HetSim: A Simulator for Task-based Scheduling on Heterogeneous Hardware

Marcel Lütke Dreimann
Universität Osnabrück
Osnabrück, Germany
marcel.luetkedreimann@uos.de

Birte Friesel
Universität Osnabrück
Osnabrück, Germany
birte.friesel@uos.de

Olaf Spinczyk
Universität Osnabrück
Osnabrück, Germany
olaf@uos.de

ABSTRACT

Server hardware is becoming more and more heterogeneous, with an increasingly diverse landscape of accelerators such as GPUs, FPGAs, or novel processing-in-memory (PIM) technologies. Designing and evaluating scheduling algorithms for these is far from trivial due to accelerator-specific setup costs, compute capabilities, and other characteristics. In fact, many existing scheduling simulators only consider some of these characteristics, or only support a specific sub-set of accelerators. To overcome these challenges, we present *HetSim*, a modular simulator for task-based scheduling on heterogeneous hardware. HetSim enables research on online and offline scheduling and placement strategies for modern compute platforms that combine CPU cores with multiple GPU, FPGA, and PIM accelerators. It is efficient, fair, and compatible with a variety of common workload descriptions, output metrics, and visualization tools. We use HetSim to reproduce results from Alebrahim and Ahmad, and examine how accelerator characteristics affect the performance of various scheduling strategies. Our results indicate that ignoring accelerator characteristics during simulation is often detrimental, and that the ideal scheduling algorithm for a given workload may depend on available accelerators and their characteristics. HetSim is available as open-source software.

CCS CONCEPTS

• **General and reference** → **Evaluation**; **Experimentation**; • **Software and its engineering** → *Scheduling*; *Memory management*; • **Human-centered computing** → Visual analytics; • **Computing methodologies** → Discrete-event simulation.

KEYWORDS

Simulator, Heterogeneous Hardware, Scheduling

ACM Reference Format:
Marcel Lütke Dreimann, Birte Friesel, and Olaf Spinczyk. 2024. HetSim: A Simulator for Task-based Scheduling on Heterogeneous Hardware. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3629527.3652275

ICPE '24 Companion, May 7–11, 2024, London, United Kingdom

ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3652275

1 INTRODUCTION

Server hardware has become more and more heterogeneous over recent years. In addition to many-core processors, it can now also include multiple GPUs or FPGAs. Moreover, new processing-in-memory technologies such as UPMEM PIM are emerging and becoming available for end users. Each of these accelerators has its own characteristics and limitations: it can achieve outstanding performance on workloads that it was designed for while being of limited use in other cases.

One common limitation is the need to allocate a CPU core for setup purposes. Consider the trace of PolyBench's *2mm* OpenCL benchmark [11] shown in Fig. 1. Even though the benchmark target is a GPU, OpenCL first spends 200 ms with CPU-only setup functions. The workload itself (*mm2_kernel2*) makes up less than 25 % of execution time [1] and is the only component that actually uses the GPU.

Hence, setup cost is far from negligible when designing or evaluating scheduling algorithms. This also applies to other accelerators. For instance, dynamic offloading with FPGAs relies on costly dynamic reconfiguration, and executing tasks on UPMEM PIM is also far from instantaneous [9].

Similarly, accelerators can use dedicated memory or share it with CPU cores. The former may require data transfers between main memory and accelerator memory. This heterogeneity and diversity of execution components poses new challenges for scheduling and placement decisions. Depending on the software layer where they are implemented, these can affect developers of operating systems, database management systems, language runtimes, or even applications.

Currently, integrating accelerators into applications is usually up to application developers. Novel system software like MxTasking [16] and userspace frameworks like StarPU [4] aim to help them with this task. These projects rely on *tasks* as control flow abstractions and offer a unified framework for programming different kinds of accelerators. They also deal with scheduling decisions that come up if tasks can be executed on several different accelerators. The increasing complexity and heterogeneity of accelerators leads to increasing complexity in dealing with these decisions.

The contributions of this paper are three-fold.

- We provide a discussion of challenges that designers and performance evaluation methods of scheduling strategies for heterogeneous hardware face (Section 3).
- We present *HetSim*, a simulator for task execution on heterogeneous hardware (Section 4). It is capable of evaluating

[1] While *clBuildProgram* latency can be reduced by using pre-compiled GPU binaries in this particular example, the discrepancy between setup cost (CPU) and execution time (GPU) remains.

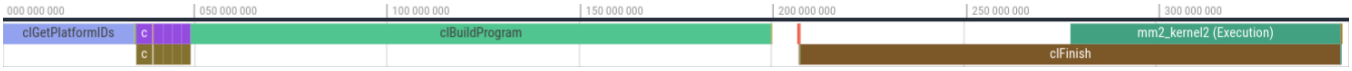

Figure 1: Execution trace of PolyBench's *2mm* OpenCL benchmark [11] on an integrated Intel UHD Graphics Xe GPU. The benchmark uses 16 execution units; the trace has been extracted by the OpenCL Intercept Layer [3].

scheduling algorithms on user-defined hardware platforms without the need for real-world measurements. In addition to a set of built-in workloads, it also accepts workload definitions provided by users or by existing DAG workload generators. An optional event log containing task phases and memory transfers allows for user-friendly visualization with existing tools such as *Perfetto*. With HetSim, we are able to reproduce results from Alebrahim and Ahmad.

- We use HetSim to examine how accelerator-specific overhead affects the performance of different scheduling strategies (Section 5).

The next section examines related work and outlines the gaps that HetSim aims to fill. We then cover our three contributions (see above) and conclude in Section 6.

2 RELATED WORK

The literature offers a variety of scheduling simulators for a diverse set of use cases.

Suranauwarat presents a simulator that exclusively deals with single-CPU scheduling algorithms [20]. It is meant to be used for educational purposes and comes with graphical animations, but does not support heterogeneous hardware.

Realtss focuses on evaluating real-time CPU scheduling policies without having to implement them in a real-time operating system [8]. It has education and research in mind.

ScSF is intended for simulation-based high-performance computing (HPC) scheduling research [18]. It offers tools for workload modeling and generation, system simulation, comparative workload analysis, and experiment orchestration.

SimGrid simulates distributed applications in grid environments [7]. It focuses on communication rather than computation, with latency- and bandwidth-bound links between nodes and bandwidth sharing in case of simultaneous transfers. While it can handle heterogeneous hardware, its *host resource* does not support accelerator-specific behavior. Moreover, it measures performance based on floating-point operations per second (FLOPS). We will explain the drawback of this metric in Section 3.5.

Due to SimGrids's detailed communication simulation and ease of use, several simulators build on top of its API. For instance, *Alea* focuses on event-based scheduling of heterogeneous jobs on heterogeneous resources with dynamic runtime changes [14]. It gathers information about resource status and simulation results, which can be visualized later.

Even though many simulators can gather statistics and offer visualizations, none that we are aware of support *heterogeneous* hardware with *accelerator-specific* behavior. They do not consider setup or reconfiguration costs, and only provide limited statistics and simulator output. Moreover, many simulators have been tailor-made for specific classes of scheduling strategies or hardware components, limiting their re-usability. Our contribution, *HetSim*, fills these gaps by dealing with accelerator-specific attributes in arbitrary schedules and hardware components. Combined with its support for third-party workload generation and visualization tools, this makes it more flexible than existing simulators.

3 DESIGN AND EVALUATION CHALLENGES

As mentioned in the introduction, modern frameworks for the management of heterogeneous computing resources use a control flow abstraction called *tasks*: self-contained units of work that cannot be preempted. The advantage of tasks over traditional abstractions such as POSIX *threads* or GPU/PIM *kernels* is their simplicity, which allows for having a unified control flow model for all supported accelerators.

While this addresses one common issue when dealing with heterogeneous accelerators, task-based scheduling algorithms still face a variety of challenges. We will now discuss the five most prominent ones that we have identified: memory heterogeneity, accelerator heterogeneity, task setup, driver frameworks, and the hardware model.

3.1 Memory Heterogeneity

While most accelerators come with dedicated memory, they can also share memory and even the last-level cache with the CPU – integrated GPUs such as Intel's UHD 630 are a prominent example for this. Additionally, only some accelerators with dedicated memory support direct memory access (DMA). So, a task will incur different amounts of data transfer overhead depending on where it is scheduled. At the same time, in both cases, subsequent tasks scheduled on the same accelerator may benefit from already-present data or warm caches. So, the performance of an individual task depends not just on its own schedule, but also on the schedule of preceding tasks that access shared data. Grouping those onto accelerators with a shared cache can improve performance or save energy [10].

3.2 Accelerator Heterogeneity

Accelerators are not one-size-fits-all devices: GPUs excel at parallel tasks that access shared memory, whereas UPMEM PIM works best with embarrassingly parallel workloads without synchronization or shared memory [17]. Executing a task on an unsuitable accelerator may result in worse performance than just running it on a CPU core [9]. Scheduling strategies have to take this into account, and simulators must be aware of it as well in order to provide useful results. This is a balance act between general-purpose algorithms and simulators that assume anything is supported anywhere, and the more complex task of encoding and utilizing knowledge about the specific properties of each accelerator.

3.3 Task Setup

Many accelerators rely on CPU support just like conventional peripheral devices. A CPU core has to allocate (part of) the accelerator, transfer the task's program code and possibly data, start the task, and handle communication (e.g. waiting for the task to finish and retrieving results). Both scheduling strategies and simulators must take this reliance on CPU cores into account, especially for short tasks where the setup time may exceed actual task execution (cf. Fig. 1). This is another balance act: depending on a task's attributes, using a less suitable accelerator or plain CPU execution may still be faster than the incurred CPU-bound setup cost.

3.4 Driver Frameworks

All accelerators that we are aware of rely on an accelerator-specific driver framework in order to execute tasks. Vendors recommend their own software for optimal performance, e.g. CUDA[2], OneAPI[3], or the UPMEM SDK[4]. However, when updates are applied to this software, the accelerator characteristic may change. Driver optimizations can reduce execution time on the accelerator, or affect the setup time of a task. If scheduling algorithms are to be evaluated on real hardware within a scheduling framework, the scheduling framework must implement the interfaces to all possible drivers. While OpenCL comes close by providing a standard (including a programming language) that encompasses a variety of accelerator types, some features and novel technologies such as UPMEM PIM are missing.

3.5 Hardware Model

Scheduling algorithms and simulators rely on a hardware model to determine the suitability of a given accelerator for a given task. While metrics such as clock frequency or FLOPS may work well in homogeneous settings (e.g. scheduling tasks on a CPU with slow efficiency and fast performance cores), they are insufficient for our purpose. Different accelerators may use different architectures, and thus respond differently to heavy use of branch instructions, vector operations, synchronization, shared memory accesses, and similar. For instance, GPUs tend to work well with floating point math but suffer from branching-induced performance penalties, whereas UPMEM PIM excels at integer vector operations [12]. So, actual accelerator performance is a function of the type of task it executes, and not just its instruction count or a related numeric metric. FLOPS and frequency, as used by e.g. SimGrid, capture none of these nuances.

When dealing with dedicated memory, data transfer overhead must be considered as well. The interconnect used for data transfer typically has a well-defined latency and throughput. However, if multiple data transfers are executed on a single interconnect at the same time, the bandwidth is shared.

4 HETSIM OVERVIEW

We now present our main contribution: *HetSim*, an event-based, discrete scheduling simulator that addresses the aforementioned challenges. It follows a modular design to allow for easy adjustments and extensions. Fig. 2 shows its main components and how they interact with each other.

The `Accelerator` base class implements common functionality and checks, such as per-task memory allocations or determining whether a given task actually has an implementation for this accelerator. Its derived classes implement accelerator-specific behavior. Each `Accelerator` instance references a single `MemoryPool` that represents the accelerator's memory and implements data object allocation and movement. Multiple accelerators can share the same memory pool to represent shared memory environments.

The `Environment` class stores a list of all accelerators (populated by `ModelLoader`) and workload tasks (generated by `TaskSetGenerator`). Each `Task` has a list of dependencies and accessed data objects; a single data object can be referenced by multiple tasks. Lastly, the simulator provides a `Profiler` that keeps track of all task phases and memory transfers. The resulting statistics can be printed on the console or saved in Google's trace format[5].

The following subsections describe how HetSim supports the analysis of scheduling and placement strategies for heterogeneous systems. We will cover workload generation, the simulation process itself, and simulation output.

4.1 Workload Generation

When evaluating scheduling algorithms, it is important to ensure that workloads resemble a wide range of real-world applications and contain randomized components to identify systematic errors in scheduling strategies. At the same time, it is desirable to obtain deterministic simulation results for debugging and reproduction purposes.

HetSim achieves both by providing a set of random number generators that can alter workload attributes, and storing the random seeds and other workload parameters of each simulation run in a `Config` object (cf. Fig. 2). Randomizable attributes include the amount and size of data objects accessed by a task, its expected runtime, and its set of supported accelerators. Each simulation run can be reproduced by loading the associated hardware model description and configuration into a subsequent HetSim invocation.

On top of this, HetSim supports four different types of task sets. It can also simulate mixed workloads that combine multiple task set types.

Random task sets make use of all possible characteristics an application can have, including data (de)allocation and dependencies between tasks. The latter can be disabled if the increased scheduling complexity caused by task dependencies is undesirable.

Database Queries often serve as motivation for heterogeneous hardware, as database operators can benefit from significant performance boosts by utilizing GPU, FPGA or PIM accelerators [5, 15, 19]. The task set is composed of query and aggregation tasks. Each query task accesses the same large data object (the database) and allocates a smaller data object for its result, and the aggregation tasks combine those intermediate results into a single data object.

Directed Acyclic Graphs (DAGs) are often used for evaluating scheduling strategies such as *Heterogeneous Earliest Finish Time* (HEFT) [2, 13]. HetSim supports the DAG task set format of two

[2]https://developer.nvidia.com/cuda-toolkit
[3]https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html
[4]https://sdk.upmem.com/
[5]https://docs.google.com/document/d/1CvAClvFfyA5R-PhYUmn5OOQtYMH4h6I0nSsKchNAySU

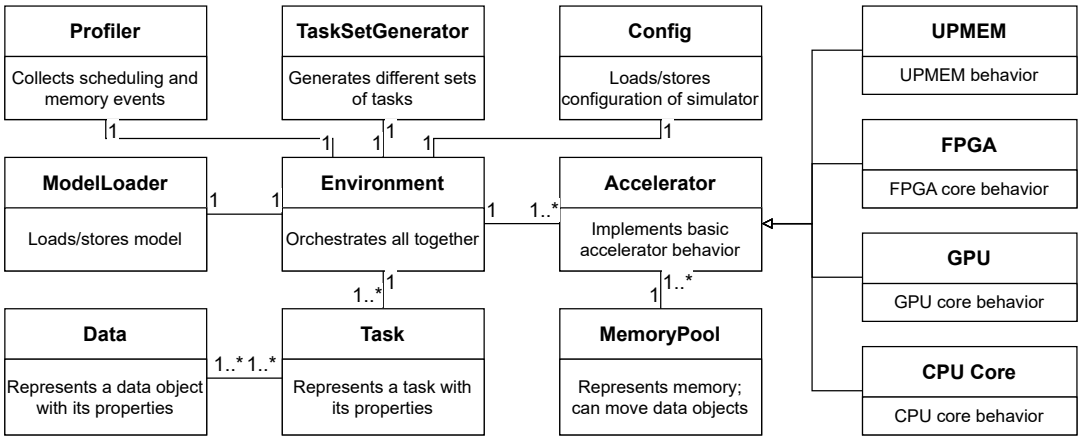

Figure 2: Overview over the most important simulator components.

existing tools: DAGGEN and Pegasus. DAGGEN generates synthetic, random task graphs that are intended for CPU scheduling [21]. Pegasus workloads, on the other hand, are based on real scientific workloads like RNA analysis or the characterization of earthquake hazards [6]. In both cases, each graph node represents a task, and each edge encodes a dependency between two tasks. DAGGEN associates tasks with random computation costs, whereas Pegasus workloads do not provide any runtime annotation. Hence, HetSim generates new computation costs for accelerators according to the provided simulator configuration, and annotates graph edges with communication costs.

User-defined task sets allow for evaluating scheduling strategies that are optimized for specific workloads, and for analyzing the performance and acceleration potential of applications rather than scheduling algorithms. Users can simulate applications that do not yet make use of heterogeneous hardware with a fixed scheduling strategy, and see how enabling the execution of tasks on specific accelerators changes application runtime and resource utilization.

4.2 Simulation

HetSim is implemented as a C++ library. This way, developers can easily port existing scheduling algorithms to the simulator, or implement new algorithms that can later be used in real systems. Additionally, it allows for the use of debugging tools to analyze scheduling algorithms in detail. HetSim supports both online and offline scheduling algorithms. It is available as open source software and extensively documented, thus enabling developers to extend its set of accelerators with custom implementations.

In addition to scheduling algorithm (C++) and task set (Sec. 4.1), users must specify the simulated hardware platform. Such a platform description consists of four elements: memory pools, accelerator architectures, the accelerators themselves, and links between memory pools. HetSim uses an XML format for platform descriptions in order to remain flexible when it comes to supporting future heterogeneous hardware platforms. Listing 1 shows an example; we will now describe its four components in detail.

A **MemoryPool** represents main memory (DRAM), a specific accelerator's dedicated memory, or similar. The memory model is not limited to volatile RAM: it also supports persistent memory such as Optane DCPMMs and SSDs. Each memory pool must have a unique ID and a maximum size.

```
<MemoryPool id="0" size="16384" />
<MemoryPool id="1" size="8192" />
<Architecture id="0" perf_min="1.0" perf_max="1.5" />
<Architecture id="1" perf_min="1.8" perf_max="2.0" />
<Accelerator id="0" archid="0" memorypool="0" type="CPUCore" setupcost="0"
        dma="1" modelerror="0.0" poweridle="10.0" powerload="25.0" />
<Accelerator id="1" archid="0" memorypool="0" type="CPUCore" setupcost="0"
        dma="1" modelerror="0.0" poweridle="10.0" powerload="25.0" />
<Accelerator id="2" archid="1" memorypool="1" type="GPU" setupcost="250" dma=
        "0" modelerror="0.0" poweridle="20.0" powerload="100.0" />
<Link src="0" dst="1" speed="100" />
<Link src="1" dst="0" speed="100" />
```

Listing 1: Excerpt of a hardware platform description.

An **Architecture** encodes performance attributes of a specific accelerator architecture, e.g. a specific GPU model. It has a unique ID as well as minimum and maximum performance factors. HetSim uses these to generate randomized computation costs based on the interval defined by the abstract performance factors. Overlapping performance ranges of architectures allow for non-linear computation costs. For example, consider two architecture with performance ranges $[1.0, 2.0]$ and $[1.5, 2.5]$. Tasks will most likely run faster on the second architecture due to its larger performance factors. However, because of the overlap $[1.5, 2.0]$, tasks can also be equally fast or the first architecture can be faster than the second. This way, HetSim can break out of simple FLOPS-based performance models like the one used in SimGrid.

An **Accelerator** is a concrete compute unit instance, e.g. a CPU core or an FPGA. It references an accelerator architecture and a memory pool by ID, and also has its own ID. The `type` indicates whether it is a CPU, GPU, FPGA, UPMEM, or other kind of accelerator. Additional parameters indicate the absolute setup cost for a task (recall Fig. 1), DMA support, model error, and power usage in idle as well as under load.

The idea behind model error is that the expected execution times of individual tasks may be inaccurate. To examine how susceptible a scheduling algorithm is to such inaccuracies, users can set

`modelerror` to a value within $[0.0, 1.0)$ that controls the dispersion of actual task execution time around the expected execution time. For instance, with `modelerror="0.1"`, actual execution time can be up to 10 % lower or higher than expected execution time.

The `poweridle` and `powerload` values are used by HetSim to estimate the power consumption of each individual accelerator and of the entire system. In order to define virtual accelerators, the corresponding accelerator can be duplicated with the same `memorypool` ID but a new accelerator ID.

Finally, each **Link** encodes a link between two memory pools and its data transfer speed. Duplex communication is modeled by two Link elements with individual bandwidths.

4.3 Statistics and Visualization

Adequate evaluation metrics and statistics as well as intuitive visualization are crucial for understanding and analyzing scheduling algorithms and application performance. HetSim supports a variety of those.

Metrics. Given a set of accelerators A and a schedule *plan*, HetSim provides its simulated total execution time (*makespan*) as well as the *SLR*, *Speedup* and *Efficiency* metrics from the literature [1].

SLR (Schedule Length Ratio) is often used to compare schedules in a way that is independent of DAG topology. It divides the makespan by the sum of task execution times on the critical path. Hence, a better strategy has a lower SLR.

$$SLR(plan) = \frac{makespan(plan)}{\sum_{ti \in CP} \min_{a \in A} w_{ti,a}}$$

Speedup describes how much the schedule benefits from using multiple accelerators. It divides the fastest sequential execution time that can be achieved when scheduling all tasks on a single accelerator by the makespan.

$$Speedup(plan) = \frac{\min_{a \in A} \sum_{ti \in T} w_{ti,a}}{makespan(plan)}$$

Efficiency, in turn, describes how well the scheduling algorithm utilizes the accelerators. It is defined as the ratio of speedup over the number of available accelerators $|A|$. An efficiency of 1.0 indicates that the total execution time of a task set is evenly split across all available accelerators.

$$Efficiency(plan) = \frac{Speedup(plan)}{|A|}$$

Task Phases and Visualization. HetSim records all memory transfers and, for each task, which of the following phases it is currently in:

- Task setup
- Task blocked (waiting for dependency)
- Task blocked (waiting for setup)
- Setup blocked (accelerator busy)
- Data transfer
- Task execution

For each phase, it records start time, duration, and affected accelerator. For memory transfers, it also records the utilization of the respective memory pool.

This allows HetSim to determine the accumulated time that each accelerator has spent in the different task phases as well as its total number of tasks and load. The recorded phases can be visualized with Google's *Perfetto* as shown in Fig. 3. Perfetto generates an interactive plot similar to a Gantt diagram and provides utilities for detailed analysis.

HetSim also determines the minimum, maximum, and average wall-clock time that the machine running the simulation spent making scheduling decisions. Simulation overhead is excluded from this *decision time*. Thus, users can compare the overhead of different scheduling algorithms.

5 EXAMPLE AND EVALUATION

We will now analyze four scheduling algorithms (three online, one offline) to demonstrate the scientific usefulness, performance, and correctness of HetSim. Those are heterogeneous Round Robin (hRR), GreedyET, GreedyDS, and HEFT. Source code, data, and analysis scripts are available at https://ess.cs.uos.de/git/artifacts/wosp-c-2024-hetsim-artifacts.

hRR is a naïve adaptation of the Round Robin (RR) CPU scheduling algorithm for heterogeneous hardware. It does not consider performance attributes, and simply iterates over all accelerators until it has found one that is capable of executing the task. GreedyET picks the accelerator with the shortest expected execution time, and GreedyDS additionally takes data transfers and setup phases into account. HEFT is an offline scheduling algorithm from the literature [2, 13]. All algorithms use CPU core 0 to run setup code for task execution on a GPU or FPGA, if needed. An important distinction is the ability to balance load across CPU cores and accelerators. In contrast to hRR and HEFT, GreedyET and GreedyDS are not able to do load balancing and might over-utilize preferred accelerators.

5.1 Performance of scheduling strategies

Our evaluation uses the Sipht[6] Pegasus task set. It consists of 33 tasks with dependencies from the genome analysis / RNA translation domain. The hardware platform has eight CPU cores in two different NUMA regions of 64 GiB each, an integrated GPU sharing the memory with node 0, and a dedicated GPU and FPGA with 16 GiB of memory each. The link between the NUMA regions has a bandwidth of 100 MB/s and all other links can transfer data with 25 MB/s.

We examine four configurations: a complex model with accelerator-specific setup times, and a simple model without those. Both come in two flavors: slow (little speedup provided by accelerators) and fast (up to two times higher speedup). Table 1 lists the performance intervals and setup costs, and Table 2 shows a HetSim configuration excerpt for the complex models. In the simple cases, `min_fpga_reconf_time` and `max_fpga_reconf_time` are set to 0. We use the SLR metric from the literature to compare the strategies. Note that this metric does not take setup costs into account when calculating the critical path. An adjusted SLR definition for heterogeneous hardware might be useful in the future.

On the simple model without setup costs, HEFT performs best (see Fig. 4). As an offline algorithm, it has prior knowledge of all tasks, so this is to be expected. Moreover, HEFT's internal model is close to our simple model, with the only exception being that HEFT allows concurrency between memory transfers and task execution.

[6] https://pegasus.isi.edu/workflow_gallery/gallery/sipht/index.php

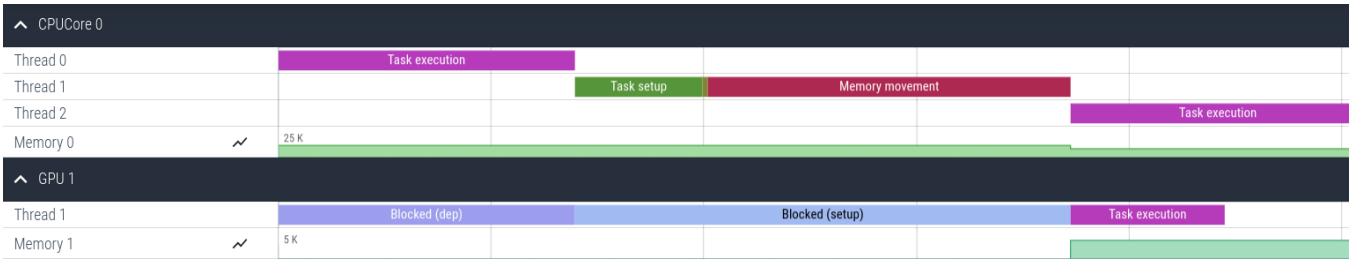

Figure 3: Small example simulator trace visualized by Perfetto.

Accelerator	slow	fast	setup time
CPU Core	[1.0, 1.0]	[1.0, 1.0]	0
iGPU	[0.8, 2.2]	[0.8, 4.4]	100
dGPU	[1.1, 3.6]	[1.1, 7.2]	120
FPGA	[1.2, 4.0]	[1.2, 8.0]	90

Table 1: Performance intervals [perf_min, perf_max] and setup times for "slow" and "fast" configurations.

Setting		Setting	
max_fpga_reconf_time	200	strict_exceptions	1
min_fpga_reconf_time	100	min_data_size	4
min_task_runtime	10	max_data_size	300
max_task_runtime	500	max_task_delay	0
partial_task_families	0	seed_uniform	3
min_data_objects	1	seed_task_runtime	1
max_data_objects	5	seed_data_size	2

Table 2: A simulator configuration that allows for FPGA reconfiguration. The seeds are different for each iteration.

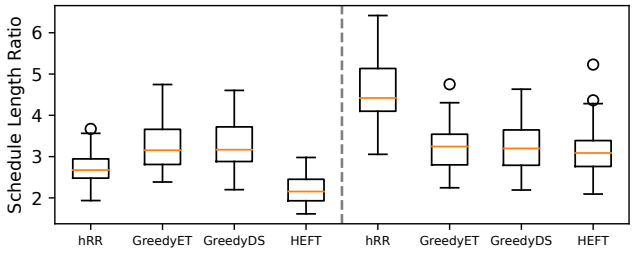

Figure 4: SLR for scheduling strategies without setup phases, using slow (left) and fast (right) accelerators.

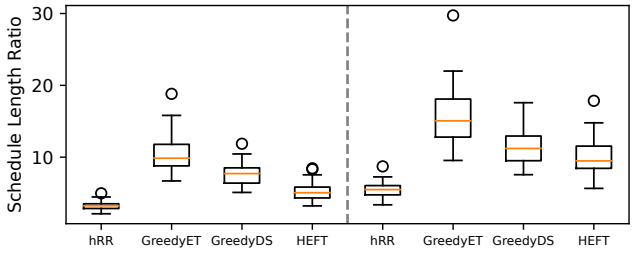

Figure 5: SLR for scheduling strategies with setup phases, using slow (left) and fast (right) accelerators.

In the slow flavor (left), hRR comes in second, and the greedy strategies perform worst. HetSim output and Perfetto diagrams reveal that hRR benefits from load balancing, whereas the greedy strategies execute all tasks on dGPU and FPGA and leave the (slower) CPU and iGPU idle. In the fast flavor (right), on the other hand, the greedy strategies outperform hRR. Here, hRR leaves accelerators idle, as its round-robin algorithm treats eight (slow) CPU cores and three (fast) accelerators as eleven equally fast compute nodes.

Fig. 5 shows the impact of including setup costs in the models. Now, hRR is best in both flavors, outperforming even the offline HEFT approach. hRR prefers CPU cores (see above) and thus incurs little CPU/FPGA setup costs, whereas HEFT chooses GPU or FPGA execution without taking setup cost into account, degrading its performance. Additionally, for HEFT and both greedy algorithms, CPU core 0 is overloaded with setup code for GPU and FPGA tasks, thus delaying their start. While GreedyET favors the FPGA due to having the lowest execution time for most tasks, it still comes up with a suboptimal schedule due to its high reconfiguration time. GreedyDS performs better, but is still behind hRR and HEFT due to its lack of load balancing and over-utilization of CPU core 0. Overall, we see that mapping tasks to devices is far from trivial. Any performance gains provided by an accelerator can easily be nullified by its setup costs.

Another interesting insight is that the overall SLR of every algorithm has increased in the fast flavor. This does not mean that the makespan has increased, but rather that the length of the critical path across the DAG was reduced. All of these findings could be highly useful during research and development of scheduling and placement strategies.

5.2 Costs of scheduling decisions

Of course, more complex scheduling decisions incur higher scheduling overhead. Fig. 6 shows the time per scheduling decision on the Sipht task set, measured on the simulating machine. hRR, with its lack of model usage, is fastest. GreedyET is five times slower, and GreedyDS's fine-grained approach results in an up to 20-fold increase in overhead.

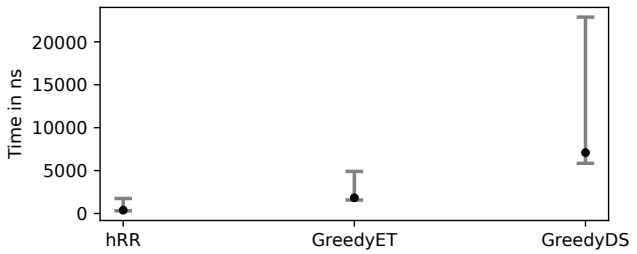

Figure 6: Benchmark of scheduling algorithms with minimum, maximum and average decision time measured on the simulating machine (i7-11850H).

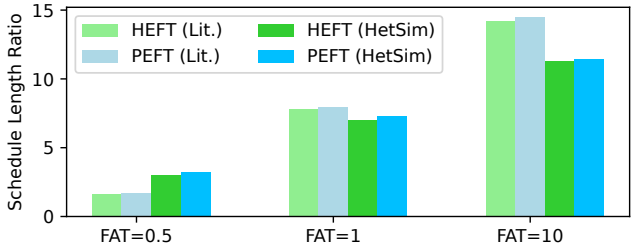

Figure 7: SLR results for HEFT and PEFT by Alebrahim and Ahmad (annotated as "Lit.") and Hetsim.

5.3 Simulation Correctness

To evaluate the correctness of HetSim, we used it to reproduce HEFT and PEFT SLRs reported by Alebrahim and Ahmad. Those build upon a hardware model with two processing units with different performance values for each task and without setup cost. The input task sets were generated with DAGGEN with variable levels of parallelism ("FAT" parameter). The HetSim reproduction uses a hardware model consisting of two CPU cores with different architectures and identical DAGGEN parameters. As Fig. 7 shows, while absolute values differ slightly due to a different simulation model, the relation between HEFT and PEFT remains the same.

When comparing HetSim's plans with the example given by Alebrahim and Ahmad, we see only a single difference. HetSim's simulation model does not allow memory transfer during task execution, while the HEFT and PEFT plan from the paper does.

5.4 Simulator performance

Lastly, we evaluated the performance of HetSim itself by using the database query benchmark with the hRR algorithm. As Fig. 8 shows, simulation time scales linearly with the number of tasks, with a maximum of just 2.2 *s* for 500,000 tasks. Despite its single-threaded implementation, HetSim is fast enough to simulate large task sets in a reasonable amount of time. In practice, the simulation time was not a limitation for us, as several simulations can run on different cores at the same time.

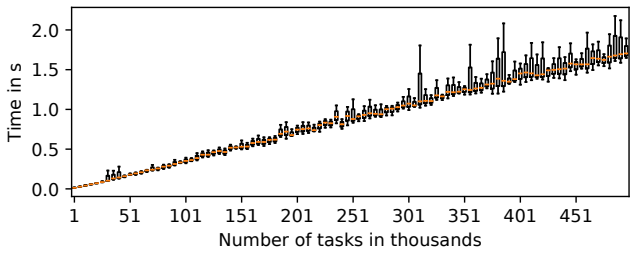

Figure 8: Simulation time for large task sets on an i7-11850H.

6 CONCLUSION AND FUTURE WORK

We have presented HetSim, a scheduling simulator for heterogeneous hardware that allows developers to study and analyze scheduling algorithms for given workloads and hardware configurations. We showed that it can help identify weaknesses of scheduling algorithms. The simulator records statistics and allows visualization of schedules by third-party tools. Furthermore, HetSim supports benchmarking scheduling decisions to help developers evaluate the performance overhead of complex algorithms. The simulator handles large simulations with hundreds of thousands of tasks in only a few seconds and can reproduce results from the literature. HetSim is freely available at https://ess.cs.uos.de/git/software/hetsim under an open source license. It can serve as a reusable evaluation tool that removes the need for writing algorithm-specific simulators.

The development of HetSim is ongoing. We plan to handle contention and bandwidth limitations if multiple data transfers happen at the same time on a single link in our simulator. HetSim could also use a real hardware model without abstract performance values to estimate absolute execution times on real hardware instead of time units. Furthermore, we plan to extend the possible use cases of HetSim. For example, our simulator could also be used to automatically explore the design space of scheduling strategies for a given hardware model.

ACKNOWLEDGMENTS

The work on this paper has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 361498541, 502565817.

REFERENCES

[1] Wakar Ahmad, Bashir Alam, and Sahil Malik. 2019. Performance analysis of list scheduling algorithms by random synthetic DAGs. In *Proceedings of 2nd International Conference on Advanced Computing and Software Engineering (ICACSE)*. https://doi.org/10.2139/ssrn.3349016

[2] Shaikhah Alebrahim and Imtiaz Ahmad. 2017. Task Scheduling for Heterogeneous Computing Systems. *J. Supercomput.* 73, 6 (jun 2017), 2313–2338. https://doi.org/10.1007/s11227-016-1917-2

[3] Ben Ashbaugh. 2018. Debugging and Analyzing Programs Using the Intercept Layer for OpenCL Applications. In *Proceedings of the International Workshop on OpenCL* (Oxford, United Kingdom) *(IWOCL '18)*. Association for Computing Machinery, New York, NY, USA, Article 14, 2 pages. https://doi.org/10.1145/3204919.3204933

[4] Cédric Augonnet, Olivier Aumage, Nathalie Furmento, Raymond Namyst, and Samuel Thibault. 2012. StarPU-MPI: Task Programming over Clusters of Machines Enhanced with Accelerators. In *Recent Advances in the Message Passing Interface*, Jesper Larsson Träff, Siegfried Benkner, and Jack J. Dongarra (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 298–299.

[5] Alexander Baumstark, Muhammad Attahir Jibril, and Kai-Uwe Sattler. 2023. Processing-in-Memory for Databases: Query Processing and Data Transfer. In

Proceedings of the 19th International Workshop on Data Management on New Hardware (Seattle, WA, USA) *(DaMoN '23)*. Association for Computing Machinery, New York, NY, USA, 107–111. https://doi.org/10.1145/3592980.3595323

[6] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. 2008. Characterization of scientific workflows. In *2008 Third Workshop on Workflows in Support of Large-Scale Science*. 1–10. https://doi.org/10.1109/WORKS.2008.4723958

[7] H. Casanova. 2001. Simgrid: a toolkit for the simulation of application scheduling. In *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*. 430–437. https://doi.org/10.1109/CCGRID.2001.923223

[8] Arnoldo Diaz, Ruben Batista, and Oskardie Castro. 2007. Realtss: a real-time scheduling simulator. In *2007 4th International Conference on Electrical and Electronics Engineering*. 165–168. https://doi.org/10.1109/ICEEE.2007.4344998

[9] Birte Friesel, Marcel Lütke Dreimann, and Olaf Spinczyk. 2023. A Full-System Perspective on UPMEM Performance. In *Proceedings of the 1st Workshop on Disruptive Memory Systems* (Koblenz, Germany) *(DIMES '23)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3609308.3625266

[10] Vıctor Garcıa, Juan Gomez-Luna, Thomas Grass, Alejandro Rico, Eduard Ayguade, and Antonio J. Pena. 2016. Evaluating the effect of last-level cache sharing on integrated GPU-CPU systems with heterogeneous applications. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*. 1–10. https://doi.org/10.1109/IISWC.2016.7581277

[11] Scott Grauer-Gray, Lifan Xu, Robert Searles, Sudhee Ayalasomayajula, and John Cavazos. 2012. Auto-tuning a high-level language targeted to GPU codes. In *2012 innovative parallel computing (InPar)*. Ieee, 1–10.

[12] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. 2022. Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System. *IEEE Access* 10 (2022), 52565–52608. https://doi.org/10.1109/ACCESS.2022.3174101

[13] Julien Herrmann, Loris Marchal, and Yves Robert. 2014. Memory-Aware List Scheduling for Hybrid Platforms. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. 689–698. https://doi.org/10.1109/IPDPSW.2014.80

[14] Dalibor Klusâcek and Hana Rudovâ. 2010. Alea 2: job scheduling simulator. ICST. https://doi.org/10.4108/ICST.SIMUTOOLS2010.8722

[15] Mehdi Moghaddamfar, Christian Färber, Wolfgang Lehner, Norman May, and Akash Kumar. 2021. Resource-Efficient Database Query Processing on FPGAs. In *Proceedings of the 17th International Workshop on Data Management on New Hardware* (Virtual Event, China) *(DAMON '21)*. Association for Computing Machinery, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3465998.3466006

[16] Michael Müller, Thomas Leich, Thilo Pionteck, Gunter Saake, Jens Teubner, and Olaf Spinczyk. 2020. He..ro DB: A Concept for Parallel Data Processing on Heterogeneous Hardware. In *Architecture of Computing Systems – ARCS 2020*, André Brinkmann, Wolfgang Karl, Stefan Lankes, Sven Tomforde, Thilo Pionteck, and Carsten Trinitis (Eds.). Springer International Publishing, Cham, 82–96.

[17] Joel Nider, Craig Mustard, Andrada Zoltan, John Ramsden, Larry Liu, Jacob Grossbard, Mohammad Dashti, Romaric Jodin, Alexandre Ghiti, Jordi Chauzi, and Alexandra Fedorova. 2021. A Case Study of Processing-in-Memory in off-the-Shelf Systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 117–130. https://www.usenix.org/conference/atc21/presentation/nider

[18] Gonzalo P. Rodrigo, Erik Elmroth, Per-Olov Östberg, and Lavanya Ramakrishnan. 2018. ScSF: A Scheduling Simulation Framework. In *Job Scheduling Strategies for Parallel Processing*, Dalibor Klusáček, Walfredo Cirne, and Narayan Desai (Eds.). Springer International Publishing, Cham, 152–173.

[19] Viktor Rosenfeld, Sebastian Breß, and Volker Markl. 2022. Query Processing on Heterogeneous CPU/GPU Systems. *ACM Comput. Surv.* 55, 1, Article 11 (jan 2022), 38 pages. https://doi.org/10.1145/3485126

[20] Sukanya Suranauwarat. 2007. A CPU scheduling algorithm simulator. In *2007 37th Annual Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports*. F2H–19–F2H–24. https://doi.org/10.1109/FIE.2007.4417885

[21] Frédéric Suter and Sascha Hunold. 2013. Daggen: A synthetic task graph generator.