UNDERSUBSCRIBED THREADING ON CLUSTERED CACHE ARCHITECTURES

Wim Heirman\textsuperscript{1,2} Trevor E. Carlson\textsuperscript{1} Kenzo Van Craeynest\textsuperscript{1} Ibrahim Hur\textsuperscript{2} Aamer Jaleel\textsuperscript{2} Lieven Eeckhout\textsuperscript{1}

\textsuperscript{1} Ghent University \textsuperscript{2} Intel Corporation

HPCA 2014, Orlando, FL
CONTEXT

• Many-core processor with 10s-100s of cores
  – E.g. Intel Xeon Phi, Tilera, GPGPU

• Running scalable, data-parallel workloads
  – SPEC OMP, NAS Parallel Benchmarks, ...

• Processor design @ fixed area/power budget
  – Spend on cores, caches?
  – Cache topology?
OVERVIEW

• Cache topology:
  – Why clustered caches?
  – Why undersubscription?

• Dynamic undersubscription
  – CRUST algorithms for automatic adaptation

• CRUST and future many-core design
Many-core Cache Architectures

- **Private**
  - Core
  - Cache
  - ... (N)

- **Clustered**
  - Core
  - Core
  - ... (C)
  - Core
  - Core
  - ... (C)
  - Cache
  - Cache
  - ... (N/C)

- **Shared (NUCA)**
  - Core
  - Core
  - Core
  - Core
  - Core
  - Core
  - Cache
  - ... (N)
MANY-CORE CACHE ARCHITECTURES

private

hit latency

sharing

clustered

shared

+ +

x

+ +

+ +

+ +

+ +

+ +

+ +

+ +

+ +

+ +
UNDERSUBSCRIBING FOR CACHE CAPACITY
UNDERSUBSCRIBING FOR CACHE CAPACITY

• Less than C active cores/threads per cluster
UNDERSUBSCRIBING FOR CACHE CAPACITY

• Less than C active cores/threads per cluster
UNDERSUBSCRIBING FOR CACHE CAPACITY

• Less than C active cores/threads per cluster

3/4 undersubscription

Core  Core  Core  Core  
Cache

Core  Core  Core  Core  Core  Core
Cache  ... (N)
UNDERSUBSCRIBING FOR CACHE CAPACITY

- Less than \( C \) active cores/threads per cluster

2/4 undersubscription

Core Core Core Core

Core Core Core Core

Cache

Cache

... (N)
UNDERSUBSCRIBING FOR CACHE CAPACITY

• Less than C active cores/threads per cluster

1/4 undersubscription
Undersubscribing for Cache Capacity

- Less than C active cores/threads per cluster
- When working set does not fit in cache

1/4 undersubscription
UNDERSUBSCRIBING FOR CACHE CAPACITY

- Less than C active cores/threads per cluster
- When working set does not fit in cache
- *Keep all cache capacity accessible*

1/4 undersubscription
### Many-core Cache Architectures

<table>
<thead>
<tr>
<th>Type</th>
<th>Hit Latency</th>
<th>Sharing</th>
<th>Undersubscription</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private</td>
<td>++</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Clustered</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Shared</td>
<td>× ×</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

(1:1) for Private, (1:C) for Clustered, (1:N) for Shared
PERFORMANCE & ENERGY: WORKING SET VS. CACHE SIZE

Baseline architecture:
- 128 cores
- private L1
- clustered L2
  1M shared per 4 cores
Performance & Energy: Working Set vs. Cache Size

**Compute bound:** use all cores for highest performance

**Bandwidth bound:** disable cores for better energy efficiency

**Capacity bound:** reduce thread count to optimize hit rate

1/4 undersubscription: 3.5x performance, 80% energy savings
CLUSTER-aware UNDERSUBSCRIBED SCHEDULING OF THREADS (CRUST)

• Dynamic undersubscription
• Integrated into the OpenMP runtime library
  – Adapt to each `#pragma omp` individually
• Optimize for performance first, save energy when possible
  – Compute bound: full subscription
  – Bandwidth bound: no* performance degradation (* <5% vs. full)
  – Capacity bound: highest performance
• Two CRUST heuristics (descend and predict) for on-line adaptation
CRUST-DESCEND

- Start with full subscription
- Reduce thread count while performance increases
CRUST-PREDICT

• Reduce number of steps required by being smarter
• Start with heterogeneous undersubscription
  – Measure LLC miss rate for each thread/cluster option
  – Predict performance of each option using PIE-like model
• Select best predicted option
**Methodology**

- **Generic many-core architecture**
  - 128 cores, 2-issue OOO @1GHz
  - 2x 32 KB private L1 I+D
  - L1-D stride prefetcher
  - **1 MB shared L2 per 4 cores**
  - 2-D mesh NoC
  - 64 GB/s total DRAM bandwidth

- **Sniper simulator, McPAT for power**

- **SPEC OMP and NAS parallel benchmarks**
  - Reduced iteration counts from ref, class A inputs
RESULTS: ORACLE (STATIC)

capacity bound  bandwidth bound  compute bound

![Bar chart showing energy efficiency improvement over full utilization for various workloads. The chart includes bars for N-ft/A, N-sp/A, O-equake/ref, N-bt/A, N-is/A, N-lu/A, O-swim/ref, N-cg/A, N-ua/A, and Mean. The chart indicates percentage improvements with labels 138% and 166% for specific workloads.]

- Energy efficiency improvement over full utilization
- Workloads: N-ft/A, N-sp/A, O-equake/ref, N-bt/A, N-is/A, N-lu/A, O-swim/ref, N-cg/A, N-ua/A, Mean
- Improvement percentages: 138% and 166%
- Static oracle approach
RESULTS: LINEAR BANDWIDTH MODELS

Linear Bandwidth Models (e.g. BAT): save energy, does not exploit capacity effects on clustered caches.
RESULTS: CRUST

CRUST: save energy when bandwidth-bound, exploit capacity effects on clustered caches
UNDERSUBSCRIPTION VS. FUTURE DESIGNS

• Finite chip area, spent on cores or caches
  – Increasing max. compute vs. keeping cores fed with data
• Undersubscription can adapt workload behavior to the architecture

  Does this allow us to build a higher-performance design?

• Sweep core vs. cache area ratio for 14-nm design
  – Fixed 600 mm² area, core = 1.5 mm², L2 cache = 3 mm²/MB
  – Clustered L2 shared by 4 cores, latency ~ log2(size)
  – 1 GB @ 512 GB/s on-package, 64 GB/s off-package

<table>
<thead>
<tr>
<th>Variant</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>96</td>
<td>128</td>
<td>160</td>
<td>192</td>
<td>224</td>
<td>256</td>
</tr>
<tr>
<td>L2 size (MB/core)</td>
<td>1.5</td>
<td>1.0</td>
<td>0.8</td>
<td>0.5</td>
<td>0.4</td>
<td>0.3</td>
</tr>
<tr>
<td>Core area</td>
<td>25%</td>
<td>33%</td>
<td>40%</td>
<td>50%</td>
<td>58%</td>
<td>64%</td>
</tr>
</tbody>
</table>
**UNDERSUBSCRIPTION FOR FUTURE DESIGNS**

**Compute bound**: linear relation between active cores and performance

**Capacity bound**: reduce thread count until combined working set fits available cache

![Graph showing relative performance comparison](image)

N-ft/C
**Undersubscription for Future Designs**

- Build *one* design with best average performance
UNDERSUBSCRIPTION FOR FUTURE DESIGNS

- Build *one* design with best average performance
- Full subscription:
  - conservative option C has highest average performance

![Diagram showing relative performance for designs A to F with full subscription at 1.24]
**Undersubscription for Future Designs**

- Build *one* design with best average performance
- Full subscription:
  - conservative option C has highest average performance
- Dynamic undersubscription: prefer more cores
  - higher max. performance for compute-bound benchmarks
  - use undersubscription to accommodate capacity-bound workloads

![Relative Performance Graph](image)

- **Relative Performance**
  - A: 0.0
  - B: 1.0
  - C: 1.24
  - D: 1.54
  - E: 1.54
  - F: 2.0

- **Legend**
  - Full Subscription
  - Dynamic Subscription
**Undersubscription for Future Designs**

- **Build one design with best average performance**
- **Full subscription:**
  - conservative option C has highest average performance
- **Dynamic undersubscription: prefer more cores**
  - higher max. performance for compute-bound benchmarks
  - use undersubscription to accommodate capacity needs

**Graph**:
- Relative performance
- **full subscription**
- **dynamic subscription**

**E vs. C:**
- 40% more cores
- 15% higher performance
CONCLUSIONS

• Use clustered caches for future many-core designs
  – Balance hit rate and hit latency
  – Exploit sharing to avoid duplication
  – Allow for undersubscription (use all cache, not all cores)

• CRUST for dynamic undersubscription
  – Adapt thread count per OpenMP parallel section
  – Performance and energy improvements of up to 50%

• Take undersubscription usage model into account when designing future many-core processors
  – CRUST-aware design: 40% more cores, 15% higher performance