Leveraging Hardware Address Sampling
Beyond Data Collection and Attribution

Xu Liu

Department of Computer Science
College of William and Mary
xl10@cs.wm.edu
Motivation: Memory is the Bottleneck

NUMA: Non-Uniform Memory Access

- **Core**
- **Cache**
- **Memory**
- **QuickPath**
- **HyperTransport**

**Local access**

**Remote access**
Memory Bottleneck Optimization

spatial locality

temporal locality

NUMA locality

cache miss
State of the Arts

- Simulation methods
  - Deep insights
  - Weaknesses:
    - 2-5x overhead
    - Not real machines

  - Low overhead with deep insights

- Measurement methods
  - Low overhead
  - Deep insights with low overhead
Hardware Address Sampling

• **Features of address sampling**
  – **necessary features**
    • sample memory-related events (memory accesses, NUMA events)
    • capture effective addresses
    • record precise IP of sampled instructions or events
  – **optional features**
    • record useful metrics: data access latency (in CPU cycle)
    • sample instructions/events not related to memory

• **Support in modern processors**
  • AMD Opteron 10h and above: instruction-based sampling (IBS)
  • IBM POWER 5 and above: marked event sampling (MRK)
  • Intel Itanium 2: data event address register sampling (DEAR)
  • Intel Pentium 4 and above: precise event based sampling (PEBS)
  • Intel Nehalem and above: PEBS with load latency (PEBS-LL)
Tools Based on Address Sampling

• Measurement methods
  – temporal/spatial locality
    • HPCToolkit, Cache Scope
  – NUMA locality
    • Memphis, MemProf, HPCToolkit

• Features
  – lightweight performance data collection
  – efficient performance data attribution
    • code-centric attribution
    • data-centric attribution

Take HPCToolkit for example
“A Data-centric Profiler for Parallel Programs”. Liu and Mellor-Crummey, SC’13
HPCToolkit: Attributing Samples

- heap allocated variables
- static variable range
- allocation path
- malloc
- data-centric attribution
- variable name
- 0x0
- 0xff
- code-centric attribution
HPCToolkit: Aggregating Profiles

heap allocated variables

allocation path

merge
LULESH on Platform of 8 NUMA Domains

- Call paths for accesses
- Call site of allocation
- Allocation call path
- Heap data: 68% remote accesses
- Z accounts for 7.7% remote accesses
- Z is allocated in a NUMA domain but accessed by others
- Interleave pages of z across NUMA nodes
- 13% improvement in running time
Existing Measurement is Inadequate

• Data collection + attribution ≠ optimal optimization
  – know problematic data objects but not know why
  – need more insights for optimization guidance
  – challenges in data analysis
    • not monitoring continuous memory accesses

• Approaches: data analysis for detailed optimization guidance
  – NUMA locality
    • offline optimization (PPoPP’14)
    • online optimization
  – cache locality
    • array regrouping (PACT’14)
    • structure splitting
    • locality optimization between SMT threads
  – scalability of memory accesses
Interleaved Allocation is NOT Always Best

Centralized allocation: poor
Interleaved allocation: sub-optimal
Co-locate data with computation: optimal

Goal: identify the best data distribution for a program
Memory Access Pattern Analysis

- Online data collection
  
  - Offline analysis
    - merge \([\text{min}, \text{max}]\) intervals along call paths
    - plot \([\text{min}, \text{max}]\) for each thread

  - can be for any context, any variable

  
  
  
  allocate \(A\) blockwise to different domains

  \[
  \begin{align*}
  \text{array } A \\
  0x00 & \quad 0xff \\
  \text{min} & \quad \text{max}
  \end{align*}
  \]

  \([\text{min}, \text{max}]\) per sampled memory access

  
  balanced allocation + maximum locality
Pinpointing First Touch

- Linux “first touch” policy
  - memory allocation at first touch
  - if T1 first touches the whole range of A
  - if threads touch different segments of A
LULESH on Platform of 8 NUMA Domains

Block-wise allocation: 25% faster running time
Interleaved allocation: 13% faster running time

z accounts for 7.7% of remote accesses

Source code:

```c
for( Index_t lnode=0 ; lnode<8 ; ++lnode )
{
    Index_t gnode = nodelist[ElemId][lnode];
    ElemPos[X_Dir][lnode] = x[gnode];
    ElemPos[Y_Dir][lnode] = y[gnode];
    ElemPos[Z_Dir][lnode] = z[gnode];
}
```
## Architectures

<table>
<thead>
<tr>
<th>Sampling mechanisms</th>
<th>Processors</th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction-based sampling</td>
<td>IBS</td>
<td></td>
</tr>
<tr>
<td>Marked event sampling</td>
<td>MRK</td>
<td></td>
</tr>
<tr>
<td>Precise event-based sampling</td>
<td>PEBS</td>
<td></td>
</tr>
<tr>
<td>Data event address registers</td>
<td>DEAR</td>
<td></td>
</tr>
<tr>
<td>PEBS with load latency</td>
<td>PEBS-LL</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processors</th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Magny-Cours</td>
<td>48</td>
</tr>
<tr>
<td>IBM POWER 7</td>
<td>128</td>
</tr>
<tr>
<td>Intel Xeon Harpertown</td>
<td>8</td>
</tr>
<tr>
<td>Intel Itanium 2</td>
<td>8</td>
</tr>
<tr>
<td>Intel Ivy Bridge</td>
<td>8</td>
</tr>
</tbody>
</table>

## Benchmarks

<table>
<thead>
<tr>
<th>LLNL</th>
<th>LANL</th>
<th>Rodinia</th>
<th>PARSEC</th>
<th>SNL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMG2006</td>
<td>Sweep3D</td>
<td>Streamcluster</td>
<td>Blackscholes</td>
<td>S3D</td>
</tr>
<tr>
<td>LULESH</td>
<td></td>
<td>NW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sphot</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMT2013</td>
<td></td>
<td></td>
<td>optimized benchmarks</td>
<td></td>
</tr>
</tbody>
</table>
# Optimization Results

<table>
<thead>
<tr>
<th>Programs</th>
<th>Optimization</th>
<th>Improvement for execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMG2006</td>
<td>NUMA locality</td>
<td>51% for the solver</td>
</tr>
<tr>
<td>Sweep3D</td>
<td>spatial locality</td>
<td>15%</td>
</tr>
<tr>
<td>LULESH</td>
<td>spatial+NUMA locality</td>
<td>25%</td>
</tr>
<tr>
<td>Streamcluster</td>
<td>NUMA locality</td>
<td>28%</td>
</tr>
<tr>
<td>NW</td>
<td>NUMA locality</td>
<td>53%</td>
</tr>
<tr>
<td>UMT2013</td>
<td>NUMA locality</td>
<td>7%</td>
</tr>
</tbody>
</table>
# Measurement Overhead

## Code- & data-centric analysis on POWER7 and Opteron

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Configuration</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMG2006</td>
<td>4 MPI * 128 threads</td>
<td>604s (+9.6%)</td>
</tr>
<tr>
<td>Sweep3D</td>
<td>48 MPI</td>
<td>90s (+2.3%)</td>
</tr>
<tr>
<td>LULESH</td>
<td>48 threads</td>
<td>19s (+12%)</td>
</tr>
<tr>
<td>Streamcluster</td>
<td>128 threads</td>
<td>27s (+8.0%)</td>
</tr>
<tr>
<td>NW</td>
<td>128 threads</td>
<td>80s (+3.9%)</td>
</tr>
</tbody>
</table>

## NUMA analysis: code-, data-, and address-centric analysis + first touch

<table>
<thead>
<tr>
<th>Methods</th>
<th>LULESH</th>
<th>AMG2006</th>
<th>Blacksholes</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBS</td>
<td>295s (+24%)</td>
<td>89 (+37%)</td>
<td>192s (+6%)</td>
</tr>
<tr>
<td>MRK</td>
<td>93s (+5%)</td>
<td>27s (+7%)</td>
<td>132s (+4%)</td>
</tr>
<tr>
<td>PEBS</td>
<td>65s (+45%)</td>
<td>96s (+52%)</td>
<td>82s (+25%)</td>
</tr>
<tr>
<td>DEAR</td>
<td>90s (+7%)</td>
<td>120s (+12%)</td>
<td>73s (+4%)</td>
</tr>
<tr>
<td>PEBS-LL</td>
<td>35s (+6%)</td>
<td>57s (+8%)</td>
<td>67s (+3%)</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

• Hardware address sampling
  – widely supported in modern architectures
  – powerful in monitoring memory behaviors
  – currently in early stage of studies
    • focusing on data collection and attribution

• Potentials of hardware address sampling
  – provide deeper insights than traditional performance counters
  – require novel analysis methods to expose performance insights

• Future work
  – integrating address sampling into Charm++ runtime for online optimization