}}_{f,q} \tag{5}$$

$$t_{\mathrm{ETI}} = \sum_{q \in \mathcal{Q}_{type}} \sum_{z \in \mathcal{Z}} t^{\mathrm{c}}_{z,q} / f_q \tag{6}$$

A configuration is considered non-dominated, if no other configurations with lower P_{TNP} and t_{ETI} are available (line 12). As there are too many remaining configurations $\mathcal{C}'$, only a fraction is selected (lines 13–15), namely every $\lfloor log_2(|\mathcal{C}'|) \rfloor$. This log_2 based selection size causes an efficient reduction of the solution space, trading subsequent algorithm run time with potential Pareto front candidates.

Originally designed for two objectives, this pruning procedure provides a notion of whether $c \in \mathcal{C}$ is a potential candidate for the final Pareto front.

Algortihm 1: Heuristic MORAM

```
Input: $\mathcal{Q}$, $\mathcal{Z}$, $\mathcal{C}$
Output: $\mathbf{M} \in \{0,1\}^{|PP| \times |\mathcal{Z}| \times |\mathcal{Q}|}$, $\mathbf{C} \in \mathbb{R}^{|PP| \times |\mathcal{Q}| \times 2}$, $\mathbf{t}^{\mathrm{e}} \in \mathbb{R}^{|PP|}$, $\mathbf{P}^{\mathcal{Q}} \in \mathbb{R}^{|PP|}$,
        $\mathbf{P}^{\mathcal{M}} \in \mathbb{R}^{|PP|}$, with $|PP|$ being the number of the final Pareto points
1  Function MORAM()
2      $\mathcal{C}_{\text{pareto}}$ = PruneSearchSpace($\mathcal{Q}$, $\mathcal{Z}$, $\mathcal{C}$);
3      $\{\mathbf{M}, \mathbf{C}, \mathbf{t}^{\mathrm{e}}, \mathbf{P}^{\mathcal{Q}}, \mathbf{P}^{\mathcal{M}}\}$ = GetParetoFront($\mathcal{Q}$, $\mathcal{Z}$, $\mathcal{C}_{\text{pareto}}$);
4      return $\mathbf{M}_{z,q} \forall z \forall q$, $\mathbf{C} \forall q$, $\mathbf{t}^{\mathrm{e}}$, $\mathbf{P}^{\mathcal{Q}}$, $\mathbf{P}^{\mathcal{M}}$;
5  Function PruneSearchSpace($\mathcal{Q}$, $\mathcal{Z}$, $\mathcal{C}$)
6      if $|\mathcal{C}| > N$ then
7          $\mathcal{C}_{\text{new}} = N$ randomly selected entries of $\mathcal{C}$;
8          $\mathcal{C} = \mathcal{C}_{\text{new}}$;
9      foreach $c \in \mathcal{C}$ do
10         c.TNP = calculate total nominal power;
11         c.ETI = calculate execution time indicator;
12     $\mathcal{C}'$ = non-dominated $c \in \mathcal{C}$ according to TNP and ETI; sort ascending by ETI;
13     for $i = 0$; $i < |\mathcal{C}'|$; $i$ += $\lfloor log_2(|\mathcal{C}'|) \rfloor$ do
14         $\mathcal{C}_{\text{pareto}}$.append($\mathcal{C}'$.at(i));
15     return $\mathcal{C}_{\text{pareto}}$
16 Function GetParetoFront($\mathcal{Q}$, $\mathcal{Z}$, $\mathcal{C}_{\text{pareto}}$)
17     foreach $c \in \mathcal{C}_{\text{pareto}}$ do
18         set c; fsize = $|c.frequencyDomains|$;
19         process mapping to $q \in \mathcal{Q}$ with lowest TNP; calculate $P^{\mathcal{M}}$, $P^{\mathcal{Q}}$ and $t^{\mathrm{e}}$;
20         for $x = 2$, $i = 0$; $i < fsize$; $x$++ do
21             i = Fibonacci(x);
22             if $i > fsize$ then
23                 i = fsize;
24             $\mathcal{Q}_i$ = take all $q \in \mathcal{Q}$ within i frequency domains with lowest TNP;
25             process mapping with minCut($|\mathcal{Q}_i|$); calculate $P^{\mathcal{M}}$, $P^{\mathcal{Q}}$ and $t^{\mathrm{e}}$;
26             process mapping with merge($|\mathcal{Q}_i|$); calculate $P^{\mathcal{M}}$, $P^{\mathcal{Q}}$ and $t^{\mathrm{e}}$;
27     return all non-dominated $c \in \mathcal{C}_{\text{pareto}}$ AND process mappings according to $P^{\mathcal{M}}$, $P^{\mathcal{Q}}$
         and $t^{\mathrm{e}}$;
```

Figure 1 exemplifies that $\mathcal{C}_{\text{pareto}}$ forms already a good approximation, when computing a minCut($|\mathcal{Q}|$) mapping for demonstration purposes. This minCut mapping strategy is explained in the next section.

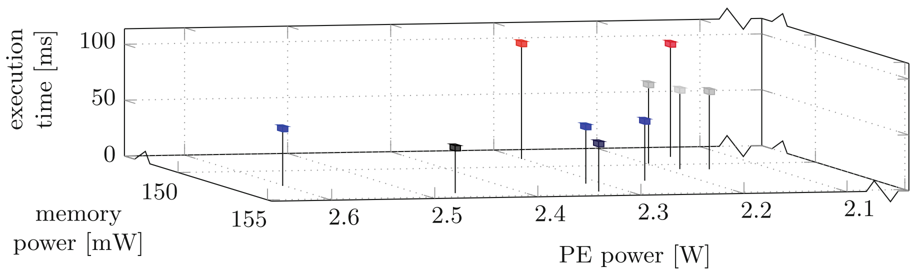

Fig. 1. Process mapping with minCut($|\mathcal{Q}|$) for each $c \in \mathcal{C}_{\text{pareto}}$, *audio filter* and HeMVP

GetParetoFront. MORAM computes three different types of mappings during the Pareto front generation: (i) A graph splitting approach is applied on the basis of minimum cuts, dubbed *minCut*. It focuses on outputting minimum memory power mappings. The KPN graph A of the application is first treated as a single set. After one cut, two sets are produced, which can be mapped to the available PEs. The minimum cut algorithm of [30] is used to keep the FIFO channel communication costs between the new subsets minimised. These cuts are done $|\mathcal{Q}|$ times to have as many subsets as PEs available.

(ii) A graph merging approach is chosen to start from the opposite side than minCut, dubbed *merge*. It is chosen to generate mappings with maximum performance. Each node of the KPN graph A is considered as an individual set. Subsets are grouped if they have high FIFO channel communication cost but do not have a high processing load after being merged. These considerations are necessary to achieve optimal performance with low communication cost. Due to the brevity of this paper, the merge procedure cannot be discussed in detail. In brief, the notion of attraction and repulsion forces acting on the subsets is used. The former occurs for high FIFO channel communication and is calculated on the basis of FIFO channel sizes. Repulsion forces between subsets are high if the processing load of the individual subsets is high. It is based on the ETI value.

(iii) Assigning all processes to the PE which has the lowest TNP is done to get a mapping solution with the slowest execution time and lowest power values. In other words, a corner case of the Pareto front is ensured to be included in the final approximation of the non-dominated set.

The entries of $\mathcal{C}_{\text{pareto}}$ are input to the final Pareto front calculation and considered further (line 17). For each $c \in \mathcal{C}_{\text{pareto}}$, mapping type (iii) is computed (line 19). Type (i) and (ii) are evaluated on the granularity of the frequency domains to save run time and iterate more coarse through the added parallel computation options (lines 20–26). Starting with one domain that hosts PEs with minimum TNP, mapping is done utilising only a few PEs to save power. With every added domain, more PEs are considered. With this procedure, the number of idling PEs, which switch to a low power mode, can be maximised in the beginning. Further, distributing the processes among more PEs offers better performance but causes higher PE and memory power consumption.

In case of a high number of frequency domains, it is not necessary to increase the current frequency domain count linearly, due to Amdahl's law. It describes the theoretical speed-up when increasing the PE count for parallel applications. Approximating resulting speed-up curve requires the most samples in the beginning, as the curve levels out. The best approximation is the Fibonacci series because adding a frequency domain results in several added PEs.

In the end, MORAM computes the final Pareto front based on all $c \in \mathcal{C}_{\text{pareto}}$ and process mappings that are non-dominated for the three objectives (line 27).

5 Experimental Results

The experimental results consist of two case studies to evaluate the quality and performance. The comparison Pareto front is calculated with the R2-EMOA.

The speed-up of MORAM is computed in relation to the R2-EMOA. The solution quality is indicated with the Hypervolume Indicator (HI) to identify which Pareto front is superior. A non-dominated front is considered better if its solutions are well distributed across the objective space and cover a larger area for each objective value. The HI compresses these conditions, i.e. diversity and dominance into one single value. It is the only method mentioned in the literature that achieves these Pareto-compliant conditions as unary indicator [8].

The comparison Pareto front is calculated with the indicator EMOA presented in [10], dubbed R2-EMOA. It has been chosen because R2 indicator based EMOA are proven to be efficient and less computational expensive in objective domains ≥ 3 [9]. Two variants are used. The unconstrained R2-EMOA has a population size of 100 and does 6000 evaluations. The constraint variant is limited to a population with 50 individuals and an iteration count of 600.

Execution times and power estimates are solely computed using the CPN simulation engine, due to sufficient accuracies (3.1 and 3.2). A set of representative parallel applications is used [7]. The number of processes is given in brackets: *audio filter* (11), *JPEG* encoder (24), multiple input multiple output orthogonal frequency division multiplexing *MIMO OFDM* transceiver (36), space-time adaptive processing *STAP* (16), and *sobel filter* (5). In-house implementations complement the benchmark set: an LTE uplink receiver physical layer benchmark *LTE* (19) [28], and a Mandelbrot set computation with 16 *Man16* and 150 *Man150* worker processes round off the benchmark set, Discrete Cosine Transformations DCT (8) typically used in video compression.

Table 1. MORAM HI performance relative to constrained R2-EMOA

	ODROID-XU3	HeMVP
Audio filter	−1.1%	++
DCT	−1.6%	++
JPEG	−2.7%	++
LTE	−3.6%	++
Man150	++	++
Man16	+	++
MIMO OFDM	++	++
Sobel filter	−7.9%	−0.1%
STAP	+	++

+: Better than constrained R2-EMOA
++: Better than unconstrained R2-EMOA

5.1 Case Study: ODROID-XU3

The ODROID-XU3 board [1] is built around the Samsung Exynos-5422 processor with ARM big.LITTLE architecture. The frequency ranges from 200 MHz to 1400 MHz (little) and 2000 MHz (big) in steps of 100 MHz per cluster. The ODROID-XU3 supports two levels of coherent caches. Each core has its own set of private L1 instruction and data caches. Per cluster, a shared L2 cache is deployed, which is connected to a 2 GB LPDDR3 DRAM running at 933 MHz. The operating system takes care of automatically setting the most efficient voltage.

The run time of MORAM ranges between 1.5 s and 138 s. Compared to the constrained R2-EMOA, the minimal speed-up is 27× and 200× on average. The run time numbers are shown in Fig. 2. Table 1 gives an overview of the HI mean performance relative to constrained and unconstrained R2-EMOA. MORAM calculates Pareto fronts that are less than 8% worse compared to the constrained R2-EMOA for half of the benchmarks. In four cases, the R2-EMOA is outperformed. Averaging over all cases, the constrained R2-EMOA is 1% better.

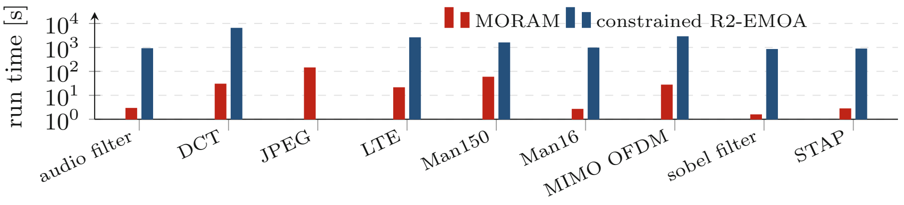

Fig. 2. R2-EMOA and MORAM run times for ODROID-XU3

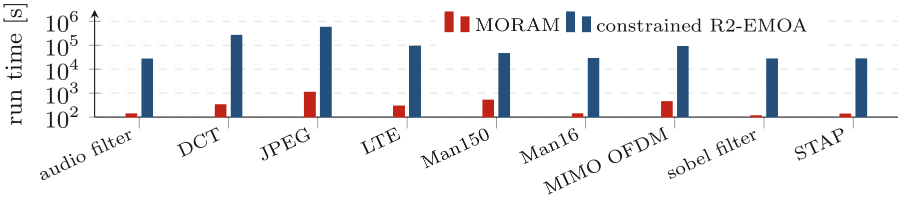

Fig. 3. R2-EMOA and MORAM run times for HeMVP

5.2 Case Study: HeMVP

A SystemC [3] in-house virtual prototype, dubbed HeMVP, models a heterogeneous platform with a hierarchical structure. The platform subsystems consist of either one ARM Cortex-A9 or ADSP Blackfin 609 DSP (BFIN), with private incoherent L1 instruction and data caches. A local memory (1 MB, the same frequency as the PE) is available per subsystem. Four ARM and four BFIN subsystems are combined into a cluster, which also contains a bus and memory (4 MB, 250 MHz). Four clusters are connected globally with a bus, which grants access to shared memory (128 MB, 100 MHz).

The HeMVP has a total of 32 PEs. Two subsystems of same PE type are grouped into a frequency domain, while four share the same voltage domain. The frequency ranges from 200 MHz to 1200 MHz for the ARMs, and from 100 MHz to 500 MHz for the Blackfins. For both, the step size is 100 MHz. This leads to $|\mathcal{C}| = 8.4 \cdot 10^{13}$.

The bare metal runtime environment sets the lowest applicable voltage per voltage domain automatically. Further, it takes care of powering down unused PEs to a clock gated state. Memories that have no data assigned to are powered off entirely. All three memory levels can be used to store the data of FIFO channels. Also, cluster memories host the data structures for synchronisation. The shared memory provides stack, heap and shared code. The memory modelling engine of [24] is used to generate viable power traces. For ARM and BFIN, the power models presented in [20,26] are deployed.

The aforementioned N is set to 10^5 to enable the pre-pruning step, as recommended in [10]. The run times for MORAM and the constrained R2-EMOA are shown in Fig. 3. Due to the larger $|\mathcal{C}|$ and mapping options, MORAM computes between 1.8 min and 18 min. This is at least 88× and on average 278× faster than the constrained R2-EMOA. Further, Table 1 reveals that the heuristic computes Pareto fronts with an HI almost always better than the unconstrained R2-EMOA. On average, MORAM is 4% better.

The reason results from domain knowledge which is explained as follows. The memory assignment of FIFO channel buffers is done implicitly, as explained in Sect. 4. The design of MORAM accounts for the process-to-PE dependent HWC placement. However, the R2-EMOA falls into the category of meta-heuristics. They incorporate none to just a few assumptions about the addressed optimisation problem. This has the advantage that it can be used for a much wider variety of problems. The drawback becomes visible for this MOOP in the form of the missing domain knowledge.

6 Conclusion

This paper proposed a software mapping heuristic approach which solves the three-dimensional optimisation problem of application performance, memory and PE power. Pareto fronts could enable trade-off evaluation and serve as an alternative to established methods for the training of online optimisation algorithms. The applicability and the quality of MORAM were evaluated using two different case studies and a state-of-the-art indicator based EMOA. The heuristic computed Pareto fronts for the ODROID-XU3 and the targeted representative benchmarks at least 80× faster, while having an HI 1% worse compared to the constrained R2-EMOA. Furthermore, testing the heuristic in a highly complex search space scenario, the HeMVP showed a minimum speed-up of 88×. On average, a 4% better HI compared to the unconstrained version was achieved.

References

1. ODROID-XU3. http://odroid.com/dokuwiki/doku.php?id=en:odroid-xu3. Accessed Jan 2020
2. Silexica GmbH. http://silexica.com. Accessed Jan 2020
3. SystemC. http://www.accellera.org/downloads/standards/systemc. Accessed Jan 2020
4. WHITEPAPER - pushing performance: analysis and optimisation of multicore communication with SLX. https://www.silexica.com/resources/#whitepapers-reached. Accessed Jan 2020
5. WHITEPAPER - SLX multi-objective optimisation (MOPT). https://www.silexica.com/resources/#whitepapers-reached. Accessed Jan 2020
6. Aalsaud, A., Shafik, R., Rafiev, A., Xia, F., Yang, S., Yakovlev, A.: Power-aware performance adaptation of concurrent applications in heterogeneous many-core systems. In: Proceedings of ISLPED 2016 (2016)
7. Aguilar, M., Jimenez, R., Leupers, R., Ascheid, G.: Improving performance and productivity for software development on TI multicore DSP platforms. In: EDERC, September 2014
8. Berghammer, R., Friedrich, T., Neumann, F.: Convergence of set-based multi-objective optimization, indicators and deteriorative cycles. Theoret. Comput. Sci. **456**, 2–17 (2012)
9. Brockhoff, D., Wagner, T., Trautmann, H.: 2 indicator-based multiobjective search. Evol. Comput. **23**(3), 369–395 (2015)
10. Führ, G., Hallawa, A., Leupers, R., Ascheid, G., Eusse, J.F.: Multi-objective optimisation of software application mappings on heterogeneous MPSoCs: TONPET versus R2-EMOA. Integration **69**, 50–61 (2019)
11. Ghose, S., Yaglikçi, A.G., Gupta, R., Lee, D., et al.: What your DRAM power models are not telling you: lessons from a detailed experimental study. ACM Meas. Anal. Comput. Syst. **2**(3), 1–41 (2018)
12. Gupta, U., Mandal, S.K., Mao, M., Chakrabarti, C., Ogras, U.Y.: A deep Q-learning approach for dynamic management of heterogeneous processors. IEEE Comput. Archit. Lett. **18**(1), 14–17 (2019)
13. Gupta, U., Patil, C.A., Bhat, G., Mishra, P., Ogras, U.Y.: DyPO: dynamic pareto-optimal configuration selection for heterogeneous MpSoCs. ACM Trans. Embed. Comput. Syst. **16**(5s), 123:1–123:20 (2017)
14. Ha, S., Jung, H.: HOPES: programming platform approach for embedded systems design. In: Ha, S., Teich, J. (eds.) Handbook of Hardware/Software Codesign, pp. 951–981. Springer, Dordrecht (2017). https://doi.org/10.1007/978-94-017-7267-9_1
15. Holzkamp, O.: Memory-aware mapping strategies for heterogeneous MPSoC systems. Ph.D. thesis, Technical University of Dortmund, Germany (2017)
16. Jung, M., Mathew, D.M., Zulian, F., Weis, C., Wehn, N.: A new bank sensitive DRAMPower model for efficient design space exploration. In: Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS) (2016)
17. Kahn, G.: The semantics of a simple language for parallel programming. In: Proceedings of Information Processing, Stockholm, Sweden, August 1974
18. Kang, S.H., Yang, H., Schor, L., Bacivarov, I., Ha, S., Thiele, L.: Multi-objective mapping optimization via problem decomposition for many-core systems. In: IEEE 10th Symposium on Embedded Systems for Real-time Multimedia, October 2012

19. Leupers, R., Aguilar, M.A., Eusse, J.F., Castrillon, J., Sheng, W.: MAPS: a software development environment for embedded multicore applications. In: Ha, S., Teich, J. (eds.) Handbook of Hardware/Software Codesign, pp. 917–949. Springer, Dordrecht (2017). https://doi.org/10.1007/978-94-017-7267-9_2
20. Onnebrink, G., et al.: Black box power estimation for digital signal processors using virtual platforms. In: RAPIDO 2016 Workshop (2016)
21. Onnebrink, G., et al.: DVFS-enabled power-performance trade-off in MPSoC SW application mapping. In: SAMOS, July 2017
22. Quan, W., Pimentel, A.D.: A hybrid task mapping algorithm for heterogeneous MPSoCs. ACM Trans. Embed. Comput. Syst. **14**, 1–25 (2015)
23. Reddy, B.K., Singh, A.K., Biswas, D., Merrett, G.V., Al-Hashimi, B.M.: Inter-cluster thread-to-core mapping and DVFS on heterogeneous multi-cores. IEEE Trans. Multi-Scale Comput. Syst. **4**(3), 369–382 (2018)
24. Rudolf, J., Strobel, M., Benz, J., Haubelt, C., Radetzki, M., Bringmann, O.: Automated sensor firmware development - generation, optimization, and analysis. In: Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV) (2019)
25. Schranzhofer, A., Chen, J.J., Thiele, L.: Dynamic power-aware mapping of applications onto heterogeneous MPSoC platforms. IEEE Trans. Industr. Inf. **6**, 692–707 (2010)
26. Schuermans, S., Leupers, R.: Power Estimation on Electronic System Level using Linear Power Models. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-01875-7
27. Singh, A.K., Shafique, M., Kumar, A., Henkel, J.: Mapping on multi/many-core systems: survey of current and emerging trends. In: Proceedings of Design Automation Conference (DAC) (2013)
28. Själander, M., McKee, S., Brauer, P., Engdal, D., Vajda, A.: An LTE uplink receiver PHY benchmark and subframe-based power management. In: Performance Analysis of Systems and Software (ISPASS) (2012)
29. Stefanov, T., Pimentel, A., Nikolov, H.: DAEDALUS: system-level design methodology for streaming multiprocessor embedded systems on chips. In: Ha, S., Teich, J. (eds.) Handbook of Hardware/Software Codesign, pp. 983–1018. Springer, Dordrecht (2017). https://doi.org/10.1007/978-94-017-7267-9_30
30. Stoer, M., Wagner, F.: A simple min-cut algorithm. J. ACM **44**(4), 585–591 (1997)
31. Zhang, Y., Gong, D.-W., Cheng, J.: Multi-objective particle swarm optimization approach for cost-based feature selection in classification. IEEE/ACM Trans. Comput. Biol. Bioinf. (TCBB) **14**(1), 64–75 (2017)
32. Zhang, Y., Gong, D.-W., Ding, Z.: A bare-bones multi-objective particle swarm optimization algorithm for environmental/economic dispatch. Inf. Sci. **192**, 213–227 (2012)