PAPI-NUMA: Middleware to Support Hardware Sampling

IVONNE LOPEZ AND SHIRLEY MOORE
UNIVERSITY OF TEXAS AT EL PASO

VINCE WEAVER
UNIVERSITY OF MAINE

SCALABLE TOOLS WORKSHOP
AUGUST 4, 2015
Motivation

- Modern architectures have complex shared cache and memory hierarchies with non-uniform memory access (NUMA).
- Sub-optimal data/thread placement resulting in non-local data accesses can seriously degrade performance.
- Application developers need tools to help diagnose NUMA performance issues.
- Tool developers have to implement low-level access to sampling data
  - Redundant effort
  - Measurement part of tool is not released or not usable on production machines.
NUMA
NUMA Example: STREAM on Stampede with 16 threads

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>44840.9</td>
<td>0.005951</td>
<td>0.003568</td>
<td>0.017847</td>
</tr>
<tr>
<td>Scale:</td>
<td>47127.0</td>
<td>0.004679</td>
<td>0.003395</td>
<td>0.012240</td>
</tr>
<tr>
<td>Add:</td>
<td>52849.9</td>
<td>0.005304</td>
<td>0.004541</td>
<td>0.011292</td>
</tr>
<tr>
<td>Triad:</td>
<td>53368.3</td>
<td>0.005225</td>
<td>0.004497</td>
<td>0.010981</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>7387.9</td>
<td>0.023044</td>
<td>0.021657</td>
<td>0.026183</td>
</tr>
<tr>
<td>Scale:</td>
<td>7259.9</td>
<td>0.023979</td>
<td>0.022039</td>
<td>0.028078</td>
</tr>
<tr>
<td>Add:</td>
<td>10768.7</td>
<td>0.025722</td>
<td>0.022287</td>
<td>0.030115</td>
</tr>
<tr>
<td>Triad:</td>
<td>10942.4</td>
<td>0.026642</td>
<td>0.021933</td>
<td>0.034551</td>
</tr>
</tbody>
</table>
Hardware Counters

- Model Specific Registers (MSRs) that count hardware events (e.g., cycles, instructions retired, cache misses, different types of operations)

- Data collection methodologies
  - Counting: count how many times a given event occurs
  - Sampling: sample event and correlate with other information (e.g., program counter, data address, access latency, data source)
PAPI

- The Performance Application Programming Interface (PAPI) aims to provide the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors.

- PAPI enables software engineers to see, in near real time, the relation between software performance and processor events.

- It is being widely used to collect low level performance metrics (e.g. instruction counts, clock cycles, cache misses) of computer systems running UNIX/Linux operating systems.
Software Stack for NUMA Sampling

- Performance analysis tools (e.g., HPCToolkit-NUMA, MemAxes, TAU)
- PAPI-NUMA
- Linux perf_event
- Platform-specific Interface (e.g., Intel PEBS-LL, AMD IBS)
- Hardware Performance Counters
Linux\ perf\_event

- Linux kernel infrastructure that exposes hardware and software events
  - Provides an abstraction of performance events to user space
  - Provides a flexible interface for architecture-specific usage

- Exposed through \texttt{perf\_event\_open()} system call
  - \texttt{int perf\_event\_open(struct perf\_event\_attr *attr, pid\_t pid, int cpu, int group\_fd, unsigned long flags)};
  - \texttt{perf\_event\_attr} struct is populated before the call
  - returns a file descriptor

- Different counting and sampling configurations

- Counted events accessed through \texttt{read()}\n
- Sampled events accessed through \texttt{mmap()}
PAPI-NUMA Interface

- **Goal:** Provide a stable sampling interface to which tool developers can program

- **PAPI-NUMA routines**
  - `PAPI_sample_init()`: sets up `perf_event_attr` structure and calls `perf_event_open` (leaves sampling disabled)
  - `PAPI_sample_start()`: enables sampling
  - `PAPI_sample_stop()`: disables sampling
PAPI_sample_init()

int PAPI_sample_init(
    int EventSet,
    int EventCode,
    int sample_type,
    int sample_period,
    int threshold,  /* user-defined threshold for latency events */
    PAPI_sample_handler_t handler);

typedef void PAPI_sampleHandler_t(int signum, siginfo_t *info,
    void *ucontext);
Getting Per-thread Samples

- Highly desirable to obtain per-thread samples, since multithreaded codes may need to be analyzed for NUMA effects.
- Remote memory access on a NUMA system can degrade performance.
- Samples are collected only for the calling process and thread.
- `perf_event` kernel code specifically blocks getting mmap samples if inherit is enabled.
- Solution: Set up a counter on each logical CPU, each with its own mmap buffer.
- Currently requires kernel patch to propagate per-thread samples
int PAPI_sample_init (  
    int EventSet,  
    int EventCode,  
    int sample_type,  
    int sample_period,  
    int threshold,  
    PAPI_sample_handler_t handler,  
    int *fds);  

• Returns file descriptor from  
  perf_event_open() for each logical  
  CPU

Client code

◦ Sets up and associates mmap  
  buffer with each file descriptor  
◦ Calls PAPI_sample_start(fd) for  
  each file-descriptor to start per- 
  thread sampling  
◦ Interrupt handler checks which file  
  descriptor is passed in and reads  
  mmap buffer for that file  
  descriptor
Utility Code

- perf_mmap_read()
  - Parses the mmap buffer
  - Determines type of record
  - For PERF_RECORD_SAMPLE
    - Prints values of fields that were requested by PAPI_sample_init()

- Example interrupt handler
  - Determines appropriate mmap buffer
  - Calls perf_mmap_read() on that buffer
  - Counts samples
Sample Results

- From instrumented OpenMP version of STREAM run with 8 threads on Stampede node

  PERF_SAMPLE_IP, IP: 4012c0
  PERF_SAMPLE_TID, pid: 3144, tid: 3144
  PERF_SAMPLE_WEIGHT, Weight: 7
  PERF_SAMPLE_DATA_SRC, Raw: 68100142
  Load Hit L1 cache No snoop Hit Level 1 TLB Level 2 TLB

  PERF_SAMPLE_IP, IP: 401a78
  PERF_SAMPLE_TID, pid: 3144, tid: 3167
  PERF_SAMPLE_WEIGHT, Weight: 28
  PERF_SAMPLE_DATA_SRC, Raw: 68100242
  Load Hit Line fill buffer No snoop Hit Level 1 TLB Level 2 TLB
How to Best Help Tool Developers?

▪ How to provide results?
  ◦ Provide common PAPI-specific generic sampling interface and have all components map their samples to it
    ◦ PAPI would need to be constantly updated to extend and handle all of the various low-level changes.
  ◦ Dump raw data for the user/tool to interpret
    ◦ Requires additional user/tool code to interpret the data (could be provided as PAPI utility code)
  ◦ Dump data in Linux perf tool format
  ◦ All of the above?

▪ Survey tool developers to determine their requirements
▪ Investigate usefulness of sampling data besides NUMA data
Conclusions and Future Work

- Initial prototype is a low-level interface intended for performance tool developers.

- Plan to make our implementation available to tool developers to get feedback

- Plan to design a higher-level interface that will not require the user to provide the signal handler nor parse the mmap buffer.

- Having per-thread sampling of memory events available on stock Linux kernels through the PAPI interface will improve tool/user accessibility to NUMA data.

- Presented at XSEDE15, considerable interest from audience
Acknowledgments

▪ This work is partially supported by the
  • Department of Energy SciDAC program under grant number DE-SC00006722
  • Air Force Office of Scientific Research under AFOSR Award No. FA9550-12-1-0476