Code profiling with respect to memory using CrayPat

JaeHyuk Kwack & Galen Arnold
SEAS group (help+bw@ncsa.illinois.edu)
The Blue Waters system

<table>
<thead>
<tr>
<th>22,640 XE6 compute nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Core Modules</td>
</tr>
<tr>
<td>Peak Performance</td>
</tr>
<tr>
<td>Memory Size</td>
</tr>
<tr>
<td>Memory Bandwidth (Peak)</td>
</tr>
<tr>
<td>Interconnect Injection Bandwidth (Peak)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4,228 XK7 compute nodes with NVIDIA Kepler (GK110) GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Processor</td>
</tr>
<tr>
<td>Host Processor Performance</td>
</tr>
<tr>
<td>Kepler Peak (DP floating point)</td>
</tr>
<tr>
<td>Host Memory</td>
</tr>
<tr>
<td>Kepler Memory</td>
</tr>
</tbody>
</table>

Code profiling with respect to memory using CrayPat
Introduction of CrayPat

• Performance measurement and analysis
  • Automatic Profiling Analysis
  • Load Imbalance
  • HW counter derived metrics
  • Predefined trace groups provide performance statistics for libraries called by program (blas, lapack, scalapack, petsc, fftw, cuda, hdf5, netcdf, etc.)
  • Support MPI, SHMEM, OpenMP, UPC, CAF, OpenACC
  • Access to network counters
  • Minimal program perturbation
• Limitations
  • Instrumenting only executable binaries
    • Profiling a code using python wrapper cannot be instrumented: Cray is working on it
  • Tracing with many MPI processes yields huge data, while sampling with many MPI ranks is fine

Code profiling with respect to memory using CrayPat
Introduction of CrayPat

- Two modes of use
  - CrayPat-lite for novice users, or convenience
    - % module unload darshan
    - % module load perftools-base perftools-lite
  - CrayPat for in-depth performance investigation and tuning assistance
    - % module unload darshan
    - % module load perftools-base perftools

```
jkwack@h2ologin2:~$ module avail perftools
```

- Compatible programming environments
  - GNU, Intel, PGI and Cray compilers for most of functions
  - Only for Cray compiler: reveal, loop work estimates (with “-h profile_generate”), and so on
How to use CrayPat

- Procedures for instrumentation
  - Building program after updating modules for CrayPat
    - No special flags required in general (e.g., -g is not required)
    - With any optimization flag (e.g., -00, -01, -02, -03)
  - Instrumenting the original program
    - For the default Automatic Profiling Analysis, % pat_build my_program
    - For predefined trace groups, % pat_build -g tracegroup my_program
    - For enabling tracing and the CrayPat API, % pat_build -w my_program
    - For instrumenting a single function, % pat_build -T tracefunc my_program
    - For instrumenting a list of functions, % pat_build -t tracefile my_program
  - This produces the instrumented executable my_program+pat
How to use CrayPat

- Running the instrumented executable
  - Running it after updating modules for CrayPat
    - `% aprun my_program+pat`
  - This produces a data file `my_program+pat+PID+node[s|t].xf`
    - `s` for sampling | `t` for tracing
    - For many MPI ranks, the folder `my_program+pat+PID+node[s|t]` is produced
- CrayPat Run Time Environment
  - “export PAT_RT_SUMMARY=0” to disable run time summarization before be saved
  - Use “PAT_RT_PERFCTR” for monitoring performance counters (will discuss it later)
- Processing raw performance data and creating report
  - `% pat_report my_program+pat+PID+node[s|t].xf`
  - This generates an .ap2 file
    - A self-contained archive that can be reopened later using the `pat_report` command
    - The exported-data file format used by Cray Apprentice2
CrayPat API

- Focusing on a certain region within the code, either to reduce sampling overhead, reduce data file size, or because only a particular region is of interest
- Inserting calls into the program source
- Turning data capture on and off at key points during program execution

- Header files
  - pat_api.h for C
  - pat_apif.h or pat_apif77.h for Fortran

- Compiler macro, CRAY_PAT from the perftools-base module
  
  ```
  #if defined (CRAY_PAT)
      <CrayPat API calls>
  #endif
  ```
CrayPat API

- API calls in C syntax
  - `PAT_record(int state)`
    - Setting the recording state to `PAT_STATE_ON` or `PAT_STATE_OFF`
  - `PAT_region_begin(int id, const char *label)`
  - `PAT_region_end(int id)`
    - Defines the boundaries of a region
    - Regions must be either separate or nested

[an example]
```
PAT_record(PAT_STATE_ON);
PAT_region_begin(1, "task_region-1");
  <tasks;>
PAT_region_end(1);
PAT_region_begin(2, "task_region-2");
  <tasks;>
PAT_region_end(2);
PAT_record(PAT_STATE_OFF);
```
Profiling w.r.t. memory

- L1 cache
  - 16 KB for data per integer core
  - Latency 3-4 clocks
- L2 cache
  - 2 MB per two integer cores
  - Latency 21 clocks
- L3 cache
  - 16 MB per socket
  - Latency 87 clocks
- DDR memory
  - 64 GB for general XE nodes
  - 128 GB for himem XE nodes
  - 32 GB for general XK nodes
  - 64 GB for himem XK nodes
**PAT_RT_PERFCTR**

- Specifying the performance counters to be monitored during the execution of a program
- More details about hardware counters for AMD Interlagos
  - Four 48-bit performance counters in AMD Interlagos
  - `% pat_help counters amd_fam15h`

```bash
jkwack2@h2login3:~> pat_help counters amd_fam15h
Additional topics that may follow "counters amd_fam15h":
  deriv
groups
native
papi

pat_help counters amd_fam15h (-quit,=back ^=up /=top ^=search)
```

- PAPI performance counters (i.e., `papi`)
  - `PAPI_L1_DCA`: Level 1 data cache accesses.
  - `PAPI_L1_DCM`: Level 1 data cache misses.
  - `PAPI_L1_DCH`: Level 1 data cache hits. (derived)
  - `PAPI_FP_OPS`: Floating point operations.
  - `PAPI_DP_OPS`: Floating point operations; optimized to count scaled double precision vector operations.

Code profiling with respect to memory using CrayPat
PAT_RT_PERFCTR

- AMD native performance counters (i.e., native)
  - cray_nb:::L3_CACHE_MISSES
    Number of L3 cache misses for accesses from each core.
    ANY_CORE: Measure on any core
    CORE_0: Measure on Core1
    CORE_1: Measure on Core1
    CORE_2: Measure on Core2
    CORE_3: Measure on Core3
    CORE_4: Measure on Core4
    CORE_5: Measure on Core5
    CORE_6: Measure on Core6
    CORE_7: Measure on Core7
    PREFETCH: Count prefetches only
    READ_BLOCK_ANY: Count any read request
    READ_BLOCK_MODIFY: Read Block Modify
    READ_BLOCK_EXCLUSIVE: Read Block Exclusive (Data cache read)
    READ_BLOCK_SHARED: Read Block Shared (Instruction cache read)

- Derived metrics (i.e., deriv)
  - D1_D2_cache_hit_ratio: D1+D2 cache hit ratio,
    Computed as min (1, (PAPI_L1_DCA - D1_D2_miss) / PAPI_L1_DCA)
  - Computational_intensity: FP Ops / L1_DCA.
    Computed as fl_pe_sum / PAPI_L1_DCA_pe_sum

Code profiling with respect to memory using CrayPat
PAT_RT_PERFCTR

• Predefined counter groups (i.e., groups)
  • 0: Summary with instructions metrics
      PAPI_TOT_INS, PAPI_FP_OPS, PAPI_L1_DCA, PAPI_L1_DCM
  • 2: L1 and L2 Metrics
      PAPI_L1_DCA, PAPI_L1_DCM, DATA_CACHE_REFILLS_FROM_L2_OR_NORTHBRIDGE:ALL,
      DATA_CACHE_REFILLS_FROM_NORTHBRIDGE
  • 3: Bandwidth information
      DATA_CACHE_REFILLS_FROM_L2_OR_NORTHBRIDGE:ALL, DATA_CACHE_REFILLS_FROM_NORTHBRIDGE,,
      OCTWORD_WRITE_TRANSFERS:ALL
  • 23: FP, D1, D2, and TLB
      PAPI_FP_OPS, PAPI_L1_DCA, PAPI_L1_DCM, PAPI_TLB_DM, DATA_CACHE_REFILLS_FROM_NORTHBRIDGE
      , DATA_CACHE_REFILLS_FROM_L2_OR_NORTHBRIDGE:ALL

Code profiling with respect to memory using CrayPat
Example 1 (profiling cache utilization with CrayPat API)

- **Description**
  - Compute square of 2D array at each MPI rank: 
    \[ OA(i,j) = A(i,j) * A(i,j) \text{ with } 0 < i,j <= 1024 \]

- **Implementation of a CrayPat region in Fortran**
  - Adding header
  - Adding a CrayPat region
  - Instrumenting the executable
    - `% pat_build -w squ_arrays_MPI+O3`
  - Running the instrumented executable

```fortran
program main
  use mpi
  implicit none
  #ifdef CRAYPAT
    include "pat_api.h"
  #endif
  ! Turning on Pat_record
  #ifdef CRAYPAT
    call PAT_record(PAT_STATE_ON,ierr)
  #endif
  ! Computing square(A)
  #ifdef CRAYPAT
    call PAT_region_begin(1,'A(i,j)^2',ierr)
  #endif
  do i=1,n
    do j=1,n
      OA(i,j) = A(i,j)*A(i,j)
    enddo
  enddo
  #ifdef CRAYPAT
    call PAT_region_end(1,ierr)
  #endif
  ! Turning off PAT_record
  #ifdef CRAYPAT
    call PAT_record(PAT_STATE_OFF,ierr)
  #endif
end program main
```

```
jkwack2@nid00014:/SEAS/Profiling_Toy_Codes> export PAT_RT_PERFCTR=2
jkwack2@nid00014:/SEAS/Profiling_Toy_Codes> aprun -n 32 ./squ_arrays_MPI+pat
CrayPat/X: Version 6.4.6 Revision 7d0d87c 02/20/17 09:52:37
Number of MPI process = 32
Number of rows/columns of the matrix = 1024
```

Experiment data file written:

```
/mnt/obc/u/staff/jkwack2/SEAS/Profiling_Toy_Codes/squ_arrays_MPI+pat+9944-32t.xf
```

Application 410615 resources: utime ~13s, stime ~7s, Rss ~41496, inblocks ~19734, outblocks ~2949

```
jkwack2@nid00014:/SEAS/Profiling_Toy_Codes> pat_report squ_arrays_MPI+pat+9944-32t.xf
```

Code profiling with respect to memory using CrayPat
Example 1

- CrayPat report

<table>
<thead>
<tr>
<th>USER / #1.A(i,j)^2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
</tr>
<tr>
<td>Time</td>
</tr>
<tr>
<td>Imb. Time</td>
</tr>
<tr>
<td>Imb. Time%</td>
</tr>
<tr>
<td>Calls</td>
</tr>
<tr>
<td>PAPI_L1_DCM</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS_FROM_L2_OR_NORTHBRIDGE:</td>
</tr>
<tr>
<td>ALL</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS_FROM_NORTHBRIDGE</td>
</tr>
<tr>
<td>D1 cache hit,miss ratios</td>
</tr>
<tr>
<td>D1 cache hit,refill ratio</td>
</tr>
<tr>
<td>D1 cache utilization (misses)</td>
</tr>
<tr>
<td>D1 cache utilization (refills)</td>
</tr>
<tr>
<td>D2 cache hit,miss ratio</td>
</tr>
<tr>
<td>D1+D2 cache hit,miss ratio</td>
</tr>
<tr>
<td>D1+D2 cache utilization</td>
</tr>
<tr>
<td>System to D1 refill</td>
</tr>
<tr>
<td>System to D1 bandwidth</td>
</tr>
<tr>
<td>System to D1 bandwidth</td>
</tr>
<tr>
<td>D2 to D1 bandwidth</td>
</tr>
<tr>
<td>Average Time per Call</td>
</tr>
</tbody>
</table>

CrayPat Overhead : Time | 0.0%
Example 1

- Updating the loop for stride-1 reference pattern

```c
! Turning on Pat_record
#ifdef CRAYPAT
  call PAT_record(PAT_STATE_ON,ierr)
#endif

! Computing square(A) updated
#ifdef CRAYPAT
  call PAT_region_begin(11,'A(i,j)^2 updated ',ierr)
#endif

do j=1,n
  do i=1,n
    OA(i,j) = A(i,j)*A(i,j)
  enddo
endo

#ifdef CRAYPAT
  call PAT_region_end(11,ierr)
#endif

! Turning off PAT_record
#ifdef CRAYPAT
  call PAT_record(PAT_STATE_OFF,ierr)
#endif
```

### USER / #11.A(i,j)^2 updated

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time %</td>
<td>2.7%</td>
</tr>
<tr>
<td>Time</td>
<td>0.005086 secs</td>
</tr>
<tr>
<td>Imb. Time</td>
<td>0.002045 secs</td>
</tr>
<tr>
<td>Imb. Time %</td>
<td>29.6%</td>
</tr>
<tr>
<td>Calls</td>
<td>196.616 /sec</td>
</tr>
<tr>
<td>PAPI_L2_DCM</td>
<td>29.951M/sec</td>
</tr>
<tr>
<td>PAPI_L2_DCA</td>
<td>261.062M/sec</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS_FROM_L2_OR_NORTHBRIDGE:</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>105.171M/sec</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS_FROM_NORTHBRIDGE:</td>
<td>47.189M/sec</td>
</tr>
<tr>
<td>D1 cache hit,miss ratios</td>
<td>88.5% hits 11.5% misses</td>
</tr>
<tr>
<td>D1 cache hit,refill ratio</td>
<td>59.7% hits 40.3% misses</td>
</tr>
<tr>
<td>D1 cache utilization (misses)</td>
<td>8.72 refs/miss 1.09 avg hits</td>
</tr>
<tr>
<td>D1 cache utilization (refills)</td>
<td>2.48 refs/refill 0.31 avg uses</td>
</tr>
<tr>
<td>D2 cache hit,miss ratio</td>
<td>55.1% hits 44.9% misses</td>
</tr>
<tr>
<td>D1+D2 cache hit,miss ratio</td>
<td>94.9% hits 5.1% misses</td>
</tr>
<tr>
<td>D1+D2 cache utilization</td>
<td>19.43 refs/miss 2.43 avg hits</td>
</tr>
<tr>
<td>System to D1 refill</td>
<td>47.189M/sec 240,007 lines</td>
</tr>
<tr>
<td>System to D1 bandwidth</td>
<td>2,880.204MiB/sec 15,360,472 bytes</td>
</tr>
<tr>
<td>D2 to D1 bandwidth</td>
<td>3,538.928MiB/sec 18,873,530 bytes</td>
</tr>
<tr>
<td>Average Time per Call</td>
<td>0.005086 secs</td>
</tr>
<tr>
<td>CrayPat Overhead : Time</td>
<td>0.1%</td>
</tr>
</tbody>
</table>

Code profiling with respect to memory using CrayPat
Example 2 (measuring flop-rates with CrayPat API)

- High Performance Geometric Multi-Grid benchmark
  - HPC performance benchmarking based on full multi-grid (FMG) F-cycle
  - Reporting number of equations solvable per second for three resolutions (i.e., 1h, 2h, and 4h)
- Objective of profiling
  - Measuring flop-rates for three resolutions

Source: Williams (hpgmg.org), HPGMG BoF, SC-16, 2016
Example 2 (measuring flop-rates with CrayPat API)

- Implementation of CrayPat regions in C
  - Add header
    ```c
    #ifdef CRAYPAT
    #include "pat_api.h"
    #endif
    #define DYNAMIC_RANGE 3
    double AverageSolveTime[DYNAMIC_RANGE];
    for(l=0;l<DYNAMIC_RANGE;l++){
        // if problem size too small)break;
        #ifdef CRAYPAT
        if(l==0) PAT_region_begin(1,"hpgm_bench_1h");
        if(l==1) PAT_region_begin(2,"hpgm_bench_2h");
        if(l==2) PAT_region_begin(3,"hpgm_bench_4h");
        #endif
        if(l>0) restriction(MG_h.levels[l],VECTOR_F, MG_h.levels[l-1],VECTOR_F,RESTRICT_CELL);
        bench_hpgm(&MG_h,l,a,b,rtol);
        #ifdef CRAYPAT
        if(l==0) PAT_region_end(1);
        if(l==1) PAT_region_end(2);
        if(l==2) PAT_region_end(3);
        #endif
        AverageSolveTime[l] = (double)MG_h.timers.MGsolve / (double)MG_h.MGsolve_performed;
    }
    if(my_rank==0){printf(stdout,"\n\n===== Timing Breakdown ===================
MPrintTiming(&MG_h,l);
}
    #ifdef CRAYPAT
    PAT_record(PAT_STATE_OFF);
    #endif
    ```
  - Add CrayPat regions
Example 2 (measuring flop-rates with CrayPat API)

- Instrumenting the executable and running it
  - % pat_build -w hpgmg
  - % export PAT_RT_PERFCTR=23
  - % aprun –n 1024 hpgmg+pat 8 1

### Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Time</th>
<th>Imb. Time</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb. Time</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>667.109019</td>
<td>--</td>
<td>--</td>
<td>6.0</td>
<td>Total</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb. Time</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.1%</td>
<td>660.781551</td>
<td>--</td>
<td>--</td>
<td>4.0</td>
<td>USER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb. Time</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>54.8%</td>
<td>365.506644</td>
<td>0.000491</td>
<td>0.0%</td>
<td>1.0</td>
<td>#1.hpgmg_bench_1h</td>
</tr>
<tr>
<td>22.4%</td>
<td>149.585381</td>
<td>58.450097</td>
<td>28.1%</td>
<td>1.0</td>
<td>main</td>
</tr>
<tr>
<td>12.3%</td>
<td>82.151491</td>
<td>1.380443</td>
<td>1.7%</td>
<td>1.0</td>
<td>#2.hpgmg_bench_2h</td>
</tr>
<tr>
<td>9.5%</td>
<td>63.538036</td>
<td>0.022500</td>
<td>0.0%</td>
<td>1.0</td>
<td>#3.hpgmg_bench_4h</td>
</tr>
</tbody>
</table>

### Table 4: Memory High Water Mark by Numa Node

<table>
<thead>
<tr>
<th>Process</th>
<th>HiMem</th>
<th>HiMem</th>
<th>HiMem</th>
<th>HiMem</th>
<th>Numanode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numa</td>
<td>(MBytes)</td>
<td>Numa</td>
<td>(MBytes)</td>
<td>Numa</td>
<td>(MBytes)</td>
</tr>
<tr>
<td>Node 0</td>
<td>452.7</td>
<td>120.7</td>
<td>111.6</td>
<td>110.6</td>
<td>109.9</td>
</tr>
<tr>
<td>Node 1</td>
<td>115.1</td>
<td>108.6</td>
<td>2.7</td>
<td>2.1</td>
<td>1.6</td>
</tr>
<tr>
<td>Node 2</td>
<td>112.8</td>
<td>4.4</td>
<td>104.5</td>
<td>2.2</td>
<td>1.7</td>
</tr>
<tr>
<td>Node 3</td>
<td>112.5</td>
<td>3.9</td>
<td>2.4</td>
<td>103.9</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td>112.4</td>
<td>3.7</td>
<td>2.0</td>
<td>2.4</td>
<td>104.3</td>
</tr>
</tbody>
</table>
Example 2

- Region for 1h

**Code profiling with respect to memory using CrayPat**

<table>
<thead>
<tr>
<th>Region</th>
<th>Time%</th>
<th>Time</th>
<th>Imb. Time</th>
<th>Imb. Time%</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Calls</strong></td>
<td>54.8%</td>
<td>365.506644 s</td>
<td>0.000491 s</td>
<td>0.0%</td>
</tr>
<tr>
<td><strong>PAPI_L1_DCM</strong></td>
<td>57.83%</td>
<td>21,138,006,091 misses</td>
<td>160,846,785 misses</td>
<td></td>
</tr>
<tr>
<td><strong>PAPI_L1_TLB</strong></td>
<td>0.44%</td>
<td>1,267,602 M/sec</td>
<td>463,316,889,033 refs</td>
<td></td>
</tr>
<tr>
<td><strong>PAPI_FP_OPS</strong></td>
<td>1,117.47%</td>
<td>408,445,909,119 ops</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DATA_CACHE_REFILLS_FROM_L2_OR_NORTHBRIDGE:**

- **ALL** 86.59% /sec | 31,650,208,551 fills
- **DATA_CACHE_REFILLS_FROM_NORTHBRIDGE** 6.37% /sec | 2,330,850,470 fills
- **HW FP Ops / User time** 1,117.47% /sec | 408,445,909,119 ops

**Computational intensity** ops/cycle 0.88 ops/ref
**MFLOPS (aggregate)** 1,144,298.24 M/sec

**TLB utilization** 2,880.49 refs/miss 5.63 avg uses
**D1 cache hit,miss ratios** 95.4% hits 4.6% misses
**D1 cache hit,refill ratio** 93.2% hits 6.8% refills
**D1 cache utilization (misses)** 21.92 refs/mss 2.74 avg hits
**D1 cache utilization (refills)** 14.64 refs/refill 1.83 avg uses
**D2 cache hit miss ratio** 97.6% hits 2.4% misses
**D1+D2 cache hit,miss ratio** 99.7% hits 0.3% misses

**D1+D2 cache utilization** 297.63 refs/mss 37.20 avg hits
**System to D1 refill** 6.377 M/sec 2,330,850,470 lines
**System to D1 bandwidth** 389.22 M/sec 149,174,430,097 bytes
**D2 to D1 bandwidth** 4,895.97 M/sec 1,876,438,917,136 bytes
**Average Time per Call** 365.506644 secs
**CrayPat Overhead : Time** 0.0%
Example 2

- Region for 2h

### User / #2.hpgm_gbench_2h

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>12.3%</td>
</tr>
<tr>
<td>Time</td>
<td>82.151491 secs</td>
</tr>
<tr>
<td>Imb. Time</td>
<td>1.380443 secs</td>
</tr>
<tr>
<td>Imb. Time%</td>
<td>1.7%</td>
</tr>
<tr>
<td>Calls</td>
<td>0.012 /sec</td>
</tr>
<tr>
<td>PAPI_L1_DOM</td>
<td>57.165M/sec</td>
</tr>
<tr>
<td>PAPI_L1_DOM Misses</td>
<td>4,696,163,776</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>0.413M/sec</td>
</tr>
<tr>
<td>PAPI_L1_DCA Misses</td>
<td>33,919,317</td>
</tr>
<tr>
<td>PAPI_L2_DCA</td>
<td>1,271.030M/sec</td>
</tr>
<tr>
<td>PAPI_L2_DCA Misses</td>
<td>104,416,984,071</td>
</tr>
<tr>
<td>PAPI_FP_OPS</td>
<td>1,057.664M/sec</td>
</tr>
<tr>
<td>PAPI_FP_OPS Misses</td>
<td>86,888,685,162</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS_FROM_L2_OR_NORTHBRIDGE:</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>83.716M/sec</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS_FROM_NORTHBRIDGE</td>
<td>6.378M/sec</td>
</tr>
<tr>
<td>HW FP Ops / User Time</td>
<td>1.057.664M/sec</td>
</tr>
<tr>
<td>HW FP Ops / User Time Misses</td>
<td>86,888,685,162</td>
</tr>
</tbody>
</table>

**Computational intensity**  
ops/cycle: 0.83 ops/ref

**MFLOPS (aggregate)**  
1,083,048.07M/sec

**TLB utilization**  
3,078.39 refs/miss: 6.01 avg uses

**D1 cache hit,miss ratios**  
95.5% hits: 4.5% misses

**D1 cache hit,refill ratio**  
93.4% hits: 6.6% refills

**D1 cache utilization (misses)**  
22.23 refs/miss: 2.78 avg hits

**D1 cache utilization (refills)**  
15.18 refs/refill: 1.90 avg uses

**D2 cache hit,miss ratio**  
92.4% hits: 7.6% misses

**D1+D2 cache hit,miss ratio**  
99.7% hits: 0.3% misses

**D1+D2 cache utilization**  
291.86 refs/miss: 36.48 avg hits

**System to D1 refill**  
6.378M/sec: 523,930,239 lines

**System to D1 bandwidth**  
389.258Mib/sec: 33,531,535,265 bytes

**D2 to D1 bandwidth**  
4,720.388Mib/sec: 406,624,029,735 bytes

**Average Time per Call**  
82.151491 secs

**CrayPat Overhead : Time**  
0.0%
### Example 2

- Region for 4h

#### Code profiling with respect to memory using CrayPat

<table>
<thead>
<tr>
<th>Time%</th>
<th>9.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>63.538036 secs</td>
</tr>
<tr>
<td>Imb. Time</td>
<td>0.022500 secs</td>
</tr>
<tr>
<td>Imb. Time%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Calls</td>
<td>0.016 /sec</td>
</tr>
<tr>
<td>PAPI_L1_DOM</td>
<td>50.25M/sec</td>
</tr>
<tr>
<td>PAPI_L1_BM</td>
<td>0.319M/sec</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>1,247.096M/sec</td>
</tr>
<tr>
<td>PAPI_FP_OPS</td>
<td>866.400M/sec</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS_FROM_L2_OR_NORTHBRIDGE:</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>69.995M/sec</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS_FROM_NORTHBRIDGE</td>
<td>5.680M/sec</td>
</tr>
<tr>
<td>HW FP Ops / User time</td>
<td>866.400M/sec</td>
</tr>
<tr>
<td>Computational intensity ops/cycle</td>
<td>0.69 ops/ref</td>
</tr>
<tr>
<td>MFLOPS (aggregate)</td>
<td>887,193.52M/sec</td>
</tr>
<tr>
<td>TLB utilization</td>
<td>3,909.37 refs/miss</td>
</tr>
<tr>
<td>D1 cache hit,miss ratios</td>
<td>96.0% hits, 4.0% misses</td>
</tr>
<tr>
<td>D1 cache hit,refill ratio</td>
<td>94.4% hits, 5.6% refill</td>
</tr>
<tr>
<td>D1 cache utilization (misses)</td>
<td>24.82 refs/miss, 3.10 avg hits</td>
</tr>
<tr>
<td>D1 cache utilization (refills)</td>
<td>17.82 refs/refill, 2.23 avg uses</td>
</tr>
<tr>
<td>D2 cache hit,miss ratio</td>
<td>91.9% hits, 8.1% misses</td>
</tr>
<tr>
<td>D1+D2 cache hit,miss ratio</td>
<td>99.7% hits, 0.3% misses</td>
</tr>
<tr>
<td>D1+D2 cache utilization</td>
<td>305.82 refs/miss, 38.23 avg hits</td>
</tr>
<tr>
<td>System to D1 refill</td>
<td>5.680M/sec, 360,901,226 lines</td>
</tr>
<tr>
<td>System to D1 bandwidth</td>
<td>346.685MiB/sec, 23,097,678,453 bytes</td>
</tr>
<tr>
<td>D2 to D1 bandwidth</td>
<td>3,925.455MiB/sec, 261,531,344,492 bytes</td>
</tr>
<tr>
<td>Average Time per Call</td>
<td>63.538036 secs</td>
</tr>
<tr>
<td>CrayPat Overhead : Time</td>
<td>0.0%</td>
</tr>
</tbody>
</table>
Example 2, after profiling several cases

**DOFS/node vs. Cache hit ratio**

- **Graph**
  - X-axis: Number of equations per node (MDOFs/node)
  - Y-axis: Cache hit ratio (%)
  - Data points for L1 data cache and L1+L2 data cache are shown.

**Computational intensity @ L1 cache vs. HPGMG**

- **Graph**
  - X-axis: Computational Intensity (OPS/ref) @ L1 cache
  - Y-axis: HPGMG per compute node (GDOFs/sec/node)
  - Linear fit equation: $y = 32.577x + 0.1643$ with $R^2 = 0.99902$
Summary

- Procedure for CrayPat instrumentation
  - `pat_build`
  - `pat_report`
- CrayPat API
  - Headers
  - `pat_regions`
- Run-time environments (PAT_RT_PERFCTR)
  - `papi`
  - `native`
  - `derive`
  - `groups`
QUESTIONS ?
Acknowledgment

This study is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.