Flexible Spatio-Temporal Energy-Efficient Runtime Management

Robert Khasanov
Chair for Compiler Construction
Technische Universität Dresden
Dresden, Germany
robert.khasanov@tu-dresden.de

Marc Dietrich
Chair for Compiler Construction
Technische Universität Dresden
Dresden, Germany
marc.dietrich@mailbox.tu-dresden.de

Jeronimo Castrillon
Chair for Compiler Construction
Technische Universität Dresden
Dresden, Germany
jeronimo.castrillon@tu-dresden.de

***Abstract*—Heterogeneous multi-core architectures, such as Arm's big.LITTLE and DynamIQ, feature multiple core types with the same ISA but varied performance-energy characteristics. These are increasingly adopted in embedded systems as they enable dynamic application mapping, balancing performance with energy efficiency. While Hybrid Application Mapping (HAM) approaches have gained popularity in systems running dynamic workloads, most solutions yield spatial mappings and neglect application migrations in output schedules, substantially limiting the solution space. This work introduces STEM and FFEMS, two algorithms utilizing the temporal aspect with job reconfigurations to generate "flexible" spatio-temporal mappings. STEM leverages Memetic Algorithms (MAs), while FFEMS uses fast greedy heuristics. Our evaluation on two heterogeneous multi-core platform models demonstrates that the flexible structure of the spatio-temporal mappings significantly improves the schedulability. On workloads from automotive and multimedia domains, STEM finds the most energy-efficient solutions, but its large overhead makes it unsuitable for use in runtime systems. In contrast, FFEMS exhibits an outstanding balance between performance and runtime overhead: Given similar runtime overhead as MMKP-MDF, the state-of-the-art approach, FFEMS schedules up to 16 % more test cases. Its "tail-switch" optimization further improves energy efficiency, though with increased overhead, which is still acceptable within runtime systems.**

***Index Terms*—resource management, energy-efficiency, spatio-temporal mapping.**

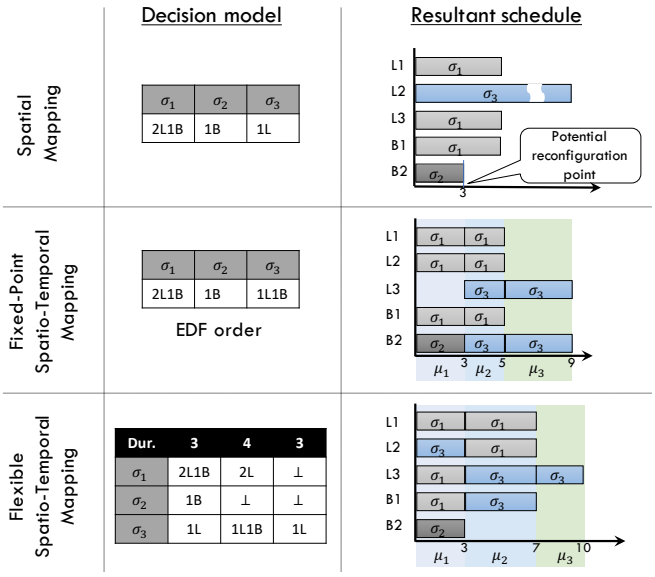

Fig. 1. Overview of multi-application mapping models. Spatial mapping selects one configuration per application, ensuring simultaneous selection. Fixed-point spatio-temporal mapping also picks one configuration per application and, with a specified application order, finalizes a spatio-temporal mapping. Flexible spatio-temporal mapping designates mapping segments and their durations, ensuring all specified configurations (with $\perp$ denoting idle) within the segment can coexist.

I. Introduction

In today's embedded systems, a trend towards heterogeneous multi-core systems is increasingly evident. Arm's big.LITTLE [1], for instance, includes two types of cores — high-performance and high-efficiency ones — grouped into two corresponding clusters. Its successor, DynamIQ [2], goes beyond the rigid cluster structure and provides flexibility in multi-core processor design, e.g., the recent DynamIQ Shared Unit-120 [3] combining up to three different types in a single cluster. The core types in such systems share the same Instruction Set Architecture (ISA), yet they differ in implementation, and thus, in performance-energy characteristics. This allows the system to dynamically map and migrate applications onto resources, effectively balancing between performance and energy efficiency, and therefore to *adapt* to the dynamic workload in an *energy-efficient* manner.

Optimizing the mapping of applications onto heterogeneous multi-core systems is a known *NP-hard* problem, with the number of possible mappings growing exponentially with the application and the platform sizes. Among various strategies [4], *Hybrid Application Mapping* (HAM) approaches [5] are most suitable for systems executing dynamic workloads that require predictable and energy-efficient solutions simultaneously. HAM strategies offload compute-intensive calculations to the design time, generating a Pareto-optimal set of mappings (called *operating points*) for each application. Then, at runtime, they use pre-generated mappings of each active application and transform them into a single consolidated multi-application mapping. These multi-application mappings are merged so that the applications do not share CPU resources to ensure the predictability of their final performance and energy consumption [6].

Conventional runtime systems in HAM are commonly based on efficient heuristics and produce *spatial* mappings [7]–[9], specifying a spatial core assignment for each active application onto the computing resources (see Fig. 1). However, these mappings are optimized solely for the current applications and do not anticipate future changes. When any application exits, processing resources are released, and a new spatial mapping is generated. This process forms a sequence of mappings, each

optimized for a specific moment in time.

The decision model could be further enhanced by adding a temporal component, forming a *spatio-temporal* mapping, or *schedule*. Such schedules may take into account expected changes in the workload (e.g., the application finishes its execution) and generate more efficient schedule plans. However, with this temporal component, the design space also increases exponentially. While `MMKP-MDF` [10], [11] does provide such spatio-temporal mappings within the milliseconds range, it inherently employs a *fixed-point* model (Fig. 1), disregarding job migrations and thus limiting its solution space.

This paper introduces approaches that generate *flexible* spatio-temporal mappings in (firm) real-time systems. These mapping algorithms consider application reconfigurations in their output schedule plans and, therefore, better adapt them to dynamic workloads. We present two approaches that generate these flexible mappings. The first, **STEM** (Spatio-Temporal Evolutionary Mapping), leverages Memetic Algorithms (MAs) [12]. MAs combine Genetic Algorithms (GAs) with diverse techniques such as local search and problem-specific approaches [13]. Our motivation in choosing MAs is twofold: first, we aim to identify near-optimal solutions to evaluate the efficacy of the fast algorithm; second, we research the impact of knowledge-guided heuristics in the search for optimal spatio-temporal mappings. The second, **FFEMS** (Fast Flexible Energy-Minimizing Scheduler), utilizes fast greedy heuristics. Its basic version schedules in 10-100ms, while an energy-optimized variant marginally increases overhead but remains below a second.

II. Related work

The use of Hybrid Application Mapping (HAM) began in the 2000s with pioneers like Yang et al. [14] who proposed it for real-time systems. Typically, HAM employs Genetic Algorithms (GAs) during Design Space Exploration (DSE) to generate a set of partial or complete (Pareto-optimal) mappings [15]–[17]. However, due to overhead, faster algorithms for operating point selection are favored at runtime. Some runtime algorithms iteratively map applications onto the platform one by one [17], [18], while others select the operating points for all applications in a joint manner [7]–[9], often expressed via a Multiple-choice Multidimensional Knapsack Problem (MMKP) [19].

Recently, Spieck et al. [20] put forth a notable contribution by presenting a scenario-aware HAM methodology. They demonstrated how DSE also classifies the input stimuli into scenarios with similar energy-performance characteristics. The runtime manager then identifies the scenarios and mediates resources using the Lagrangian relaxation heuristic on MMKP.

The aforementioned runtime approaches only generate *spatial* multi-application mappings. Mukherjee et al. [21] explored spatio-temporal job scheduling in heterogeneous data centers. Although their algorithm SCINT, based on GAs, finds optimal solutions, it is time-consuming and does not consider the varying resource-performance requirements of different application mappings. Moreover, this algorithm operates under purely stochastic (genetic) operators, without the incorporation of knowledge-based heuristics. In contrast, `MMKP-MDF` [10], [11], a fast greedy heuristic-based algorithm (as implied by its name, also based on MMKP), can generate spatio-temporal schedules within a millisecond range. However, as mentioned in the Introduction, its fixed-point decision model considerably restricts the solution space. To the best of our knowledge, no existing mapping algorithm works within the millisecond range and generates *flexible* spatio-temporal mappings.

III. System model and problem formulation

Hybrid application mapping splits the mapping generation process into design-time and runtime stages. At design time, DSE generates Pareto-optimal mappings for each application *in isolation*. These spatial mappings, enhanced with non-functional characteristics, are called operating points. At runtime, the Resource Manager (RM) constructs a spatio-temporal multi-application mapping using one or several operating points for each active application, adapting them to ensure spatial isolation at every mapping segment. In the following, we formalize the system model and formulate the optimization problem, partially following the notation in [10].

A. System model

Architecture model: We represent a heterogeneous *platform* $\mathcal{P}$ with m resource types and core counts by the vector $\mathcal{P}[\vec{\Theta}] = (\Theta_1, \ldots, \Theta_m)$. While all core types share the same ISA, they each exhibit different performance-energy characteristics. The platform executes (multi-threaded) applications capable of malleable reconfigurations [22], core migration, and preemption.

Operating point model: For each application λ, the RM receives the set of operating points Φ^λ. Each *operating point* $\phi_i \in \Phi$ is denoted by required resources $\vec{\theta}$, (worst-case) execution time τ, and energy consumption ξ, i.e., $\phi = \phi\langle\vec{\theta}, \tau, \xi\rangle$. Operating points are Pareto-filtered, meaning each point is better than any other in at least one parameter, such as fewer cores of a particular processor type θ, lower execution time τ, or energy consumption ξ.

Workload model: Upon a new request's arrival, the RM activates and works with a set of requests, Σ, including new and unfinished requests. Each job[1] $\sigma \in \Sigma$ is denoted by its arrival time α, the (relative) deadline δ, the application λ, and the remaining progress ratio $\rho \in [0, 1]$, i.e., $\sigma = \sigma\langle\alpha, \delta, \lambda, \rho\rangle$.

Decision model: The RM aims to construct a *spatio-temporal mapping*, defined as a sequence of mapping segments M_i, i.e., $K = [M_1, M_2, ..., M_{|K|}]$. Each mapping segment M has a duration $M[\Delta]$ and assigns each job σ an operating point $M[\sigma] = \phi^* \in \Phi^{\sigma[\lambda]} \cup \perp$, where $\perp$ represents no mapping. Fig. 1 depicts an example of a flexible spatio-temporal mapping model.

B. Optimization problem

The RM's goal is to minimize overall energy consumption while ensuring Quality of Services (QoS). Using the introduced notation, we define the problem as follows:

$$\text{minimize } E(K) = \sum_{M \in K} \sum_{\sigma \in \Sigma} M[\sigma][\xi] \cdot P(M, \sigma) \qquad (1)$$

where $P(M, \sigma) = \frac{M[\Delta]}{M[\sigma][\tau]}$ is a progress ratio of a job σ during the mapping segment M.

[1]For simplicity, we use the terms "jobs" and "requests" interchangeably.

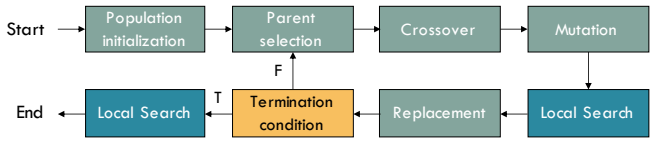

Fig. 2. The STEM algorithm flow.

The solution must satisfy the following constraints:

1) *Resource constraint* ensures resources required for each mapping segment do not exceed the available resources:

$$M[\vec{\theta}] = \sum_{\sigma \in \Sigma} M[\sigma][\vec{\theta}] \leq \mathcal{P}[\vec{\Theta}], \forall M \in K. \quad (2)$$

2) *Deadline constraint* ensures QoS:

$$\forall \sigma \in \Sigma, \widetilde{n} = \max i : M_i[\sigma] \neq \perp, T^{\widetilde{n}}(K) \leq \sigma[\delta], \quad (3)$$

where $T^{\widetilde{n}}(K) = \sum_{i=1}^{\widetilde{n}} M_i[\Delta]$ is the total duration of first $\widetilde{n}$ segments $M_i \in K$.

3) *Completion constraint* ensures job completion:

$$\sum_{M \in K} P(M, \sigma) = \sigma[\rho], \forall \sigma \in \Sigma. \quad (4)$$

Note that the problem does not account for the runtime overhead associated with preemption and switching operating points. While some platforms, including the one evaluated in our study, exhibit negligible runtime overhead [6], this is not true for all platforms. We opted for a simpler formulation by excluding overhead. Nevertheless, overhead can be easily considered in our proposed algorithms.

IV. Spatio-Temporal Evolutionary Mapping

Now we introduce the Spatio-Temporal Evolutionary Mapping (STEM), which uses Memetic Algorithms (MAs) for creating spatio-temporal mappings, achieving the combined benefits of a Genetic Algorithm (GA) and knowledge-guided heuristics. Fig. 2 illustrates the STEM algorithm flow. In the following, we present the components of STEM, including chromosome representation and its evaluation, population initialization, genetic operators, local search methods, and survival selection.

A. Chromosome Representation and Evaluation

In evolutionary algorithms, a chromosome (sometimes referred to as an individual) represents a potential solution to the problem at hand. In the context of STEM, a chromosome embodies a flexible spatio-temporal mapping, as depicted in Fig. 1. During evolution, redundancies can arise, for instance, when jobs complete execution mid-segment or when operating points are assigned to completed jobs. To manage these redundancies, we discard mappings post-job completion and take any discrepancy into account during the fitness evaluation (therefore, relaxing Eq. 4).

Fitness Evaluation: Chromosomes are classified into three categories based on compliance with resource and deadline constraints (Eq. 2 and 3): (1) valid chromosomes that fulfill all constraints, (2) chromosomes that satisfy resource constraints but violate deadlines, and (3) chromosomes that violate resource constraints.

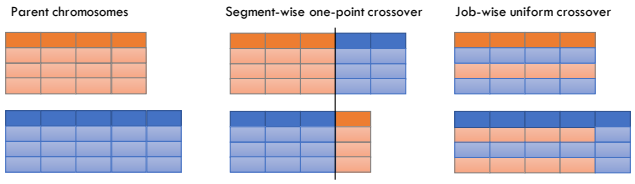

Fig. 3. Crossover operators applied to parent individuals (left): segment-wise one-point (center) and job-wise uniform (right).

Fitness is a tuple, with the first value indicating the category and the second value measuring group-specific metrics — lower values indicate better solutions. For valid chromosomes, this reflects energy consumption and a genotype-to-phenotype discrepancy. For the second category, it is the average deadline violation as a fraction of the deadline. For the third, it reflects both the average deadline violation and the average overuse of processor types per schedule segment.

B. Population Initialization

STEM generates an initial population of $P = 90$ individuals using a structured approach rather than a purely random generation. This method helps to avoid the creation of an excessive number of unfit solutions. First, we randomly determine the number of segments from the range $[1..2 \cdot |\Sigma|]$. Next, segment durations $M[\Delta]$ are generated using a normal distribution $\mathcal{N}(m_d, (m_d/2)^2)$, where m_d is the maximum deadline divided by the number of segments (the value is rounded up to the nearest valid duration). For each segment, we randomly generate a number $p \in [1.. \min(|\Sigma|, |\mathcal{P}[\vec{\Theta}]|)]$ and randomly sample p jobs. The algorithm then randomly selects an operating point for these jobs, while assigning $\perp$ to all others.

C. Parent selection and genetic operators

In each iteration, STEM selects a pair of parent individuals, upon which crossover and mutation operators are applied to produce a pair of offspring.

Parent Selection: STEM uses *exponential ranking selection* for parent selection [23]. In this strategy, the population is first sorted incrementally, placing the fittest individuals at the beginning: $fit_1 \leq fit_2 \leq \cdots \leq fit_s$ The selection probability of ind_i is given by $p_{sel}(ind_i) = f^{i-1}/c$, where $f < 1$ is typically set close to 1, e.g., 0.97 in our implementation, and c is a normalizing coefficient. We then employ *stochastic universal sampling* [12] to select two individuals as parents from the sorted population.

Crossover: The selected pair undergoes crossover with a probability of $p_c = 0.7$. STEM randomly chooses one of two crossover operators shown in Fig. 3. Given l is the smallest number of segments in the parent chromosomes, the segment-level crossover applies a one-point crossover technique, cutting the parent chromosome at a random point $p \in [1..l]$ and swapping subsequent segments. The job-level crossover decides for each job randomly if its mappings should be swapped individually (i.e., *uniform crossover*). If so, the swap is performed for the first l segments.

Mutation: After crossover, each offspring undergoes mutation with a probability of $p_m = 0.6$. We define three mutation operators for this step.

The first operator modifies the schedule structure by (a) swapping two random segments, (b) inserting a new segment at a random position, or (c) removing a random segment and redistributing its duration. The other two mutation operators alter specific chromosome values. One changes a randomly selected segment's duration, and the other changes the operating point index of a random job within a random segment, selecting the $\perp$ value with a $p_\perp = 0.5$ chance.

D. Local Search Methods

After generating a pair of offspring through genetic operators, the *memetic* part of the algorithm – comprising several local search heuristics – comes into play. These heuristics, leveraging problem-specific knowledge, accelerate the search for optimal solutions. The selection of methods is based on their efficiency and runtime overhead, and individuals are subjected to different methods based on their constraint violations.

Individuals violating resource constraints undergo the following refinement method with a probability $p_{r3} = 0.5$:

Resource Overuse Reduction: This method targets the segment with the highest resource overuse. It assigns the mapping of each job to $\perp$ and selects the one with the best fitness.

Other individuals undergo refinement at a probability $p_{r1} = p_{r2} = 0.8$, randomly selecting one of the these methods:

Chromosome Simplification: This method reduces a genotype-to-phenotype gap by removing false-active segments. It identifies the last active segment for each job, marks subsequent segment mappings as $\perp$, and erases idle segments.

Segment Manipulations: These methods explicitly modify the segment count. The first method merges two segments into one in one of two ways: either by merging two segments with identical operating points or by removing the shortest duration segment and adding its duration to a longer segment. The second method splits segments where a job finishes mid-segment. Following a split, it applies chromosome simplification and alters job mappings in the second part of the split segment.

Segment Duration Adjustment: This method adjusts segment durations within permissible bounds, with increments or decrements by a power of two to minimize exploration.

Front Propagation of Operating Points: This method collects all operating points used in the current individual and attempts to use them in earlier segments where the job has a $\perp$ mapping. Preference is given to jobs with the most significant deadline violation or highest energy consumption.

E. Survivor Selection and Termination

At the end of each iteration, the survivor selection strategy determines which individuals from the merged population can make it into the next generation. We adopt *round-robin tournament selection* [12]: For each individual, the method randomly selects $q = 8$ competitors, compares their fitness levels, and assigns a score based on the number of competitors the individual wins. The two individuals with the lowest scores are removed from the population, with ties resolved randomly.

STEM terminates when the maximum number of iterations is reached. However, the termination condition could be further refined, e.g., by terminating if no significant improvement is observed for a certain number of generations.

V. Fast Flexible Energy-Minimizing Scheduler

This section introduces an alternative resource management algorithm, Fast Flexible Energy-Minimizing Scheduler (FFEMS), which schedules jobs in an Earliest Deadline First (EDF) manner. FFEMS ensures minimized energy consumption by scheduling each job using mappings from an incrementally expanding Candidate Mappings Set (CMS).

Algorithm 1 FFEMS

Input: Σ, $\mathcal{P}[\vec{\Theta}]$, Φ^σ for each $\sigma \in \Sigma$
Output: Schedule K

1: Initialize an empty schedule K
2: Sort jobs in $\sigma \in \Sigma$ by Earliest Deadline First (EDF)
3: **for** each job $\sigma \in \Sigma$ **do**
4: Sort the operating points $\phi \in \Phi^\sigma$ by energy
5: CMS $\leftarrow$ InitCMS(σ, Φ^σ)
6: $s \leftarrow False$
7: **while** $\neg s \wedge$ CMS $\neq \varnothing$ **do**
8: $t_s \leftarrow 0$, $\rho \leftarrow \sigma[\rho]$, $\delta_{miss} \leftarrow False$
9: **for** each segment $M \in K$ **do**
10: $K' \leftarrow$ ScheduleTail(K, σ, CMS, M, ρ)
11: **if** $K' \neq \varnothing$ **then**
12: $K \leftarrow K'$, $\rho \leftarrow 0$
13: **break**
14: **if** $\sigma[\delta] \leq t_s + M_i[\Delta]$ **then**
15: $\delta_{miss} \leftarrow True$
16: **break**
17: $\phi^* \leftarrow \text{argmax}_{\text{CMS}}\{\phi[\tau] \mid \phi[\vec{\theta}] + M[\vec{\theta}] \leq \mathcal{P}[\vec{\Theta}]\}$
18: $M[\sigma] \leftarrow \phi^*$
19: **if** $\phi^* \neq \perp$ **then**
20: $\rho \leftarrow \rho - \frac{M[\Delta]}{\phi^*[\tau]}$
21: **if** $\delta_{miss} = False$ **then**
22: **if** $\rho > 0$ **then**
23: $K' \leftarrow$ ScheduleTail(K, σ, CMS, $\varnothing$, ρ)
24: **if** $K' \neq \varnothing$ **then**
25: $K \leftarrow K'$, $\rho \leftarrow 0$
26: **if** $\rho = 0$ **then**
27: $s \leftarrow True$
28: **break**
29: **if** CMS $= \Phi^\sigma$ **then**
30: CMS $\leftarrow \varnothing$
31: **else**
32: CMS $\leftarrow$ IncrementCMS(CMS, Φ^σ)
33: **if** $\neg s$ **then**
34: **for** all segment $M \in K$ **do**
35: $M[\sigma] \leftarrow \perp$
36: **return** K

Algorithms 1 and 2 detail the operation of FFEMS. Initially, jobs are sorted in EDF order (1:2)[2], with the operating points of each job by energy consumption (1:4). The CMS is initialized to include the first and all lower-energy mappings that allow the job to finish within its deadline (1:5). The CMS limits the energy consumption of the job. If the algorithm fails to schedule the job using the current CMS, it extends

[2]In this section, algorithm lines are referenced as `A:L`, where `A` is the algorithm number and `L` is the line number.

the CMS by including the next mapping from the sorted list, thereby gradually increasing the energy budget of the job.

Algorithm 2 ScheduleTail

Input: Schedule K, $\mathcal{P}[\vec{\Theta}]$, σ, CMS, Start segment M (or $\varnothing$ for append), Remaining ratio ρ
Output: New schedule K, or $\varnothing$ if unsuccessful

1: $\phi \leftarrow$ FINDMAPPINGFORTAIL(K, $\mathcal{P}[\vec{\Theta}]$, σ, CMS, M, ρ)
2: **if** $\phi = \perp$ **then**
3: **return** $\varnothing$
4: $t_r \leftarrow \phi[\tau] \cdot \rho$
5: **for** each segment $M_i \in [M..M_{|K|}] \subset K$ **do**
6: **if** $M_i[\Delta] \leq t_r$ **then**
7: $M_i[\sigma] \leftarrow \phi$, $t_r \leftarrow (t_r - M_i[\Delta])$
8: **else**
9: $M', M'' \leftarrow$ SPLIT(K, M_i, t_r)
10: $M'[\sigma] \leftarrow \phi$, $t_r \leftarrow 0$
11: **break**
12: **if** $t_r > 0$ **then**
13: $M' \leftarrow$ APPEND(K, t_r), $M'[\sigma] \leftarrow \phi$
14: **return** K

During scheduling, for each job and its corresponding CMS, the scheduler iterates over the current schedule's segments. It first attempts to schedule the job's "tail" (1:10), seeking a single mapping from the CMS that can be utilized until job completion. For each potential mapping, the scheduler checks for sufficient free resources from the start time of the segment to the job's potential end time. This search corresponds to a call to `FindMappingForTail` (2:1). If a suitable mapping is found, it is used until job completion, with segments appended (2:13) or split (2:9) as necessary.

If no suitable mapping till the job completion is found, the scheduler selects the fastest mapping for the current segment, anticipating a more energy-efficient switch in subsequent segments (1:17). If the job remains incomplete at the end of the scheduling plan, a tail mapping segment is appended (1:23).

If the job cannot be scheduled with the current CMS, it is extended by the next mapping (1:32). If scheduling remains impossible after the last iteration, the job is rejected (1:34).

In a worst-case scenario, the time complexity of FFEMS is $O(|\Sigma| \cdot |\Phi|^2 \cdot |K|)$. This derives from the iterative process over all jobs in the set Σ, each operating point in the set Φ during the CMS incrementation, each segment of the schedule K, and each operating point within the CMS during tail scheduling.

A. Tail-Switch Optimization

FFEMS's energy efficiency can be improved using "tail-switching". If a power-intensive configuration is initially selected in `ScheduleTail`, the system can switch to a slower but more energy-efficient mapping at a later point This optimization entails iterating over all mapping pairs, ϕ_1 and ϕ_2, determining an optimal switch point from a power-intensive configuration (ϕ_1) to an energy-efficient one (ϕ_2), ensuring deadline constraint. The pair minimizing energy use is selected. However, this adds an extra level of computational complexity, resulting in a time complexity of $O(|\Sigma| \cdot |\Phi|^3 \cdot |K|)$.

TABLE I
BOUNDS OF FACTOR RANGES USED DURING WORKLOAD GENERATION.

Platform	Deadline Level	Factor Range Bounds Lower	Upper
4B4L	Weak	$1.5 + 0.1 \cdot \|\Sigma\|$	$3 + 0.1 \cdot \|\Sigma\|$
	Tight	1	$1 + 0.3 \cdot \|\Sigma\|$
8B8L	Weak	$1 + 0.1 \cdot \|\Sigma\|$	$1.5 + 0.1 \cdot \|\Sigma\|$
	Tight	1	$1 + 0.1 \cdot \|\Sigma\|$

VI. EVALUATION

This section provides a comparative evaluation of our proposed approaches with existing state-of-the-art solutions. Each algorithm was assessed on two heterogeneous platform models in terms of success rate, energy efficiency, and overhead.

A. Experimental setup

Our STEM and FFEMS approaches were implemented and evaluated in the Mocasin prototyping tool [24]. The evaluations were conducted on two platform models: the Odroid XU4 with an Exynos 5422 big.LITTLE chip featuring four Cortex-A15 cores and four Cortex-A7 cores, running at 1.8 GHz and 1.5 GHz, respectively. We denote it as `4B4L`. The second, denoted `8B8L`, is a larger system akin to the Odroid XU4 but with double the cores — eight big and eight little.

Application models: In our experiments, we utilized three dataflow applications from the automotive and multimedia domains: *speaker recognition* [25], *audio filter* [6], and a *pedestrian recognition* algorithm [10]. For the 4B4L platform, operating points were obtained by benchmarking the real platform, with the number of operating points ranging from 28 to 36 [10]. The 8B8L platform's operating points were generated using a genetic algorithm in Mocasin, with 40 operating points selected using the k-means method [26].

Workload generation: Test cases are represented as tables of requests, each containing an application, progress ratio, and deadline. We generated 2000 cases per platform, altering the job number from 1 to 10 in the request table, providing 200 cases per job count. Half of these tests featured weak deadlines, and the other half had tight deadlines, thus facilitating observation under more stress-intensive situations. Each test randomly assigned an application to each job and allocated progress ratios between 0 and 0.9 (with the first job set to 0, emulating a newly arrived job). Deadlines were determined by first selecting a random configuration, calculating the remaining time, based on the configuration and remaining progress ratio, and then multiplying it by a factor. The factor was randomly chosen within a range defined by the deadline level and request number, as detailed in Table I.

Evaluated algorithms: In our evaluation, we tested several variations of the proposed algorithms. For STEM, we considered four variations: $\texttt{STEM}^{\texttt{100K}}$ (100K MAs iterations), $\texttt{STEM}^{\texttt{500K}}$ (500K iterations), $\texttt{STEM}^{\texttt{5M}}$ (5M iterations), and $\texttt{STEM}^{\texttt{500K}}_{\texttt{GA}}$ (omitting the memetic part of the algorithm to assess the effect of knowledge-guided heuristics). For FFEMS, we included a basic version, `FFEMS`, and its derivative, $\texttt{FFEMS}^{\texttt{TS}}$, which applies tail-switch optimization. Additionally, we assessed two state-of-the-art approaches, namely `MMKP-LR` [9] and `MMKP-MDF` [10]. The former generates

TABLE II
SCHEDULER RESULTS FOR DIFFERENT PLATFORMS AND DEADLINES.

Scheduler	4B4L					8B8L				
	Weak Deadlines		Tight Deadlines		Avg.	Weak Deadlines		Tight Deadlines		Avg.
	Suc. R.	Rel. Energy	Suc. R.	Rel. Energy	Overhead	Suc. R.	Rel. Energy	Suc. R.	Rel. Energy	Overhead
MMKP-LR	94.6 %	1.2981	71.0 %	1.2442	447.4 ms	97.9 %	1.1552	81.0 %	1.2103	14.4 ms
MMKP-MDF	96.8 %	1.0260	75.6 %	1.0755	9.3 ms	98.9 %	1.0746	80.7 %	1.0852	10.9 ms
STEM^{100K}	99.8 %	1.0329	88.5 %	1.0341	65.6 s	99.4 %	1.0261	88.4 %	1.0292	68.9 s
STEM_{GA}^{500K}	99.2 %	1.0672	80.5 %	1.0696	216 s	97.9 %	1.0396	78.6 %	1.0494	217 s
STEM^{500K}	100 %	1.0180	91.7 %	1.0220	320 s	99.8 %	1.0163	92.1 %	1.0213	342 s
STEM^{5M}	100 %	1.0070	94.4 %	1.0071	3376 s	99.9 %	1.0087	94.4 %	1.0113	3671 s
FFEMS	100 %	1.0376	91.6 %	1.0982	4.8 ms	100 %	1.0442	93.6 %	1.0784	5.5 ms
FFEMS^{TS}	100 %	1.0270	91.8 %	1.0649	15.6 ms	100 %	1.0150	94.3 %	1.0445	17.2 ms

spatial mappings using the Lagrangian Relaxation algorithm (with 200 iterations), we use it to construct the schedule segment-by-segment. The latter constructs fixed-point spatio-temporal mappings.

B. Success rate and energy-efficiency

We evaluated the schedulers based on success rate, defined by the percentage of successfully scheduled test cases, and relative energy consumption, defined as the geometric mean of relative energies compared to the minimum energy value across considered schedules.

Impact of the knowledge-guided heuristics in STEM: Table II highlights that the incorporation of knowledge-guided heuristics in STEM greatly enhanced both the scheduling rate and energy efficiency. This improvement is especially prominent in test scenarios with tight deadlines, on which STEM^{500K} scheduled up to 13.5 % more test cases than its counterpart, STEM_{GA}^{500K}. Noteworthy, STEM with just 100K MA iterations yields outperforms the version with 500K GA iterations.

However, in spite of the inclusion of the local search heuristics, STEM still requires a substantial number of iterations. Specifically, STEM^{100K} trails FFEMS in terms of success rate by as much as 5 %. Given its enormous overhead, it is evident that deploying STEM at runtime is not feasible. In the following, we focus on STEM^{5M}, as this version is most indicative of the algorithm's peak potential in discovering optimal solutions.

Impact of the mapping decision model: The choice of decision model determines the efficiency of scheduling algorithms. The MMKP-LR approach, which solely generates spatial mappings, often yields suboptimal solutions in both success rate and energy efficiency. Its performance varies with platform size and worsens under increased resource pressure: under tight deadlines, it achieves a similar success rate as MMKP-MDF on 8B8L, but underperforms by 4 % on the smaller 4B4L.

In contrast, MMKP-MDF, utilizing the fixed-point spatio-temporal mapping model, exhibits a notable improvement in energy efficiency, overshadowing MMKP-LR by a significant 26.5 % on 4B4L.

Nevertheless, the peak of performance is achieved by our novel methods, FFEMS and STEM, which harness the full flexibility of spatio-temporal mappings. As depicted on Table II, FFEMS significantly outperform the state-of-the-art approaches: under weak deadlines, it schedules all test cases, and under tight ones, FFEMS schedules 16 % more test cases on 4B4L and 12.9 % on 8B8L. As Fig. 4 depicts, the difference in success rate grows with the number of applications, peaking at 25 % with ten applications on 4B4L. The FFEMS^{TS} variant, with the tail-switch optimization, shows a comparable success rate but improves on energy efficiency, saving up to 3.4 % more energy than FFEMS. Interestingly, FFEMS algorithms even outperform STEM^{5M} on 8B8L in terms of success rate, particularly with a bigger number of applications, as shown in Fig. 4. This observation might indicate that STEM performance declines with the platform and application sizes.

Fig. 4 (bottom left) presents the relative energy consumption across test scenarios using a monotonic curve layout. In this figure, relative energy values for all test cases are arranged in ascending order. The curve's rightmost endpoint indicates the success rate, and its contour provides insights into the distribution of relative energy values. These curves illustrate the differences between the decision models.

The curve corresponding to MMKP-LR (employing the spatial mapping model) reveals that for most test cases, it fails to select the most energy-efficient configurations for spatial mapping. This behavior can be attributed to MMKP-LR's inability to consider postponing certain job executions, leading it to opt for less energy-efficient configurations. In contrast, MMKP-MDF's curve is more aligned with the ideal energy values, showing a slight improvement in the success rate. This improvement is explained by a more relaxed constraint, allowing the selection of the configurations otherwise infeasible within a single mapping segment. Lastly, the adoption of flexible spatio-temporal mappings significantly improves the success rate. This improvement is attributed to the flexible decision structure, which permits jobs to be dynamically reconfigured to meet their deadlines.

C. Scheduling overhead

Fig. 4 (bottom right) illustrates the execution times of different algorithms through box plots. The results indicate an increase in scheduling overhead with the number of applications for all schedulers. MMKP-LR exhibits a higher overhead on the smaller platform where resource pressure is increased, possibly due to its inability to converge until the final iteration. MMKP-MDF and FFEMS display similar execution times, both capable of scheduling ten applications within 100 ms. Given that FFEMS schedules significantly more test cases than

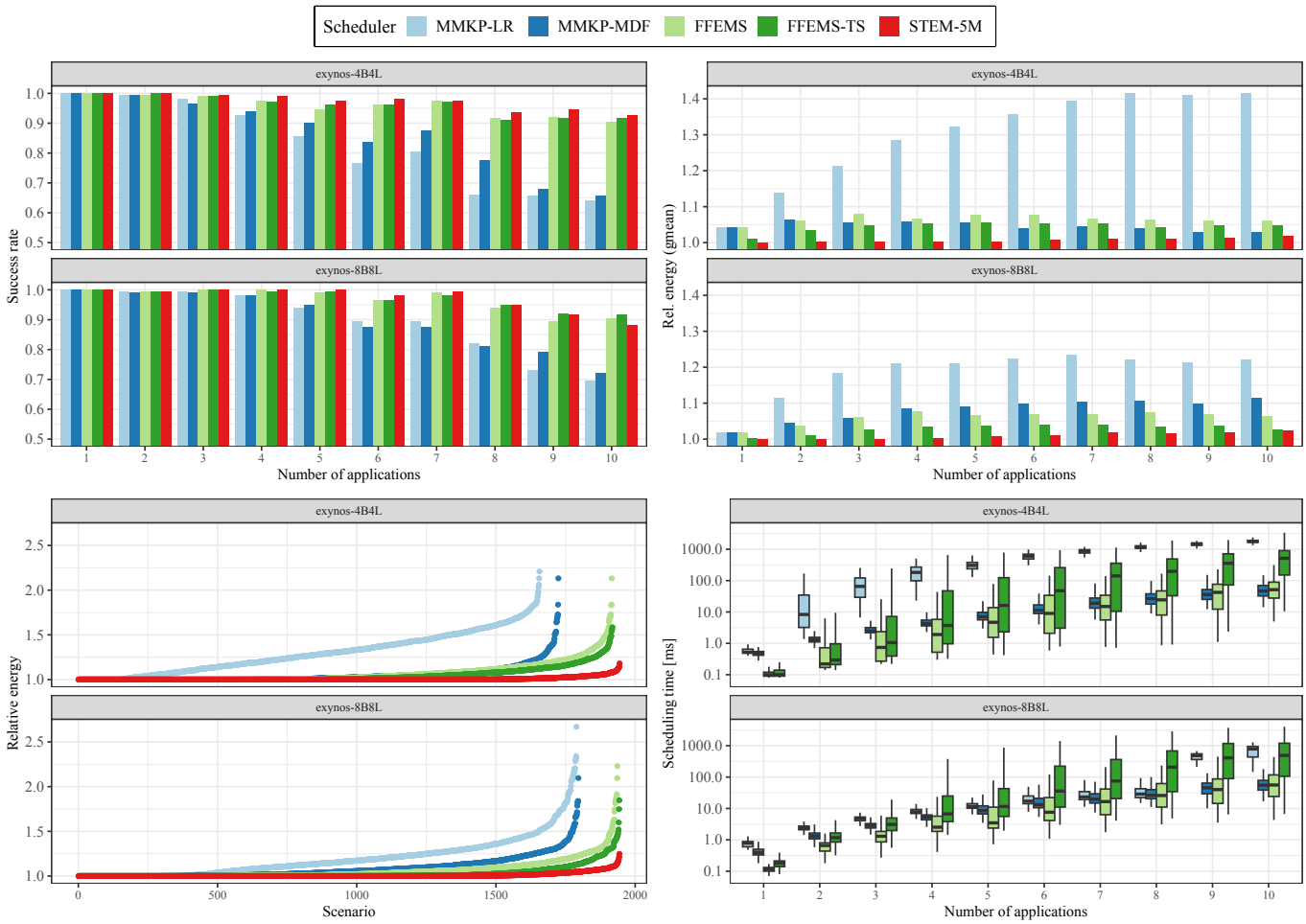

Fig. 4. Performance comparison of schedulers across two heterogeneous platforms concerning success rate (top left), relative energy consumption (top right), and scheduling time (bottom right). The bottom left plot showcases the monotonic curves of the relative energies.

MMKP-MDF, it emerges as the superior option. As expected, the time complexity of FFEMS^{TS} is higher, scheduling 10 jobs within 100 ms – 1 s, a trade-off for improved energy efficiency.

VII. Conclusion

In this study, we investigated the impact of the mapping decision model, especially the temporal component of the spatio-temporal mappings, on both schedulability and energy efficiency in real-time systems. We introduced two novel algorithms, STEM and FFEMS, designed for flexible spatio-temporal mappings for heterogeneous multi-core systems. The results indicate that employing fixed-point spatio-temporal mapping models enhances the energy efficiency of the found solutions, while the flexible variant of this model significantly improves the schedulability. Among the presented algorithms, STEM finds the most energy-efficient solutions. However, its enormous overhead makes it unsuitable for use in runtime systems. In contrast, FFEMS exhibits an outstanding balance between performance and runtime overhead: Given similar runtime overhead as the state-of-the-art MMKP-MDF (up to 100 ms for 10 jobs), FFEMS schedules up to 16 % more test cases. Moreover, its "tail-switch" optimization further improves energy efficiency, albeit with an increase in runtime overhead, which is nonetheless acceptable in runtime systems.

Acknowledgments

This work was funded in part by the German Federal Ministry of Education and Research (BMBF) through the project "E4C" (16ME0426K) and the programme of "Souverän. Digital. Vernetzt." (Joint project 6G-life, 16KISK001K). We thank the team at Silexica (now with AMD) for providing the applications and their support with tooling.

References

[1] P. Greenhalgh, "big. little technology: The future of mobile," *Arm Limited, White Paper*, p. 12, 2013.

[2] G. Wathan, "Arm dynamiq: Technology for the next era of compute," 2017.

[3] "Arm dynamiq shared unit 120 technical reference manual," 2023. Available: https://developer.arm.com/documentation/102547/0100 [Accessed: 2023-07-18].

[4] J. Castrillon, K. Desnos, A. Goens, and C. Menard, *Dataflow Models of Computation for Programming Heterogeneous Multicores*, pp. 1–40. Singapore: Springer Nature Singapore, Jan. 2023.

[5] B. Pourmohseni, M. Glaß, J. Henkel, H. Khdr, M. Rapp, V. Richthammer, T. Schwarzer, F. Smirnov, J. Spieck, J. Teich, A. Weichslgartner, and S. Wildermann, "Hybrid application mapping for composable many-core systems: Overview and future perspective," *Journal of Low Power Electronics and Applications*, vol. 10, no. 4, 2020.

[6] A. Goens, R. Khasanov, J. Castrillon, M. Hähnel, T. Smejkal, and H. Härtig, "Tetris: A multi-application run-time system for predictable execution of static mappings," in *Proceedings of the 20th International Workshop on Software and Compilers for Embedded Systems*, SCOPES '17, (New York, NY, USA), p. 11–20, Association for Computing Machinery, 2017.

[7] C. Ykman-Couvreur, V. Nollet, F. Catthoor, and H. Corporaal, "Fast multi-dimension multi-choice knapsack heuristic for mp-soc run-time management," in *Proceedings of SOC*, pp. 1–4, Nov 2006.

[8] S. Wildermann, M. Glaß, and J. Teich, "Multi-objective distributed run-time resource management for many-cores," in *Proceedings of DATE*, pp. 221:1–221:6, 2014.

[9] S. Wildermann, A. Weichslgartner, and J. Teich, "Design methodology and run-time management for predictable many-core systems," in *Proceedings of ISORCW*, pp. 103–110, April 2015.

[10] R. Khasanov and J. Castrillon, "Energy-efficient runtime resource management for adaptable multi-application mapping," in *Proceedings of the 2020 Design, Automation and Test in Europe Conference (DATE)*, DATE '20, pp. 909–914, IEEE, Mar. 2020.

[11] R. Khasanov, J. Robledo, C. Menard, A. Goens, and J. Castrillon, "Domain-specific hybrid mapping for energy-efficient baseband processing in wireless networks," *ACM Transactions on Embedded Computing Systems (TECS). Special issue of the International Conference on Compilers, Architecture, and Synthesis of Embedded Systems (CASES)*, vol. 20, Sept. 2021.

[12] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd ed., 2015.

[13] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts : Towards memetic algorithms," 1989.

[14] P. Yang, P. Marchal, C. Wong, S. Himpe, F. Catthoor, P. David, J. Vounckx, and R. Lauwereins, "Managing dynamic concurrent tasks in embedded real-time multimedia systems," in *Proceedings of the 15th International Symposium on System Synthesis*, ISSS '02, (New York, NY, USA), p. 112–119, Association for Computing Machinery, 2002.

[15] G. Ascia, V. Catania, and M. Palesi, "Multi-objective mapping for mesh-based noc architectures," in *International Conference on Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004.*, pp. 182–187, 2004.

[16] G. Mariani, V. Sima, G. Palermo, V. Zaccaria, C. Silvano, and K. Bertels, "Using multi-objective design space exploration to enable run-time resource management for reconfigurable architectures," in *Proceedings of DATE*, pp. 1379–1384, March 2012.

[17] A. Weichslgartner, D. Gangadharan, S. Wildermann, M. Glaß, and J. Teich, "Daarm: Design-time application analysis and run-time mapping for predictable execution in many-core systems," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2014 International Conference on*, pp. 1–10, IEEE, 2014.

[18] A. K. Singh, A. Kumar, and T. Srikanthan, "Accelerating throughput-aware runtime mapping for heterogeneous mpsocs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, jan 2013.

[19] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

[20] J. Spieck, S. Wildermann, and J. Teich, "A learning-based methodology for scenario-aware mapping of soft real-time applications onto heterogeneous mpsocs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 28, dec 2022.

[21] T. Mukherjee, A. Banerjee, G. Varsamopoulos, S. K. Gupta, and S. Rungta, "Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers," *Computer Networks*, vol. 53, no. 17, pp. 2888–2904, 2009. Virtualized Data Centers.

[22] R. Khasanov, A. Goens, and J. Castrillon, "Implicit data-parallelism in kahn process networks: Bridging the macqueen gap," in *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM'18), co-located with 13th International Conference on High-Performance and Embedded Architectures and Compilers (HiPEAC)*, PARMA-DITAM '18, (New York, NY, USA), pp. 20–25, ACM, Jan. 2018.

[23] T. Blickle and L. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Evolutionary Computation*, vol. 4, no. 4, pp. 361–394, 1996.

[24] C. Menard, A. Goens, G. Hempel, R. Khasanov, J. Robledo, F. Teweleitt, and J. Castrillon, "Mocasin—rapid prototyping of rapid prototyping tools: A framework for exploring new approaches in mapping software to heterogeneous multi-cores," in *Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools Proceedings*, DroneSE and RAPIDO '21, (New York, NY, USA), p. 66–73, Association for Computing Machinery, 2021.

[25] H. Bouraoui, J. Castrillon, and C. Jerad, "Comparing dataflow and openmp programming for speaker recognition applications," in *Proceedings of the 10th and 8th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, PARMA-DITAM 2019, (New York, NY, USA), Association for Computing Machinery, 2019.

[26] M. Ahmed, R. Seraj, and S. M. S. Islam, "The k-means algorithm: A comprehensive survey and performance evaluation," *Electronics*, vol. 9, no. 8, p. 1295, 2020.