Programming Fully Disaggregated Systems

Christoph Anneser, Lukas Vogel, Ferdinand Gruber, Maximilian Bandle, Jana Giceva
School of Computation, Information and Technology
Technical University of Munich

19th Workshop on Hot Topics in Operating Systems
Providence, Rhode Island, USA
22–24 June 2023
Motivation

How can we develop & optimize applications for heterogeneous, disaggregated environments?
Motivation

How can we develop & optimize applications for heterogeneous, disaggregated environments?
Motivation

How can we develop & optimize applications for heterogeneous, disaggregated environments?

Christoph Anneser (TUM) | Programming Fully Disaggregated Systems | June 24, 2023
Motivation

Table: Memory device properties as seen from a CPU.

<table>
<thead>
<tr>
<th>Name</th>
<th>Bw.</th>
<th>Lat.</th>
<th>Gran.</th>
<th>Attached</th>
<th>Sync</th>
<th>Persist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>++</td>
<td>++</td>
<td>1 B</td>
<td>CPU</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>HBM</td>
<td>++</td>
<td>+</td>
<td>64 B</td>
<td>CPU</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>DRAM</td>
<td>+</td>
<td>+</td>
<td>64 B</td>
<td>CPU</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>PMem</td>
<td>◦</td>
<td>◦</td>
<td>256 B</td>
<td>CPU</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CXL-DRAM</td>
<td>◦</td>
<td>◦</td>
<td>64 B</td>
<td>PCIe</td>
<td>✓/✓</td>
<td>✓/✓</td>
</tr>
<tr>
<td>Disagg. Mem.</td>
<td>◦</td>
<td>−</td>
<td>?</td>
<td>NIC</td>
<td>X</td>
<td>✓/✓</td>
</tr>
<tr>
<td>SSD</td>
<td>−</td>
<td>−</td>
<td>4 KiB</td>
<td>PCIe</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>HDD</td>
<td>−</td>
<td>−</td>
<td>4 KiB</td>
<td>SATA</td>
<td>X</td>
<td>✓</td>
</tr>
</tbody>
</table>
Motivation

Table: Memory device properties as seen from a CPU.

<table>
<thead>
<tr>
<th>Name</th>
<th>Bw.</th>
<th>Lat.</th>
<th>Gran.</th>
<th>Attached</th>
<th>Sync</th>
<th>Persist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>++</td>
<td>++</td>
<td>1 B</td>
<td>CPU</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>HBM</td>
<td>++</td>
<td>+</td>
<td>64 B</td>
<td>CPU</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>DRAM</td>
<td>+</td>
<td>+</td>
<td>64 B</td>
<td>CPU</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>PMem</td>
<td>◦</td>
<td>◦</td>
<td>256 B</td>
<td>CPU</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CXL-DRAM</td>
<td>◦</td>
<td>◦</td>
<td>64 B</td>
<td>PCIe</td>
<td>✓/✗</td>
<td>✓/✗</td>
</tr>
<tr>
<td>Disagg. Mem.</td>
<td>◦</td>
<td>−</td>
<td>?</td>
<td>NIC</td>
<td>X</td>
<td>✓/✗</td>
</tr>
<tr>
<td>SSD</td>
<td>−</td>
<td>−</td>
<td>4 KiB</td>
<td>PCIe</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HDD</td>
<td>−−</td>
<td>−−</td>
<td>4 KiB</td>
<td>SATA</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

⇒ How can we develop & optimize applications for heterogeneous, disaggregated environments?
CXL Enables a Memory-Centric View

- Leverage memory regions as abstraction layer for disaggregated memory!
- Memory Regions are logical view on physical memory!

CXL Enables a Memory-Centric View

Memory Pool

- DRAM 1
- DRAM 2
- PMEM
- CXL DRAM

Latency: low
Bandwidth: high
Latency: medium
Persistent: ✓

Tofte and Talpin: “Region-based Memory Management” (1997)
CXL Enables a Memory-Centric View

- Leverage memory regions as abstraction layer for disaggregated memory!
- Memory Regions are logical view on physical memory!

CXL Enables a Memory-Centric View

CXL Enables a Memory-Centric View


Christoph Anneser (TUM) | Programming Fully Disaggregated Systems | June 24, 2023
CXL Enables a Memory-Centric View

– Leverage memory regions [1, 3] as abstraction layer for disaggregated memory!

CXL Enables a Memory-Centric View

– Leverage memory regions [1, 3] as abstraction layer for disaggregated memory!

CXL Enables a Memory-Centric View

- Leverage memory regions \([1, 3]\) as abstraction layer for disaggregated memory!

CXL Enables a Memory-Centric View

– Leverage memory regions [1, 3] as abstraction layer for disaggregated memory!

CXL Enables a Memory-Centric View

– Leverage **memory regions** [1, 3] as abstraction layer for disaggregated memory!
– Memory Regions are logical view on physical memory!

---

## Mapping Memory Regions to Devices

### Memory Region Properties:

- **MR 1**: low lat., sync
- **MR 2**: low lat., persistent, async
- **MR 3**: low lat., high bandwidth, sync

### Handovers:

- **MR 1**: T0 Output, T1 Input
- **MR 2**: T1 Output, T2 Input

### Device Utilization

<table>
<thead>
<tr>
<th></th>
<th>DRAM</th>
<th>PMEM</th>
<th>GDDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CPU**

**GPU**

**Runtime System**

- Flexible mapping at runtime → Late Binding

Christoph Anneser (TUM) | Programming Fully Disaggregated Systems | June 24, 2023
Mapping Memory Regions to Devices

- Memory Region Properties:
  - MR 1: low lat., sync
  - MR 2: low lat., persistent, async
  - MR 3: low lat., high bandwidth, sync

- Handovers:
  - MR 1: T0 Output, T1 Input
  - MR 2: T1 Output, T2 Input

- Device Utilization:
  - DRAM
  - PMEM
  - GDDR

- CPU
  - Task 1
  - Task 0
  - Task 2

- GPU

- Runtime System

Flexible mapping at runtime → Late Binding
Mapping Memory Regions to Devices

- Task Placement

---

- **Task 0**
  - CPU

- **Task 1**
  - CPU

- **Task 2**
  - GPU
Mapping Memory Regions to Devices

- Task Placement

- Memory Region Properties:
  - MR$_1$: low lat., sync
  - MR$_2$: low lat., persistent, async
  - MR$_3$: low lat., high bandwidth, sync

Diagram:

- Tasks: Task 0 → Task 1 → Task 2
- Devices: CPU → GPU

Flexible mapping at runtime → Late Binding
Mapping Memory Regions to Devices

– Task Placement

– Memory Region Properties:
  - \( MR_1 \): low lat., sync
  - \( MR_2 \): low lat., persistent, async
  - \( MR_3 \): low lat., high bandwidth, sync

![Diagram of task and memory region placement](image-url)
Mapping Memory Regions to Devices

- Task Placement

- Memory Region Properties:
  - MR₁: low lat., sync
  - MR₂: low lat., persistent, async
  - MR₃: low lat., high bandwidth, sync
Mapping Memory Regions to Devices

- Task Placement

- Memory Region Properties:
  - \( \text{MR}_1 \): low lat., sync
  - \( \text{MR}_2 \): low lat., persistent, async
  - \( \text{MR}_3 \): low lat., high bandwidth, sync

---

**Runtime System**

```
CPU
  ┌─────────────────────┐
  │ MR1                │ DRAM             │ MR1          │
  │                   ├── PMEM           └── MR2          │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  └─────────────────────┘

GPU
  ┌─────────────────────┐
  │ MR3                │ GDDR             │ MR3          │
  │                   ├── PMEM           └── MR2          │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  │                   │                 │               │
  └─────────────────────┘
```

---

Handovers:
- \( \text{MR}_1 \): T0 Output, T1 Input
- \( \text{MR}_2 \): T1 Output, T2 Input

- Device Utilization

- Flexible mapping at runtime → Late Binding

Christoph Anneser (TUM) | Programming Fully Disaggregated Systems | June 24, 2023
Mapping Memory Regions to Devices

– Task Placement

– Memory Region Properties:
  
  MR₁: low lat., sync  
  MR₂: low lat., persistent, async  
  MR₃: low lat., high bandwidth, sync
Mapping Memory Regions to Devices

- Task Placement

- Memory Region Properties:
  - \( MR_1 \): low lat., sync
  - \( MR_2 \): low lat., persistent, async
  - \( MR_3 \): low lat., high bandwidth, sync

Runtime System

- Flexible mapping at runtime → Late Binding
Mapping Memory Regions to Devices

– Task Placement

– Memory Region Properties:
  - MR$_1$: low lat., sync
  - MR$_2$: low lat., persistent, async
  - MR$_3$: low lat., high bandwidth, sync

– Handovers:
  - MR$_1$: T0 Output, T1 Input
  - MR$_2$: T1 Output, T2 Input

runtime System

Flexible mapping at runtime → Late Binding
Mapping Memory Regions to Devices

– Task Placement

– Memory Region Properties:
  - MR₁: low lat., sync
  - MR₂: low lat., persistent, async
  - MR₃: low lat., high bandwidth, sync

– Handovers:
  - MR₁: T₀ Output, T₁ Input
  - MR₂: T₁ Output, T₂ Input

– Device Utilization

![Diagram showing task placement and device utilization with MR properties and handovers]
Mapping Memory Regions to Devices

– Task Placement

– Memory Region Properties:
  MR$_1$ : low lat., sync
  MR$_2$ : low lat., persistent, async
  MR$_3$ : low lat., high bandwidth, sync

– Handovers:
  MR$_1$ : T0 Output, T1 Input
  MR$_2$ : T1 Output, T2 Input

– Device Utilization

Flexible mapping at runtime $\rightarrow$ Late Binding
Dataflow Systems on Disaggregated Systems
Dataflow Systems on Disaggregated Systems

Declaratively attach properties to:
- Memory Regions
- Tasks
- Pipelines
- Jobs
- Applications

Job 1

\[ \cdots \]

T 1

T 2

T 3

T 4

T 5

Pipeline
Dataflow Systems on Disaggregated Systems
Dataflow Systems on Disaggregated Systems

- Memory Regions
- Tasks
- Pipelines
- Jobs
- Applications

Job 1
Job 2
...
T 1
T 2
T 3
T 4
T 5

Pipeline

confidential
materialize
Declaratively attach properties to

- Memory Regions
- Tasks
- Pipelines
- Jobs
- Applications
Declaratively attach properties to
- Memory Regions

![Diagram showing dataflow systems on disaggregated systems with jobs, tasks, and pipelines.](image-url)
Dataflow Systems on Disaggregated Systems

Declaratively attach properties to
- Memory Regions
- Tasks
Dataflow Systems on Disaggregated Systems

Declaratively attach properties to
- Memory Regions
- Tasks
- Pipelines
Dataflow Systems on Disaggregated Systems

Declaratively attach properties to
– Memory Regions
– Tasks
– Pipelines
– Jobs
Dataflow Systems on Disaggregated Systems

Declaratively attach properties to
– Memory Regions
– Tasks
– Pipelines
– Jobs
– Applications
Typed Memory Regions [2]

Dataflow applications use memory for . . .

Typed Memory Regions [2]

Dataflow applications use memory for . . .

. . . Communication
  – Purpose: Syncing tasks, message passing, . . .
  – Properties: coherent, sync

Typed Memory Regions [2]

Dataflow applications use memory for . . .

. . . Communication
- Purpose: Syncing tasks, message passing, . . .
- Properties: coherent, sync

⇒ Global State

Typed Memory Regions [2]

Dataflow applications use memory for . . .

. . . Communication

– Purpose: Syncing tasks, message passing, . . .
– Properties: coherent, sync

. . . Exchanging Data (e.g. Caching)

– Properties: coherent, async

⇒ Global State

Typed Memory Regions [2]

Dataflow applications use memory for . . .

. . . Communication
- Purpose: Syncing tasks, message passing, . . .
- Properties: coherent, sync

. . . Exchanging Data (e.g. Caching)
- Properties: coherent, async

⇒ Global State
⇒ Global Scratch

Typed Memory Regions [2]

Dataflow applications use memory for . . .

. . . Communication
– Purpose: Syncing tasks, message passing, . . .
– Properties: coherent, sync

⇒ Global State

. . . Exchanging Data (e.g. Caching)
– Properties: coherent, async

⇒ Global Scratch

. . . Thread-local State
– Properties: non-coherent, sync, fast

Typed Memory Regions [2]

Dataflow applications use memory for . . .

. . . Communication
– Purpose: Syncing tasks, message passing, . . .
– Properties: coherent, sync

. . . Exchanging Data (e.g. Caching)
– Properties: coherent, async

. . . Thread-local State
– Properties: non-coherent, sync, fast

⇒ Global State
⇒ Global Scratch
⇒ Private Scratch

How will different application types use the Typed Memory Regions?

<table>
<thead>
<tr>
<th>Priv. Scratch</th>
<th>Glob. State</th>
<th>Glob. Scratch</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS</td>
<td>operator state (hashtables, ...)</td>
<td>synchronization (locks, ...)</td>
</tr>
<tr>
<td></td>
<td>(temp) indexes, caches</td>
<td></td>
</tr>
<tr>
<td>ML/AI</td>
<td>model training state (metadata, worker state)</td>
<td>input data, cached transf. data</td>
</tr>
<tr>
<td>HPC</td>
<td>node-local working mem. (job metadata, node states)</td>
<td>object/blob storage</td>
</tr>
<tr>
<td>Streaming</td>
<td>cache/buffer (send, recv.)</td>
<td>cluster/worker state (result/data cache)</td>
</tr>
</tbody>
</table>
Typed Memory Regions – cont’d

How will different application types use the Typed Memory Regions?

<table>
<thead>
<tr>
<th></th>
<th>Priv. Scratch</th>
<th>Glob. State</th>
<th>Glob. Scratch</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS</td>
<td>operator state (hashtables, . . .)</td>
<td>synchronization (locks, . . .)</td>
<td>(temp) indexes, caches</td>
</tr>
</tbody>
</table>
Typed Memory Regions – cont’d

How will different application types use the Typed Memory Regions?

<table>
<thead>
<tr>
<th></th>
<th>Priv. Scratch</th>
<th>Glob. State</th>
<th>Glob. Scratch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBMS</strong></td>
<td>operator state</td>
<td>synchronization</td>
<td>(temp) indexes, caches</td>
</tr>
<tr>
<td></td>
<td>(hashtables, …)</td>
<td>(locks, …)</td>
<td></td>
</tr>
<tr>
<td><strong>ML/AI</strong></td>
<td>model training state</td>
<td>metadata, worker state</td>
<td>input data, cached transf. data</td>
</tr>
</tbody>
</table>
How will different application types use the Typed Memory Regions?

<table>
<thead>
<tr>
<th>Application Type</th>
<th>Priv. Scratch</th>
<th>Glob. State</th>
<th>Glob. Scratch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBMS</strong></td>
<td>operator state (hashables, ...)</td>
<td>synchronization (locks, ...)</td>
<td>(temp) indexes, caches</td>
</tr>
<tr>
<td><strong>ML/AI</strong></td>
<td>model training state</td>
<td>metadata, worker state</td>
<td>input data, cached transf. data</td>
</tr>
<tr>
<td><strong>HPC</strong></td>
<td>node-local working mem.</td>
<td>job metadata, node states</td>
<td>object/blob storage</td>
</tr>
</tbody>
</table>
Typed Memory Regions – cont’d

How will different application types use the Typed Memory Regions?

<table>
<thead>
<tr>
<th>Application Type</th>
<th>Priv. Scratch</th>
<th>Glob. State</th>
<th>Glob. Scratch</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS</td>
<td>operator state (hashables, …)</td>
<td>synchronization (locks, …)</td>
<td>(temp) indexes, caches</td>
</tr>
<tr>
<td>ML/AI</td>
<td>model training state</td>
<td>metadata, worker state</td>
<td>input data, cached transf. data</td>
</tr>
<tr>
<td>HPC</td>
<td>node-local working mem.</td>
<td>job metadata, node states</td>
<td>object/blob storage</td>
</tr>
<tr>
<td>Streaming</td>
<td>cache/buffer (send, recv.)</td>
<td>cluster/worker state</td>
<td>result/data cache</td>
</tr>
</tbody>
</table>
Lifetime Management of Memory Regions

- Ownership of Memory Regions:
  - Unique Ownership
  - Transfer Ownership
  - Shared Ownership

Task 1
Task 2

Task 3
Task 2
Lifetime Management of Memory Regions

- **Challenge**: Memory Regions might outlive CPU/GPU/... Tasks/Processes
Lifetime Management of Memory Regions

- **Challenge**: Memory Regions might outlive CPU/GPU/. . . Tasks/Processes
- **Ownership** of Memory Regions:

  **Unique Ownership**

  ![Diagram](image-url)
Lifetime Management of Memory Regions

- **Challenge:** Memory Regions might outlive CPU/GPU/. . . Tasks/Processes
- **Ownership** of Memory Regions:

  **Unique Ownership**

  Task \(\rightarrow\) Memory Region

  **Transfer Ownership**

  Task 1 \(\rightarrow\) Memory Region
  Task 2

  Task 1 \(\rightarrow\) own Memory Region
  Task 2

  Transfer Ownership
Lifetime Management of Memory Regions

– **Challenge**: Memory Regions might outlive CPU/GPU/... Tasks/Processes
– **Ownership** of Memory Regions:

**Unique Ownership**

- Task → Memory Region

**Transfer Ownership**

- Task 1 → Memory Region → Task 2

**Shared Ownership**

- Task 1 → Memory Region
- Task 3 → Memory Region
- Task 2 → Memory Region
Our Vision

**Challenge:** Developing dataflow applications for fully disaggregated systems

We propose a new programming model:

1. A memory-centric view based on *logical memory regions*
Our Vision

**Challenge:** Developing dataflow applications for fully disaggregated systems

We propose a new programming model:

1. A memory-centric view based on *logical memory regions*
2. Requesting memory *declaratively* based on properties
Our Vision

**Challenge:** Developing dataflow applications for fully disaggregated systems

We propose a new programming model:

1. A memory-centric view based on *logical memory regions*
2. Requesting memory *declaratively* based on properties
3. *Typed memory regions*
Our Vision

Challenge: Developing dataflow applications for fully disaggregated systems

We propose a new programming model:

(1) A memory-centric view based on *logical memory regions*

(2) Requesting memory *declaratively* based on properties

(3) *Typed memory regions*

(4) A *runtime system* that co-optimizes data- and compute-placement
The Way Forward

How can we turn our vision into a programming model?

(1) The Runtime System . . .
   – What functionality is required from the RTS?
The Way Forward

How can we turn our vision into a programming model?

(1) The Runtime System . . .
   - What functionality is required from the RTS?
   - Where should the RTS/control plane be placed?
The Way Forward

How can we turn our vision into a programming model?

(1) The Runtime System . . .
   - What functionality is required from the RTS?
   - Where should the RTS/control plane be placed?
   - Page- or object-based memory allocations?
The Way Forward

How can we turn our vision into a programming model?

(1) The Runtime System . . .
   – What functionality is required from the RTS?
   – Where should the RTS/control plane be placed?
   – Page- or object-based memory allocations?
   – What layer supports the RTS memory deployment?
The Way Forward

How can we turn our vision into a programming model?

(1) The Runtime System . . .
   – What functionality is required from the RTS?
   – Where should the RTS/control plane be placed?
   – Page- or object-based memory allocations?
   – What layer supports the RTS memory deployment?

(2) The Programming Model . . .
The Way Forward

How can we turn our vision into a programming model?

(1) The Runtime System . . .
   – What functionality is required from the RTS?
   – Where should the RTS/control plane be placed?
   – Page- or object-based memory allocations?
   – What layer supports the RTS memory deployment?

(2) The Programming Model . . .
   – Can we combine declarative and iterative concepts?
The Way Forward

How can we turn our vision into a programing model?

(1) The Runtime System . . .
   – What functionality is required from the RTS?
   – Where should the RTS/control plane be placed?
   – Page- or object-based memory allocations?
   – What layer supports the RTS memory deployment?

(2) The Programming Model . . .
   – Can we combine declarative and iterative concepts?

Thank you for your attention!

anneser@in.tum.de
Sources

Roadmap

- Set-up the testbed
  - Machine with CXL-support (e.g. Intel's Sapphire Rapids)
  - Identify suitable Workloads for CXL
    - Dataflow over heterogeneous hardware
    - Memory extension for large intermediate state
  - Understand CXL's performance implications through benchmarks

- Design and implement the RTS key components & building blocks
  - CXL-enabled vmcache, overcoming the accelerator’s limited capacities
  - Optimizer for data- and compute-placement