Jenga: Software-Defined Cache Hierarchies

Po-An Tsai, Nathan Beckmann, and Daniel Sanchez
Executive summary

- Heterogeneous caches are traditionally organized as a **rigid** hierarchy
  - Easy to program but introduce expensive overheads when hierarchy is not helpful

- Jenga builds **application-specific** cache hierarchies on the fly

- Key contribution: New algorithms to find near-optimal hierarchies
  - Arbitrary application behaviors & changing resource constraints
  - Full system optimization at 36 cores in <1 ms

- Jenga improves EDP by up to 85% vs. state-of-the-art
Deep, rigid hierarchies are running out of steam
Deep, rigid hierarchies are running out of steam

Past

L1
~1ns

L2
~10ns

Main Memory
~100ns

Systems had few cache levels with widely different sizes and latencies
Deep, rigid hierarchies are running out of steam

**Past**

L1 \(\sim 1\) ns \(\rightarrow\) L2 \(\sim 10\) ns \(\rightarrow\) Main Memory \(\sim 100\) ns

**Now**

L1 \(\sim 1\) ns \(\rightarrow\) L2 \(\sim 5\) ns

Systems had few cache levels with widely different sizes and latencies
Deep, rigid hierarchies are running out of steam

Past

Systems had few cache levels with widely different sizes and latencies

Now

Distributed SRAM L3

~1ns ~5ns ~25ns
Deep, rigid hierarchies are running out of steam

**Past**

- L1 ~1ns
- L2 ~10ns
- Main Memory ~100ns

**Systems had few cache levels with widely different sizes and latencies**

**Now**

- L1 ~1ns
- L2 ~5ns
- Distributed SRAM L3 ~25ns
- Distributed DRAM L4 ~50ns
Deep, rigid hierarchies are running out of steam

**Past**

L1 \(\sim 1\)ns \(\rightarrow\) L2 \(\sim 10\)ns \(\rightarrow\) Main Memory \(\sim 100\)ns

**Systems had few cache levels with widely different sizes and latencies**

**Now**

L1 \(\sim 1\)ns \(\rightarrow\) L2 \(\sim 5\)ns

Distributed SRAM L3

\(\sim 25\)ns

Distributed DRAM L4

\(\sim 50\)ns \(\rightarrow\) Main Memory \(\sim 100\)ns
Deep, rigid hierarchies are running out of steam

Past

Systems had few cache levels with widely different sizes and latencies

Now

Higher overheads due to closer sizes and latencies across hierarchy levels
Rigid hierarchies must cater to the conflicting needs of many applications

App 1: Scan through a 256MB array repeatedly
Rigid hierarchies must cater to the conflicting needs of many applications.

App 1: Scan through a 256MB array repeatedly.
Rigid hierarchies must cater to the conflicting needs of many applications.

App 1: Scan through a 256MB array repeatedly
Rigid hierarchies must cater to the conflicting needs of many applications.

App 1: Scan through a 256MB array repeatedly

0% hit rate

0% hit rate

100% hit rate

Array data

Main Memory
Rigid hierarchies must cater to the conflicting needs of many applications.

App 1: Scan through a 256MB array repeatedly

Hit latency = ~5ns + ~25ns + ~50ns = ~80ns
Rigid hierarchies must cater to the conflicting needs of many applications.

App 1: Scan through a 256MB array repeatedly

Hit latency = \( \sim 5\text{ns} \) + \( \sim 20\text{ns} \) + \( \sim 50\text{ns} \) = \( \sim 80\text{ns} \)
Rigid hierarchies must cater to the conflicting needs of many applications.

App 1: Scan through a 256MB array repeatedly

- Hit latency = \(\sim 5\,\text{ns} + \sim 25\,\text{ns} + \sim 50\,\text{ns} = \sim 80\,\text{ns}\)
- Hit latency = \(\sim 5\,\text{ns} + 0\,\text{ns} + \sim 50\,\text{ns} = \sim 55\,\text{ns} (30\% \text{ lower})\)
Rigid hierarchies must cater to the conflicting needs of many applications

App 1: Scan through a 256MB array repeatedly

Hit latency = \( \sim 5\text{ns} \) + \( \sim 5\text{ns} \) + \( \sim 50\text{ns} \) = \( \sim 80\text{ns} \)

Hit latency = \( \sim 5\text{ns} \) + 0ns + \( \sim 50\text{ns} \) = \( \sim 55\text{ns} \) (30% lower)
Rigid hierarchies must cater to the conflicting needs of many applications.

App 1: Scan through a 256MB array repeatedly

Hit latency = \(~5\text{ns}\) + \(~25\text{ns}\) + \(~50\text{ns}\) = \(~80\text{ns}\)

Hit latency = \(~5\text{ns}\) + \(0\text{ns}\) + \(~50\text{ns}\) = \(~55\text{ns}\) (30% lower)
Rigid hierarchies must cater to the conflicting needs of many applications

App 1: Scan through a 256MB array repeatedly

<table>
<thead>
<tr>
<th></th>
<th>SRAM L3</th>
<th>DRAM L4</th>
<th>Main Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hit latency</strong></td>
<td>~5ns</td>
<td>~50ns</td>
<td></td>
</tr>
<tr>
<td><strong>0% hit rate</strong></td>
<td>~25ns</td>
<td>~5ns</td>
<td>~50ns</td>
</tr>
<tr>
<td><strong>100% hit rate</strong></td>
<td>0ns</td>
<td>~5ns</td>
<td>~40ns</td>
</tr>
</tbody>
</table>

- Hit latency = ~5ns + ~25ns + ~50ns = ~80ns
- Hit latency = ~5ns + 0ns + ~50ns = ~55ns (30% lower)
- Hit latency = ~5ns + 0ns + ~40ns = ~45ns (45% lower)
Rigid hierarchies must cater to the conflicting needs of many applications.

Even the best rigid hierarchy is a bad compromise!

(See paper for details)
Jenga: Software-defined cache hierarchies
Jenga: Software-defined cache hierarchies

Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.
Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.

App 1: Scan through a 256MB array

Ideal hierarchy

App 1 → Private L1 & L2 → 256MB cache → Main Memory

DRAM bank

SRAM bank
Jenga: Software-defined cache hierarchies

Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.

App 1: Scan through a 256MB array

Ideal hierarchy

App 1 \[\rightarrow\] Private L1 & L2 \[\rightarrow\] 256MB cache \[\rightarrow\] Main Memory

- DRAM bank
- SRAM bank
Jenga: Software-defined cache hierarchies

Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.

**App 1:** Scan through a 256MB array

**Ideal hierarchy**

App 1 → Private L1 & L2 → 256MB cache → Main Memory

**App 2:** Lookup a 5MB hashmap

**Ideal hierarchy**

App 2 → Private L1 & L2 → 5MB cache → Main Memory

**DRAM bank**

**SRAM bank**
Jenga manages distributed and heterogeneous banks as a single resource pool and builds *virtual hierarchies* tailored to each application in the system.

**App 1: Scan through a 256MB array**

Ideal hierarchy

```
App 1 → Private L1 & L2 → 256MB cache → Main Memory
```

**App 2: Lookup a 5MB hashmap**

Ideal hierarchy

```
App 2 → Private L1 & L2 → 5MB cache → Main Memory
```
Jenga: Software-defined cache hierarchies

Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.
Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.

**App 3**: Scan through two arrays (1MB and 256MB)

- **1MB cache**
- **256MB cache**
- **Private L1 & L2**
- **DRAM bank**
- **SRAM bank**
Jenga: Software-defined cache hierarchies

Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.

App 3: Scan through two arrays (1MB and 256MB)

App 3

1MB cache

256MB cache

SRAM bank

DRAM bank
Prior work to mitigate the cost of rigid hierarchies
Prior work to mitigate the cost of rigid hierarchies

- Bypass levels to avoid cache pollutions
  - Do not install lines at specific levels
  - Give lines low priority in replacement policy
Prior work to mitigate the cost of rigid hierarchies

- Bypass levels to avoid cache pollutions
  - Do not install lines at specific levels
  - Give lines low priority in replacement policy

- Speculatively access up the hierarchy
  - Hit/miss predictors, prefetchers
  - Hide latency with speculative accesses
Prior work to mitigate the cost of rigid hierarchies

- Bypass levels to avoid cache pollutions
  - Do not install lines at specific levels
  - Give lines low priority in replacement policy

- Speculatively access up the hierarchy
  - Hit/miss predictors, prefetchers
  - Hide latency with speculative accesses

- They **must** still check all levels for correctness!
  - Waste energy and bandwidth
Prior work to mitigate the cost of rigid hierarchies

- Bypass levels to avoid cache pollutions
  - Do not install lines at specific levels
  - Give lines low priority in replacement policy

- Speculatively access up the hierarchy
  - Hit/miss predictors, prefetchers
  - Hide latency with speculative accesses

- It’s better to build the right hierarchy and avoid the root cause: unnecessary accesses to unwanted cache levels

- They must still check all levels for correctness!
  - Waste energy and bandwidth
Jenga = flexible hardware + smart software
Jenga = flexible hardware + smart software

Software

Hardware

Time
Jenga = flexible hardware + smart software
Jenga = flexible hardware + smart software
Jenga = flexible hardware + smart software
Jenga = flexible hardware + smart software
Jenga = flexible hardware + smart software

- Hardware
- Software

Optimize hierarchies

Read hardware monitors

Update hierarchies

Time

100ms
Jenga hardware: supporting virtual hierarchies (VHs)

- Cores consult **virtual hierarchy table (VHT)** to find the access path
  - Similar to Jigsaw [PACT’13, HPCA’15], but it supports two levels
Cores consult **virtual hierarchy table (VHT)** to find the access path.

Similar to Jigsaw [PACT’13, HPCA’15], but it supports two levels.
Jenga hardware: supporting virtual hierarchies (VHs)

- Cores consult **virtual hierarchy table (VHT)** to find the access path
  - Similar to Jigsaw [PACT’13, HPCA’15], but it supports two levels

![Diagram showing Jenga hardware components]

- SRAM Bank
- NoC Router
- Core: TLB, Private $, VHT
- DRAM bank
- Two-level using both SRAM and DRAM
Cores consult **virtual hierarchy table (VHT)** to find the access path

- Similar to Jigsaw [PACT’13, HPCA’15], but it supports two levels

Jenga hardware: supporting virtual hierarchies (VHs)
Accessing a two-level virtual hierarchy

Access path: SRAM bank → DRAM bank → Mem
Accessing a two-level virtual hierarchy

Access path: **SRAM bank** $\rightarrow$ **DRAM bank** $\rightarrow$ Mem

**Tile 10**

**Core 1**

**SRAM (bank 10)**

**Virtual L1 (VL1)**

**VHT**

**Private Caches**

**DRAM cache bank**

**Access path:**
- Core miss $\rightarrow$ VL1 bank
- SRAM bank $\rightarrow$ DRAM bank $\rightarrow$ Mem
Accessing a two-level virtual hierarchy

Access path: SRAM bank → DRAM bank → Mem

1. Core miss → VL1 bank
2. VL1 miss → VL2 bank
Accessing a two-level virtual hierarchy

Access path: SRAM bank → DRAM bank → Mem

Tile 10
Virtual L1 (VL1)
SRAM (bank 10)
Core miss → VL1 bank

Virtual L2 (VL2)
DRAM (bank 38)
VL1 miss → VL2 bank

Tile 10
Core 1
VHT
Private Caches

Core 1
Tile

Access path:
1. Core miss → VL1 bank
2. VL1 miss → VL2 bank
3. VL2 hit, serve line
Accessing an single-level VH using SRAM + DRAM

- With VHT, software can group any combinations of banks to form a VH
Accessing an single-level VH using SRAM + DRAM

- With VHT, software can group any combinations of banks to form a VH

![Diagram showing memory organization with Core, Private Caches, VHT, Main Memory, and Single-level using both SRAM and DRAM]
Accessing an single-level VH using SRAM + DRAM

- With VHT, software can group any combinations of banks to form a VH.
With VHT, software can group any combinations of banks to form a VH.
Accessing an single-level VH using SRAM + DRAM

With VHT, software can group any combinations of banks to form a VH.
Periodically, Jenga reconfigures VHs to minimize data movement.
Periodically, Jenga reconfigures VHs to minimize data movement.
Periodically, Jenga reconfigures VHs to minimize data movement
Periodically, Jenga reconfigures VHs to minimize data movement.
Jenga software: finding near-optimal hierarchies

- Periodically, Jenga reconfigures VHs to minimize data movement.
Periodically, Jenga reconfigures VHs to minimize data movement.
Modeling performance of heterogeneous caches

- Treat SRAM and DRAM as different “flavors” of banks with different latencies
Modeling performance of heterogeneous caches

- Treat SRAM and DRAM as different “flavors” of banks with different latencies
Modeling performance of heterogeneous caches

- Treat SRAM and DRAM as different “flavors” of banks with different latencies.
Modeling performance of heterogeneous caches

- Treat SRAM and DRAM as different “flavors” of banks with different latencies
Treat SRAM and DRAM as different “flavors” of banks with different latencies
Modeling performance of heterogeneous caches

- Treat SRAM and DRAM as different “flavors” of banks with different latencies

---

<table>
<thead>
<tr>
<th>DRAM bank</th>
<th>Start</th>
<th>Color → latency</th>
</tr>
</thead>
</table>

---

- Miss curve from hardware monitors

---

<table>
<thead>
<tr>
<th>Cache</th>
<th>Access Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Access latency</th>
<th>Miss latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

13
Modeling performance of heterogeneous caches

- Treat SRAM and DRAM as different “flavors” of banks with different latencies.

---

Latency curve for single-level, heterogeneous cache

Miss curve from hardware monitors

Access latency
Miss latency
Total latency

Start Color ➔ latency

Virtual Cache size
Optimizing hierarchies by minimizing system latency
Optimizing hierarchies by minimizing system latency

- Our prior work has proposed algorithms to take latency curves, allocate capacity and place them on chip to minimize system latency.
- But only builds *single-level VHs*. 
Optimizing hierarchies by minimizing system latency

- Our prior work has proposed algorithms to take latency curves, allocate capacity and place them on chip to minimize system latency
- But only builds single-level VHs
Optimizing hierarchies by minimizing system latency

- Our prior work has proposed algorithms to take latency curves, allocate capacity and place them on chip to minimize system latency.
  - But only builds single-level VHs.
Optimizing hierarchies by minimizing system latency

- Our prior work has proposed algorithms to take latency curves, allocate capacity and place them on chip to minimize system latency
  - But only builds single-level VHs
Multi-level hierarchies are much more complex
Multi-level hierarchies are much more complex

- Many intertwined factors
  - Best VL1 size depends on VL2 size
  - Best VL2 size depends on VL1 size
  - Should we have VL2? (Depends on total size)
Multi-level hierarchies are much more complex

- Many intertwined factors
  - Best VL1 size depends on VL2 size
  - Best VL2 size depends on VL1 size
  - Should we have VL2? (Depends on total size)

- Jenga encodes these tradeoffs in a single curve
  - Can reuse prior allocation algorithms
How to get a latency curve for a multi-level VH
How to get a latency curve for a multi-level VH

Two-level hierarchies form a latency surface!
How to get a latency curve for a multi-level VH

Two-level hierarchies form a latency surface!

Best 1- and 2-level hierarchy at every size
How to get a latency curve for a multi-level VH

Two-level hierarchies form a latency *surface*!

![Graph showing latency surface for one-level and two-level hierarchies](image)

- Best 1- and 2-level hierarchy at every size
How to get a latency curve for a multi-level VH

Two-level hierarchies form a latency surface!

Best 1- and 2-level hierarchy at every size

Best overall hierarchy at every size
How to get a latency curve for a multi-level VH

Two-level hierarchies form a latency surface!

- Best 1- and 2-level hierarchy at every size
- Best overall hierarchy at every size
How to get a latency curve for a multi-level VH

Two-level hierarchies form a latency surface!

Best 1- and 2-level hierarchy at every size

Best overall hierarchy at every size

Curve lets us optimize multi-level hierarchies!
Allocating virtual hierarchies

Latency curves

VH1

VH2

VH3
Allocating virtual hierarchies

Latency curves

VH1

VH2

VH3

Cache allocation algorithm
Allocating virtual hierarchies

Latency curves

Total capacity of each VH

Cache allocation algorithm

VH1

VH2

VH3
Allocating virtual hierarchies

Latency curves

Total capacity of each VH

Cache allocation algorithm

Decide the best hierarchy

VH1

VH2

VH3
Allocating virtual hierarchies

Latency curves

Total capacity of each VH

Cache allocation algorithm

Capacity

Decide the best hierarchy

Virtual hierarchy size and levels
Bandwidth-aware virtual hierarchy placement

DRAM bank

SRAM bank

VL1

VL1

VL1

VL2
Bandwidth-aware virtual hierarchy placement

Place data close without saturating DRAM bandwidth
Bandwidth-aware virtual hierarchy placement

- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga …
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency
Bandwidth-aware virtual hierarchy placement

- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga ...
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency
Bandwidth-aware virtual hierarchy placement

- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga ...
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency

![Diagram of virtual hierarchy placement](image)
Bandwidth-aware virtual hierarchy placement

- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga ...
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency

1.0X Latency

1.0X Latency
Bandwidth-aware virtual hierarchy placement

- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga ...
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency
Bandwidth-aware virtual hierarchy placement

- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga ...
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency

![Diagram showing VH placement and latency comparisons]

1.0X Latency

1.1X Latency

1.3X Latency
Bandwidth-aware virtual hierarchy placement

- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga ...
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency

![Diagram showing virtual hierarchy placement and latency comparisons.]
Bandwidth-aware virtual hierarchy placement

- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga ...
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency
Bandwidth-aware virtual hierarchy placement

- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga...
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency

![Diagram showing virtual hierarchy placement with latency levels](image)
Bandwidth-aware virtual hierarchy placement

- **Place data close without saturating DRAM bandwidth**

- **Every iteration, Jenga ...**
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency
Jenga adds small overheads
Jenga adds small overheads

- **Hardware overheads**
  - VHT requires \( \sim 2.4 \) KB/tile
  - Monitors are 8 KB x 2/tile
  - In total, Jenga adds \( \sim 20 \) KB per tile, 4% of the SRAM banks
  - Similar to Jigsaw
Jenga adds small overheads

- **Hardware overheads**
  - VHT requires \( \sim 2.4 \) KB/tile
  - Monitors are 8 KB x 2/tile
  - In total, Jenga adds \( \sim 20 \) KB per tile, 4\% of the SRAM banks
  - Similar to Jigsaw

- **Software overheads**
  - 0.4\% of system cycles at 36 tiles
  - Runs concurrently with applications; only needs to pause cores to update VHTs
  - Trivial to parallelize
See paper for ...

- Hardware support for
  - Fast reconfiguration
  - Page reclassification

- Efficient implementation of hierarchy allocation

- OS integration
Evaluation
Evaluation

- Modeled system
  - 36 cores on 6x6 mesh
  - 18MB SRAM
  - 1GB Stacked DRAM
Evaluation

- Modeled system
  - 36 cores on 6x6 mesh
  - 18MB SRAM
  - 1GB Stacked DRAM

- Workloads
  - 36 copies of same app (SPECrate)
  - Random 36 SPECCPU apps mixes
Evaluation

- Modeled system
  - 36 cores on 6x6 mesh
  - 18MB SRAM
  - 1GB Stacked DRAM

- Workloads
  - 36 copies of same app (SPECrate)
  - Random 36 SPECCPU apps mixes

- Compared 5 schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>SRAM</th>
<th>DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-NUCA</td>
<td>Rigid L3</td>
<td>-</td>
</tr>
<tr>
<td>Alloy</td>
<td>Rigid L3</td>
<td>Rigid L4</td>
</tr>
<tr>
<td>Jigsaw</td>
<td>App-specific L3</td>
<td>-</td>
</tr>
<tr>
<td>JigAlloy</td>
<td>App-specific L3</td>
<td>Rigid L4</td>
</tr>
<tr>
<td>Jenga</td>
<td>App-specific Virtual Hierarchies</td>
<td></td>
</tr>
</tbody>
</table>
Case study: 36 copies of xalanc
Case study: 36 copies of xalanc

Working set: 6MB \times 36 = 216 \text{ MB}
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

S-NUCA

Private L2

Rigid SRAM L3

Data
Case study: 36 copies of xalanc

Working set: $6\text{MB} \times 36 = 216\text{ MB}$

- S-NUCA
- Private L2
- Rigid SRAM L3
- Data
- Memory

~100% miss rate
Case study: 36 copies of xalanc

Working set: $6\text{MB} \times 36 = 216\text{ MB}$

Wasteful accesses to L3, should have gone to memory directly

~100% miss rate
Case study: 36 copies of xalanc

Working set: $6\text{MB} \times 36 = 216 \text{ MB}$

- Wasteful accesses to L3, should have gone to memory directly
- $\sim 100\%$ miss rate
Case study: 36 copies of xalanc

Working set: \(6\text{MB} \times 36 = 216\text{ MB}\)

EDP improv. vs. S-NUCA

Speedup vs. S-NUCA
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

Private L2

Diagram showing comparisons of different benchmarks (S-NUCA, Alloy, Jigsaw, JigAlloy, Jenga) with labels for EDP improvement and speedup vs. S-NUCA.
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

~100% miss rate
Case study: 36 copies of xalanc

Working set: $6MB \times 36 = 216$ MB

- Memory
  - Rigid DRAM L4
  - ~0% miss rate

- Private L2
  - Rigid SRAM L3
  - ~100% miss rate
  - ~100% miss rate

EDP improv. vs. S-NUCA
- xalanc
  - Speedup vs. S-NUCA
  - 2.0

- Alloy
  - Jigsaw
  - JigAlloy
  - Jenga
Case study: 36 copies of xalanc

Working set: $6\text{MB} \times 36 = 216\text{ MB}$

- Memory:
  - Rigid DRAM L4
  - ~0% miss rate

- Cache working sets with DRAM L4
  - Rigid SRAM L3
  - ~100% miss rate

- Private L2

Graph:
- EDP improv. vs. S-NUCA
- Speedup vs. S-NUCA

- xalanc
- S-NUCA
- Alloy
- Jigsaw
- JigAlloy
- Jenga

Graph data:
- ~100% miss rate
- ~0% miss rate
Case study: 36 copies of xalanc

Working set: $6\text{MB} \times 36 = 216\text{ MB}$
Case study: 36 copies of xalanc

Working set: $6\text{MB} \times 36 = 216 \text{ MB}$

- Jigsaw
- Private L2
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

EDP improv. vs. S-NUCA

Speedup vs. S-NUCA

Private L2

App-specific SRAM L3
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

~90% miss rate

App-specific SRAM L3

Memory

EDP improv. vs. S-NUCA

Speedup vs. S-NUCA
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

Reduce 10% misses with app-specific SRAM L3

～90% miss rate

Memory

App-specific SRAM L3

Private L2

Jigsaw

S-NUCA

JigAlloy

Alloy

Jenga

EDP improv. vs. S-NUCA

Speedup vs. S-NUCA

xalanc

xalanc
Case study: 36 copies of xalanc

Working set: $6\text{MB} \times 36 = 216 \text{ MB}$
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

Private L2
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

EDP improv. vs. S-NUCA

Speedup vs. S-NUCA
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

App-specific SRAM L3

~90% miss rate

EDP improv. vs. S-NUCA

Speedup vs. S-NUCA
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

Private L2 → App-specific SRAM L3 → Rigid DRAM L4

~90% miss rate

EDP improv. vs. S-NUCA

Speedup vs. S-NUCA
Case study: 36 copies of xalanc

Working set: $6\text{MB} \times 36 = 216 \text{ MB}$

Memory

- Rigid DRAM L4
- App-specific SRAM L3
- JigAlloy
- Private L2

$\sim 90\%$ miss rate

$\sim 0\%$ miss rate

EDP improv. vs. S-NUCA

- S-NUCA
- Jigsaw
- JigAlloy
- Alloy
- Jenga

Speedup vs. S-NUCA

- xalanc

25
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

Combines Jigsaw’s and Alloy’s benefits, but still a rigid hierarchy

~0% miss rate

~90% miss rate

EDP improv. vs. S-NUCA

Speedup vs. S-NUCA

xalanc

xalanc
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

EDP improv. vs. S-NUCA

Speedup vs. S-NUCA
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

6MB, SRAM + DRAM
VL1-only hierarchy

Private L2

EDP improv. vs. S-NUCA
Speedup vs. S-NUCA
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

6MB, SRAM + DRAM
VL1-only hierarchy

~0% miss rate

Memory

<table>
<thead>
<tr>
<th>xalanc</th>
<th>S-NUCA</th>
<th>Alloy</th>
<th>Jenga</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jigsaw</td>
<td>JigAlloy</td>
<td>Jenga</td>
<td></td>
</tr>
</tbody>
</table>

EDP improv. vs. S-NUCA

Speedup vs. S-NUCA
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

Memory

Private L2

6MB, SRAM + DRAM
VL1-only hierarchy

Single lookup to the working set!
No wasteful lookups!

~0% miss rate

EDP improv. vs. S-NUCA

Speedup vs. S-NUCA

Jenga

S-NUCA

Alloy

Jigsaw

JigAlloy

Jenga

60% better

20% better
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

6MB, SRAM + DRAM
VL1-only hierarchy

Private L2

Single lookup to the working set!

No wasted lookups!

Working set: 6MB x 36 = 216 MB

Jenga improves performance and energy efficiency by creating the right hierarchy using the best available resources!
Jenga works across a wide range of behaviors

App with two-level working set

App with flat working set

EDP improv. vs. S-NUCA

S-NUCA  Alloy  Jigsaw  JigAlloy  Jenga

omnet  xalanc  leslie
Jenga works across a wide range of behaviors

Working set

- Jenga works across a wide range of behaviors
- Working set
- Jenga VHs

- App with two-level working set
  - astar: 0.5MB + 16MB
  - bzip2: 1MB + 8MB

- App with flat working set
  - omnet: S-NUCA
  - xalanc: S-NUCA
  - leslie: S-NUCA
Jenga works across a wide range of behaviors

<table>
<thead>
<tr>
<th>App with two-level working set</th>
<th>SRAM VL1</th>
<th>SRAM+DRAM VL1</th>
<th>DRAM VL2</th>
<th>DRAM VL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>astar</td>
<td></td>
<td></td>
<td>0.5MB +</td>
<td>0.5MB +</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>16MB</td>
<td>16MB</td>
</tr>
<tr>
<td>bzip2</td>
<td></td>
<td></td>
<td>1MB +</td>
<td>1MB +</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8MB</td>
<td>8MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>App with flat working set</th>
<th>SRAM+</th>
<th>SRAM+</th>
<th>DRAM VL1</th>
<th>DRAM VL1</th>
<th>No caching</th>
</tr>
</thead>
<tbody>
<tr>
<td>omnet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xalanc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>leslie</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.5MB</td>
<td>2.5MB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8MB</td>
<td>8MB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&gt;50MB</td>
<td>&gt;50MB</td>
<td></td>
</tr>
</tbody>
</table>
Jenga works for random multi-program mixes
Jenga works for random multi-program mixes

2.6X over S-NUCA

20% over JigAlloy
Jenga works for random multi-program mixes

**EDP improv. vs. S-NUCA**

- **2.6X** over S-NUCA
- **20%** over JigAlloy

**WSpeedup vs. S-NUCA**

- **1.7X** over S-NUCA
- **10%** over JigAlloy
Jenga works for random multi-program mixes

Jenga consistently outperforms the other schemes for multi-program mixes

- 2.6X over S-NUCA
- 1.7X over S-NUCA
- 20% over JigAlloy
- 10% over JigAlloy
See paper for more results

- Full result for SPECCPU-rate
- Multithreaded apps
- Sensitivity study for Jenga’s software techniques
- 2.5D DRAM architectures
- Jigsaw SRAM L3 + Jigsaw DRAM L4
- And more
Rigid, multi-level cache hierarchies are ill-suited to many applications

- They cause significant overhead when they are not helpful
Rigid, multi-level cache hierarchies are ill-suited to many applications
- They cause significant overhead when they are not helpful

We propose Jenga, a software-defined, reconfigurable cache hierarchy
- Adopts application-specific organization on-the-fly
- Uses new software algorithm to find near-optimal hierarchy efficiently
Rigid, multi-level cache hierarchies are ill-suited to many applications
- They cause significant overhead when they are not helpful

We propose Jenga, a software-defined, reconfigurable cache hierarchy
- Adopts application-specific organization on-the-fly
- Uses new software algorithm to find near-optimal hierarchy efficiently

Jenga improves both performance and energy efficiency, by up to 85% in EDP, over a combination of state-of-art techniques
Thanks! Questions?

- Rigid, multi-level cache hierarchies are ill-suited to many applications
  - They cause significant overhead when they are not helpful

- We propose Jenga, a software-defined, reconfigurable cache hierarchy
  - Adopts application-specific organization on-the-fly
  - Uses new software algorithm to find near-optimal hierarchy efficiently

- Jenga improves both performance and energy efficiency, by up to 85% in EDP, over a combination of state-of-art techniques
Jenga: Software-Defined Cache Hierarchies

Thank you for your attention!

Questions?