High Performance Profiling Tools

Georges Markomanolis
Master internship
Graal Team

September 7, 2009
Supervisors: Frédéric Desprez and Frédéric Suter
Introduction

- Many profiling tools in order to profile/trace a parallel application
- Communication bottlenecks
- TAU and Scalasca
- In the next sections an application will be profiled with both programs and a comparison of overhead will take place at the end.
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 supports 103 events.
- The events are hardware counters on the cpu.
- Every cpu has 2-8 counters.
The NAS Parallel Benchmarks (NPB) are a set of programs in order to evaluate the performance of parallel computers. The benchmarks are derived from computational fluid dynamics and they consist of five kernels and three applications. NAS Parallel Benchmarks are executed on Grid’5000.
NAS Parallel Benchmarks

- **EP**, this code implements the random-number generator. The code is “embarrassingly” parallel in that no communication is required for the generation of the random numbers itself. There is no special requirement on the number of processors used for running the benchmark.

- **FT**, this code implements the time integration of a three-dimensional partial differential equation using the Fast Fourier Transform.

- Data Traffic benchmark **DT** is written in C. DT benchmark takes one argument: BH, WH, or SH. This argument specifies the communication graph Black Hole, White Hole, or SHuffle respectively.

- **LU**, a regular-sparse, block \((5 \times 5)\) lower and upper triangular system solution. This problem represents the computations associated with the implicit operator of a newer class of implicit CFD algorithms.
NAS’s Classes

There are 7 classes at the NAS: W,S,A,B,C,D,E. Each one represent the size of the problem (input data).

- The W,S are very small classes.
- The A,B classes are used for a small number of processors.
- The C,D classes are used for executing big benchmarks.
- The class E is for huge benchmarks.
TAU

- Tuning and Analysis Utilities
- Support for multiple parallel programming paradigms: MPI, Multi-threading, Hybrid.
- Access to hardware counters.
- Automatically instruments your code.
- Implemented by University of Oregon
Using TAU

- Declaration of environment variables such as $TAU_MAKEFILE, $TAU_OPTIONS.
- Selective instrumentation by inserting TAU macros or automatically.
- Program Database Toolkit (PDT).
- For automatic instrumentation replace the compiler with TAU compiler script
- Many flags for configuration
<table>
<thead>
<tr>
<th>GNU Compilers</th>
<th>TAU shell scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpicc</td>
<td>tau_cc.sh</td>
</tr>
<tr>
<td>mpicxx</td>
<td>taucxx.sh</td>
</tr>
<tr>
<td>mpif77 mpif90</td>
<td>tau_f90.sh</td>
</tr>
</tbody>
</table>
TAU Instrumentation

- Support for standard programs events
  - Routines, classes and templates
  - Statement-level blocks
  - Begin/End events
- Support for user-defined events
  - Begin/End events specified by user
  - Atomic events (e.g., size of memory)
- Static events and dynamic events
MPI Wrapper Interposition Library

- Standard MPI Profiling Interface
- TAU instrumented MPI library
  - Interpose between MPI and TAU
  - During program link, -lmpi replaced by -lTauMpi -lpmpi
  - No change to the source code.
Selective Instrumentation File

- Specify a list of events or files to exclude or include

```
BEGIN_EXCLUDE_LIST
Verify
Tak#
common.f
END_EXCLUDE_LIST

BEGIN_INCLUDE_LIST
comp#
ff*.f
END_INCLUDE_LIST
```

- The # is wildcard for in a routine name and the * is wildcard character
Specify a list of user instrumentation commands

```
BEGIN_INSTRUMENT_SECTION
dynamic phase name"test1" file="ft.f" line=148 to line=173
loops file="ft.f" routine="#"
END_INSTRUMENT_SECTION
```
Demo - Paraprof manager for FT benchmark, class C
Demo - Explanation of exclusive and inclusive time
Demo - Exclusive time
## Demo - Exclusive time for an event (MPI_Alltoall)

![Exclusive time chart]

**File: TAU: ParaProf; Function Data Window: /home/yon/gmarkomanolis/nas/NPB3.3/w/tau/NPB3.3-MPI/bin/delete/tauprof**

**Name:** FF \((f(0);(79,7);194,9]) \(\Rightarrow\) test1 [0] \(\Rightarrow\) FFT \((f(0);(79,7);194,9]) \(\Rightarrow\) TRANSPOSE \((X,y) \((f(0);(1194,7);1213,9]) \(\Rightarrow\) TRANSPOSE_GLOBAL \((f(0);(1312,7);1333,9]) \(\Rightarrow\) MPI_AlltoallQ

**Metric:** CET_TIME_OF_DAY

**Value:** Exclusive

**Units:** seconds
Demo - Histogram for ILOG2()
Demo - PerfExplorer, Efficiency

![Graph showing the relative efficiency of PerfExplorer compared to TAU over different numbers of processors. The x-axis represents the number of processors ranging from 30 to 130, and the y-axis represents the value of relative efficiency. The graph demonstrates an increase in efficiency as the number of processors increases.]
Demo - PerfExplorer, Speedup

Relative Speedup - Default App:Default Exp:GET_TIME_OF_DAY

Value

30 40 50 60 70 80 90 100 110 120 130

Number of Processors

Default Exp Ideal
Demo - PerfExplorer, Efficiency by event

![Graph showing relative efficiency by event for different numbers of processors. The x-axis represents the number of processors, ranging from 30 to 130. The y-axis represents the value, ranging from 0 to 1.7. The graph includes multiple lines representing different events, such as CFXTS1, CFXTS2, CFZT, EVOLVE, FFT2, MPL_AIT맬러, MPL_Comm_split0, MPL_Init0, TRANSPOSE2_FINISH, TRANSPOSE2_LOCAL, and other.]
Demo - PerfExplorer, Speedup by event
Demo - MPI Time / Total time

![Graph showing MPI GET_TIME_OF_DAY / Total GET_TIME_OF_DAY](image-url)
Demo - The Runtime Breakdown chart
Demo - Communication Matrix

TOTAL VOLUME BYTES

DISPLAY OPTIONS

Callpath:
MPI_Send

Dataset:
TOTAL VOLUME BYTES
Demo - Comparison of two executions

![ParaProf Comparison Window](image)

**Module Comparison**

- **High Performance Profiling Tools**
- **Georges Markomanolis Master internship Graal Team**
- **Introduction**
- **NAS Parallel Benchmarks**
- **TAU**
- **Scalasca**
- **References**
Demo - Visualization II
One way to visualize the trace output from TAU is the Jumpshot (integrated with TAU).

Supported file formats: OTF, Vampir Trace Format, EPILOG, Slog2

For creating slog2 it is needed to merge all the trace files.
Demo - Visualizing trace files with Jumpshot
Demo - Visualizing trace files with Jumpshot
Pros and Cons

- **Pros**
  - Visualizing MPI calls is great help
  - User can identify message sizes, time taken
  - Automatic selective instrumentation
  - Communication Matrix
  - Tools for visualizing many useful data, save experiments to databases
  - Memory tracking
  - TAU_Cuda is on the way

- **Cons**
  - No automatic recognition of bottlenecks
  - Large trace files for visualising.
  - Slow tracefile reading
  - Bad structure of output file format
Scalasca

- Scalable Performance Analysis of Large-Scale Applications
- Profile analysis.
- Time-line analysis (trace)
- Automatic identification of bottlenecks
- Pattern analysis
- Automatically instruments your code.
- Implemented by Julich Supercomputing Centre
Using Scalasca

- Just add at your makefile a variable $PREP before mpicc and declare that PREP="scalasca -instrument"
- For executing your application with profiling, execute scalasca -analyze mpirun ...
- For executing your application with tracing, execute scalasca -analyze -t mpirun ...
- For analyzing profiling data, execute scalasca -examine name of folder with results of the above command.
- For analyzing tracing data, execute mpirun ... scout name of folder with results of the second command and after execute scalasca -examine name of folder.
Selective Instrumentation File

- Scalasca does not support selective instrumentation as TAU, it is possible to declare only the functions that should be excluded.

- The good news is that it will be available soon and it will be implemented like TAU, using PDT for automatic instrumentation.
Demo - Parallel program analysis report exploration (Cube3) for FT benchmark, class C
Demo - Selective instrumentation (phase)
Demo - Topology
Demo - Load balancing I

[Image of a visualization tool showing load balancing metrics]

Selected "4222.40 Time"
Demo - Load balancing II
Demo - Load balancing III
Demo - Late communication I

- High Performance Profiling Tools
  - Georges Markomanolis
  - Master internship Graal Team

Introduction

NAS Parallel Benchmarks

TAU
  - Introduction
  - Instrumentation
  - Demo
  - Conclusion

Scalasca
  - Introduction
  - Demo
  - Conclusion

Overhead comparison of Scalasca and TAU

References
N x N Completion Time

Description:
This pattern refers to the time spent in MPI n-to-n collectives after the first process has left the operation.

Note that the time reported by this pattern is not necessarily completely waiting time since some processes could -- at least theoretically -- still communicate with each other while others have already finished communicating and exited the operation.
Wait at N x N Time

Description:
Collective communication operations that send data from all processes to all processes (i.e., n-to-n) exhibit an inherent synchronization among all participants, that is, no process can finish the operation until the last process has started it. This pattern covers the time spent in n-to-n operations until all processes have reached it. It applies to the MPI calls MPI_Reduce_scatter(), MPI_Allgather(), MPI_Allgatherv(), MPI_Allreduce(), MPI_Alltoall(), MPI_Alltoallv().

Note that the time reported by this pattern is not necessarily completely waiting time since some processes could -- at least theoretically -- already communicate with each other while others have not yet entered the operation.
Late Broadcast Time

Description:
Collective communication operations that send data from one source process to all processes (i.e., 1-to-n) may suffer from waiting times if destination processes enter the operation earlier than the source process, that is, before any data could have been sent. The pattern refers to the time lost as a result of this situation. It applies to the MPI calls MPI_Bcast(), MPI_Scatter() and MPI_Scatterv().

![Late Broadcast Time Diagram](image.png)
Demo - Overhead

### Overhead comparison of Scalasca and TAU

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Scalasca</th>
<th>TAU</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2.34</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

#### Scalasca
- 2.34 Overhead
- 100.00 Visits
- 0.00 Synchronizations
- 0.00 Point-to-point
- 0.00 Sends
- 0.00 Late Receivers
- 0.00 Receivers
- 0.00 Late Senders

#### TAU
- 0.00 (100.00%)
- 0.00 (100.00%)
Many configuration flags
- Declare buffer size for tracing files
- Configuration file epik.conf
# EPIK user instrumentation API

- `#include "epik_user.h"
- EPIK_USER_REG(phase, "<< phase >> "
- EPIK_USER_START(phase)
- EPIK_USER_END(phase)"
Cube3 utilities

- the command `cube3_diff name1.cube name2.cube`, creates a diff.cube file with the difference of both cube files.
- `cube3_cut -r '<< phase >>' name.cube` creates a cube with the phase sub-tree.
Pros and Cons

- **Pros**
  - Parallel analyzer SCOUT
  - Automatic recognition of bottlenecks
  - A new release will be available during SC ’09 with many upgrades.
  - Very good structure of the output file format.

- **Cons**
  - Limited selective instrumentation
  - No Communication Matrix
### Overhead comparison of Scalasca and TAU about profiling

<table>
<thead>
<tr>
<th>Benchmark, Class, Cpus</th>
<th>TAU</th>
<th>Scalasca</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT, C, 32</td>
<td>5.02</td>
<td>2.17</td>
</tr>
<tr>
<td>FT, C, 64</td>
<td>5.5</td>
<td>2.37</td>
</tr>
<tr>
<td>FT, C, 128</td>
<td>9.45</td>
<td>8.08</td>
</tr>
<tr>
<td>FT, D, 128</td>
<td>1.03</td>
<td>0.79</td>
</tr>
<tr>
<td>EP, D, 32</td>
<td>0.29</td>
<td>0.2</td>
</tr>
<tr>
<td>EP, D, 64</td>
<td>1.79</td>
<td>1.2</td>
</tr>
<tr>
<td>EP, D, 128</td>
<td>3.7</td>
<td>2.82</td>
</tr>
<tr>
<td>EP, D, 192</td>
<td>5.44</td>
<td>3.8</td>
</tr>
<tr>
<td>DT, B, SH, 192</td>
<td>8.16</td>
<td>3.5</td>
</tr>
</tbody>
</table>
Overhead comparison of Scalasca and TAU about profiling II

<table>
<thead>
<tr>
<th>Benchmark, Class, Cpus</th>
<th>TAU</th>
<th>Scalasca</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT, C, BH, 85</td>
<td>0.37</td>
<td>0.18</td>
</tr>
<tr>
<td>DT, C, WH, 85</td>
<td>0.97</td>
<td>0.21</td>
</tr>
<tr>
<td>LU, C, 32</td>
<td>3.46</td>
<td>1.64</td>
</tr>
<tr>
<td>LU, C, 64</td>
<td>4.51</td>
<td>4.39</td>
</tr>
<tr>
<td>LU, C, 128</td>
<td>7.02</td>
<td>3.05</td>
</tr>
</tbody>
</table>

NAS Parallel Benchmarks
References 1

1. mpiP, TAU, Scalasca, on Blue Gene, Prashobh Balansundaram, IBM Dublin Software Labs, 2008


3. PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux, Rick Kufrin, National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, 2005

References II


7. Workload Characterization Using the TAU Performance System.


11. Review of Performance Analysis Tools for MPI Parallel Programs, Shirley Moore, David Cronk, Kevin London, Jack Dongarra

12. NAS Parallel Benchmarks version 2.4, Rob F. Van der Wijngaart, Computer Sciences Corporation, NASA Advanced Supercomputing (NAS) Division.
Thank you for your attention!

Questions?

Merci de votre attention!