Memory Traffic and Complete Application Profiling with PAPI Multi-Component Measurements

Daniel Barry, Heike Jagode, Anthony Danalis, Jack Dongarra

May 15, 2023
Motivation

• Accurately monitor mem. traffic
Motivation

- Accurately monitor mem. traffic
- Mem. traffic counters are in the *nest*
Motivation

- Accurately monitor mem. traffic
- Mem. traffic counters are in the nest
- Require elevated privileges to access

Software Stack

- Performance API (PAPI)
  - Perf Uncore Comp.
  - PCP Comp.
- PCP Daemon
- Nest Hardware Counters
Motivation

• Accurately monitor mem. traffic

• Mem. traffic counters are in the nest

• Require elevated privileges to access

• IBM chose Performance Co-Pilot (PCP)
  • Third-party performance monitoring infrastructure

Software Stack

Performance API (PAPI)

Perf Uncore Comp.  PCP Comp.

PCP Daemon

Nest Hardware Counters
Computing Environments

• Summit (ORNL)
  • Nest access via PCP daemon.

• Tellico (UTK)
  • Direct access to nest counters (elevated privileges).
  • Baseline measurement to gauge overhead of PCP.
Computing Environments

- **Summit (ORNL)**
  - Nest access via PCP daemon.

- **Tellico (UTK)**
  - Direct access to nest counters (elevated privileges).
  - Baseline measurement to gauge overhead of PCP.

**Software Stack**

- Performance API (PAPI)
- Perf Uncore Comp.
- Nest Hardware Counters
Validation Experiments

1. Common BLAS Kernels

2. 3D-FFT Case Study

3. Profiling Full Applications with Multiple Components of PAPI
Validation Experiments

1. Common BLAS Kernels

2. 3D-FFT Case Study

3. Profiling Full Applications with Multiple Components of PAPI
How Reliably Do Memory Events Correspond to BLAS Kernels?

### GEMM

```plaintext
for ( i = 0; i < N; i++ )
    for ( j = 0; j < N; j++ ) {
        sum = 0.0;
        for ( k = 0; k < N; k++ )
            sum += A[i][k]*B[k][j];
        C[i][j] = sum;
    }
```

### GEMV

```plaintext
for ( i = 0; i < M; i++ ) {
    sum = 0.0;
    for ( k = 0; k < N; k++ )
        sum += A[i][k]*x[k];
    y[i] = sum;
}
```

\[ C = A \times B \quad y = A \times x \]
Does Infrastructure Make a Difference?
Does Infrastructure Make a Difference?

**Summit: Serial GEMM (POWER9, 1 Rep., 1T)**

- Mem. Reads
- Mem. Writes
- $8 \times 3 \times N^2 / 64$
- $8 \times N^2 / 64$

**Tellico: Serial GEMM (POWER9, 1 Rep., 1T)**

- Mem. Reads
- Mem. Writes
- $8 \times 3 \times N^2 / 64$
- $8 \times N^2 / 64$
Does Infrastructure Make a Difference?
Does Infrastructure Make a Difference?
The More GEMMs the Merrier

Summit: Serial GEMM (POWER9, Adapt. Reps., 1T)

Tellico: Serial GEMM (POWER9, Adapt. Reps., 1T)
Using a Batched GEMM

• “Batched” = Each core executes a 1-thread GEMM (using OpenMP).

GEMM

```c
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++) {
        sum = 0.0;
        for (k = 0; k < N; k++)
            sum += A[i][k]*B[k][j];
        C[i][j] = sum;
    }
```

Batched GEMM

```c
#pragma omp parallel for schedule(static)
for (idx = 0; idx < numThreads; idx++)
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++) {
            sum = 0.0;
            for (k = 0; k < N; k++)
                sum += A[idx][i][k]*B[idx][k][j];
            C[idx][i][j] = sum;
        }
```
IBM POWER9 Core Topology

= 1 Core  = 1 Thread

Core Reserved for Sys. Service Tasks
Batched GEMM

= 1 GEMM  = 1 Core  = 1 Thread
Shared-Work GEMM

= 1 GEMM   = 1 Core   = 1 Thread
Results from Batched GEMM

Summit: Batched GEMM (POWER9, Adapt. Reps., 21T)

Tellico: Batched GEMM (POWER9, Adapt. Reps., 16T)
Caveats of the GEMV

- GEMM experiments have shown need for lots of work.
- GEMV does even less work than GEMM:
  \[ y = A \times x \]
- For an MxN matrix A, GEMV’s reading is \( MN + M + N \).
- But writing is only M.
- We run out of memory before M is large enough for meaningful measurements.
Capping the Size of GEMV

\[ y[i] = A[i \% P][:] \cdot x[:]. \]
Capping the Size of GEMV

\[ y[i] = A[i\%P][:] \cdot x[:]. \]
Capping the Size of GEMV

\[ y[i] = A[i\%P][:] \cdot x[:], \]

Memory Saved per A Matrix

Actual Memory Usage for A
Batch of Capped GEMVs

Summit: Batched, Capped GEMV (POWER9, 128 Reps., 21T)

Tellico: Batched, Capped GEMV (POWER9, 128 Reps., 16T)
Validation Experiments

1. Common BLAS Kernels

2. 3D-FFT Case Study

3. Profiling Full Applications with Multiple Components of PAPI
Case Study: 3D-FFT

• 3D-FFT is a workhorse kernel:
  • Computationally heavy 1D-FFT phases
  • Memory re-sorting phases
  • All2All MPI communication phases

• Used in applications: HACC, GESTS, QMCPACK, etc.

• Memory re-sorting phases:
  • “S1CF” — store_1st_colwise_forward
  • “S2CF” — store_2nd_colwise_forward
  • “S1PF” — store_1st_planewise_forward
  • “S2PF” — store_2nd_planewise_forward
Case Study: 3D-FFT

• 3D-FFT is a workhorse kernel:
  • Computationally heavy 1D-FFT phases
  • Memory re-sorting phases
  • All2All MPI communication phases

• Used in applications: HACC, GESTS, QMCPACK, etc.

• Memory re-sorting phases:
  • “S1CF” — store_1st_colwise_forward
  • “S2CF” — store_2nd_colwise_forward
  • “S1PF” — store_1st_planewise_forward
  • “S2PF” — store_2nd_planewise_forward
Domain Decomp. & Re-sorting Visualized

Domain Decomposition

ROWS → COLS → PLANES → c

S1CF → S2CF
```c
#pragma omp parallel for schedule(static)
for ( plane = 0; plane < PLANES; plane++ ) {
    for ( row = 0; row < ROWS; row++ ) {
        for ( col = 0; col < COLS; col++ ) {
            tmp[plane][row][col] = in[plane*ROWS*COLS + row*COLS + col];
        }
    }
}
```
#pragma omp parallel for schedule(static)
for ( col = 0; col < COLS; col++ ) {
    for ( plane = 0; plane < PLANES; plane++ ) {
        for ( row = 0; row < ROWS; row++ ) {
            out[col*PLANES*ROWS + plane*ROWS + row] = tmp[plane][row][col];
        }
    }
}
#pragma omp parallel for schedule(static)
for ( col = 0; col < COLS; col++ ) {
    for ( plane = 0; plane < PLANES; plane++ ) {
        for ( row = 0; row < ROWS; row++ ) {
            out[col*PLANES*ROWS + plane*ROWS + row] = tmp[plane][row][col];
        }
    }
}
#pragma omp parallel for schedule(static)
for ( col = 0; col < COLS; col++ ) {
    for ( plane = 0; plane < PLANES; plane++ ) {
        for ( row = 0; row < ROWS; row++ ) {
            out[col*PLANES*ROWS + plane*ROWS + row] = tmp[plane][row][col];
        }
    }
}
X = COLS/r; Y = r;

#pragma omp parallel for schedule(static)
for ( plane = 0; plane < PLANES; plane++ ) {
    for ( x = 0; x < X; x++ ) {
        for ( y = 0; y < Y; y++ ) {
            for ( row = 0; row < ROWS; row++ ) {
                idx_out = plane*X*Y*ROWS + x*Y*ROWS + y*ROWS + row;
                idx_in = y*PLANES*X*ROWS + plane*X*ROWS + x*ROWS + row;

                out[idx_out] = in[idx_in];
            }
        }
    }
}
Assembly & Avoiding Cache on PowerPC

**PowerPC**

The PowerPC provides the following data prefetch instructions [14]:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>dcbt</td>
<td>Data Cache Block Touch</td>
</tr>
<tr>
<td>dcbtst</td>
<td>Data Cache Block Touch for Store</td>
</tr>
</tbody>
</table>

There are no alignment restrictions on the address of the data to prefetch.

GCC: https://gcc.gnu.org/projects/prefetch.html
Assembly & Avoiding Cache on PowerPC

PowerPC

The PowerPC provides the following data prefetch instructions [14]:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcbt</td>
<td>Data Cache Block Touch</td>
</tr>
<tr>
<td>dcbtst</td>
<td>Data Cache Block Touch for Store</td>
</tr>
</tbody>
</table>

There are no alignment restrictions on the address of the data to prefetch.

GCC: https://gcc.gnu.org/projects/prefetch.html
Assembly & Avoiding Cache on PowerPC

PowerPC

The PowerPC provides the following data prefetch instructions [14]:

<table>
<thead>
<tr>
<th>dcbt</th>
<th>Data Cache Block Touch</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcbtst</td>
<td>Data Cache Block Touch for Store</td>
</tr>
</tbody>
</table>

There are no alignment restrictions on the address of the data to prefetch.

GCC: https://gcc.gnu.org/projects/prefetch.html

-fprefetch-loop-arrays
A Closer Look at S1CF: Loop Nest 1
A Closer Look at S1CF: Loop Nest 1

S1CF, Loop Nest 1 (POWER9, 1 Rep., 4N, 1T)

Nest Event Counts

Mem. Reads
Mem. Writes
16*2*N^3 / 64
16*N^3 / 64

Size [N]

S1CF, Loop Nest 1 (POWER9, 1 Rep., 4N, 1T)

Nest Event Counts

Mem. Reads
Mem. Writes
16*2*N^3 / 64
16*N^3 / 64

Size [N]

-fprefetch-loop-arrays

404: 2c 4a 00 7c dcbt 0.r9
408: ec 41 00 7c dcbtst 0.r8
A Closer Look at S1CF: Loop Nest 2
A Closer Look at S1CF: Loop Nest 2

-fprefetch-loop-arrays
#pragma omp parallel for schedule(static)
for (plane = 0; plane < PLANES; plane++) {
    for (row = 0; row < ROWS; row++) {
        for (col = 0; col < COLS; col++) {
            out[col*PLANES*ROWS + plane*ROWS + row] = in[plane*ROWS*COLS + row*COLS + col];
        }
    }
}
A Closer Look at S2CF

![Graph showing S2CF (POWER9, 1 Rep., 4N, 1T)]
A Closer Look at S2CF

-fprefetch-loop-arrays
Examples of Larger Jobs

3D-FFT (IBM POWER9, 1 Rep., 16N, 21T)

<table>
<thead>
<tr>
<th></th>
<th>S1CF</th>
<th>S1CF</th>
<th>S2CF</th>
<th>S2CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>1344</td>
<td>2016</td>
<td>1344</td>
<td>2016</td>
</tr>
</tbody>
</table>

- Read-Expectation
- Read-Measurement
- Write-Expectation
- Write-Measurement

Nest Event Counts

0 1x10^9 2x10^9 3x10^9 4x10^9 5x10^9
Validation Experiments

1. Common BLAS Kernels

2. 3D-FFT Case Study

3. Profiling Full Applications with Multiple Components of PAPI
Profiling the 3D-FFT on Summit

3D-FFT: Memory, Power, and Network Traffic Profile
(N=2688, 64 MPI Ranks, IBM POWER9, NVIDIA Tesla V100)
Profiling QMCPACK on Summit

QMCPACK: Memory, Power, and Network Traffic Profile
(NiO-fcc-S32-dmc, 32 MPI Ranks, IBM POWER9, NVIDIA Tesla V100)
Profiling QMCPACK on Summit

QMCPACK: Memory, Power, and Network Traffic Profile
(NiO-fcc-S32-dmc, 32 MPI Ranks, IBM POWER9, NVIDIA Tesla V100)
Profiling QMCPACK on Summit

QMCPACK: Memory, Power, and Network Traffic Profile
(NiO-fcc-S32-dmc, 32 MPI Ranks, IBM POWER9, NVIDIA Tesla V100)
Conclusions

• Measurements of memory traffic for tiny data are unreliable.
Conclusions

• Measurements of memory traffic for tiny data are unreliable.
• Memory traffic measurements are sensitive to micro-architectural details
  • e.g. localized L3 slices, cache-avoidant writes
Conclusions

• Measurements of memory traffic for tiny data are unreliable.
• Memory traffic measurements are sensitive to micro-architectural details
  • e.g. localized L3 slices, cache-avoidant writes

• PAPI allows users to generate complete profiles for the entire system:
  • GPU Power
  • Memory Traffic
  • Infiniband Network Traffic
  • CPU, GPU, on- and off-chip memory, IO, networks, and more!
Future Work

• Focusing on other BLAS operations.

• Upcoming IBM architectures, such as POWER10.

• Categories of nest hardware other than solely memory traffic.
Acknowledgements

This research was supported in part by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration; and by the National Science Foundation under award No. 1900888 “ANACIN-X.”