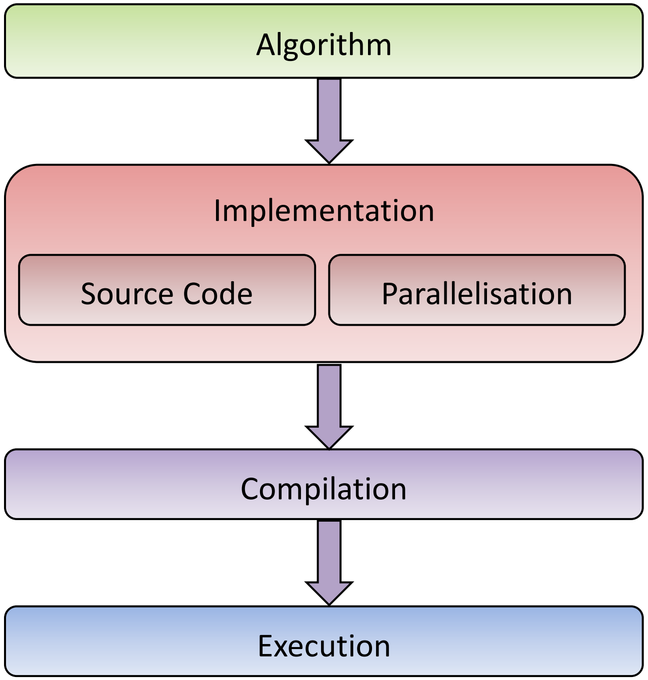
MAQAO
Performance Analysis and Optimization Tool

Performance Evaluation Team, University of Versailles

http://www.maqao.org

Performance Analysis and Optimisation

- **Where** is the application spending most execution time and resources?
- **Why** is the application spending time there?
  - Algorithm, implementation, runtime or hardware?
  - Data access or computation?
- **How** to improve the application?
  - At which step(s) of the design process?
  - What additional information is needed?
- **How much** gain can be expected?
  - What would the effort/gain ratio be?



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
```

Where are the bottlenecks?

Motivating Example

- Code of a loop representing ~10% walltime

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
```

1) High number of statements

4) DIV/SQRT

3) Indirect accesses

5) Reductions

2) Non-unit stride accesses

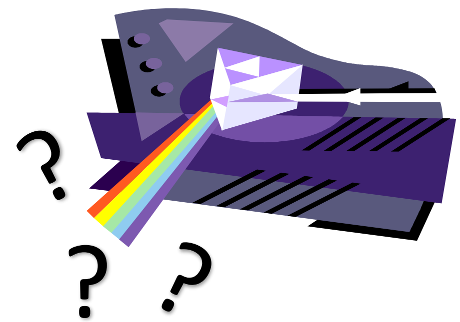
1) High number of statements
2) Non-unit stride accesses
3) Indirect accesses
4) DIV/SQRT
5) Reductions
6) Variable number of iterations

Which is the dominant one?

➔ Need analysis tools to evaluate performance issues

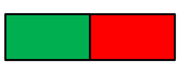
A Multifaceted Problem

- **What type of problems are we facing?**
  - CPU or data access problems
  - Identifying the dominant issues: Algorithms, implementation, parallelisation, …
- What transformations to apply?
  - Compiler switches, partial/full vectorisation
  - Loop blocking/array restructuring, If removal, Full unroll
  - Binary transforms (prefetch)
  - …



- 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**

Our Approach

Nobody wants problems everybody wants solutions ☺

- Our primary targets:
  - Developers: code experts but not performance experts
  - Benchmarkers: performance experts but not code experts
- Focusing on the knobs that code developers can operate:
  - Compiler flags and runtime settings
  - Code restructuring: Change loop/parallel construct body (remove dependencies, simplify control flow, …), insert pragmas …
  - Data restructuring
- Helping the developer/benchmarker in using these knobs

➔ Instead of pinpointing problems, guiding the user towards a way to address them.

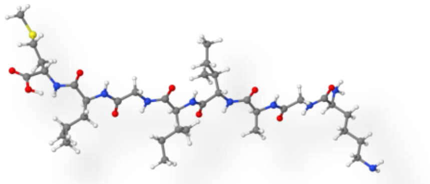
MAQAO:
Modular Assembly Quality Analyzer and Optimizer

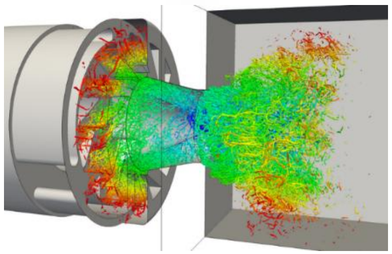
- Objectives:
  - Characterizing performance of HPC applications
  - Focusing on performance at the **core level**
  - **Guiding users** through optimization process
  - Estimating return of investment (**R.O.I.**)
- Characteristics:
  - Support for **x86-64** and **Aarch64** (beta)
    - On-going development on GPU support
  - **Modular tool** offering complementary views
  - LGPL3 Open Source software
  - Developed at UVSQ since 2004
  - Binary release available as **static executable**
- Philosophy: Analysis at Binary Level
  - Compiler optimizations increase the distance between the executed code and the source code
  - Source code instrumentation may prevent the compiler from applying certain transformations

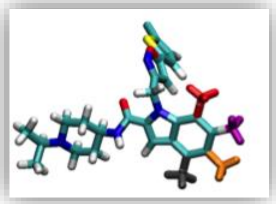
➔ What You Analyse Is What You Run

Success stories:
Optimisation of Industrial and Academic HPC Applications

- 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
- Ongoing effort
  - TREX CoE project codes
  - CEA DAM codes







Partnerships

- MAQAO is part of the POP Centre of Excellence
  - Provides performance optimisation and productivity services for academic and industrial codes
  - https://pop-coe.eu/

- MAQAO has been funded by UVSQ, Intel and CEA (French department of energy) through Exascale Computing Research (ECR) and various European projects (H4H, COLOC, PerfCloud, ELCI, MB3, POP2 CoE, TREX CoE, etc...)

- Provides core technology to be integrated with other tools:
  - TAU performance tools with MADRAS patcher through MIL (MAQAO Instrumentation Language)
    - X86_64 only, aarch64 under development
  - Intel Advisor

MAQAO Team and Collaborators

- **MAQAO Team**
  - William Jalby, Prof.
  - Cédric Valensi, Ph.D.
  - Emmanuel Oseret, Ph.D.
  - Mathieu Tribalat, M.Sc.Eng
  - Salah Ibn Amar, M.Sc.Eng
  - Hugo Bolloré , M.Sc.Eng
  - Kévin Camus, Eng.
  - Max Hoffer, Eng.
  - Aurélien Delval, Eng.

- **Collaborators**
  - David J. Kuck, Prof. (Intel Fellow)
  - Eric Petit, Ph.D. (Intel)
  - Pablo de Oliveira, Ph.D. (McF UVSQ)
  - David Wong, Ph.D. (Intel)
  - Othman Bouizi, Ph.D. (Intel)
  - AbdelHafid Mazouz Ph.D.(Intel)
  - Jeongnim Kim (Intel)

- **Past Collaborators or Team members**
  - *Andrés S. Charif-Rubial, Ph.D.*
  - Denis Barthou, Prof. (Univ. Bordeaux)
  - Jean-Thomas Acquaviva, Ph.D. (DDN)
  - Stéphane Zuckerman, Ph.D. (McF Univ Cergy)
  - Julien Jaeger, Ph.D. (CEA DAM)
  - Souad Koliaï, Ph.D. (CELOXICA)
  - Zakaria Bendifallah, Ph.D. (ATOS)
  - Tipp Moseley, Ph.D. (Google)
  - Jean-Christophe Beyler, Ph.D. (Google)
  - Vincent Palomarès, Ph.D. (Google)
  - José Noudohouenou, Ph.D. (AMD)
  - Franck Talbart, M. Sc. Eng (DDN)
  - Nicolas Triquenaux, Ph.D. (DDN)
  - Jean-Baptiste Le Reste , M.Sc.Eng
  - Sylvain Henry, Ph.D.
  - Aleksandre Vardoshvili , M.Sc.Eng
  - Romain Pillot, Eng
  - Youenn Lebras, Ph.D.

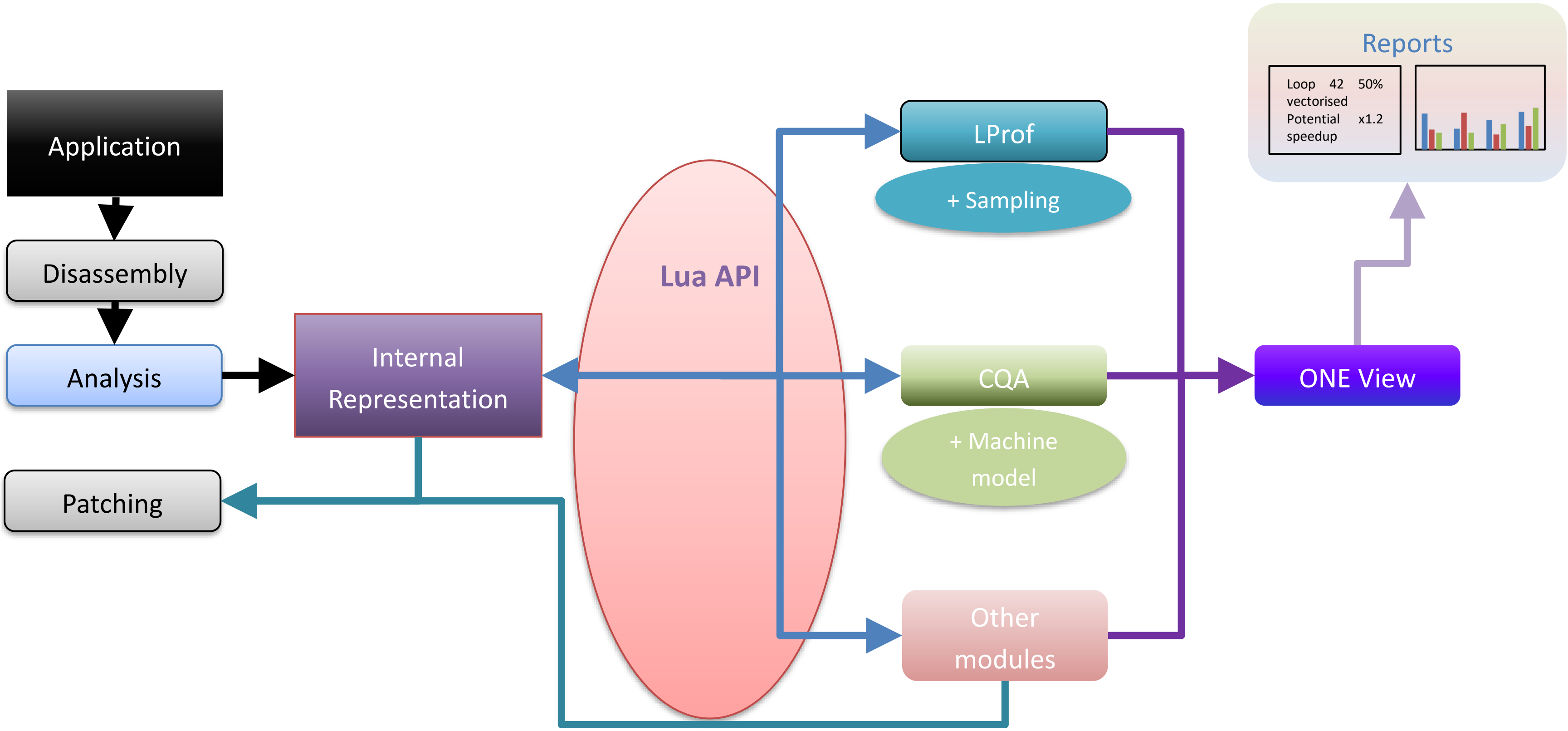
Website & resources

- MAQAO website: www.maqao.org
  - Mirror: https://maqao.liparad.uvsq.fr
- 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: http://www.maqao.org/publications.html
- Repository of MAQAO analyses: http://datafront.exascale-computing.eu/public
- Email: contact@maqao.org

MAQAO Main Features

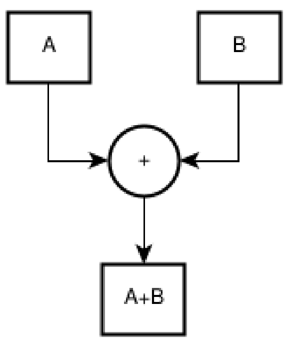
- Binary layer
  - Builds internal representation from binary
    - Construct high level structures (CFG, DDG, SSA, …)
    - Links binary instructions to source code
    - ⚠ A single source loop can be compiled as multiple assembly loops ➔ Affecting unique identifiers to loops
  - Allows patching through binary rewriting
- Profiling
  - LProf: Lightweight sampling-based Profiler operating at process, thread, function and loops level
- Static analysis
  - CQA (Code Quality Analyzer): Evaluates the quality of the binary code and offers hints for improving it
- Performance view aggregation module: ONE View
  - Goal: Guiding the user through the analysis & optimization process.
  - Synthesizes information provided by different MAQAO modules
  - Automatizes execution of experiments invoking other MAQAO modules and aggregates their results to produce high-level reports in HTML or XLSX format

MAQAO Main Structure

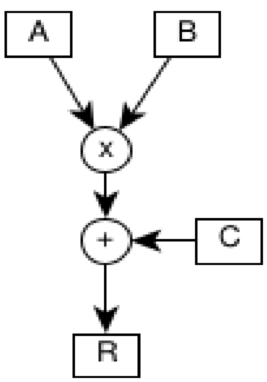


SIMD/Vectorization/Data Parallelism

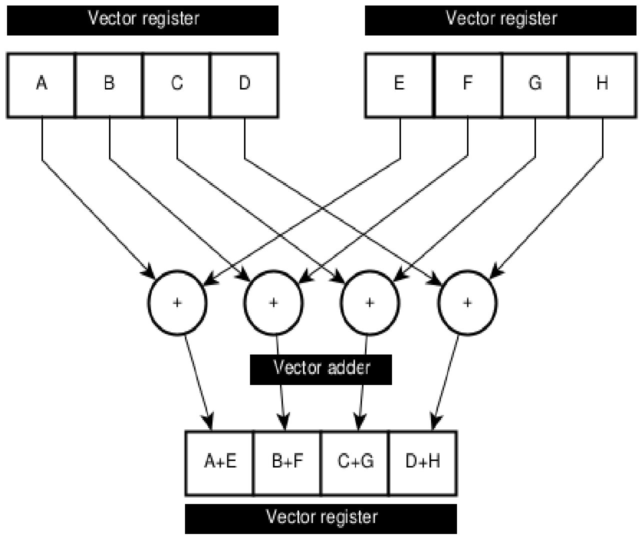
- Scalar pattern (C): a[i] = b[i] + c[i]
- Vector pattern (FORTRAN): a(i, i + 8) = b(i, i + 8) + c(i, i + 8)
- Benefits : increases memory bandwidth and **IPC**
- Implementations:
  - x86 : SSE, AVX, AVX512
  - ARM : Neon, SVE
- FMA/MAC: (the core operation of LinAlg/DSP algorithms)
  - Fused-Multiply-Add
  - Multiply-Accumulate



Scalar addition



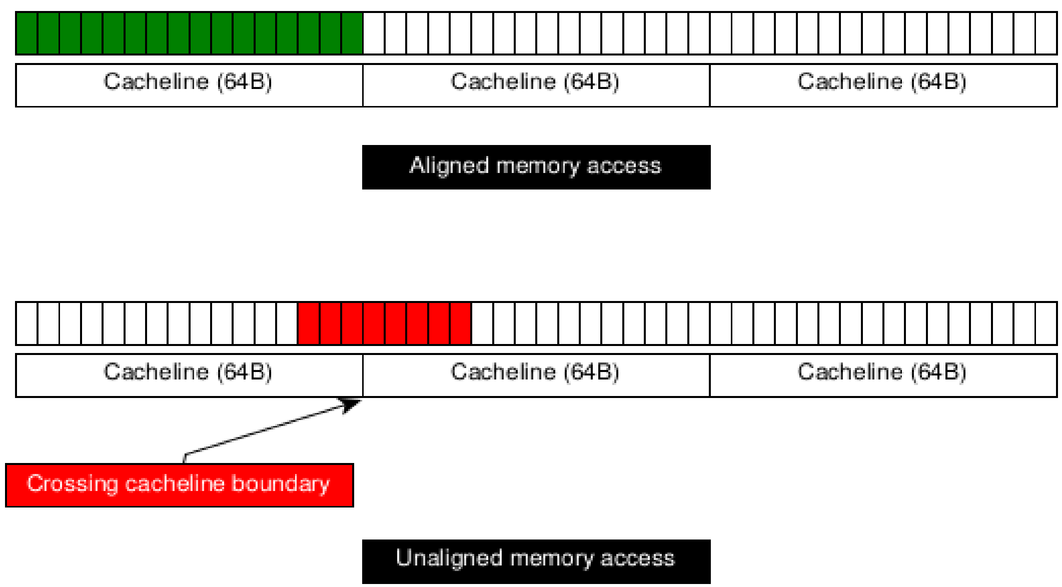
FMA / MAC



Vector addition

Memory and caches

- Computations are, in general, faster than memory accesses
- Alignment/Contiguity of memory (x86) : posix_memalign, aligned_alloc, ...
- Are caches (L1, L2, L3) used properly?
- Memory performance → Maximum bandwidth

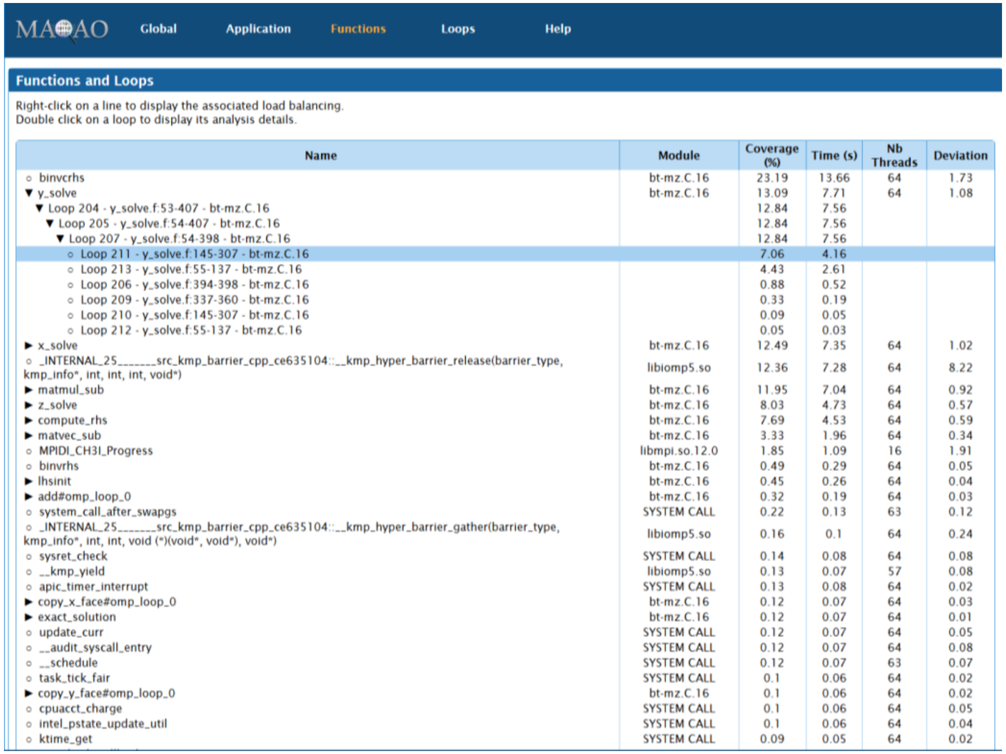


Compiler optimisations

- Compiler flags:
  - Loop unrolling: -funroll-loops
    - Reduce branches
    - Fill the pipeline (more instructions per iteration)
    - Increases memory bandwidth and IPC
  - Function inlining: -finline-functions
  - Vectorization: -ftree-vectorize, -ftree-slp-vectorize, ...
  - Target micro-architectures: -march or -mtune or -xHOST
- Compiler directives:
  - OpenMP directives: #pragma omp simd, #pragma omp parallel for, …
  - Intel compiler specific: #pragma simd, #pragma unroll, #pragma inline, …
- Compiler/language keywords/features:
  - Using restrict for pointers aliasing in C/C++
  - Using inline for function inlining in C
  - Using array sections in FORTRAN

MAQAO LProf: Lightweight Profiler

- **Goal**: Localization of application hotspots
- Features:
  - **Lightweight**
  - **Sampling** based
  - Access to hardware counters
  - Analysis at **function** and **loop** granularity
- Strengths:
  - **Non intrusive**: No recompilation necessary
  - **Low overhead**
  - Agnostic with regard to parallel runtime


MAQAO Global Application Functions Loops Help

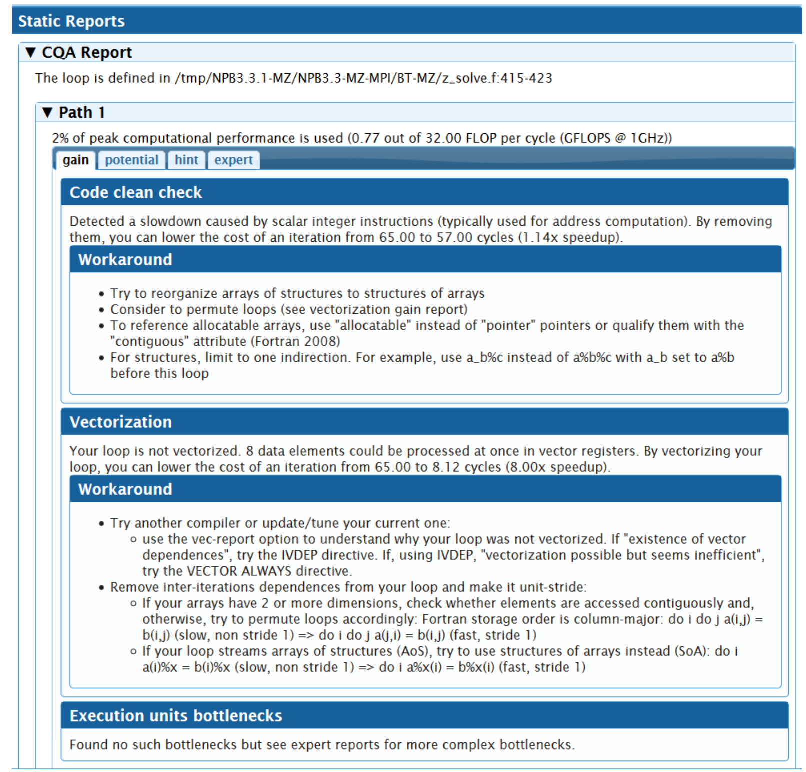
Functions and Loops

Right-click on a line to display the associated load balancing.
Double click on a loop to display its analysis details.

Name	Module	Coverage (%)	Time (s)	Nb Threads	Deviation
binvcrhs	bt-mz.C.16	23.19	13.66	64	1.73
y_solve	bt-mz.C.16	13.09	7.71	64	1.08
Loop 204 - y_solve.f:53-407 - bt-mz.C.16		12.84	7.56		
Loop 205 - y_solve.f:54-407 - bt-mz.C.16		12.84	7.56		
Loop 207 - y_solve.f:54-398 - bt-mz.C.16		12.84	7.56		
Loop 211 - y_solve.f:145-307 - bt-mz.C.16		7.06	4.16		
Loop 213 - y_solve.f:55-137 - bt-mz.C.16		4.43	2.61		
Loop 206 - y_solve.f:394-398 - bt-mz.C.16		0.88	0.52		
Loop 209 - y_solve.f:337-360 - bt-mz.C.16		0.33	0.19		
Loop 210 - y_solve.f:145-307 - bt-mz.C.16		0.09	0.05		
Loop 212 - y_solve.f:55-137 - bt-mz.C.16		0.05	0.03		
x_solve	bt-mz.C.16	12.49	7.35	64	1.02
_INTERNAL_25_______src_kmp_barrier_cpp_ce635104::__kmp_hyper_barrier_release(barrier_type, kmp_info*, int, int, int, void*)	libiomp5.so	12.36	7.28	64	8.22
matmul_sub	bt-mz.C.16	11.95	7.04	64	0.92
z_solve	bt-mz.C.16	8.03	4.73	64	0.57
compute_rhs	bt-mz.C.16	7.69	4.53	64	0.59
matvec_sub	bt-mz.C.16	3.33	1.96	64	0.34
MPIDI_CH3I_Progress	libmpi.so.12.0	1.85	1.09	16	1.91
binvrhs	bt-mz.C.16	0.49	0.29	64	0.05
lhsinit	bt-mz.C.16	0.45	0.26	64	0.04
add#omp_loop_0	bt-mz.C.16	0.32	0.19	64	0.03
system_call_after_swapgs	SYSTEM CALL	0.22	0.13	63	0.12
_INTERNAL_25_______src_kmp_barrier_cpp_ce635104::__kmp_hyper_barrier_gather(barrier_type, kmp_info*, int, int, void (*)(void*, void*), void*)	libiomp5.so	0.16	0.1	64	0.24
sysret_check	SYSTEM CALL	0.14	0.08	64	0.08
__kmp_yield	libiomp5.so	0.13	0.07	57	0.08
apic_timer_interrupt	SYSTEM CALL	0.13	0.08	64	0.02
copy_x_face#omp_loop_0	bt-mz.C.16	0.12	0.07	64	0.03
exact_solution	bt-mz.C.16	0.12	0.07	64	0.01
update_curr	SYSTEM CALL	0.12	0.07	64	0.05
__audit_syscall_entry	SYSTEM CALL	0.12	0.07	64	0.08
__schedule	SYSTEM CALL	0.12	0.07	63	0.07
task_tick_fair	SYSTEM CALL	0.1	0.06	64	0.02
copy_y_face#omp_loop_0	bt-mz.C.16	0.1	0.06	64	0.02
cpuacct_charge	SYSTEM CALL	0.1	0.06	64	0.05
intel_pstate_update_util	SYSTEM CALL	0.1	0.06	64	0.04
ktime_get	SYSTEM CALL	0.09	0.05	64	0.02

MAQAO CQA: Code Quality Analyzer

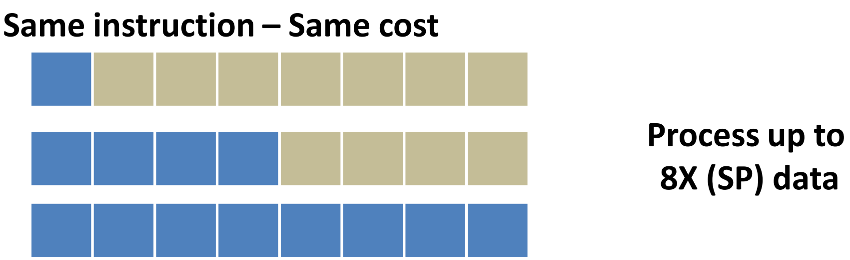
- Goal: **Assist developers** in improving code performance
- Features:
  - Static analysis: **no execution** of the application
  - Allows **cross-analysis** of/on multiple architectures
  - **Evaluates the quality** of compiler generated code
  - Proposes **hints and workarounds** to improve quality / performance
  - **Loop centric**
    - In HPC loops cover most of the processing time
  - Targets **compute-bound** codes



MAQAO CQA Main Concepts

- Applications only exploit at best 5% to 10% of the peak performance

- Main elements of analysis:
  - Peak performance
  - Execution pipeline
  - Resources/Functional units



- Key performance levers for core level efficiency:
  - Vectorising
  - Avoiding high latency instructions if possible (e.g. DIV/SQRT)
  - Guiding the compiler code optimisation
  - Reorganizing memory and data structures layout

“What If” Scenarios: Vectorization

- Code “Clean”
  - Generate an Assembly “Clean” variant : **keep only FP Arithmetic and Memory operations, suppress all other**
  - Generate a CQA Performance estimate on the “Clean” Variant
- Code “FP Vector”
  - Generate an Assembly “FP Vector” variant : **only replace scalar FP Arithmetic by Vector FP Arithmetic equivalent**. Generate additional instructions to fill in Vector Registers.
  - Generate a CQA Performance estimate
- Code “Full Vector”
  - Generate an Assembly “Full Vector” variant **: replace both scalar FP Arithmetic and FP Load/Store by their Vector equivalent**.
  - Generate a CQA Performance estimate
- All of these “What If Scenarios” are generated in a fully static manner.

MAQAO CQA Guiding the compiler and hints

- Compiler can be driven using flags, pragmas and keywords:
  - 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)

- Hints for implementation changes
  - Improve data access
    - Memory alignment
    - Loop interchange
    - Change loop stride
    - Reshaping arrays of structures
  - Avoid instructions with high latency (SQRT, DIV, GATHER, SCATTER, …)

MAQAO CQA Advanced Features
Vector Efficiency

- Ex: vectorized SSE code on AVX machine
- Compiler: “LOOP WAS VECTORIZED”
- In reality 50% vectorization speedup loss
- CQA:
  - vectorization ratio: 100% (“all instructions vectorized”)
  - vec. efficiency ratio: 50% (“but using only half vector width”)
  - hint: “recompile with –xHost” (on Intel compilers)

128 bits vectorized:	256 bits vectorized:
Vec. ratio = 100%	*Vec. ratio = 100%*
`ADDPD (xmm)`	`VADDPD (ymm)`
`MULPD (xmm)`	`VMULPD (ymm)`
`etc…`	`etc…`
50 %	100 %

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
```

1) High number of statements

7) Vector vs scalar

4) DIV/SQRT

3) Indirect accesses

5) Reductions

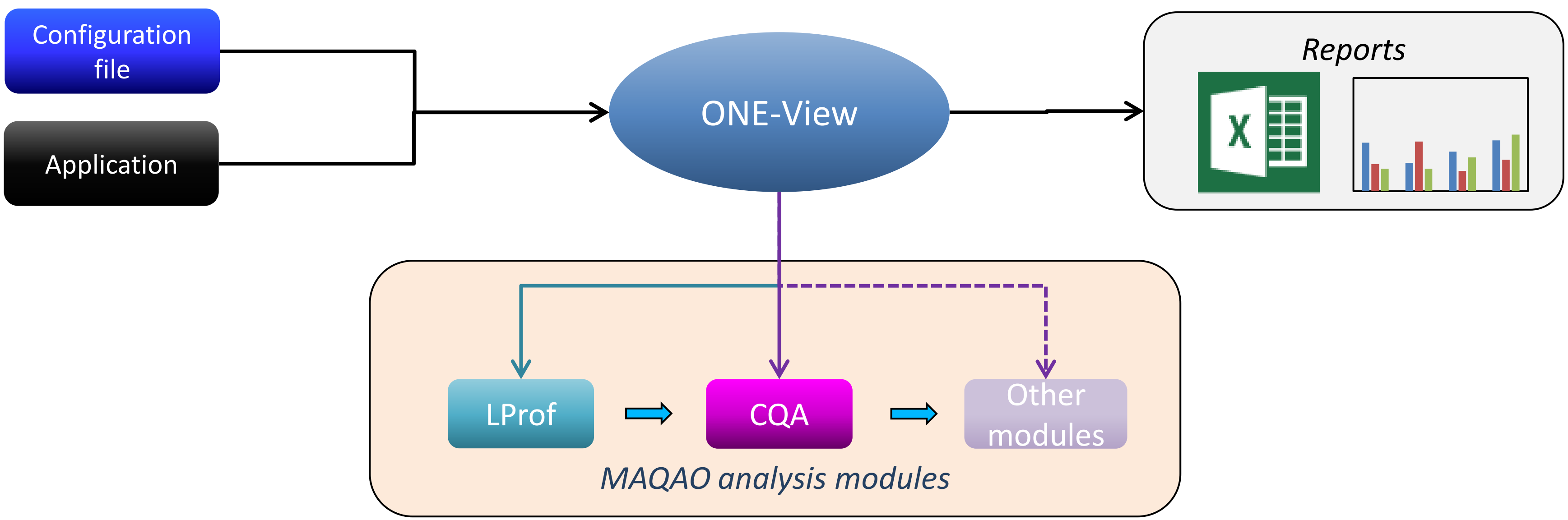
2) Non-unit stride accesses

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**

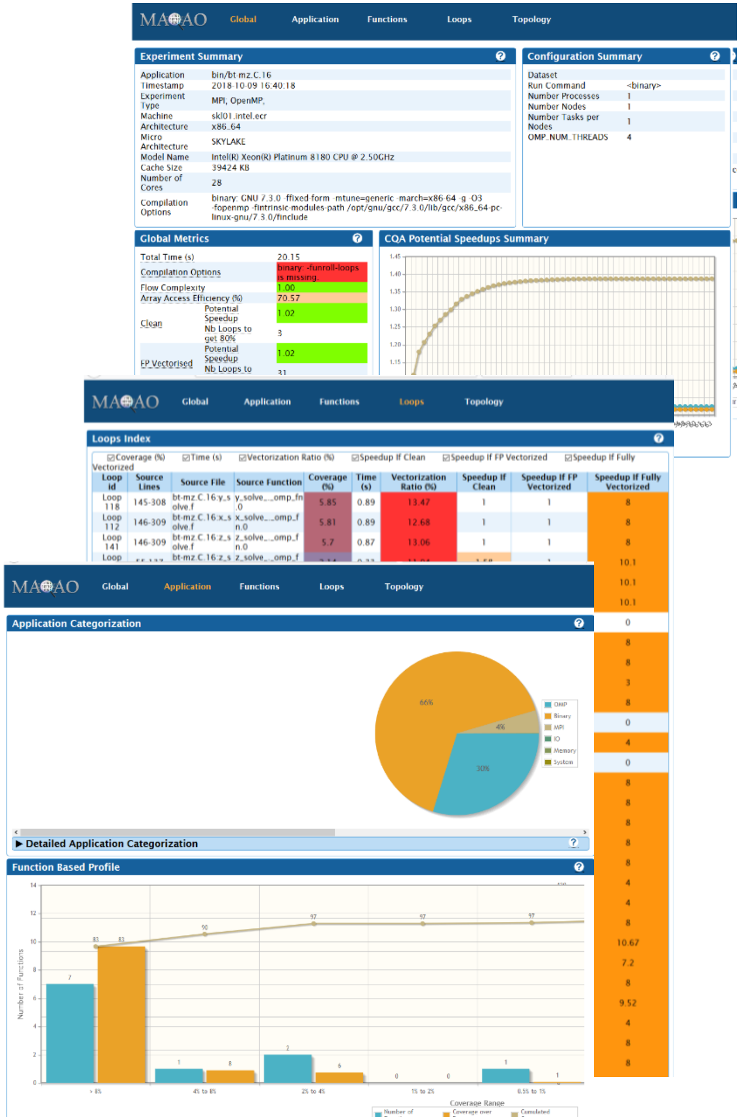
MAQAO ONE View: Performance View Aggregator

- Goal: **Automating** the whole analysis process
  - Invoke multiple MAQAO modules
  - Generate **aggregated performance views**
    - Reports in HTML or XLS format



MAQAO ONE View: Performance View Aggregator

- Main steps:
  - Invokes LProf to **identify hotspots**
  - Invokes CQA on **loop hotspots**
- Available results:
  - **Speedup** predictions
  - **High-level** summary
  - Global code **quality** metrics
  - **Hints** for improving performance
  - Detailed analyses results
  - Parallel **efficiency** analysis



ONE View Reports Levels

- ONE VIEW ONE
  - Requires a single run of the application
  - Profiling of the application using LProf
  - Static analysis using CQA
- Scalability
  - Requires as many additional runs as parallel configurations
  - Can be executed in addition to another report
  - Profiling using LProf on different parallel configurations
- Comparison mode
  - Comparison of multiple runs (iso-binary or iso-source)
  - Allows to compare performance across different datasets, compilers, or hardware platforms
- Stability mode
  - Multiple runs with identical parameters
  - Allows to assess the stability of execution time

Comparative Analysis

- Basic principles: run different “code versions” and compare them on “appropriate levels”.
- TRIAL AND ERROR and comparison are fundamental techniques in scientific approach.
- Different “code versions”
  - Different runtime settings (on different number of cores, etc..)
  - Different compilers
  - Different hardware (X86, ARM, …) with same or different ISA
  - Different code versions
- “Appropriate levels”:
  - ISOBINARY: the same binary is compared in different settings
  - ISOSOURCE: the same source is compared
  - ISOFUNCTION STRUCTURE: the source code can be different but the function structure is preserved.
  - Generic: much harder to compare
- **Not very sophisticated at first but very useful and implementation is a bit subtle**

Analysing an application with MAQAO

- ONE View execution
- Provide all parameters necessary for executing the application
  - Parameters can be passed on the command line or as a configuration file
  - Parameters include binary name, MPI commands, dataset directory, …

```
$ maqao oneview --create-report=one --executable=bt-mz.C.16 --mpi_command="mpirun -n 16"
```

```
$ maqao oneview --create-report=one --config=my_config.json"
```

- Analyses can be tweaked if necessary
  - Report level **one** corresponds to lightweight profiling (**LProf**) and code quality analysis (**CQA**)
- ONE View can reuse an existing experiment directory to perform further analyses
- Results available in HTML format by default
  - XLS spreadsheets and textual output generation are also available
- Online help is available:

```
$ maqao oneview --help
```

Analysing an application with MAQAO

MAQAO modules can be invoked separately for advanced analyses

- LProf
  - Profiling

```
$ maqao lprof xp=exp_dir --mpi-command="mpirun -n 16" -- ./bt-mz.C.16
```

  - Display functions profile

```
$ maqao lprof xp=exp_dir -df
```

  - Displaying the results from a ONE View run

```
$ maqao lprof xp=oneview_xp_dir/lprof_npsu -df
```

- CQA

```
$ maqao cqa loop=42 bt-mz.C.16
```

Online help is available:

```
$ maqao lprof --help
```

```
$ maqao cqa --help
```

Questions ?

THANKS FOR YOUR ATTENTION!

NAVIGATING ONE VIEW REPORTS

MAQAO ONE View Global Summary

- 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

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

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

TYPICAL ONE VIEW GLOBAL TAB

Global Metrics		
Total Time (s)		63.86
Profiled Time (s)		61.31
Time in analyzed loops (%)		61.6
Time in analyzed innermost loops (%)		61.2
Time in user code (%)		61.6
Compilation Options		OK
Perfect Flow Complexity		1.01
Iterations Count		1.00
Array Access Efficiency (%)		88.3
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.02
	Nb Loops to get 80%	7
FP Vectorised	Potential Speedup	1.01
	Nb Loops to get 80%	4
Fully Vectorised	Potential Speedup	1.04
	Nb Loops to get 80%	11
FP Arithmetic Only	Potential Speedup	1.16
	Nb Loops to get 80%	11

FOCUS: on transformations and impact at the application level

Perfect flow complexity: evaluate performance gain if innermost loops had no branches

Iteration count: evaluate the impact of having all loop iteration count over 100

Array Access Efficiency: Percentage of Unit Stride access

TYPICAL ONE VIEW GLOBAL TAB

Global Metrics

Total Time (s)		63.86
Profiled Time (s)		61.31
Time in analyzed loops (%)		61.6
Time in analyzed innermost loops (%)		61.2
Time in user code (%)		61.6
Compilation Options		OK
Perfect Flow Complexity		1.01
Iterations Count		1.00
Array Access Efficiency (%)		88.3
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.02
	Nb Loops to get 80%	7
FP Vectorised	Potential Speedup	1.01
	Nb Loops to get 80%	4
Fully Vectorised	Potential Speedup	1.04
	Nb Loops to get 80%	11
FP Arithmetic Only	Potential Speedup	1.16
	Nb Loops to get 80%	11

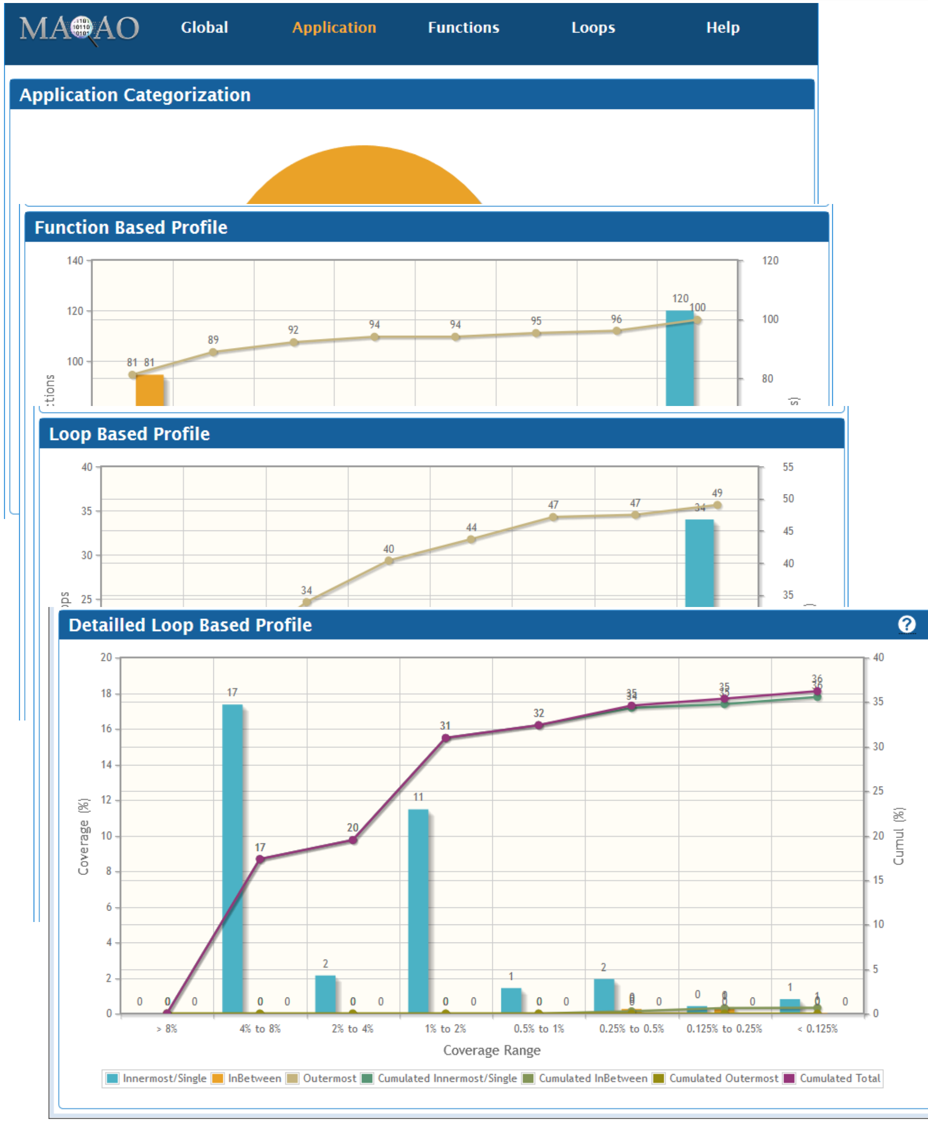
FOCUS: on transformations and impact at the application level

FP vectorized: Performance gain if all the FP arithmetic operations were vectorized

Fully vectorized: Performance gain if all the FP arithmetic operations+ Load/Store instructions were vectorize

MAQAO ONE View Application Characteristics

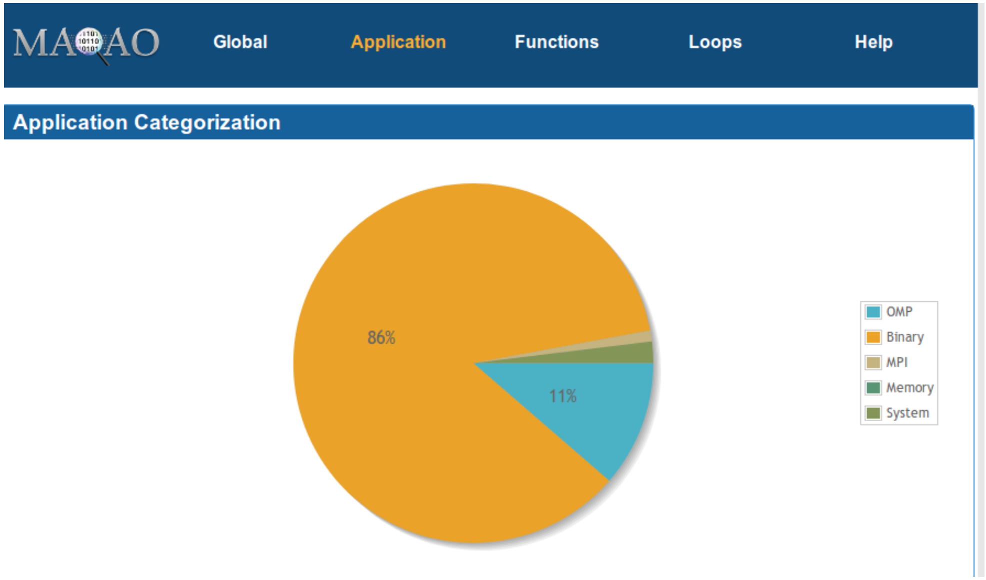
- Application categorisation
  - Time spent in different regions of code
- Function based profile
  - Functions by coverage ranges
- Loop based profile
  - Loops by coverage ranges
- Detailed loop based profile
  - Loop types by coverage ranges



MAQAO ONE View Application Characteristics Time Categorisation

Identifying at a glance where time is spent

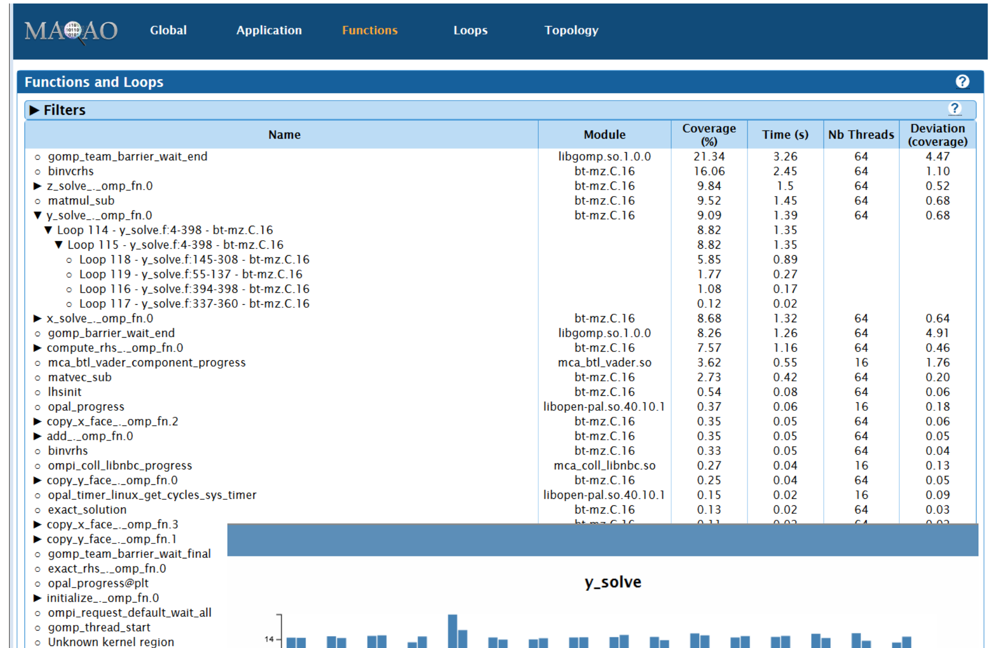
- Application
  - Main executable
- Parallelization
  - Threads
  - OpenMP
  - MPI
- System libraries
  - I/O operations
  - String operations
  - Memory management functions
- External libraries
  - Specialised libraries such as libm / libmkl
  - Application code in external libraries



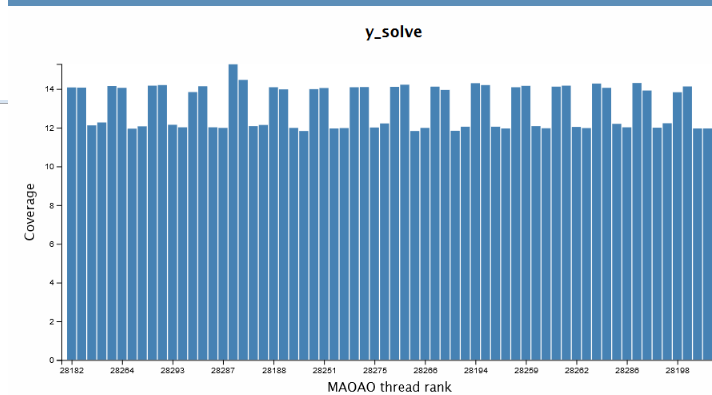
MAQAO ONE View: Functions Profiling

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 Innermost Profile | Open Expert Summary

Loops Index

▶ Filters

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

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
18403	qmcpack:MultiBsplineValue.hpp:56-57	qmcplusplus::BsplineSet >::evaluate	26.71	Innermost	3.61	100	1	1	1	8.25
26027	qmcpack:cmath:261-464	qmcplusplus::SoaDistanceTableAA::moveOnSphere	12.01	Single	1.62	100	1	1	1	1.03
18424	qmcpack:MultiBsplineVGLH.hpp:187-207	qmcplusplus::BsplineSet >::evaluate	10.81	Innermost	1.46	100	1.06	1	1	4.15
18474	qmcpack:MultiBsplineVGLH.hpp:187-207	qmcplusplus::BsplineSet >::evaluate_notranspose	4.84	Innermost	0.65	100	1.06	1	1	4.52
26026	qmcpack:cmath:261-464	qmcplusplus::SoaDistanceTableAA::evaluate	2.78	Single	0.38	100	1	1	1	1.05
26028	qmcpack:cmath:261-464	qmcplusplus::SoaDistanceTableAA::move	2.64	Single	0.36	100	1	1	1	1.03
8754	qmcpack:CoulombPBCAA.cpp:425-427	qmcplusplus::CoulombPBCAA::evalSR	1.57	Innermost	0.21	0	1.64	2.59	7.67	1.01
12711	qmcpack:BsplineFunctor.h:690-695	qmcplusplus::J2OrbitalSoA >::ratioGrad	1.41	Innermost	0.19	0	1.3	1	16	1.08
18501	qmcpack:SplineC2RAdoptor.h:325-373	void qmcplusplus::SplineC2RSoA::assign_vgl >, qmcplusplus::Vector, std::allocator > > >	1.22	Single	0.16	100	1.01	1	1	3.51

MAQAO ONE View Loop Expert Summary

- All metrics derived from CQA, VProf and DECAN analyses

Expert Summary

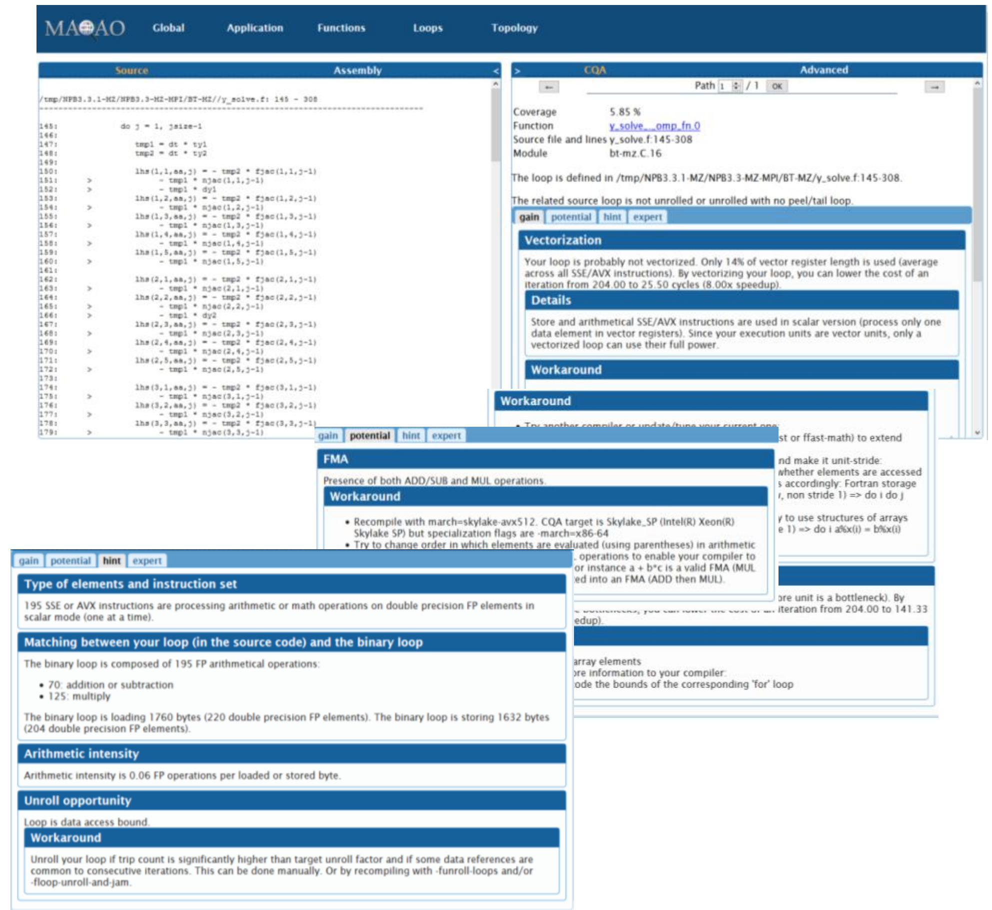
☑ Analysis ☐ CQA speedup if clean ☐ CQA speedup if FP arith vectorized ☐ CQA speedup if fully vectorized ☐ Number of paths ☑ ORIG / DL1 ☐ Saturation ratio (MAX(DL1,LS)/REF) ☐ Saturation
☐ FP/CQA(FP) ☐ DL1/CQA(DL1) ☑ FP/LS ☐ Frequency Impact ☑ ORIG (cycles per iteration) ☑ STA (ORIG) ☐ REF (cycles per iteration) ☐ STA (REF) ☑ FP (cycles per iteration) ☑ STA (FP)
☑ LS (cycles per iteration) ☑ STA (LS) ☑ DL1 (cycles per iteration) ☑ STA (DL1) ☑ FES (cycles per iteration) ☑ STA (FES) ☐ CQA cycles ☐ CQA cycles if clean ☐ CQA cycles if FP arith vectorized
☐ CQA cycles if fully vectorized ☑ Iteration count ☐ Function ☐ Source ☐ Nb FP_ADD / CPI ☐ Nb FP_MUL / CPI ☐ CAP(FP) ☐ BW(FP) ☐ SAT(FP) ☐ CAP(L1R) ☐ BW(L1R) ☐ SAT(L1R)
☐ CAP(L1W) ☐ BW(L1W) ☐ SAT(L1W) ☐ CAP(L2) ☐ BW(L2) ☐ SAT(L2) ☐ CAP(L3) ☐ BW(L3) ☐ SAT(L3) ☐ CAP(RAM_R) ☐ CAP(RAM_W) ☑ Select all

ID	Module	Coverage (% app. time)	Analysis	ORIG / DL1	FP/LS	ORIG (cycles per iteration)	STA (ORIG)	FP (cycles per iteration)	STA (FP)	LS (cycles per iteration)	STA (LS)	DL1 (cycles per iteration)	STA (DL1)	FES (cycles per iteration)	STA (FES)	Iteration count
▶ Loop 18403	binary	26.71	RAM bound	8.49	0.09	82.88	0.80	6.40	0.21	73.20	0.34	9.76	0.31	8.72	0.12	25
▶ Loop 26027	binary	12.01	Balanced workload (back-end starvation)	1.01	1.01	150.69	0.15	155.04	0.11	153.48	0.19	148.98	0.10	137.44	0.13	96
▶ Loop 18424	binary	10.81	RAM bound	3.53	0.42	66.94	0.21	32.12	0.03	75.73	0.27	18.98	0.03	17.41	0.02	51
▶ Loop 18474	binary	4.84	RAM bound	4.18	0.37	78.98	0.40	32.24	0.02	88.04	0.46	18.90	0.07	17.25	0.01	51
▶ Loop 26026	binary	2.78	L1 bound	0.98	0.71	173.54	0.18	175.29	0.07	246.79	0.25	177.54	0.12	163.29	0.07	96
▶ Loop 26028	binary	2.64	Balanced workload (back-end starvation)	0.98	1.04	171.90	0.24	174.58	0.05	168.48	0.06	175.27	0.07	165.73	0.15	96
▶ Loop 8754	binary	1.57	Balanced workload (fast front-end supply)	5.37	0.77	47.72	0.03	4.35	0.01	5.65	0.06	9.09	0.01	4.84	0.01	1489
▶ Loop 12711	binary	1.41	L1 bound	1.00	0.97	9.86	0.16	10.68	0.14	10.99	0.25	9.88	0.15	10.38	0.13	384
▼ Loop 18501	binary	1.22	RAM bound	4.40	0.32	325.64	0.08	79.09	0.01	248.73	0.19	74.09	0.02	63.73	0.00	22
▼ Bucket 9		98.86	RAM bound	4.40	0.32	325.64	0.08	79.09	0.01	248.73	0.19	74.09	0.02	63.73	0.00	22
▼ Bucket 10		1.08	RAM bound	6.56	0.18	488.18	0.53	79.00	0.01	435.73	0.63	74.36	0.01	64.27	0.01	22
▶ Loop 12729	binary	1.12	Balanced workload (back-end starvation)	1.04	1.22	8.94	0.19	10.36	0.08	8.47	0.18	8.61	0.22	8.91	0.12	384
▶ Loop 12688	binary	1.09	RAM bound	2.28	0.93	31.92	0.24	33.17	0.06	35.67	0.92	14.00	0.14	11.00	0.07	24
▶ Loop 8912	binary	0.97	Balanced workload (fast front-end supply)	5.01	0.93	49.52	0.07	5.75	0.03	6.16	0.10	9.88	0.05	6.23	0.04	128
▶ Loop 26800	binary	0.85	Balanced workload (fast front-end supply)	0.83	1.16	119.00	0.14	122.50	0.38	106.00	0.21	143.25	0.13	107.00	0.24	8
▶ Loop 20240	binary	0.78	RAM bound	1.28	0.57	10.60	0.21	10.00	0.05	17.50	0.04	8.30	0.03	6.05	0.02	384
▶ Loop 8755	binary	0.47	Balanced workload (fast front-end supply)	5.02	1.17	52.47	0.13	4.92	0.04	4.20	0.02	9.46	0.00	4.16	0.01	372

MAQAO ONE View Loop Analysis Report

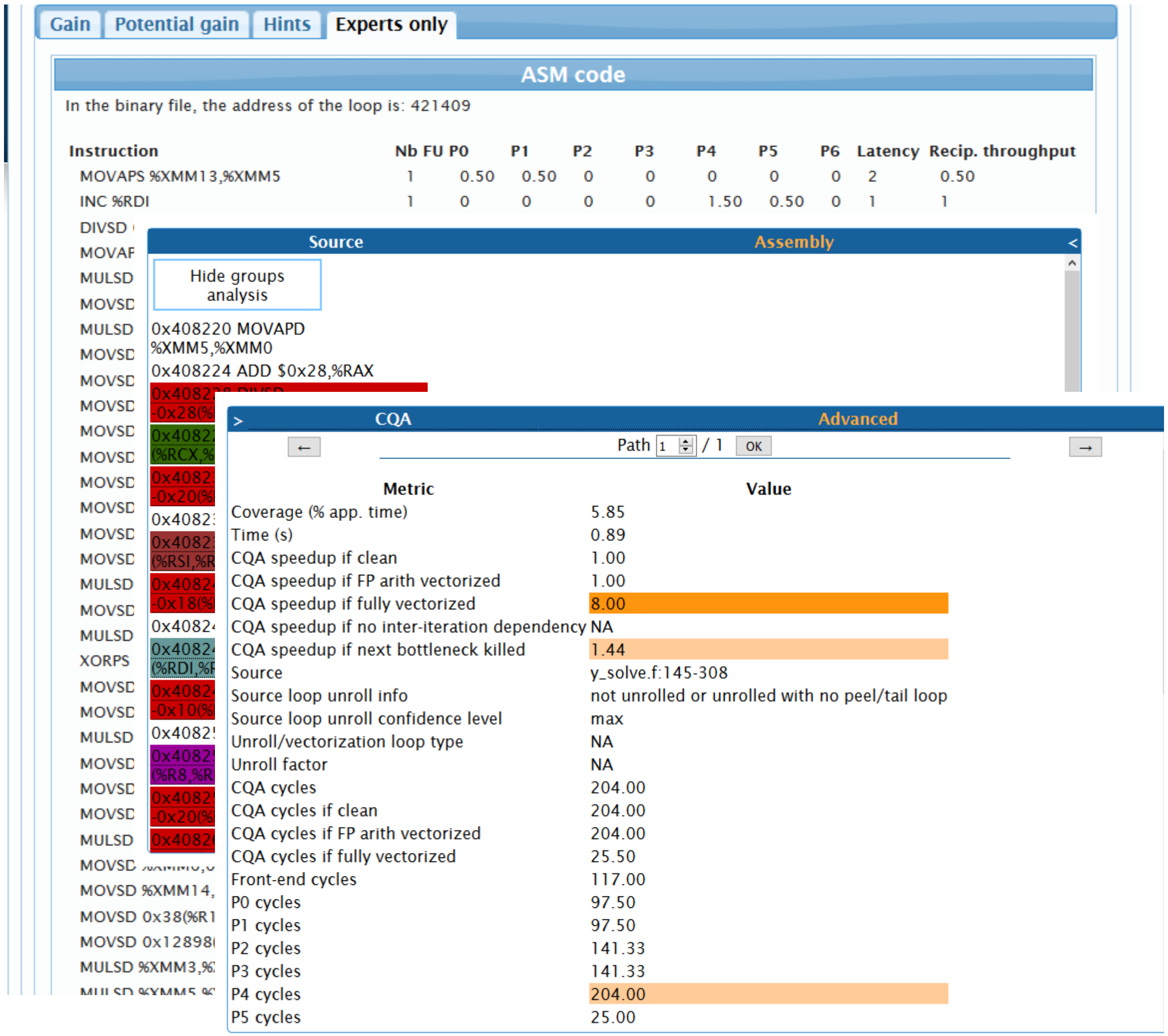
High level reports

- 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

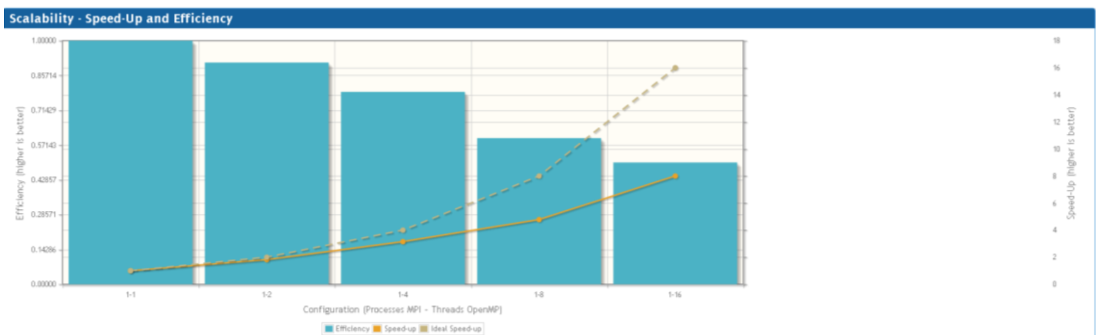
MAQAO Global Application Functions Loops Topology Help

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

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



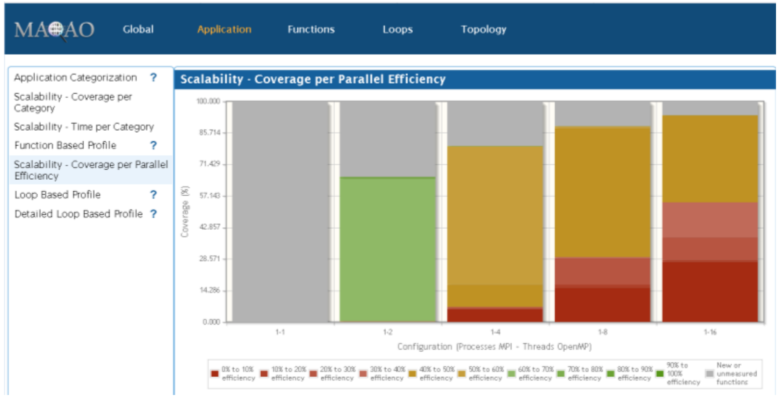
▼ **Detailed Speed-Up and Efficiency**

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

MAQAO ONE View Scalability Reports
Application View

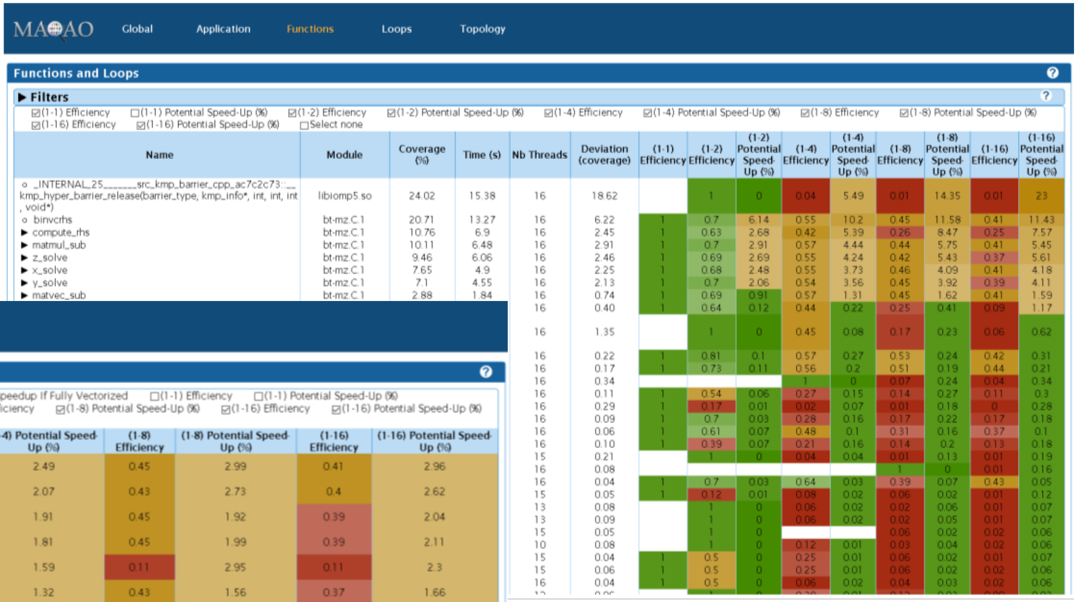
- Coverage per category
  - Comparison of categories for each run

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

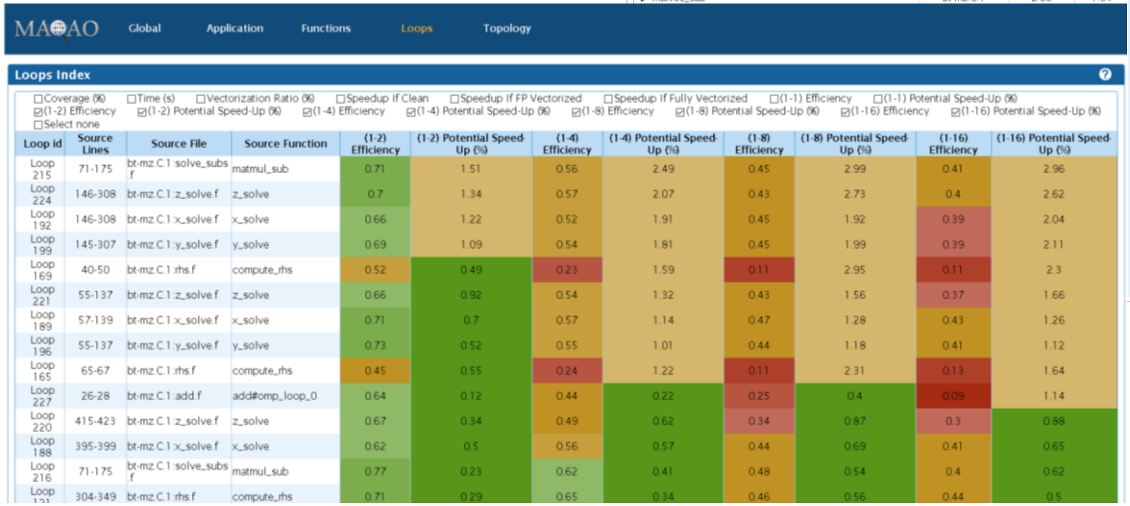


MAQAO ONE View Scalability Reports Functions and Loops Views

- Displays metrics for each function/loop
- Efficiency
- Potential speedup if efficiency=1



Name	Module	Coverage (%)	Time (s)	Nb Threads	Deviation (coverage)	(1-1) Efficiency	(1-2) Efficiency	(1-2) Potential Speed-Up (%)	(1-4) Efficiency	(1-4) Potential Speed-Up (%)	(1-8) Efficiency	(1-8) Potential Speed-Up (%)	(1-16) Efficiency	(1-16) Potential Speed-Up (%)
_INTERNAL_25_______src_kmp_barrier_cpp_ac7c2c73::__kmp_hyper_barrier_release(barrier_type, kmp_info*, int, int, int, void*)	libiomp5.so	24.02	15.38	16	18.62		1	0	0.04	5.49	0.01	14.35	0.01	23
binvcrhs	bt-mz.C.1	20.71	13.27	16	6.22	1	0.7	6.14	0.55	10.2	0.45	11.58	0.41	11.43
compute_rhs	bt-mz.C.1	10.76	6.9	16	2.45	1	0.63	2.68	0.42	5.39	0.26	8.47	0.25	7.57
matmul_sub	bt-mz.C.1	10.11	6.48	16	2.91	1	0.7	2.91	0.57	4.44	0.44	5.75	0.41	5.45
z_solve	bt-mz.C.1	9.46	6.06	16	2.46	1	0.69	2.69	0.55	4.24	0.42	5.43	0.37	5.61
x_solve	bt-mz.C.1	7.65	4.9	16	2.25	1	0.68	2.48	0.55	3.73	0.46	4.09	0.41	4.18
y_solve	bt-mz.C.1	7.1	4.55	16	2.13	1	0.7	2.06	0.54	3.56	0.45	3.92	0.39	4.11
matvec_sub	bt-mz.C.1	2.88	1.84	16	0.74	1	0.69	0.91	0.57	1.31	0.45	1.62	0.41	1.59



Loop id	Source Lines	Source File	Source Function	(1-2) Efficiency	(1-2) Potential Speed-Up (%)	(1-4) Efficiency	(1-4) Potential Speed-Up (%)	(1-8) Efficiency	(1-8) Potential Speed-Up (%)	(1-16) Efficiency	(1-16) Potential Speed-Up (%)
Loop 215	71-175	bt-mz.C.1:solve_subs.f	matmul_sub	0.71	1.51	0.56	2.49	0.45	2.99	0.41	2.96
Loop 224	146-308	bt-mz.C.1:z_solve.f	z_solve	0.7	1.34	0.57	2.07	0.43	2.73	0.4	2.62
Loop 192	146-308	bt-mz.C.1:x_solve.f	x_solve	0.66	1.22	0.52	1.91	0.45	1.92	0.39	2.04
Loop 199	145-307	bt-mz.C.1:y_solve.f	y_solve	0.69	1.09	0.54	1.81	0.45	1.99	0.39	2.11
Loop 169	40-50	bt-mz.C.1:rhs.f	compute_rhs	0.52	0.49	0.23	1.59	0.11	2.95	0.11	2.3
Loop 221	55-137	bt-mz.C.1:z_solve.f	z_solve	0.66	0.92	0.54	1.32	0.43	1.56	0.37	1.66
Loop 189	57-139	bt-mz.C.1:x_solve.f	x_solve	0.71	0.7	0.57	1.14	0.47	1.28	0.43	1.26
Loop 196	55-137	bt-mz.C.1:y_solve.f	y_solve	0.73	0.52	0.55	1.01	0.44	1.18	0.41	1.12
Loop 165	65-67	bt-mz.C.1:rhs.f	compute_rhs	0.45	0.55	0.24	1.22	0.11	2.31	0.13	1.64
Loop 227	26-28	bt-mz.C.1:add.f	add#omp_loop_0	0.64	0.12	0.44	0.22	0.25	0.4	0.09	1.14
Loop 220	415-423	bt-mz.C.1:z_solve.f	z_solve	0.67	0.34	0.49	0.62	0.34	0.87	0.3	0.88
Loop 188	395-399	bt-mz.C.1:x_solve.f	x_solve	0.62	0.5	0.56	0.57	0.44	0.69	0.41	0.65
Loop 216	71-175	bt-mz.C.1:solve_subs.f	matmul_sub	0.77	0.23	0.62	0.41	0.48	0.54	0.4	0.62
Loop 1[illegible]	304-349	bt-mz.C.1:rhs.f	compute_rhs	0.71	0.29	0.65	0.34	0.46	0.56	0.44	0.5

ISO BINARY: SCALABILITY RUNS (1)

MINIQMC: Weak Scalability Analysis

r0 : 1 core r1: 2 cores r2: 4 cores r3: 8 cores r4: 16 cores r5: 32 cores r6: 64 Cores

Global Metrics

Metric		r0	r1	r2	r3	r4	r5	r6
Total Time (s)		54.59	56.02	56.89	59.12	67.23	93.17	156.70
Profiled Time (s)		53.81	55.22	56.06	57.98	65.20	89.03	148.10
Time in analyzed loops (%)		51.7	50.7	50.1	49.5	47.5	48.8	46.2
Time in analyzed innermost loops (%)		51.6	50.6	50.0	49.4	47.4	48.7	46.1
Time in user code (%)		52.2	51.3	50.6	49.9	48.0	49.2	46.5
Compilation Options Score (%)		25.0	25.0	25.0	25.0	25.0	25.0	25.0
Perfect Flow Complexity		1.00	1.00	1.00	1.00	1.00	1.00	1.00
Array Access Efficiency (%)		Not Available	Not Available	Not Available	Not Available	Not Available	Not Available	Not Available
Perfect OpenMP + MPI + Pthread		1.00	1.00	1.00	1.00	1.00	1.01	1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00	1.01	1.01	1.01	1.01	1.01	1.01
No Scalar Integer	Potential Speedup	1.02	1.02	1.02	1.02	1.02	1.01	1.01
	Nb Loops to get 80%	4	4	4	4	4	4	4
FP Vectorised	Potential Speedup	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Nb Loops to get 80%	3	3	3	3	3	3	3
Fully Vectorised	Potential Speedup	1.02	1.02	1.02	1.02	1.02	1.02	1.01
	Nb Loops to get 80%	4	4	4	5	5	5	6
Only FP Arithmetic	Potential Speedup	1.16	1.15	1.15	1.15	1.13	1.12	1.10
	Nb Loops to get 80%	6	6	6	6	7	6	7
Scalability - Gap		1.00	1.03	1.04	1.08	1.23	1.71	2.87

ISO BINARY: SCALABILITY RUNS (2)

MINIQMC: Weak Scalability Analysis

r0 : 1 core r1: 2 cores r2: 4 cores r3: 8 cores r4: 16 cores r5: 32 cores r6: 64 Cores

▼ Colums Filter

- [ ] Coverage m1o1 (%)
- [ ] Coverage m1o2 (%)
- [ ] Coverage m1o4 (%)
- [ ] Coverage m1o8 (%)
- [ ] Coverage m1o16 (%)
- [ ] Coverage m1o32 (%)
- [ ] Coverage m1o64 (%)
- [x] Max Time Over Threads m1o1 (s)
- [x] Max Time Over Threads m1o2 (s)
- [x] Max Time Over Threads m1o4 (s)
- [x] Max Time Over Threads m1o8 (s)
- [x] Max Time Over Threads m1o16 (s)
- [x] Max Time Over Threads m1o32 (s)
- [x] Max Time Over Threads m1o64 (s)
- [ ] Time w.r.t. Wall Time m1o1 (s)
- [ ] Time w.r.t. Wall Time m1o2 (s)
- [ ] Time w.r.t. Wall Time m1o4 (s)
- [ ] Time w.r.t. Wall Time m1o8 (s)
- [ ] Time w.r.t. Wall Time m1o16 (s)
- [ ] Time w.r.t. Wall Time m1o32 (s)
- [ ] Time w.r.t. Wall Time m1o64 (s)
- [ ] Nb Threads m1o1
- [ ] Nb Threads m1o2
- [ ] Nb Threads m1o4
- [ ] Nb Threads m1o8
- [ ] Nb Threads m1o16
- [ ] Nb Threads m1o32
- [ ] Nb Threads m1o64
- [ ] Deviation (coverage) m1o1
- [ ] Deviation (coverage) m1o2
- [ ] Deviation (coverage) m1o4
- [ ] Deviation (coverage) m1o8
- [ ] Deviation (coverage) m1o16
- [ ] Deviation (coverage) m1o32
- [ ] Deviation (coverage) m1o64
- [ ] Categories m1o1
- [ ] Categories m1o2
- [ ] Categories m1o4
- [ ] Categories m1o8
- [ ] Categories m1o16
- [ ] Categories m1o32
- [ ] Categories m1o64
- [ ] Compilation Options
- [ ] (m1o1) Efficiency
- [ ] (m1o1) Potential Speed-Up (%)
- [ ] (m1o2) Efficiency
- [ ] (m1o2) Potential Speed-Up (%)
- [ ] (m1o4) Efficiency
- [ ] (m1o4) Potential Speed-Up (%)
- [ ] (m1o8) Efficiency

Name	Module	Max Time Over Threads m1o1 (s)	Max Time Over Threads m1o2 (s)	Max Time Over Threads m1o4 (s)	Max Time Over Threads m1o8 (s)	Max Time Over Threads m1o16 (s)	Max Time Over Threads m1o32 (s)	Max Time Over Threads m1o64 (s)
○ dgemm_sve_big	libarmpl.so	21.99	22.86	23.6	24.73	29.12	37.66	60.43
▶ void qmcplusplus::DTD_BConds<double, 3u, 39>::computeDistances<qmcplusplus::TinyVector<double, 3u>, qmcplusplus::VectorSoAContainer<double, 3u, 32ul, qmcplusplus::Mallocator<double, 32ul> >, qmcplusplus::VectorSoAContainer<dou...	miniqmc	10.95	11.01	11.15	11.28	11.39	11.6	12.27
▶ void miniqmcreference::MultiBsplineEvalRef::evaluate_v<double>(qmcplusplus::bspline_traits<double, 3u>::SplineType const*, double, double, double, double*, unsigned long)	miniqmc	6.83	6.87	7.01	7.31	8.78	18.58	35.49
▶ void miniqmcreference::MultiBsplineEvalRef::evaluate_vgh<double>(qmcplusplus::bspline_traits<double, 3u>::SplineType const*, double, double, double, double*, double*, double*, unsigned long)	miniqmc	5.44	5.69	5.73	5.98	6.88	8.89	13.9
○ interleave_2vl_sve_kernel_dc	libarmpl.so	1.71	1.92	1.9	2.01	2.21	4.42	11.45
▶ void qmcplusplus::DTD_BConds<double, 3u, 39>::computeDistances<qmcplusplus::TinyVector<double, 3u>, qmcplusplus::VectorSoAContainer<double, 3u, 32ul, q	miniqmc	0.93	0.92	1.05	1.05	1.04	1	1.16

ISO SOURCE: COMPILER COMPARISON

MINIQMC: ARM Clang versus ARM Clang + ARM PL

▼ Compared Reports

- r0: miniqmc_ov1_armclang_o1m1
- r1: miniqmc_ov1_armclang_o1m1_pl

Global Metrics

Metric		r0	r1
Total Time (s)		231.75	53.89
Profiled Time (s)		231.03	53.16
Time in analyzed loops (%)		11.8	51.9
Time in analyzed innermost loops (%)		11.8	51.7
Time in user code (%)		12.0	52.3
Compilation Options Score (%)		25.0	25.0
Perfect Flow Complexity		1.00	1.00
Array Access Efficiency (%)		Not Available	Not Available
Perfect OpenMP + MPI + Pthread		1.00	1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00	1.00
No Scalar Integer	Potential Speedup	1.00	1.02
	Nb Loops to get 80%	4	6
FP Vectorised	Potential Speedup	1.00	1.00
	Nb Loops to get 80%	3	3
Fully Vectorised	Potential Speedup	1.00	1.02
	Nb Loops to get 80%	5	7
Only FP Arithmetic	Potential Speedup	1.03	1.16
	Nb Loops to get 80%	6	7

ISO FUNCTION STRUCTURE

Global Metrics

Metric		r0	r1	r2	r3	r4
Total Time (s)		29.12	22.53	21.32	19.63	21.80
Profiled Time (s)		28.78	22.18	20.92	19.25	21.49
Time in analyzed loops (%)		87.3	81.1	79.7	79.8	78.7
Time in analyzed innermost loops (%)		37.8	47.1	43.8	51.4	51.7
Time in user code (%)		94.6	90.7	90.0	88.8	88.5
Compilation Options		OK	OK	OK	OK	OK
Perfect Flow Complexity		1.00	1.05	1.00	1.00	1.00
Iterations Count		1.04	1.02	1.03	1.03	1.02
Array Access Efficiency (%)		79.6	81.9	71.8	70.3	71.3
Perfect OpenMP + MPI + Pthread		1.00	1.00	1.00	1.00	1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00	1.00	1.00	1.00	1.00
No Scalar Integer	Potential Speedup	1.23	1.20	1.19	1.17	1.16
	Nb Loops to get 80%	12	13	10	12	11
FP Vectorised	Potential Speedup	1.18	1.27	1.27	1.29	1.27
	Nb Loops to get 80%	14	14	14	17	18
Fully Vectorised	Potential Speedup	3.69	2.86	2.73	2.63	2.58
	Nb Loops to get 80%	41	41	41	41	41
Only FP Arithmetic	Potential Speedup	2.01	1.65	1.65	1.59	1.69
	Nb Loops to get 80%	26	29	28	35	37
Data In L1 Cache	Potential Speedup	1.05	1.06	1.07	1.10	1.08
	Nb Loops to get 80%	5	4	5	6	6

5 successive code versions of CHAMP

Unicore runs SKL

Regular gains except for the last one!!

CHAMP Unicore on SKL

Functions

Name	Module	Time (s) champ_01apr_ov3_energy_15k	champ_26apr_ov3_energy_15k	champ_27apr_ov3_energy_15k	champ_29apr_ov3_energy_15k	champ_11may_ov3_energy_15k
multideterminante	vmc.mov1	7.37	4.6	4.07	3.45	3.55
basis_fns	vmc.mov1	2.19	1.85	2.09	1.96	1.93
compute_ymat	vmc.mov1	6.01	0.13	0.13	0.08	3.6
orbitals	vmc.mov1	1.49	1.56	1.47	1.43	1.48
nonloc	vmc.mov1	1.37	1.28	1.38	1.32	1.44
multideterminante_grad	vmc.mov1	1.09	1.09	1.11	1.11	1.19
multideterminant_hpsi	vmc.mov1	1.29	0.9	0.76	0.7	0.82
orbitalse	vmc.mov1	0.79	0.87	0.8	0.81	0.86
matinv	vmc.mov1	0.85	0.94	0.93	0.56	0.7
__powr8i4	vmc.mov1	0.62	0.76	0.71	0.68	0.7
idiff	vmc.mov1	0.65	0.65	0.7	0.66	0.66
splfit	vmc.mov1	0.56	0.55	0.51	0.58	0.61
detsav	vmc.mov1	1.31	0.56	0.5	0.2	0.21
__intel_avx_rep_memset	vmc.mov1	0.12	0.57	0.47	0.53	0.5
__intel_avx_rep_memcpy	vmc.mov1	0.25	0.14	0.42	0.46	0.82
determinante_psit	vmc.mov1	0.49	0.3	0.36	0.32	0.55
update_ymat	vmc.mov1	0.54	0.3	0.24	0.23	0.31
__libm_log_l9	vmc.mov1	0.24	0.31	0.25	0.23	0.23
psinl	vmc.mov1	0.13	0.16	0.14	0.14	0.17
slm	vmc.mov1	0.15	0.16	0.15	0.12	0.13
multideterminants_define	vmc.mov1	0.11	0.13	0.07	0.11	0.12
__libm_exp_l9	vmc.mov1	0.13	0.1	0.09	0.11	0.09
jastrow4e	vmc.mov1	0.14	0.06	0.07	0.05	0.07
compute_determinante_grad	vmc.mov1	0.07	0.04	0.07	0.05	0.07

ISO SOURCE: DIFFERENT HARDWARE

All runs were unicore and used the same compiler GNU 11
Code MAHYCO (Arcane framework)

r0: SKL   r1: ZEN_2   r2: ZEN_3

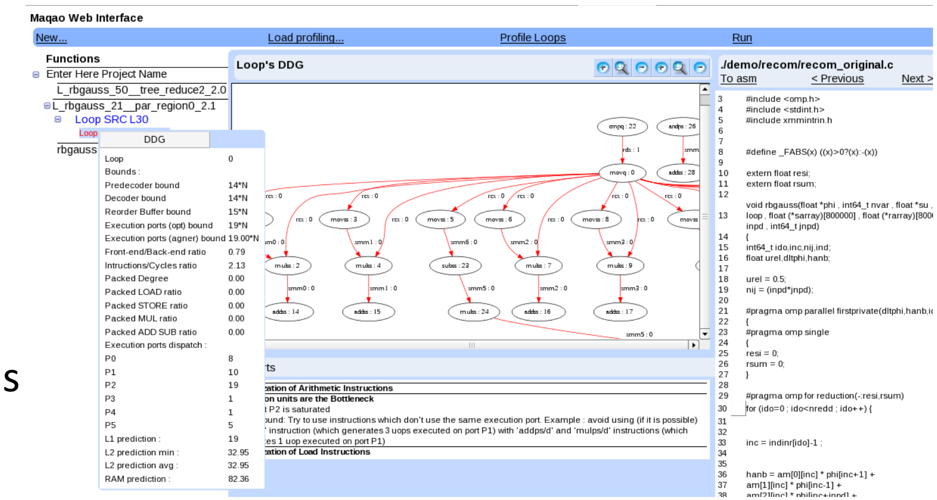
Global Metrics

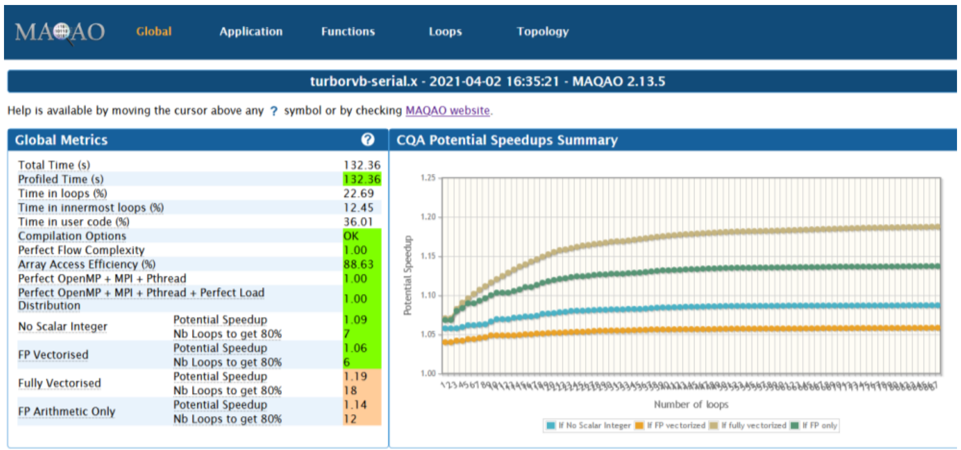
Metric		r0	r1	r2
Total Time (s)		916.81	738.02	592.37
Profiled Time (s)		915.78	734.50	590.03
Time in analyzed loops (%)		72.8	69.3	68.2
Time in analyzed innermost loops (%)		41.7	40.1	41.4
Time in user code (%)		87.7	86.6	85.9
Compilation Options		OK	OK	OK
Perfect Flow Complexity		1.36	1.26	1.28
Array Access Efficiency (%)		64.0	62.1	61.7
Perfect OpenMP + MPI + Pthread		1.00	1.00	1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00	1.00	1.00
No Scalar Integer	Potential Speedup	1.26	1.17	1.16
	Nb Loops to get 80%	5	5	5
FP Vectorised	Potential Speedup	1.41	1.31	1.29
	Nb Loops to get 80%	5	7	7
Fully Vectorised	Potential Speedup	2.26	1.91	1.88
	Nb Loops to get 80%	16	14	14
Only FP Arithmetic	Potential Speedup	1.46	1.42	1.33
	Nb Loops to get 80%	7	6	6

BACKUP SLIDES

MAQAO History

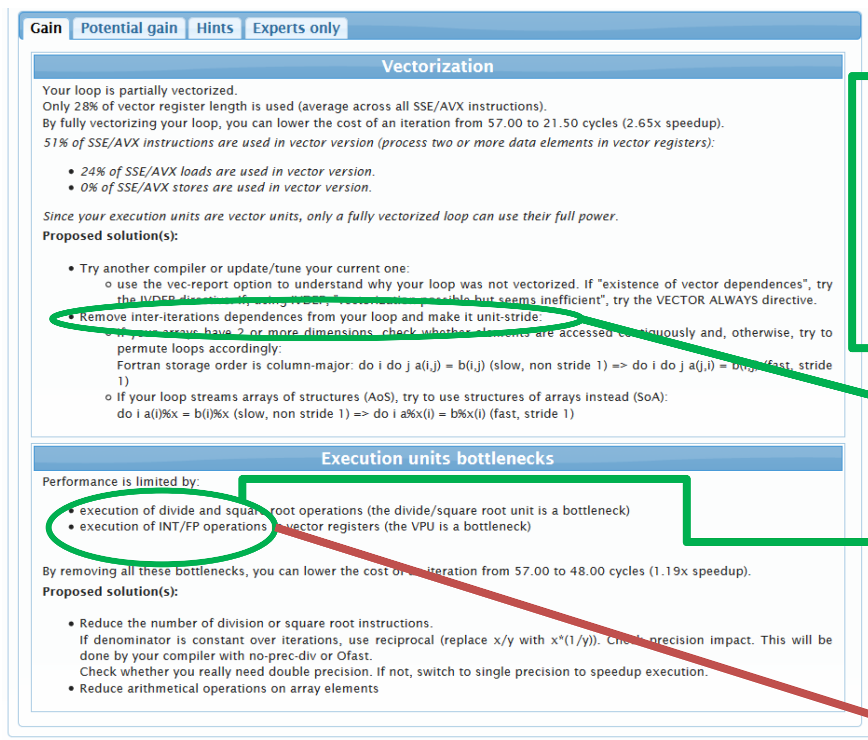
- 2004: Begun development
  - Focusing on Intel Itanium architecture
  - Analysis of assembly files
- 2006: Transition to Intel x86-64
- 2009: Binary analysis support
  - First version of decremental analysis
- 2012: Support of KNC architecture
- 2014: Profiling features
- 2015: First version of ONE View
- 2017: Prototype support of ARM architecture
- 2018: Scalability mode
- 2020: Comparison mode
- 2022: Support of ARM (beta)
- 2023: High-level summary
- 2024: Ongoing developments
  - GPU profiler prototype
  - ARM decremental analysis & value profiling

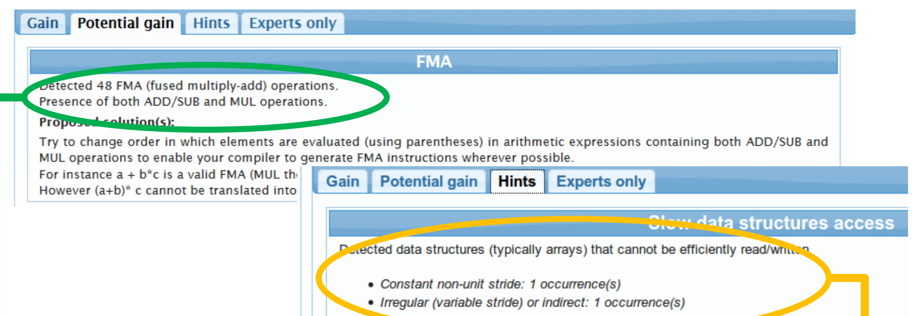




MAQAO CQA: Code Quality Analyzer

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**