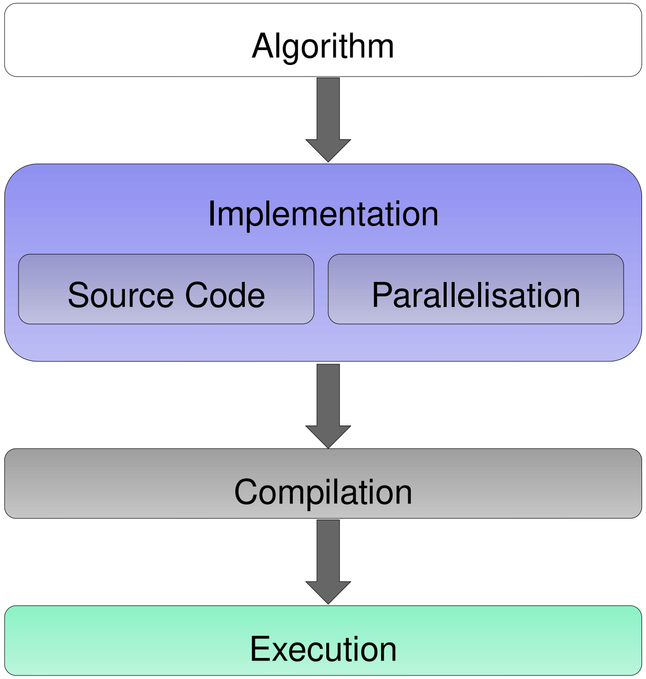
Guided Performance Analysis and Optimization using MAQAO

Performance Evaluation Team, University of Versailles

http://www.maqao.org

- **How much** of an application can be optimized?
- **Where** are the bottlenecks?
  - Data accesses, computations, I/O, ...
- **Why** is the application spending time there?
  - Algorithm, implementation or hardware?
- **How** can the situation be improved?
  - In which step(s) of the design process?
  - What additional information is needed?



Motivating Example

Code of a loop representing ~10% walltime

6) Variable number of iterations

2) Non-unit stride accesses

1) High number of statements

```
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))
   drtest2 = cnij/(rtest2 + 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*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
```

4) DIV/SQRT

3) Indirect accesses

5) Reductions

2) Non-unit stride accesses

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

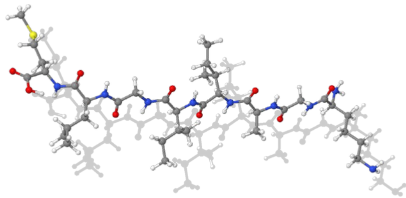
MAQAO: Modular Assembly Quality Analyzer and Optimizer

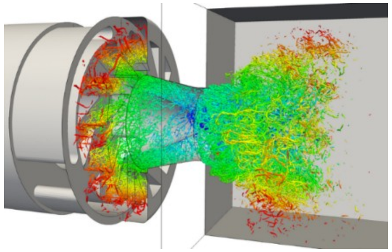
- Objectives:
  - Characterizing performance of HPC applications
  - Focusing on performance at the **core level**
  - **Guiding users** through optimization process
  - Estimating return of investment (**R.O.I.**)

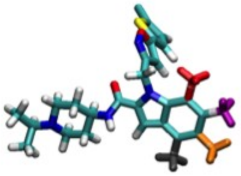
- Characteristics:
  - **Modular tool** offering complementary views
  - Support for **Intel x86-64** and **Xeon Phi**
  - LGPL3 Open Source software
  - Developed at UVSQ since 2004
  - Binary release available as **static executable**

- www.maqao.org

- QMC=CHEM (IRSAMC)
  - Quantum chemistry simulation
  - Speedup: > 3x
    - Moved invocation of function with identical parameters out of loop body
- Yales2 (CORIA)
  - Computational fluid dynamics
  - Speedup: up to 2,8x
    - IF removal for better vectorisation
    - 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





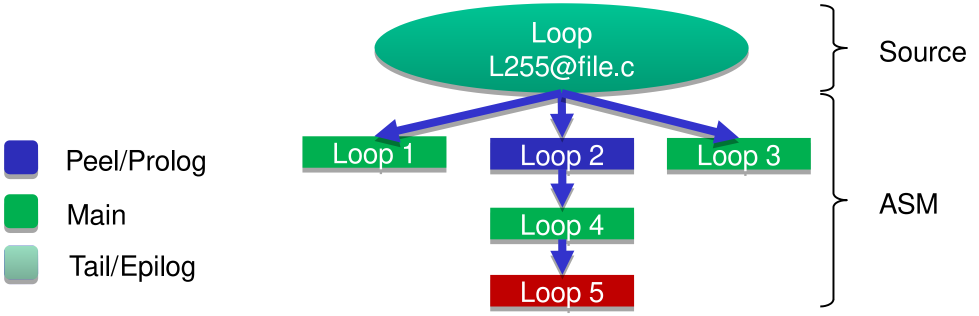


Partnerships

- MAQAO is funded by the UVSQ, Intel, and CEA (French department of energy) through Exascale Computing Research (ECR) and the French Ministry of Industry under 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

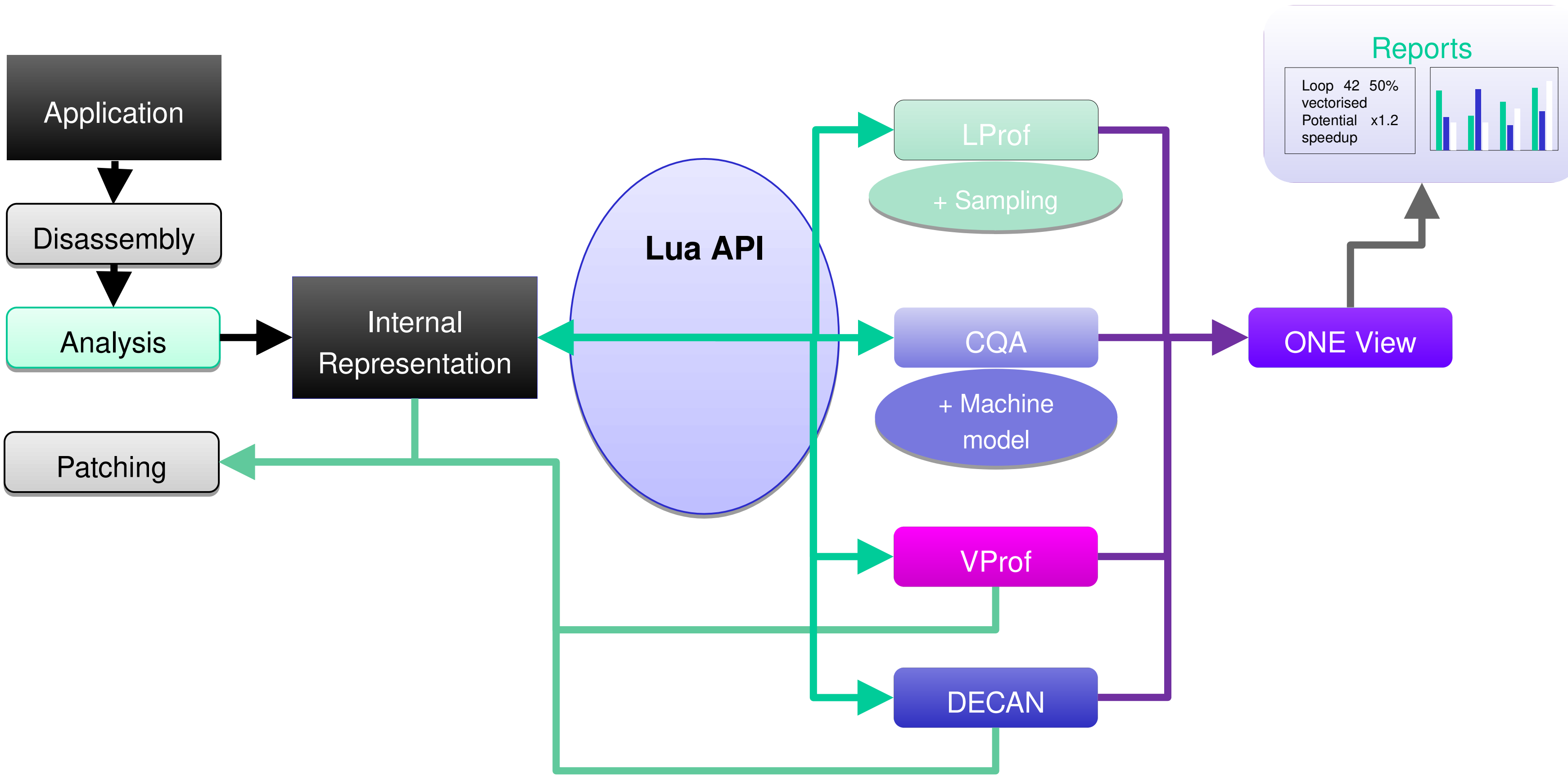
- **What You Analyze Is What You Run**
- Advantages of binary analysis:
  - Compiler optimizations increase the distance between the executed code and the source code
  - Source code instrumentation may prevent the compiler from applying some transformations
- Main steps:
  - Reconstruct the program structure
  - Relate the analyses to source code using debug symbols
    - A single source loop can be compiled as multiple assembly loops
    - Affecting unique identifiers to loops

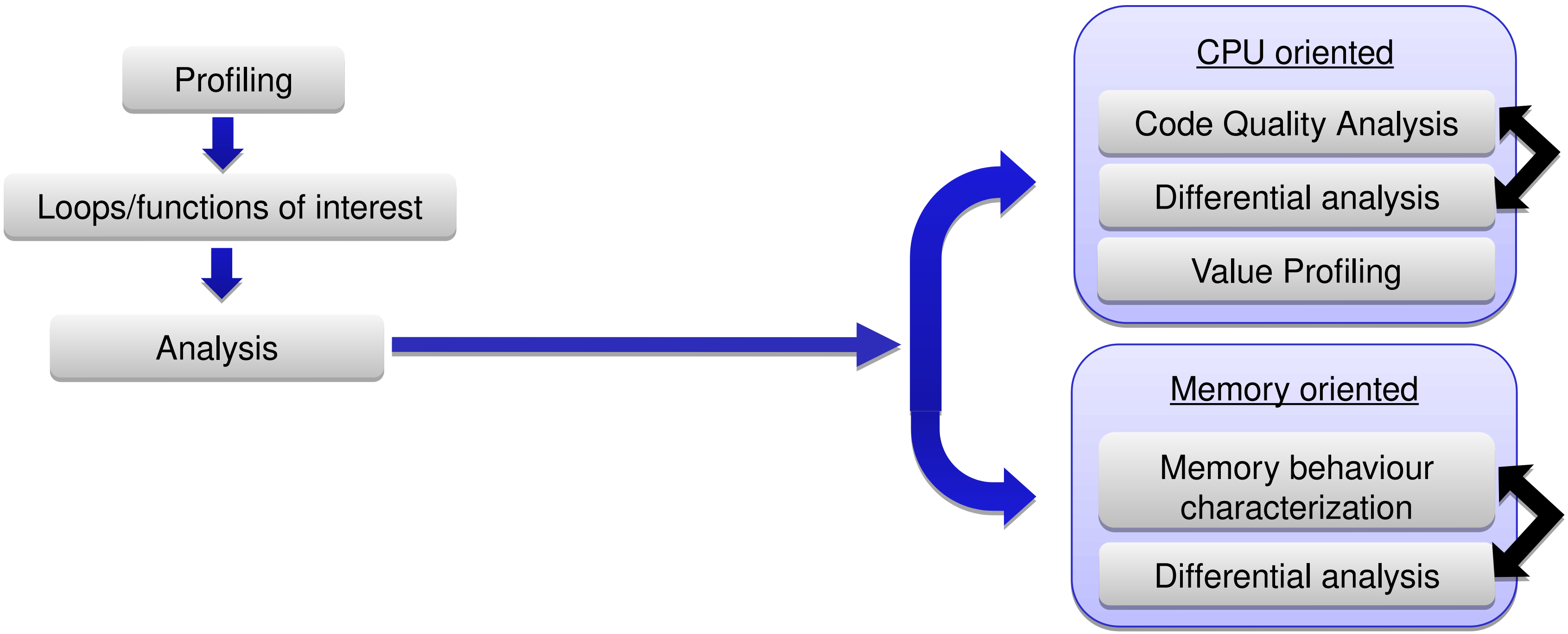


MAQAO Main Features/Modules

- Binary layer
  - Builds internal representation from a binary file
  - 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 assembly code and offers hints for improvements
  - UFS (Uops Flow Simulator): Cycle-accurate CPU 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




Profiling
Loops/functions of interest
Analysis
CPU oriented
Code Quality Analysis
Differential analysis
Value Profiling
Memory oriented
Memory behaviour characterization
Differential analysis

MAQAO LProf: Lightweight Profiler

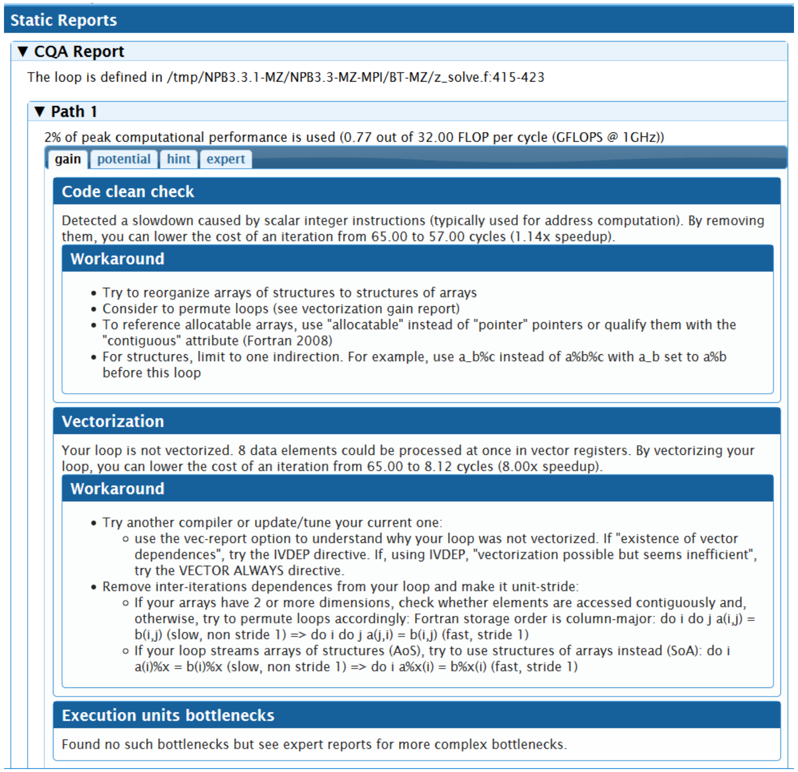
- **Goal**: Lightweight detection of application hotspots

- Features:
  - **Sampling** based
  - Access to hardware counters for additional information
  - Results at function and loop granularity

- Strengths:
  - **Non intrusive**: No recompilation necessary
  - **Low overhead**
  - Agnostic with regard to parallel runtime

MAQAO CQA: Code Quality Analyzer

- Goal: **Assist developers** in improving code performance
- Features:
  - Evaluates the **quality** of the compiler generated assembly 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**



Static Reports

▼ CQA Report

The loop is defined in /tmp/NPB3.3.1-MZ/NPB3.3-MZ-MPI/BT-MZ/z_solve.f:415-423

▼ Path 1

2% of peak computational performance is used (0.77 out of 32.00 FLOP per cycle (GFLOPS @ 1GHz))

gain | potential | hint | expert

Code clean check

Detected a slowdown caused by scalar integer instructions (typically used for address computation). By removing them, you can lower the cost of an iteration from 65.00 to 57.00 cycles (1.14x speedup).

Workaround

- Try to reorganize arrays of structures to structures of arrays
- Consider to permute loops (see vectorization gain report)
- To reference allocatable arrays, use "allocatable" instead of "pointer" pointers or qualify them with the "contiguous" attribute (Fortran 2008)
- For structures, limit to one indirection. For example, use a_b%c instead of a%b%c with a_b set to a%b before this loop

Vectorization

Your loop is not vectorized. 8 data elements could be processed at once in vector registers. By vectorizing your loop, you can lower the cost of an iteration from 65.00 to 8.12 cycles (8.00x speedup).

Workaround

- Try another compiler or update/tune your current one:
  - use the vec-report option to understand why your loop was not vectorized. If "existence of vector dependences", try the IVDEP directive. If, using IVDEP, "vectorization possible but seems inefficient", try the VECTOR ALWAYS directive.
- Remove inter-iterations dependences from your loop and make it unit-stride:
  - If your arrays have 2 or more dimensions, check whether elements are accessed contiguously and, otherwise, try to permute loops accordingly: Fortran storage order is column-major: do i do j a(i,j) = b(i,j) (slow, non stride 1) => do i do j a(j,i) = b(i,j) (fast, stride 1)
  - If your loop streams arrays of structures (AoS), try to use structures of arrays instead (SoA): do i a(i)%x = b(i)%x (slow, non stride 1) => do i a%x(i) = b%x(i) (fast, stride 1)

Execution units bottlenecks

Found no such bottlenecks but see expert reports for more complex bottlenecks.

MAQAO CQA Main Concepts

- Most of the time applications only exploit at best 5% to 10% of the peak performance

- Concepts:
  - Peak performance
  - Execution pipeline
  - Resources/Functional units

- Key performance levers for core level efficiency:
  - Vectorization
  - Avoid high latency instructions if possible
  - Guide the compiler to generate an efficient code
  - Reorganize memory layout

MAQAO CQA Application to Motivating Example

Issues identified by CQA

6) Variable number of iterations

2) Non-unit stride accesses

```
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))
   drtest2 = cnij/(rtest2 + 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*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
```

4) DIV/SQRT

3) Indirect accesses

5) Reductions

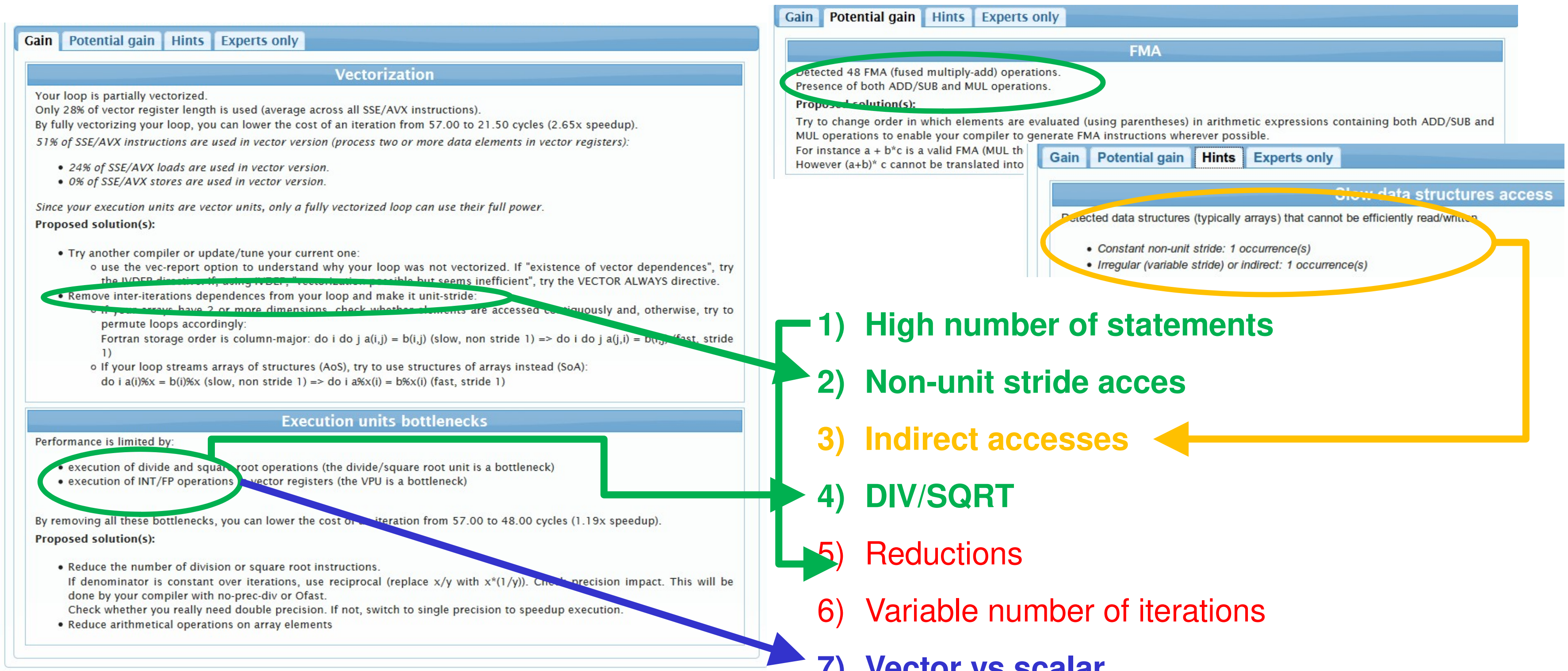
2) Non-unit stride accesses

1) High number of statements

7) Vector vs scalar

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**



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

MAQAO DECAN: Decremental Analysis

- Goal: modify the application to
  - Identify the cause of bottlenecks
  - Estimate associated ROI

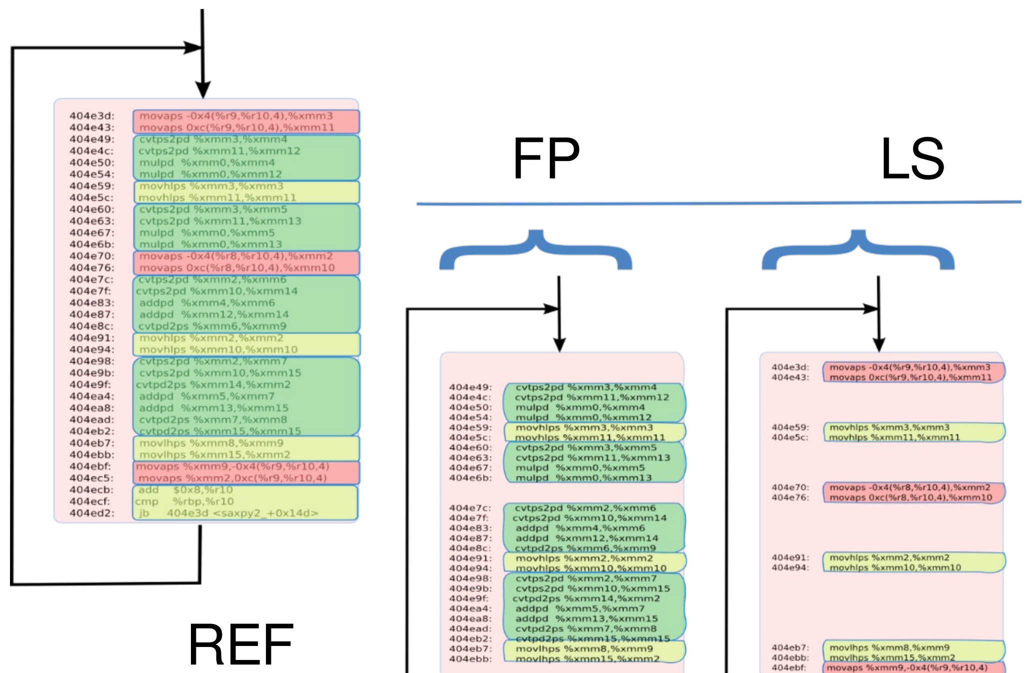
- Differential analysis:
  - Targets innermost loops
  - Transforms loops
  - Compare performance of original and transformed variant

- Transformations
  - Remove or modify groups of instructions
  - Targets memory accesses or computation

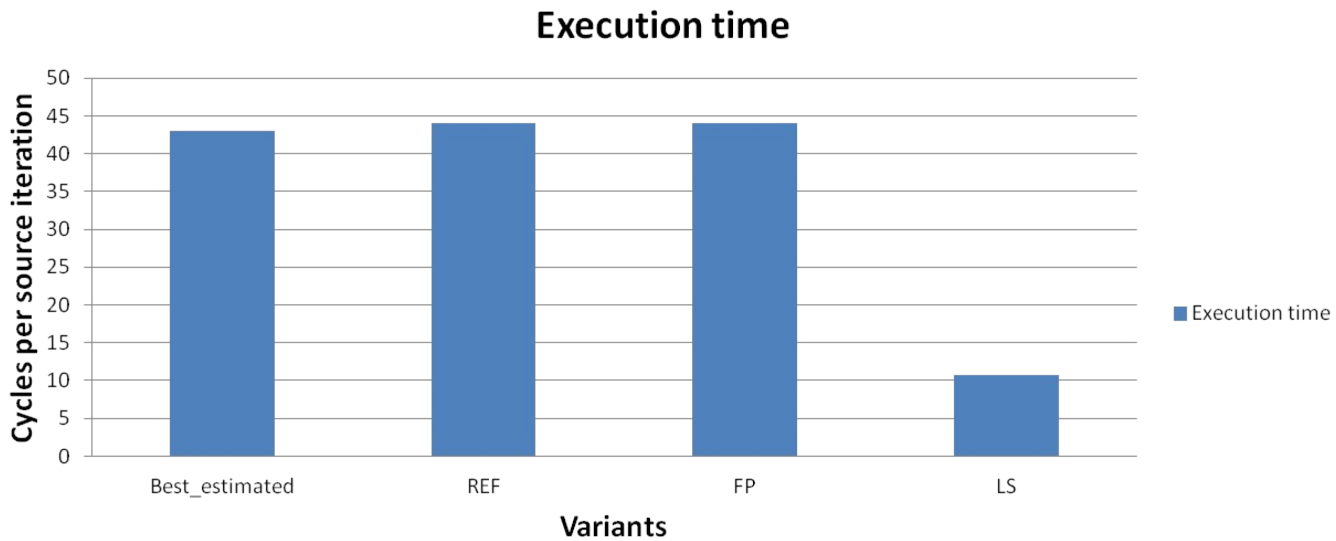
MAQAO DECAN Transformations

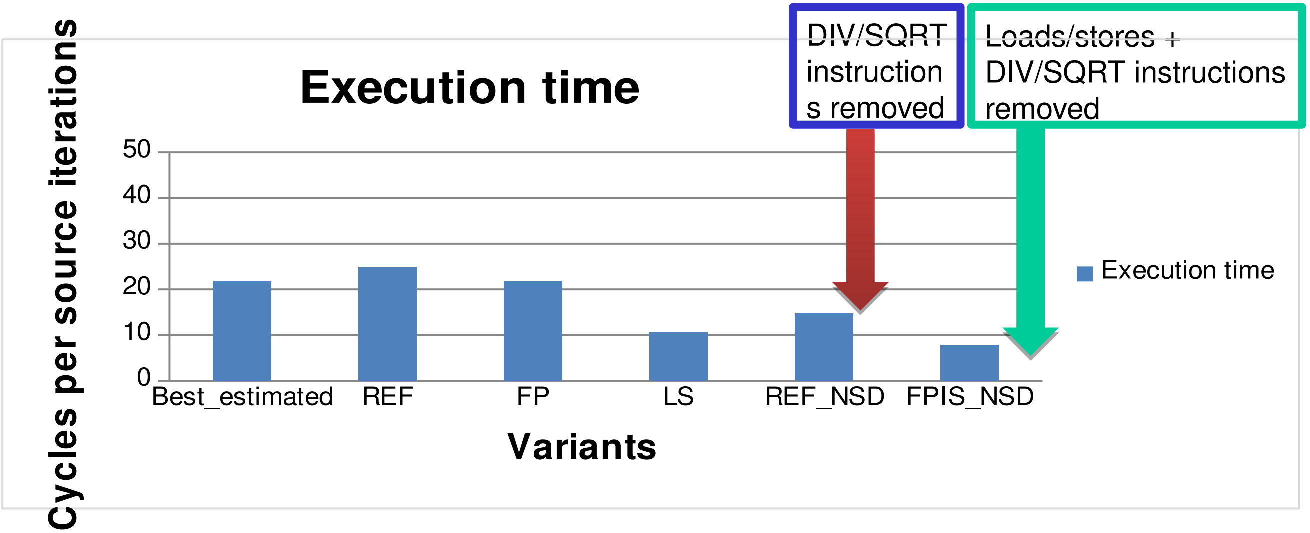
- Typical transformations:
  - FP: only FP arithmetic instructions are preserved
    - => loads and stores are removed
  - LS: only loads and stores are preserved
    - => compute instructions are removed
  - DL1: memory references replaced with global variables ones
    - => data now accessed from L1

MAQAO DECAN Example



- ROI = FP / LS = 4,1
- Imbalance between the two streams
  => Try to consume more elements inside one iteration.





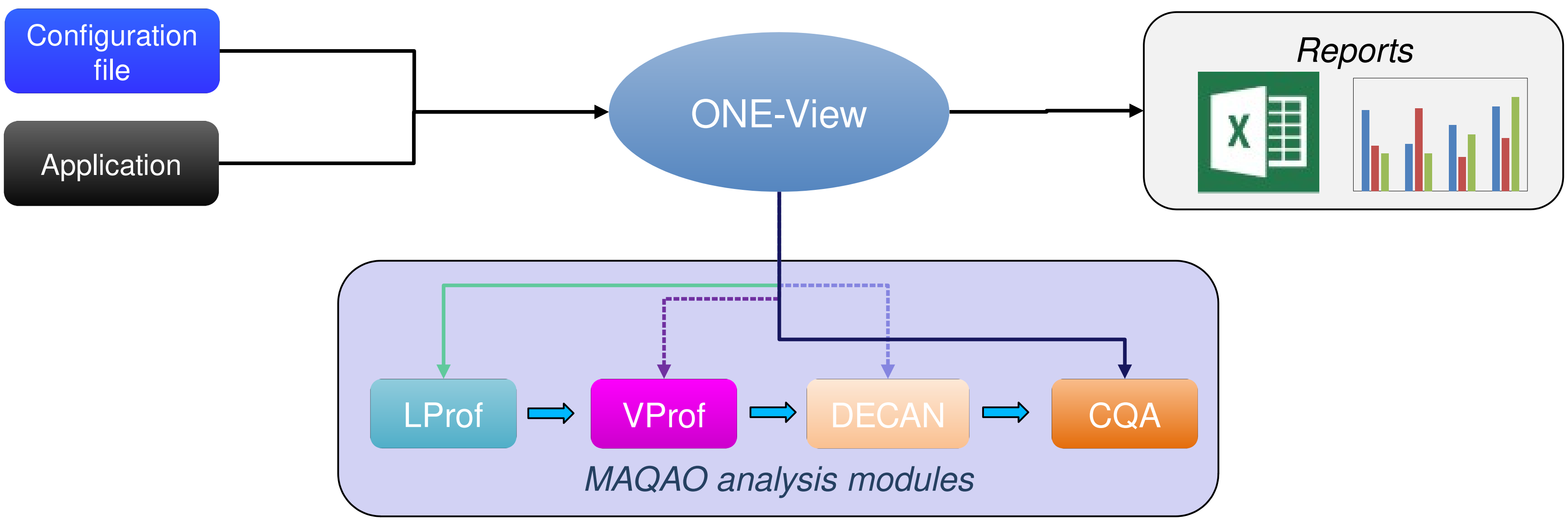
REF_NSD : removing DIV/SQRT instructions provides a 1.5 x speedup
=> the bottleneck is the presence of these DIV/SQRT instructions

FPLS_NSD : removing loads/stores after DIV/SQRT provides a small additional speedup

Conclusion: No room left for improvement here (algorithm bound)

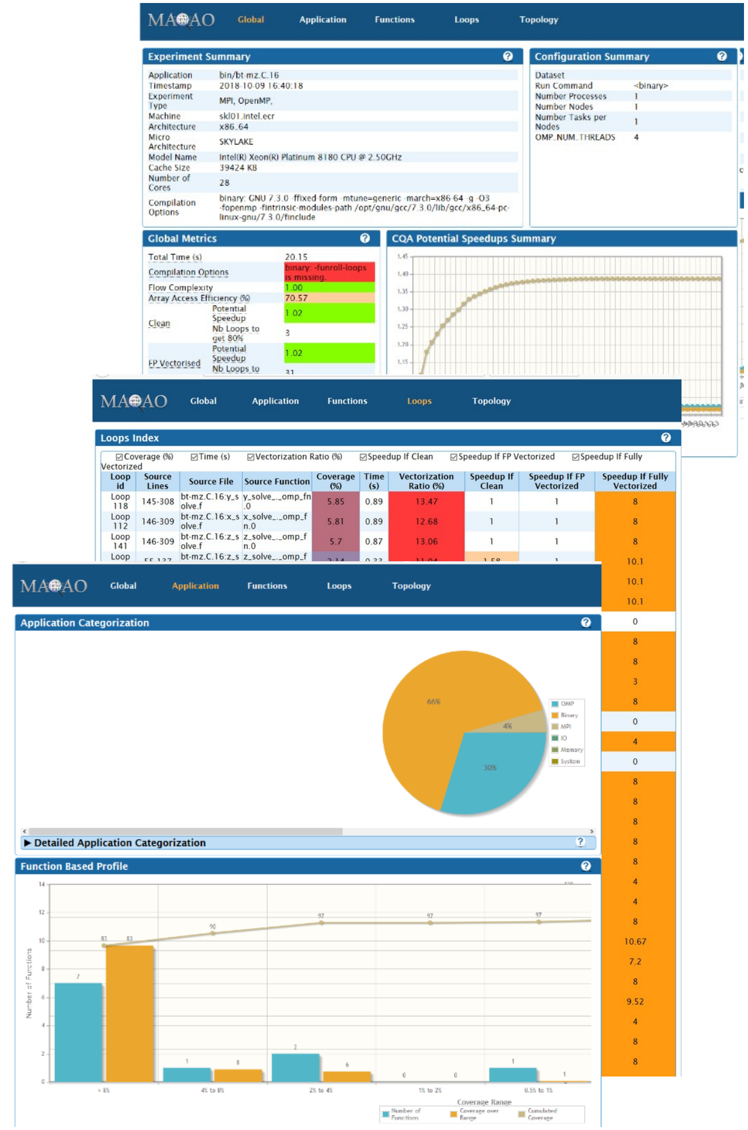
- Value profiling
  - Targets loops or functions
  - Instrumentation based
  - Iteration count, loop paths, function parameters, …
- Metrics
  - Detection of stable values
  - Loop characterisation through the number of iterations
- Provides insights and leads for code specialization

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



MAQAO ONE View: Performance View Aggregator

- 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



ONE View Reports Levels

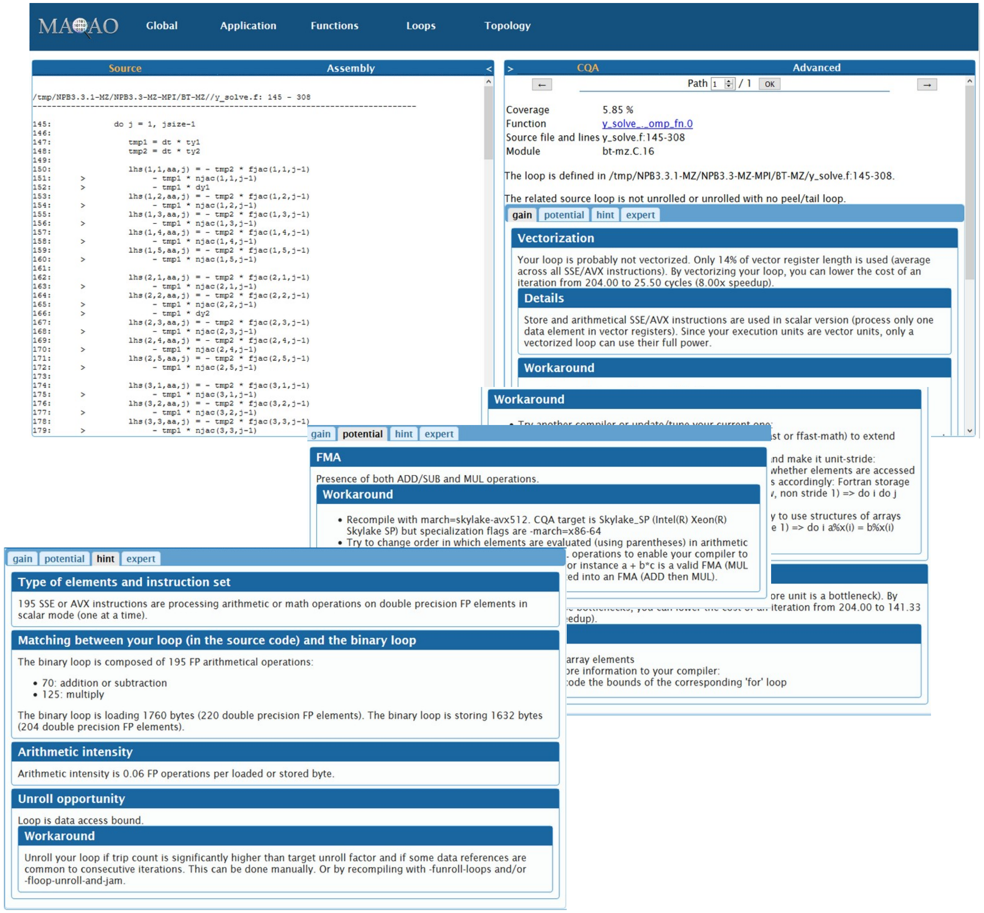
- ONE VIEW **ONE**
  - Requires a single run of the application
  - Profiling of the application using **LProf**
  - Static analysis using **CQA**
- ONE VIEW **TWO** (includes analyses from report **ONE**)
  - Requires **3 or 4 runs** on average
  - Value profiling using **VProf** to identify loop iteration 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**
  - Requires as many additional runs as parallel configurations
  - Can be executed in addition to another report
  - Profilings using **LProf** on different parallel configurations

ONE View Global Metrics

- 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
- 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

High level reports

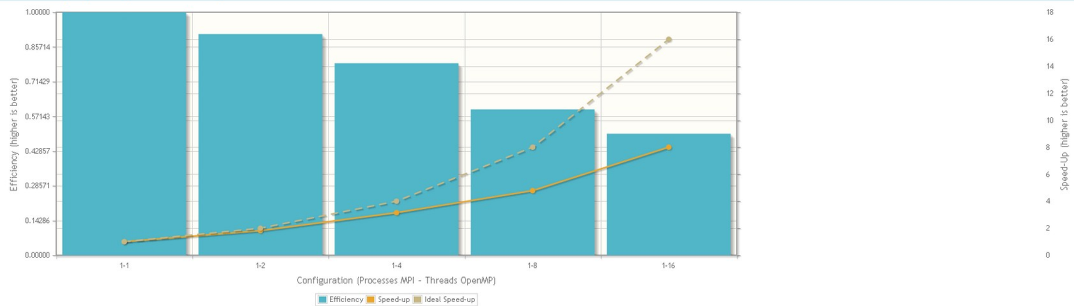
- Reference to the source code
- Bottleneck description
- Hints for improving performance
- Reports categorized by probability that applying hints will yield predicted gain
  - Gain: Good probability
  - Potential gain: Average probability
  - Hints: Lower probability



MAQAO ONE View Scalability Reports

- Goal: Provide a view of the application scalability
  - Profiles with different numbers of threads/processes
  - Displays efficiency metrics for application

Scalability - Speed-Up and Efficiency

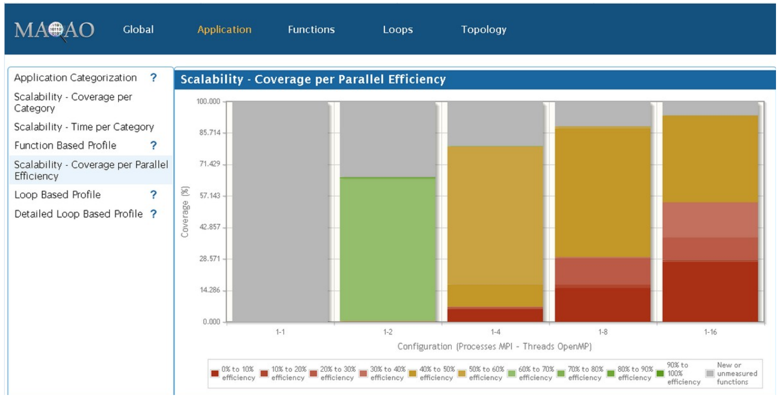


▼ Detailed Speed-Up and Efficiency

Number of threads	Configuration (Processes MPI - Threads OpenMP)	Efficiency (ideal is 1)	Speed-Up	Ideal Speed-Up	Time (s)
2	1-1	1	1	1	546.3
3	1-2	0.91	1.82	2	300.66
5	1-4	0.79	3.15	4	173.42
9	1-8	0.6	4.78	8	114.18
17	1-16	0.5	8	16	68.26

- Coverage per category
  - Comparison of categories for each run

- Coverage per parallel efficiency
    - Distinguishing functions only represented in parallel or sequential
  - Displays efficiency by coverage



More on MAQAO

- 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 and Collaborators

MAQAO Team

- Prof. William Jalby
- Cédric Valensi, Ph D
- Emmanuel Oseret, Ph D
- Mathieu Tribalat
- Salah J. Ibnamar
- Kévin Camus

Collaborators

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

Past Collaborators/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
- Youenn Lebras

Thanks for your attention!