OSNABRÜCK UNIVERSITY

Embedded Software Systems

Towards Task-Based Scheduling on Truly Heterogeneous Systems

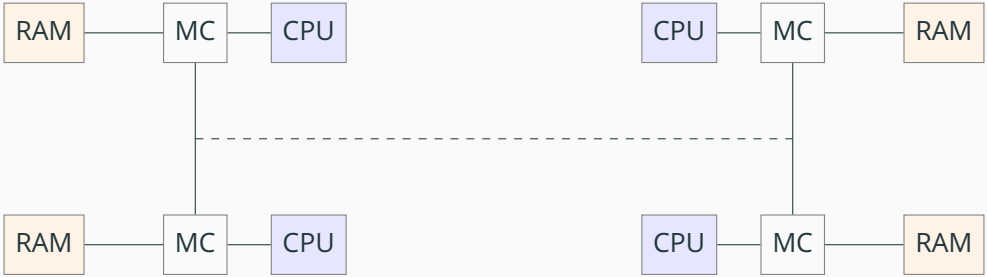
Birte Friesel, **Marcel Lütke Dreimann**, Mario Porrmann, Olaf Spinczyk

February 26th, 2026

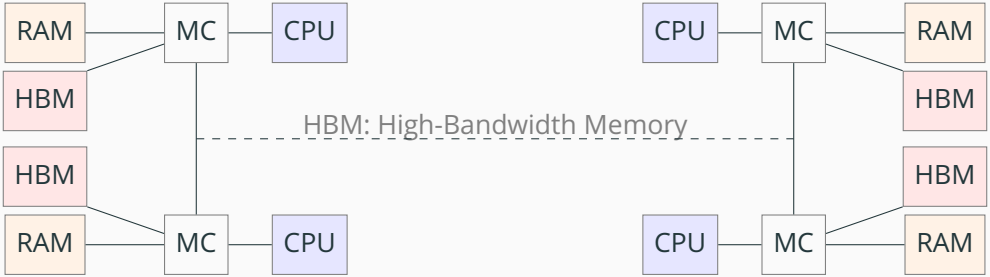
ess.cs.uos.de/~mld

marcel.luetkedreimann@uos.de

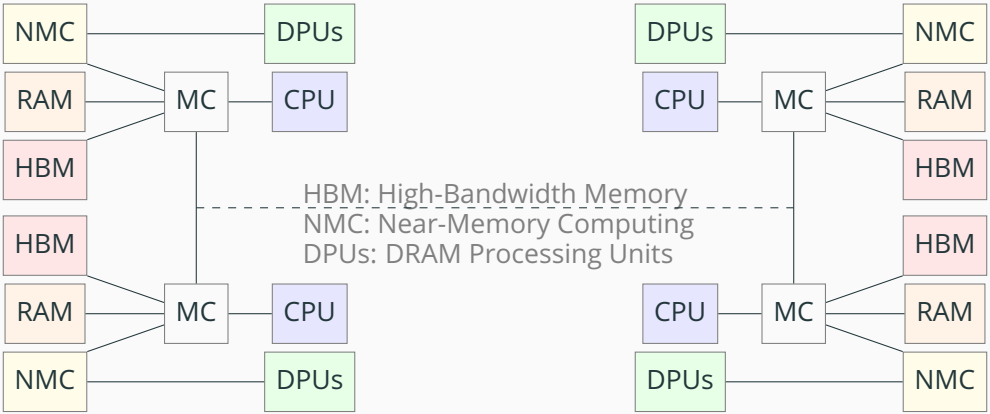
Motivation: Modern Hardware



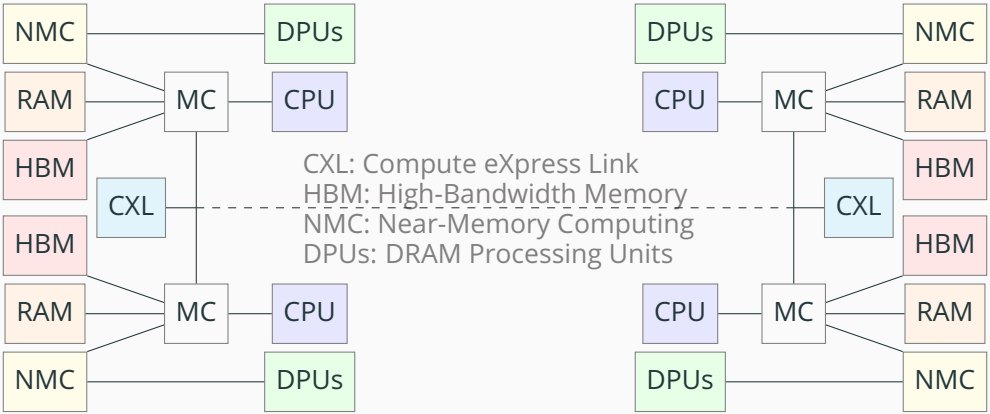
Motivation: Modern Hardware



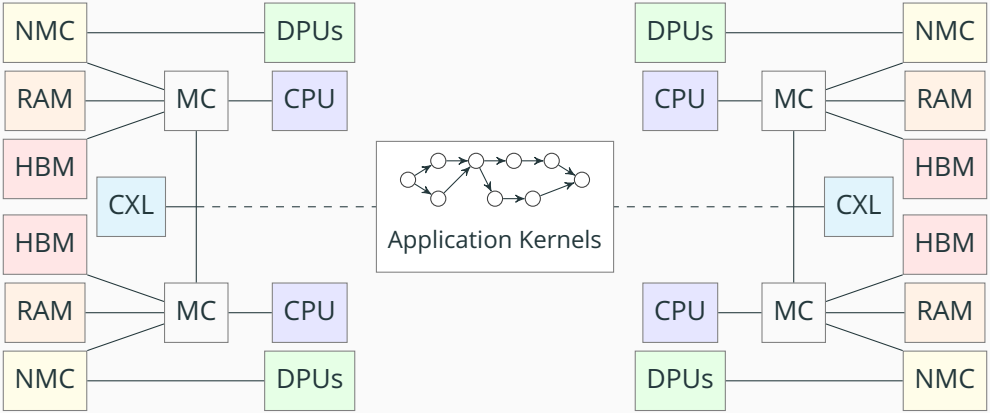
Motivation: Modern Hardware



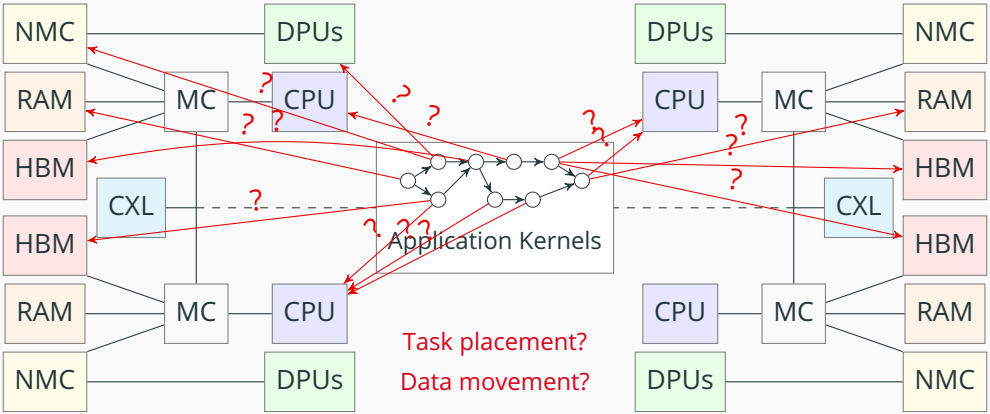
Motivation: Modern Hardware



Motivation: Modern Hardware

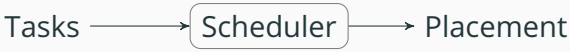


Motivation: Modern Hardware



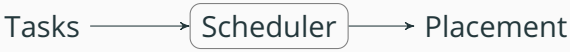
Scheduling Algorithms

Scheduling Algorithms



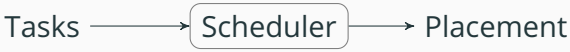
- Linux: threads ↦ CPU cores
  - Data placement: manually (e.g. libnuma) or automatic NUMA balancing

Scheduling Algorithms



- Linux: threads ↦ CPU cores
  - Data placement: manually (e.g. libnuma) or automatic NUMA balancing
- Generic: tasks, data ↦ compute units, memory pools
  - Offline placement: workload known in advance

Scheduling Algorithms

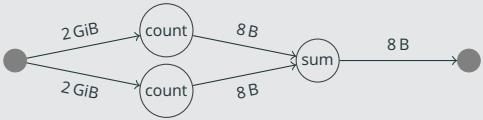


- Linux: threads ↦ CPU cores
  - Data placement: manually (e.g. libnuma) or automatic NUMA balancing
- Generic: tasks, data ↦ compute units, memory pools
  - Offline placement: workload known in advance
- Wide-spread algorithm: HEFT

HEFT: Heterogeneous Earliest Finish Time

- State of the art for the past 24 years [THW02; Bad+24]
- Input: workload as directed acyclic graph (DAG)

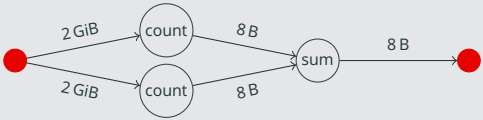
Example (SQL `COUNT(...)` `WHERE ...` Query with 2 Threads)



HEFT: Heterogeneous Earliest Finish Time

- State of the art for the past 24 years [THW02; Bad+24]
- Input: workload as directed acyclic graph (DAG)

Example (SQL `COUNT(...)` `WHERE ...` Query with 2 Threads)

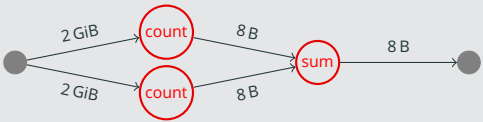


- Single start and end node

HEFT: Heterogeneous Earliest Finish Time

- State of the art for the past 24 years [THW02; Bad+24]
- Input: workload as directed acyclic graph (DAG)

Example (SQL `COUNT(...) WHERE ...` Query with 2 Threads)

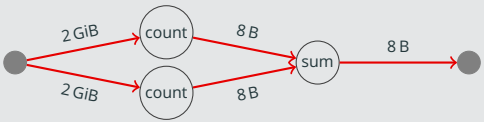


- Single start and end node
- Each inner node ≙ task (non-preemptible compute kernel)

HEFT: Heterogeneous Earliest Finish Time

- State of the art for the past 24 years [THW02; Bad+24]
- Input: workload as directed acyclic graph (DAG)

Example (SQL `COUNT(...) WHERE ...` Query with 2 Threads)



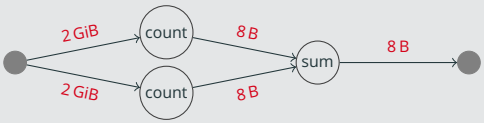
Edge E ≙ dependency

- Single start and end node
- Each inner node ≙ task (non-preemptible compute kernel)

HEFT: Heterogeneous Earliest Finish Time

- State of the art for the past 24 years [THW02; Bad+24]
- Input: workload as directed acyclic graph (DAG)

Example (SQL `COUNT(...) WHERE ...` Query with 2 Threads)



Edge $E \mathrel{\widehat{=}}$ dependency

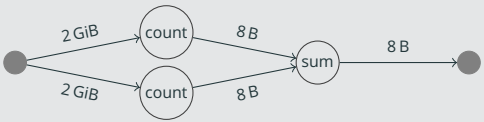
Data transfer $D : E \to \mathbb{N}_{\geq 0}$

- Single start and end node
- Each inner node $\mathrel{\widehat{=}}$ task (non-preemptible compute kernel)

HEFT: Heterogeneous Earliest Finish Time

- State of the art for the past 24 years [THW02; Bad+24]
- Input: workload as directed acyclic graph (DAG)

Example (SQL `COUNT(...) WHERE ...` Query with 2 Threads)



Edge $E \mathrel{\widehat{=}}$ dependency

Data transfer $D : E \to \mathbb{N}_{\geq 0}$

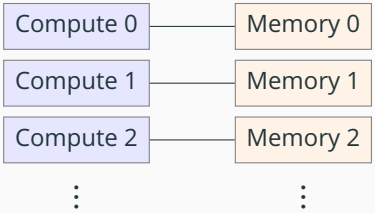
Task latency $W : V \times C \to \mathbb{R}_{\geq 0}$

- Single start and end node
- Each inner node $\mathrel{\widehat{=}}$ task (non-preemptible compute kernel)
- Task latency depends on placement (compute units C)

HEFT: Hardware / Platform Model

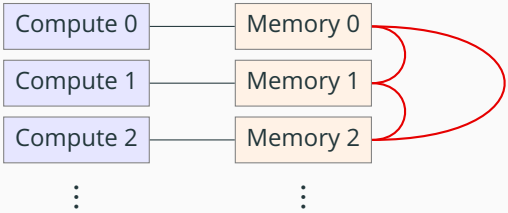
- Compute unit ≙ CPU core, GPU, FPGA, (individual HPC server), …

HEFT: Hardware / Platform Model



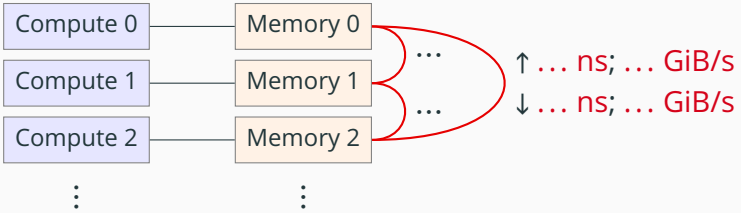
- Compute unit ≙ CPU core, GPU, FPGA, (individual HPC server), …
- One memory pool for each compute unit

HEFT: Hardware / Platform Model



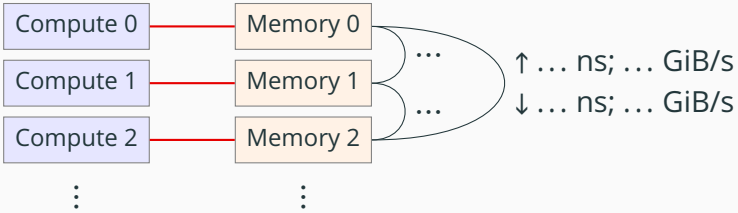
- Compute unit ≙ CPU core, GPU, FPGA, (individual HPC server), …
- One memory pool for each compute unit
- Fully-connected mesh between memory pools

HEFT: Hardware / Platform Model



- Compute unit ≙ CPU core, GPU, FPGA, (individual HPC server), …
- One memory pool for each compute unit
- Fully-connected mesh between memory pools
  - Annotated with startup latency and throughput → data transfer latency

HEFT: Hardware / Platform Model

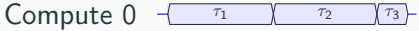


- Compute unit ≙ CPU core, GPU, FPGA, (individual HPC server), …
- One memory pool for each compute unit
- Fully-connected mesh between memory pools
  - Annotated with startup latency and throughput → data transfer latency
- Tasks always use local data

HEFT: Data Transfer

Example

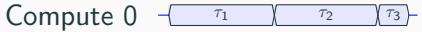
- All predecessors on same core: input data already availalable



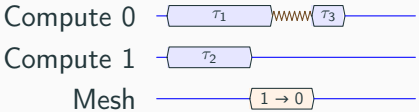
HEFT: Data Transfer

Example

- All predecessors on same core: input data already availalable



- Predecessor on non-local core → data transfer (startup latency + $\frac{\text{size}}{\text{throughput}}$)



HEFT: Task Execution Model

- Assumption: data transfer in background
  - Compute units can execute tasks during data transfer
  - No contention between concurrent data transfers

HEFT: Task Execution Model

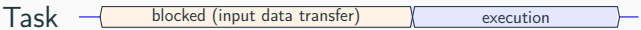
- Assumption: data transfer in background
  - Compute units can execute tasks during data transfer
  - No contention between concurrent data transfers

→ Total task-specific latency: data transfer (if needed) + execution

HEFT: Task Execution Model

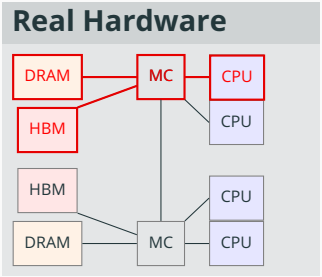
- Assumption: data transfer in background
  - Compute units can execute tasks during data transfer
  - No contention between concurrent data transfers

→ Total task-specific latency: data transfer (if needed) + execution

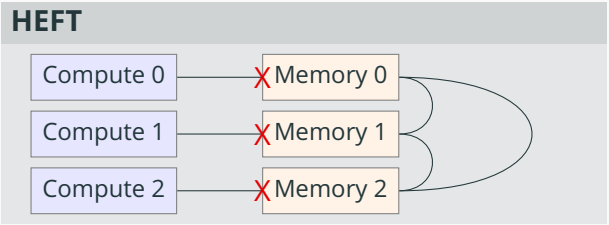


- Data transfer latency ← placement of predecessors, data size (task graph)
- Task execution latency ← placement of task

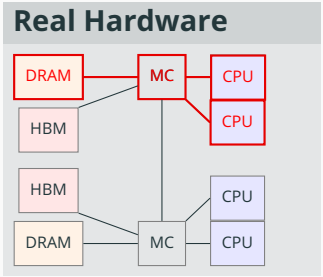
The Real World: Hardware Platform



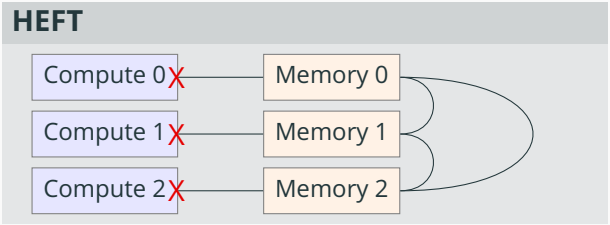
- Multiple local memories per compute unit



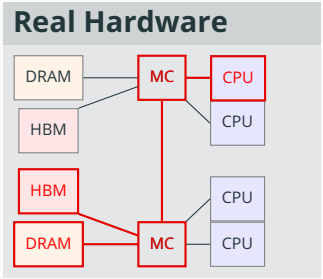
The Real World: Hardware Platform



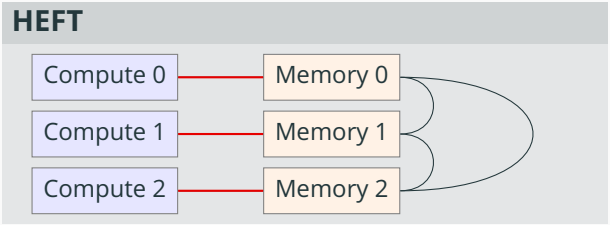
- Multiple local memories per compute unit
- Multiple compute units per memory



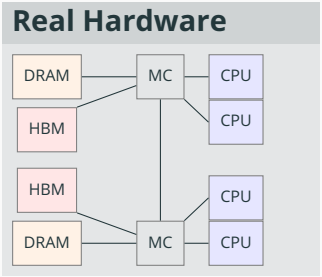
The Real World: Hardware Platform



- Multiple local memories per compute unit
- Multiple compute units per memory
- Tasks can access (some) remote memory

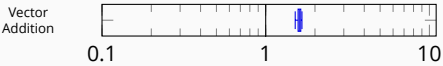


The Real World: Hardware Platform

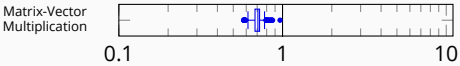


- Multiple local memories per compute unit
- Multiple compute units per memory
- Tasks can access (some) remote memory

- These challenges are not just theoretical in nature [FLS24]

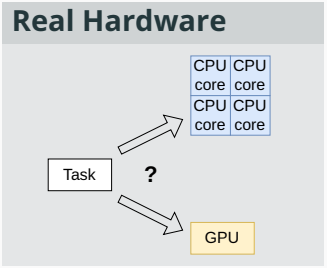


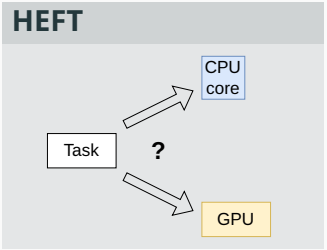
Speedup (log): remote vs. local DRAM



Speedup (log): HBM vs. DRAM

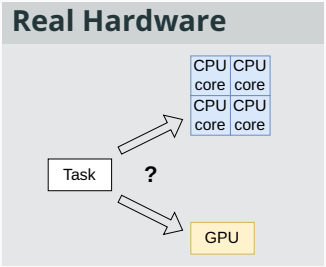
The Real World: Application

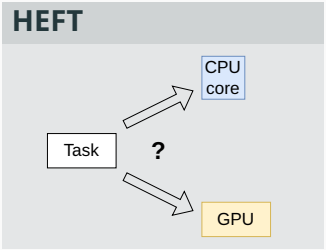




- Task granularity is different for compute units

The Real World: Application



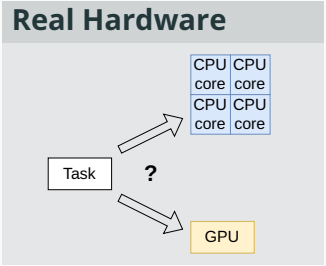


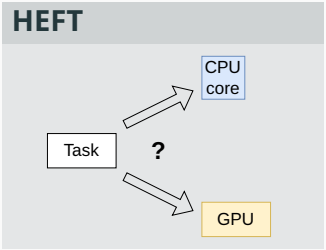
- Task granularity is different for compute units
- Some compute units require setup tasks

Task — setup @ CPU | data transfer to GPU | execution @ GPU

  - FPGA: reconfiguration
  - GPU: binary compilation and upload
  - NMC: allocation and binary upload

The Real World: Application





- Task granularity is different for compute units
- Some compute units require setup tasks

Task — setup @ CPU | data transfer to GPU | execution @ GPU

  - FPGA: reconfiguration
  - GPU: binary compilation and upload
  - NMC: allocation and binary upload

→ Is HEFT still appropriate for today's server hardware?

Task-Based Scheduling in the Literature

- HEFT: published 24 years ago [THW02], still state of the art [Bad+24]
- More than 4,500 citations; hundreds of proposed adaptations / extensions

Task-Based Scheduling in the Literature

- HEFT: published 24 years ago [THW02], still state of the art [Bad+24]
- More than 4,500 citations; hundreds of proposed adaptations / extensions

Literature review: four clusters

① Algorithm improvements (identical task / platform model) [Hol+25]

② Additional optimization goals [Hos+23; Wu+25]

③ Adaptation for specialized use cases [Ben+18]

④ No task graphs [Das+20; DeL+22; LU17]

Task-Based Scheduling in the Literature

- HEFT: published 24 years ago [THW02], still state of the art [Bad+24]
- More than 4,500 citations; hundreds of proposed adaptations / extensions

Literature review: four clusters

① Algorithm improvements (identical task / platform model) [Hol+25]

② Additional optimization goals [Hos+23; Wu+25]

③ Adaptation for specialized use cases [Ben+18]

④ No task graphs [Das+20; DeL+22; LU17]

Additional Optimization Goals: Examples

- HEFT goal: minimize makespan (workload latency) [THW02]

Additional Optimization Goals: Examples

- HEFT goal: minimize makespan (workload latency) [THW02]
- Dynamic Voltage and Frequency Scaling [Che+22; Zha+17]
  - Task latency $\propto$ CPU frequency^{-1}
  - Scheduling under energy constraints / minimize energy usage

Additional Optimization Goals: Examples

- HEFT goal: minimize makespan (workload latency) [THW02]
- Dynamic Voltage and Frequency Scaling [Che+22; Zha+17]
  - Task latency $\propto$ CPU frequency^{-1}
  - Scheduling under energy constraints / minimize energy usage
- Cloud computing / federated clouds [JB19; PJ15]
  - Goal: minimize (financial) cost of task execution

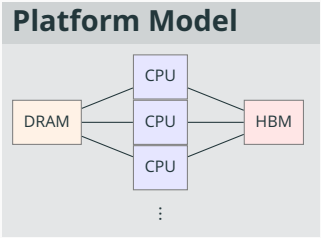
Additional Optimization Goals: Examples

- HEFT goal: minimize makespan (workload latency) [THW02]
- Dynamic Voltage and Frequency Scaling [Che+22; Zha+17]
  - Task latency $\propto$ CPU frequency^{-1}
  - Scheduling under energy constraints / minimize energy usage
- Cloud computing / federated clouds [JB19; PJ15]
  - Goal: minimize (financial) cost of task execution

→ Only minor extensions to task / platform model

HEFT Extensions: Example

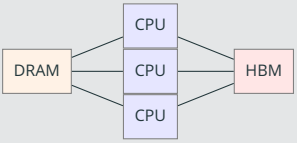
- “[…] Workflows on High-Bandwidth Memory Architectures” [Ben+18]



HEFT Extensions: Example

- “[…] Workflows on High-Bandwidth Memory Architectures” [Ben+18]

Platform Model

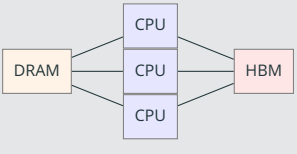


- Distinct DRAM / HBM with finite capacity

HEFT Extensions: Example

- “[…] Workflows on High-Bandwidth Memory Architectures” [Ben+18]

Platform Model

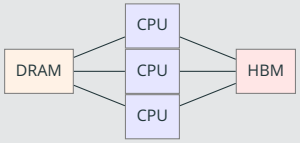


- Distinct DRAM / HBM with finite capacity
- Fine-granular data placement
  - Individual data blocks, not entire objects
  - Contention-aware

HEFT Extensions: Example

- "[…] Workflows on High-Bandwidth Memory Architectures" [Ben+18]

Platform Model

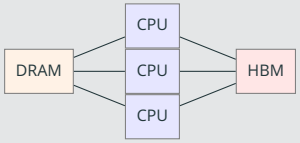


- Distinct DRAM / HBM with finite capacity
- Fine-granular data placement
  - Individual data blocks, not entire objects
  - Contention-aware
- Data access: continuous streams

HEFT Extensions: Example

- "[…] Workflows on High-Bandwidth Memory Architectures" [Ben+18]

Platform Model

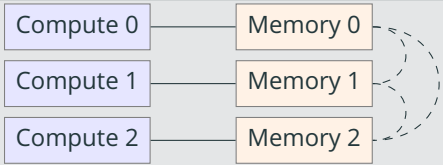


- Distinct DRAM / HBM with finite capacity
- Fine-granular data placement
  - Individual data blocks, not entire objects
  - Contention-aware
- Data access: continuous streams

- Just two memories; no support for NUMA, CXL, GPUs, FPGAs, NMC, …

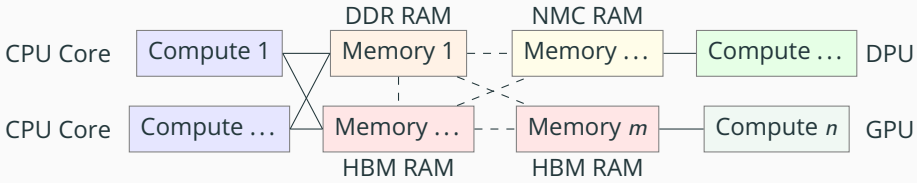
Proposed Platform Model

HEFT



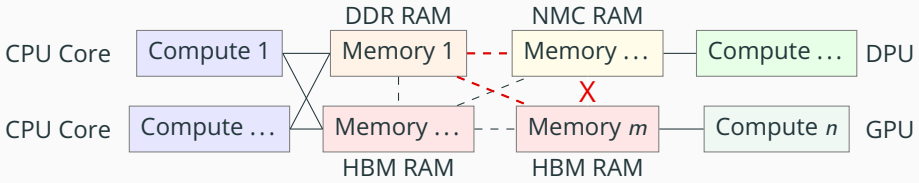
- Compute ≙ memory (1:1 map)
- Compute: only local memory
- Memory: mesh topology

Proposed Platform Model



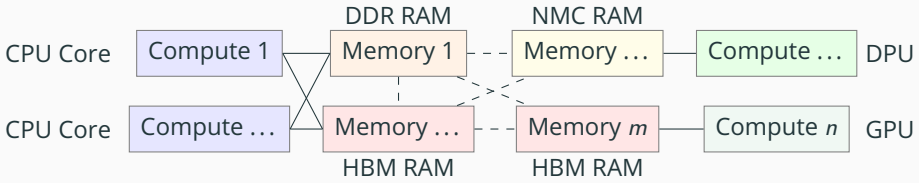
- Independent sets of compute units and memories

Proposed Platform Model



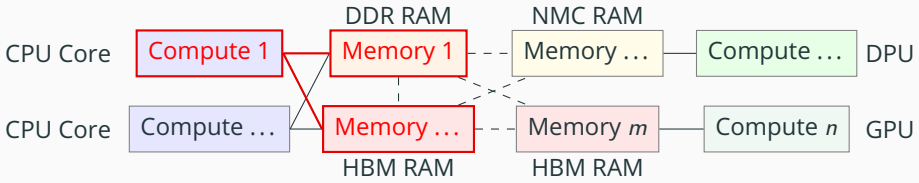
- Independent sets of compute units and memories
- Memory: no mesh topology (e.g. GPU ↔ DRAM ↔ NMC RAM)

Proposed Platform Model



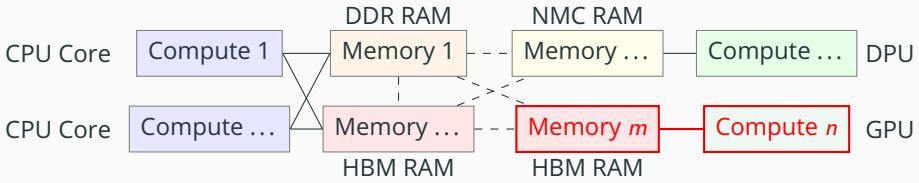
- Independent sets of compute units and memories
- Memory: no mesh topology (e.g. GPU ↔ DRAM ↔ NMC RAM)
- Fine-grained compute ↔ memory access (e.g. CPU vs. dedicated GPU)

Proposed Platform Model



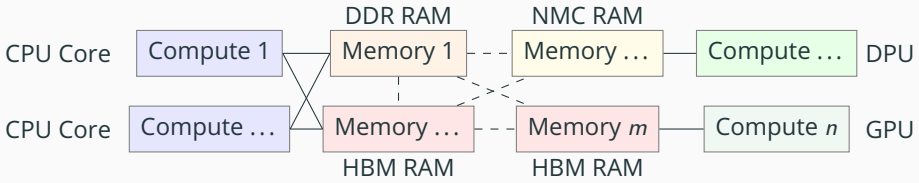
- Independent sets of compute units and memories
- Memory: no mesh topology (e.g. GPU ↔ DRAM ↔ NMC RAM)
- Fine-grained compute ↔ memory access (e.g. CPU vs. dedicated GPU)

Proposed Platform Model



- Independent sets of compute units and memories
- Memory: no mesh topology (e.g. GPU ↔ DRAM ↔ NMC RAM)
- Fine-grained compute ↔ memory access (e.g. CPU vs. dedicated GPU)

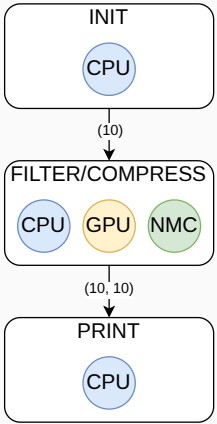
Proposed Platform Model



- Independent sets of compute units and memories
- Memory: no mesh topology (e.g. GPU ↔ DRAM ↔ NMC RAM)
- Fine-grained compute ↔ memory access (e.g. CPU vs. dedicated GPU)
- Challenge: model / makespan accuracy vs. algorithmic complexity
  → No contention or block-wise data operations (yet)

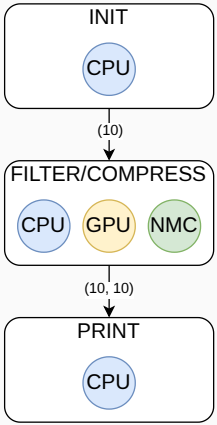
Proposed Application Model (1): Prim SEL [FLS23]

- Each node has an implementation for a subset of compute units



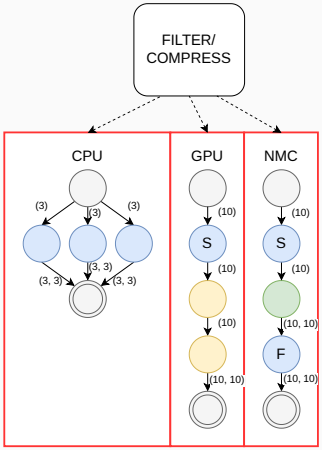
Proposed Application Model (1): Prim SEL [FLS23]

- Each node has an implementation for a subset of compute units
- Instead of fixed data size, edges have a set of data object sizes
  - Improves the accuracy if many small data objects are shared



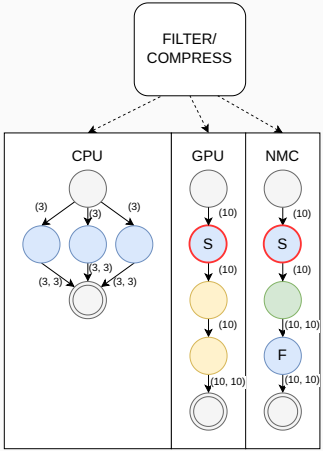
Proposed Application Model (2): Prim SEL [FLS23]

- Define a subgraph for each compute unit
  - Accelerator-specific task granularity



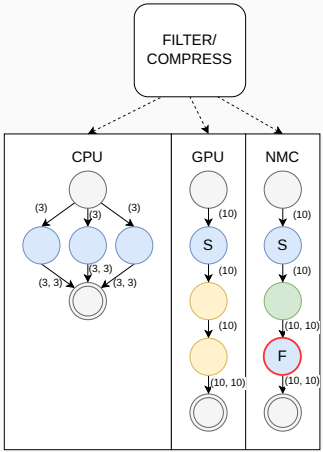
Proposed Application Model (2): Prim SEL [FLS23]

- Define a subgraph for each compute unit
  - Accelerator-specific task granularity
  - Setup can be represented and scheduled on appropriate compute unit



Proposed Application Model (2): Prim SEL [FLS23]

- Define a subgraph for each compute unit
  - Accelerator-specific task granularity
  - Setup can be represented and scheduled on appropriate compute unit
  - Accelerator-specific follow-up tasks



HEFT Limitations: Example

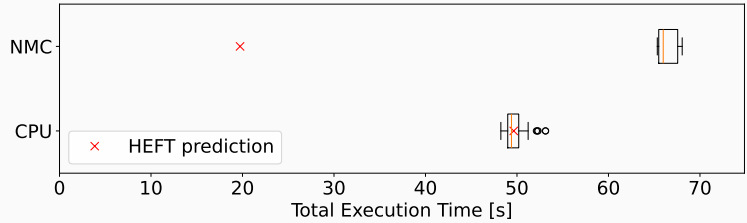
- Prim SEL on a server with 88 threads and 2500 DPUs
- HEFT decides for NMC based on compute power and transfer time

HEFT Limitations: Example

- Prim SEL on a server with 88 threads and 2500 DPUs
- HEFT decides for NMC based on compute power and transfer time
- Measurement of total execution time:

HEFT Limitations: Example

- Prim SEL on a server with 88 threads and 2500 DPUs
- HEFT decides for NMC based on compute power and transfer time
- Measurement of total execution time:



$\Longrightarrow$ HEFT decision not optimal because of setup, follow-up and bandwidth r/w difference

Conclusion

- HEFT's assumptions do not hold anymore for today's server hardware
- Update of platform and task model is overdue
- Proposed solutions
  - Subgraphs in DAGs
  - Extended platform model
- Open question: Overhead of more complex model

References i

[Bad+24] Jonathan Bader et al. **“Lotaru: Locally predicting workflow task runtimes for resource management on heterogeneous infrastructures”.** In: Future Generation Computer Systems 150 (2024), pp. 171–185. ISSN: 0167-739X. DOI: 10.1016/j.future.2023.08.022.

[Ben+18] Anne Benoit et al. **“A Performance Model to Execute Workflows on High-Bandwidth-Memory Architectures”.** In: Proceedings of the 47th International Conference on Parallel Processing. ICPP ’18. Eugene, OR, USA: Association for Computing Machinery, 2018. ISBN: 9781450365109. DOI: 10.1145/3225058.3225110. URL: https://doi.org/10.1145/3225058.3225110.

References ii

[Che+22] Jinchao Chen et al. **“Energy-aware scheduling for dependent tasks in heterogeneous multiprocessor systems”.** In: Journal of Systems Architecture 129 (2022), p. 102598. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2022.102598.

[Das+20] Rathish Das et al. **“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, 2020, pp. 187–199. ISBN: 9781450369350. DOI: 10.1145/3350755.3400233.

References iii

[DeL+22] Daniel DeLayo et al. **“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, 2022, pp. 147–159. ISBN: 9781450391467. DOI: 10.1145/3490148.3538570.

[FLS23] Birte Friesel, Marcel Lütke Dreimann, and Olaf Spinczyk. **“A Full-System Perspective on UPMEM Performance”.** In: Proceedings of the 1st Workshop on Disruptive Memory Systems. DIMES ’23. Koblenz, Germany: Association for Computing Machinery, Oct. 2023, pp. 1–7. ISBN: 979-8-4007-0300-3. DOI: 10.1145/3609308.3625266.

References iv

[FLS24] Birte Friesel, Marcel Lütke Dreimann, and Olaf Spinczyk. **“Performance Models for Task-based Scheduling with Disruptive Memory Technologies”.** In: Proceedings of the 2nd Workshop on Disruptive Memory Systems. DIMES ’24. Austin, TX, USA: Association for Computing Machinery, Nov. 2024, pp. 1–8. ISBN: 979-8-4007-1303-3. DOI: 10.1145/3698783.3699376.

[Hol+25] Jonas Hollmann et al. **“A Practical Survey on Static Task Scheduling Optimization Approaches for Heterogeneous Architectures”.** In: Proceedings of the Euro-Par Parallel Processing Workshops. Euro-Par ’24. Cham: Springer Nature Switzerland, 2025, pp. 425–437. DOI: 10.1007/978-3-031-90200-0.

References v

[Hos+23] Mehdi Hosseinzadeh et al. **“Task Scheduling Mechanisms for Fog Computing: A Systematic Survey”.** In: IEEE Access 11 (2023), pp. 50994–51017. DOI: 10.1109/ACCESS.2023.3277826.

[JB19] Amanda Jayanetti and Rajkumar Buyya. **“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. UCC’19. Auckland, New Zealand: Association for Computing Machinery, 2019, pp. 199–208. ISBN: 9781450368940. DOI: 10.1145/3344341.3368822.

References vi

[LU17] Mohammad Laghari and Didem Unat. **"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, 2017, pp. 129–136. DOI: 10.1109/SBAC-PAD.2017.24.

[PJ15] Sanjaya K. Panda and Prasanta K. Jana. **"Efficient task scheduling algorithms for heterogeneous multi-cloud environment".** In: The Journal of Supercomputing 71 (4 2015), pp. 1505–1533. ISSN: 1573-0484. DOI: 10.1007/s11227-014-1376-6.

References vii

[THW02] H. Topcuoglu, S. Hariri, and Min-You Wu. **"Performance-effective and low-complexity task scheduling for heterogeneous computing".** In: IEEE Transactions on Parallel and Distributed Systems 13.3 (2002), pp. 260–274. DOI: 10.1109/71.993206.

[Wu+25] Yujian Wu et al. **"Task Scheduling in Geo-Distributed Computing: A Survey".** In: IEEE Transactions on Parallel and Distributed Systems 36.10 (2025), pp. 2073–2088. DOI: 10.1109/TPDS.2025.3591010.

[Zha+17] Longxin Zhang et al. **"Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems".** In: Information Sciences 379 (2017), pp. 241–256. ISSN: 0020-0255. DOI: 10.1016/j.ins.2016.08.003.