Domain-specific hybrid mapping for energy-efficient baseband processing in wireless networks

ROBERT KHASANOV, JULIAN ROBLEDO, and CHRISTIAN MENARD, TU Dresden, Germany
ANDRÉS GOENS*, Barkhausen Intitut, Germany
JERONIMO CASTRILLON, TU Dresden, Germany

Advancing telecommunication standards continuously push for larger bandwidths, lower latencies, and faster data rates. The receiver baseband unit not only has to deal with a huge number of users expecting connectivity but also with a high workload heterogeneity. As a consequence of the required flexibility, baseband processing has seen a trend towards software implementations in cloud Radio Access Networks (cRANs). The flexibility gained from software implementation comes at the price of impoverished energy efficiency. This paper addresses the trade-off between flexibility and efficiency by proposing a *domain-specific* hybrid mapping algorithm. Hybrid mapping is an established approach from the model-based design of embedded systems that allows us to retain flexibility while targeting heterogeneous hardware. Depending on the current workload, the runtime system selects the most energy-efficient mapping configuration without violating timing constraints. We leverage the structure of baseband processing, and refine the scheduling methodology, to enable efficient mapping of 100s of tasks at the millisecond granularity, improving upon state-of-the-art hybrid approaches. We validate our approach on an Odroid XU4 and virtual platforms with application-specific accelerators. On different LTE workloads, our hybrid approach shows significant improvements both at design time and at runtime. At design-time, mappings of similar quality to those obtained by state-of-the-art methods are generated around four orders of magnitude faster. At runtime, multi-application schedules are computed 37.7 % faster than the state-of-the-art without compromising on the quality.

ACM Reference Format:
Robert Khasanov, Julian Robledo, Christian Menard, Andrés Goens, and Jeronimo Castrillon. 2018. Domain-specific hybrid mapping for energy-efficient baseband processing in wireless networks. *J. ACM* 37, 4, Article 111 (August 2018), 25 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The rapid growth of mobile data demands on wireless networks represents a huge challenge for traditional distributed radio access networks (RANs). Compared to its predecessor 4G LTE, the fifth generation (5G) network is envisioned to support 1000× higher data traffic [4]. It introduces new use cases that expand the variety of possible applications to be handled in a base band unit (BBU). As the number of connected devices increases in the Internet of Things (IoT), data traffic

*Work done in part at the Chair for Compiler Construction at TU Dresden

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), 2021.
Authors' addresses: Robert Khasanov, robert.khasanov@tu-dresden.de; Julian Robledo, julian.robledo@tu-dresden.de; Christian Menard, christian.menard@tu-dresden.de, TU Dresden, Chair for Compiler Construction, Dresden, Germany; Andrés Goens, andres.goens@barkhauseninstitut.org, Barkhausen Intitut, Dresden, Germany; Jeronimo Castrillon, jeronimo.castrillon@tu-dresden.de, TU Dresden, Chair for Compiler Construction, Dresden, Germany.

0004-5411/2018/8-ART111 $15.00
https://doi.org/10.1145/1122445.1122456

will not only be generated by conventional mobile phones but a variety of devices with diverse requirements. To support this workload heterogeneity, future networks require many more and flexible BBUs, ranging from full-fledged stations to femtocells or even smart surfaces [40]. These challenges are only expected to increase with upcoming telecommunication standards [34].

For multiple reasons, baseband processing has seen a trend towards software implementations. First, the introduction of cloud RANs (cRANs) [14] allowed an efficient resource allocation in a shared centralized system. Then, virtualized RANs (vRANs) [46] enabled the move from costly, COTS-based solutions to software-based ones, leveraging cRANs and the utilization of programmable hardware. Software is more flexible, it permits implementing multiple standards and variations, besides adapting better to heterogeneous, dynamic workloads. This allows cutting down on development costs and a shorter time-to-market. A similar motivation led to a large body of research on software defined radio (SDR) for handheld devices a decade ago [43]. Compared to handhelds, the more dynamic and multi-user nature of BBUs makes cRAN considerably more challenging.

The flexibility gained from moving to software implementations comes at a high price: (energy) efficiency. Application Specific Integrated Circuits (ASICs) are orders of magnitude more efficient at performing specific tasks compared to a general-purpose processor, especially for baseband processing kernels [8]. They are, however, inflexible to changes in the standard. In addition to this, virtualization and the diversity of target hardware makes it hard to specialize baseband implementations in the context of a vRAN, so as to leverage hardware acceleration. Reconfigurable hardware (e.g. FPGAs) is an alternative to ASICs that provides a better trade-off between flexibility and efficiency, albeit one that is ill-suited to deal with the runtime changes of a heterogeneous workload. Modern hardware-based approaches that exploit dynamic reconfigurability in FPGAs, coarse-grained reconfigurable arrays (CRGAs) [21], or application-specific processors [35] address these issues, targeting different points in the spectrum of flexibility and efficiency, further increasing the hardware heterogeneity of systems. Programming these complex, heterogeneous systems requires a structured approach to software/hardware co-design [1, 6, 49].

A well-studied class of structured approaches to hardware/software co-design is what is known as model-based design [31]. In model-based design, the execution of the application is governed by a well-defined Model of Computation (MoC). Using models of the hardware architecture and the application, efficient execution of the application can be derived via design-space exploration. There is plenty of research on model-based design for signal processing systems in general [19] and for deeply specialized baseband processing systems [13, 36]. For dealing with dynamic behavior in model-based design, we can use hybrid mappings, where multiple static mappings are generated at compile-time and one is selected dynamically. It has been shown that hybrid mappings improve the efficiency [22, 32, 47] and predictability [18] of the execution while enabling dynamic behavior.

In their full generality, domain-oblivious approaches for hybrid mapping of model-based applications have been developed and validated for comparatively smaller applications, consisting of a couple of dozen of tasks, at most. In BBU design, in contrast, the processing for each user equipment (UE) consists of several dozens of computational tasks. Scaling these methods accordingly requires domain-specific improvements which leverage the structure of the application.

In this paper, we propose a domain-specific, model-based hybrid mapping approach for BBU design, addressing the trade-off between flexibility and efficiency. Crucially, static mappings enable both, generating code for software-only solutions, but also hardware-accelerated implementations. With a hybrid approach, we can retain the flexibility. In particular, we show we can greatly improve the scalability at design time w.r.t. existing methods (e.g., [17, 24, 29]) up to hundreds of tasks, using knowledge from the structure of the problem (Section 3.2). We also reduce the runtime overhead w.r.t. other hybrid solutions (e.g., [22, 27, 48]) by refining the scheduling methodology (Section 3.3).

The main contribution of this paper is showing how hybrid mapping strategies are viable for flexible and efficient executions of baseband processing. We work with a benchmark of an LTE base station and extract a parameterized model of computation for baseband processing (Section 2). Given that the number of parameters involved tends to increase with the upcoming standards, having this parameterization will allow us to easily extend the model for newer protocols. We evaluate this on a model validated on an Odroid XU4, extrapolating from real recorded LTE traces (Section 4).

2 BASEBAND PROCESSING

Baseband processing is a computationally demanding task, historically requiring considerable effort by expert teams at telecommunication companies in hardware-software co-design. At the side of the base station, for the uplink communication in LTE, incoming data is structured into multiple blocks. These blocks, referred to as subframes, each corresponds to 1 ms in the time domain. Every subframe consists of two adjacent slots lasting 0.5 ms each, and each slot contains 14 symbols. The frequency domain is divided into sub-carriers spaced 15 kHz apart from each other. Data coming in every subframe contains the information from up to 10 user equipments (UEs) received by the antennas of the base station. Each UE transmits an integer number of physical resource blocks (PRBs), where a PRB corresponds to the minimum possible amount of data allocated to a single UE, or more precisely, one subframe in the time domain and 12 subcarriers in the frequency domain [2].

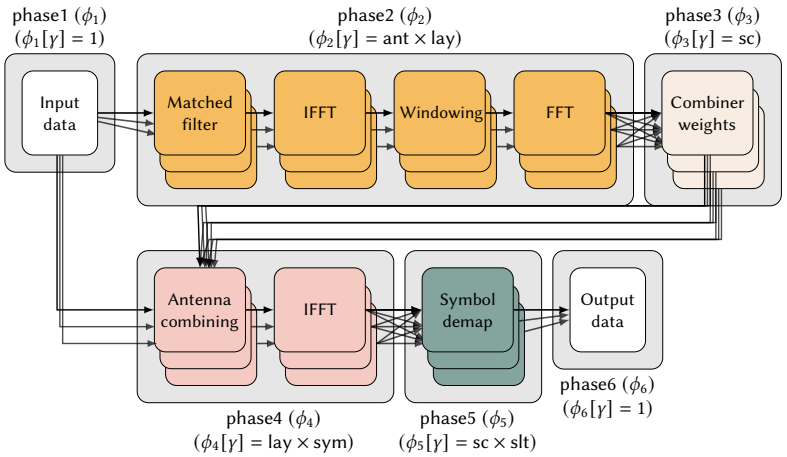


Fig. 1. Block diagram of a baseband receiver

A typical model of a baseband receiver is shown in Figure 1. The physical layer of the baseband processing consists of a series of computational tasks grouped into phases ϕ. Such tasks are used to reconstruct the data sent by the transmitter. They then pass it to the upper layers of the communication stack. Typical tasks include a matched filter, fast Fourier transform (FFT), windowing, inverse FFT (IFFT), antenna combining, combiner weights calculation, or soft symbol demaping. These tasks exhibit a high runtime variation since their functionality is controlled by multiple parameters, such as the number of UEs, the number of PRBs allocated to each of them, the number of antennas (ant) at the base station, the modulation scheme, the number of concurrent streams of data being sent by the UE, also known as layers (lay), the number of slots (slt) per subframe, which is fixed to two in LTE, but is more variable in 5G, the number of subcarriers (sc) per slot in the frequency domain and the number of symbols (sym) per slot in the time domain. This parameterization is a key source of the flexibility of modern and upcoming wireless communication

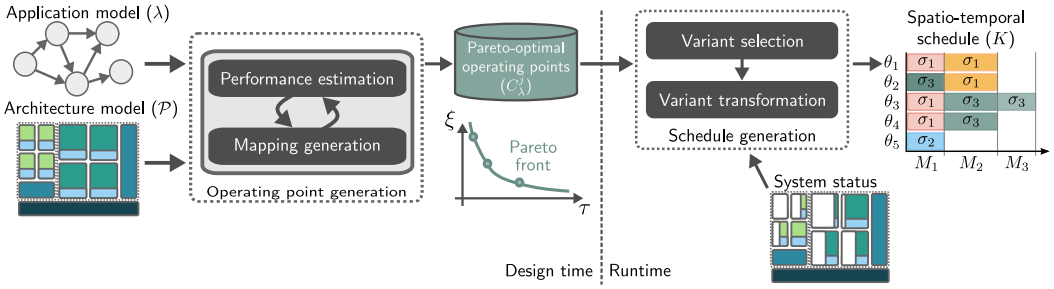


Fig. 2. General hybrid mapping flow

standards. It is also a key factor driving the trend away from inflexible hardware and towards more flexible software implementations.

Depending on the aforementioned parameters, the baseband processing chain for a single UE within a subframe varies in terms of its runtime and computational structure. The parameters control the degree of data parallelism for each phase ϕ, where all tasks of a phase have the same degree of parallelism, indicated by γ in the figure. As mentioned before, parameters such as the number of PRBs allocated to the UE and the modulation scheme, also control the computational intensity of the task instances and, thus, the performance requirements. Each user in a subframe may be served by a different dataflow graph. The degree of parameterization is already higher in 5G, which operates at a higher bandwidth. In addition to the classical mobile communication with enhanced mobile broadband (eMBB), 5G introduces new uses cases: massive machine-type communications (mMTC), and ultra-reliable low-latency communications (URLCC) [51]. Every UE request in an upcoming subframe has to be processed within a certain deadline, otherwise the request is discarded. Different use cases have different deadlines, namely, 2.5 ms for eMBB and mMTC, and 0.5 ms for URLCC. There is a clear increasing trend in upcoming standards for richer parameterization and supported use cases.

For the work in this paper, we use the PHY benchmark [37], which features a structure similar to that from Figure 1. The benchmark is an implementation of an LTE baseband processing system and it is suitable for handling realistic LTE workloads. It captures the parallelism of LTE with a configurable multi-threaded implementation.

3 HYBRID MAPPING FOR BASEBAND PROCESSING

Figure 2 gives an overview of hybrid mapping approaches in general, which can be divided into a design-time and a runtime flow. At design time, mappings are generated using a design-space exploration (DSE) technique. By leveraging models of the application and the target architecture, this DSE step can pre-compute a set of energy-efficient mappings that allow the system constraints to be met. These pre-computed mappings, sometimes also called operating points, approximate a set of Pareto-optimal points in the multi-dimensional objective space of energy efficiency, execution time, and resource utilization. These might be partial mappings, given as constraints [47] or complete mappings, which are later to be transformed at runtime [18].

The runtime resource manager receives the set of (approximately) Pareto-optimal operating points, which correspond to different mapping variants. When executing a new application at runtime, it selects a (spatial) mapping variant based on the current system workload. The selected variant is then adapted to the current system workload, either by finalizing the mapping computation from a partial mapping or by transforming a complete mapping to utilize the free resources in the platform. This yields a mapping that can be executed in the system.

In some approaches, the resource manager also generates a temporal schedule for the application. It is worth noting that spatial mapping is a special case of the general scheduling problem, which can also be described as a spatio-temporal mapping problem. It is common, as we do in this paper, to split this problem into two sub-problems, the spatial mapping problem and the temporal schedule. Our hybrid approach calculates both, based on statically generated (spatial) mappings.

In the following, we present our approach, including our novel domain-specific specializations. The general problem formulation and runtime resource managing algorithm are based on the formulation in [22]. We borrow some terminology from that work and explain here for context.

3.1 System model

In this section we introduce the input models: a platform model, an application model and a task model. Based on these, we describe the different stages of the hybrid mapping methodology.

3.1.1 Platform model. For the *platform model* we assume a heterogeneous platform $\mathcal{P}$ that contains a set of processing elements Θ. Each processing element $\theta \in \Theta$ is of one of the m resource types $\Omega = \{\Omega_1, \ldots, \Omega_m\}$, and ω_i denotes the number of resources of type Ω_i. We thus denote the number of resources of each type in the platform by $\mathcal{P}[\vec{\omega}] = (\omega_1, \ldots, \omega_m)$. Each processing element θ could represent either a regular processor core or an application-specific accelerator. Below, where needed, we explicitly distinguish the general-purpose processor cores and the accelerators. The crucial difference being that accelerators cannot execute arbitrary tasks, whereas regular cores can.

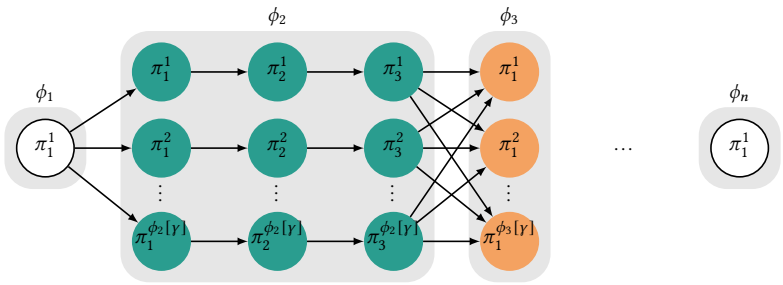


Fig. 3. General task graph with a phase-sequential structure.

3.1.2 Application model. The platform executes jobs represented by a task graph with a *phase-sequential* structure depicted in Figure 3. Each phase $\phi = \phi\langle\Pi, \gamma\rangle$ contains the ordered set of tasks $\phi[\Pi] = (\pi_1, \ldots, \pi_{|\phi[\Pi]|})$, which are connected in sequential order, i.e., there is a communication edge from π_i to π_{i+1}, for $i \in \{1, \ldots, |\phi[\Pi]| - 1\}$. The whole phase could be data-level parallelized with a parallelization factor $\phi[\gamma]$ and the tasks in each replica $\phi[\Pi^k] = (\pi_1^k, \ldots, \pi_{|\phi[\Pi]|}^k)$ do not depend on tasks in other replicas of the same phase.

The phases $\phi \in \Phi$ in the task graph are ordered sequentially: The final tasks of a phase ϕ_i are connected to all the replicas of the first task in the subsequent phase ϕ_{i+1}, i.e., there is a communication edge from $\phi_i[\pi^{k'}_{|\phi_i[\Pi]|}]$ to $\phi_{i+1}[\pi_1^{k''}]$ $\forall k' \in \{1, \ldots, \phi_i[\gamma]\}$, $k'' \in \{1, \ldots, \phi_{i+1}[\gamma]\}$. A *task graph* is defined as an ordered set of phases, i.e. $\lambda = \lambda_{\vec{\eta}}\langle\Phi\rangle$, where $\vec{\eta}$ is the list of parameters. In the case of the model presented in Section 2, we have $(\eta_1, \eta_2, \eta_3, \eta_4) = (\text{PRBs}, \text{mod}, \text{lay}, \text{ant})$.

3.1.3 Task model. A task π is annotated with energy-performance data $\langle\vec{\tau}, \vec{\xi}\rangle$ ($\vec{\tau} = (\tau_1, \ldots, \tau_m)$, $\vec{\xi} = (\xi_1, \ldots, \xi_m)$), where $\pi[\tau_i]$ and $\pi[\xi_i]$ are the latency and the energy consumption of the task π on Ω_i. If the task cannot be executed on the particular resource type Ω_i, then $\pi[\tau_i] = \pi[\xi_i] = \perp$.

Table 1. Summary of the notations used in this work.

Notation	Comment
$X\langle \ldots \rangle$	An object X with the parameters.
$X[\bullet]$	Access to the parameter in the object X
$\mathcal{P}\langle \Theta, \Omega, \Theta \rightarrow \Omega \rangle$	A platform $\mathcal{P}$ with the resources Θ of the resource types Ω
$\lambda_{(\text{PRBs,mod,lay})}\langle \Phi \rangle$	An application λ with phases Φ
$\phi\langle \Pi, \gamma \rangle$	A phase structure: an ordered set of tasks Π and a replication factor γ
$\phi[\Pi_i^k]$	The i-th process in the k-th replica of the phase ϕ
$\pi\langle \vec{\tau}, \vec{\xi} \rangle$	A task with latency τ and energy consumption ξ for each resource type Ω
$c_\lambda^j\langle \mu, \tau, \xi \rangle$	An operating point characterized by a mapping μ, latency τ and energy ξ
Σ_t	A set of (non-finished) requests registered at the runtime manager, including the newly arrived requests at time t
$\sigma\langle \alpha, \delta, \lambda, \rho \rangle$	A request information: an application λ, α is arrival time, δ is deadline, ρ is a remaining progress ratio
$\nu\langle \sigma, \lambda, c, \mu \rangle$	A finalized job configuration: a job σ, an application λ, a configuration c, the final mapping μ
M, Δ_M	A multi-aplication mapping M and a time interval $\Delta_M = [\underline{\Delta_M}, \overline{\Delta_M})$ of the mapping M
K	A schedule

3.2 Generation of operating points

As depicted in Figure 2, the hybrid mapper requires a set of Pareto-optimal operating points generated at design time. In this section, we define what operating points are and introduce a fast mapping generator based on load balancing, which exploits the phase-sequential structure of the task graphs and targets both regular processing elements and accelerators.

3.2.1 Operating points. An *operating point* is characterized by the mapping μ, the (worst-case) latency τ and the energy consumption ξ, i.e., $c = c_\lambda^j\langle \mu, \tau, \xi \rangle$. We denote the set of resources used in the mapping μ by $\mu[\Theta]$. We assume that at each operating point c the mappings $c[\mu]$ are generated in such way that all resources $c[\mu][\Theta]$ (or simply $c[\Theta]$) process the workload with a constant progress rate. Generated operating points are assumed to be Pareto-filtered, such that each operating point is better than any other in at least one parameter, i.e., less number of cores of any particular type $\vec{\omega}$, lower latency τ or energy consumption ξ. Additionally, we assume the operating points also to be filtered with the application requirements. Concretely, we assume that no operating points violating the real-time constraints are considered. In total, for each application λ, we generate N_λ operating points.

3.2.2 Fast mapping algorithm. We propose a fast mapping algorithm which exploits the phase-sequential structure of the task graphs. As input, the algorithm takes the task graph with the inherent structure information $\lambda\langle \Phi \rangle$, the energy-performance data of each task $\pi\langle \vec{\tau}, \vec{\xi} \rangle$, the set of the processor resources Θ and accelerators $\tilde{\Theta}$. Mapping of the tasks to the resources is done sequentially on each phase $\phi \in \lambda[\Phi]$. The implementation of a single phase mapping is written in Algorithm 1, and mainly consists of two steps: it maps to regular resources (lines 2-5), and then remaps some of the tasks to the accelerators (lines 6-19). Along with the formal description, we

Algorithm 1 Phased-Fast mapping algorithm.

Input: Phase structure ϕ, regular resources Θ, accelerators $\tilde{\Theta}$
Output: Phase mapping $\mu_\phi : \pi \mapsto \theta$, delay τ_ϕ, energy ξ_ϕ

// Calculate per-core the delay of all processes $t[\theta]$, and initialize the total delay $T[\theta]$
1: **for all** $\theta \in \Theta$ **do** $t[\theta] = \sum_{\pi \in \phi[\Pi]} \pi[\tau_\theta]$, $T[\theta] = 0$

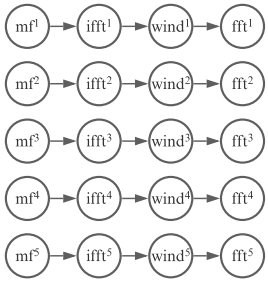
// Perform load balancing on regular resources
2: **for** $j \leftarrow 1$ to $\phi[\gamma]$ **do**
3: $\quad\theta^* \leftarrow \text{argmin}_{\theta \in \Theta}\, t[\theta] + T[\theta]$
4: $\quad T[\theta^*] \leftarrow T[\theta^*] + t[\theta^*]$
5: $\quad$**for all** $\pi \in \phi[\Pi]$ **do** $\mu_\phi[\pi^j] \leftarrow \theta^*$

// Re-balancing to accelerators
6: **for all** $\theta \in \tilde{\Theta}$ **do**
7: $\quad T[\theta] = 0$
// Accelerator has no workload till the first data is available
8: $\quad$**for all** $\pi \in \phi[\Pi]$ **do**
9: $\quad\quad$**if** $\pi[\tau_\theta] \neq \perp$ **then break**
10: $\quad\quad T[\theta] \leftarrow T[\theta] + \min_{\theta' \in \Theta \cup \tilde{\Theta} \setminus \theta} \pi[\tau_{\theta'}]$
11: **for all** $\pi \in \phi[\Pi]$ **do**
12: $\quad\tilde{\Theta}^\pi \leftarrow \{\theta \mid \theta \in \tilde{\Theta} \wedge \pi[\tau_\theta] \neq \perp\}$
13: $\quad$**if** $\tilde{\Theta}^\pi = \varnothing$ **then continue**
14: $\quad$**for** $j \leftarrow 1$ to $\phi[\gamma]$ **do**
15: $\quad\quad\theta' \leftarrow \text{argmax}_{\{\theta \in \Theta \mid \exists k, \mu[\pi^k] = \theta\}}\, T[\theta]$
16: $\quad\quad\theta'' \leftarrow \text{argmin}_{\theta \in \tilde{\Theta}^\pi}\, T[\theta]$
17: $\quad\quad$**if** $T[\theta'] - \pi[\tau_{\theta'}] < T[\theta''] + \pi[\tau_{\theta''}]$ **then break**
18: $\quad\quad\mu_\phi[\pi^j] \leftarrow \theta''$
19: $\quad\quad T[\theta'] \leftarrow T[\theta'] - \pi[\tau_{\theta'}]$, $T[\theta''] \leftarrow T[\theta''] + \pi[\tau_{\theta''}]$
// Calculate phase delay and energy
20: $\tau_\phi \leftarrow \max_{\theta \in \Theta \cup \tilde{\Theta}} T[\theta]$
21: $\xi_\phi \leftarrow \sum_{\pi \in \phi[\Pi], j \in \{1,\ldots,\phi[\gamma]\}} \pi[\xi_{\mu_\phi[\theta]}]$
22: **return** μ_ϕ, τ_ϕ, ξ_ϕ

also provide step-by-step visualization of the algorithm in Figure 4, in which we use the phase 2 of the baseband receiver as the input model of the task graph phase (cf. Figure 1).

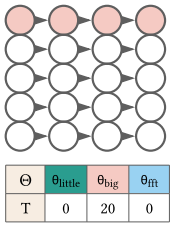
The sequence of tasks in a single replica of a single phase is independent of the sequences of tasks in the other replicas of that phase (cf. Figure 3). For this reason, we map all tasks in the same phase replica to a single regular resource. Therefore, at this stage, we consider the tasks in the replica virtually fused and calculate the latency of the replica $t[\theta]$ as the sum of the latencies of its tasks (line 1). We initialize $T[\theta]$ denoting the total latency of already mapped processes to the resource θ. For each replica, we determine the resource θ, which would have the least total latency after adding the replica to it, and maps the replica to it (lines 2-5). This heuristic attempts to equalize the total latency across regular resources Θ.

Figure 4 illustrates the working of the algorithm for a target platform with three processing elements, namely a *big* core θ_{big}, a *little* core θ_{little} and an FFT/IFFT accelerator θ_{fft}. The sum of the latencies on a big core is 20, and on a little core is 32 (the setup and the numbers are consistent with the platforms we use in the evaluation in Section 4). First, the algorithm maps the whole line of



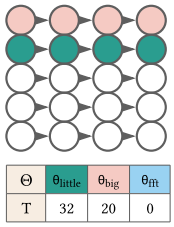
Θ \ π	mf	wind	fft/ifft
θlittle	5	3	12
θbig	4	2	7
θfft	-	-	3

(a) A phase structure with the performance metadata of the tasks.



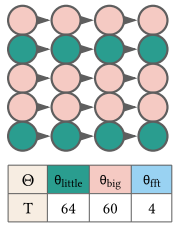
Θ	θlittle	θbig	θfft
T	0	20	0

(b) Mapping of the first line of replicas to θ_{big}.



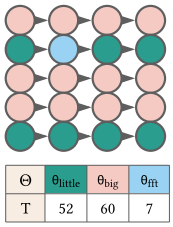
Θ	θlittle	θbig	θfft
T	32	20	0

(c) Mapping of the second line to θ_{little}.



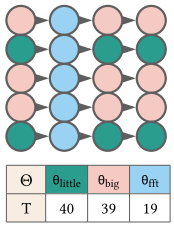
Θ	θlittle	θbig	θfft
T	64	60	4

(d) Mapping of the remaining replicas and initializing $T[\theta_{\text{fft}}]$.



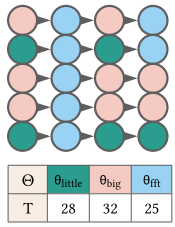
Θ	θlittle	θbig	θfft
T	52	60	7

(e) The first *ifft* task is moved from θ_{little} to θ_{fft}.



Θ	θlittle	θbig	θfft
T	40	39	19

(f) Migrating the remaining *ifft* tasks to the accelerator.



Θ	θlittle	θbig	θfft
T	28	32	25

(g) Moving two *ffts* to the accelerator. The algorithm is finished.

Fig. 4. Step-by-step visualization of the fast mapping algorithm applied on a phase of the baseband receiver onto a platform with three resources. The latency values on the resources are taken from measurements of the LTE PHY benchmark on Odroid XU4, and the FFT kernel on Xilinx Ultra96 FPGA board (see Section 4).

replicas to the big resource since this decision minimizes the maximum delay after assignment (4b). Similarly, the second line of replicas is assigned to the little resource (4c). After all replica lines are distributed among the regular resources, the total delays on both resources are roughly equal (4d).

The algorithm then remaps some of the early mapped resources to the accelerators $\tilde{\Theta}$. We initialize $T[\theta], \theta \in \tilde{\Theta}$ with the minimum delay before the accelerator might receive the first task replica to execute (lines 7-10). In our example, the minimum delay before the accelerator could receive the first *ifft* task equals the minimum delay of *mf*, i.e. 4 (4d).

After the initialization, the algorithm performs task re-balancing. For an intuition behind the lines 15-19, let us consider a single step of remapping of the single process π to an accelerator. The algorithm searches for the regular resource θ', which has the largest total latency, containing at least one replica of the task π. The remapping is performed if the total accelerator time does not exceed the largest total latency of regular processes after migration (17). If there are many accelerators, the algorithm maps the task to the accelerator with the least total latency θ'' (16). In Figure 4, the first candidate for remapping is chosen from the little core, one with the largest $T[\theta]$. When the task is moved, the values of $T[\Theta]$ are update accordingly (4e). When all *ifft*s are migrated (4f), the algorithm continues loading the accelerator with *fft*s until the stop condition is reached: the total delay of the accelerator should not exceed the total delay of the regular processors (4g).

Finally, the total latency of the phase is calculated as the maximum of total latencies on each resource (20), and the energy consumption is calculated as the sum of energy consumptions of each tasks according to the resources they mapped to (21).

Once all phase mappings μ_ϕ have been computed, the final mapping is constructed $\mu \leftarrow \bigcup_{\phi \in \Phi} \mu_\phi$, with the total latency and energy calculated as: $\bullet \leftarrow \sum_{\phi \in \Phi} \bullet_\phi$.

Note that the algorithm attempts to distribute the resources over the tasks in each phase evenly, so the platform will process the workload on each generated mapping with an approximately constant progress rate. This property of the generated mappings increases the predictability at the runtime scheduling in case of compelled job migrations.

3.2.3 On generating Pareto-optimal operating points. Algorithm 1 describes the generation of a single operating point. To generate the (Pareto-optimal) set of points, we run the algorithm for many different subsets of the cores Θ, in which we vary the amount of cores of each particular type $\vec{\omega}$. In this work, we generate the operating points for all $\prod_{\Omega_i \in \Omega}(\omega_i + 1) - 1$ combinations of processing resources. The runtime manager supplied with fine-grained operating points may generate more optimal schedules. However, if the platform size is too large, this strategy will lead to an extremely large amount of operating points, which, in turn, has a direct impact on the overhead of runtime selection. To reduce the runtime overhead, some heuristics could be applied.

3.2.4 Limitations. Algorithm 1 has a couple of limitations. First, the algorithm does not consider the communication delays, which was an intentional choice for the sake of simplicity. Second, the algorithm assumes that the platform has a regular power model, i.e., the dynamic power usage of the cores is constant, and thus the energy consumption of tasks mapped to the specific core is proportional to latency. As a result, the mappings generated by this algorithm are not guaranteed to be Pareto-optimal, but as we will see in the evaluation in Section 4, they nevertheless result in good mappings.

3.3 Schedule generation

As depicted in Figure 2, the (Pareto-optimal) operating points generated at design-time are supplied to the Resource Manager (RM). At runtime, the RM receives requests to execute new jobs. The RM attempts to schedule them along with other currently running jobs. The successfully scheduled requests are then accepted, and a new schedule is generated. In this section, we define the model of requests, the schedule, and describe our scheduling algorithm for energy-efficient schedules.

3.3.1 Resource manager activations and request model. At every subframe in which new requests to process UEs arrive, the RM is activated to schedule these requests along with previously admitted UEs. The RM might accept all incoming requests, or only a subset of them. The successfully scheduled requests are accepted, and corresponding jobs are dispatched to the system. We denote by $\Sigma_{t'}$ a set of requests that includes the newly arrived requests at time t' along with previously admitted requests, which were not finished by $t = t'$. For a given job[1] $\sigma \in \Sigma_{t'}$, $\sigma[\alpha]$ denotes the arrival time, $\sigma[\delta]$ the (absolute) deadline, $\sigma[\lambda]$ the task graph, including the associated parameters **prbs**, **mod**, **lay**, **ant**, and $\sigma[\rho] \in [0, 1]$ the remaining progress ratio of the job, i.e., $\sigma = \sigma\langle\alpha, \delta, \lambda, \rho\rangle$.

3.3.2 Spatio-temporal schedule model. Given a set of requests $\Sigma_{t'}$, the RM attempts to find a spatio-temporal *schedule* K, which is defined as a list of multi-application mappings M applied during consecutive time segments (cf. Figure 2): $K = \{M_i \times \Delta_{M_i}\}$ $(0 \le i < N)$, where N is a number of segments, $\Delta_M = [\underline{\Delta_M}, \overline{\Delta_M})$ is a time interval of the mapping M, $\underline{\Delta_{M_0}} = t'$ and $\overline{\Delta_{M_i}} = \underline{\Delta_{M_{i+1}}}$, $0 \le i < N - 1$. Each multi-application mapping M contains selected job configurations $\nu = \nu\langle\sigma, \lambda, c, \mu\rangle$, which express

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

that the job σ associated with the task graph λ runs with the mapping μ, which is associated with the operating point c, i.e. $\exists j, c \equiv c_{\lambda}^{j}$. The final mapping is equivalent to the one specified in the operating point, i.e $\nu[\mu] \cong c[\mu]$, the amount of the used cores of each particular type is equal, i.e. $\nu[\mu][\vec{\omega}]$, or simply $\nu[\vec{\omega}] = c[\vec{\omega}]$.

Note that this formulation of schedule allows jobs to be remapped from one configuration to another across different segments in the schedule.

3.3.3 Optimization problem. The goal of this work is to generate energy-efficient schedules which do not violate the UE demands. Using the notation introduced above, we can now leverage the state-of-the-art runtime selection formulation from [22]:

$$\text{minimize} \sum_{M \in K} \sum_{\nu \in M} \nu[c][\xi] \frac{|\Delta_M|}{\nu[c][\tau]} \tag{1a}$$

$$\text{subject to} \sum_{\nu \in M} \nu[c][\vec{\omega}] \leq \mathcal{P}[\vec{\omega}], \qquad \forall M \in K \tag{1b}$$

$$\nu[\mu] \cong \nu[c][\mu], \qquad \forall \nu \in M \tag{1c}$$

$$\nu_1[\sigma] \neq \nu_2[\sigma], \qquad \forall \nu_1, \nu_2 \in M, \nu_1 \neq \nu_2 \tag{1d}$$

$$\sum_{\substack{M \in K \\ \nu \in M \\ \nu[\sigma]=\sigma}} \frac{|\Delta_M|}{\nu[c][\tau]} = \sigma[\rho], \qquad \forall \sigma \in \Sigma \tag{1e}$$

$$\max_{\substack{M \in K \\ \nu \in M \\ \nu[\sigma]=\sigma}} \overline{\Delta_M} \leq \sigma[\delta], \qquad \forall \sigma \in \Sigma. \tag{1f}$$

Constraint (1b) ensures that the resources required for mapping in the segment M do not exceed the available resources per type $\mathcal{P}[\vec{\omega}]$. Constraint (1c) describes that the final mapping μ is equivalent to the one specified in the operating point $c[\mu]$, but is not necessarily equal to it (variant transformation in Figure 2). Equation (1d) specifies that at each mapping segment at most one job mapping relates to each request σ. Constraint (1e) expresses that each job will run until the end. Finally, each job must finish the execution before the deadline (1f). If there is a feasible solution to this problem, the RM admits the request and changes the schedule accordingly. Otherwise, the request is rejected.

3.3.4 Joint scheduling algorithm. To solve the optimization problem, we propose a novel algorithm that extends on the MMKP-MDF algorithm from [22]. In MMKP-MDF, the RM performs the schedulability test for each new request individually, leading to multiple reschedules of jobs that have been already accepted. This considerably increases the runtime overhead, which is critical for baseband processing. We address the identified issues and propose a modified version of the algorithm called **MMKP-MDF-Joint**.

Similar to [22], we consider the resource types as knapsacks with certain *capacities*, and the job configurations c as *items* with certain *weights*. Weights are defined as the required processing time to finish the job multiplied by the required number of resources of corresponding type, i.e. $c[\tau] \times c[\vec{\theta}]$, while the capacities $\vec{J}$ express the available processing time per each resource type. Each job forms a group of items, in which exactly one item must be chosen. After negating the energy *values* of each item, the optimization goal becomes to maximize the overall (negative) value which can be expressed as a multiple-choice multi-dimensional knapsack problem (MMKP) [25]. Algorithm 2 describes our modified algorithm.

Algorithm 2 MMKP-MDF-Joint mapping heuristic.

Input: Set of jobs Σ_t, platform $\vec{\Theta}$, application table c_λ, current schedule K'
Output: Schedule K

```
1: J⃗ ← Θ⃗ × max(σ[δ] − t | σ ∈ Σ_t), K ← K′
2: for all σ ∈ Σ_t do
3:     jc[σ] ← ⊥
4:     if ∃M′ ∈ K′, ν′ ∈ M′ : ν′[σ] = σ then jc[σ] = ν′[c]
5: while ∃σ ∈ Σ_t : jc[σ] = ⊥ do
6:     σ*, cl ← NextJobMDF(Σ_t, jc, J⃗, c_λ)
7:     while jc[σ*] = ⊥ do
8:         if |cl| = 0 then jc[σ*] ← c_∅ continue
9:         c* ← argmin_{c∈cl}{c[ξ]}
10:        jc* ← jc, jc*[σ*] ← c*
11:        K* ← ScheduleJobs(Σ_t, jc*, Θ⃗, c)
12:        if K* ≠ ∅ then
13:            jc ← jc*, K ← K*
14:            J⃗ ← J⃗ − c*[θ⃗] × c*[τ] × σ*[ρ]
15:        else
16:            cl ← cl \ c*
17: return K
```

At each activation of the RM, the containers $\vec{J}$ are initialized with the overall processing time per resource type limited by the largest job deadline (the time scope of the analysis), the schedule K is initialized with the current schedule K' (line 1). In Line 4, the algorithm analyzes the previously generated schedule K' and extracts the selected job configurations. All newly arrived jobs are mapped in the main loop (5), the order is determined with the function NextJobMDF, which along with the next job to map σ^* returns a list of filtered configurations cl for σ^* (6). The job σ^* is determined by the heuristic "Maximum Difference First" (MDF) described in [22]. The filtered configurations include only those items which can meet deadlines and fit the containers $\vec{J}$. In Lines 7-16, the algorithm iterates over the configurations in non-decreasing order of energy consumption. It then attempts to schedule the job σ^* with already mapped jobs in ScheduleJobs similar as in [22]. Once the job is successfully scheduled, its configuration and a new schedule is saved, and the containers $\vec{J}$ are updated (lines 13-14). If no feasible schedule was found, the algorithm marks the job with a "zero" configuration $j_\varnothing$ and the corresponding request is rejected (8). Unlike MMKP-MDF, our algorithm does not stop here and continues scheduling the remaining jobs.

As mentioned in Section 3.2, the scheduling overhead and the quality of generated solutions depend on the amount (and granularity) of supplied operating points. In case of the presented algorithm, the scheduling overhead grows linearly with the number of operating points.

3.3.5 Discussion on the voltage-frequency scaling. While our algorithm aims to generate energy-efficient schedules, one could notice that we do not consider the voltage-frequency scaling of the system, which theoretically may let us achieve further energy reductions. However, building the hybrid approach which interplays with these two components is not straightforward. Suppose several cores are included in the same frequency domain (e.g., as in Odroid XU4), the RM has to determine the proper frequency of all CPUs in the domain. Once frequencies are selected, there should be a reliable way to assess the energy-performance estimations to keep predictability

Table 2. Frequency and power characteristics of the platform cores.

Paremeter	CORTEX-A7	CORTEX-A15	FFT Accelerator
Frequency	1500 MHz	1800 MHz	300 MHz
Idle power	140.3 mW	214.8 mW	—
Dynamic power	320.2 mW	1319.6 mW	62.5 mW

acceptable. One way of addressing it would be generating operating points across multiple frequency values. But this strategy will dramatically increase the number of operating points. Moreover, this amount increases exponentially with the number of frequency domains. Onnebrink et al. [28] proposed a pruning algorithm to generate the Pareto-optimal values across the spatial and frequency components. However, to the best of our knowledge, there are no hybrid methods that interplay both components at runtime and apply it to task-parallel real-time applications.

4 EVALUATION

In this section, we evaluate our approach on a virtual prototype. This virtual prototype includes a virtual platform, an application model of the workload in an LTE physical layer uplink receiver, and an implementation of the hybrid mapping strategy described in the previous section. We create and validate this prototype based on measurements obtained from executions of realistic LTE workloads, as we will describe in more detail in the following. Based on this virtual prototype, we run simulations to compare our hybrid strategy with other state-of-the-art strategies. Due to the nature of a virtual prototype, we can easily extrapolate from the real platform we used for our measurements and evaluate our strategy on a wider range of virtual platforms, simply by adding more cores or accelerators.

4.1 Platform setup

We use the recent open-source Mocasin [26] prototyping tool to create and simulate our virtual prototype. With this framework, we can define virtual platforms and simulate the execution of the workloads in an LTE uplink receiver while using various mapping and scheduling strategies.

Our platform model is based on the Hardkernel Odroid XU4 board featuring a heterogeneous Exynos 5422 big.LITTLE chip with four ARM Cortex-A15 and four ARM Cortex-A7 cores, fixed at frequencies of 1.8 GHz and 1.5 GHz respectively [33]. This platform surely does not provide sufficient computational power to meet the high demands of general LTE base stations. Indeed, the LTE benchmark implementation could not process a single UE with more than 63 PRBs (and any modulation scheme) within the 2.5 ms deadline. Nevertheless, it is a readily available heterogeneous platform that we can extrapolate from. Moreover, the Odroid XU4 platform has already been used for power-efficient realizations of LTE femtocells [9].

For power modeling, we assume that each processor core executes in one of the two modes, run and idle [52][2]. The tasks are assumed to be executed only in the run mode. If the core does not execute the task, it is put into the idle mode. At each of these two power modes, the processor core consumes either a run power P_{run} or an idle power P_{idle}. To calculate the energy consumption of the tasks (Section 3.2), we use the dynamic component of the power, defined as $P_{\text{dyn}} = P_{\text{run}} - P_{\text{idle}}$. We also report the dynamic energy consumption of the platform, defined as the sum of the dynamic energy of the executed tasks.

[2]Zeng et al [52] include a third power mode, inactive. Due to the fine granularity of the workload, processor cores are assumed to be never put in this mode [15].

In order to obtain a power model of Odroid XU4, we measured its power consumption using the ZES Zimmer LMG450 Power Analyzer connected to DC input with an external readout rate of 20 Samples/s. To obtain the dynamic power consumption of the individual cores, we run CPU stressors from the `stress-ng` tool[3] on different subsets of the platform's cores and measured power consumption with the power meter. For the idle power of the cores, we use numbers from the model that is shipped with SLX Tool Suite as part of its commercial platform models. To take into account the power consumption of other components of the platform, we add an extra static power of around 763.3 mW, which is calculated as the difference of the average measured power consumption of the platform in the idle state and the idle power consumption of the processor cores. Table 2 shows the frequency and power model of each core in the platform.

Based on the data for individual cores, we extrapolate from the original Odroid platform and create virtual platforms with a larger number of cores. In our experiments, we use up to 8 cores of each type. To explore how the availability of specific accelerators influences our mapping strategy, we also created a virtual FFT accelerator that we can optionally add to the platform (up to two instances in our experiments). To calibrate the accelerator model, we measure the performance and power characteristics of an FFT implementation in a Xilinx Ultra96 FPGA board (see Table 2). Due to the large overestimation of the idle power consumption of the FPGA, we only report the dynamic energy consumption in our experiments.

4.2 Reference methodologies

We evaluate both the design time and the runtime phases of our proposed hybrid approach by comparing them to reference methods proposed in the literature.

4.2.1 Design Space Exploration. At design-time, we evaluate the generation of the Pareto-optimal operating points w.r.t. design-time overhead and quality of the points. Our design-time approach described in Section 3.2 is referred to as `Phased-Fast`. We compare this to state-of-the-art mapping heuristics, like a multi-objective evolutionary algorithm, similar to the one from the Sesame framework [17], which we denote by `genetic`. Similarly, we use a `simulated annealing` heuristic based on [29] and a `tabu search` one based on the one described in [24]. Finally, we also compare with a simple but fast load-balancing heuristic based on a static version of the Linux CFS scheduler, denoted as `static cfs`. These static approaches are compared in Section 4.5.

4.2.2 Runtime scheduling. As a baseline, we consider the `Work-Stealing` scheduler that resembles the behavior of the scheduler used in the PHY benchmark [37]—an open-source implementation of the LTE uplink receiver workload. The scheduler maintains task queues for each general-purpose resource and assigns all tasks collected from a newly arriving UE to a single queue. Per resource, a worker thread executes tasks from its task queue until the queue is empty. When the queue is empty, the worker thread steals tasks from other queues. Accelerators do not have an associated queue and always steal tasks of a specific type (e.g., FFT) from the work queues assigned to the general-purpose resources.

Along with our `MMKP-MDF-Joint` algorithm, we evaluate two other similar hybrid approaches. To evaluate the impact of the modifications introduced in our algorithm (cf. Section 3.3), we compared to the original `MMKP-MDF` method introduced in [22].

We further compare to the `MMKP-LR` hybrid approach, which uses the Lagrangian relaxation described in [48] to solve the MMKP problem. Internally, it uses a subgradient method (limited to 100 iterations) and iteratively maps applications in the increasing order of the minimum configuration costs. Since the algorithm does not reuse the previously generated schedules, we implemented it in

[3]https://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html

Table 3. Characteristics of LTE traffic traces used for our evaluation. Traces `trace1`-`trace4` are the real LTE traffic traces, while `trace0` is constructed by removing heavy UEs.

Trace	Number of UEs	Non-empty subframes	Average PRBs	Max. PRBs in subframe
`trace0`	587	543	10.96	48
`trace1`	688	603	17.21	100
`trace2`	1473	1197	24.67	100
`trace3`	1577	1342	31.95	100
`trace4`	2091	1736	54.83	100

a way that multiple requests coming in the same subframe are scheduled in an iterative manner (similar to `MMKP-MDF`). We evaluate the runtime phase of all hybrid approaches in Sections 4.6 and 4.7. In these evaluations, we fixed the algorithm at design time to `Phased-Fast`.

4.3 Workload model

The workload in our virtual prototype is modeled as a dataflow graph similar to the one shown in Figure 3. We obtained this graph by analyzing the PHY benchmark [37]—an open-source implementation of the LTE uplink receiver workload. Each instance of the graph represents the workload for processing one UE. The precise size and structure of the graph depend on the parameters of the specific UE (e.g. number of PRBs, number of layers, and modulation scheme). By measuring the execution time of each computational kernel in the PHY benchmark on Odroid XU4 for all possible parameter combinations, we created a detailed model indicating the expected execution time of each task in the task graph $\pi[\tau_\theta]$. Given a power model of the platform, we calculate the dynamic energy of the task on the resource type Ω as $\pi[\xi_\Omega] = P_{\text{dyn}} \cdot \pi[\tau_\Omega]$.

During a simulation run, instances of the graphs are created on the fly based on a workload trace detailing the UEs to process in each sub-frame. The graphs are then mapped to the available computational resources, and the execution of the workload is simulated. The number of UEs, as well as their parameters, may change significantly between sub-frames in the workload trace. This setup thus allows to explore the dynamic behavior in telecommunication applications and analyzing the performance of various mapping and scheduling strategies.

In order to perform our simulation for realistic workloads, we use LTE traffic traces recorded from real base stations. Every trace records the workload processed by a specific base station during a period of time as a list subframes. The subframes are made up of a set of UEs requests with their corresponding parameters: the number of PRBs, the modulation scheme, and the number of layers. These traces were obtained in a five-hour period spread over 15 days and feature real data with over 1.2 million Radio Network Temporary Identifiers (RNTIs) from different base stations [10]. For our simulations we used the first 5000 subframes of four different base stations with varying levels of activity. Additionally, we constructed a trace in which all evaluated approaches reach an almost-perfect success rate, i.e. all UEs were accepted and finished within the deadline (`trace0`). The goal of this trace is to separate the energy savings obtained from rejecting UEs from those gained from more efficient resource utilization. To construct this trace, we used `trace1` and removed the UEs with a large number of PRBs. This resulted in a trace where the maximum number of PRBs in a subframe is always below 50. The main characteristics of the traces are summarized in Table 3. In all used traces, the numbers of antennas and layers are fixed to 4. Due to this, all UEs have the same number of tasks, namely 150 (cf. Figure 1).

4.4 Virtual prototype validation

In order to validate our virtual prototype, we compared the simulated energy consumption and execution time with real measurements obtained by running the PHY benchmark on Odroid XU4. In our simulation, we used a platform configuration with four little and four big cores as well as the `Work-Stealing` scheduler in order to accurately resemble the behavior of the PHY benchmark on the real Odroid XU4.

For the validation, instead of the real traces used later for evaluating the approach, we used 40 pseudo-random LTE traces with different workloads. We generated a varied set of traces, with lengths ranging from 100 to 1000 subframes. The parameterization of the subframes also varies from trace to trace, some of them are based on real LTE traces, while the rest were parameterized by following a random approach. A random number of a maximum of 10 UEs is created for every subframe, with each UE having a small number of PRBs for the first traces, we then progressively increase the number of PRBs for the successive traces. In this way, we generated traces with different processing intensity demands. The actual PHY benchmark does not interrupt the processing of UEs that missed their deadline, which we also accounted for in the simulation.

Our simulation is deterministic, while the execution on the actual platform is not. As it cannot account for the non-determinism, our simulation aims to model the average behavior of the platform. For this reason, for each constructed trace, we measured the execution time and energy consumption on Odroid XU4 10 different times and recorded the average. We then simulated the LTE uplink receiver processing in our virtual prototype and compared it to the average performance in the real platform.

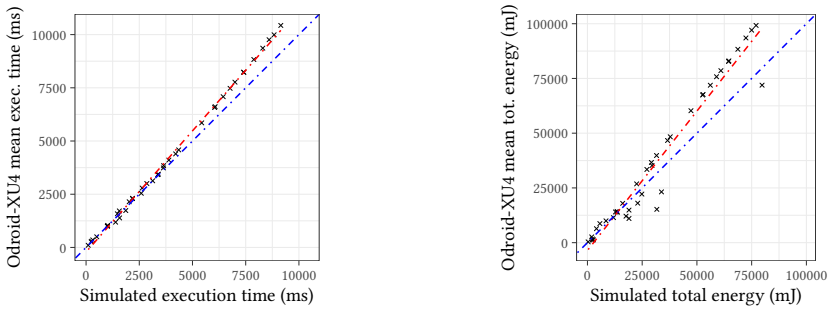


Fig. 5. Validation of our LTE prototype. Every point represents the simulated and its corresponding average measured execution time and energy (over 10 measurements) for a random workload.

Figure 5 presents the results of this validation, showing a dot for each trace. The blue lines represent an ideal correlation between measured and simulated values. The red line corresponds to a linear regression over the observations. In terms of fidelity, the data features a high Spearman's correlation of $\rho_{\text{execution time}} = 0.999$ and $\rho_{\text{energy}} = 0.972$. This indicates that we can reliably compare the effects of various mapping strategies in Mocasin since a lower estimated simulation time also indicates better performance in the real platform.

4.5 Generation and estimation of operating points

As discussed in Section 3.2, we generate (approximately Pareto-optimal) operating points for all valid combinations of processor resources in the platform. For example, for Odroid XU4, we generate 24 operating points, and for the virtual platform with the accelerators, 72 points are generated. These operating points are saved internally as an array, with each element representing a single task in the task graph, and the value is a number ($[1..|\Omega|]$) representing the processor resource. For example, assuming compact encoding on the Odroid XU4 (3 bit per value), this array would

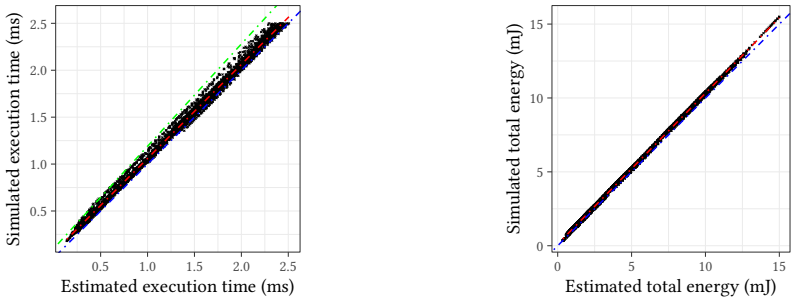


Fig. 6. Validation of our estimation of execution time and energy consumption on Odroid XU4.

require 450 bits, or 57 bytes, *plus* 8 bytes for the energy-performance metadata. The whole set of operating points would thus require 1560 bytes.

Apart from the mapping generation, Algorithm 1 also estimates the latency and energy of each generated mapping. In this section, we evaluate our proposed method in two aspects. First, we validate the estimated latency and energy values against the simulation. Second, we compare our algorithm with the aforementioned approaches w.r.t. generation overhead and quality of the operating points.

4.5.1 Energy-performance data validation. To validate the estimations of energy and performance characteristics of the generated operating points, we compare them with the corresponding values obtained in simulations using Mocasin. Using Algorithm 1, we generated in total 12000 operating points: 24 operating points for 500 different types of UEs, in which we varied the number of PRBs and the modulation scheme. Figure 6 show the validation points on Odroid XU4. The plots only include the points for which the simulated time does not exceed the highest deadline for processing a UE, namely 2.5 ms. Once again, the blue line corresponds to the perfect correlation and the red line to the result of linear regression. As seen in the figures, the values obtained in the fast estimation have high accuracy. The energy estimations are almost ideal, whereas for execution times the plot show slightly higher dispersion, which is explained by the lacked of modeling of communication delays.

Since the runtime resource manager relies on the exact values of execution times, we attempt to find optimal effective execution times after the estimation. For that we express the effective execution time as the linear function of the estimated time: $t_{\text{eff}} = a \cdot t_{\text{est}} + b$. Then we simulate the runtime resource manager on a sample trace with different values of a and b and choose the values that minimize the number of UE rejections and deadline misses. Figure 6 shows the effective time with the green lines, which goes right at the top edge of the validation points. This means the formula for effective execution time overestimates the actual execution times only slightly, which, in turn, improves the predictability at runtime.

4.5.2 Comparison with the state-of-the-art mapping methods. We now compare our `Phased-Fast` heuristic with the other static mapping methodologies mentioned in Section 4.2. To this end, we execute the different mapping algorithms to find static mappings for UEs with multiple different workloads, ranging between 5 and 100 PRBs. For each of these UEs, we execute the five different static mapping heuristics, `Phased-Fast`, `genetic`, `tabu search`, `simulated annealing` and `static cfs`.

We evaluate the heuristics in terms of three measures: (i) the exploration time, which corresponds to the execution time of the mapping algorithm itself and thus to the design-time overhead, (ii) the mapping runtime, which is the execution time of the UE in the simulation, and measures the

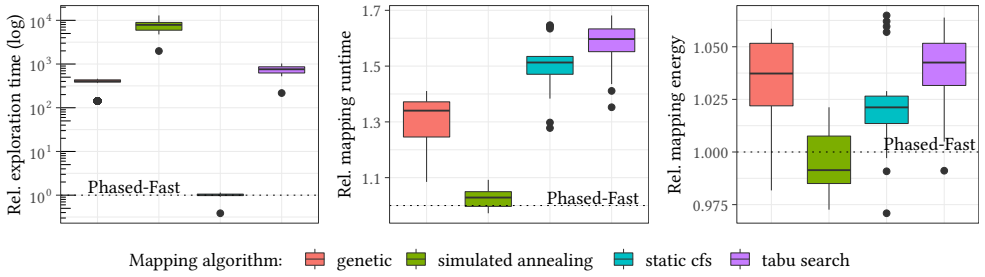


Fig. 7. Comparison of multiple static mapping algorithms relative to our Phased-Fast heuristic as baseline.

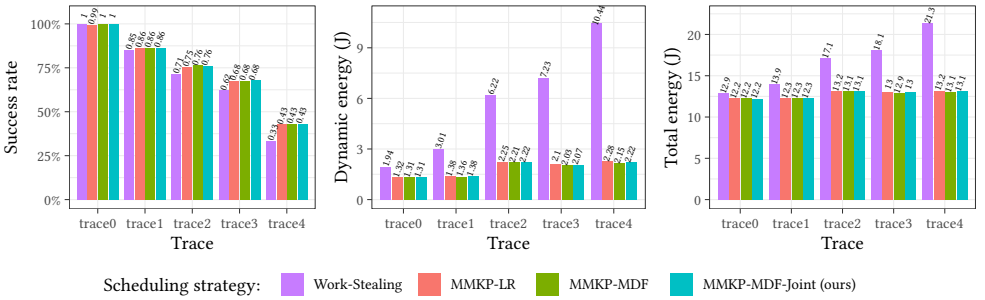


Fig. 8. Successfully executed UEs and energy efficiency on Odroid XU4.

quality of the mapping, and (iii), the mapping energy consumption, which is a different measure of quality in this multi-objective setting. For all three of these metrics, a lower value is better.

Figure 7 shows the comparison of the static mapping heuristics, using our `Phased-Fast` method as baseline (indicated by the dotted line). This way, we can also uniformly compare the different workloads of the UEs. In terms of exploration time, `Phased-Fast` is the fastest of the heuristics, taking just 0.36 s in median. Out of the other heuristics, only `static cfs` is not a meta-heuristic, i.e., it does not iteratively evaluate mappings. Consequently, it is the only one that is comparatively fast (in median only about 0.5% slower than `Phased-Fast`).

In terms of mapping quality, only the `simulated annealing` heuristic produces results comparable to our domain-specific static mapping heuristic. While our mappings result to be slightly faster, they also consume slightly more energy. Overall, the mapping quality of the two heuristics is comparable, yet significantly better than all the other state-of-the-art heuristics for this concrete problem. The only heuristic producing comparable results is around 4 orders of magnitude slower than our method. Due to the huge parameter space of UEs and the enormous number of tasks in the task graphs, it is important to have fast mapping heuristics, even at design time.

4.6 Energy-efficient runtime mapping

Apart from energy consumption, we are interested in two additional metrics, namely, the percentage of UEs executed successfully within the real-time constraints, and the resource utilization, measured as the resources required in a virtual platform to achieve a given concrete performance. We divide this section into two parts: First, we evaluate all methods on the two fixed platforms, namely Odroid XU4 and the virtual platform with the accelerators. Then we vary the platform sizes to also evaluate the resource utilization.

4.6.1 Evaluation on fixed platforms. Figure 8 shows the evaluation of the different LTE traces (cf. Table 3) on Odroid XU4. On the left, it shows the percentage of UEs executed successfully

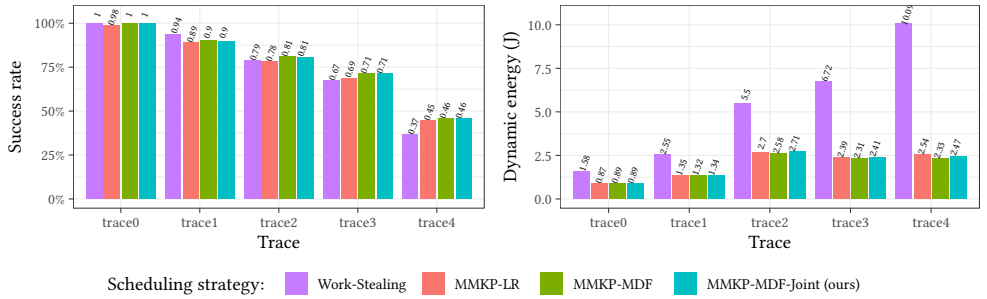


Fig. 9. Successfully executed UEs and energy efficiency on a virtual platform with accelerators.

within the real-time deadline. In general, all the three hybrid approaches show better success rates than the `Work-Stealing` scheduler on the real LTE traces. The difference becomes more notorious for higher intensity of the workload, where our approach reaches 29.8 % better success rate on the most intensive `trace4` compared to `Work-Stealing`. Compared to the other hybrid schedulers, `MMKP-MDF-Joint` behaves slightly better on 3 out of 4 real LTE traces. The differences are mostly negligible, however. In terms of energy efficiency, all hybrid approaches are also on par and show better energy-efficiency than the baseline: the dynamic energy savings of schedules generated with our joint algorithm varies from 54.1 % to 78.7 % on the real LTE traces compared to the work-stealing scheduling. On the other hand, the differences with respect to the other hybrid approaches is at most 3.2 % for the most intensive workload. A dynamic energy saving of 32.7 % for `trace0` shows the high potential of hybrid approaches that, compared to the baseline, save energy only by selecting energy-efficient configurations at runtime. In addition, the hybrid approaches can save further energy for intensive workloads, by not executing rejected requests.

Similarly to the previous plots, the total energy savings reported in the right plot of Figure 8, behave almost identically for all the three hybrid approaches, with negligible differences. Depending on the workload intensity, our approach saves between 4.9 % and 38.5 % on all traces with respect to the `Work-Stealing` algorithm. While total energy savings are more interesting to engineers, dynamic energy savings show the potential of this approach on the platforms with better energy proportionality.

As seen in Figure 9, the results on the virtual platform which augments Odroid XU4 with FFT accelerators are very similar to ones obtained on the platform without accelerators. In general, all hybrid approaches are on par both in terms of successfully executed UEs and energy-efficiency. Compared to the baseline, our approach schedules 4.3 % less requests on the lightest real trace, `trace1`. On all other traces, `MMKP-MDF-Joint` shows a better Quality of Service (QoS) with a maximum improvement of 24.6 %. Our also algorithm reduces the dynamic energy consumption by an amount between 43.5 % and 75.5 %. Since we do not have a good static model for the accelerators, we do not report the total energy savings for this virtual platform.

4.6.2 Varying the amount of resources. The success rates of real traces reported in Figures 8 and 9 are unacceptably low for any production-level base station. Even increasing the number of cores in a virtual platform, the Cortex-A7 and Cortex-A15 cores executing the PHY bench implementation are not fast enough to deal with all individual workloads on time. Indeed, both Odroid XU4 and the PHY bench implementation are ill-suited for production-level baseband processing, even for today's LTE requirements. These values serve as a proof-of-concept evaluation to extrapolate to the higher demands of future technologies.

In a real base station, the QoS requirements stipulate a minimal success rate that should be met. As such, comparing the success rate of the approaches is not as conclusive. To alleviate this, we

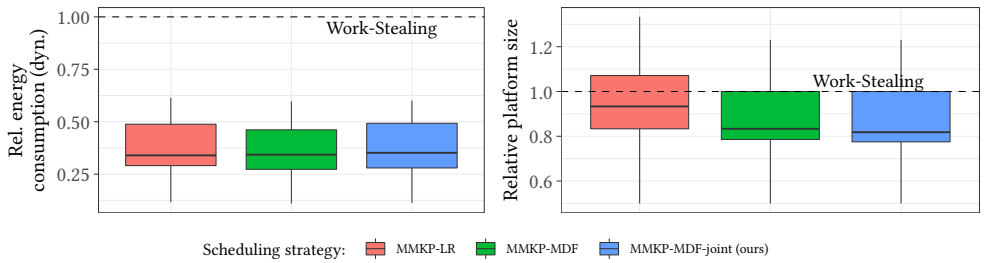


Fig. 10. Relative dynamic energy consumption and resource utilization of the hybrid mapping approach while controlling for the success rate.

controlled for the success rate. Using our virtual platform, we increased the number of Cortex-A7 and Cortex-A15 on each of the traces, improving the success rates for all the strategies. We then compared the approaches on different virtual platforms, matching the instances that achieved the same success rates with the baseline (up to a difference of 2%). This yielded 112 distinct points, grouped in pairs, each achieving the same performance with a different number of resources (cores) and different energy consumption values. Equating the performance this way, we compared the relative energy consumption of the baseband using each of the approaches, as well as the total number of processing elements (including accelerators)[4]. Note that we only compared virtual platforms with accelerators to other virtual platforms with accelerators, and similarly those without accelerators only against other configurations without them. Mixing between the two groups is not very meaningful.

Figure 10 shows the relative energy consumption and resource utilization of the hybrid mapping algorithms, compared to the `Work-Stealing` scheduler (which is the baseline of 1). Our dynamic mapping algorithm `MMKP-MDF-Joint` achieves a median dynamic energy consumption that is only 35.2 % of the energy consumption of the work-stealing scheduler. These gains in energy consumption are achieved with comparable amounts of resources: the median relative resource utilization is 82 % of the resource utilization of the work-stealing scheduler. In other words, in the median case, our hybrid mapper achieved the same performance than a dynamic work-stealing scheduler, with slightly lower hardware requirements, while only consuming around a third of the dynamic energy. Beyond the concrete heuristics, this paper's main concern is to analyze the feasibility of hybrid methodologies for baseband processing. However, the performance of our proposed heuristics in comparison to other state-of-the-art hybrid approaches is still worth comparing. Compared to ours, the two other approaches `MMKP-LR` and `MMKP-MDF` both achieved about a percentage point less dynamic energy consumption, while using 11 and 1 percentage points more relative resources in the median. The differences in terms of performance are negligible, especially considering the trade-off between energy consumption and resource utilization. The main advantage to these approaches comes in terms of the runtime overhead, which we will continue to evaluate in Section 4.7.

It is worth nothing that by resource utilization we do not mean the dynamic resource utilization, i.e. the core utilization at runtime. Instead, as explained above, we mean the hardware required in the virtual platform to achieve a given (equal) success rate for a given trace. Both when designing hardware, as well as when assigning resources e.g. in a load balancer in a cRAN environment, the important metric is how many PEs should be included in or assigned to the system.

Due to the absence of the static energy model for the accelerators, we exclude these from the values for the total energy consumption. These values are shown in Figure 11, only for the virtual

[4]Whenever two instances of the same scheduling approach on the same trace had the same success rate, we chose the one with the minimal resource utilization/energy consumption.

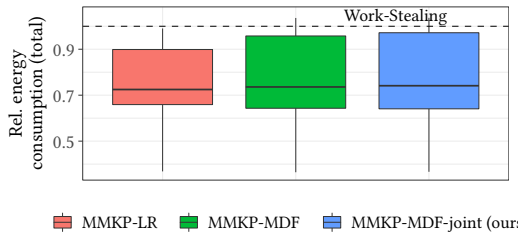


Fig. 11. Relative total energy consumption of the hybrid mapping approach while controlling for the success rate on the virtual platforms without accelerators.

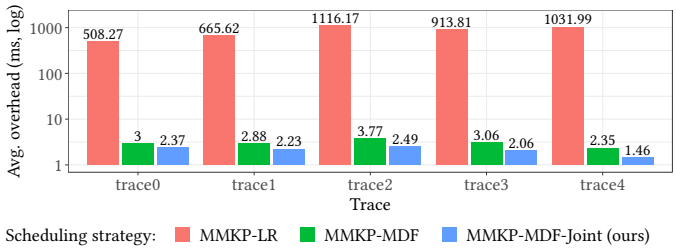


Fig. 12. Average scheduling overhead per RM activation on Odroid XU4.

platforms without accelerators. In this case, the reduction is less radical, with a median relative total energy consumption of 74.1 %. It is nevertheless a significant reduction.

4.7 Runtime overhead analysis

We have seen how greatly the hybrid approaches can improve the energy efficiency of baseband processing. However, the more complex scheduling decisions also incur a scheduling overhead. In this section we analyze the runtime overhead of all three hybrid solutions.

Figure 12 presents the average scheduling time per RM activation on Odroid XU4 for all the hybrid approaches. These values are not comparable with the execution times of the benchmark, they are measured on our prototype implementation written in Python 3 using Mocasin [26], which runs on a system with a 3.40 GHz Intel Core i7-6700 CPU and 32 Gb RAM. The relative execution times between the heuristics, however, are comparable, since they are all implemented in this framework.

As seen in the figure, the scheduling overhead of `MMKP-MDF` and `MMKP-MDF-Joint` is drastically smaller than that of `MMKP-LR`, they behave at least 99.5 % better for all evaluated LTE traces. Comparing `MMKP-MDF`-based approaches, we see that our proposed solution requires less scheduling time. Moreover, the speedup increases with the workload intensity, from 20.8 % on the lightest `trace0` till 37.7 % on the heaviest `trace4`. This result highlights the positive impact of our proposed modifications to the scheduling algorithm.

While our algorithm is ahead of other hybrid approaches, the absolute values of the scheduling overhead are not acceptable for a real implementation of the runtime resource manager. As mentioned above, our current prototype is executed with the Python interpreter, causing a large overhead. We expect that a real scheduler written efficiently in a low-level programming language could execute $10 - 200\times$ faster[5].

[5]This expectation is based on the performance evaluation of other benchmarks: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/gpp-python3.html. While this is not a scientifically rigorous comparison, it serves as a guidance for the order of magnitude of improvement we could reasonably expect.

5 RELATED WORK

Modern communication standards are changing the system architecture of base stations. cRAN architectures have been proposed for 5G, where distributed remote radio-heads (RRHs) with incorporated antennas preprocess incoming streams before forwarding them to a centralized pool of BBUs [3]. These architectures are enabled by Network Function Virtualization (NFV) [7], which allows the control plane to dynamically activate BBUs according to the network traffic demands. Our novel mapping approach is an orthogonal key enabling methodology that allows more efficient utilization of BBUs for future communication standards.

The need for flexibility has motivated plenty of research on programmable platforms for baseband processing. Early works successfully deployed BBU-side LTE baseband processing onto general purpose processors (GPPs), using single instruction multiple data (SIMD) and function look-up tables for reduced latency [20], as well as multi-threading [50]. The latter uses static mappings, so it cannot adapt to dynamic changes in the workload. The open-source baseband processing implementation PHY bench [37] utilizes a work-stealing task scheduler to evenly distribute the current workload on the available GPPs at run time. However, as we have shown in this paper, such a purely dynamic approach does not benefit from the trade-offs a heterogeneous architecture provides. A hybrid strategy can better exploit knowledge about the platform and, for instance, use trade-offs to achieve better energy efficiency.

For a small cell base station, the authors in [38] proposed a heterogeneous multi-processor platform SoC, with integrated hardware accelerators and application specific instruction-set processors (ASIPs). For resource management, the authors use a task graph and describe a mapping approach that incurs high power consumption. More recent approaches introduce an array-based multi-core architecture for baseband processing [44, 45], paired with a mapping methodology based on dataflow models. However, the mapping methodologies are tied to the specific hardware architecture and cannot be easily extended to support heterogeneous platforms. Although the baseband application contains certain irregularities potentially leading to divergent control, authors in [23] proposed a novel algorithm that is well-suited to a GPU cluster.

Researchers have also leveraged low-power techniques such as dynamic voltage frequency scaling (DVFS) and power gating in multi-cores to further improve the efficiency of programmable BBUs, e.g., for an Odroid XU3 in [9] and the TILEPro64 platform in [37]. As discussed in Section 3.3, power management techniques can further improve the energy efficiency achieved by our approach. The interplay between such techniques and our domain-specific hybrid mapping algorithm is an important direction for future work. In this paper, we adapted a more generic model-based methodology for resource management in baseband processing that can be applied to different platforms. In the future, such a methodology can enable moving computation across virtualized BBUs.

There is plenty of work on generic model-based mapping methodologies [11]. For example, SystemCoDesigner [19] maps the discrete-event model in SystemC to design digital signal processing systems. Similarly, flows like Sesame [30], DOL [41], MAPS [12] or SDF^3 [39] use models like Kahn Process Networks or Synchronous Dataflow (SDF) for model-based design in embedded systems. These approaches are foremost static, with a focus on design-space exploration (DSE) at compile-time [17, 24, 29] to derive efficient mappings of computation and data to hardware resources.

In the last decade, the focus on DSE shifted to more dynamic methods, such as [5, 32, 47], which combine this static DSE with run-time decisions in hybrid approaches. The focus of these approaches is generally performance or resource utilization, although in many cases, the general formulations as optimization problems permit optimizing for energy consumption as well. Authors

in [16, 22, 42] specifically target energy efficiency, achieving better results than general problem formulations, at the cost of generality. To better cater for baseband processing applications, our work in this paper further specializes the methodology for computational graphs such as the ones described in Section 2.

Most of the model-based approaches mentioned above were designed and validated on applications with modest model sizes, which seldom exceed a couple of dozen dataflow tasks or actors. Computations in baseband processing, on the other hand, can quickly exceed many hundreds of tasks when processing multiple UEs in parallel. We do not propose a novel general hybrid mapping approach, but instead show how we can specialize existing approaches to the task of baseband processing, leveraging the structure of the application to handle the large application sizes and refining the scheduling strategy. To the best of our knowledge, no previous work has combined the advantages of hybrid approaches with domain knowledge specific to baseband processing. This allows us to be more flexible than hardware solutions and more efficient than purely dynamic software solutions.

6 CONCLUSION

The demands for performance and flexibility of baseband processing rise with changing telecommunication standards. We require novel methods to mitigate the increase in energy consumption associated with the rise in requirements. In this paper we have investigated hybrid mapping methodologies from model-based design for baseband processing. In particular, we have shown they are a viable avenue to address the upcoming demands of flexibility and high energy efficiency.

Hybrid mappings allow us to combine specialized and heterogeneous hardware with highly efficient software-based implementations and flexibility at runtime. Our domain-specific approach leverages the particular computational structure of the task graphs, as well as refines the scheduling methodology, allowing it to scale to the domain of baseband processing. Indeed, the performed evaluations highlight significant improvements in different aspects. At design time, our fast mapping algorithm generates mappings that have a similar quality to those obtained by the state-of-the-art `simulated annealing` algorithm, but those mappings were generated almost four orders of magnitude faster. At runtime, our evaluation on multiple virtual platforms showed the potential energy savings that structured hybrid approaches could bring over traditional software scheduling. In a median execution, with the same performance using the same hardware resources, all explored hybrid approaches, including ours, were on par and consumed about a third of the dynamic energy, compared to a work-stealing software scheduler. Compared against the other existing hybrid methods, our proposed solution also shows a significant reduction, up to 37.7 %, on runtime overhead against the fastest state-of-the-art `MMKP-MDF` solution. As shown in the evaluation, the models in our virtual platforms were validated on an Odroid XU4, showing extremely high fidelity. This means that the improvements observed in simulations are likely to translate directly to improvements in silicon.

This work serves as a proof-of-concept, showing the potential of hybrid approaches to baseband processing. Several limitations ought to be addressed in future work to show the viability without reservations. The PHY Bench benchmark is based on the less dynamic LTE standard, and as an academic benchmark, it does not boast the same level of optimization as a production-level implementation would. Moreover, the Odroid XU4 board is not well-suited as a BBU to deal with the high demands of 5G and beyond. Future work should test hybrid approaches like ours on more demanding benchmarks for newer standards and higher-performing platforms.

On the side of the hybrid mapping and scheduling strategies, many opportunities remain open for further research. For example, the Pareto point selection can be pruned further at compile-time, with more sophisticated heuristics to preserve only a handful of relevant mappings. This

will also reduce the runtime overhead. More importantly, the presented here approach still has a potential for a further energy reduction by interplaying with the voltage-frequency scaling. In future work, we plan to better understand and push the limits of our approach. While these limitations are important, they do not cancel out the usefulness of hybrid mappings. If the approach ceases providing a net improvement at some threshold, a simple fall-back mechanism can be used to ensure we can still reap the benefits before reaching that threshold. This should ultimately still lead to significant improvements in energy consumption, which are crucial in today's and tomorrow's energy-constrained world.

ACKNOWLEDGMENTS

From Tulika Mitra's group, we thank Arka Maity for providing a modified version of the PHY benchmark supporting workload descriptions, and Nishant Budhdev for providing the LTE traffic traces from real base stations. We thank also Silexica for providing the power model of Odroid XU4.

This work was funded in part by Dr. James Truchard, by National Instruments, by the the German Federal Ministry of Education and Research (BMBF) through the E4C project (16ME0189), by the German Research Foundation (DFG) within ROSI (GRK 1907) and TraceSymm (366764507), and by the Studienstiftung des Deutschen Volkes.

REFERENCES

[1] 2017. *Handbook of Hardware/Software Codesign.* Springer Netherlands.

[2] 3GPP. 2017. *Evolved Universal Terrestrial Radio Access (E-UTRA); Physical channels and modulation.* Technical Specification (TS) 36.211. 3rd Generation Partnership Project (3GPP). Version 14.2.0.

[3] Aini Li, Yan Sun, Xiaodong Xu, and Chunjing Yuan. 2016. An energy-effective network deployment scheme for 5G Cloud Radio Access Networks. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS).* 684–689. https://doi.org/10.1109/INFCOMW.2016.7562164

[4] Ali Alnoman and Alagan Anpalagan. 2017. Towards the fulfillment of 5G network requirements: technologies and challenges. *Telecommunication Systems* 65, 1 (2017), 101–116. https://doi.org/10.1007/s11235-016-0216-9

[5] Giuseppe Ascia, Vincenzo Catania, and Maurizio Palesi. 2004. Multi-objective mapping for mesh-based NoC architectures. In *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis.* 182–187.

[6] Felice Balarin, Paolo Giusto, Attila Jurecska, Michael Chiodo, Claudio Passerone, Ellen Sentovich, Harry Hsieh, Luciano Lavagno, Bassam Tabbara, Alberto Sangiovanni-Vincentelli, et al. 1997. *Hardware-software co-design of embedded systems: the POLIS approach.* Springer Science & Business Media.

[7] Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hines. 2020. 5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges. *Computer Networks* 167 (2020), 106984. https://doi.org/10.1016/j.comnet.2019.106984

[8] Sandro Belfanti, Christoph Roth, Michael Gautschi, Christian Benkeser, and Qiuting Huang. 2013. A 1Gbps LTE-advanced turbo-decoder ASIC in 65nm CMOS. In *2013 Symposium on VLSI Circuits.* C284–C285.

[9] N. Budhdev, M. C. Chan, and T. Mitra. 2018. PR3: Power Efficient and Low Latency Baseband Processing for LTE Femtocells. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications.* 2357–2365. https://doi.org/10.1109/INFOCOM.2018.8486276

[10] Nishant Budhdev, Mun Choon Chan, and Tulika Mitra. 2020. IsoRAN: Isolation and Scaling for 5G RANvia User-Level Data Plane Virtualization. arXiv:2003.01841 [cs.NI]

[11] Jeronimo Castrillon, Karol Desnos, Andrés Goens, and Christian Menard. 2021. Dataflow Models of Computation for Programming Heterogeneous Multicores. In *Handbook of Computer Architecture (to appear)*, Anupam Chattopadhyay et al. (Ed.). Springer.

[12] Jeronimo Castrillon, Rainer Leupers, and Gerd Ascheid. 2013. MAPS: Mapping Concurrent Dataflow Applications to Heterogeneous MPSoCs. *IEEE Transactions on Industrial Informatics* 9, 1 (Feb. 2013), 527–545. https://doi.org/10.1109/TII.2011.2173941

[13] Jeronimo Castrillon, Stefan Schürmans, Anastasia Stulova, Weihua Sheng, Torsten Kempf, Rainer Leupers, Gerd Ascheid, and Heinrich Meyr. 2011. Component-based Waveform Development: The Nucleus Tool Flow for Efficient and Portable Software Defined Radio. *Analog Integrated Circuits and Signal Processing* 69, 2-3 (Dec. 2011), 173–190. https://doi.org/10.1007/s10470-011-9670-1

[14] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann. 2015. Cloud RAN for Mobile Networks — A Technology Overview. *IEEE Communications Surveys Tutorials* 17, 1 (2015), 405–426. https://doi.org/10.1109/COMST.2014.2355255

[15] Alexei Colin, Arvind Kandhalu, and Ragunathan (Raj) Rajkumar. 2016. Energy-Efficient Allocation of Real-Time Applications onto Single-ISA Heterogeneous Multi-Core Processors. *J. Signal Process. Syst.* 84, 1 (July 2016), 91â€“110. https://doi.org/10.1007/s11265-015-0987-3

[16] Anup Das, Bashir M Al-Hashimi, and Geoff V Merrett. 2016. Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)* 15, 2 (2016), 1–25.

[17] C. Erbas, S. Cerav-Erbas, and A.D. Pimentel. 2006. Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. *IEEE Transactions on Evolutionary Computation* 10, 3 (2006), 358–374. https://doi.org/10.1109/TEVC.2005.860766

[18] A. Goens, R. Khasanov, M. Hähnel, T. Smejkal, H. Härtig, and J. Castrillon. 2017. 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)* (Sankt Goar, Germany) *(SCOPES '17)*. ACM, New York, NY, USA, 11–20. https://doi.org/10.1145/3078659.3078663

[19] Christian Haubelt, Joachim Falk, Joachim Keinert, Thomas Schlichter, Martin Streubühr, Andreas Deyhle, Andreas Hadert, and Jürgen Teich. 2007. A SystemC-based design methodology for digital signal processing systems. *EURASIP Journal on Embedded Systems* 2007 (2007), 1–22.

[20] N. Kai, S. Jianxing, H. Zhiqiang, and K. K. Chai. 2012. LTE eNodeB prototype based on GPP platform. In *2012 IEEE Globecom Workshops*. 279–284. https://doi.org/10.1109/GLOCOMW.2012.6477583

[21] Manupa Karunaratne, Aditi Kulkarni Mohite, Tulika Mitra, and Li-Shiuan Peh. 2017. Hycube: A cgra with reconfigurable single-cycle multi-hop interconnect. In *Proceedings of the 54th Annual Design Automation Conference 2017*. 1–6.

[22] Robert Khasanov and Jeronimo Castrillon. 2020. Energy-efficient Runtime Resource Management for Adaptable Multi-application Mapping. In *Proceedings of the 2020 Design, Automation and Test in Europe Conference (DATE)* (Grenoble, France) *(DATE '20)*. IEEE, 909–914. https://doi.org/10.23919/DATE48585.2020.9116381

[23] Kaipeng Li, Rishi Sharan, Yujun Chen, Tom Goldstein, Joseph R. Cavallaro, and Christoph Studer. 2017. Decentralized Baseband Processing for Massive MU-MIMO Systems. arXiv:1702.04458 [cs.IT]

[24] Sorin Manolache, Petru Eles, and Zebo Peng. 2008. Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints. *ACM Transactions on Embedded Computing Systems (TECS)* 7, 2 (2008), 1–35.

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

[26] Christian Menard, Andrés Goens, Gerald Hempel, Robert Khasanov, Julian Robledo, Felix Teweleitt, and Jeronimo Castrillon. 2021. 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* (Budapest, Hungary) *(DroneSE and RAPIDO '21)*. Association for Computing Machinery, New York, NY, USA, 66–73. https://doi.org/10.1145/3444950.3447285

[27] Mina Niknafs, Ivan Ukhov, Petru Eles, and Zebo Peng. 2019. Runtime resource management with workload prediction. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[28] Gereon Onnebrink, Ahmed Hallawa, Rainer Leupers, Gerd Ascheid, and Awaid-Ud-Din Shaheen. 2019. A Heuristic for Multi Objective Software Application Mappings on Heterogeneous MPSoCs. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference* (Tokyo, Japan) *(ASPDAC '19)*. Association for Computing Machinery, New York, NY, USA, 609â€“614. https://doi.org/10.1145/3287624.3287651

[29] Heikki Orsila, Tero Kangas, Erno Salminen, Timo D. Hämäläinen, and Marko Hännikäinen. 2007. Automated memory-aware application distribution for multi-processor system-on-chips. *J. of Sys. Arch.* 53, 11 (2007), 795–815.

[30] A. D. Pimentel, C. Erbas, and S. Polstra. 2006. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Trans. Comput.* 55, 2 (2006), 99–112.

[31] Claudius Ptolemaeus (Ed.). 2014. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org. http://ptolemy.org/books/Systems

[32] W. Quan and A. D. Pimentel. 2015. A hybrid task mapping algorithm for heterogeneous MPSoCs. *ACM Transactions on Embedded Computing Systems (TECS)* 14, 1 (2015), 14.

[33] Rob Roy and Venkat Bommakanti. [n.d.]. *ODROID-XU4 User Manual*. Hardkernel. https://magazine.odroid.com/wp-content/uploads/odroid-xu4-user-manual.pdf

[34] W. Saad, M. Bennis, and M. Chen. 2020. A Vision of 6G Wireless Systems: Applications, Trends, Technologies, and Open Research Problems. *IEEE Network* 34, 3 (2020), 134–142. https://doi.org/10.1109/MNET.001.1900287

[35] Shahriar Shahabuddin, Aarne Mämmelä, Markku Juntti, and Olli Silvén. 2021. ASIP for 5G and Beyond: Opportunities and Vision. *IEEE Transactions on Circuits and Systems II: Express Briefs* 68, 3 (2021), 851–857.

[36] Chung-Ching Shen, William Plishker, Hsiang-Huang Wu, and Shuvra S Bhattacharyya. 2010. A lightweight dataflow approach for design and implementation of SDR systems. In *Proceedings of the Wireless Innovation Conference and Product Exposition*. Citeseer, 640–645.

[37] Magnus Själander, Sally McKee, Peter Brauer, David Engdal, and Andras Vajda. 2012. An LTE Uplink Receiver PHY Benchmark and Subframe-based Power Management. In *2012 IEEE International Symposium on Performance Analysis of Systems Software*. 25–34. https://doi.org/10.1109/ISPASS.2012.6189203

[38] M. Srinivasan, C. S. R. Murthy, and A. Balasubramanian. 2015. Modular performance analysis of Multicore SoC-based small cell LTE base station. In *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. 37–42. https://doi.org/10.1109/VLSI-SoC.2015.7314388

[39] Sander Stuijk, Marc Geilen, and Twan Basten. 2010. A predictable multiprocessor design flow for streaming applications with dynamic behaviour. In *2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*. IEEE, 548–555.

[40] F. Tariq, M. R. A. Khandaker, K. K. Wong, M. A. Imran, M. Bennis, and M. Debbah. 2020. A Speculative Study on 6G. *IEEE Wireless Communications* 27, 4 (2020), 118–125. https://doi.org/10.1109/MWC.001.1900488

[41] L. Thiele, I. Bacivarov, W. Haid, and K. Huang. 2007. Mapping applications to tiled multiprocessor embedded systems. In *Application of Concurrency to System Design, 2007. ACSD 2007. Seventh International Conference on*. IEEE, 29–40.

[42] Stavros Tzilis, Pedro Trancoso, and Ioannis Sourdis. 2019. Energy-efficient runtime management of heterogeneous multicores using online projection. *ACM Transactions on Architecture and Code Optimization (TACO)* 15, 4 (2019), 1–26.

[43] T. Ulversoy. 2010. Software Defined Radio: Challenges and Opportunities. *IEEE Communications Surveys Tutorials* 12, 4 (2010), 531–550. https://doi.org/10.1109/SURV.2010.032910.00019

[44] V. Venkataramani, B. Bodin, A. Kulkarni, T. Mitra, and L. S. Peh. 2020. Time-Predictable Software-Defined Architecture with Sdf-Based Compiler Flow for 5g Baseband Processing. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 1553–1557. https://doi.org/10.1109/ICASSP40776.2020.9054285

[45] Vanchinathan Venkataramani, Aditi Kulkarni, Tulika Mitra, and Li-Shiuan Peh. 2020. SPECTRUM: A Software-Defined Predictable Many-Core Architecture for LTE/5G Baseband Processing. *ACM Trans. Embed. Comput. Syst.* 19, 5, Article 32 (Sept. 2020), 28 pages. https://doi.org/10.1145/3400032

[46] X. Wang, C. Cavdar, L. Wang, M. Tornatore, H. S. Chung, H. H. Lee, S. M. Park, and B. Mukherjee. 2017. Virtualized Cloud Radio Access Network for 5G Transport. *IEEE Communications Magazine* 55, 9 (2017), 202–209. https://doi.org/10.1109/MCOM.2017.1600866

[47] A. Weichslgartner, D. Gangadharan, S. Wildermann, M. Glaß, and J. Teich. 2014. 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*. IEEE, 1–10.

[48] Stefan Wildermann, Andreas Weichslgartner, and Jürgen Teich. 2015. Design methodology and run-time management for predictable many-core systems. In *2015 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. IEEE, 103–110.

[49] Robert Wittig, Andrés Goens, Christian Menard, Emil Matus, Gerhard P. Fettweis, and Jeronimo Castrillon. 2020. Modem Design in the Era of 5G and Beyond: The Need for a Formal Approach. In *Proceedings of the 27th International Conference on Telecommunications (ICT)* (Virtual. Bali, Indonesia). 1–5. https://doi.org/10.1109/ICT49546.2020.9239539

[50] Xiaofeng Tao, Yanzhao Hou, Haiyang He, Kaidong Wang, and Yingyue Xu. 2012. GPP-based soft base station designing and optimization (invited paper). In *7th International Conference on Communications and Networking in China*. 49–53. https://doi.org/10.1109/ChinaCom.2012.6417446

[51] A. Zaidi, F. Athley, J. Medbo, U. Gustavsson, G. Durisi, and X. Chen. 2018. *5G Physical Layer: Principles, Models and Technology Components*. Elsevier Science. https://books.google.de/books?id=mtJKDwAAQBAJ

[52] G. Zeng, T. Yokoyama, H. Tomiyama, and H. Takada. 2009. Practical Energy-Aware Scheduling for Real-Time Multiprocessor Systems. In *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. 383–392. https://doi.org/10.1109/RTCSA.2009.47