Run-Time Monitor (RTM)

Mikyung Kang, Inkeun Cho, Dong-In Kang, Stephen Crago, Spundun Bhatt

USC/ISI

Inseok Choi, Xu Yang, Meng-Ju Wu, Donald Yeung

University of Maryland at College Park

July 20, 2009
-UNCLASSIFIED-

Outline

- System framework
- Software performance monitor
- High level abstraction of data
- Accessing PMCs on Maestro
System Framework

- Parallel Performance Analysis Tool
- Run-Time System
- Run-Time Monitor
- Software
  - Instrumented Library (MPI, pthread, iLib)
- Hardware
  - PAPI/Perf mon2
- Maestro
Outline

- System framework
- Software performance monitor
- High level abstraction of data
- Accessing PMCs on Maestro
Software Performance Monitor

**What**

<table>
<thead>
<tr>
<th>Software</th>
<th>Message Passing Model</th>
<th>Shared Memory Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Communication Pattern&lt;br&gt;- Dependency&lt;br&gt;- Synchronization</td>
<td>- Synchronization&lt;br&gt;- Ownership of data region&lt;br&gt;- Limited info. of data access pattern</td>
</tr>
<tr>
<td></td>
<td>- By instrumenting iLib&lt;br&gt;- By instrumenting MPI</td>
<td>- By instrumenting pthread&lt;br&gt;- By instrumenting TMC&lt;br&gt;- By instrumenting iLib</td>
</tr>
<tr>
<td>Hardware</td>
<td>- Network stall, memory stall&lt;br&gt;- Cache behavior, other architectural information</td>
<td>- Through perfmon2/PAPI</td>
</tr>
</tbody>
</table>

**How**

- Periodic collection of information
- Stand-alone tool implementation
  - RTM Daemon Server: initialization and periodic system monitoring
  - RTM Daemon Client: allocating shared memory and collecting local information
## Run-Time Monitor Status

<table>
<thead>
<tr>
<th></th>
<th>iLib</th>
<th>pthread</th>
<th>MPI</th>
<th>TMC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version</strong></td>
<td>MDE 2.0</td>
<td>MDE 2.0</td>
<td>MDE 2.0</td>
<td>MDE 2.0</td>
</tr>
<tr>
<td><strong>How</strong></td>
<td>By instrumenting iLib</td>
<td>By instrumenting pthread</td>
<td>By instrumenting MPI</td>
<td>By instrumenting tmc</td>
</tr>
<tr>
<td></td>
<td>#pragma weak</td>
<td>#pragma weak</td>
<td>#pragma weak</td>
<td>#pragma weak</td>
</tr>
<tr>
<td><strong>Instrumented Functions</strong></td>
<td>ilib_msg_send ilib_msg_receive ilib_msg_start_send ilib_msg_start_receive ilib_msg_barrier …</td>
<td>pthread_mutex_lock pthread_mutex_unlock pthread_cond_wait pthread_cond_signal …</td>
<td>MPI_Send/Recv MPI_Barrier MPI_Bcast MPI_Reduce MPI_Alltoall …</td>
<td>tmc_sync_mutex_lock tmc_sync_mutex_unlock tmc_sync_barrier_wait tmc_spin_mutex_lock tmc_spin_mutex_unlock tmc_spin_barrier_wait …</td>
</tr>
<tr>
<td><strong>Provided Monitoring Information</strong></td>
<td>Communication Pattern Synchronization H/W information</td>
<td>Synchronization H/W information</td>
<td>Communication Pattern Synchronization H/W information</td>
<td>Synchronization H/W information</td>
</tr>
<tr>
<td></td>
<td>1. data-flow graph 2.synch graph</td>
<td>1. synch graph</td>
<td>1.data-flow graph 2.synch graph</td>
<td>1. synch graph</td>
</tr>
<tr>
<td><strong>Status</strong></td>
<td>RTM-iLib Implementation (Done) Instrumentation (In progress)</td>
<td>RTM-pthread Implementation (Debug) Instrumentation (In progress)</td>
<td>RTM-MPI Implementation (Done) Instrumentation (In progress)</td>
<td>RTM-TMC Implementation (Debug) Instrumentation (In progress)</td>
</tr>
</tbody>
</table>
Run-Time Monitor Stand-alone Model

**RTM (Run-time Monitor) Server**

1. **Daemon for client initialization**
   - `listen(); While(1) {accept the client’s request(); read mem_addr; write it to msg_queue}`

2. **Collecting H/W and S/W information**
   - Periodically:
     - `get mem_addr from msg_queue; list update(mem_addr); collect info (S/W & H/W);`

3. **Generating statistical information**

4. **Message Queue**
   - (background processes)

**RTM Client**

1. **Daemon for client initialization**
   - Set the Shared Memory Region using `shmget(key)``; Send the mem_addr to Server using socket;

2. **Updating H/W and S/W information**
   - Periodically:
     - `Update H/W info; Write it to shared memory;`

3. **Generating statistical information**

4. **Message Queue**

5. **Collecting H/W and S/W information**
   - While(1)
     - `if (Event Occur in App) Write Log to shared memory;`
- UNCLASSIFIED -

Outline

- System framework
- Software performance monitor
- High level abstraction of data
- Accessing PMCs on Maestro
## Message Passing Model

### Tracing information collected at software level

<table>
<thead>
<tr>
<th>Data</th>
<th>Event [communication]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Source, Destination) ranks and tile location</td>
<td>transferred data amount, timestamp for each event</td>
</tr>
<tr>
<td>Period</td>
<td>Saving on each tile: whenever event happens</td>
</tr>
<tr>
<td></td>
<td>Collecting by monitoring tile: defined interval</td>
</tr>
<tr>
<td>Output</td>
<td>Statistical result for each (source, destination) pair with timestamp</td>
</tr>
<tr>
<td></td>
<td>Statistical result for each process</td>
</tr>
<tr>
<td></td>
<td>xml file → task graph : task dependency/precedence, load</td>
</tr>
</tbody>
</table>

### Tracing information collected at hardware level

<table>
<thead>
<tr>
<th>Data</th>
<th>CYC, MP_BUNDLE_RETIRE, HIT, L2_HIT, TLB_HIT, TLB_EXC, MISS_I, MISS_D_RD, MISS_D_WR, MP_DATA_CACHESTALL, MP_INSTRUCTION_CACHESTALL, MP_ICACHE_HIT_ISSUED, MP_CONDITIONAL_BRANCH_MISSPREDICT, MP_CONDITIONAL_BRANCH_ISSUED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period</td>
<td>Saving on each tile</td>
</tr>
<tr>
<td></td>
<td>Collecting by monitoring tile: defined interval → xml file</td>
</tr>
</tbody>
</table>
Task Graph: Communication Pattern
## Shared Memory Model

### Tracing information collected at software level

<table>
<thead>
<tr>
<th>Data</th>
<th>Event [Barrier/Mutex/Lock/Cond]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The number of occurrence of each event</td>
</tr>
<tr>
<td></td>
<td>Max/Min/Ave time of each thread/process for each event, Process/pthread ID, address of each event group</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Period</th>
<th>Saving on each tile: whenever event happens</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Collecting by monitoring tile: defined interval</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Statistical result for each process/pthread/event</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>xml file → synch graph : synchronization</td>
</tr>
</tbody>
</table>

### Tracing information collected at hardware level

<table>
<thead>
<tr>
<th>Data</th>
<th>CYC, MP_BUNDLE RETIRED, HIT, L2_HIT, TLB_HIT, TLB_EXC, MISS_I, MISS_D_RD, MISS_D_WR, MP_DATA_CACHE_STALL, MP_INSTRUCTION_CACHE_STALL, MP_ICACHE_HIT_ISSUED, MP_CONDITIONAL_BRANCH_MISSPREDICT, MP_CONDITIONAL_BRANCH_ISSUED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period</td>
<td>Saving on each tile</td>
</tr>
<tr>
<td></td>
<td>Collecting by monitoring tile: defined interval → xml file</td>
</tr>
</tbody>
</table>
Synch Graph: Synchronization
Graph View/Info Ported on MDE-2.0

- Tile-eclipse v3.2.0
- Java v1.6.0

Graph View

Graph Info

Task Graph View

Synch Graph View

Export to png file
Accessing PMCs on Maestro

Inseok Choi, Xu Yang, Meng-Ju Wu
and Donald Yeung

Department of Electrical and Computer Engineering
Institute for Advanced Computer Studies
University of Maryland at College Park
Outline

• Performance Measurement Counters (PMCs)
• PAPI
• Status
Accessing Maestro’s PMCs

- Maestro has 2 hardware performance measurement counters (PMCs) per tile
  - Each PMC can bind to one of 99 different events
  - Can monitor instructions, memory, network, and I/O
- OProfile
  - Profiler for accessing PMCs provided by Linux
  - Limitations: missing events, system profiling only
- PAPI
  - Exports all 99 events
  - Flexible instrumentation control
  - PMC Virtualization
    - 64-bit emulation
    - Per-process / per-thread virtual PMCs
    - Time multiplexing

Ported PAPI to Tilera MDE 2.0
PAPI Architecture

- PAPI based on raw PMC interface
  - A few PMC functions were implemented at user level
  - Not possible to implement OS-level virtualization
- PAPI based on `perfctr`
  - Kernel device driver for accessing PMCs
  - PMC functions mostly implemented at user level
- Current PAPI is based on **Perfmon2** (from HP Labs)
  - Most PMC functionality pushed into kernel
  - PMC functions accessed via system call interface

```
USER          PAPI (libpapi)
              Perfmon2 Library (libpfm)

OS            Linux Kernel + Perfmon2 Extension

HW            Performance Measurement Counters
```

- PAPI becomes a “wrapper” around Perfmon2
- In the future, Maestro programmers may just use Perfmon2
Porting Perfmon2’s Kernel Layer to Maestro

Linux Kernel

Intr. Handler

Catch context switch

ctxsw

Perfmon2 Kernel Extension

Virtualized PMC Contexts

thread 0

thread 1

thread n

Architecture Specific API Layer

HW

PMC1

PMC2

Architecture Specific Files

- arch/tile/kernel/futex.S
- arch/tile/kernel/intvec.S
- arch/tile/kernel/perfcount.c
- arch/tile/kernel/process.c
- arch/tile/kernel/syscall_table.S
- arch/tile/perfmon/perfmon.c
- arch/tile/perfmon/perfmon_tile.c
- include/asm-tile/perfmon.h
- include/asm-tile/system.h
- include/asm-tile/thread_info.h
- include/asm-tile/unistd.h
Status

• Basic PAPI / Perfmon2 functionality implemented
  • Binding events to counters
  • Start / stop counters
  • Time multiplexing

• More advanced functionality in progress
  • Sampling
  • Event-driven multiplexing

• Testing in progress
  • PAPI tests: 53 / 70 passed
  • Perfmon2 tests: 9 / 20 passed

• PAPI / Perfmon2 integration with other tools
  • Runtime monitor
  • PPW (University of Florida)