Capturing Performance Knowledge for Automated Analysis

Kevin A. Huck\textsuperscript{1}, Oscar Hernandez\textsuperscript{2}, Van Bui\textsuperscript{2}, Sunita Chandrasekaran\textsuperscript{3}, Barbara Chapman\textsuperscript{2}, Allen D. Malony\textsuperscript{1}, Lois Curfman McInnes\textsuperscript{4}, Boyana Norris\textsuperscript{4}

\textsuperscript{1}University of Oregon
\textsuperscript{2}University of Houston
\textsuperscript{3}Nanyang Technological University
\textsuperscript{4}Argonne National Laboratory

SC’08 – Austin, TX – Nov. 20, 2008
Objectives

• To capture and automate performance analysis process and higher level reasoning (meta-analysis)
  – Design flexible analysis components and usable interfaces for integration
  – Engage the parallel programming and tuning environments to use knowledge-based analysis automation capabilities

• Make this available for other problem solving scenarios
Motivation

• Parallel performance analysis is complicated and intimidating
  – Management of multi-experiment performance data
  – Application of multi-step processes can introduce errors if done manually

• Lack of support for automation translates to loss of knowledge
  – Which analysis methods are useful for each performance problem type
  – How performance models are obtained and validated
  – How to interpret performance results relative to opportunities for optimization
Application of Analysis Automation

- **Application**: provide runtime performance data to the OpenUH compiler to improve analysis for optimization (for time, efficiency, power)
- **Long term goal**: to improve cost model computation for auto-parallelizing code with feedback-based optimization
  - Loop Nest Optimization (LNO)
- **Medium term goal**: to improve OpenMP performance with feedback-based optimization
- **Short term goal**: capture expertise from hand-optimized application code as re-usable analysis process
PerfExplorer 2.0

- Data mining framework for parallel profile performance data and metadata
- Programmable, extensible workflow automation
- Rule-based inference for expert system analysis
Automation & Knowledge Engineering

Analysis Components:
- Correlation
- Derive Metric
- Difference Extractions
- K-Means
- Smart K-Means
- Linear Regression
- Log Transform
- Merge Trials
- PCA
- Scale Metric
- Split
- Process Rules
- Save
- Draw Chart
OpenUH Compiler

- C, C++, Fortran95 compiler
- Complete support for OpenMP 2.5
- Front end, IPA and middle/back end:
  - Loop nest optimizer (LNO)
  - Auto parallelizer (with an OpenMP module)
  - Global optimizer (WOPT)
  - Code generator (CG)

- Each module supports feedback-directed optimizations*
OpenUH Cost Model

• Some optimization guided by cost model
  – Loop Nest Optimizer:
    • Processor model
    • Cache model
    • Parallel overhead model

• Cost model computed with static information (and control-flow feedback)

• Long term goal: improve the cost model accuracy using runtime analysis feedback
OpenUH & PerfExplorer Integration

Source Code → OpenUH → Instrumented, Compiled Application → TAU Profiles

Current

User Recommendations → Analysis Results → PerfExplorer

Future

Analysis Scripts → Inference Rules

PerfDMF
Example #1 – Multiple String Alignment (MSA)

- Compare protein sequences with unknown function to sequences with known function
- Widely used heuristic: progressive alignment (Smith-Waterman)
  - Compute a pairwise distance matrix (90% of time spent here)
  - Construct a guide tree
  - Progressive alignment along the tree
- OpenMP parallelism did not scale well
MSA – OpenMP Load Imbalance

```c
#pragma omp for
for (m=first; m<=last; m++) {
    for (n=m+1; n<=last; n++) {
        ...
    }
}
```
MSA – Improved Scaling

```c
#pragma omp for schedule (dynamic,1) nowait
```

- Before: efficiency < 70% with 16 processors, 400 sequence set
- After: efficiency > 92.5% with 16 processors, 400 sequence set
- Efficiency ~= 80% with 128 processors, 1000 sequence set
Analysis Workflow, Inference Rules

for each instrumented region:
- compute mean, stddev across all threads
- compute, assert stddev/mean ratio
- correlate region against all other regions
- assert correlation
- assert “severity” of event (exclusive time)

Rule1: IF severity(r) > 0.05 AND ratio(r) > 0.25
THEN alert(“load imbalance: r1”) AND assert imbalanced(r)

Rule2: IF imbalanced(r1) AND imbalanced(r2) AND calls (r1,r2) AND correlation(r1,r2) < -0.5
THEN alert(“new schedule suggested: r1, r2”)
Example output

--------------- PerfExplorer test script start ---------------
--- Looking for load imbalances ---
Loading Rules... Reading rules: openuh/OpenUHRules.drl... done.
loading the data... Main Event: main
Firing rules...

The event LOOP #3 [file:/mnt/netapp/home1/khuck/openuh/src/fpga/msap.c <63, 163>] has a high load imbalance for metric P_WALL_CLOCK_TIME
Mean/Stddev ratio: 0.667, Stddev actual: 6636425.1875
Percentage of total runtime: 27.15%

The event LOOP #2 [file:/mnt/netapp/home1/khuck/openuh/src/fpga/msap.c <65, 158>] has a high load imbalance for metric P_WALL_CLOCK_TIME
Mean/Stddev ratio: 0.260, Stddev actual: 1.74530281875E7
Percentage of total runtime: 71.40%

LOOP #3 [file:/mnt/netapp/home1/khuck/openuh/src/fpga/msap.c <63, 163>] calls LOOP #2 [file:/mnt/netapp/home1/khuck/openuh/src/fpga/msap.c <65, 158>], and they are both showing signs of load imbalance.

If these events are in an OpenMP parallel region, consider methods to balance the workload, such as dynamic instead of static work assignment.

...done with rules.
--------------- PerfExplorer test script end ---------------

✓ Rule1 true!

✓ Rule1 true!

✓ Rule2 true!
Example #2 – GenIDLEST

• **Generalized Incompressible Direct and Large-Eddy Simulations of Turbulence**
• Overlapping multi-block body-fitted structured mesh topology, and unstructured inter-block topology
• SPMD parallelism, using MPI and/or OpenMP
• Test cases: investigate turbine cooling duct, 45 and 90 degree ribs
  – Detached Eddy Simulations (45)
  – Large Eddy Simulations (90)
Problems mainly related to remote memory references on NUMA architecture, excessive memory copies initiated by master thread.
Analysis Workflow, Inference Rules

for each instrumented region, exclusive:
  derive, assert inefficiency metric
  derive, assert memory/total stall cycles metric
  derive, assert memory cycles metric
  derive, assert remote memory accesses ratio metric
  assert “severity” of event
also compute values for main, inclusive

Rule1: IF severity(r) > 0.02 AND inefficiency(r) > inefficiency(main)
      THEN alert (“inefficient, r”) AND assert (inefficient(r))

Rule2: IF inefficient(r) AND tsm(r) > 0.9
      THEN alert (“memory stalls, r”) AND assert (memstall(r))

Rule3: IF memstall(r) AND memory(r) > memory(main)
      THEN alert (“memory cycles, r”)

Rule4: IF memstall(r) AND remote(r) > remote(main)
      THEN alert (“remote references, r”)
Example output

Firing rules...

The event *exchange_var* has a higher than average stall / cycle rate

Average stalls per cycle: 0.79877, Event stalls per cycle: 0.95439

Percentage of total runtime: 31.16% ✓ Rule1 true!

... The event *exchange_var* has a high percentage of stalls due to L1 data cache misses and FP Stalls.

Percent of Stalls due to these two reasons: 99.88% ✓ Rule2 true!

... The event *exchange_var* has a higher than average number of cycles handling memory references.

Average memory cycles: 73.72%, Event memory cycles: 100.09% ✓ Rule3 true!

... The event *bicgstab* has a lower than average local memory reference percentage. If this is an OpenMP parallel region, consider methods for parallelizing data initialization.

Average percentage: 93.77%, Event ratio: 90.44% ✓ Rule4 true!

...done with rules.

-------------- JPython test script end --------------
Example #3 – Power Estimation

• May want to optimize for metric other than time
• Hardware counter data can be used to estimate power consumption
• Simplified model – Itanium2:

\[
\begin{align*}
\text{CPU} &= \left( \frac{\text{instructions}}{\text{cycles}} \right) \times 0.0459 \times 122 \\
\text{L1} &= \left( \frac{\text{L1 references}}{\text{cycles}} \right) \times 0.0017 \times 122 \\
\text{L2} &= \left( \frac{\text{L2 references}}{\text{cycles}} \right) \times 0.0171 \times 122 \\
\text{L3} &= \left( \frac{\text{L3 references}}{\text{cycles}} \right) \times 0.935 \times 122
\end{align*}
\]

TOTAL = CPU + L1 + L2 + L3
### Power Estimation – Results

<table>
<thead>
<tr>
<th>Metric</th>
<th>-O0</th>
<th>-O1</th>
<th>-O2</th>
<th>-O3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>1.0</td>
<td>0.338</td>
<td>0.071</td>
<td>0.049</td>
</tr>
<tr>
<td>Instructions Completed</td>
<td>1.0</td>
<td>0.471</td>
<td>0.059</td>
<td>0.056</td>
</tr>
<tr>
<td>Instructions Issued</td>
<td>1.0</td>
<td>0.472</td>
<td>0.063</td>
<td>0.061</td>
</tr>
<tr>
<td>Instructions Completed Per Cycle</td>
<td>1.0</td>
<td>1.397</td>
<td>0.857</td>
<td>1.209</td>
</tr>
<tr>
<td>Instructions Issued Per Cycle</td>
<td>1.0</td>
<td>1.400</td>
<td>0.909</td>
<td>1.316</td>
</tr>
<tr>
<td>Power Consumed (Watts)</td>
<td>1.0</td>
<td>1.025</td>
<td>1.001</td>
<td>1.029</td>
</tr>
<tr>
<td>Energy Consumed (Joules)</td>
<td>1.0</td>
<td>0.346</td>
<td>0.071</td>
<td>0.050</td>
</tr>
<tr>
<td>FLOP/Joule</td>
<td>1.0</td>
<td>2.867</td>
<td>13.684</td>
<td>19.305</td>
</tr>
</tbody>
</table>
Future Work

• Modify cost model calculation to integrate feedback from runtime data analysis
• Feed information about sources of overhead and causes to OpenMP infrastructure
• Implement strategies for variable privatization and first touch policies
• Parallel model could be improved for auto-parallelized code
• Optimizations for performance and power
Conclusion

• Initial work into capturing analysis process
• Automation and expert knowledge to direct processing, interpret results, and provide decision support
• Flexible scripting, rule-based system is reusable, extensible to other analysis scenarios
Acknowledgements

• US Department of Energy (DOE)
  – Office of Science
• US National Science Foundation (NSF)
• Argonne National Lab
• NASA / CSC (Altix 300)
• NCSA (Altix 4700)
• Virginia Tech (GenIDLEST application)