MAQAO
Performance Analysis and Optimization Tool

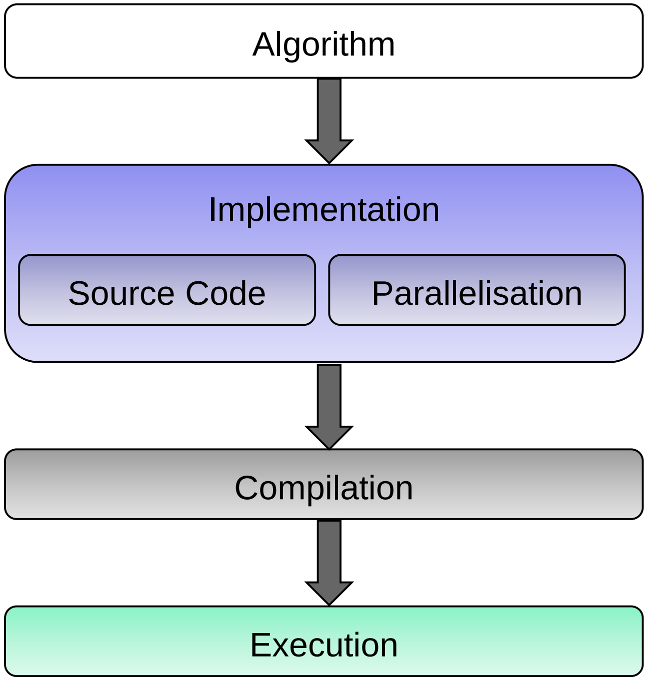
Cédric Valensi, Emmanuel Oseret
cedric.valensi@uvsq.fr, emmanuel.oseret@uvsq.fr
Performance Evaluation Team, University of Versailles
http://www.maqao.org

- **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?

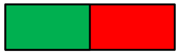


A Multifaceted Problem

- **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

Motivating Example

Code of a loop representing ~10% walltime

```
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
```

Annotations on the code:

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

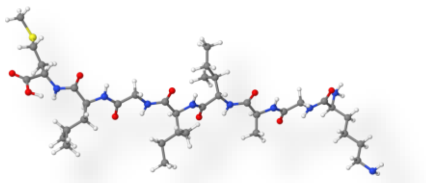
Source code and associated issues:

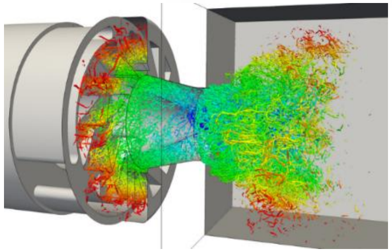
1) High number of statements
2) Non-unit stride accesses
3) Indirect accesses
4) DIV/SQRT
5) Reductions
6) Variable number of iterations

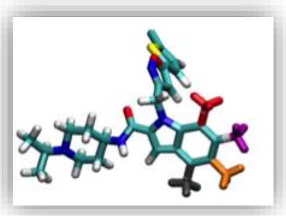
- Objectives:
  - Characterizing performance of HPC applications
  - Focusing on performance at the **core level**
  - **Guiding users** through optimization process
  - Estimating return of investment (**R.O.I.**)

- 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

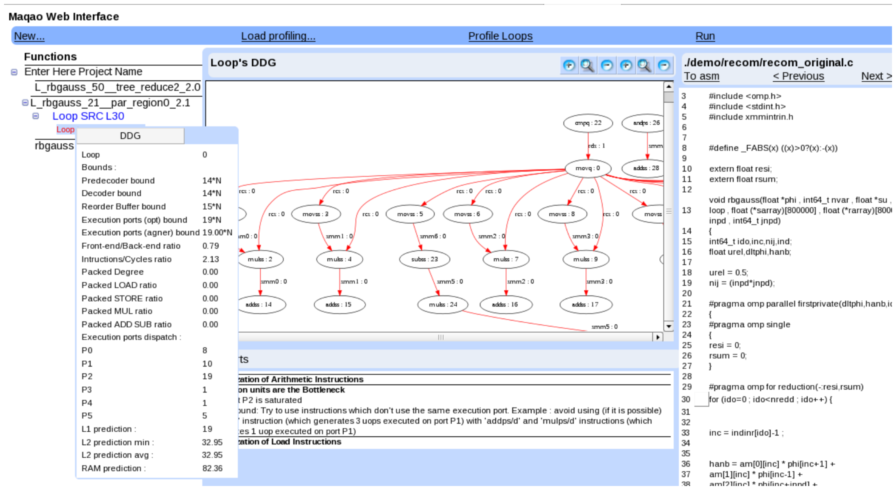






MAQAO History

- 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



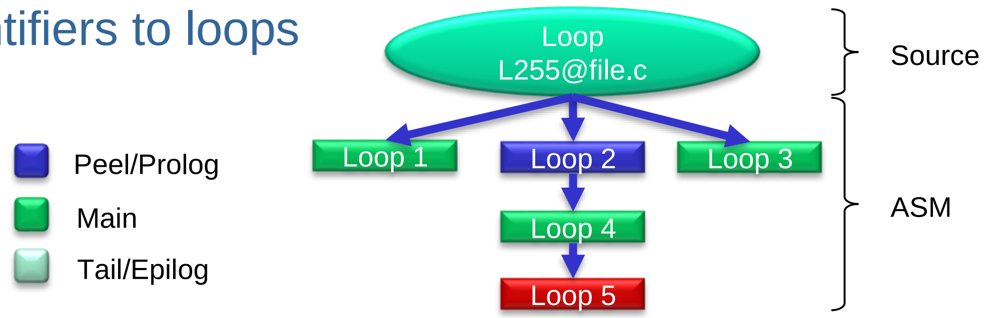
Partnerships

- 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

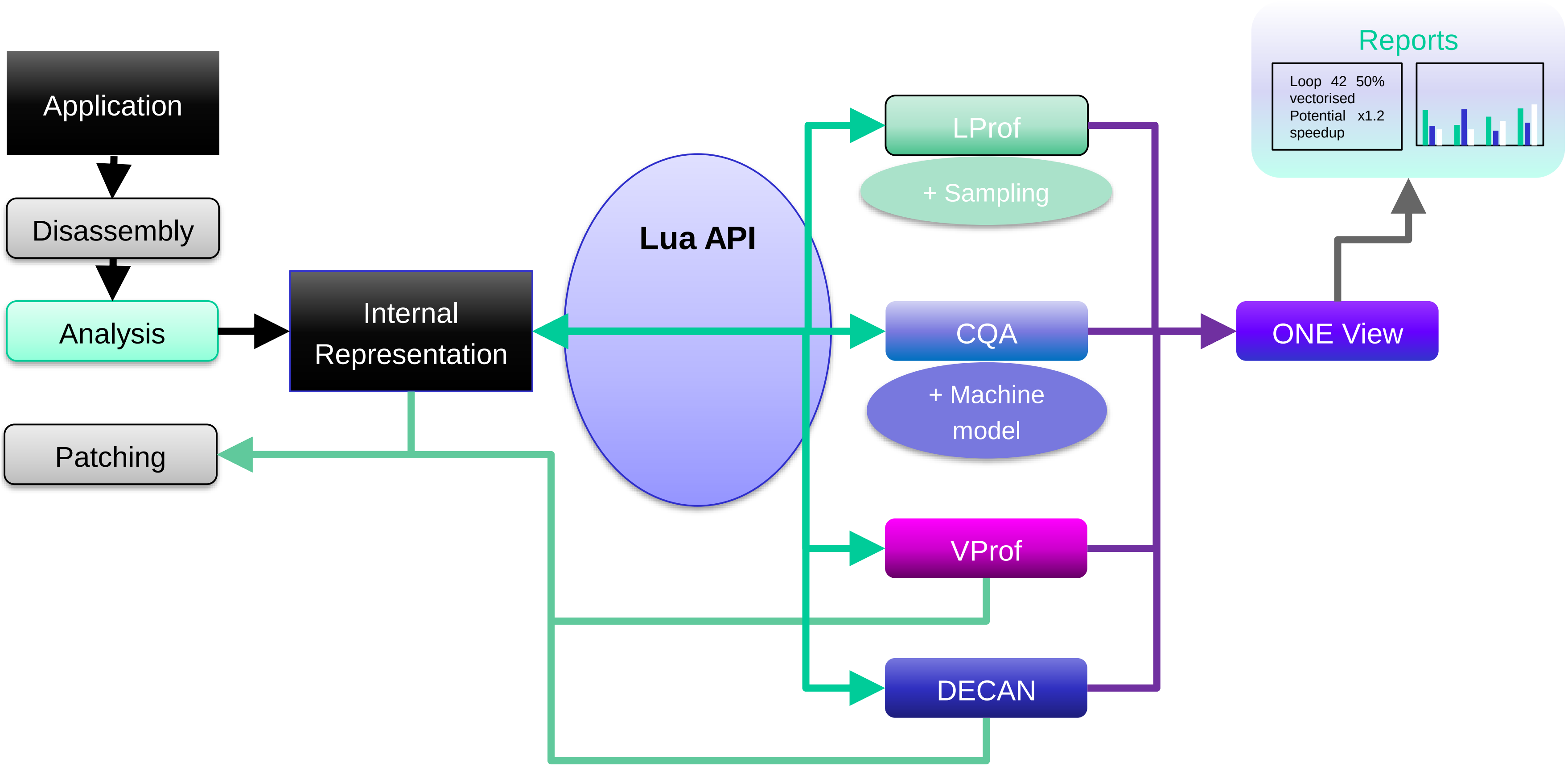
- Advantages of binary analysis:
  - Compiler optimizations increase the distance between the executed code and the source
  - Source code instrumentation may prevent the compiler from applying some transformations
- We want to evaluate the “real” executed code: **What You Analyse Is What You Run**
- 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



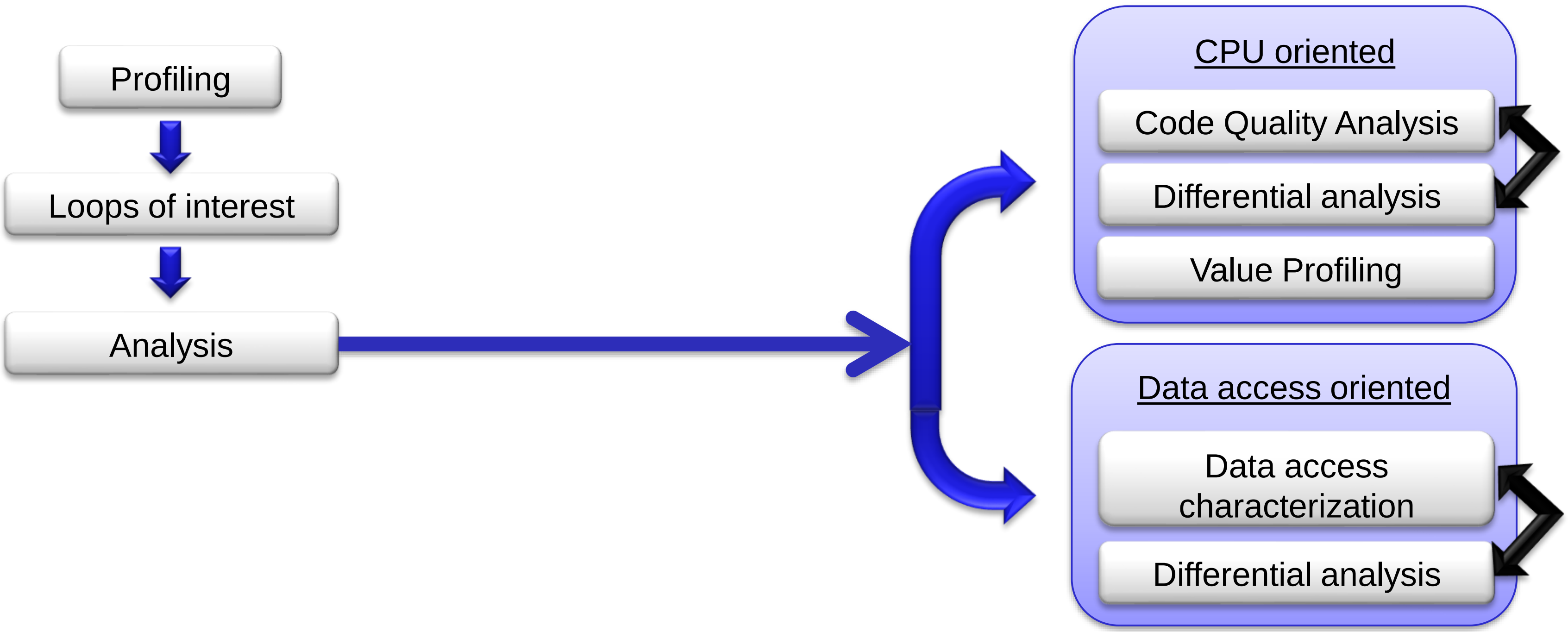
MAQAO Main Features

- 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



- Decision tree



MAQAO LProf: Lightweight Profiler

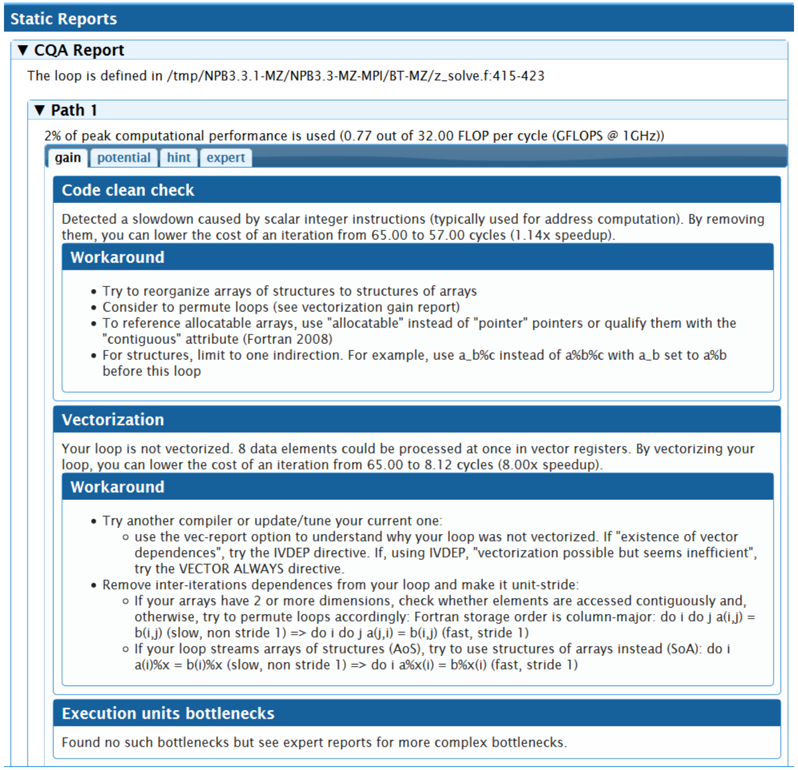
- **Goal**: Lightweight localization of application hotspots

- 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

MAQAO CQA: Code Quality Analyzer

- 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**


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

- 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

Same instruction – Same cost

Process up to 8X (SP) data

- 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

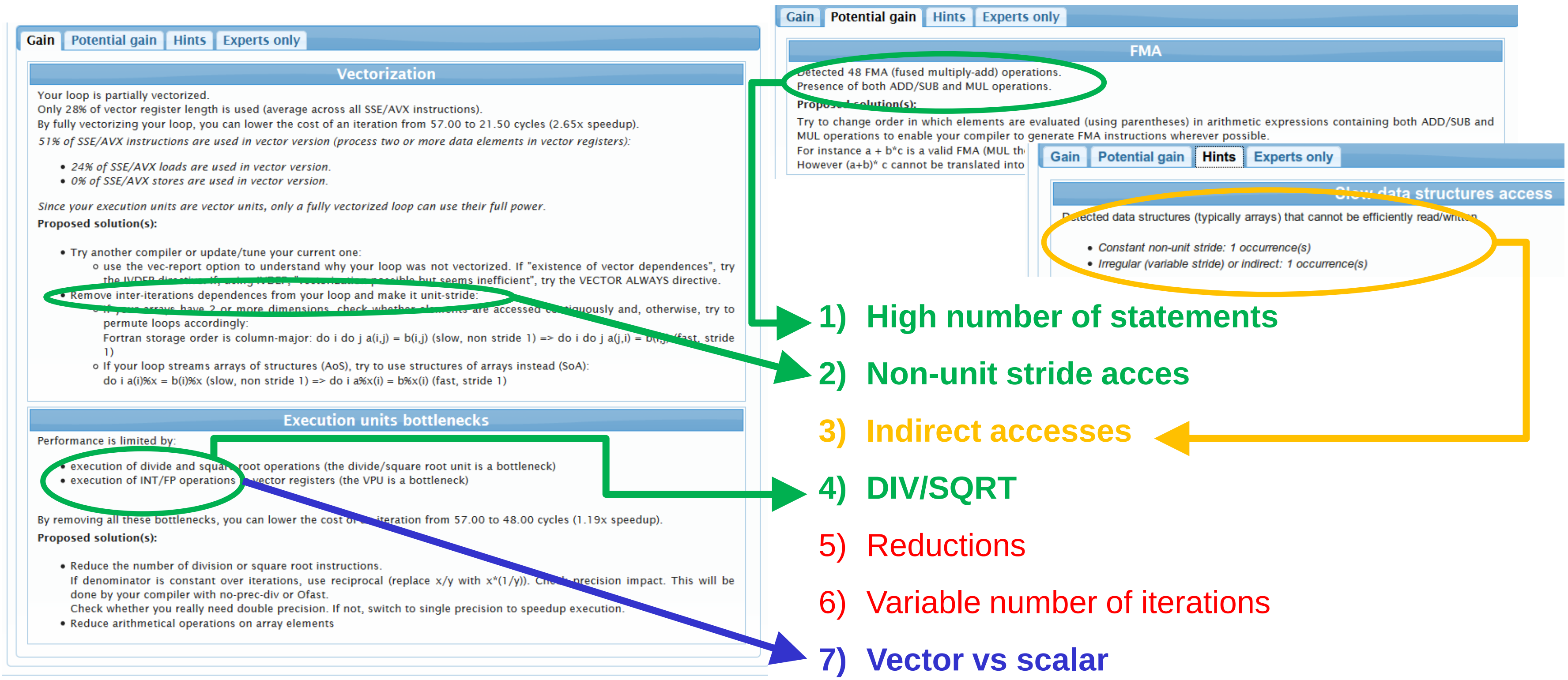
Issues identified by CQA

```
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
```


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

MAQAO CQA Application to motivating example



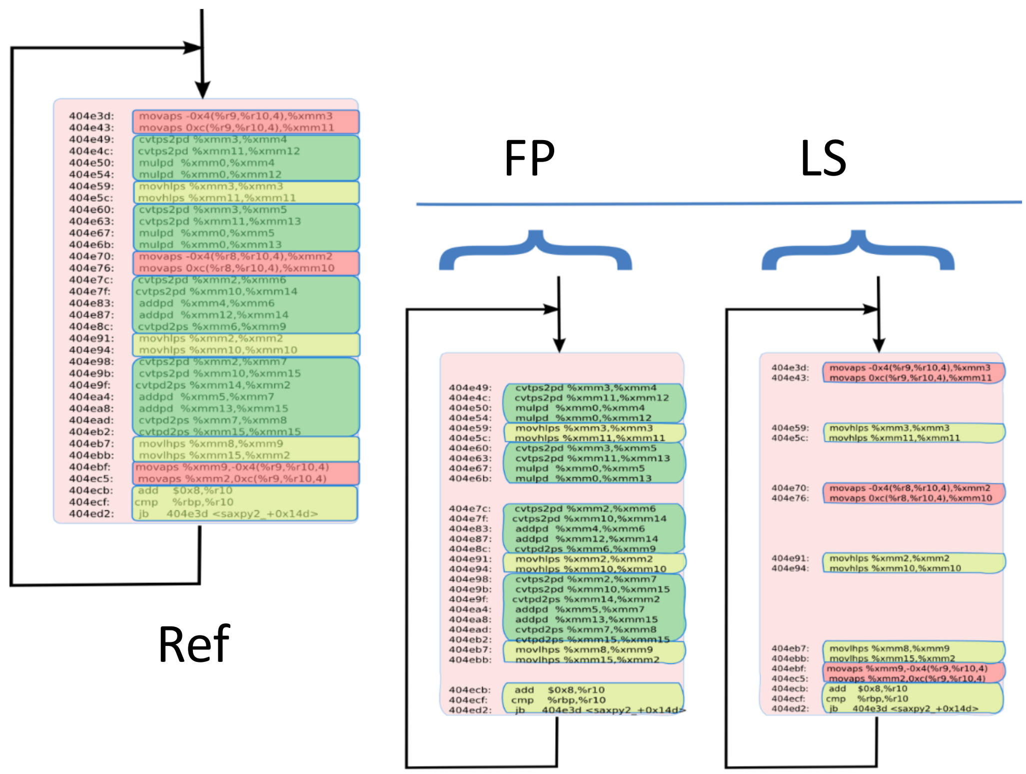
- Goal: modify the application to
  - Identify cause of bottlenecks
  - Estimate associated ROI

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

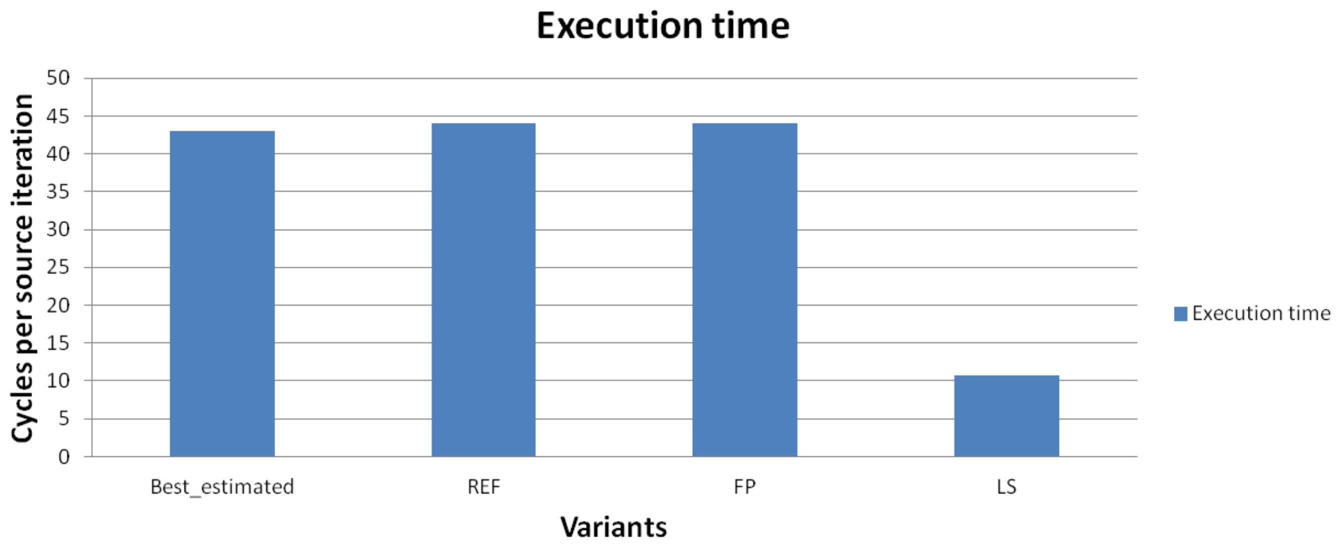
- Transformations
  - Remove or modify groups of instructions
  - Targets memory accesses or computation

- 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

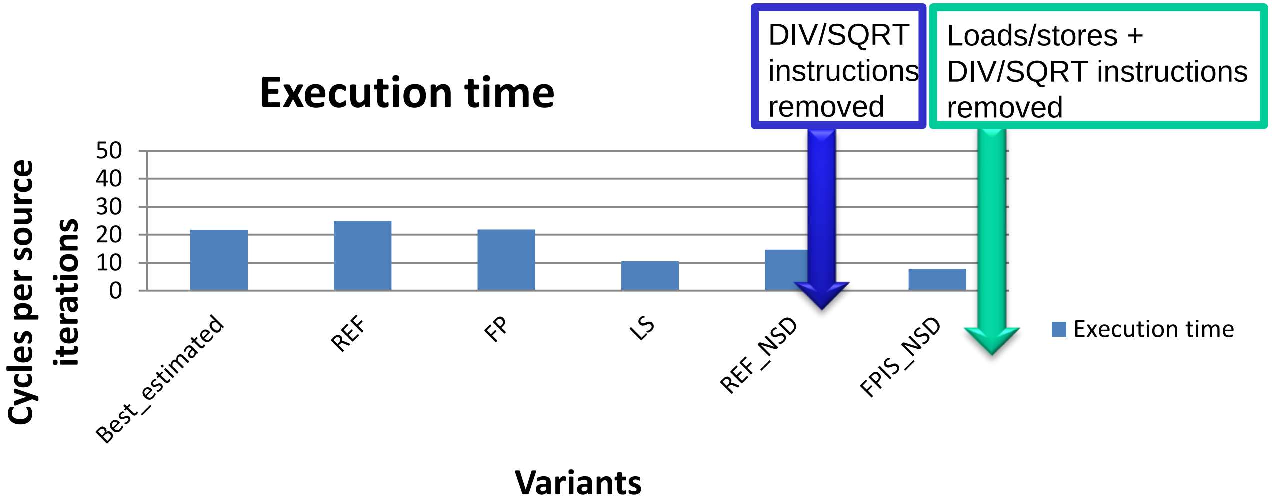
MAQAO DECAN Example



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



MAQAO DECAN
Application to Motivating Example

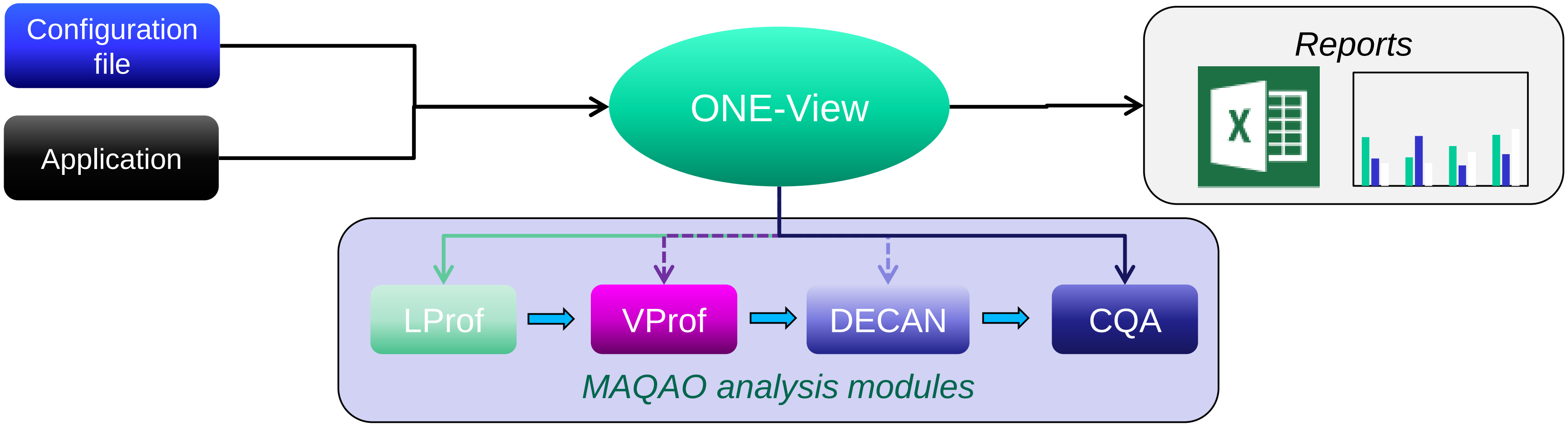


REF_NSD : removing DIV/SQRT instructions provides a 1.5 x speedup
=> the bottleneck is the presence of these DIV/SQRT instructions
FPLS_NSD : removing loads/stores after DIV/SQRT provides a small additional speedup
Conclusion: No room left for improvement here (algorithm bound)

MAQAO VProf: Value Profiling

- 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



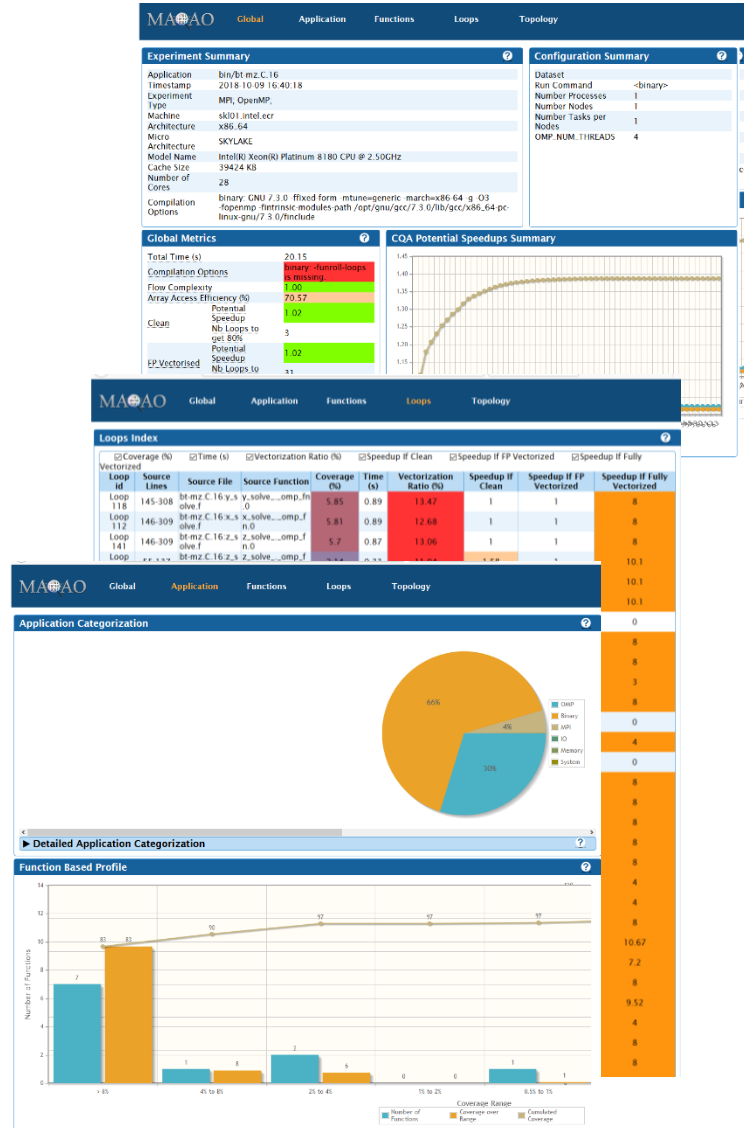
ONE View Reports Levels

- 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

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



- 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

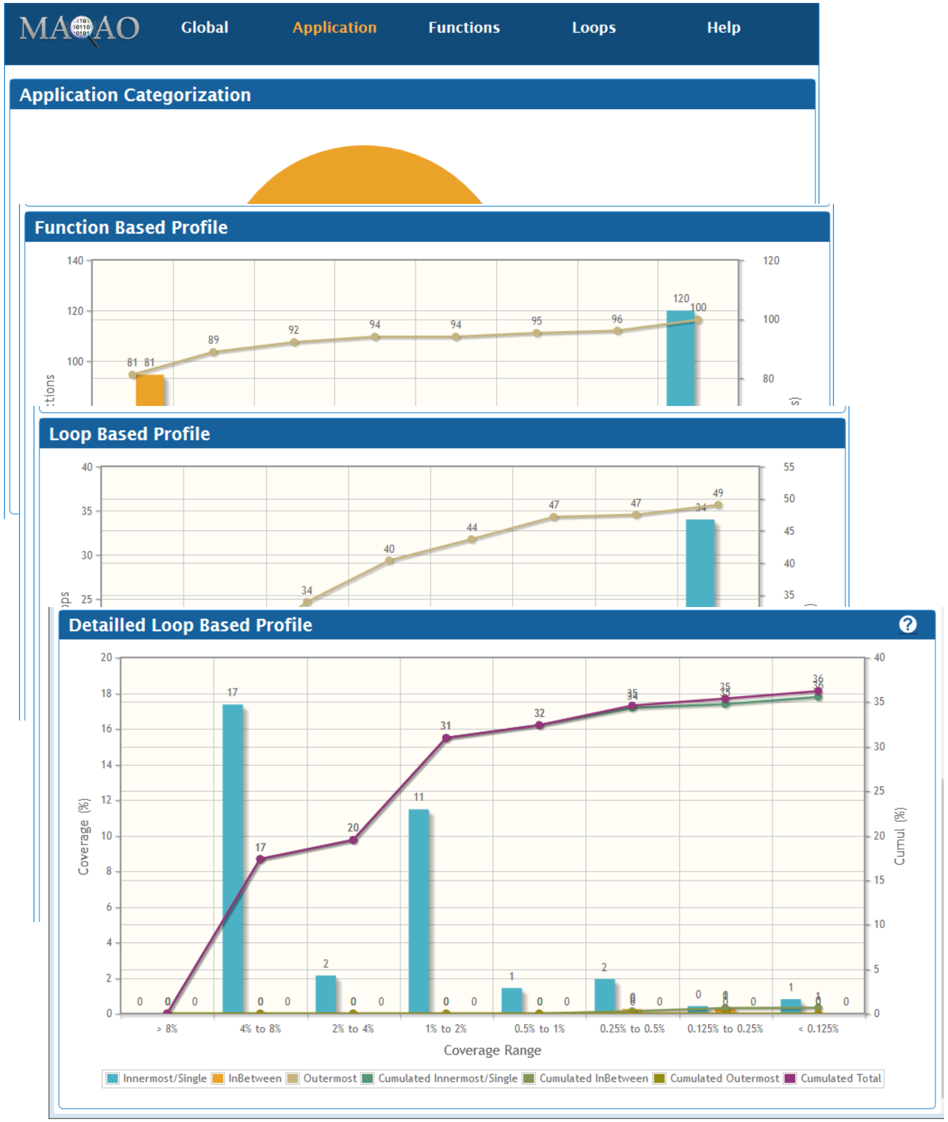
- Global metrics
  - General quality metrics derived from MAQAO analyses
  - Global speedup predictions
- Potential speedups
  - Speedup prediction depending on the number of optimised loops
  - Ordered speedups to identify the loops to optimise in priority

$$Global\ Speedup = \sum_{loops} coverage \ * potential\ speedup$$

- 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

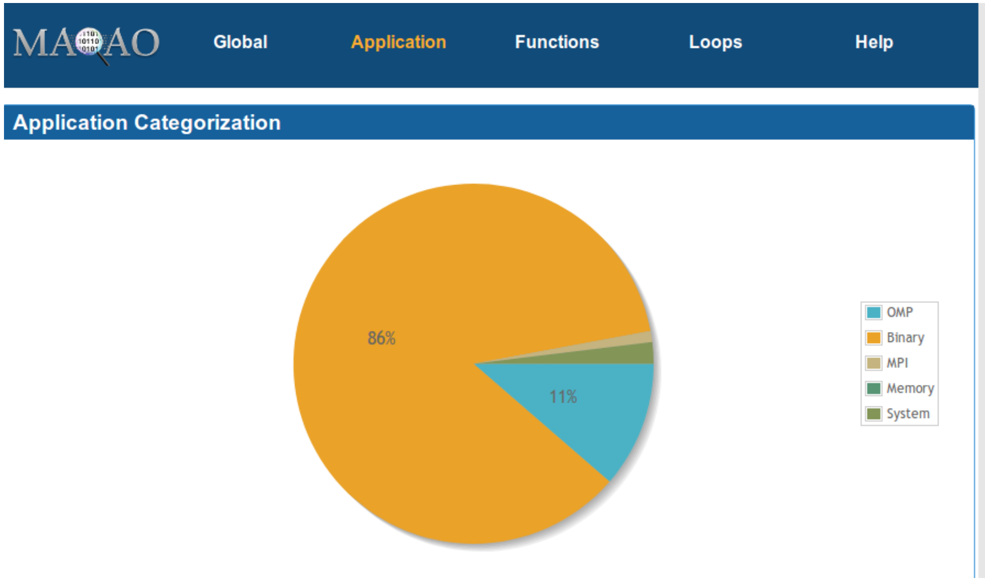
MAQAO ONE View Application Characteristics

- 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



MAQAO ONE View Application Characteristics Time Categorisation

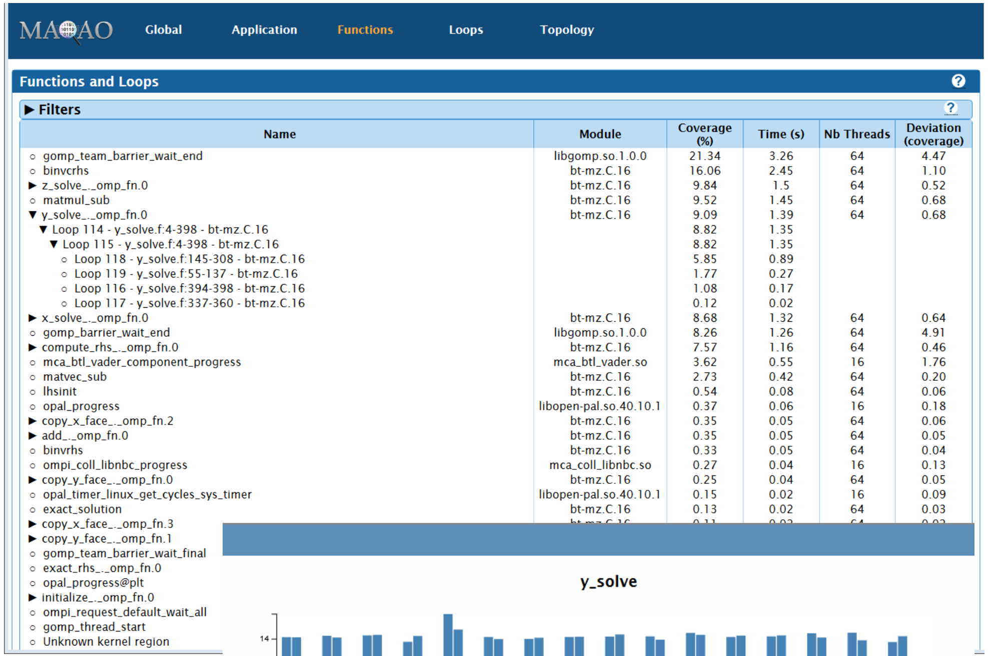
- Goal: allowing to identify at a glance where time is spent
  - Categories based on functions or libraries names
- Application
  - Main executable
- Parallelization
  - Threads
  - OpenMP
  - MPI
- System libraries
  - I/O operations
  - String operations
  - Memory management functions (allocation, free)
- External libraries
  - Specialised libraries such as libm / libmkl
  - Application code in external libraries



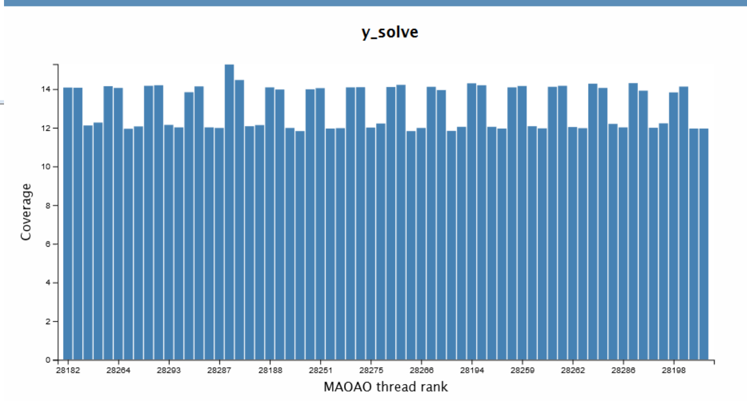
MAQAO ONE View: Functions Profiling

Identifying hotspots

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



Name	Module	Coverage (%)	Time (s)	Nb Threads	Deviation (coverage)
gomp_team_barrier_wait_end	libgomp.so.1.0.0	21.34	3.26	64	4.47
binvcrhs	bt-mz.C.16	16.06	2.45	64	1.10
▶ z_solve_._omp_fn.0	bt-mz.C.16	9.84	1.5	64	0.52
matmul_sub	bt-mz.C.16	9.52	1.45	64	0.68
▼ y_solve_._omp_fn.0	bt-mz.C.16	9.09	1.39	64	0.68
▼ Loop 114 - y_solve.f:4-398 - bt-mz.C.16		8.82	1.35		
▼ Loop 115 - y_solve.f:4-398 - bt-mz.C.16		8.82	1.35		
Loop 118 - y_solve.f:145-308 - bt-mz.C.16		5.85	0.89		
Loop 119 - y_solve.f:55-137 - bt-mz.C.16		1.77	0.27		
Loop 116 - y_solve.f:394-398 - bt-mz.C.16		1.08	0.17		
Loop 117 - y_solve.f:337-360 - bt-mz.C.16		0.12	0.02		
▶ x_solve_._omp_fn.0	bt-mz.C.16	8.68	1.32	64	0.64
gomp_barrier_wait_end	libgomp.so.1.0.0	8.26	1.26	64	4.91
▶ compute_rhs_._omp_fn.0	bt-mz.C.16	7.57	1.16	64	0.46
mca_btl_vader_component_progress	mca_btl_vader.so	3.62	0.55	16	1.76
matvec_sub	bt-mz.C.16	2.73	0.42	64	0.20
lhsinit	bt-mz.C.16	0.54	0.08	64	0.06
opal_progress	libopen-pal.so.40.10.1	0.37	0.06	16	0.18
▶ copy_x_face_._omp_fn.2	bt-mz.C.16	0.35	0.05	64	0.06
▶ add_._omp_fn.0	bt-mz.C.16	0.35	0.05	64	0.05
binvrhs	bt-mz.C.16	0.33	0.05	64	0.04
ompi_coll_libnbc_progress	mca_coll_libnbc.so	0.27	0.04	16	0.13
▶ copy_y_face_._omp_fn.0	bt-mz.C.16	0.25	0.04	64	0.05
opal_timer_linux_get_cycles_sys_timer	libopen-pal.so.40.10.1	0.15	0.02	16	0.09
exact_solution	bt-mz.C.16	0.13	0.02	64	0.03
▶ copy_x_face_._omp_fn.3					
▶ copy_y_face_._omp_fn.1					
gomp_team_barrier_wait_final					
exact_rhs_._omp_fn.0					
opal_progress@plt					
▶ initialize_._omp_fn.0					
ompi_request_default_wait_all					
gomp_thread_start					
Unknown kernel region					



▼ matmul_sub
- Loop 230 - solve_subs.f:71-175 - bt-mz.C.16
- Loop 231 - solve_subs.f:71-175 - bt-mz.C.16 — Single

▼ z_solve
- ▼ Loop 232 - z_solve.f:53-423 - bt-mz.C.16 — Outermost
  - ▼ Loop 233 - z_solve.f:54-423 - bt-mz.C.16 — Inbetween
    - ▼ Loop 236 - z_solve.f:54-423 - bt-mz.C.16 — Inbetween
      - Loop 239 - z_solve.f:146-308 - bt-mz.C.16 — Innermost
      - Loop 235 - z_solve.f:55-137 - bt-mz.C.16
      - Loop 234 - z_solve.f:415-423 - bt-mz.C.16

MAQAO ONE View Loop Profiling Summary

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

MAQAO | Global | Application | Functions | Loops | Topology

Show Full Profile | Open Expert Summary

Loops Index

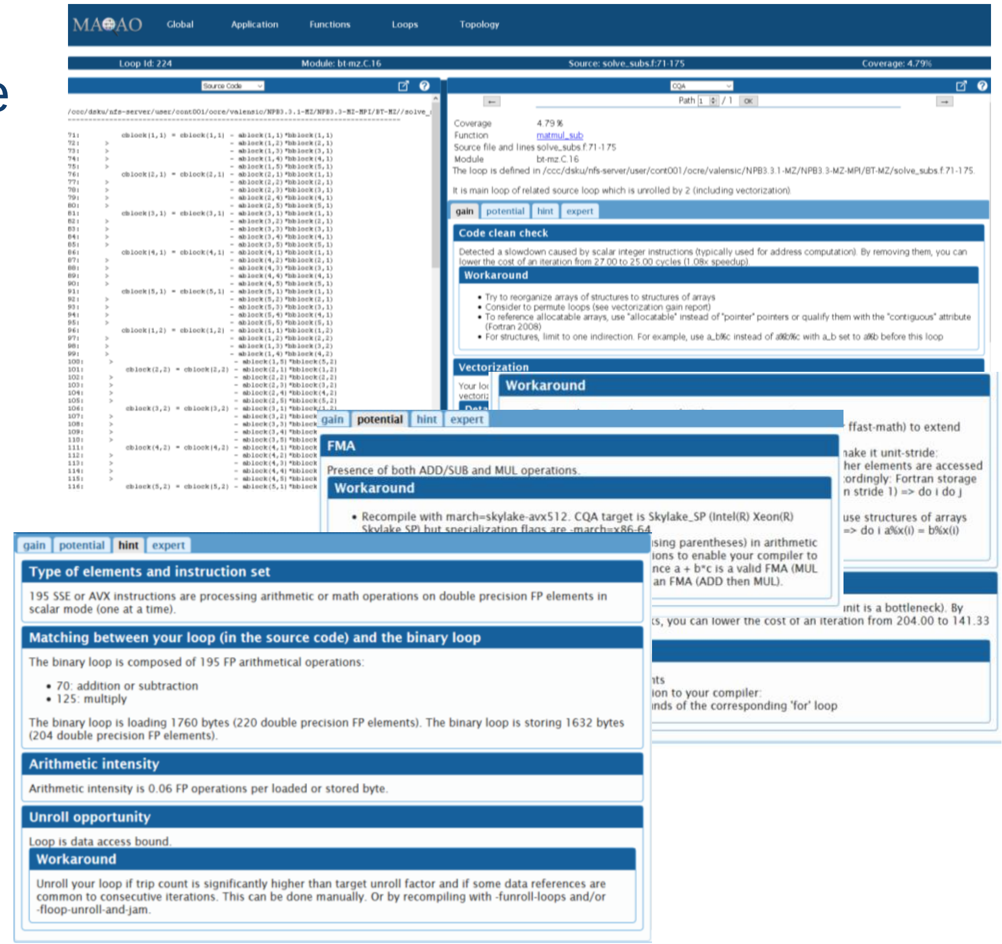
☑Coverage (%) ☑Level ☑Time (s) ☑Vectorization Ratio (%) ☑Speedup If Clean ☑Speedup If FP Vectorized ☑Speedup If Fully Vectorized ☑Speedup If Data in L1 ☐Select none

Loop id	Source Location	Source Function	Coverage (%)	Level	Time (s)	Vectorization Ratio (%)	Speedup If Clean	Speedup If FP Vectorized	Speedup If Fully Vectorized	Speedup If Data in L1
26831	qmcpack:ParticleBConds3DSoa.h:231-262	qmcplusplus::SoaDistanceTableAA::moveOnSphere	21.06	Single	2.54	100	1	1	1	1.05
19042	qmcpack:MultiBsplineValue.hpp:56-57	qmcplusplus::BsplineSet >::evaluate	15.42	Innermost	1.86	100	1	1	1	2.6
19064	qmcpack:MultiBsplineVGLH.hpp:187-207	qmcplusplus::BsplineSet >::evaluate	9.45	Innermost	1.14	100	1.03	1	1	2.4
13028	qmcpack:BsplineFunctor.h:639-643	qmcplusplus::J2OrbitalSoA >::ratio	4.64	Innermost	0.56	0	1.38	1	8	1.08
19116	qmcpack:MultiBsplineVGLH.hpp:187-207	qmcplusplus::BsplineSet >::evaluate_notranspose	4.31	Innermost	0.52	100	1.03	1	1	3.79
26829	qmcpack:ParticleBConds3DSoa.h:231-262	qmcplusplus::SoaDistanceTableAA::evaluate	3.98	Single	0.48	100	1	1	1	1.09
26833	qmcpack:ParticleBConds3DSoa.h:231-262	qmcplusplus::SoaDistanceTableAA::move	3.32	Single	0.4	100	1	1	1	NA
19142	qmcpack:SplineC2RAdoptor.h:323-373	void qmcplusplus::SplineC2RSoA::assign_vgl >, qmcplusplus::Vector, std::allocator > > >	2.49	Single	0.3	56.36	1	1	1.78	1.51
19038	qmcpack:SplineC2RAdoptor.h:224-234	qmcplusplus::BsplineSet >::evaluate	1.33	Single	0.16	48.39	1	1	1.69	1.66
[illegible]	qmcpack:ParticleBConds3DS	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

MAQAO ONE View Loop Analysis Report

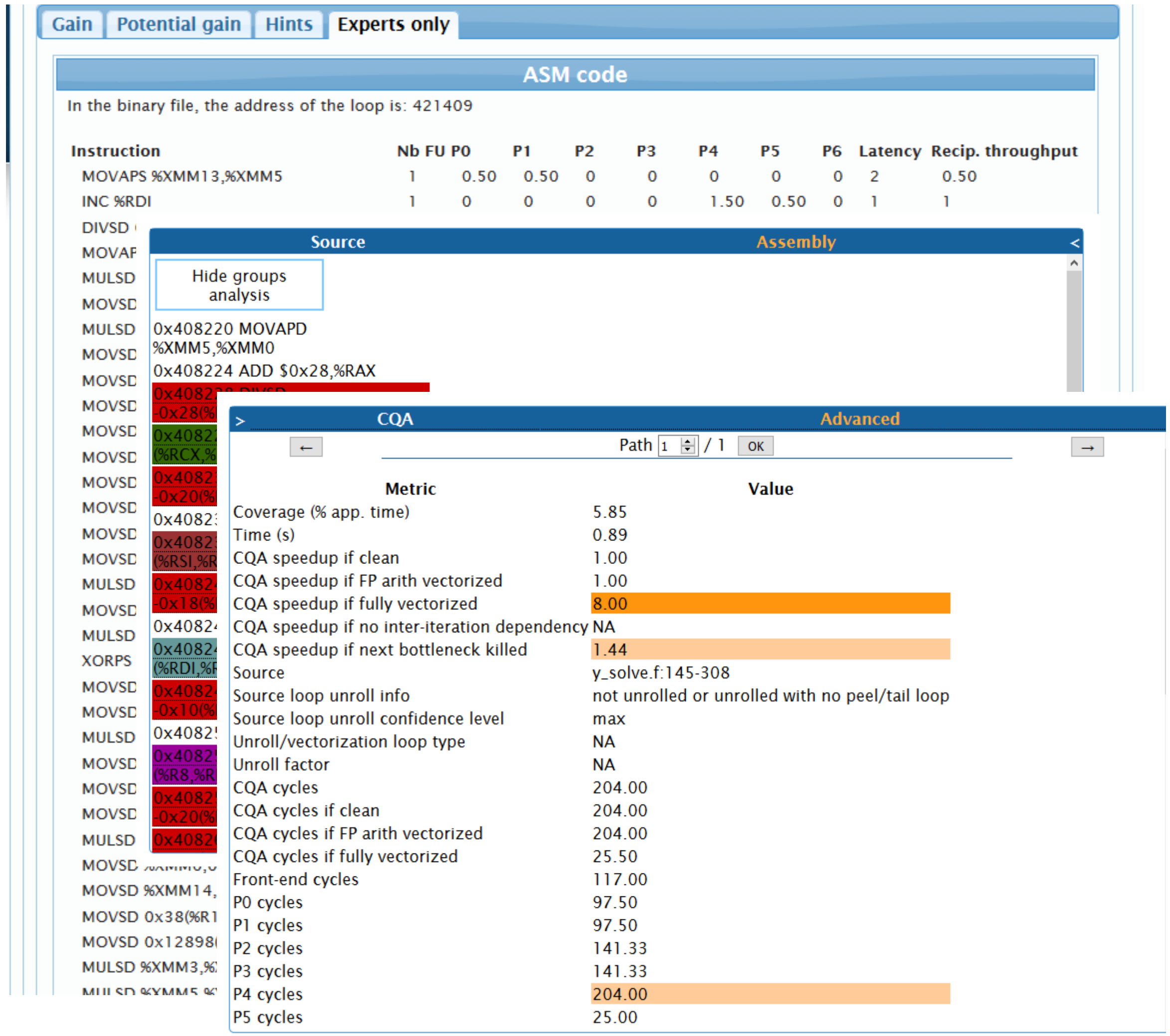
High level reports

- 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 Global Application Functions Loops Topology Help

Software Topology

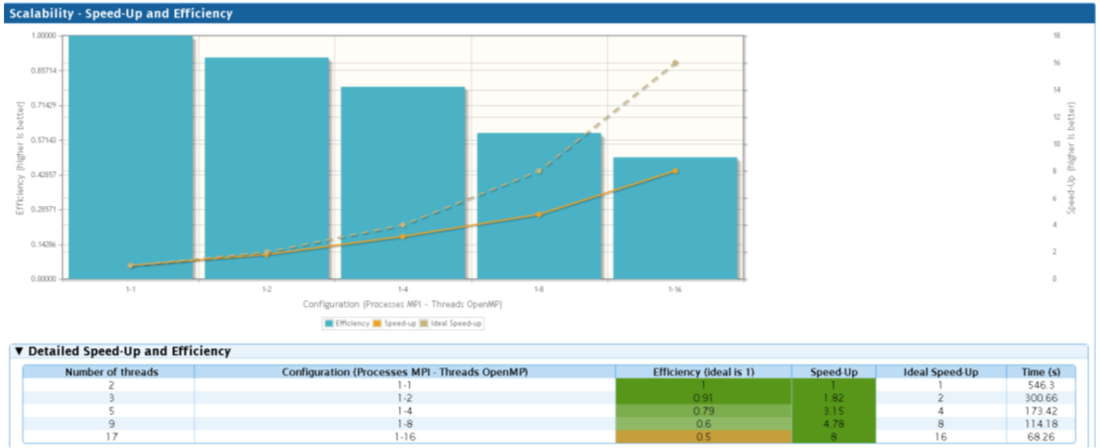
ID	Time(s)
▼ Node skylake01	11.22
▼ Process 359337	11.22
○ Thread 359337	11.22
▶ Process 359338	11.16
▶ Process 359352	11.22
▶ Process 359351	11.2
▶ Process 359353	11.22
▶ Process 359354	11.22
▶ Process 359355	11.18
▶ Process 359356	11.18
▶ Process 359357	11.22
▶ Process 359358	11.18
▶ Process 359359	11.18
▶ Process 359360	11.16
▶ Process 359361	11.18
▶ Process 359362	11.08
▶ Process 359364	11.22
▶ Process 359366	11.19
○ AVERAGE	11.22

MAQAO Global Application Functions Loops Topology Help

Profiling node skylake01 - process 359337 - thread 359337

Name	Module	Coverage (%)	Time (s)
○ MPIDI_CH3I_Progress	libmpi.so.12.0	20.62	2.31
▶ calc_data_gradient	3D_cylinder	4.95	0.56
▶ ics_advance_velocity_tfv4a_4th	3D_cylinder	3.75	0.42
▶ calc_data_tridiag_op_product	3D_cylinder	3.58	0.4
○ MPIR_Allreduce_group	libmpi.so.12.0	3.22	0.36
▶ filter_real_data	3D_cylinder	2.43	0.27
▶ update_int_comm	3D_cylinder	2.42	0.27
○ system_call_after_swapgs	SYSTEM CALL	1.66	0.19
▶ adv_scalar_w_u_tfv4a_4th	3D_cylinder	1.59	0.18
▶ solve_linear_system_deflated_pcg	3D_cylinder	1.45	0.16

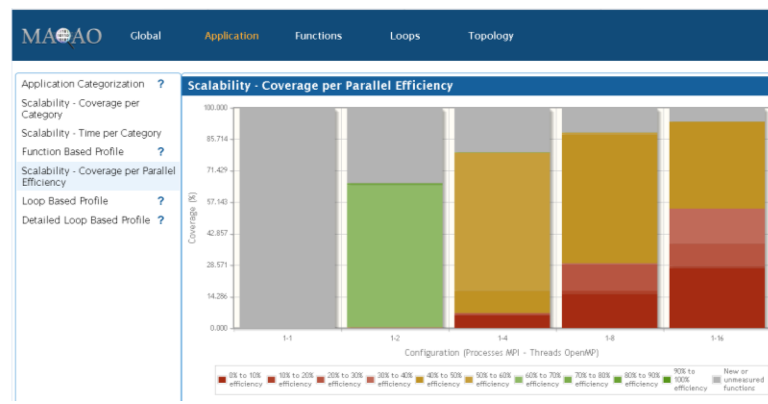
- Goal: Provide a view of the application scalability
  - Profiles with different numbers of threads/processes
  - Displays efficiency metrics for application



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

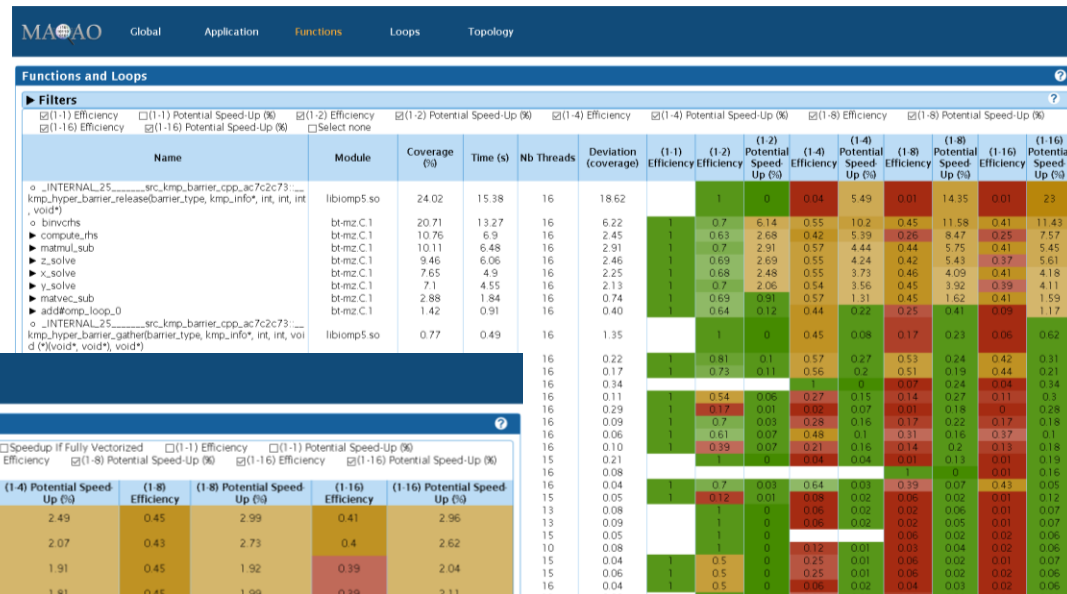
- Coverage per category
    - Comparison of categories for each run

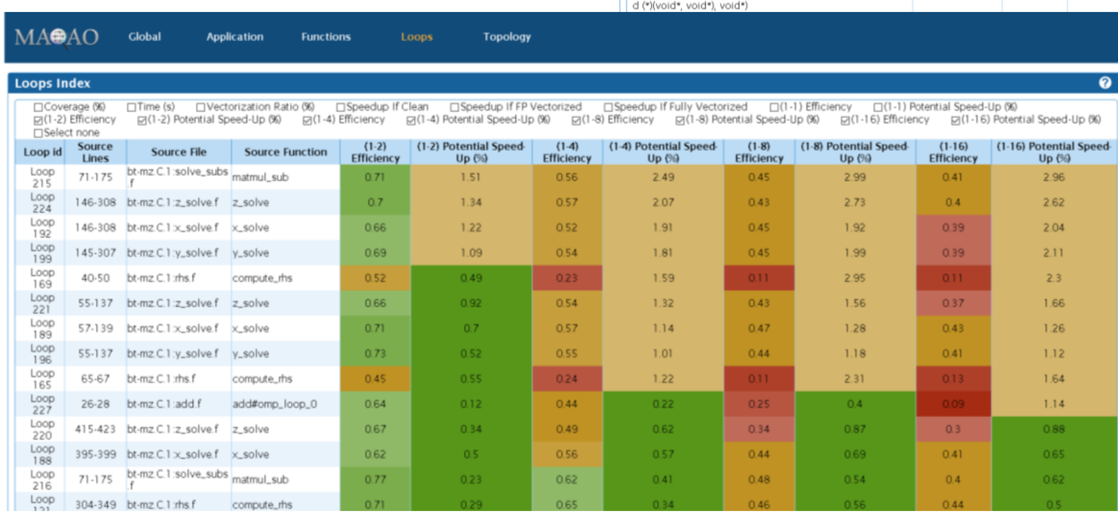
- Coverage per parallel efficiency
    - $Efficiency = \frac{T_{sequential}}{T_{parallel} * N_{threads}}$
        - Distinguishing functions only represented in parallel or sequential
    - Displays efficiency by coverage



Displays metrics for each function/loop

- Efficiency
- Potential speedup if efficiency=1





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 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 ?