Performance Analysis with TAU, part II

George S. Markomanolis

8 August 2019
MPI+OpenMP
MiniWeather MPI+OpenMP compilation

- module load pgi
- module load tau
- export TAU_MAKEFILE=/sw/summit/.../ibm64linux/lib/Makefile.tau-pgi-papi-mpi-pdt-openmp-opari-pgi
- Export TAU_OPTIONS=‘-optLinking=-lpnetcdf –optPreProcess’
- Replace mpicxx with tau_cxx.sh in the Makefile
- make openmp
MiniWeather MPI compilation and execute

• Compile with: make openmp
• Execution:

  export TAU_METRICS=TIME:PAPI_TOT_INS:PAPI_TOT_CYC:PAPI_FP_OPS
  export TAU_PROFILE=1
  export TAU_TRACK_MESSAGE=1
  export TAU_COMM_MATRIX=1
  #TAU_CALLPATH=1
  #TAU_CALLPATH_DEPTH=10
  jsrun -n 64 -r 8 -a 1 -c 4 -b packed:4 ./miniWeather_mpi_openmp
Paraprof - OpenMP
Paraprof – OpenMP - TIME

Open slide master to edit
Paraprof – OpenMP - parallel loop

<table>
<thead>
<tr>
<th>Node</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.583</td>
<td>node 1, thread 1</td>
</tr>
<tr>
<td>1.583</td>
<td>node 45, thread 1</td>
</tr>
<tr>
<td>1.582</td>
<td>node 37, thread 2</td>
</tr>
<tr>
<td>1.582</td>
<td>node 37, thread 1</td>
</tr>
<tr>
<td>1.581</td>
<td>node 13, thread 2</td>
</tr>
<tr>
<td>1.581</td>
<td>node 25, thread 1</td>
</tr>
<tr>
<td>1.581</td>
<td>node 55, thread 1</td>
</tr>
<tr>
<td>1.581</td>
<td>node 13, thread 1</td>
</tr>
<tr>
<td>1.58</td>
<td>node 29, thread 1</td>
</tr>
<tr>
<td>1.58</td>
<td>node 61, thread 2</td>
</tr>
<tr>
<td>1.58</td>
<td>node 21, thread 3</td>
</tr>
<tr>
<td>1.571</td>
<td>node 21, thread 0</td>
</tr>
<tr>
<td>1.57</td>
<td>mean</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.436</td>
<td>node 44, thread 1</td>
</tr>
<tr>
<td>1.436</td>
<td>node 8, thread 0</td>
</tr>
<tr>
<td>1.436</td>
<td>node 7, thread 0</td>
</tr>
<tr>
<td>1.434</td>
<td>node 32, thread 0</td>
</tr>
<tr>
<td>1.433</td>
<td>node 6, thread 0</td>
</tr>
<tr>
<td>1.433</td>
<td>node 0, thread 0</td>
</tr>
<tr>
<td>1.433</td>
<td>node 56, thread 2</td>
</tr>
<tr>
<td>1.433</td>
<td>node 40, thread 0</td>
</tr>
<tr>
<td>1.433</td>
<td>node 62, thread 2</td>
</tr>
<tr>
<td>1.433</td>
<td>node 47, thread 0</td>
</tr>
<tr>
<td>1.432</td>
<td>node 30, thread 0</td>
</tr>
<tr>
<td>1.431</td>
<td>node 43, thread 3</td>
</tr>
<tr>
<td>1.431</td>
<td>node 15, thread 2</td>
</tr>
<tr>
<td>1.431</td>
<td>node 24, thread 3</td>
</tr>
<tr>
<td>1.431</td>
<td>node 10, thread 0</td>
</tr>
<tr>
<td>1.431</td>
<td>node 51, thread 0</td>
</tr>
<tr>
<td>1.431</td>
<td>node 28, thread 0</td>
</tr>
<tr>
<td>1.431</td>
<td>node 11, thread 3</td>
</tr>
<tr>
<td>1.431</td>
<td>node 12, thread 0</td>
</tr>
<tr>
<td>1.431</td>
<td>node 15, thread 1</td>
</tr>
</tbody>
</table>
Paraprof – OpenMP - IPC
GPU
MiniWeather MPI+OpenACC compilation

• module load tau
• Use mpicxx in the Makefile **not** tau_cxx.sh (for now)
• make openacc
MiniWeather MPI compilation and execute

- Compile with: make openacc
- Execution:
  
  ```
  export TAU_METRICS=TIME
  export TAU_PROFILE=1
  export TAU_TRACK_MESSAGE=1
  export TAU_COMM_MATRIX=1
  jsrun -n 6 -r 6 --smiargs="-gpu" -g 1 tau_exec -T mpi,pgi,pdt -openacc .;/miniWeather_mpi_openacc
  ```

- CUPTI metrics for OpenACC to be available up to SC19
Paraprof – OpenACC
Paraprof – OpenACC
Paraprof – OpenACC

From the main window right click one label and select “Show User Event Statistics Window”
CUPTI Metrics

- [https://docs.nvidia.com/cupti/Cupti/r_main.html#metrics-reference](https://docs.nvidia.com/cupti/Cupti/r_main.html#metrics-reference)

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>achieved_occupancy</td>
<td>Ratio of the average active warps per active cycle to the maximum number of</td>
<td>Multi-context</td>
</tr>
<tr>
<td></td>
<td>warps supported on a multiprocessor</td>
<td></td>
</tr>
<tr>
<td>alu_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute</td>
<td>Multi-context</td>
</tr>
<tr>
<td></td>
<td>integer and floating-point arithmetic instructions on a scale of 0 to 10.</td>
<td></td>
</tr>
<tr>
<td>atomic_replay_overhead</td>
<td>Average number of replays due to atomic and reduction bank conflicts for</td>
<td>Multi-context</td>
</tr>
<tr>
<td></td>
<td>each instruction executed</td>
<td></td>
</tr>
<tr>
<td>atomic_throughput</td>
<td>Global memory atomic and reduction throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>atomic_transactions</td>
<td>Global memory atomic and reduction transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>atomic_transactions_per_request</td>
<td>Average number of global memory atomic and reduction transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td></td>
<td>performed for each atomic and reduction instruction</td>
<td></td>
</tr>
<tr>
<td>branch_efficiency</td>
<td>Ratio of non-divergent branches to total branches expressed as percentage.</td>
<td>Multi-context</td>
</tr>
<tr>
<td></td>
<td>This is available for compute capability 3.0.</td>
<td></td>
</tr>
<tr>
<td>cf_executed</td>
<td>Number of executed control-flow instructions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>cf_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute</td>
<td>Multi-context</td>
</tr>
<tr>
<td></td>
<td>control-flow instructions on a scale of 0 to 10</td>
<td></td>
</tr>
<tr>
<td>cf_issued</td>
<td>Number of issued control-flow Instructions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_read_throughput</td>
<td>Device memory read throughput. This is available for compute capability 3.0,</td>
<td>Multi-context’</td>
</tr>
<tr>
<td></td>
<td>3.5 and 3.7.</td>
<td></td>
</tr>
<tr>
<td>dram_read_transactions</td>
<td>Device memory read transactions. This is available for compute capability 3.0,</td>
<td>Multi-context’</td>
</tr>
<tr>
<td></td>
<td>3.5 and 3.7.</td>
<td></td>
</tr>
</tbody>
</table>
module load gcc
export TAU_METRICS=TIME, achieved_occumency
jsrun --smiargs="-gpu" --nrs 2 --tasks_per_rs 1 --gpu_per_rs 1 --rs_per_host 2 --cpu_per_rs 8 --bind rs tau_exec -T mpi,pdt,papi,cupti,openmp -ompt -cupti $EXECUTABLE $INPUT

• Error: **Only counters for a single GPU device model can be collected at the same time.**

• **Achieved_occumency= CUDA.Tesla_V100-SXM2-16GB.domain_d.active_warps/CUDA.Tesla_V100-SXM2-16GB.domain_d.active_cycles**
LSMS MPI+OpenMP+CUDA execution

- module load gcc
- jsrun --smpiargs="-gpu" --nrs 2 --tasks_per_rs 1 --gpu_per_rs 1 --rs_per_host 2 --cpu_per_rs 8 --bind rs tau_exec -T mpi,pdt,papi,cupti,openmp -ompt -cupti $EXECUTABLE $INPUT
LSMS - Paraprof – MPI+OpenMP+CUDA

Options -> Uncheck Stack Bars Together
LSMS - Paraprof – MPI+OpenMP+CUDA

• Click on any color
LSMS - Paraprof – MPI+OpenMP+CUDA

• Statistics for thread on CPU
LSMS - Paraprof – MPI+OpenMP+CUDA

• Statistics for thread on GPU
LSMS - Paraprof – MPI+OpenMP+CUDA

- User event window
Benchmark for demonstration

```bash
export TAU_METRICS=TIME,achieved_occupancy
tsrun -n 2 -r 2 -g 1 tau_exec -T mpi,pdt,papi,cupti,openmp -ompt -cupti ./add
```

- Output folders:
  MULTI__TAUGPU_TIME
  MULTI__CUDA.Tesla_V100-SXM2-16GB.domain_d.active_warps
  MULTI__CUDA.Tesla_V100-SXM2-16GB.domain_d.active_cycles
  MULTI__achieved_occupancy
Bechmark - Paraprof - MPI+OpenMP+CUDA
LSMS - Parapronf – MPI+OpenMP+CUDA

- Select the metric achieved occupancy

[Image of a chart showing achieved occupancy metrics for different nodes and threads.]
LSMS - Paraprof – MPI+OpenMP+CUDA

• Click on the colored bar
• The achieved occupancy for this simple benchmark is 6.2%
• Similar approach for other metrics, not all of them can be used.

• TAU provides a tool called `tau_cupti_avail` where we can see the list of available metrics, then we have to figured out which CUPTI metrics use these ones.
TAU and tracing

- export TAU_TRACE=1
- export TAU_TRACE_FORMAT=otf2

- Currently, supported MPI and OpenSHMEM applications
- Use Vampir for visualization
Overhead

TAU Overhead

Should use PDT to exclude files/routines that cause overhead
TAU mechanisms

• TAU_THROTTLE
  - TAU by default excludes from the instrumentation routines that could cause overhead
  - Rule: If a routine is called more than 100,000 times and it spends up to 10 usecs/call, then exclude it.
  - Adjustable: TAU_THROTTLE_NUMCALLS, TAU_THROTTLE_PERCALL
Selective Instrumentation

- Do not instrument routine `sort*(int *)`

File `select.tau`:

```
BEGIN_EXCLUDE_LIST
void sort_#(int *)
END_EXCLUDE_LIST

TAU_OPTIONS="-optTauSelectFile=select.tau"
```

- Dynamic phase

```
BEGIN_INSTRUMENT_SECTION
dynamic phase name="phase1" file="miniWeather_mpi.cpp" line=300 to line=327
END_INSTRUMENT_SECTION
```
Static Phase

File phases.tau:
BEGIN_INSTRUMENT_SECTION
static phase name="phase1" file="miniWeather_mpi.cpp" line=300 to line=327
static phase name="phase2" file="miniWeather_mpi.cpp" line=333 to line=346
END_INSTRUMENT_SECTION

TAU_OPTIONS="-optTauSelectFile=phases.tau"
Creating static Phases

```c
for (kout; kout < n); kout++) {
    for (i; i < iend; i++) {
        // Use fourth-order interpolation from four cell averages to compute the value at the interface in question
        for (l = 1; l <= L; l++) {
            for (m = 1; m <= M; m++) {
                for (n = 1; n <= N; n++) {
                    // Use fourth-order accurate interpolation of the state
                    val[l][m][n] = stencil[l][m][n] + 2 * stencil[l][m][n] + 2 * stencil[l][m][n] + stencil[l][m][n];
                    // Use first-order accurate interpolation of the third spatial derivative of the state
                    d3Val[l][m][n] = stencil[l][m][n] + stencil[l][m][n] + stencil[l][m][n];
                }
            }
        }
    }
}
}
```

Compute density, wind, wind potential temperature, and pressure (r,u,v,w,p respectively)

```c
// Compute density, wind, wind potential temperature, and pressure (r,u,v,w,p respectively)
r = vals[ID浼BENS] + vals[ID浼BINS];
u = vals[ID浼BINS] / r;
v = vals[ID浼BINS] / r;
wind = (vals[ID浼BINS] + vals[ID浼BINS]) / r;
p = (B*(pow(r,5)))/r;
```

Compute the flux vector with hyperviscosity

```c
// Compute the flux vector with hyperviscosity
flux[ID浼BINS][x] = vals[ID浼BINS] + vals[ID浼BINS];
flux[ID浼BINS][x] = vals[ID浼BINS] + vals[ID浼BINS];
```

// Use thread to parallelize the code
// Use thread to parallelize the code
Creating static Phases
Creating static Phases
PerfExplorer

- PerfExplorer is a framework for parallel performance data mining and knowledge discovery
- Unfortunately not working efficient on Summit for now
- Perfexplorer should be installed, if it is not configured, it will propose the next steps
- In this example we execute MiniWeather with MPI for 8,16,32,64 processes, we load them to paraprof, select the name of the experiment, right click and select “Upload Trial to DB”
Right click on the highlighted name of the experiment with the path and select “Upload Trial to DB”, repeat for all the experiments.
PerfExplorer – Total Execution

Select the name of the experiment (arrow) and then select Charts -&gt; Total Execution Time -&gt; Select the metric TIME-&gt; Click OK
Although the duration of computation decreases as we increase the number of processes, some MPI calls remain similar duration and the `MPI_File_write_at` is increasing.

Click Charts -> Stacked Bar Chart -> Select the metric TIME -> Click OK
PerfExplorer – Tip

If the data include more than one metric, and the user wants to visualize another metric, then click in any of the experiments in the green arrow and then back to the name of the experiments (grey arrow). Then in the next visualization, PerfExplorer will require to choose which metric to visualize.
PerfExplorer – PAPI_TOT_INS

Click Charts -> Stacked Bar Chart -> Select the metric TIME -> Click OK

MPI calls are increasing instructions as we increase the number of MPI processes.
PerfExplorer – Relative efficiency

Click Charts -> Relative Efficiency -> “The problem size remains constant” -> Click OK
PerfExplorer – Relative Efficiency by event

Click Charts -> Relative Efficiency by Event -> Metric TIME -> “The problem size remains constant” -> Click OK
PerfExplorer – Relative Speedup

Click Charts -> Relative Speedup -> Metric TIME -> Click OK -> “The problem size remains constant” -> Click OK
PerfExplorer – Relative Speedup

Click Charts -> Relative Speedup by Event -> Metric TIME -> Click OK -> “The problem size remains constant” -> Click OK
PerfExplorer – Relative Speedup

Click Charts -> Correlate Events with Total Runtime -> Metric TIME -> Click OK -> “The problem size remains constant” -> Click OK
Click Charts -> Runtime Breakdown -> Metric TIME -> Click OK -> “The problem size remains constant” -> Click OK
PerfExplorer – Relative Speedup

Click Charts -> Runtime Breakdown -> Metric PAPI_TOT_INS -> Click OK -> "The problem size remains constant" -> Click OK
Conclusions

• TAU is a promising tool, with some good features to be improved related to GPU performance analysis

• Interesting functionalities to identify loop issues and create dynamic phases of an iterative analysis

• We would like OTF2 traces with CUDA/OpenACC

• OpenACC with support of CUPTI metrics is coming around to SC19

• It supports Python instrumentation but not activated during compilation