MAQAO
Performance Analysis and Optimization Tool

Cédric Valensi, Emmanuel Oseret
cedric.valensi@uvsq.fr, emmanuel.oseret@uvsq.fr
Performance Evaluation Team, University of Versailles
http://www.maqao.org
Performance Analysis and Optimisation

➢ **How much** can I optimise my application?
  • Can it actually be done?
  • What would the effort/gain ratio be?

➢ **Where** can I gain time?
  • Where is my application wasting time?

➢ **Why** is the application spending time there?
  • Algorithm, implementation or hardware?
  • Data access or computation?

➢ **How** can I improve the situation?
  • In which step(s) of the design process?
  • What additional information do I need?
A Multifaceted Problem

- **Pinpointing** the performance bottlenecks

- **Identifying** the dominant issues
  - Algorithms, implementation, parallelisation, …

- Making the **best use** of the machine features
  - Complex multicore and manycore CPUs
  - Complex memory hierarchy

- Finding the **most rewarding** issues to be fixed
  - 40% total time, expected **10%** speedup
    - TOTAL IMPACT: **4%** speedup
  - 20% total time, expected **50%** speedup
    - TOTAL IMPACT: **10%** speedup

=> Need for dedicated and complementary tools

MAQAO Performance Analysis and Optimization Tool
Motivating Example

MAQAO Performance Analysis and Optimization Tool

Code of a loop representing ~10% walltime

```plaintext
do j = ni + nvalue1, nato

nj1 = ndim3d*j + nc ; nj2 = nj1 + nvalue1 ; nj3 = nj2 + nvalue1
u1 = x11 - x(nj1) ; u2 = x12 - x(nj2) ; u3 = x13 - x(nj3)
rttest2 = u1*u1 + u2*u2 + u3*u3 ; cnij = eci*qEold(j)
rij = demi*(rvwi + rvwalc1(j))
drtest2 = cnij/(rttest2 + rij) ; drtest = sqrt(drtest2)
Eq = qq1*qq(j)*drtest
ntj = nti + ntype(j)
Ed = ceps(ntj)*drtest2*drtest2*drtest2
Eqc = Eqc + Eq ; Ephob = Ephob + Ed
gE = (c6*Ed + Eq)*drtest2 ; virt = virt + gE*rttest2
u1g = u1*gE ; u2g = u2*gE ; u3g = u3*gE
g1c = g1c - u1g ; g2c = g2c - u2g ; g3c = g3c - u3g
gr(nj1, thread_num) = gr(nj1, thread_num) + u1g
gr(nj2, thread_num) = gr(nj2, thread_num) + u2g
gr(nj3, thread_num) = gr(nj3, thread_num) + u3g

end do
```

Source code and associated issues:

1) High number of statements
2) Non-unit stride accesses
3) Indirect accesses
4) DIV/SQRT
5) Reductions
6) Variable number of iterations
MAQAO: Modular Assembly Quality Analyzer and Optimizer

➢ Objectives:
  • Characterizing performance of HPC applications
  • Focusing on performance at the core level
  • Guiding users through optimization process
  • Estimating return of investment (R.O.I.)

➢ Characteristics:
  • Modular tool offering complementary views
  • Support for Intel x86-64 and Xeon Phi
    ▪ ARM under development
  • LGPL3 Open Source software
  • Developed at UVSQ since 2004
  • Binary release available as static executable
QMC=CHEM (IRSAMC)
- Quantum chemistry
- Speedup: > 3x
  - Moved invocation of function with identical parameters out of loop body

Yales2 (CORIA)
- Computational fluid dynamics
- Speedup: up to 2.8x
  - Removed double structure indirections

Polaris (CEA)
- Molecular dynamics
- Speedup: 1.5x – 1.7x
  - Enforced loop vectorisation through compiler directives

AVBP (CERFACS)
- Computational fluid dynamics
- Speedup: 1.08x – 1.17x
  - Replaced division with multiplication by reciprocal
  - Complete unrolling of loops with small number of iterations
2004: Begun development
  • Focusing on Intel Itanium architecture
  • Analysis of assembly files
2006: Transition to Intel x86-64
2009: Binary analysis support
2010: First version of decremental analysis
2012: Support of KNC architecture
2014: Profiling features
2015: First version of ONE View
2017: Prototype support of ARM architecture
MAQAO was funded by UVSQ, Intel and CEA (French department of energy) through Exascale Computing Research (ECR) and the French Ministry of Industry through various FUI/ITEA projects (H4H, COLOC, PerfCloud, ELCI, MB3, etc...)

Provides core technology to be integrated with other tools:
- TAU performance tools with MADRAS patcher through MIL (MAQAO Instrumentation Language)
- ATOS bullxprof with MADRAS through MIL
- Intel Advisor
- INRIA Bordeaux HWLOC

PeXL ISV also contributes to MAQAO:
- Commercial performance optimization expertise
- Training and software development
Advantages of binary analysis:
- Compiler optimizations increase the distance between the executed code and the source
- Source code instrumentation may prevent the compiler from applying some transformations

We want to evaluate the “real” executed code: **What You Analyse Is What You Run**

Main steps:
- Reconstruct the program structure
- Relate the analyses to source code
  - A single source loop can be compiled as multiple assembly loops
  - Affecting unique identifiers to loops
MAQAO Main Features

➢ **Binary layer**
  • Builds internal representation from binary
  • Allows patching through binary rewriting

➢ **Profiling**
  • LProf: Lightweight sampling-based profiler
  • VProf: Instrumentation-based value profiler

➢ **Static analysis**
  • CQA (Code Quality Analyzer): Evaluates the quality of the binary code and offers hints for improving it
  • UFS (Uops Flow Simulator): Cycle-accurate CPU engine simulator

➢ **Dynamic analysis**
  • DECAN (DECremental Analyzer): Modifies the application to evaluate the impact of groups of instructions on performance

➢ **Performance view aggregation module**
  • ONE View: Invokes the modules and produces reports aggregating their results
MAQAO Main Structure

- Application
  - Disassembly
  - Analysis
- Patching
- Lua API
  - LProf + Sampling
  - CQA + Machine model
  - VProf
  - DECAN
- Reports
  - Loop 42 50% vectorised Potential x1.2 speedup
- ONE View

MAQAO Performance Analysis and Optimization Tool
MAQAO Methodology

➢ Decision tree

- Profiling
  - Loops of interest
  - Analysis

- CPU oriented
  - Code Quality Analysis
  - Differential analysis
  - Value Profiling

- Data access oriented
  - Data access characterization
  - Differential analysis
Goal: Lightweight localization of application hotspots

Features:
- **Sampling** based
- Access to hardware counters for additional information
  - Can also access OS timers for unsupported architectures
- Results at function and loop granularity

Strengths:
- **Non intrusive**: No recompilation necessary
- **Low overhead**
- Agnostic with regard to parallel runtime
Goal: Assist developers in improving code performance

Features:

• Evaluates the quality of the compiler generated code
• Returns hints and workarounds to improve quality
• Focuses on loops
  ▪ In HPC most of the time is spent in loops
• Targets compute-bound codes

Static analysis:

• Requires no execution of the application
• Allows cross-analysis
Relies on simplified CPU model
- Allows faster analyses
- More precise but slower analyses available with UFS

Machine model:
- Execution pipeline
- Port throughput
- L1 data access
- Buffers ignored if not UFS

Key performance levers for core level efficiency:
- Vectorising
- Avoiding high latency instructions if possible
- Having the compiler generate an efficient code
- Reorganizing memory layout
Compiler can be driven using flags and pragmas:

- Ensuring full use of architecture capabilities (e.g., using flag -xHost on AVX capable machines)
- Forcing optimization (unrolling, vectorization, alignment…)
- Bypassing conservative behaviour when possible (e.g., 1/X precision)

Implementation changes

- Improve data access
  - Loop interchange
  - Change loop stride
  - Reshaping arrays of structures
- Avoid instructions with high latency
MAQAO CQA Application to Motivating Example

Issues identified by CQA

CQA can detect and provide hints to resolve most of the identified issues:

1) High number of statements
2) Non-unit stride accesses
3) Indirect accesses
4) DIV/SQRT
5) Reductions
6) Variable number of iterations
7) Vector vs scalar

```plaintext
do j = ni + nvalue1, nato
    nj1 = ndim3d*j + nc ; nj2 = nj1 + nvalue1 ; nj3 = nj2 + nvalue1
    u1 = x11 - x(nj1) ; u2 = x12 - x(nj2) ; u3 = x13 - x(nj3)
    rtest2 = u1*u1 + u2*u2 + u3*u3 ; cnij = eci*qEold(j)
    rij = demi*(rvwi + rvwalc1(j))
    dtest2 = cnij/(rtest2 + rij) ; dtest2 = sqrt(dtest2)
    Eq = qq1*qq(j)*dtest
    ntj = nti + ntype(j)
    Ed = ceps(ntj)*dtest2*dtest2*dtest2
    Eqc = Eqc + Eq ; Ephob = Ephob + Ed
    gE = (c6*Ed + Eq)*dtest2 ; virt = virt + gE*rtest2
    u1g = u1*gE ; u2g = u2*gE ; u3g = u3*gE
    g1c = g1c - u1g ; g2c = g2c - u2g ; g3c = g3c - u3g
    gr(nj1, thread_num) = gr(nj1, thread_num) + u1g
    gr(nj2, thread_num) = gr(nj2, thread_num) + u2g
    gr(nj3, thread_num) = gr(nj3, thread_num) + u3g
end do
```
MAQAO CQA Application to motivating example

1) High number of statements
2) Non-unit stride accesses
3) Indirect accesses
4) DIV/SQRT
5) Reductions
6) Variable number of iterations
7) Vector vs scalar
Goal: modify the application to
• Identify cause of bottlenecks
• Estimate associated ROI

Differential analysis:
• Targets innermost loops
• Transforms loops
• Measure and compare performance of original and transformed copy

Transformations
• Remove or modify groups of instructions
• Targets memory accesses or computation
Typical transformations:

- **FP**: only FP arithmetic instructions are preserved
  - => loads and stores are removed

- **LS**: only loads and stores are preserved
  - => compute instructions are removed

- **DL1**: memory references replaced with global variables ones
  - => data now accessed from L1
MAQAO DECAN Example

MAQAO Performance Analysis and Optimization Tool
- ROI = FP / LS = 4.1
- Imbalance between the two streams
  => Try to consume more elements inside one iteration.
MAQAO DECAN
Application to Motivating Example

REF_NSD: removing DIV/SQRT instructions provides a 1.5 x speedup
  => the bottleneck is the presence of these DIV/SQRT instructions
FPLS_NSD: removing loads/stores after DIV/SQRT provides a small additional speedup
Conclusion: No room left for improvement here (algorithm bound)
Value profiling
- Targets loops or functions
- Instrumentation
- Iteration count, loop path uses, function parameters, …

Metrics
- Detection of stable values
- Loop characterisation through number of iterations

Provides leads for code specialisation
Goal: **Automating** the whole analysis process
- Invocation of the required MAQAO modules
- Generation of **aggregated performance views** as HTML or XLS files

Report levels of increasing analysis complexities
- Each level includes the analyses of the levels below it
- An experiment directory can be reused for generating a higher level report
ONE View Reports Levels

➢ ONE VIEW ONE
  • Requires a single run of the application
  • Profiling of the application using LProf
  • Static analysis using CQA

➢ ONE VIEW TWO (includes analyses from report ONE)
  • Requires 3 or 4 runs on average
  • Value profiling using VProf to identify loop iteration count
  • Decremental analysis for L1 projection using DECAN

➢ ONE VIEW THREE (includes analyses from report TWO)
  • Requires 20 to 30 runs
  • Decremental analyses using all DECAN variants
  • Collects hardware performance events

➢ Scalability
  • Require as many additional runs as parallel configurations
  • Can be executed in addition of another report
  • Profilings using LProf on different parallel configurations
Main steps:
- Invokes LProf to identify hotspots
- Invokes CQA, VPROF and DECAN on loop hotspots

Available results:
- **Speedup** predictions
- Global code **quality** metrics
- **Hints** for improving performance
- Detailed analyses results
- Parallel efficiency
Experiment summary
• Characteristics of the machine where the experiment took place

Global metrics
• General quality metrics derived from MAQAO analyses
• Global speedup predictions
  ▪ Speedup prediction depending on the number of vectorised loops
  ▪ Ordered speedups to identify the loops to optimise in priority
Global metrics
- General quality metrics derived from MAQAO analyses
- Global speedup predictions

Potential speedups
- Speedup prediction depending on the number of optimised loops
- Ordered speedups to identify the loops to optimise in priority

\[ \text{Global Speedup} = \sum_{\text{loops}} \text{coverage} \times \text{potential speedup} \]

- LProf provides coverage of the loops
- CQA and DECAN provide speedup estimation for loops
  - Speedup if loop vectorised or without address computation
  - All data in L1 cache
MAQAO ONE View Application Characteristics

- Application categorisation
  - Time spent in different regions of code

- Function based profile
  - Functions by coverage ranges

- Loop based profile
  - Loops by coverage ranges

- Detailed loop based profile
  - Loop types by coverage ranges
- Goal: allowing to identify at a glance where time is spent
  - Categories based on functions or libraries names

- Application
  - Main executable

- Parallelization
  - Threads
  - OpenMP
  - MPI

- System libraries
  - I/O operations
  - String operations
  - Memory management functions (allocation, free)

- External libraries
  - Specialised libraries such as libm / libmkl
  - Application code in external libraries
Identifying hotspots

- Exclusive coverage
- Load balancing across threads
- Loops nests by functions

MAQAO ONE View: Functions Profiling

MAQAO Performance Analysis and Optimization Tool

MAQAO thread rank

```
matmul_sub
- Loop 230 - solve_sub.f:71-175 - bt-mz_C.16
- Loop 231 - solve_sub.f:71-175 - bt-mz_C.16

z_solve
- Loop 232 - z_solve.f:53-423 - bt-mz_C.16
- Loop 233 - z_solve.f:54-423 - bt-mz_C.16
- Loop 236 - z_solve.f:54-423 - bt-mz_C.16
- Loop 239 - z_solve.f:146-308 - bt-mz_C.16
- Loop 235 - z_solve.f:55-137 - bt-mz_C.16
- Loop 234 - z_solve.f:415-423 - bt-mz_C.16
```
**Identifying loop hotspots**

- Vectorisation information
- Potential speedups by optimisation
  - **Clean**: Removing address computations
  - **FP Vectorised**: Vectorising floating-point computations
  - **Fully Vectorised**: Vectorising floating-point computations and memory accesses

---

### Loops Index

<table>
<thead>
<tr>
<th>Loop Id</th>
<th>Source Location</th>
<th>Source Function</th>
<th>Coverage (%)</th>
<th>Level</th>
<th>Time (s)</th>
<th>Vectorization Ratio (%)</th>
<th>Speedup if Clean</th>
<th>Speedup if FP Vectorized</th>
<th>Speedup if Fully Vectorized</th>
<th>Speedup if Data in L1</th>
<th>Speedup if Data in L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>26831</td>
<td>qmcplus: ParticleBConds3DS oah.231-262</td>
<td>qmcplus: ScaDistanceTableAA: moveOnSphere</td>
<td>21.06</td>
<td>Single</td>
<td>2.54</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.05</td>
</tr>
<tr>
<td>19042</td>
<td>qmcplus: MultiB splineValue.h pp.55-57</td>
<td>qmcplus: BsplineSet &gt;: evaluate</td>
<td>15.42</td>
<td>Innermost</td>
<td>1.86</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2.6</td>
</tr>
<tr>
<td>19064</td>
<td>qmcplus: MultiB splineValue.h pp.187-207</td>
<td>qmcplus: BsplineSet &gt;: evaluate</td>
<td>9.45</td>
<td>Innermost</td>
<td>1.14</td>
<td>100</td>
<td>1.03</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2.4</td>
</tr>
<tr>
<td>13028</td>
<td>qmcplus: BsplineFunctor.h9.643</td>
<td>qmcplus: J2OrbitaSoA &gt;: ratio</td>
<td>4.64</td>
<td>Innermost</td>
<td>0.56</td>
<td>0</td>
<td>1.38</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>19116</td>
<td>qmcplus: MultiB splineValue.h pp.187-207</td>
<td>qmcplus: BsplineSet &gt;: evaluate; notranspose</td>
<td>4.31</td>
<td>Innermost</td>
<td>0.52</td>
<td>100</td>
<td>1.03</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3.79</td>
</tr>
<tr>
<td>26829</td>
<td>qmcplus: ParticleBConds3DS oah.231-262</td>
<td>qmcplus: ScaDistanceTableAA: evaluate</td>
<td>3.98</td>
<td>Single</td>
<td>0.48</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.98</td>
</tr>
<tr>
<td>26833</td>
<td>qmcplus: ParticleBConds3DS oah.231-262</td>
<td>qmcplus: ScaDistanceTableAA: move</td>
<td>3.32</td>
<td>Single</td>
<td>0.4</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>19142</td>
<td>qmcplus: SplineC2RAdaptor.h 323-373</td>
<td>qmcplus: Spline2RSoA::assign_vgl; qmcplus::Vector, std::allocator:: &gt; &gt;</td>
<td>2.49</td>
<td>Single</td>
<td>0.3</td>
<td>56.36</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.78</td>
<td>1.51</td>
</tr>
<tr>
<td>19038</td>
<td>qmcplus: SplineC2RAdaptor.h 224-234</td>
<td>qmcplus: BsplineSet &gt;: evaluate</td>
<td>1.33</td>
<td>Single</td>
<td>0.16</td>
<td>48.39</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.69</td>
<td>1.66</td>
</tr>
</tbody>
</table>
High level reports

➢ Reference to the source code
➢ Bottleneck description
➢ Hints for improving performance
➢ Reports categorized by probability that applying hints will yield predicted gain
  • Gain: Good probability
  • Potential gain: Average probability
  • Hints: Lower probability
MAQAO ONE View Loop Analysis Reports
Expert View

- Low level reports for performance experts
  - Assembly-level
  - Instructions cycles costs
  - Instructions dispatch predictions
  - Memory access analysis
- Assembly code
  - Highlights groups of instructions accessing the same memory addresses
- CQA internal metrics
MAQAO ONE View Thread/Process View

- Software Topology
  - Nodes list
  - Processes by node
  - Thread by process

- View by thread
  - Function profile at the thread or process level

MAQAO Performance Analysis and Optimization Tool
Goal: Provide a view of the application scalability

- Profiles with different numbers of threads/processes
- Displays efficiency metrics for application
MAQAO ONE View Scalability Reports
Application View

➢ Coverage per category
  • Comparison of categories for each run

➢ Coverage per parallel efficiency
  • \( \text{Efficiency} = \frac{T_{\text{sequential}}}{T_{\text{parallel}} \times N_{\text{threads}}} \)
    ▪ Distinguishing functions only represented in parallel or sequential
  • Displays efficiency by coverage
MAQAO ONE View Scalability Reports

Functions and Loops Views

Displays metrics for each function/loop

- Efficiency
- Potential speedup if efficiency=1

MAQAO Performance Analysis and Optimization Tool
MAQAO website: www.maqao.org
• Documentation: www.maqao.org/documentation.html
  ▪ Tutorials for ONE View, LProf and CQA
  ▪ Lua API documentation
• Latest release: http://www.maqao.org/downloads.html
  ▪ Binary releases (2-3 per year)
  ▪ Core sources
• Publications around MAQAO:
  http://www.maqao.org/publications.html
MAQAO Team

- Prof. William Jalby
- Cédric Valensi, Ph D
- Emmanuel Oseret, Ph D
- Mathieu Tribalat
- Salah Ibn Amar
- Youenn Lebras
- Kévin Camus

Collaborators

- Prof. David J. Kuck
- David Wong, Ph D
- Othman Bouizi, Ph D
- Andrés S. Charif-Rubial, Ph D
- Eric Petit, Ph D
- Pablo de Oliveira, Ph D

Past Collaborators or Team members

- Prof. Denis Barthou
- Jean-Thomas Acquaviva, Ph D
- Stéphane Zuckerman, Ph D
- Julien Jaeger, Ph D
- Souad Koliaï, Ph D
- Zakaria Bendifallah, Ph D
- Tipp Moseley, Ph D
- Jean-Christophe Beyler, Ph D
- Hugo Bolloré
- Jean-Baptiste Le Reste
- Sylvain Henry, Ph D
- José Noudohouennou, Ph D
- Aleksandre Vardoshvili
- Romain Pillot
Thanks for your attention!

Questions?