Towards Task-Based Scheduling on Truly Heterogeneous Systems

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

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

Mario Porrmann
mario.porrmann@uos.de
Universität Osnabrück
Osnabrück, Germany

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

Abstract

In the past decade, technologies such as non-volatile memory (NVM), high-bandwidth memory (HBM), compute express link (CXL), and near-memory computing (NMC) have disrupted established task and data placement heuristics. Conventional abstractions such as the memory hierarchy are no longer valid: for instance, using HBM as cache for DDR RAM can slow down workloads [8, 12]. Task-based scheduling algorithms aim to resolve this by decomposing workloads into non-preemptible work units (*tasks*) and minimizing workload execution time (*makespan*) by appropriate placement of tasks and data objects. However, their applicability to systems with novel, disruptive memory technologies such as HBM, CXL, or NMC depends on the underlying task and platform model. By examining the models used within *heterogeneous* task-based scheduling algorithms published in the past two decades, we find that they have remained largely unchanged over the past 25 years, and are inappropriate for today's level of heterogeneity in compute and memory components. Based upon this, and related work that offers at least partial improvements, we identify gaps in existing models, showcase examples to underline their relevance, and present ideas to remedy some of those. Our goal is not to give answers for all open questions, but rather to provide pointers towards appropriate abstractions for future, operating system-centric, task-based scheduling research.

Keywords

Scheduling Algorithms, Heterogeneous Systems, Platform Models, Task Models

DOI: https://dx.doi.org/10.1145.nnnnn. GI/ITG FG Operating Systems, February 2026, Düsseldorf, DE.

1 Introduction

Task-based scheduling is the process of assigning task-based workloads to compute resources in setups such as chiplets, multi-core servers, data centres, or federated clouds [1, 6, 23]. Each workload is defined as a directed acyclic graph (DAG), wherein each node expresses a single (non-preemptible) task, and each edge expresses a data dependency between a pair of tasks. Typically, the goal is to minimize the *makespan*: the latency from the start of the DAG's first task to the successful execution of its last one.

Makespan-optimal scheduling of task graphs is an NP-complete problem already in the homogeneous case, where task latency is independent of task placement [2, 24]. Hence, for the past decades, researchers have looked into heuristics instead, and used empirical evaluations on mixtures of synthetic and real-world, scientific workloads to assess their quality. While these heuristics have steadily improved over the past 25 years [23, 25], and researchers have also examined additional optimization goals such as energy and cost [17, 19, 30], the task and platform models they build upon have remained largely unchanged.

This paper examines whether those models still align with the realities of today's highly heterogeneous hardware, and provides suggestions for extended task and platform models so that scheduling research remains applicable to real-world data processing environments. In essence, it is a combination of mini-survey and vision paper, partially augmented with proof-of-concept implementations and evaluations.

In the next section, we present the task and platform model that the majority of task-based scheduling research has built upon for the past 25 years. We then review publications on task-based scheduling on heterogeneous systems, with a focus on model use cases and extensions, in Section 3. Following up, Section 4 presents our own assessment of the accuracy and practical viability of their level of abstraction, and gives suggestions for evolving task and platform models

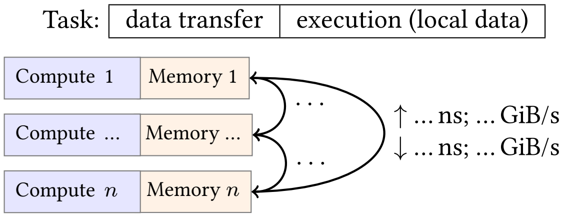

Figure 1: Platform model and task execution phases in HEFT.

to deal with the constraints and peculiarities imposed by real-world hardware components. We underline some of our proposed changes with a case study on a SELECT database kernel in Section 5, and conclude in Section 6.

2 Task and Platform Model

The *Heterogeneous Earliest Finish Time* (*HEFT*) algorithm has served as a baseline for task-based scheduling on heterogeneous systems for the past 25 years [23], and, as of 2024, is still considered as state of the art [3]. As almost all publications that we reviewed build upon it, we now present its task and platform model in order to have a common foundation for discussing limitations and extensions.

On the hardware side, HEFT focuses on systems that consist of multiple, heterogeneous compute units. These compute units can be anything from CPU cores in a single machine to servers in a networked compute cluster. Crucially, they are heterogeneous: task latency is a function of task placement, and algorithms must not just find a suitable order of tasks, but also place them on appropriate compute units while taking data transfer overhead into account.

Each compute unit is associated with a single, local memory with unlimited capacity; tasks running on it cannot access memory on other units. If a pair of dependent tasks $v_1 \longrightarrow v_2$ is scheduled on different units c_1, c_2, the system runtime must transfer output data of v_1 from c_1 to c_2 so that it can be used as input data for v_2.

HEFT assumes that these data transfers occur concurrently with task execution and without contention; data transfer duration is a function of data size as well as data transfer latency and throughput. This results in the following *platform* and *task models*; platform model and task concept are additionally visualized in Fig. 1.

Definition 2.1 (HEFT Platform Model [23]). A platform is a tuple $\mathcal{M} = (C, L, B)$, consisting of compute units C, communication startup latencies $L : C \longrightarrow \mathbb{R}_{\geq 0}$, and communication bandwidths $B : C^2 \longrightarrow \mathbb{R}_{\geq 0}$.

Definition 2.2 (HEFT Task Model [23]). A workload is a directed acyclic graph (DAG) $\mathcal{T} = (V, E, W, D)$. It consists of tasks (nodes) V, data dependencies (edges) $E \subset V^2$, task latencies (weights) $W : V \times C \longrightarrow \mathbb{R}_{\geq 0}$, and annotations $D : E \longrightarrow \mathbb{N}_{\geq 0}$ that identify the amount of exchanged data.

Running task $v \in V$ on node $c \in C$ takes $W(v, c)$ time units (e.g., seconds). For each dependency $(v_1, v_2) \in E$, $D(v_1, v_2)$ describes the amount of data that is transferred from task v_1 to task v_2 in an arbitrary, but fixed, unit (e.g., Bytes or blocks). In practice, W and D are often implemented as two-dimensional matrices.

Communication startup latencies are specific to the node holding the requested data. So, with task v_1 scheduled on compute unit c_1, and task v_2 scheduled on unit $c_2 \neq c_1$, transferring $D(v_1, v_2)$ units of data takes $L(c_1) + \frac{D(v_1,v_2)}{B(c_1,c_2)}$ time units (e.g., seconds). If $c_1 = c_2$, data is already available and thus there is no data transfer overhead.

These models can be applied to many real-world systems, including CPU cores in a multi-core or multi-socket server with non-uniform memory architecture (NUMA). Although those typically share a single memory region with *multiple* compute units, this can be expressed by discarding startup latency ($L(c) = 0$ for all $c \in C$) and setting $B(c_1, c_2) = \infty$ for any pair c_1, c_2 of cores that share the same NUMA node.

Note that HEFT and all other algorithms and models examined in this paper assume that the entire task graph is provided in advance. All tasks, task execution times, and data dependencies (including the amount of data transferred between task pairs) are deterministic and known beforehand. As such, we only consider *offline* algorithms that precompute the entire schedule. We also specifically focus on task-based scheduling for individual servers that are outfitted with multiple different compute and memory technologies, e.g., CPU, GPU, DDR RAM, HBM, and NMC.

3 Related Work

HEFT has been proposed nearly 25 years ago and is still widely used as a baseline for comparison and improvement. As of January 2026, Google Scholar lists more than 4,500 citations for the original publication by Topcuoglu et al. [23]. Hence, for our review of follow-up developments, we combined a keyword-based literature review with a review of pubilcations that cite the original HEFT algorithm, focusing on task-based scheduling for heterogeneous or disaggregated compute and memory systems. We note that, in line with the scope of this paper, this merely serves as a mini-survey to gain a sense of research directions – we did not perform a structured literature review.

Overall, we found four clusters of publications:

(1) algorithm improvements with identical or nearly unchanged task and platform model,
(2) additional optimization goals (e.g., energy usage),
(3) adjustments towards specialized use cases, partially with simplified models, and

(4) approaches that are incompatible with task graphs.

In addition to the application domains listed in the previous section, we observed a focus on cloud computing and multi-cloud federation (cost optimization) as well as edge and fog computing (energy usage optimization).

The first cluster covers improvements such as *Lotaru* [3], *PEFT* (predict earliest finish time) [2], *IPPTS* (improved predict priority task scheduling) [10], *READYS* [14], and *AEFT* (average earliest finish time) [25]. All of these, and many more [1, 16, 18], leave the underlying task and hardware model unchanged. Two decades after HEFT's publication, Lotaru still references it as state of the art [3].

MPQGA uses a slightly more fine-granular model [29]. It splits task latency W into two components: amount of computation per task ($W' : V \to \mathbb{R}_{\geq 0}$, independent of placement) and task-specific computing speed ($W'' : V \times C \to \mathbb{R}_{\geq 0}$). This way, it retains compatibility with HEFT and others: $W(v, c) = \frac{W'(v)}{W''(v,c)}$ for all $v \in V, c \in C$.

The second cluster focuses on energy and cost optimization or constraints, often in addition to low latency.

For instance, Chen et al. aim to minimize schedule makespan on homogeneous compute units with dynamic voltage and frequency scaling (DVFS) support under energy constraints [7]. They first simplify the model so that each task is associated with a worst-case execution time (WCET) $W' : V \to \mathbb{R}_{\geq 0}$ that is independent of task placement, and that data transfer latency is only a function of size and bandwidth: $\frac{D(v_1,v_2)}{B(c_1,c_2)}$, with no communication startup latency. Each compute unit c supports different DVFS levels $s_{q,1}, \ldots, s_{q,k}$ that affect task latency and power usage, and both in turn determine the task's energy usage. Again, task latency remains compatible: $W(v, c) = \frac{W'(v)}{s_{c,i}}$ for task v, compute unit c, and DVFS level i.

Zhang et al. also consider DVFS and DVFS-specific power usage as part of task and platform model, and additionally introduce the transient fault probability of individual compute units. With this, their goal is to minimize both energy usage and probability of failure [30].

Meanwhile, Jayanetti and Buyya aim to optimize the cost of workload execution in a cloud setting. Their contribution to the model is a placement-specific task cost in addition to the already-present placement-specific task latency [19].

Panda and Jana tackle concurrent scheduling of multiple workloads (i.e., multiple DAGs) in a multi-cloud setting, but with a limited model [22]. Here, tasks only have execution dependencies; data transfers and associated latencies are assumed to be negligible and thus left out.

Wang et al. consider task sets with non-deterministic components in an industrial internet of things (IIoT) setting [26]. They define a *conditional task graph* (CTG), wherein certain tasks may or may not be executed. For instance, if an earlier task has not identified any anomalies, its follow-up task(s) for anomaly analysis will not be executed. Task latency remains unchanged (i.e., deterministic), and data transfer bandwidth is simplified to be symmetric.

Altogether, most model changes we found focus on extensions towards additional optimization goals. Crucially, they leave the underlying architectural assumptions (mesh topology; one memory per compute unit) as-is [17, 28].

Moving on to the third cluster, Bathie et al. present one of very few approaches that considers memory constraints [4]. However, it is limited to a single, shared memory, making it ill-suited for practical applications such as NUMA, HBM, GPUs, or NMC.

Benoit et al. present a HEFT extension for servers equipped with DRAM and HBM that is both more and less flexible than the original HEFT models [5]. On the one hand, they explicitly consider a system with two different memories: high capacity, low bandwidth (e.g., DDR RAM) and low capacity, high bandwidth (e.g., HBM), and place each *block* of data individually. This is much finer than HEFT, which always places entire data objects of size $D(v_1, v_2)$. Moreover, they no longer assume that each compute unit has a single local memory: applications can access both memories, data may be freely distributed across them, and the algorithm takes contention from concurrent data accesses into account.

On the other hand, they strongly simplify the compute and memory model: there are just two memory regions (DDR RAM and HBM), and all compute units can access these with the same region-specific latency and bandwidth. Again, NUMA and components with dedicated memory (such as GPUs or NMC) are not supported.

Finally, especially when it comes to HBM-related placement algorithms, many approaches use models that are not compatible with DAG task sets and corresponding platform models. Examples for this fourth cluster include channel arbitration optimization or the placement of individual memory allocations, which do not take tasks into account [8, 9, 20].

Overall, we find that, nearly 25 years later, the underlying model of the HEFT algorithm is still in wide-spread use for offline scheduling of task sets on chiplets, CPUs, GPUs, computer networks, clouds, and more [6, 14, 19, 23].

4 Limitations and Recommendations

While this model is suitable for systems from 25 years ago, hardware has evolved dramatically since then. Nowadays, specialized accelerators such as GPUs, FPGAs or NMC devices require time-intensive *setup* or *reconfiguration* before they can execute a task [11, 21]. At the same time, compute and memory resources are no longer synonymous: a compute unit may have two "local" memories (e.g., DDR RAM and HBM), and memory regions may come without associated compute units (e.g., CXL-attached DRAM). Applications may also benefit from directly accessing data on remote memory rather than waiting for it to be copied to local memory [12].

None of these developments can be expressed with the traditional task and platform model used by HEFT and dozens of related algorithms, where each compute unit is associated with a single memory and tasks always work on local data. In the following subsections, we will discuss these limitations in detail, and present ways of addressing them. We will cover three aspects: task definition, setup / reconfiguration of accelerators, and data transfer.

4.1 Task Definition

At HEFT's inception, heterogeneity typically referred to processors with different performance levels. Tasks could run faster on certain cores (or servers) than on others, but the concept of a task itself was well-defined. Today, heterogeneous systems combine FPGAs, GPUs, or NMC accelerators, where a uniform definition of a task no longer applies. Not only are the processed instructions different, but the Flynn classification and the amount of work represented by a task differ as well. CPU cores typically process one (or few) data points per instruction (SISD), while a GPU-enabled application function can process thousands at the same time (SIMD).

Due to this, the required granularity of tasks (and, thus, the workload's DAG itself) depends on the targeted accelerator. While large workloads are usually distributed across multiple CPU cores and thus correspond to multiple tasks, only one task is required for the GPU. Furthermore, individual tasks may only support a subset of available compute units, for instance due to accelerator-specific restrictions or missing implementations for non-CPU execution.

Wu et al. offer a way to solve parts of this dilemma, where the placement of individual tasks affects the structure of the DAG but can, in turn, only be determined once the entire DAG is known. They propose *hierarchical* DAGs, wherein individual tasks may reference CPU-specific sub-DAGs [27]. Thus, parts of a workload can either be distributed across multiple CPU cores or make use of a single GPU. We propose to also allow sub-DAGs for accelerators. This way, scheduling algorithms know the entire DAG in advance, but can still respect accelerator-specific task structures. Additionally, references to *empty* sub-DAGs can indicate that the corresponding compute units are not supported by a task. This approach brings even more advantages, which we will revisit in the following subsections.

4.2 Setup and Reconfiguration

Non-CPU compute units, such as GPUs or NMC devices, must be initialized before they can be used. This typically involves software framework initialization, resource allocation, and upload of the task's machine code. This process can take dozens to hundreds of milliseconds [13], and may even require more time than the task itself [21]. In addition, initialization must typically be handled by a CPU core, which is unavailable for other tasks during this time.

A simple solution based on the hierarchical DAGs from Section 4.1 is the explicit representation of this phase as a task in the accelerator-specific sub-DAG. Section 5 presents an example for this. This approach not only allows setup costs to be taken into account at all, but also enables the scheduler to place setup tasks on appropriate compute units, thus blocking them for other tasks during the setup phase. Moreover, the latency of a setup task can depend both on the accelerator chosen for the main task and the placement of the setup task, closely resembling real-world behaviour.

4.3 Data Transfer

Another oversimplification in the HEFT model can be found in data transfer: no system has unlimited memory capacity. When tasks are scheduled on an accelerator with limited local memory capacity, they may not have enough space to store all required data objects, causing the resulting schedule to be infeasible in practice. Additionally, HEFT's model assumes a fully-connected mesh between compute units and associated memories. While this is valid when viewing modern accelerators in isolation (e.g., data can be transferred to/from GPU memory both via DDR RAM and HBM), multi-accelerator systems violate this assumption. For instance, there is no direct connection between an FPGA's on-board memory and a GPU's dedicated memory – data must be transferred via main memory (e.g., DDR RAM or HBM).

Moreover, HEFT's assumption that data is moved in the background (without an explicit copy task running on a CPU core) is only valid for compute units with DMA support. This is not generally the case – for instance, NMC as implemented in UPMEM's "PIM" memory relies on a CPU thread pool to handle data transfers [15]. Finally, other types of compute units do not have any dedicated memory and instead share memory with CPU cores (e.g., integrated GPUs).

4.4 Proposal

Naturally, increasing the task and platform models' level of detail will also increase the complexity of associated scheduling algorithms – the ideal level of abstraction that achieves a suitable compromise between algorithmic complexity and model accuracy will have to be determined empirically. However, our own experience indicates that the original HEFT model can be improved at least to some extent without sacrificing scheduling performance [21].

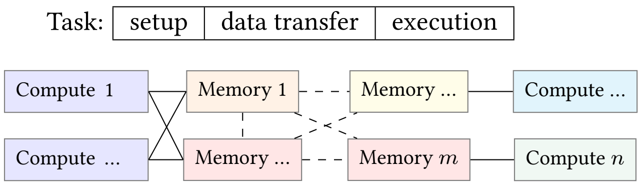

Figure 2: An extended task and platform model example with heterogeneous, partially-connected compute (here: CPU / NMC / GPU) and memory (here: DDR RAM / NMC / HBM).

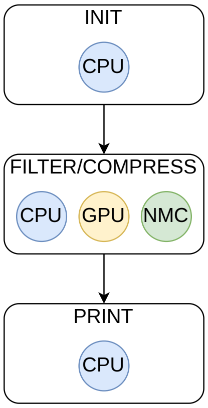

Figure 3: Coarse-grained DAG for SEL, excluding data transfer annotations. The filter/compress task can be executed either on the CPU, a GPU, or an NMC device.

That being said, we suggest to extend HEFT's models as shown in Fig. 2: compute units and memory pools are distinct entities, and each compute unit may have access to multiple memory pools (solid lines). Dashed lines indicate data transfer possibilities between memory pools, using either a CPU-bound data transfer task or DMA. Each memory pool is annotated with a capacity, and all links are annotated with the usual latency and bandwidth figures.

For the task model, it is also advantageous to specify a set of data objects rather than just a single data transfer size [21]. This allows applications such as PrIM TRNS, where many small memory transfers occur and communication startup latency B is the main cause of overall execution latency, to be presented accurately [15].

We deliberately do not give a formal definition of our proposed model extension – as stated above, further empirical research is needed to determine whether this concrete proposal is viable. Instead, we will now examine a case study to show the benefits of some of these extensions.

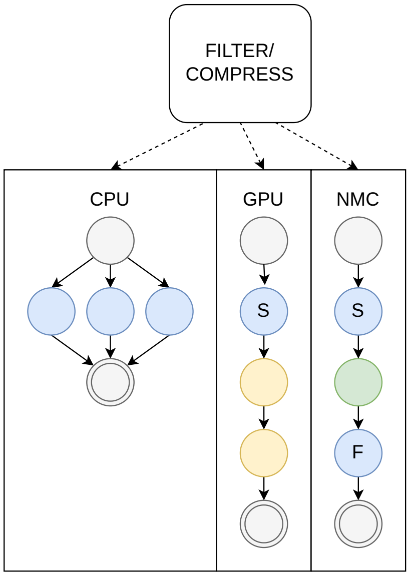

Figure 4: Sub-DAGs for SEL's filter/compress task. Node colour refers to the supported compute unit (CPU / GPU / NMC). "S" refers to setup tasks, and "F" to follow-up tasks.

5 Example: PrIM SEL

We use PrIM SEL to showcase the benefits of an extended task and platform model. It is part of the ”Processing In Memory“ benchmark suite [15], and provides CPU, GPU, and NMC (UPMEM PIM) implementations of a select kernel that returns all data objects matching a certain attribute.

5.1 Task Model

In the conventional model, SEL consists of three sequential tasks: initialization, filter / compression (i.e., the select kernel itself), and output of the results. Fig. 3 shows the corresponding DAG task graph.

However, the CPU, GPU, and NMC implementations of the filter / compression task exhibit very different computation and communication patterns. SEL's CPU implementation uses a simple, fork-join style, multi-threaded workload partitioning scheme. Meanwhile, GPU and NMC implementation require a CPU task to run setup code before actual GPU / NMC execution can take place. The GPU variant then uses two distinct sub-kernels on the GPU, whereas the NMC implementation consists of a single NMC kernel and a CPU task that post-processes the data provided by the NMC kernel.

Fig. 4 shows how these differences can be represented with hierarchical DAGs. The filter/compress task points to a sub-DAG for each supported compute unit, and the sub-DAGs describe the computation and communication patterns in detail. They also encode the fact that the GPU and NMC

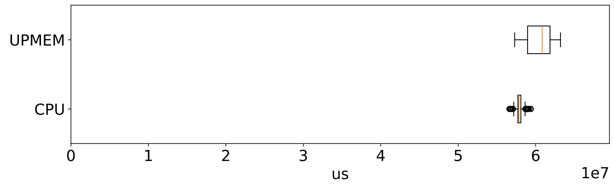

Figure 5: Time to completion for PrIM SEL with 2500 NMC processors (top) and 16 CPU cores (bottom).

variants do not exclusively run on the GPU or NMC device, but require CPU-only setup and (in case of NMC) follow-up tasks. This is crucial for obtaining realistic schedules.

5.2 Hardware Model

Our evaluation platform consists of two Intel Xeon Silver 4215 CPUs, each with 8 cores (16 threads) and 64 GiB of DDR4 memory. Additionally, it is equipped with 20 UPMEM PIM modules, providing 160 GiB of memory with a total of 2,560 NMC compute units. The system has two NUMA nodes (one per socket); NMC compute units can only access their associated partition of UPMEM PIM memory.

We used a tool bundled with our *HetSim* scheduling simulator to generate the hardware model [21], and augmented it with microbenchmarks to determine the latency and bandwidth between NUMA regions and to/from NMC memory [12]. The structure of the model corresponds to our proposal shown in Fig. 2.

5.3 Example Schedule

When using HEFT with the conventional model as shown in Fig. 3, HEFT decides to run the filter/compress task on NMC. From HEFT's perspective, this is the right choice: thanks to the massive parallelism provided by NMC, both modelled and actual task latency for NMC are more than an order of magnitude lower than, e.g., for CPU execution [11, 15].

However, when running the task set on actual hardware, and taking the *total* execution time from start to finish into account, Fig. 5 shows that the CPU variant is actually slightly faster. This is due to the NMC-specific setup and follow-up task, whose combined latency thwarts the speedup gained by NMC's massively parallel execution. HEFT's model does not support this kind of compute unit-specific setup or follow-up latency, and hence HEFT did not consider it. Our proposal, in contrast, can represent these latencies and thus enable better scheduling decisions.

6 Conclusion

We have examined models for task-based scheduling on truly heterogeneous systems, where workloads are expressed as directed acyclic graphs (DAGs) of individual, non-preemptible work units (tasks).

Our review of related work has shown that the majority of publications in this field have built upon the task and platform models that were first proposed nearly 25 years ago as part of the HEFT scheduling algorithm [23]. Considering the complexity and diversity of today's hardware, these models are no longer suitable for task-based scheduling on individual servers. For instance, GPUs, FPGAs or NMC devices require CPU-bound setup tasks before they can be used, and data transfer between tasks is not as simple as the original model implies. Moreover, the DAG's structure itself depends on where individual tasks will be scheduled.

Following up, we have provided suggestions for improving the task and platform models to be in line with the reality of today's hardware, while retaining a level of abstraction that is suitable for task-based scheduling algorithms. These stem partially from related work, such as hierarchical DAGs [27], and partially from our own experience with task-based scheduling on heterogeneous systems [12, 21]. Combined with the SEL example given in the previous section, these show that more fine-grained models, such as the one proposed in Fig. 2 can enable more accurate scheduling decisions.

Naturally, this is only a proposal, and we are far from having examined all of its ramifications for latency prediction accuracy and scheduling algorithm complexity. After all, each abstraction (here: each task and platform model) is a compromise between these two aspects. Still, in our opinion, it is past due for an update to HEFT's decades-old task and platform models so that they can more accurately represent the reality of today's server hardware. In the end, a suitable level of abstraction, and how closely it aligns with our proposal, will have to be determined empirically.

Acknowledgments

This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 502565817.

References

[1] DI George Amalarethinam and A Maria Josphin. 2015. Dynamic task scheduling methods in heterogeneous systems: a survey. *International Journal of Computer Applications* 110, 6 (2015), 12–18.

[2] Hamid Arabnejad and Jorge G. Barbosa. 2014. List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table. *IEEE Transactions on Parallel and Distributed Systems* 25, 3 (2014), 682–694. doi:10.1109/TPDS.2013.57

[3] Jonathan Bader, Fabian Lehmann, Lauritz Thamsen, Ulf Leser, and Odej Kao. 2024. Lotaru: Locally predicting workflow task runtimes for resource management on heterogeneous infrastructures. *Future*

Generation Computer Systems 150 (2024), 171–185. doi:10.1016/j.future.2023.08.022

[4] Gabriel Bathie, Loris Marchal, Yves Robert, and Samuel Thibault. 2020. Revisiting dynamic DAG scheduling under memory constraints for shared-memory platforms. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW '20)*. IEEE, 597–606. doi:10.1109/IPDPSW50202.2020.00102

[5] Anne Benoit, Swann Perarnau, Loïc Pottier, and Yves Robert. 2018. A Performance Model to Execute Workflows on High-Bandwidth-Memory Architectures. In *Proceedings of the 47th International Conference on Parallel Processing* (Eugene, OR, USA) *(ICPP '18)*. Association for Computing Machinery, Article 36, 10 pages. doi:10.1145/3225058.3225110

[6] Wanli Chang, Yili Guo, Weijie Wang, Yaqi Yao, Fuyang Zhao, Yinjie Fang, Kuan Jiang, and Liyun Shang. 2025. Invited Paper: Resource Management on Heterogeneous Chiplets Systems. In *Proceedings of the International Conference On Computer Aided Design (ICAD '25)*. 1–8. doi:10.1109/ICCAD66269.2025.11240955

[7] Jinchao Chen, Yu He, Ying Zhang, Pengcheng Han, and Chenglie Du. 2022. Energy-aware scheduling for dependent tasks in heterogeneous multiprocessor systems. *Journal of Systems Architecture* 129 (2022), 102598. doi:10.1016/j.sysarc.2022.102598

[8] Rathish Das, Kunal Agrawal, Michael A. Bender, Jonathan Berry, Benjamin Moseley, and Cynthia A. Phillips. 2020. How to Manage High-Bandwidth Memory Automatically. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '20)*. Association for Computing Machinery, 187–199. doi:10.1145/3350755.3400233

[9] Daniel DeLayo, Kenny Zhang, Kunal Agrawal, Michael A. Bender, Jonathan W. Berry, Rathish Das, Benjamin Moseley, and Cynthia A. Phillips. 2022. Automatic HBM Management: Models and Algorithms. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '22)*. Association for Computing Machinery, 147–159. doi:10.1145/3490148.3538570

[10] Hamza Djigal, Jun Feng, Jiamin Lu, and Jidong Ge. 2021. IPPTS: An Efficient Algorithm for Scientific Workflow Scheduling in Heterogeneous Computing Systems. *IEEE Transactions on Parallel and Distributed Systems* 32, 5 (2021), 1057–1071. doi:10.1109/TPDS.2020.3041829

[11] 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. doi:10.1145/3609308.3625266

[12] Birte Friesel, Marcel Lütke Dreimann, and Olaf Spinczyk. 2024. Performance Models for Task-based Scheduling with Disruptive Memory Technologies. In *Proceedings of the 2nd Workshop on Disruptive Memory Systems* (Austin, TX, USA) *(DIMES '24)*. Association for Computing Machinery, 1–8. doi:10.1145/3698783.3699376

[13] Birte Friesel and Olaf Spinczyk. 2025. Understanding Product Line Runtime Performance with Behaviour Models and Regression Model Trees. In *Proceedings of the 29th ACM International Systems and Software Product Line Conference - Volume A* (A Coruña, Spain) *(SPLC-A '25)*. New York, NY, USA, 142–148. doi:10.1145/3744915.3748472

[14] Nathan Grinsztajn, Olivier Beaumont, Emmanuel Jeannot, and Philippe Preux. 2021. READYS: A Reinforcement Learning Based Strategy for Heterogeneous Dynamic Scheduling. In *International Conference on Cluster Computing (CLUSTER '21)*. IEEE, 70–81. doi:10.1109/Cluster48925.2021.00031

[15] 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. doi:10.1109/ACCESS.2022.3174101

[16] Jonas Hollmann, Matthias Lüders, Jakob Arndt, Ioannis Kyriakopoulos, and Holger Blume. 2025. A Practical Survey on Static Task Scheduling Optimization Approaches for Heterogeneous Architectures. In *Proceedings of the Euro-Par Parallel Processing Workshops (Euro-Par '24)*. Springer Nature Switzerland, Cham, 425–437. doi:10.1007/978-3-031-90200-0

[17] Mehdi Hosseinzadeh, Elham Azhir, Jan Lansky, Stanislava Mildeova, Omed Hassan Ahmed, Mazhar Hussain Malik, and Faheem Khan. 2023. Task Scheduling Mechanisms for Fog Computing: A Systematic Survey. *IEEE Access* 11 (2023), 50994–51017. doi:10.1109/ACCESS.2023.3277826

[18] Bushra Jamil, Humaira Ijaz, Mohammad Shojafar, Kashif Munir, and Rajkumar Buyya. 2022. Resource Allocation and Task Scheduling in Fog Computing and Internet of Everything Environments: A Taxonomy, Review, and Future Directions. *ACM Comput. Surv.* 54, 11s, Article 233 (Sept. 2022), 38 pages. doi:10.1145/3513002

[19] Amanda Jayanetti and Rajkumar Buyya. 2019. J-OPT: A Joint Host and Network Optimization Algorithm for Energy-Efficient Workflow Scheduling in Cloud Data Centers. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing* (Auckland, New Zealand) *(UCC'19)*. Association for Computing Machinery, 199–208. doi:10.1145/3344341.3368822

[20] Mohammad Laghari and Didem Unat. 2017. Object Placement for High Bandwidth Memory Augmented with High Capacity Memory. In *Proceedings of the 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD '17)*. IEEE, 129–136. doi:10.1109/SBAC-PAD.2017.24

[21] 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* (London, UK) *(ICPE '24 Companion)*. Association for Computing Machinery, New York, NY, USA, 261–268. doi:10.1145/3629527.3652275

[22] Sanjaya K. Panda and Prasanta K. Jana. 2015. Efficient task scheduling algorithms for heterogeneous multi-cloud environment. *The Journal of Supercomputing* 71 (2015), 1505–1533. Issue 4. doi:10.1007/s11227-014-1376-6

[23] H. Topcuoglu, S. Hariri, and Min-You Wu. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 13, 3 (2002), 260–274. doi:10.1109/71.993206

[24] Jeffrey D Ullman. 1975. NP-complete scheduling problems. *Journal of Computer and System sciences* 10, 3 (1975), 384–393.

[25] Min Wang, Haoyuan Wang, Sibo Qiao, Jiawang Chen, Qin Xie, and Cuijuan Guo. 2025. Heterogeneous system list scheduling algorithm based on improved optimistic cost matrix. *Future Generation Computer Systems* 164 (2025), 107576. doi:10.1016/j.future.2024.107576

[26] Yong Wang, Bingtao Hu, Yixiong Feng, Zhiwu Li, Yiping Feng, and Jianrong Tan. 2023. A Decomposition-Based Approach for Multitask Scheduling With Execution Uncertainty in Industrial Internet of Things. *IEEE Internet of Things Journal* 10, 12 (2023), 10222–10235. doi:10.1109/JIOT.2023.3237727

[27] Wei Wu, Aurelien Bouteiller, George Bosilca, Mathieu Faverge, and Jack Dongarra. 2015. Hierarchical DAG Scheduling for Hybrid Distributed Systems. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '15)*. 156–165. doi:10.1109/IPDPS.2015.56

[28] Yujian Wu, Shanjiang Tang, Ce Yu, Bin Yang, Chao Sun, Jian Xiao, Hutong Wu, and Jinghua Feng. 2025. Task Scheduling in Geo-Distributed Computing: A Survey. *IEEE Transactions on Parallel and Distributed Systems* 36, 10 (2025), 2073–2088. doi:10.1109/TPDS.2025.3591010

[29] Yuming Xu, Kenli Li, Jingtong Hu, and Keqin Li. 2014. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Information Sciences* 270 (2014), 255–287. doi:10.1016/j.ins.2014.02.122

[30] Longxin Zhang, Kenli Li, Changyun Li, and Keqin Li. 2017. Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems. *Information Sciences* 379 (2017), 241–256. doi:10.1016/j.ins.2016.08.003