The Long and Winding Road Toward Efficient High-Performance Computing

C. Valensi, E. Oseret, M. Tribalat, S. Ibnamar, K. Camus, A. Delval, C. Astier, G. Dos Santos, M. Hoffer, F.-X. Mordant, F. Santoro + M. Popov

W. Jalby

UVSQ/UPSaclay

Overview

- Well known: performance evolution of very Big Systems

- Less known: A close look at 2007 – 2016: performance evolution smaller machines running real applications

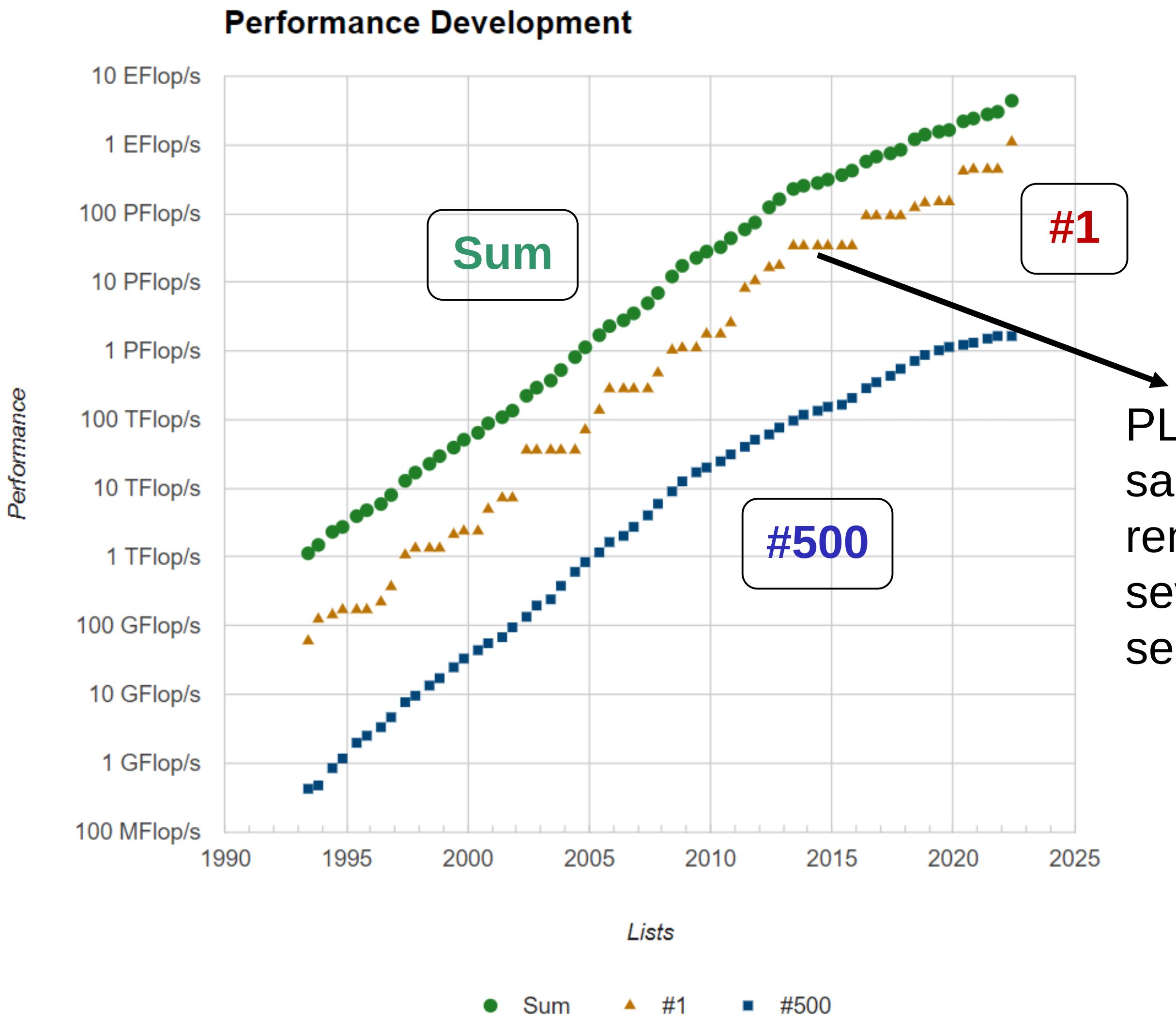
- A few hopes : how to exploit more effiently new systems

TOP 500

www.top500.org: The TOP500 project ranks and details the 500 most powerful “official” computer systems in the world. The project was started in 1993 and publishes an updated list of the supercomputers twice a year (Wikipedia).

“POWERFUL”: performance is measured on solving a dense linear system (N x N) using Linpack package

EVOLUTION OVER THE YEARS

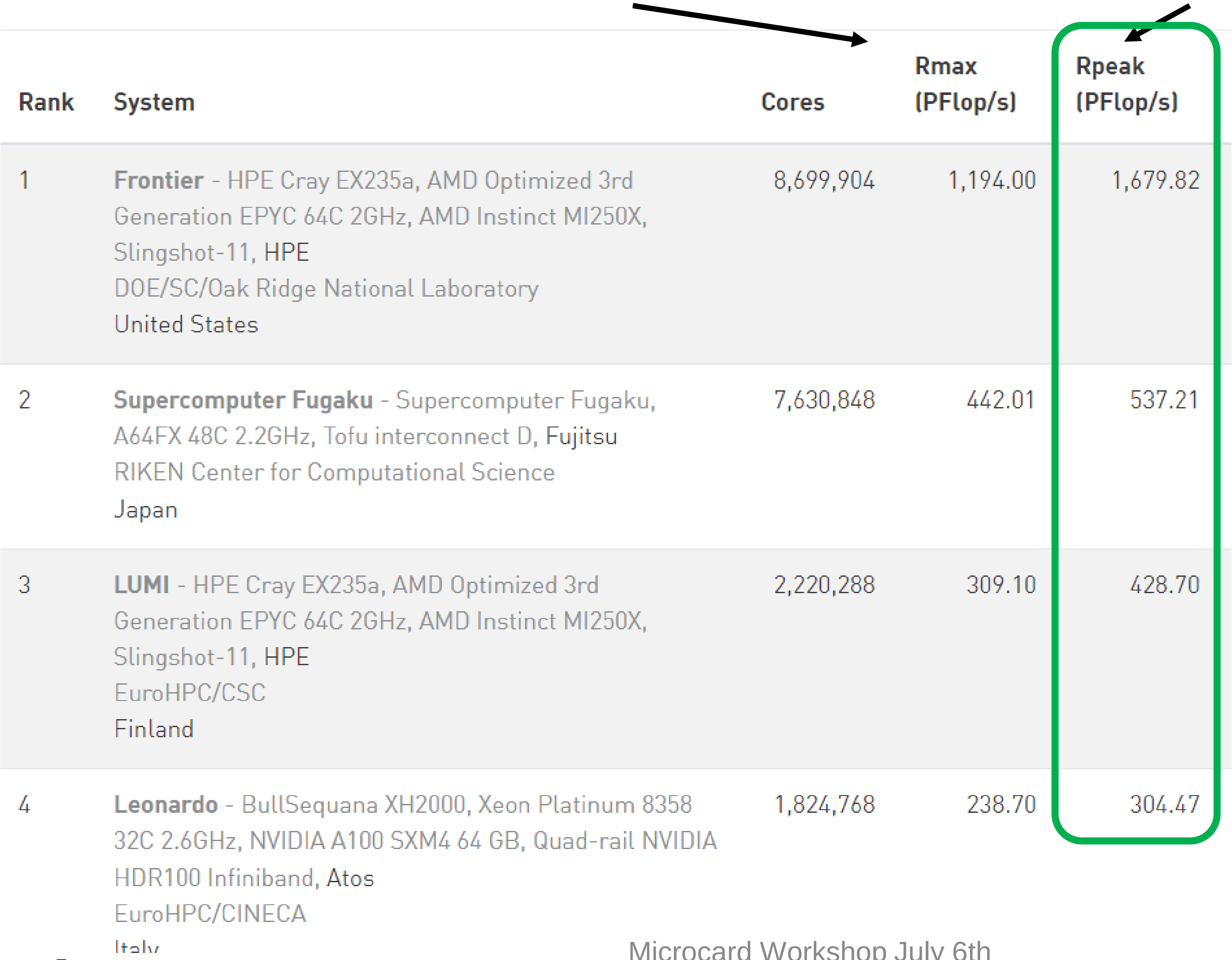


PLATEAU: the same system remains #1 for several semesters

LINPACK JUNE 2023 RESULTS

Rmax: best performance measured on Linpack

Rpeak: Peak (nominal) performance



Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,824,768	238.70	304.47

Rmax/Rpeak less than 0,8 !!

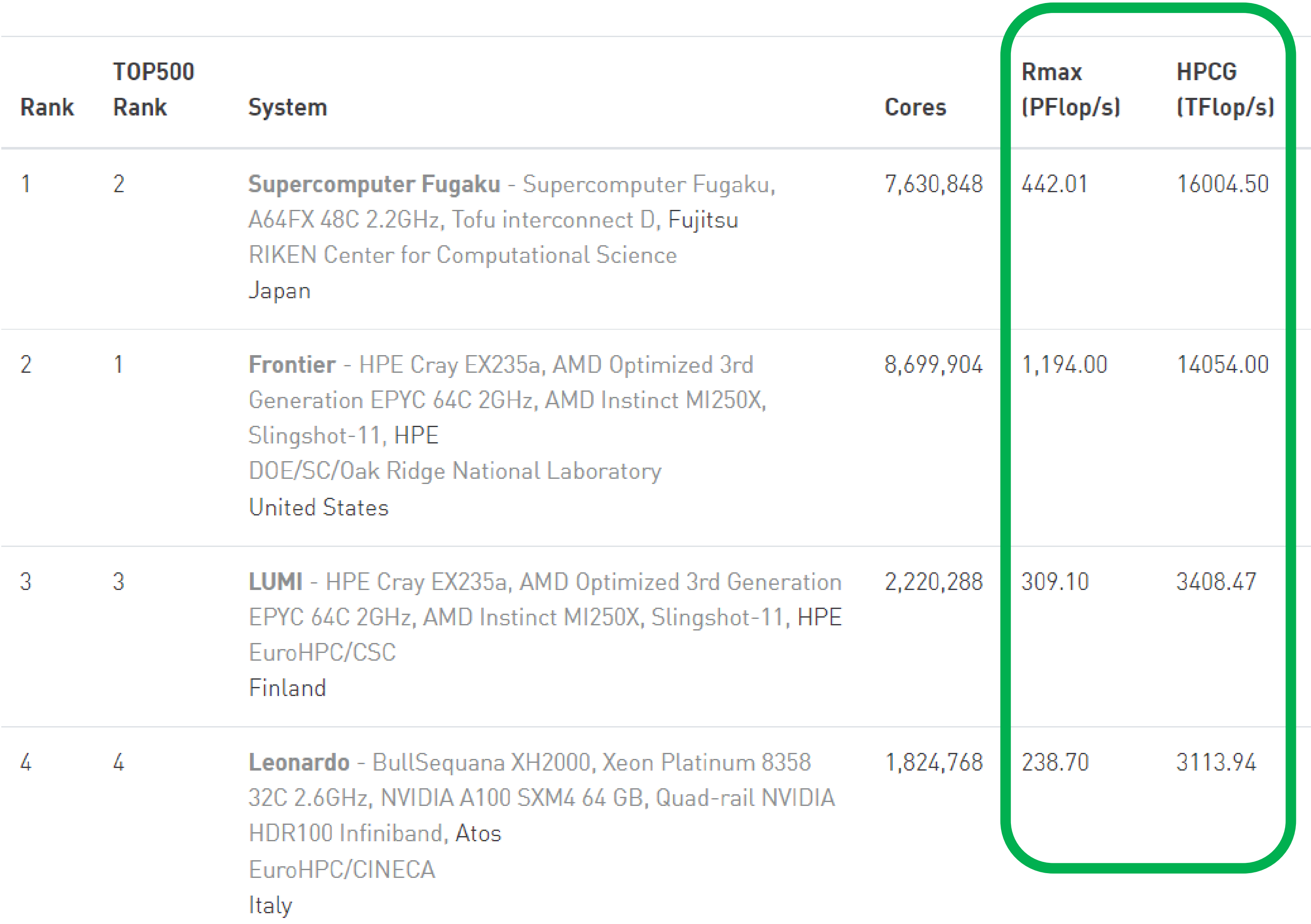
TOP 500 REVISITED

BE MORE REALISTIC: Instead of solving a dense linear system, let us solve a sparse linear system (much more frequent problem)

HPCG: High Performance Conjugate Gradient

HPL: High Performance Linpack

HPCG JUNE 2023 RESULTS



Rank	TOP500 Rank	System	Cores	Rmax (PFlop/s)	HPCG (TFlop/s)
1	2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	16004.50
2	1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	14054.00
3	3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	3408.47
4	4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,824,768	238.70	3113.94

Rmax/HPCG varies between 27 and 80

Conclusion on big systems

- Essentially show off machines….

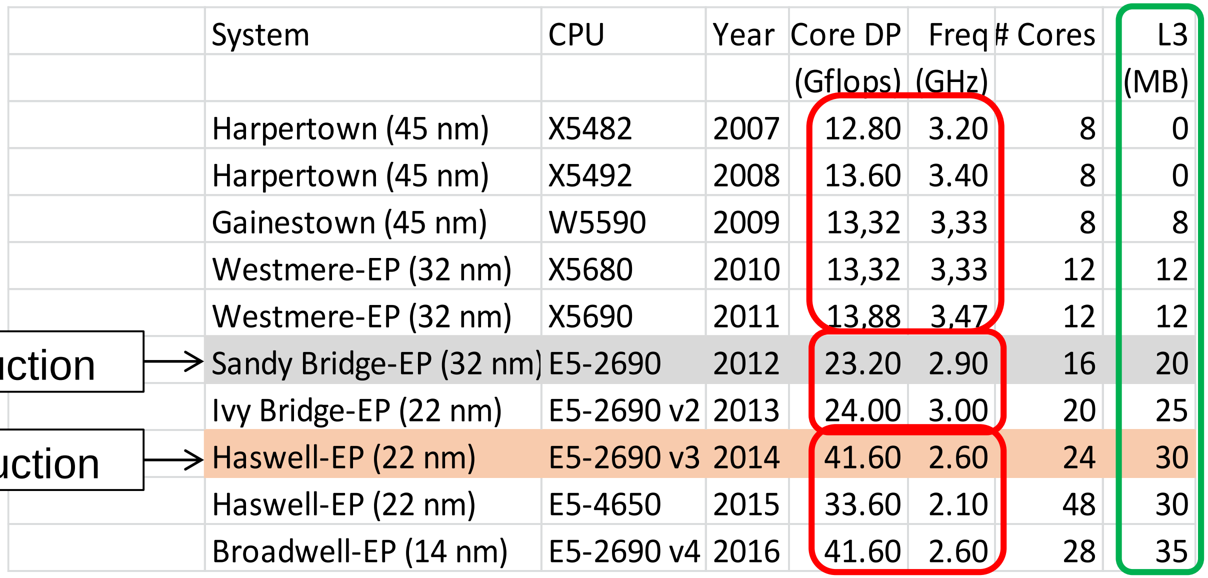
- Almost never used as a single system on a single task: much too expensive.

- Large spread between HPL and HPCG
  - HPL performance around 75% of Rpeak
  - HPCG performance less than 2% of Rpeak

- HPL and HPCG are not real applications: they are “toy” programs

- Too much focus on FP performance: in fact data access performance is as important as FP performance

A CLOSE LOOK AT A DECADE (2007 - 2016)



	System	CPU	Year	Core DP	Freq	# Cores	L3
				(Gflops)	(GHz)		(MB)
	Harpertown (45 nm)	X5482	2007	12.80	3.20	8	0
	Harpertown (45 nm)	X5492	2008	13.60	3.40	8	0
	Gainestown (45 nm)	W5590	2009	13,32	3,33	8	8
	Westmere-EP (32 nm)	X5680	2010	13,32	3,33	12	12
	Westmere-EP (32 nm)	X5690	2011	13,88	3,47	12	12
	Sandy Bridge-EP (32 nm)	E5-2690	2012	23.20	2.90	16	20
	Ivy Bridge-EP (22 nm)	E5-2690 v2	2013	24.00	3.00	20	25
	Haswell-EP (22 nm)	E5-2690 v3	2014	41.60	2.60	24	30
	Haswell-EP (22 nm)	E5-4650	2015	33.60	2.10	48	30
	Broadwell-EP (14 nm)	E5-2690 v4	2016	41.60	2.60	28	35

AVX introduction → Sandy Bridge-EP (32 nm)

FMA Introduction → Haswell-EP (22 nm)

- 10 INTEL reference architectures from 2007 to 2016
- AVX then FMA (Fused Multiply Add) introductions

SPEC

The **Standard Performance Evaluation Corporation (SPEC)** is an American non-profit organization that aims to "produce, establish, maintain and endorse a standardized set" of performance benchmarks for computers (Wikipedia)

SPEC was founded in 1988. SPEC benchmarks are widely used to evaluate the performance of computer systems; the test results are published on the SPEC website (www.spec.org).

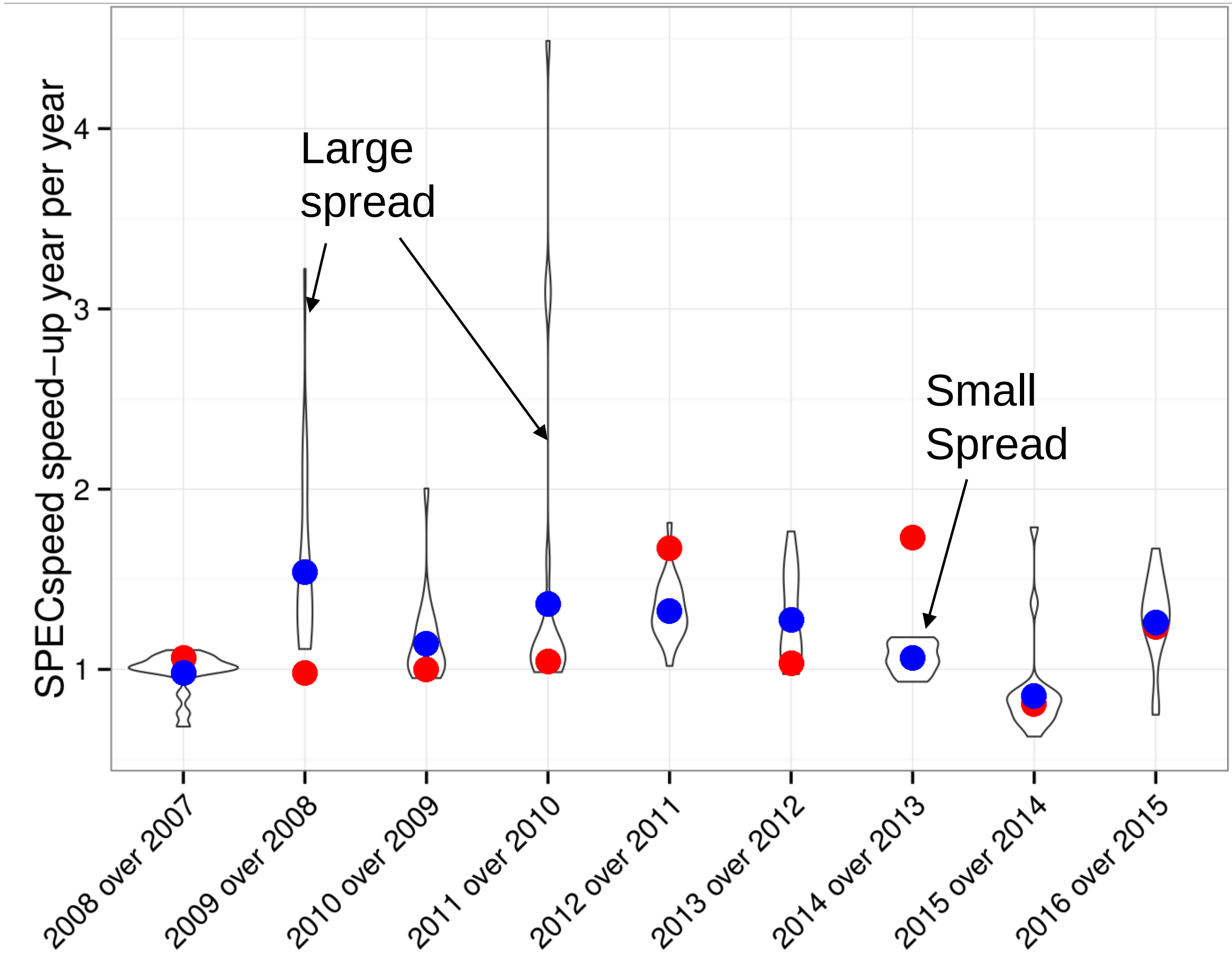
Members are hardware/software companies, a few Universities and research center

GOAL: provide an unbiased way of comparing systems performance using a precise protocol and several reference programs representative of “real” workload. Public database available

REPRESENTATIVE FOR A USER: OK if his favorite program is included in the reference list, otherwise representativeness can be argued for ever.

Performance Progress from 2007 to 2016 (1)

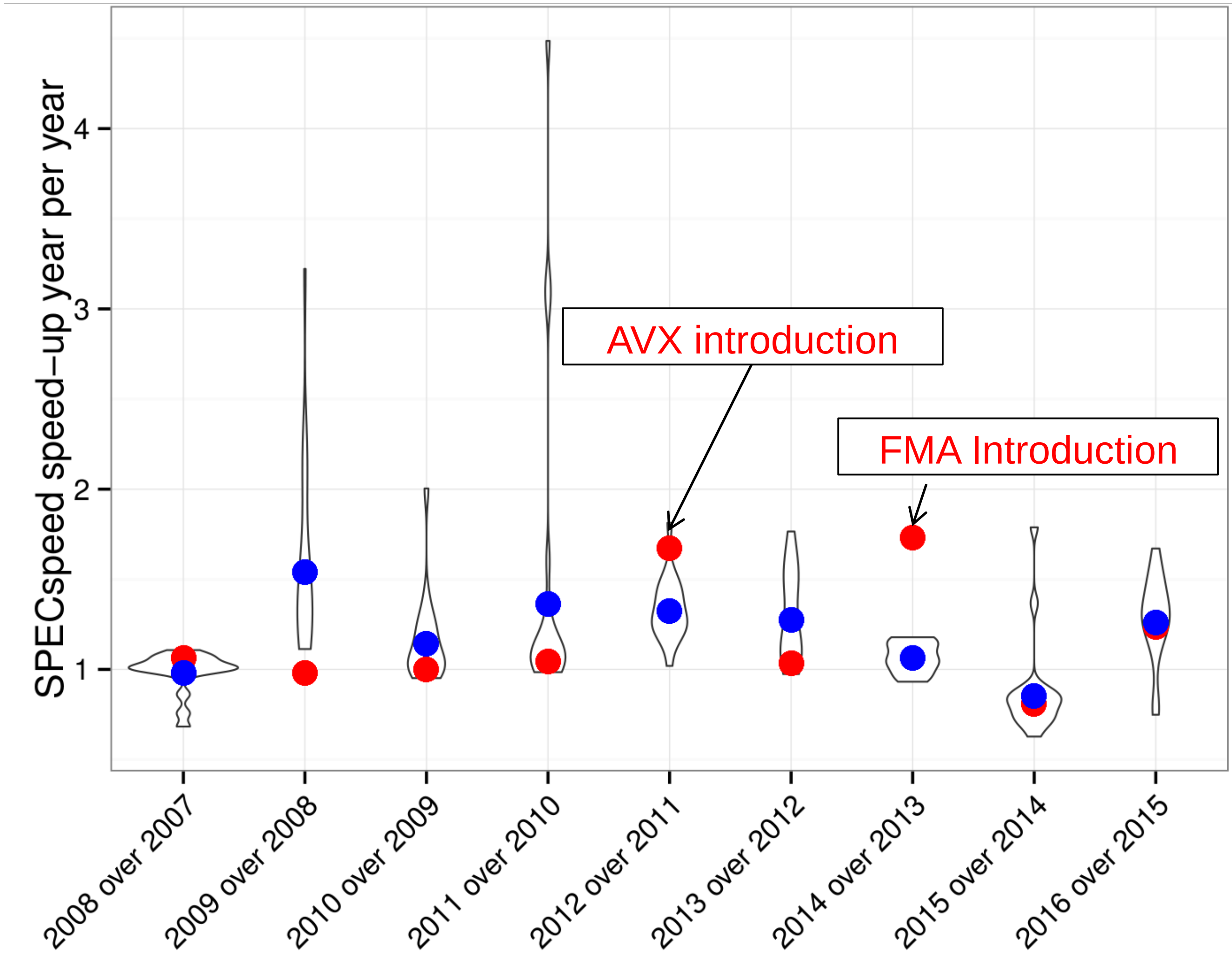
- Speedup from one year to the next one
- SPEC FP 2006 : Full set of “real” scientific applications
- Violin: distribution for the full set of SPECFP
- Architecture/compilers vary from year to year BUT Source code are invariant
- Unicore measurements
- Baseline compiler options but standard – O3 options



WJ_2023

Performance Progress from 2007 to 2016 (2)

- Speedup from one year to the next
- Red Dots: speedup of peak FP performance: most of the time, speedup around one.
- Blue Dots: geometric mean of Speedups
- Unicore measurements
- Baseline



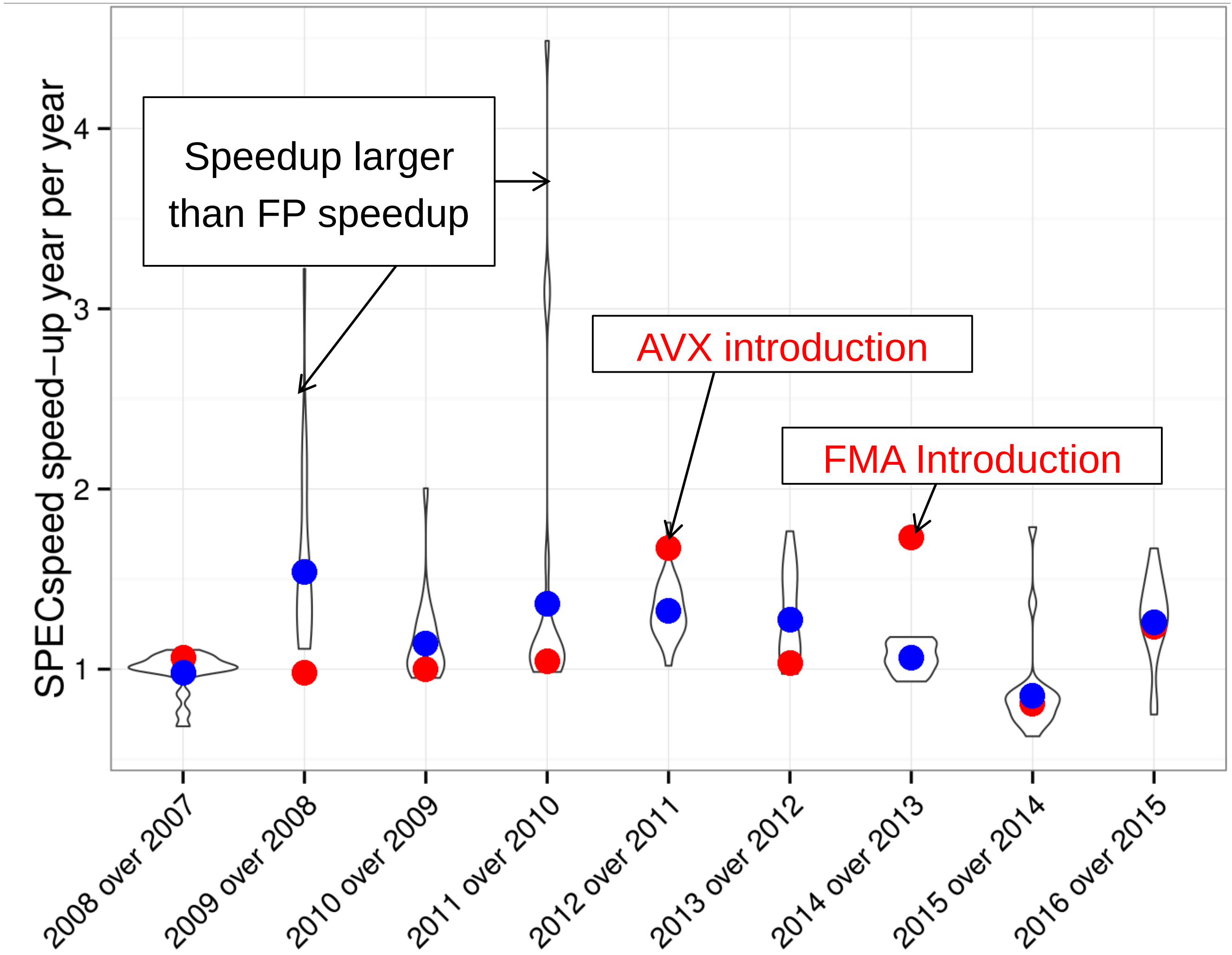
AVX introduction

FMA Introduction

WJ_2023

Performance Progress from 2007 to 2016

- Blue dots and red dots don’t move together
- Peak FP is not the single major factor, memory organization plays a key tole too
- Red Dots: speedup of peak FP performance
- Blue Dots: geometric mean of Speedups
- WARNING: Unicore measurements: only one core accessing the whole memory system



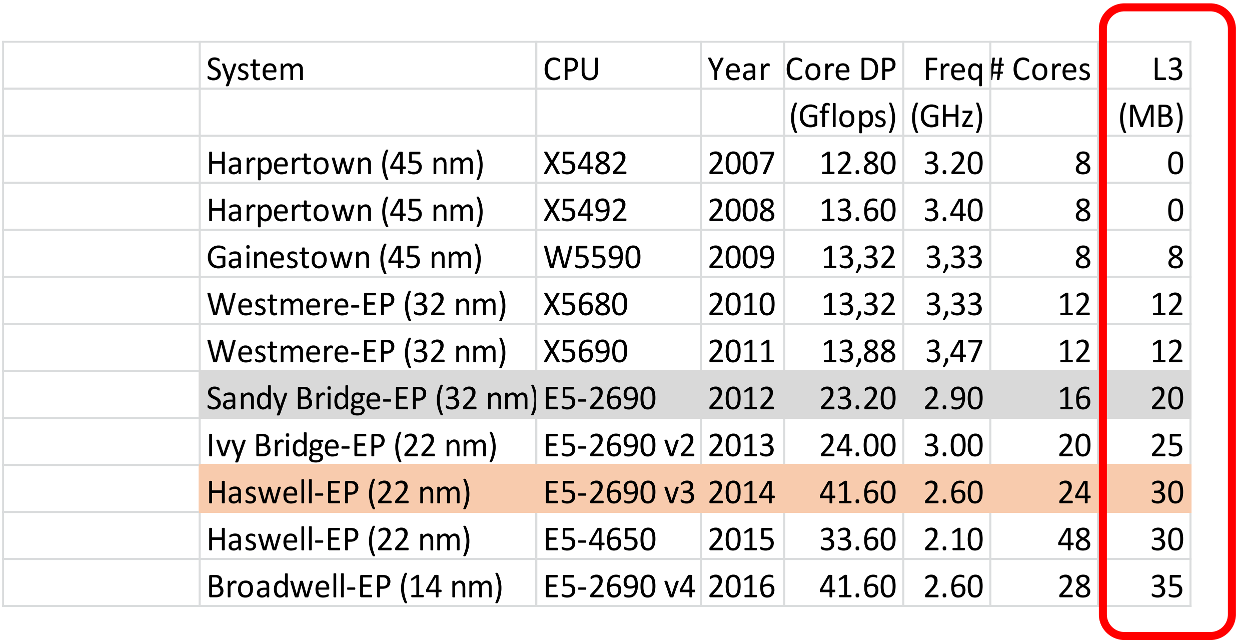
Speedup larger than FP speedup

AVX introduction

FMA Introduction

Unicore versus throughput

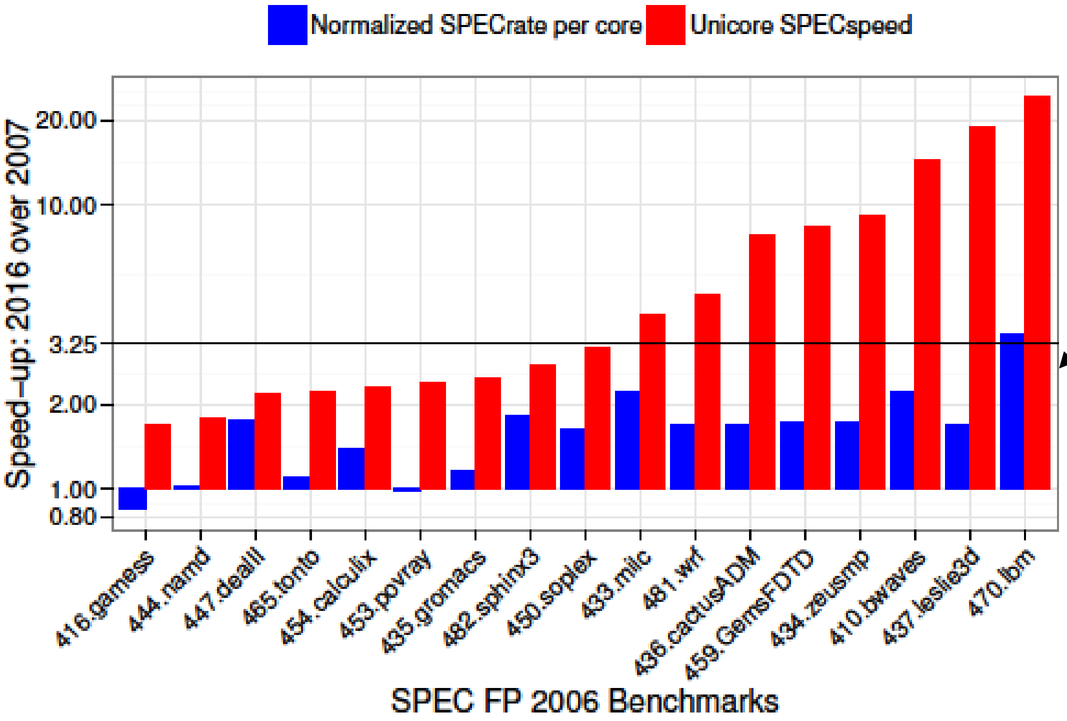
- ➢ Unicore measurements: only one core accessing the whole memory system: unrealistic, in real systems all of the cores will be active and share the memory system
- ➢ Launch multiple copies of the same program, one per core: each core will access a fair share of the system and therefore performance measured will be more realistic: SPECrate measurements.



	System	CPU	Year	Core DP	Freq	# Cores	L3
				(Gflops)	(GHz)		(MB)
	Harpertown (45 nm)	X5482	2007	12.80	3.20	8	0
	Harpertown (45 nm)	X5492	2008	13.60	3.40	8	0
	Gainestown (45 nm)	W5590	2009	13,32	3,33	8	8
	Westmere-EP (32 nm)	X5680	2010	13,32	3,33	12	12
	Westmere-EP (32 nm)	X5690	2011	13,88	3,47	12	12
	Sandy Bridge-EP (32 nm)	E5-2690	2012	23.20	2.90	16	20
	Ivy Bridge-EP (22 nm)	E5-2690 v2	2013	24.00	3.00	20	25
	Haswell-EP (22 nm)	E5-2690 v3	2014	41.60	2.60	24	30
	Haswell-EP (22 nm)	E5-4650	2015	33.60	2.10	48	30
	Broadwell-EP (14 nm)	E5-2690 v4	2016	41.60	2.60	28	35

A more realistic view on performance progress (1)

Ratio of 2016 performance over 2007

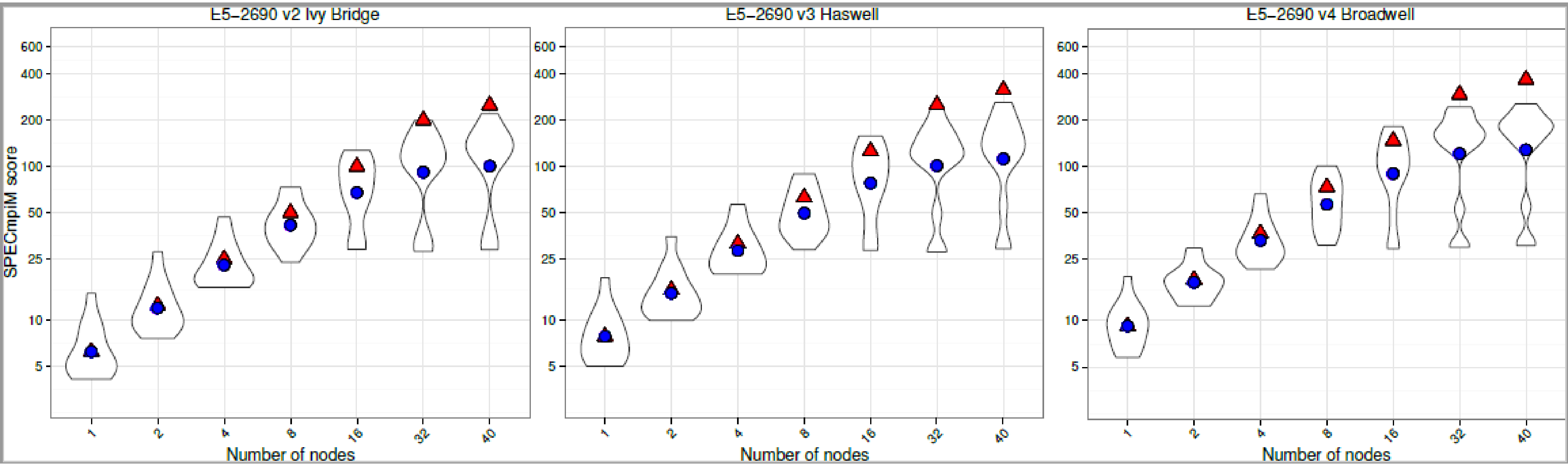


Peak FP 2016/Peak FP 2007

- Baseline numbers (no specific hand tuning but standard –O3)
- Performance gains highly dependant upon applications
- Red bars much higher than blue bars
- Blue bars always under the ratio of peak FP…..

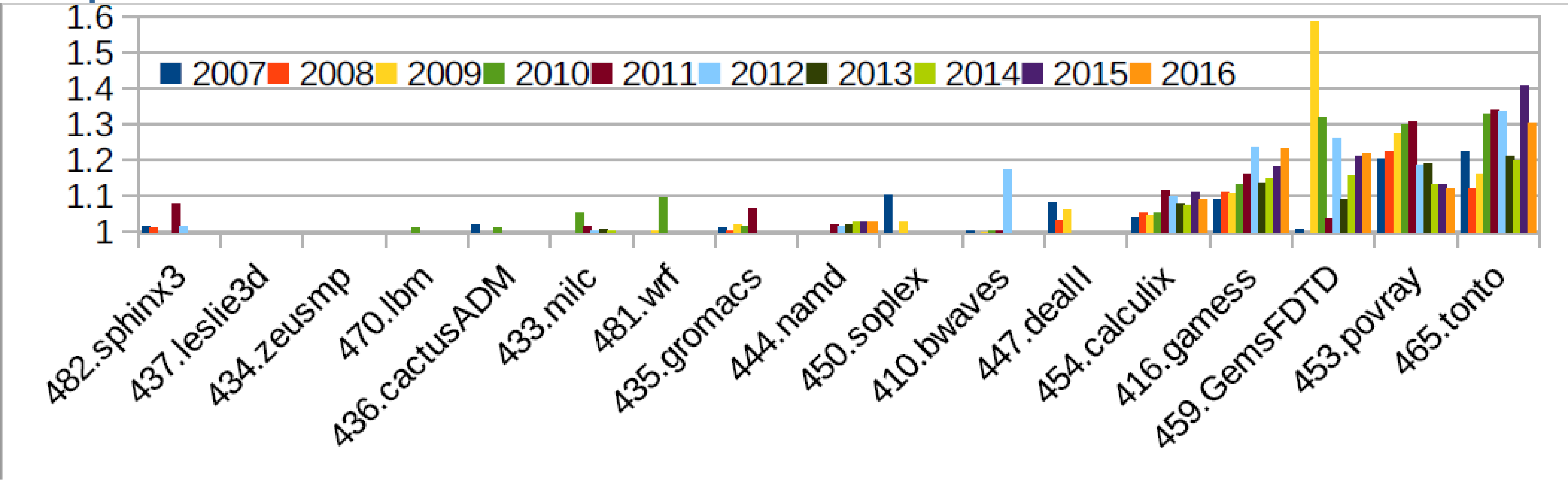
SPEC MPI performance progress over 4 years

A few multinode numbers: red triangles perfect node speedup, blue dots SPEC_MPI geom mean



CPU	cores per node	peak (TFlops)	SPEC result	result / peak
E5-2690 v2	20	19,2	100,26	5,22
E5-2690 v3	24	39,94	111,61	2,79
E5-2690 v4	28	46,59	128,94	2,77

Peak versus baseline



- Baseline: standard flags (typical –O3)
- Peak: hand picked flags
- Y axis: speedup of peak versus baseline
- X axis: sorted first by SPEC FP 2006 codes and second by year (same reference architectures)
- Profile Guided Optimization (PGO) is the most profitable (rightmost part of the digram)

GORDON BELL AWARD 2007 - 2016

Year	Machine	Cores	Tflops obtained	Tflops peak	Obtained over Peak (%)	Computations
2007	BlueGene/L	131K	0,11	0,28	39	Micron-Scale Atomistic Simulation of Kelvin-Helmholtz Instability
2008	Cray XT4	31K	0,2	0,26	77	Simulations of disorder effects in high-Tc superconductors
2009	Cray XT5	147K	1,03	1,36	76	Ab initio computation of free energies
2010	Cray XT5-HE	200K	0,7	2,3	30	Direct numerical simulation of blood flow
2011	K computer	442K	3,08	7,07	44	First-principles calculations of electron states of a silicon nanowire
2012	K computer	663K	4,45	10,6	42	Astrophysical N -Body Simulation
2013	Sequoia	1.6M	11	20,1	55	Cloud Cavitation Collapse
2014	Anton 2	NA	NA	NA	NA	Molecular dynamics
2015	Sequoia	1.6M	0,69	20,1	3	Implicit Solver for Complex PDEs
2016	Sunway TaihuLight	10.56M	7,85	125	6	Fully-Implicit Solver for Nonhydrostatic Atmospheric Dynamics

- Very impressive results: high efficiency except for the last 2 years
- Very different results from SPEC evolution
- Codes have been fully customized: using (??) top of the line performance evaluation tools ☺
- Are these codes more than Proof of Concept ?? Impact on standard apps ??

A FEW LESSONS

SPEC numbers over 2007 - 2016 are completely depressing.

REMARK: similar analysis was pursued on real apps and with different architectures for the period beyond 2017 and similar “depressing” results were obtained.

On real applications, the gap between peak (nominal) performance and observed performance is increasing!!

Two possible choices:

1. Improve Compiler Autotuning: OK but will provide at best 10 to 20% perf improvement
2. Rewrite applications: OK but might be very costly and might be difficult for most application fields

A FEW DIRECTIONS

More powerful approaches

- ➢ Split performance optimizations into 3 subproblems
  - Identify performance issue (diagnostic)
  - Identify potential remedies
  - Implement code transformations (directives, pragma or rewriting)
- ➢ Use performance tools to guide application restructuring but stop giving detailed diagnostics that a standard user cannot understand or lead to the wrong path. Instead of pointing to problems, suggest and evaluate potential solutions. THINK AS A DOCTOR ☺
- ➢ BEYOND CODE OPTIMIZATION: Use performance tools to conduct application performance characterization and drive hardware/software co design

THE STAGE

- A few basic ingredients: latency, bandwidth, dependencies,

- A very large number of possible combinations
  - Quantitative nature of Basic ingredients: cache miss ratio can vary incrementally between 0 and 100%
  - Raw values of basic ingredients might be unusable: number of stalls due to a buffer full)
  - Raw values of values are useless only combinations are meaning full: for example cache miss ratio is not meaningful only cache miss x average time for miss is meaningful

- The user is often lost by hardware counters and to some extent he does not care because they are not accurate enough: within a loop, you need to precisely identify the array which are causing trouble because it is where you will have to focus your efforts.

OUR VIEW ON PERFORMANCE COUNTERS

Hardware performance counters:

- Too many of them
- Very little documentation if any
- Broken counters are not publicized
- Needs detailed understanding of micro architecture to exploit them
- Hard to distinguish between cause and consequence
- When a resource is saturated, does it hurt performance ??
- They change with every processor generation.

OUR APPROACH

NOBODY WANTS PROBLEMS EVERYBODY WANTS SOLUTIONS ☺

- It is nice to correlate source line numbers with hardware counters values but this is not enough because the user cannot change hardware in general ☺
- We should concentrate on the knobs that the code developer has at his disposal:
  - Better compiler options
  - Code restructuring
  - Data restructuring
- More precisely, MAQAO (our toolset) will
  - Identify well known issues: small trip count, complex control flow, lack of vectorization, poor vectorization etc…..
  - Use what if methods to predict (more or less accurately) the impact of removing the issue
  - Also Use what if methods to predict the impact of standard transformations: partial of full vectorization etc….
  - Use comparison to get a better understanding of code behaviour

MAQAO “WHAT IF” SCENARIOS: UNICORE

- Generate performance estimates for different transformations

- Work at the innermost loop level (ASM)

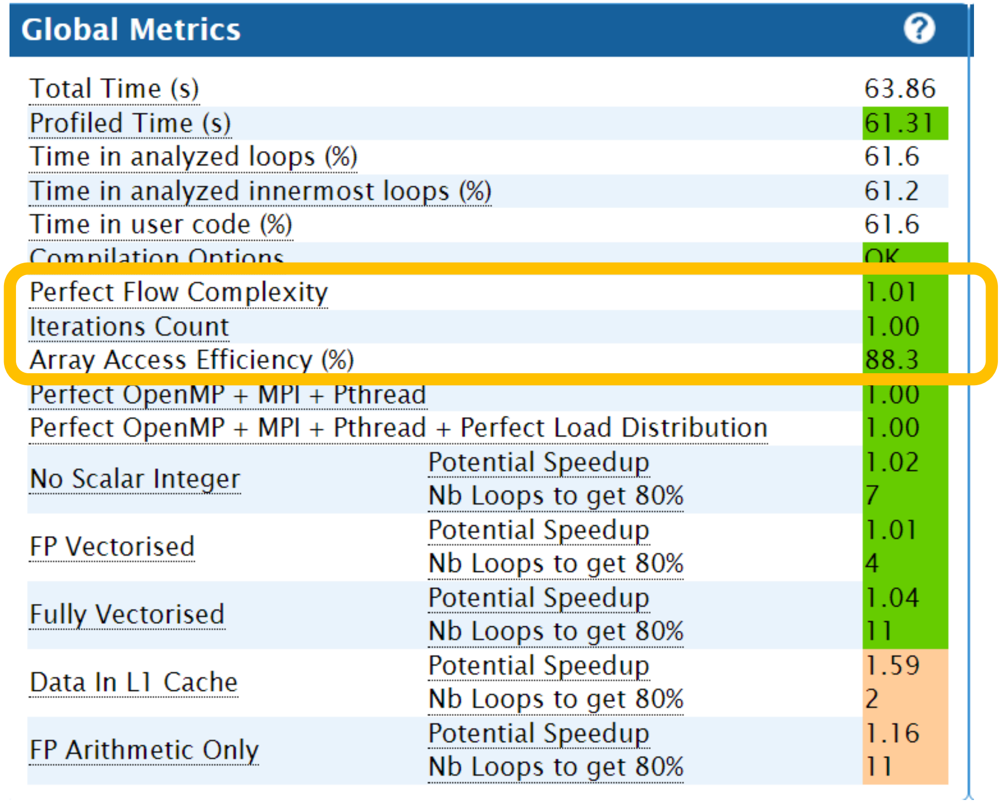
- First modify ASM to take into account code transformation: “Clean” version (only FP operations are kept), “FP Vector” (only vectorizes FP arithmetic), “Full Vector” (vectorizes all FP operations), “DL1” (forces all of the operands to come from L1, ……

- Second generate performance estimates
  - Either using static more or less simplified simulators
  - Or embed mofied ASM in the real code and measure it.

- Many more what if scenario can be derived: suppressing branches, suppressing costly FP operations (Div/SQRT)

- All of thesesv metrics will be reorgniazed and aggregated through the Oneview module

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
Data In L1 Cache	Potential Speedup	1.59
	Nb Loops to get 80%	2
FP Arithmetic Only	Potential Speedup	1.16
	Nb Loops to get 80%	11

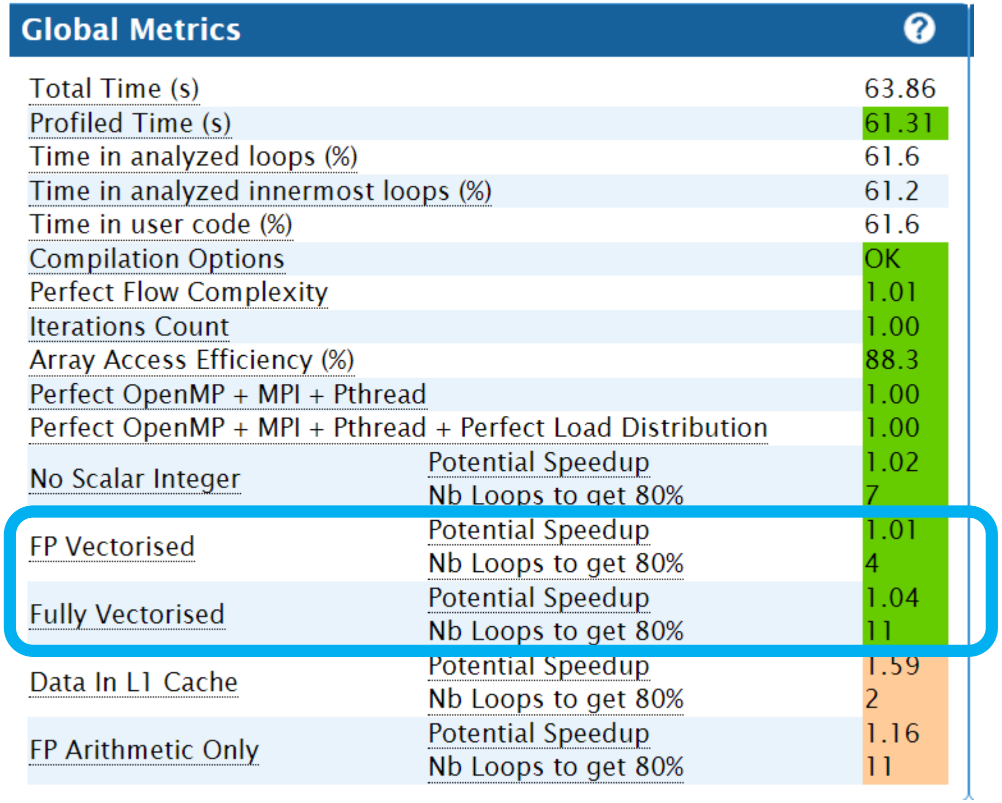
FOCUS: on transformations and impact at the application level MiniQMC (proxy for QMCPACK) code running on SKL and ICC 19.

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

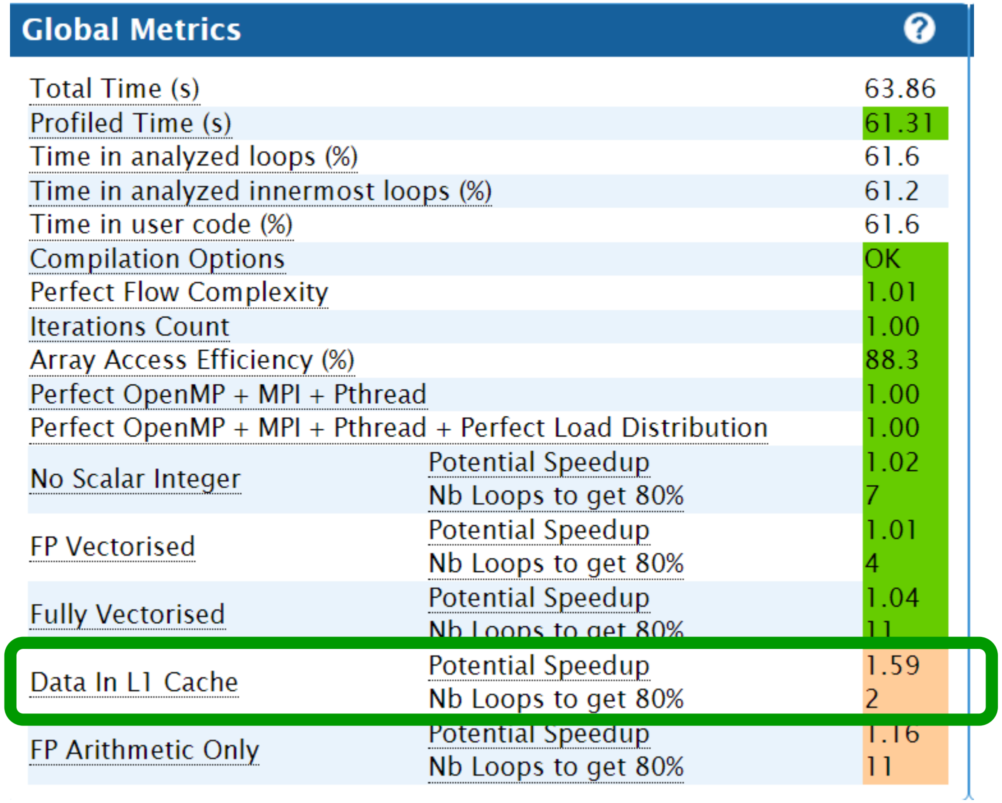
Total Time (s)		63.86
Profiled Time (s)		61.31
Time in analyzed loops (%)		61.6
Time in analyzed innermost loops (%)		61.2
Time in user code (%)		61.6
Compilation Options		OK
Perfect Flow Complexity		1.01
Iterations Count		1.00
Array Access Efficiency (%)		88.3
Perfect OpenMP + MPI + Pthread		1.00
Perfect 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
Data In L1 Cache	Potential Speedup	1.59
	Nb Loops to get 80%	2
FP Arithmetic Only	Potential Speedup	1.16
	Nb Loops to get 80%	11

FOCUS: on transformations and impact at the application level MiniQMC (proxy for QMCPACK) code running on SKL and ICC 19.

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 vectorized

TYPICAL ONE VIEW GLOBAL TAB



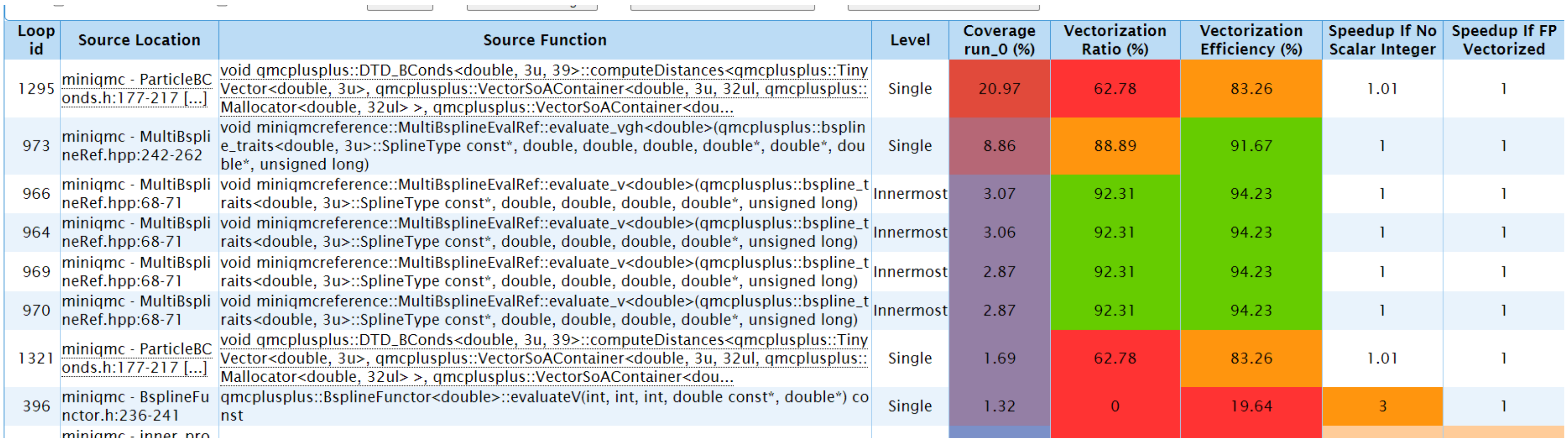
Global Metrics		
Total Time (s)		63.86
Profiled Time (s)		61.31
Time in analyzed loops (%)		61.6
Time in analyzed innermost loops (%)		61.2
Time in user code (%)		61.6
Compilation Options		OK
Perfect Flow Complexity		1.01
Iterations Count		1.00
Array Access Efficiency (%)		88.3
Perfect OpenMP + MPI + 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
Data In L1 Cache	Potential Speedup	1.59
	Nb Loops to get 80%	2
FP Arithmetic Only	Potential Speedup	1.16
	Nb Loops to get 80%	11

FOCUS: on transformations and impact at the application level MiniQMC (proxy for QMCPACK) code running on SKL and ICC 19.

Data in L1 cache: Performance gain if all of the operands are coming from L1

ONE VIEW: LOOP TAB (2)

MINIQMC: ARM Clang + ARM PL

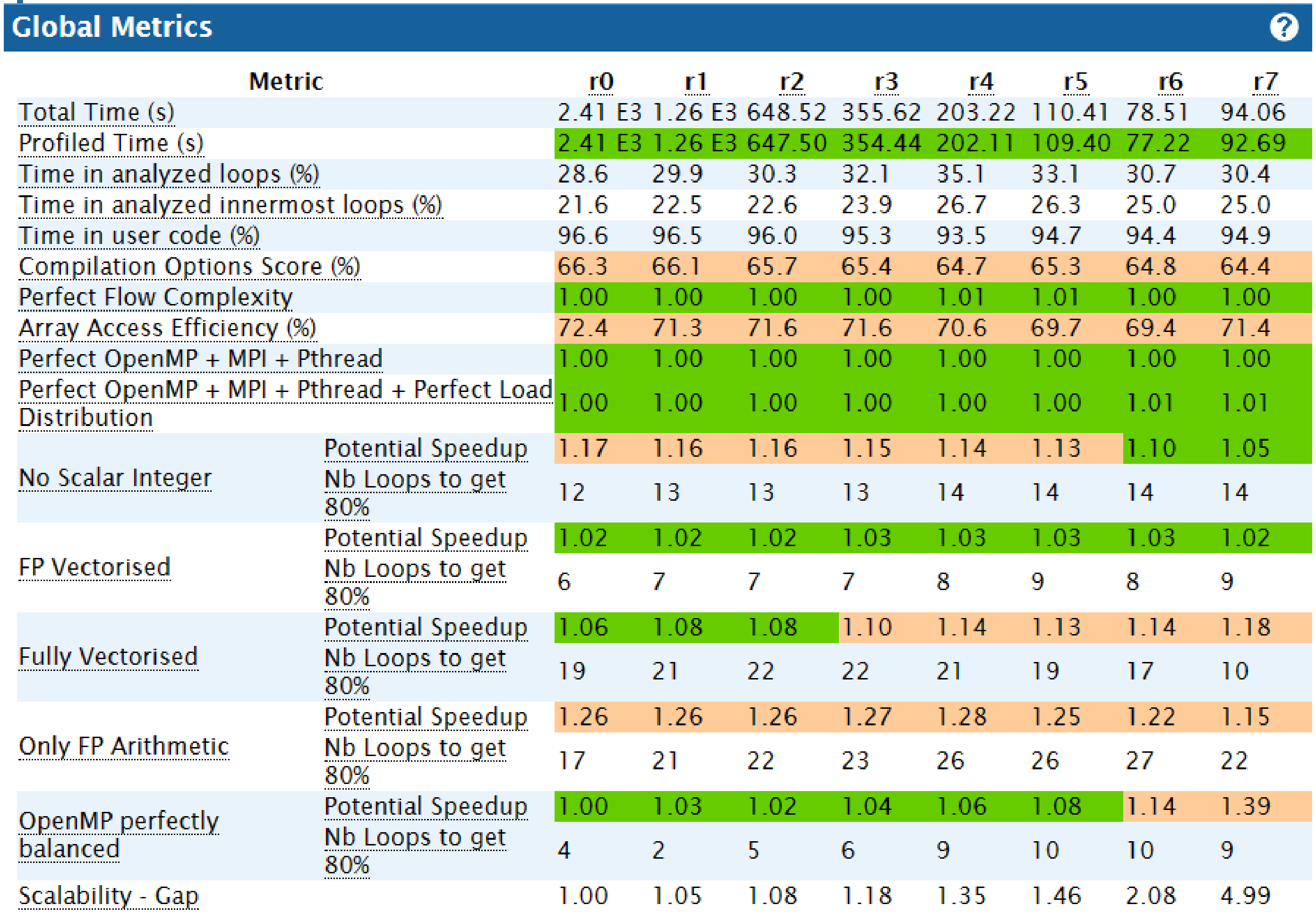


Loop id	Source Location	Source Function	Level	Coverage run_0 (%)	Vectorization Ratio (%)	Vectorization Efficiency (%)	Speedup If No Scalar Integer	Speedup If FP Vectorized
1295	miniqmc - ParticleBConds.h:177-217 [...]	void qmcplusplus::DTD_BConds<double, 3u, 39>::computeDistances<qmcplusplus::TinyVector<double, 3u>, qmcplusplus::VectorSoAContainer<double, 3u, 32ul, qmcplusplus::Mallocator<double, 32ul> >, qmcplusplus::VectorSoAContainer<dou...	Single	20.97	62.78	83.26	1.01	1
973	miniqmc - MultiBsplineRef.hpp:242-262	void miniqmcreference::MultiBsplineEvalRef::evaluate_vgh<double>(qmcplusplus::bspline_traits<double, 3u>::SplineType const*, double, double, double, double*, double*, double*, unsigned long)	Single	8.86	88.89	91.67	1	1
966	miniqmc - MultiBsplineRef.hpp:68-71	void miniqmcreference::MultiBsplineEvalRef::evaluate_v<double>(qmcplusplus::bspline_traits<double, 3u>::SplineType const*, double, double, double, double*, unsigned long)	Innermost	3.07	92.31	94.23	1	1
964	miniqmc - MultiBsplineRef.hpp:68-71	void miniqmcreference::MultiBsplineEvalRef::evaluate_v<double>(qmcplusplus::bspline_traits<double, 3u>::SplineType const*, double, double, double, double*, unsigned long)	Innermost	3.06	92.31	94.23	1	1
969	miniqmc - MultiBsplineRef.hpp:68-71	void miniqmcreference::MultiBsplineEvalRef::evaluate_v<double>(qmcplusplus::bspline_traits<double, 3u>::SplineType const*, double, double, double, double*, unsigned long)	Innermost	2.87	92.31	94.23	1	1
970	miniqmc - MultiBsplineRef.hpp:68-71	void miniqmcreference::MultiBsplineEvalRef::evaluate_v<double>(qmcplusplus::bspline_traits<double, 3u>::SplineType const*, double, double, double, double*, unsigned long)	Innermost	2.87	92.31	94.23	1	1
1321	miniqmc - ParticleBConds.h:177-217 [...]	void qmcplusplus::DTD_BConds<double, 3u, 39>::computeDistances<qmcplusplus::TinyVector<double, 3u>, qmcplusplus::VectorSoAContainer<double, 3u, 32ul, qmcplusplus::Mallocator<double, 32ul> >, qmcplusplus::VectorSoAContainer<dou...	Single	1.69	62.78	83.26	1.01	1
396	miniqmc - BsplineFunctor.h:236-241	qmcplusplus::BsplineFunctor<double>::evaluateV(int, int, int, double const*, double*) const	Single	1.32	0	19.64	3	1
	miniqmc - inner_pro							

“WHAT IF” SCENARIOS MULTICORE (1)

- Evaluate the performance impact of code transformation.
- Rely on performance measurement (either profiling or tracing (OMPT))
- Generate performance estimate:
  - Perfect OpenMP + MPI + Pthread: for each thread, get rid of time spent in OpenMP, MPI, Thread libraries and then takes max
  - Perfect OpenMP + MPI + Pthread + Perfect Load Distribution: for each thread, get rid of time spent in OpenMP, MPI, Thread libraries and then takes average
- Performance estimate generates either globally or at the loop level

WHAT IF MUTICORE SCENARIO



Global Metrics

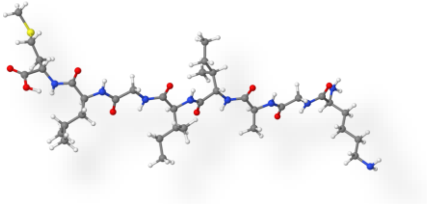
Metric		r0	r1	r2	r3	r4	r5	r6	r7
Total Time (s)		2.41 E3	1.26 E3	648.52	355.62	203.22	110.41	78.51	94.06
Profiled Time (s)		2.41 E3	1.26 E3	647.50	354.44	202.11	109.40	77.22	92.69
Time in analyzed loops (%)		28.6	29.9	30.3	32.1	35.1	33.1	30.7	30.4
Time in analyzed innermost loops (%)		21.6	22.5	22.6	23.9	26.7	26.3	25.0	25.0
Time in user code (%)		96.6	96.5	96.0	95.3	93.5	94.7	94.4	94.9
Compilation Options Score (%)		66.3	66.1	65.7	65.4	64.7	65.3	64.8	64.4
Perfect Flow Complexity		1.00	1.00	1.00	1.00	1.01	1.01	1.00	1.00
Array Access Efficiency (%)		72.4	71.3	71.6	71.6	70.6	69.7	69.4	71.4
Perfect OpenMP + MPI + Pthread		1.00	1.00	1.00	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	1.00	1.01	1.01
No Scalar Integer	Potential Speedup	1.17	1.16	1.16	1.15	1.14	1.13	1.10	1.05
	Nb Loops to get 80%	12	13	13	13	14	14	14	14
FP Vectorised	Potential Speedup	1.02	1.02	1.02	1.03	1.03	1.03	1.03	1.02
	Nb Loops to get 80%	6	7	7	7	8	9	8	9
Fully Vectorised	Potential Speedup	1.06	1.08	1.08	1.10	1.14	1.13	1.14	1.18
	Nb Loops to get 80%	19	21	22	22	21	19	17	10
Only FP Arithmetic	Potential Speedup	1.26	1.26	1.26	1.27	1.28	1.25	1.22	1.15
	Nb Loops to get 80%	17	21	22	23	26	26	27	22
OpenMP perfectly balanced	Potential Speedup	1.00	1.03	1.02	1.04	1.06	1.08	1.14	1.39
	Nb Loops to get 80%	4	2	5	6	9	10	10	9
Scalability - Gap		1.00	1.05	1.08	1.18	1.35	1.46	2.08	4.99

r0 (resp. r1 , r2, etc…) denotes runs on 1 (resp. 2, 4 etc…) cores

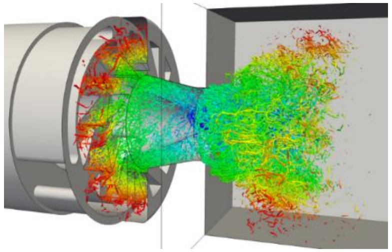
GROMACS running on a 2x 64 cores AMD EPYC7H12:. The dataset was chosen small enough to show scaling issues with more than 16 cores .

A few Success stories

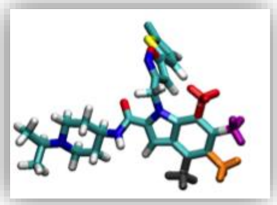
MAQAO is used for optimizing industrial and academic HPC applications:



- QMC=CHEM (IRSAMC)
  - Quantum chemistry
  - Speedup: **> 3x**
    - Optimization: moved invocations of functions with identical parameters out of the loop body



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



- Polaris (CEA)
  - Molecular dynamics
  - Speedup: **1.5x – 1.7x**
    - Optimization: enforcing loop vectorization through compiler directives
- AVBP (CERFACS)
  - Computational fluid dynamics
  - Speedup: **1.08x – 1.17x**
    - Replaced divisions by reciprocal multiplications
    - Complete unrolling of loops with a small number of iterations

FIRST WRAP UP ON MAQAO/OV

- Optimizing (complex) for complex recent architectures is becoming more and more difficult
- We need a new generation of performance tools to guide the code/developer through that task

MAQAO/ONE VIEW provides a new approach

- Provides an application centric view
- Provides synthetic/aggregated view meaningful for the user
- Provides performance estimates of potential gains (what if scenarios)
- + Other goodies: wait for the next talks….

ENERGY: New and mandatory challenges

C: Capacity: performance metric (Flops per cycle, Transactions per cycle, etc.…)

E: Energy consumed by a computation

- Only maximizing C is no longer a correct objective because it might lead to unacceptable power/energy costs
- Only minimizing E is not a correct objective either because it leads to low capacities.
- Race to Halt strategies are also too short minded because they essentially assume constant power

Performance and energy models are needed

Real Objectives

- Maximising a Quality metric (C, C/E) under constraints (Constant Power, Constant Capacity)
- To be correctly addressed, such objectives needs performance models which will use as an essential component “measurements”
- Performance tools needs to add predictive power to predict power behavior and performance behavior.

There is still a long way to go……

A better title for this talk: “What a Long Strange Trip It’s Been”