Performance Analysis and Optimization Tool

Andres S. CHARIF-RUBIAL

andres.charif@uvsq.fr
Performance Analysis Team, University of Versailles
http://www.maqao.org
Introduction
Performance Analysis

- Develop performance analysis and optimization tools: MAQAO Framework and Toolsuite

- Establish partnerships

- Optimize industrial applications
Introduction
Performance Analysis

- Understand the performance of an application
  - How well it behaves on a given machine
- What are the issues?
- Generally a multifaceted problem
  - Maximizing the number of views = better understand
- Use techniques and tools to understand issues
- Once understood → Optimize application
Compiler remains your best friend

- Be sure to select proper flags (e.g., -xavx)

- Pragmas: Unrolling, Vector alignment

- O2 V.S. O3

- Vectorisation/optimisation report
Introduction

MAQAO Tool

- Open source (LGPL 3.0)
  - Currently binary release

- Available for x86-64 and Xeon Phi
  - Looking forward in porting MAQAO on BlueGene
Easy install

- Packaging: ONE (static) standalone binary
  - Easy to embed

Audience

- User/Tool developer: analysis and optimisation tool
- Performance tool developer: framework services
  - TAU: tau_rewrite (MIL)
  - ScoreP: on-going effort (MIL)
Introduction
MAQAO Tool

Binary Manipulation Layer
- Disassembler Generator
  - Disassemble
  - Re-assemble
  - Patch/Rewrite

Analysis Layer
- Functions
- Loops
- Instructions
- Basic blocks
- Demangling
- Debug symbols
- Other analysis algorithms (SSA, Dominance, ...)

MAQAO Lua Plugins
- API bindings to low-level layers
  - STAN
  - MTL
  - MIL
  - Profiler
Introduction

MAQAO Tool

Scripting language

- Lua language: simplicity and productivity
- Fast prototyping
- MAQAO Lua API: Access to services
Introduction

MAQAO Tool

- Built on top of the Framework
- Loop-centric approach
- Produce reports
  - We deal with low level details
  - You get high level reports
A lot of tools! Which one to use? When

Our approach/experience: decision tree

Currently working on HPC
- Multi-node > Node > Socket > Core
- Classify IO/Memory/MPI/OpenMP/Application

PAMDA methodology
- to be published: 7th Parallel Tools Workshop
- https://tools.zih.tu-dresden.de/2013/
Outline

- Introduction
- Pinpointing hotspots
  - Functions, loops
  - MPI characterization
- Code quality analysis
Pinpointing hotspots
Measurement methodology

- MAQAO Profiling
  - Instrumentation
    - Through binary rewriting
    - High overhead / More precision

- Sampling
  - Hardware counter through perf_event_open system call
  - Very low overhead / less details
Pinpointing hotspots
Parallelism level

- **SPMD**
  - Program level

- **SIMD**
  - Instruction level

- By default MAQAO only considers system processes and threads
Pinpointing hotspots

Display

- Display functions and their exclusive time
  - Associated callchains and their contribution
  - Loops

- Innermost loops can then be analyzed by the code quality analyzer module (CQA)

- Command line and GUI (HTML) outputs
<table>
<thead>
<tr>
<th>Name</th>
<th>Median Excl %Time</th>
<th>Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>matmul__sub__ - <a href="mailto:56@solve_subs.f">56@solve_subs.f</a></td>
<td>17.16</td>
<td>0.26</td>
</tr>
<tr>
<td>compute_rhs__ - <a href="mailto:4@rhs.f">4@rhs.f</a></td>
<td>10</td>
<td>0.03</td>
</tr>
<tr>
<td>y_solve_cell__ - <a href="mailto:385@y_solve.f">385@y_solve.f</a></td>
<td>9.32</td>
<td>0.54</td>
</tr>
<tr>
<td>z_solve_cell__ - <a href="mailto:385@z_solve.f">385@z_solve.f</a></td>
<td>8.96</td>
<td>0.14</td>
</tr>
<tr>
<td>x_solve_cell__ - <a href="mailto:391@x_solve.f">391@x_solve.f</a></td>
<td>8.68</td>
<td>0.17</td>
</tr>
<tr>
<td>MPI_DCH3i_Progress</td>
<td>5.22</td>
<td>3.66</td>
</tr>
<tr>
<td>matvec_sub__ - <a href="mailto:5@solve_subs.f">5@solve_subs.f</a></td>
<td>3.92</td>
<td>0.11</td>
</tr>
<tr>
<td>x_backsSubstitute__ - <a href="mailto:330@x_solve.f">330@x_solve.f</a></td>
<td>3.09</td>
<td>0.14</td>
</tr>
<tr>
<td>y_backsSubstitute__ - <a href="mailto:329@y_solve.f">329@y_solve.f</a></td>
<td>2.05</td>
<td>0.03</td>
</tr>
<tr>
<td>z_backsSubstitute__ - <a href="mailto:329@z_solve.f">329@z_solve.f</a></td>
<td>1.98</td>
<td>0.06</td>
</tr>
<tr>
<td>copy_faces__ - <a href="mailto:4@copy_faces.f">4@copy_faces.f</a></td>
<td>0.88</td>
<td>0.06</td>
</tr>
<tr>
<td>MPID nem dapi_rc_poll_dyn_opt</td>
<td>0.74</td>
<td>0.62</td>
</tr>
<tr>
<td>MPID nem int shm_start_send</td>
<td>0.68</td>
<td>0.06</td>
</tr>
</tbody>
</table>
Pinpointing hotspots
GUI snapshot (2/4)

Performance Evaluation - Profiling results

Hotspots - Functions

| exact_solution__ - 4@exact_solution.f | 0.21 | 0.03 |
| x_unpack_solve_info__ - 114@x_solve.f | 0.14 | 0.03 |
### Pinpointing hotspots

GUI snapshot (3/4)

<table>
<thead>
<tr>
<th>Name</th>
<th>Excl %Time</th>
<th>Excl Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>matmul_sub__ - 56@solve_subsf</td>
<td>16.92</td>
<td>16.48</td>
</tr>
<tr>
<td>compute_rhs__ - <a href="mailto:4@rhs.f">4@rhs.f</a></td>
<td>9.92</td>
<td>9.66</td>
</tr>
<tr>
<td>y_solve_cell__ - <a href="mailto:385@y_solve.f">385@y_solve.f</a></td>
<td>9.08</td>
<td>8.84</td>
</tr>
<tr>
<td>loops</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop 267 - y_solve.f@415</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Loop 268 - y_solve.f@425</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Loop 272 - y_solve.f@426</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>Loop 270 - y_solve.f@524</td>
<td>6.57</td>
<td></td>
</tr>
<tr>
<td>Loop 271 - y_solve.f@436</td>
<td>2.22</td>
<td></td>
</tr>
<tr>
<td>Loop 269 - y_solve.f@716</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>x_solve_cell__ - <a href="mailto:391@x_solve.f">391@x_solve.f</a></td>
<td>9.01</td>
<td>8.78</td>
</tr>
<tr>
<td>loops</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop 235 - x_solve.f@420</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Loop 236 - x_solve.f@429</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Loop 237 - x_solve.f@709</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>Loop 239 - x_solve.f@431</td>
<td>2.71</td>
<td></td>
</tr>
<tr>
<td>Loop 238 - x_solve.f@519</td>
<td>6.24</td>
<td></td>
</tr>
</tbody>
</table>
### Pinpointing hotspots

#### GUI snapshot (4/4)

#### cirrus5003 - Process #53572 - Thread #1

<table>
<thead>
<tr>
<th>Name</th>
<th>Excl %Time</th>
<th>Excl Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>matmul_sub__ - 56@solve_subsf</td>
<td>16.92</td>
<td>16.48</td>
</tr>
<tr>
<td>compute_rhs__ - <a href="mailto:4@rhs.f">4@rhs.f</a></td>
<td>9.92</td>
<td>9.66</td>
</tr>
<tr>
<td>y_solve_cell__ - <a href="mailto:385@y_solve.f">385@y_solve.f</a></td>
<td>9.08</td>
<td>8.84</td>
</tr>
<tr>
<td>loops</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop 267 - y_solve.f@415</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Loop 268 - y_solve.f@425</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Loop 272 - y_solve.f@426</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>Loop 270 - y_solve.f@524</td>
<td>6.57</td>
<td></td>
</tr>
<tr>
<td>Loop 271 - y_solve.f@436</td>
<td>2.22</td>
<td></td>
</tr>
<tr>
<td>Loop 269 - y_solve.f@716</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>x_solve_cell__ - <a href="mailto:391@x_solve.f">391@x_solve.f</a></td>
<td>9.01</td>
<td>8.78</td>
</tr>
<tr>
<td>loops</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop 235 - x_solve.f@420</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Loop 236 - x_solve.f@429</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Loop 237 - x_solve.f@709</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>Loop 239 - x_solve.f@431</td>
<td>2.71</td>
<td></td>
</tr>
<tr>
<td>Loop 238 - x_solve.f@519</td>
<td>6.24</td>
<td></td>
</tr>
</tbody>
</table>
Outline

- Introduction
- Pinpointing hotspots
  - Functions, loops
  - MPI characterization
- Code quality analysis
Our methodology

- Coarse grain: overview, global trends/patterns
- Fine grain: filtering precise issues

Tracing issues

- Scalability
- Memory size: can we reduce it?
- Trace size: can we reduce it?
- IO’s wall: remove it?
Pinpointing hotspots
MPI characterization: overview

- Scalable coarse grain analysis

[Diagram showing MPI APPLICATION, MAQAO, profile.js, and Web Visualizer flow]
Online profiling

- Aggregated metrics
- No traces
- No IOs (only one result file)
- Reduced memory footprint
- Scalable on 100+ procs
Pinpointing hotspots
MPI characterization: visualization

- Web based visualizer
- Only requires a web browser
MPI primitives high level profile (hits,time,size)
Function scattering over time

MPI Function Scattering over Time

- Stacked
- Stream
- Expanded

- MPI_Barrier
- MPI_Bcast
- MPI_Comm_split
- MPI_Recv
- MPI_Isend
- MPI_Reduce
- MPI_Waitall
- MPI_Wtime
Probability densities

**Probability Densities**

- **MPI_Isend**
- **Time Distribution**

- **Time in Seconds**
  - **min**: 2.91482e-7
  - **max**: 0.00023
  - **max**: 19.7279
  - **min**: 0
Pinpointing hotspots
MPI characterization: analyses (4/5)

- Topology view (1/2)

- 3D topology
- Information Panel
- Color scale
- Force driven topology view
- Selector
Pinpointing hotspots
MPI characterization: analyses (4/5)

➤ Topology view (2/2)

Click on a node in the force layout to display its information
Hover a node to see its MPI rank
Hover an edge to see its value
Pinpointing hotspots
MPI characterization: analyses (5/5)

- Communication matrix (1/3)
Pinpointing hotspots
MPI characterization: analyses (5/5)

➢ Communication matrix (2/3)
Pinpointing hotspots
MPI characterization: analyses (5/5)

- Communication matrix (3/3)
Pinpointing hotspots
MPI characterization: next steps

- Fine grained analyses should be:
  - Investigated using (MPI) tracing tools
  - And filtering on specific processes/events of interest detected thanks to this tool
Outline

- Introduction
- Pinpointing hotspots
  - Functions, loops
  - MPI characterization
- Code quality analysis
Main performance issues:
- Core level
- Multicore interactions
- Communications

Most of the time core level is forgotten
Static performance modeling

Goals

- Static performance model
  - Targets innermost loops
    - source loop V.S. assembly loop
  - Take into account processor (micro)architecture
- Assess code quality
  - Estimate performance
  - Degree of vectorization
  - Impact on micro architecture
Static performance modeling

Model

- Simulates the target (micro)architecture
  - Instructions description (latency, uops dispatch...)
  - Machine model

- For a given binary and micro-architecture, provides
  - Quality metrics (how well the binary is fitted to the micro architecture)
  - Static performance (lower bounds on cycles)
  - Hints and workarounds to improve static performance
Static performance modeling

Metrics

- Vectorization (ratio and speedup)
  - Allows to predict vectorization (if possible) speedup and increase vectorization ratio if it’s worth

- High latency instructions (division/square root)
  - Allows to use less precise but faster instructions like RCP (1/x) and RSQRT (1/sqrt(x))

- Unrolling (unroll factor detection)
  - Allows to statically predict performance for different unroll factors (find main loops)
Static performance modeling
Output example (1/2)

MAQAO Code quality analysis

Source loop ending at line 682

MAQAO binary loop id: 238

The loop is defined in MPI/BT/x_solve.f:519-682
15% of peak computational performance is used (1.23 out of 8.00 FLOP per cycle (GFLOPS @ 1GHz))

Gain Potential gain Hints Experts only

Vectorization

Your loop is processing FP elements but is NOT OR PARTIALLY VECTORIZED and could benefit from full vectorization.
By fully vectorizing your loop, you can lower the cost of an iteration from 190.00 to 60.75 cycles (3.13x speedup).
Since your execution units are vector units, only a fully vectorized loop can use their full power.

Proposed solution(s):
Two propositions:
- Try another compiler or update/tune your current one:
- Remove inter-iterations dependences from your loop and make it unit-stride.

Bottlenecks

By removing all these bottlenecks, you can lower the cost of an iteration from 190.00 to 143.00 cycles (1.33x speedup).

Source loop ending at line 734
Static performance modeling

Output example (2/2)

Code quality analysis

Source loop ending at line 682

MAQAO binary loop id: 238

The loop is defined in MPI/BT/x_solve.f:519-682
15% of peak computational performance is used (1.23 out of 8.00 FLOP per cycle (GFLOPS @ 1GHz))

Gain Potential gain Hints Experts only

Type of elements and instruction set

234 SSE or AVX instructions are processing arithmetic or math operations on double precision FP elements in scalar mode (one at a time).

Vectorization status

Your loop is probably not vectorized (store and arithmetical SSE/AVX instructions are used in scalar mode and, for others, at least one is in vector mode).

Only 28% of vector length is used.

Matching between your loop (in the source code) and the binary loop

The binary loop is composed of 234 FP arithmetical operations:
- 95: addition or subtraction
- 139: multiply

The binary loop is loading 1600 bytes (200 double precision FP elements).
The binary loop is storing 616 bytes (77 double precision FP elements).

Arithmetic intensity

Arithmetic intensity is 0.11 FP operations per loaded or stored byte.
Thanks for your attention!

Questions?