Application Characterization:

A comprehensive way of analyzing and understanding interaction between hardware and software (applications/compiler/runtime): performance and energy

Jean Christophe Beyler, Intel ECR

BOF Submitted by:

Marie-Christine Sawley, Intel ECR & Gregori Fursin, INRIA

ECR Team

ECR team includes over 25 high level researchers

ECR Team is part of research labs in Europe and North America

ECR is a member of the Intel EMEA HPC Exascale labs together with ExaScience lab and ExaCluster lab

Part of DSG-DCSG

2 main Streams of Research

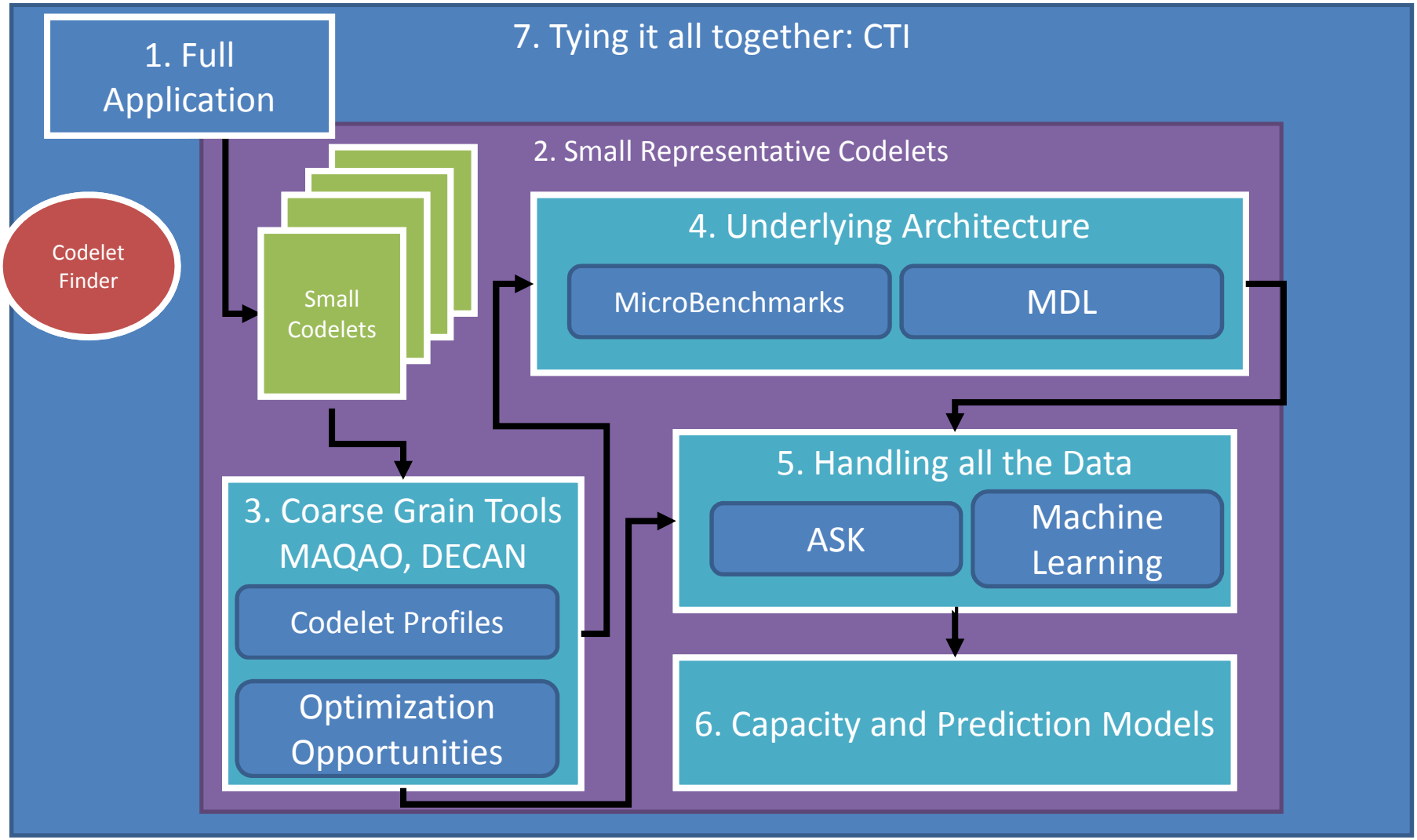
Software for application characterization and performance optimization

Extract fine grain information about the interaction of whole software with the underlying architecture

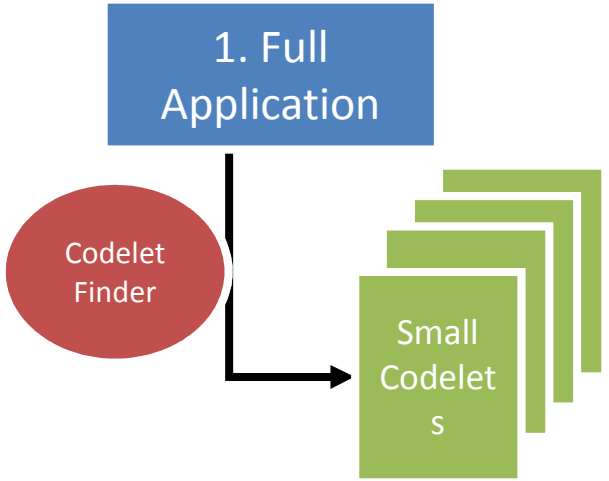
Application co-design

Leveraging from the low level information and the capacity of new architectures for enabling progress in science using computational power

Outline


1. Full Application
7. Tying it all together: CTI
2. Small Representative Codelets
Codelet Finder
Small Codelets
4. Underlying Architecture
MicroBenchmarks
MDL
5. Handling all the Data
ASK
Machine Learning
3. Coarse Grain Tools MAQAO, DECAN
Codelet Profiles
Optimization Opportunities
6. Capacity and Prediction Models

Cutting the Application Up

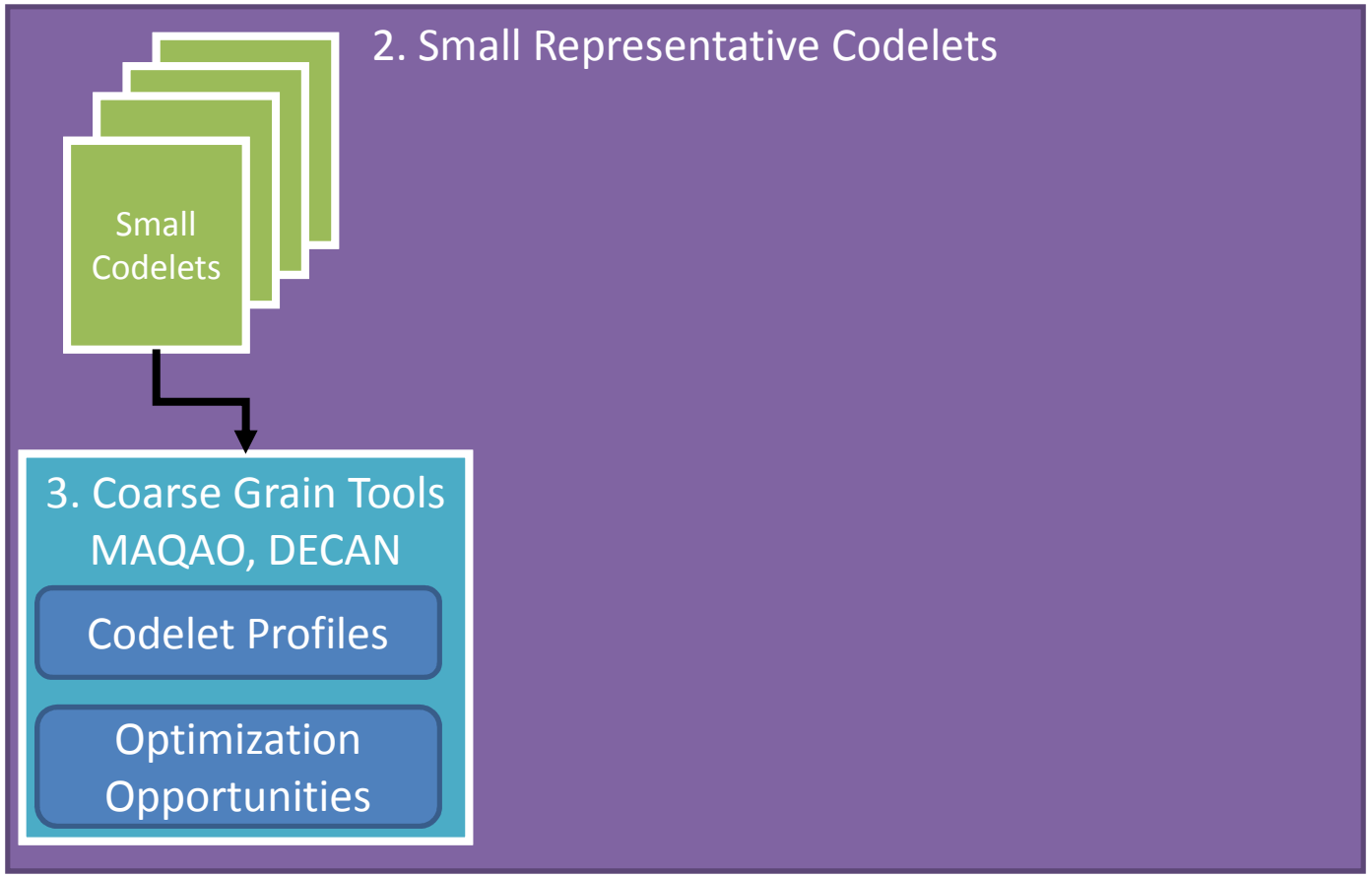


- First step: finding the hot spots
- Considering full applications is difficult
  - Study the hotspots
  - Automatic solution:
    - Using CAPS Enterprise's Codelet Finder tool
- Second step: work on the codelets separately

Codelet Finder

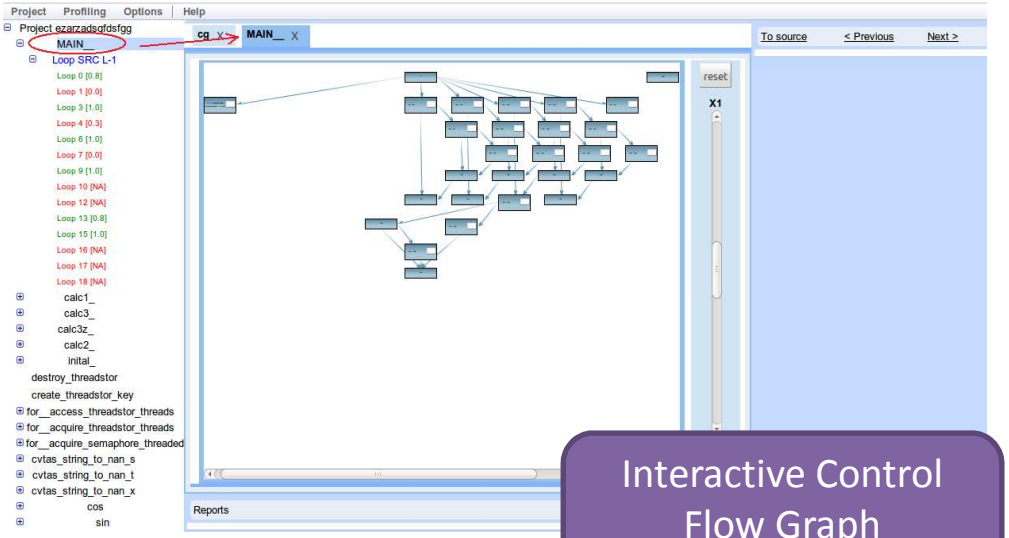
- Key features
  - Implemented by CAPS Enterprise
  - Handles C or Fortran codes
  - Automatically detects hotspots and extracts loops into:
    - Kernel, wrapper, data input
  - Data input is retrieved by a core dump before the kernel
- Future work
  - Allow users to modify input data easily
  - Add more supported constructs

Outline


2. Small Representative Codelets
Small Codelets
3. Coarse Grain Tools MAQAO, DECAN
Codelet Profiles
Optimization Opportunities

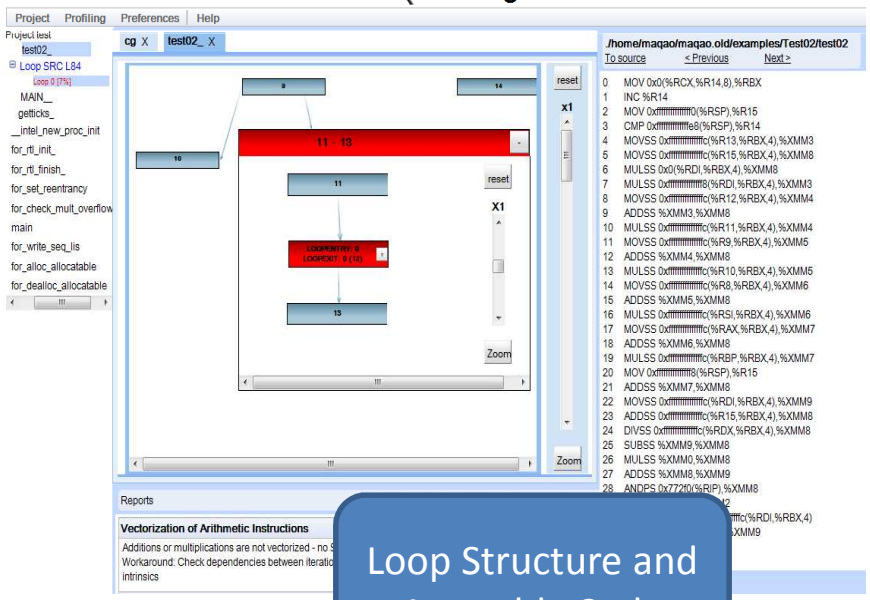
Tools to Quantify

- Logical next step
- Measure performance with profiling tools such as MAQAO
  - Provides important information
    - What can be optimized?
    - What is obtainable?



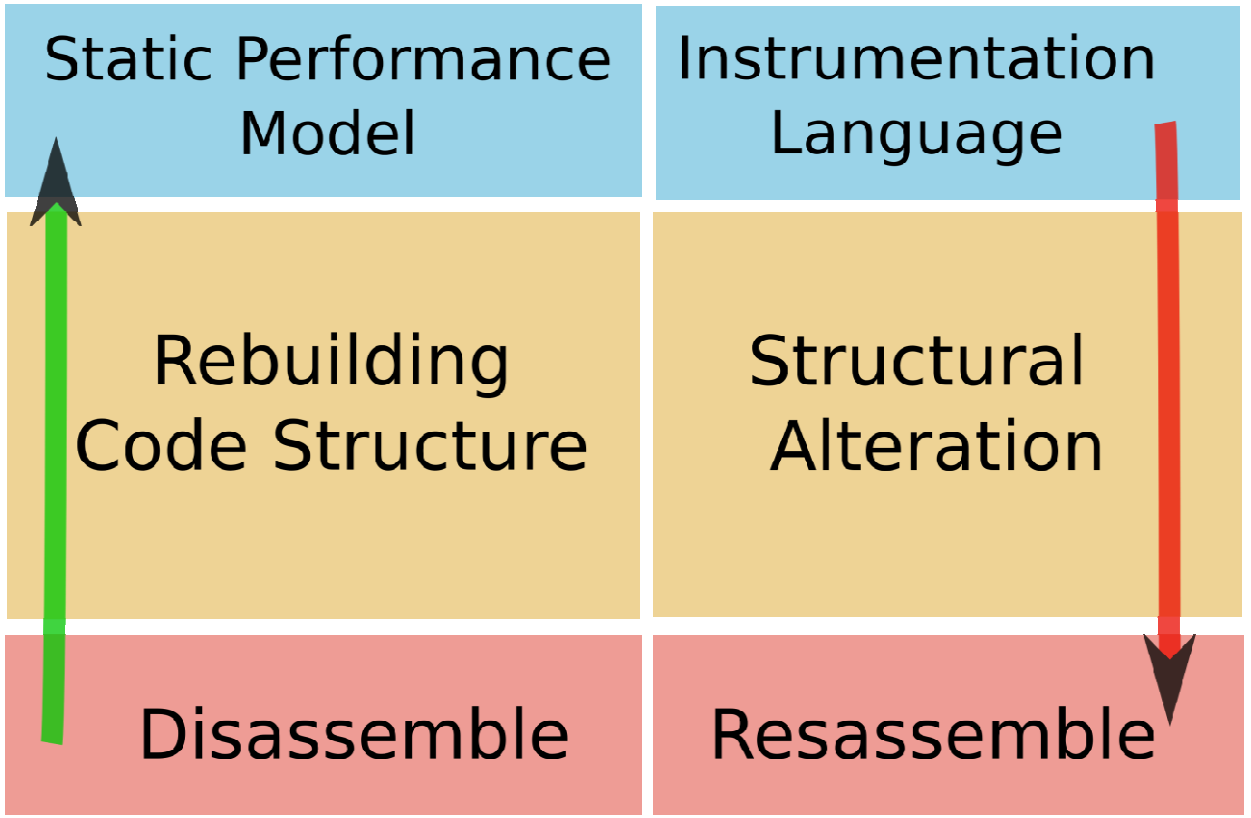
Interactive Control Flow Graph (Exagraph)

MAQAO.org Web Interface



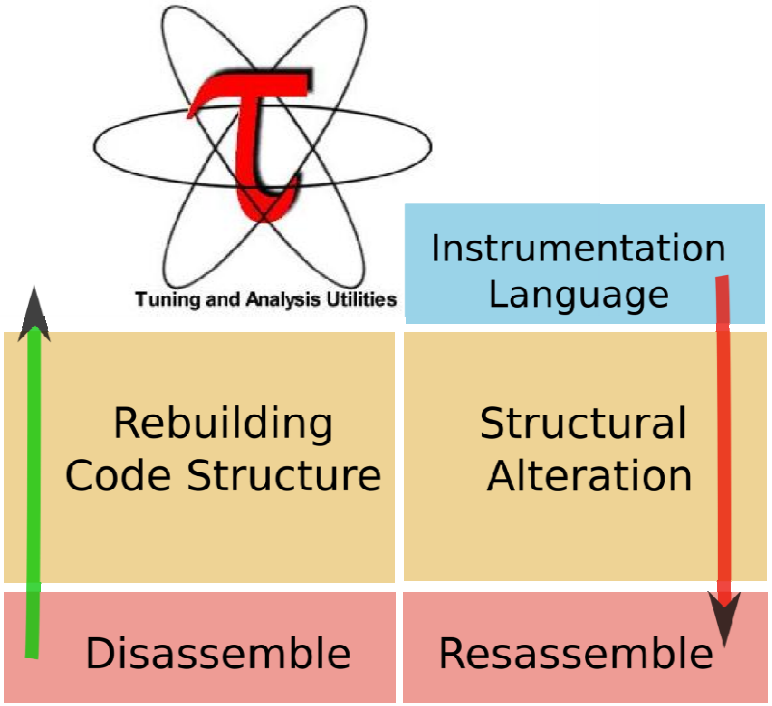
Loop Structure and Assembly Code

- Disassemble or reassemble SSE and AVX binaries
- Performance model for Core2, Nehalem, and Sandy Bridge
- Low overhead profiler
  - < 100 cycles per probe
  - OMP compliant



- Performance analysis
  - No pragma or source code alteration
  - Vectorization ratio
  - Detailed pipeline model:
    - Dispatch, decoder, LSD, per port pressure
  - Memory traffic
  - Aggregate memory instructions per group
  - Unrolling factor
- Static performance prediction
  - 'What if' the code is fully vectorized
  - 'What if' the data is stored in L1

- Modular Architecture
  - Demo of TAU using binary capacities of MAQAO at SC11



QMC == CHEM with MAQAO

```
!DIR$ VECTOR ALIGNED
do j=1,LDC
 C1(j,i)=C1(j,i)+(A(j,k_vec(1))*d11 &
                + A(j,k_vec(2))*d21 &
                + A(j,k_vec(3))*d31 &
                + A(j,k_vec(4))*d41)&

 C2(j,i)=C2(j,i)+(A(j,k_vec(1))*d12 &
                + A(j,k_vec(2))*d22 &
                + A(j,k_vec(3))*d32 &
                + A(j,k_vec(4))*d42)&
enddo
```

In green, SSE/AVX vectorized loops

	Nb used reg.		Performance in L1				
ID	XMM	YMM	Nb cycles	FLOP/cycle	IPC	Bytes loaded / cycle	Bytes stored / cycle
1a	12	0	4	2	3	5	1
1b	0	12	10	12.8	2.1	32	6.4
2a	16	0	8	2	3.125	3	1
2b	0	10	10	12.8	2.1	32	6.4

	Nb used reg.		Performance in L1				
ID	XMM	YMM	Nb cycles	FLOP/cycle	IPC	Bytes loaded / cycle	Bytes stored / cycle
1	0	15	8	16	3.125	24	8
2	0	15	8	16	3.125	24	8

MAQAO static analysis before (top) and after (bottom) optimization

- Dealing with the two hottest loops in the application
  - Dense x sparse matrix multiply
- FLOP/cycle not optimal:
  - 12.8 but should be 16
    - AVX, 32 bits elements, perfect ADD/MUL balance
- Replacing LDC with its value ”hard coded” allows the compiler to factor for the two matrices C1 and C2

Tools to Explain

- MAQAO and similar tools provide information
  - Profiler detects hot spot
  - MAQAO goes beyond and evaluates the gap
    - Current and optimal static performance
  - It remains the discrepancy is difficult to understand

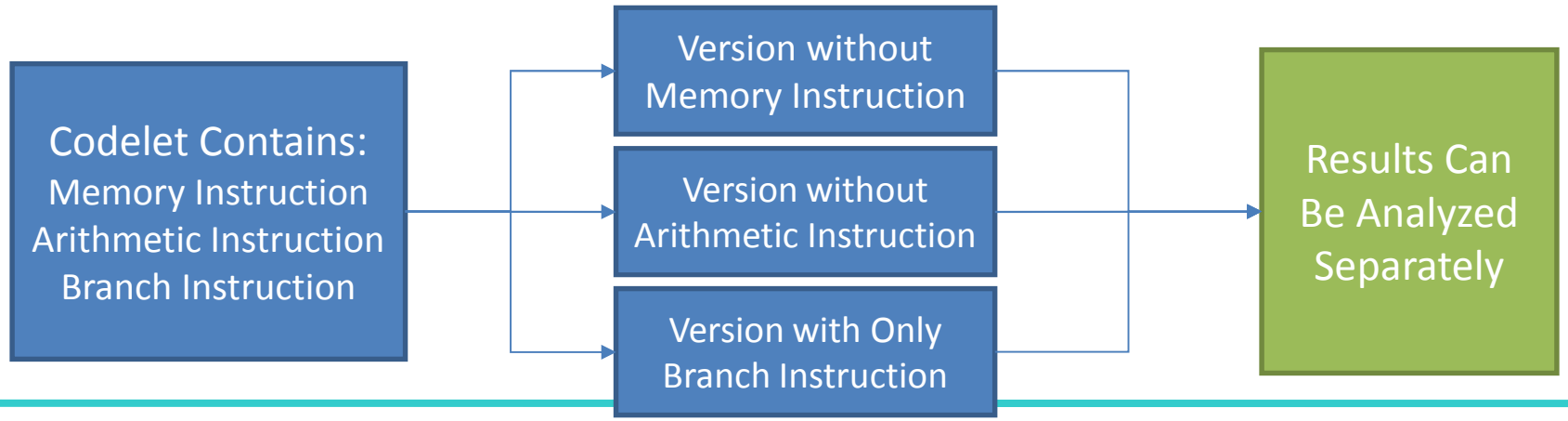
- DECAN is an exploratory tool

DECAN

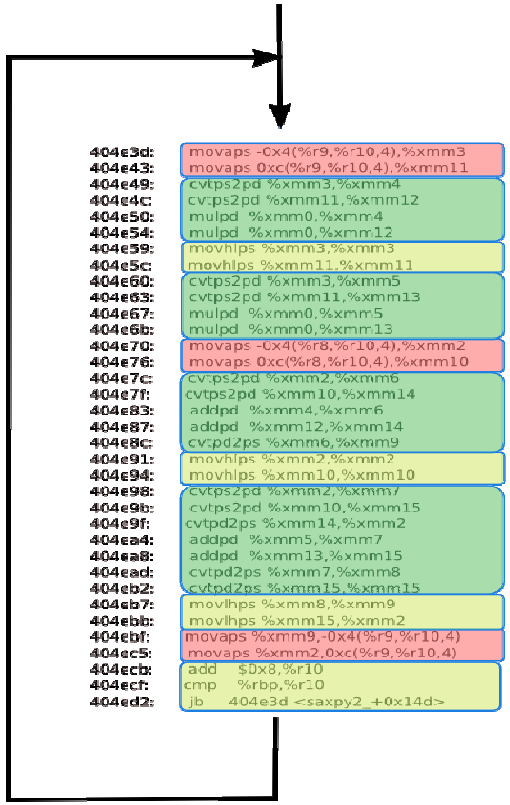
- DECAN's concept is simple
  - Measure the original binary
  - Patch and replace the selected instructions group in the original binary
  - New binary is generated for each patch
  - Measure new binaries
  - Measurements are represented in a CSV file
    - Analyze and compare

DECAN

- Codelet Decomposition
  - MISTREAM
    - All vector arithmetic instructions are deleted
  - FPSTREAM
    - All vector loads and store instructions are deleted
  - NOFPNOMISTREAM
    - All vector arithmetic, load, and store instructions are deleted



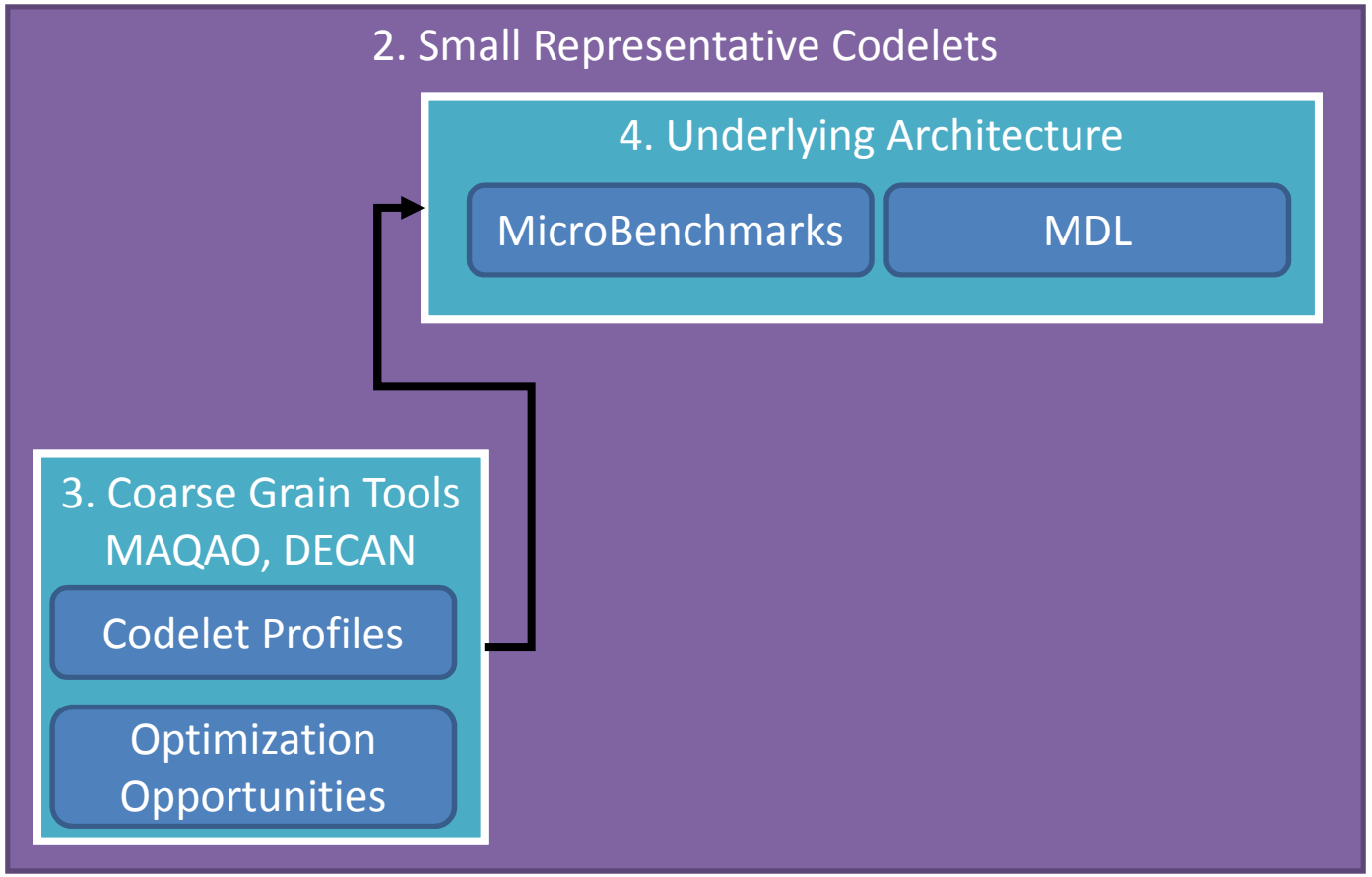
DECAN


404e3d: movaps -0x4(%r9,%r10,4),%xmm3
404e43: movaps 0xc(%r9,%r10,4),%xmm11
404e49: cvtps2pd %xmm3,%xmm4
404e4c: cvtps2pd %xmm11,%xmm12
404e50: mulpd %xmm0,%xmm4
404e54: mulpd %xmm0,%xmm12
404e59: movhlps %xmm3,%xmm3
404e5c: movhlps %xmm11,%xmm11
404e60: cvtps2pd %xmm3,%xmm5
404e63: cvtps2pd %xmm11,%xmm13
404e67: mulpd %xmm0,%xmm5
404e6b: mulpd %xmm0,%xmm13
404e70: movaps -0x4(%r8,%r10,4),%xmm2
404e76: movaps 0xc(%r8,%r10,4),%xmm10
404e7c: cvtps2pd %xmm2,%xmm6
404e7f: cvtps2pd %xmm10,%xmm14
404e83: addpd %xmm4,%xmm6
404e87: addpd %xmm12,%xmm14
404e8c: cvtpd2ps %xmm6,%xmm9
404e91: movhlps %xmm2,%xmm2
404e94: movhlps %xmm10,%xmm10
404e98: cvtps2pd %xmm2,%xmm7
404e9b: cvtps2pd %xmm10,%xmm15
404e9f: cvtpd2ps %xmm14,%xmm2
404ea4: addpd %xmm5,%xmm7
404ea8: addpd %xmm13,%xmm15
404ead: cvtpd2ps %xmm7,%xmm8
404eb2: cvtpd2ps %xmm15,%xmm15
404eb7: movlhps %xmm8,%xmm9
404ebb: movlhps %xmm15,%xmm2
404ebf: movaps %xmm9,-0x4(%r9,%r10,4)
404ec5: movaps %xmm2,0xc(%r9,%r10,4)
404ecb: add $0x8,%r10
404ecf: cmp %rbp,%r10
404ed2: jb 404e3d <saxpy2_+0x14d>

MAQAO + DECAN Provides

- Speed-up by a factor of 4
  - Up to 37% of the peak performance on Sandy Bridge
- Vectorization ratio crucial on Sandy Bridge
- Value profiling

Outline

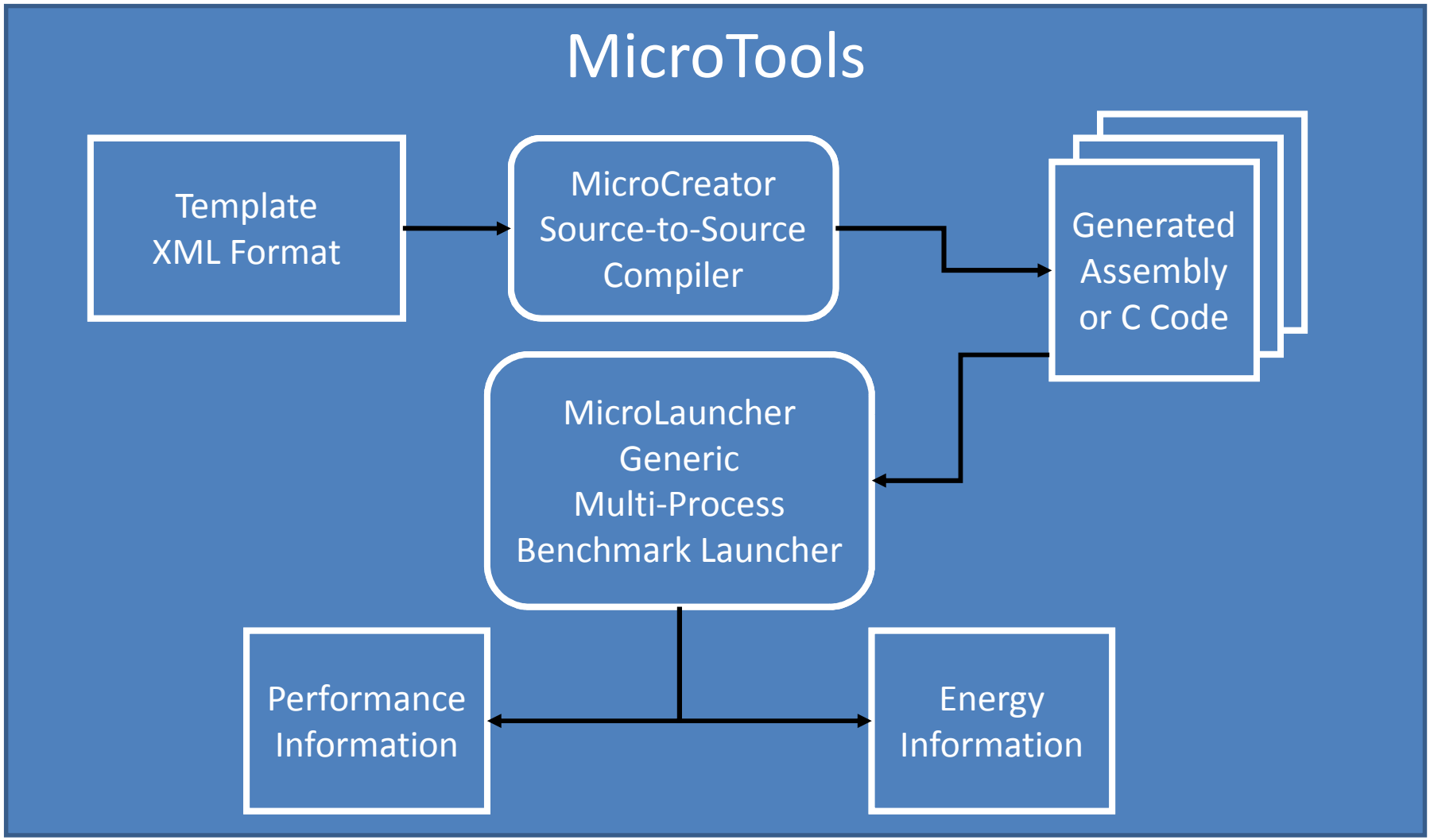

2. Small Representative Codelets
4. Underlying Architecture
MicroBenchmarks
MDL
3. Coarse Grain Tools
MAQAO, DECAN
Codelet Profiles
Optimization
Opportunities

Underlying Architecture

- Understanding the target architecture
  - Gives insight on potential bottlenecks
  - Provides solutions to optimize a code

- How is it done?
  - Emulators or simulators are slow if even available
  - Microbenchmarking considers the hardware as a black box

MICROBENCHMARKS



MicroTools Usage

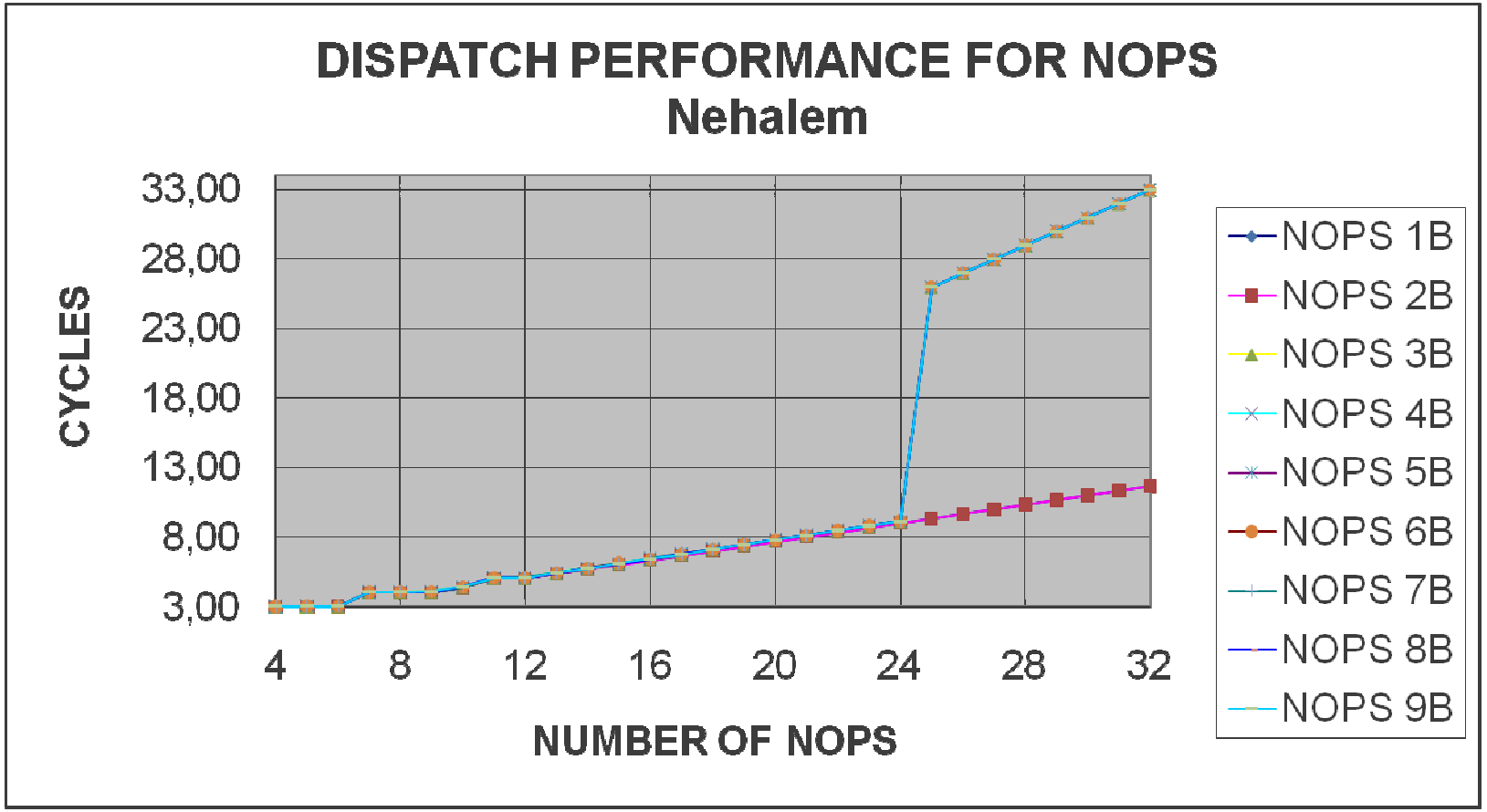
MicroTools enables an exhaustive exploration of architecture performance.

1) Two real case-studies
   - NOPS impact on dispatcher
   - Memory + arithmetic interaction
2) How to deal with all the data collected?
   - Automated reports analysis

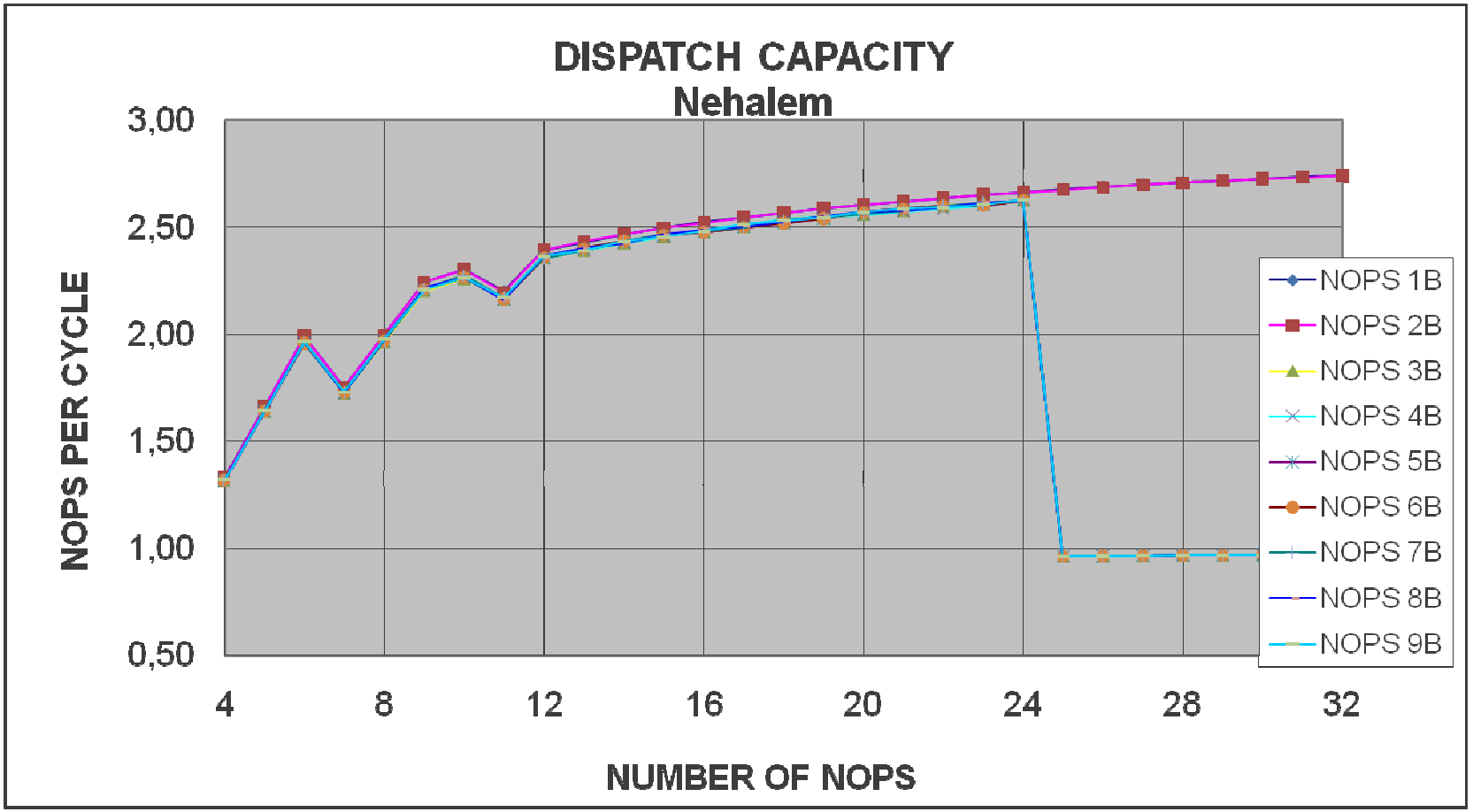
NOPS Experiment

- Goal: Evaluate the dispatch unit
- Loop body parameters
  - The NOP instruction size:
    - Varies from 1 to 9 bytes
  - The number of NOP instructions in the loop body:
    - Varies from 4 to 32
- Each loop body tested consists of the same NOP instructions repeated from 4 to 32 times

NOPS Experiment



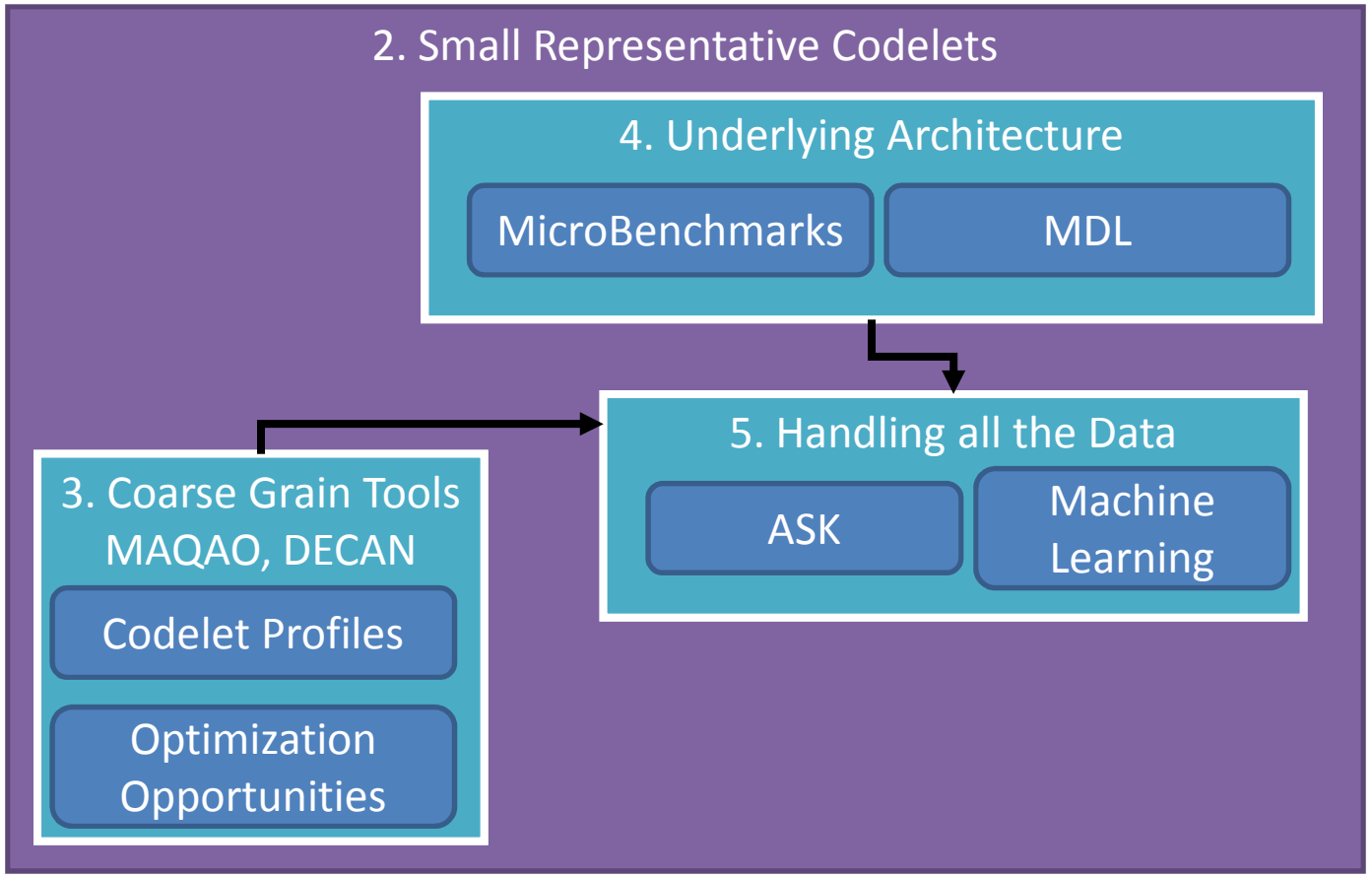
NOPS Experiment



NOPS Experiment

- Dispatcher capacity is around 3 nops/cycle
- For 1-byte and 2-byte nops
  - Dispatch behavior is linear
- For large size nops
  - Dispatch rate falls down to 0.96 instructions per cycle

Outline

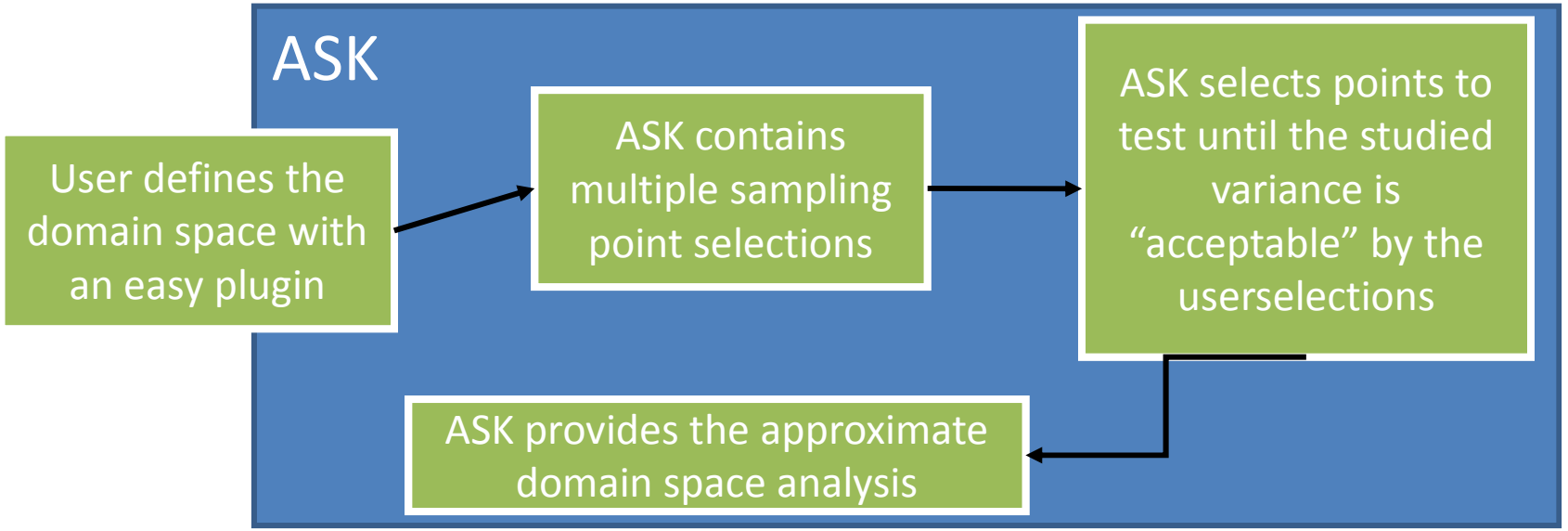

2. Small Representative Codelets
4. Underlying Architecture
MicroBenchmarks
MDL
5. Handling all the Data
ASK
Machine Learning
3. Coarse Grain Tools
MAQAO, DECAN
Codelet Profiles
Optimization Opportunities

HANDLING THE DATA

- MicroCreator philosophy:
  - Exhaustively search around a given program specification
  - Sometimes, we need to reduce the space (ASK)
- A lot of data produced
  - We need automatic data handling tools
    - Validate the stability of the results
    - Identify unexpected situations to help the engineer.

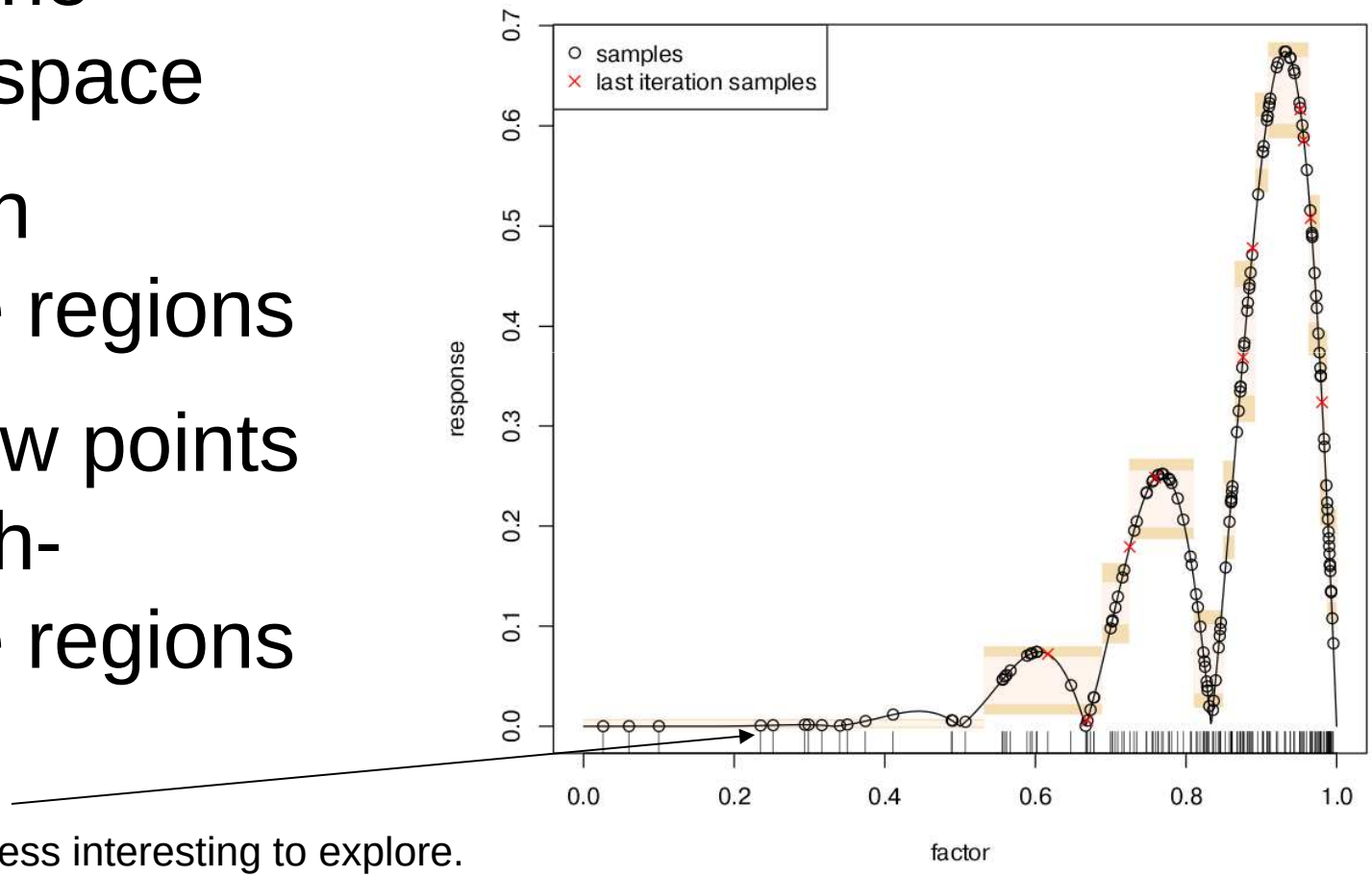
ASK (Adaptive Sampling Kit)

- ASK is a toolkit providing state of the art sampling strategies
- Modular and Extensible Pipeline:
  - Combine different static and dynamic strategies
  - Easy to add new custom sampling strategy
  - Easy integration with benchmarking tools



ASK: An example

- Search the domain space
- Find high variance regions
- Draw new points from high-variance regions



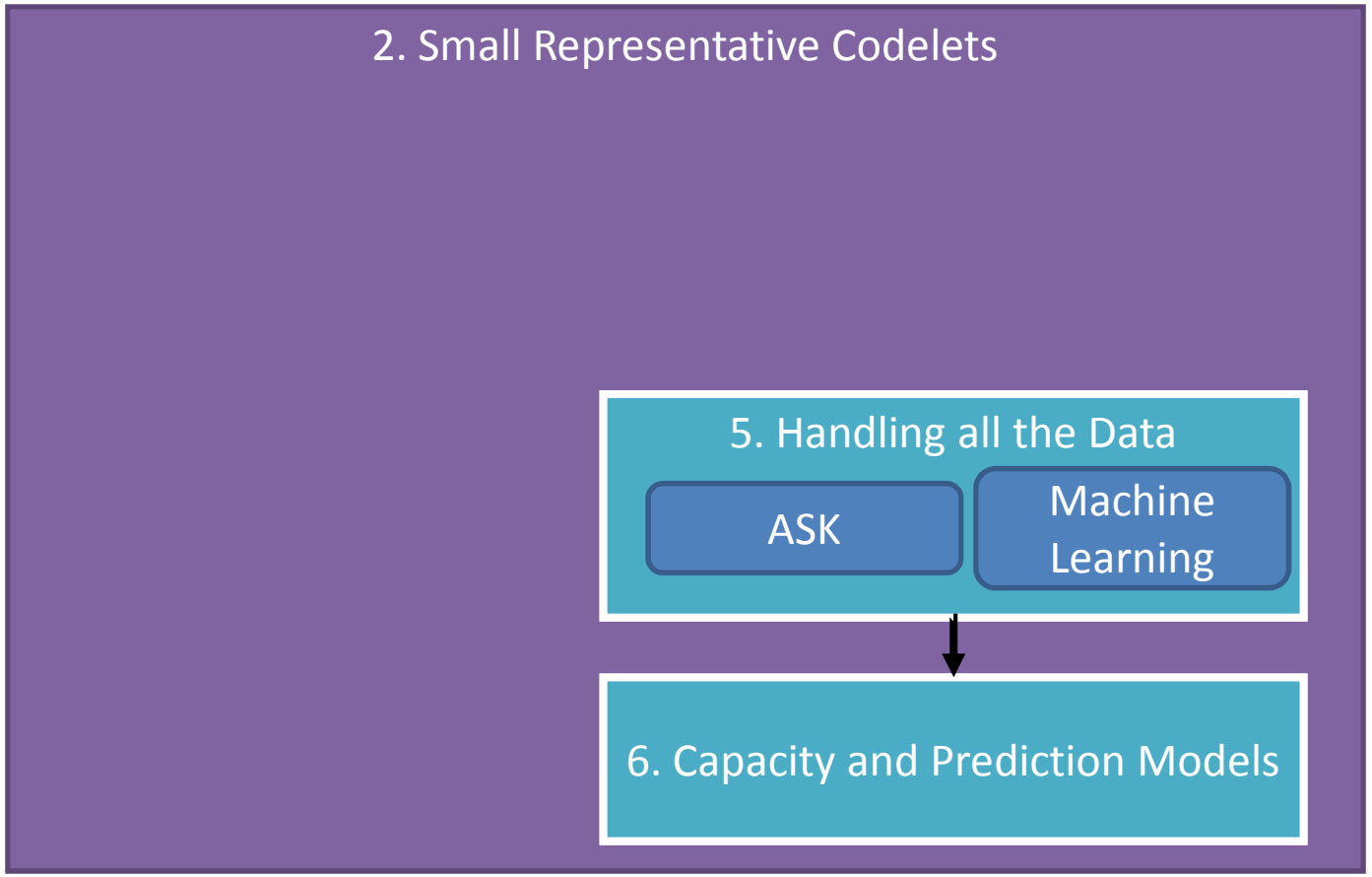
"Flat" regions are less interesting to explore.

Order Influence Report

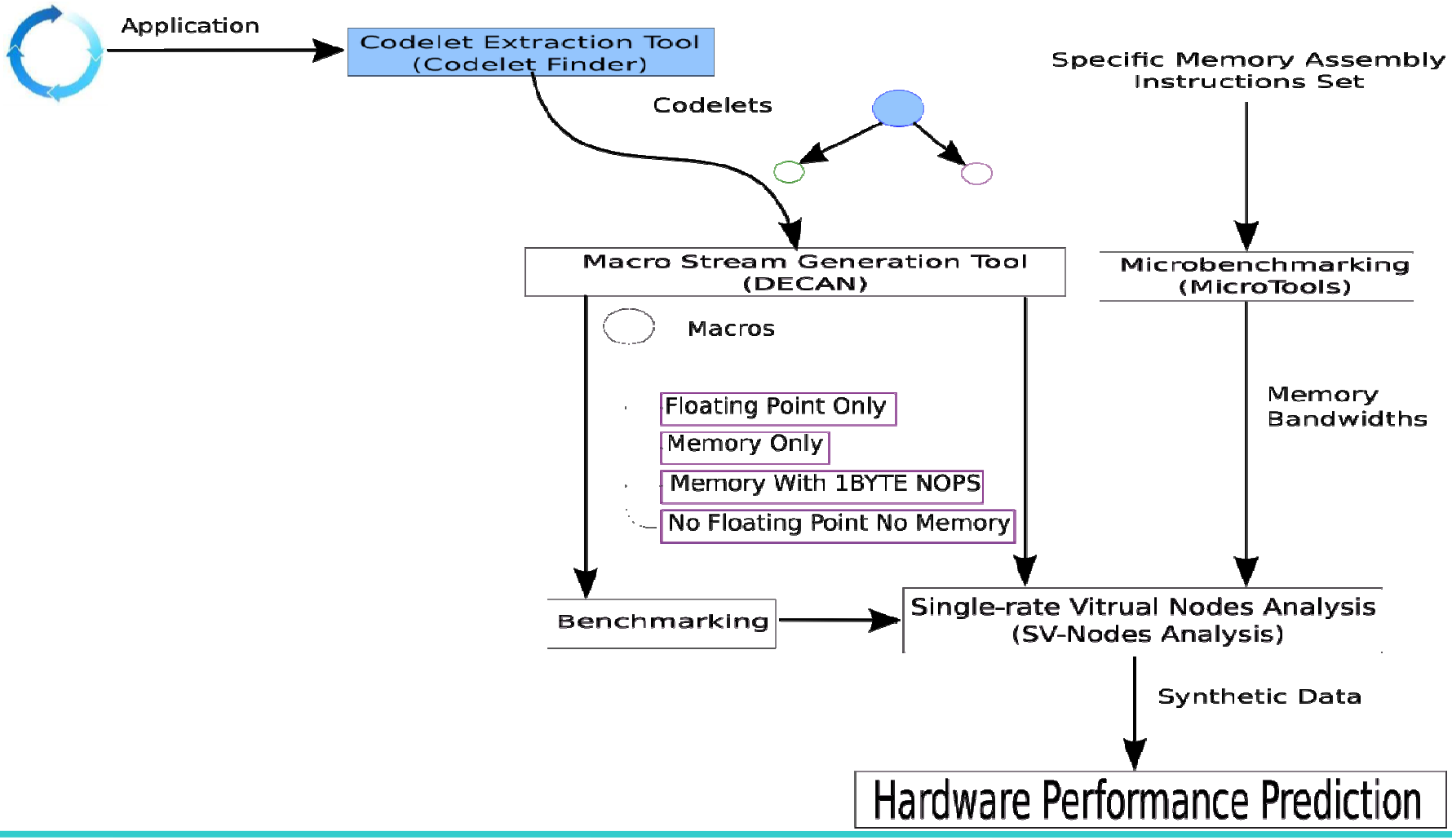
- Decomposes results per number of stores and loads
- Quickly identify configurations where performance depends on the order of instructions

MOVAPS Sandybridge L3														
#L \ #S		1		2		3		4		5		6		7
1	7.29	7.6	17.85	19.15	24.29	33.07	38.22	38.69	44.19	52.57	57.22	58.51	63.41	71.37
2	9.8	10.6	20.19	20.86	26.16	34.89	31.21	41.95	45.96	54.65	51.57	59.83		
3	12.88	13.18	21.46	22.81	27.92	37.55	33.22	45.15	36.77	56.53				
4	15.67	15.74	23.08	24.35	28.43	39.41	33.81	45.77						
5	18.56	19.01	24.89	25.7	30.55	41.53								
6	21.42	21.54	26.88	28.25										
7	24.55	25.67												

Outline


2. Small Representative Codelets
5. Handling all the Data
ASK
Machine Learning
6. Capacity and Prediction Models

Capacity Model

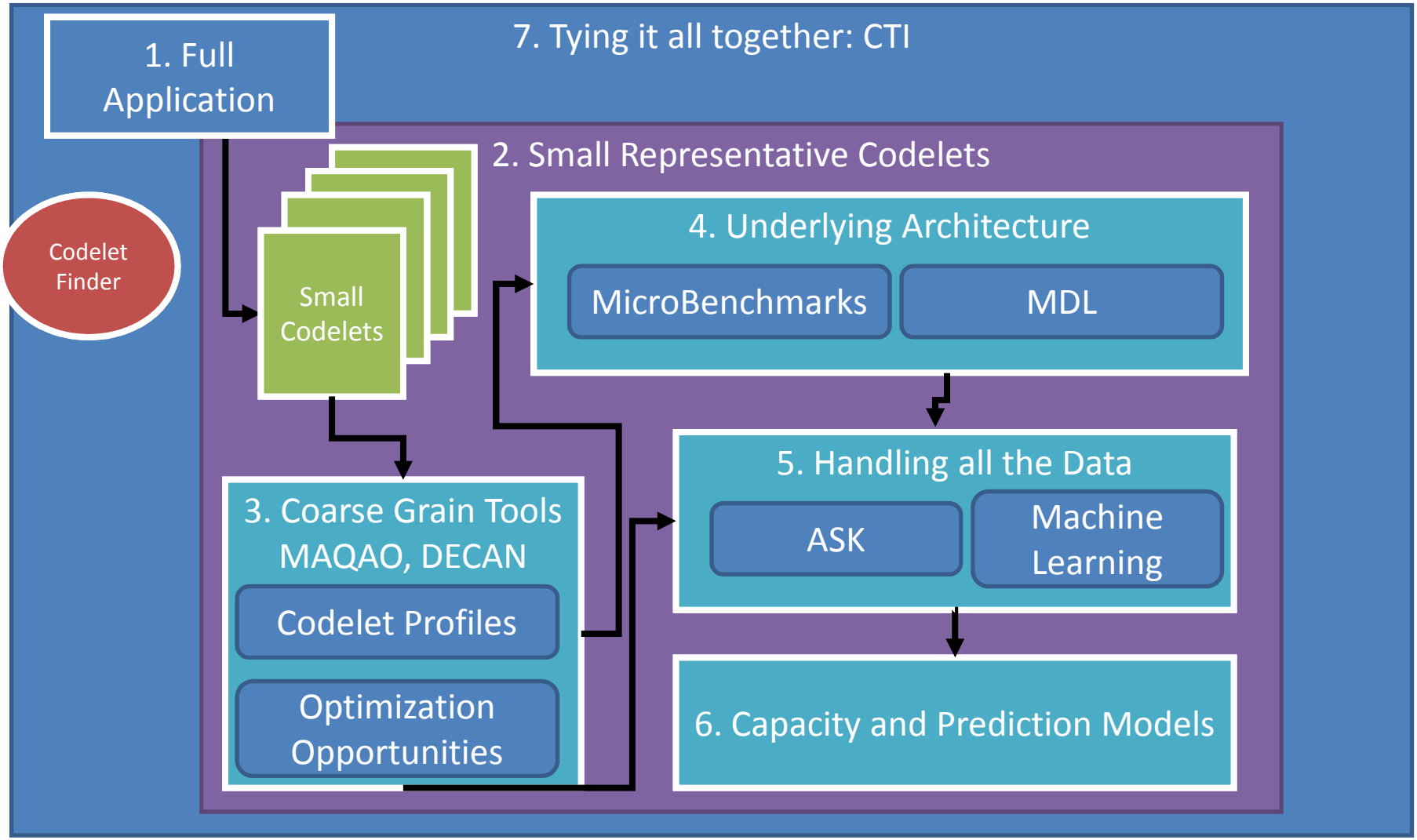


Capacity Model

- Capacity Model and MDL
  - Provide prediction and modelization of program performance
- However, alone the tools are less valuable
  - A need to centralize the data and analysis:

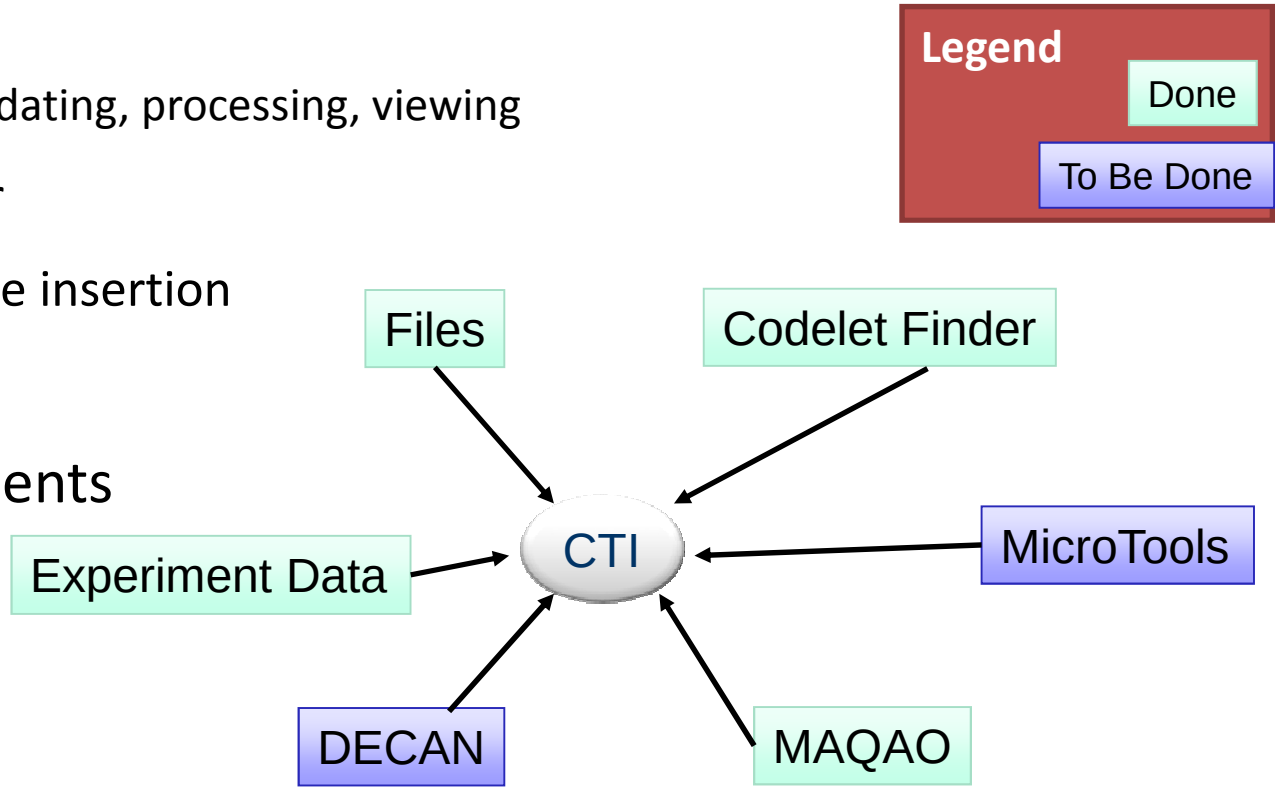
CTI : Codelet Tuning Instrastructure

Outline

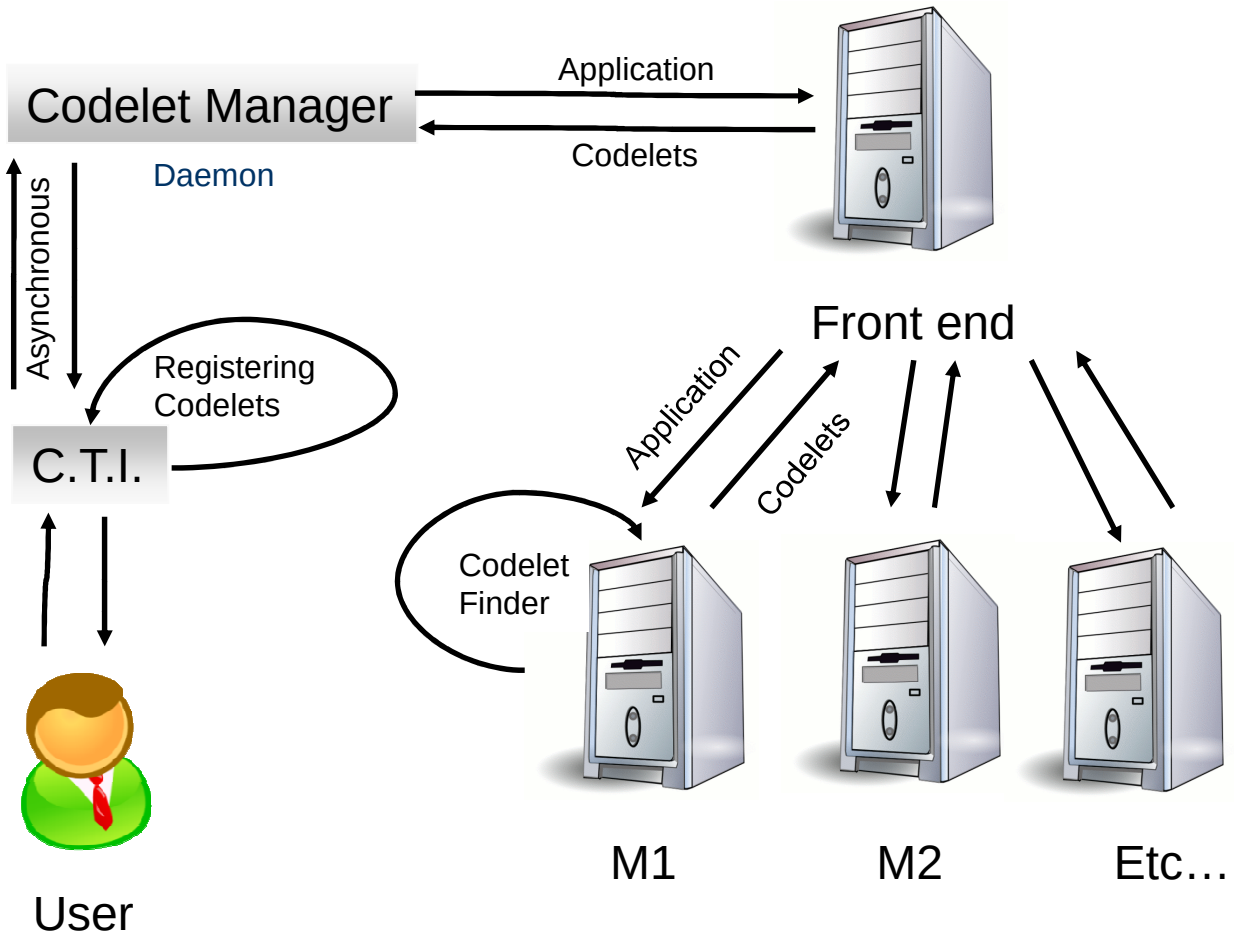

7. Tying it all together: CTI
1. Full Application
2. Small Representative Codelets
Codelet Finder
Small Codelets
4. Underlying Architecture
MicroBenchmarks
MDL
5. Handling all the Data
ASK
Machine Learning
3. Coarse Grain Tools MAQAO, DECAN
Codelet Profiles
Optimization Opportunities
6. Capacity and Prediction Models

Codelet Tuning Infastructure (CTI)

- A single place to store a huge amount of data
  - File manager
    - File sharing, updating, processing, viewing
  - Codelet manager
  - CSV automatic file insertion
  - Query the data
- Automate experiments
- Tools integrator



CTI Codelet Manager



CTI Codelet Manager

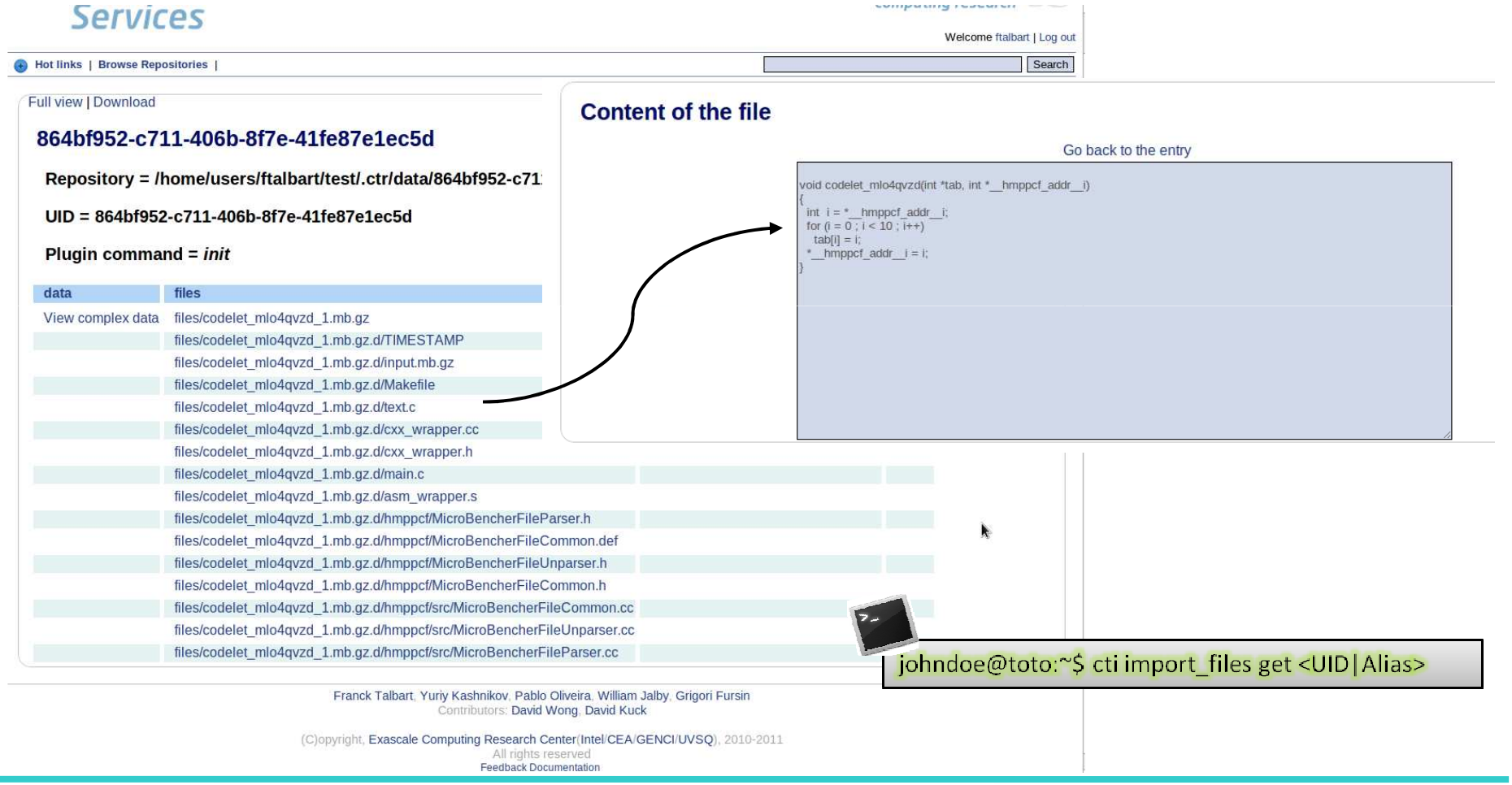
- Launching the « create » process

CTI Codelet Manager

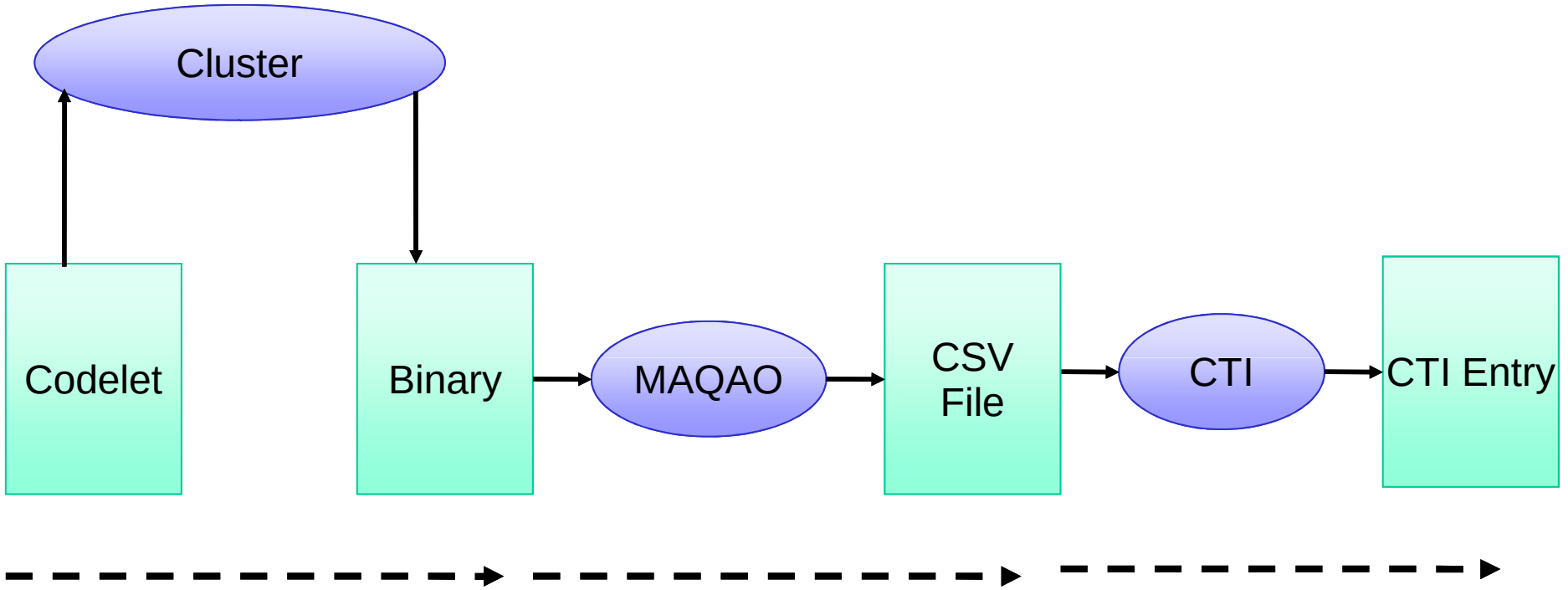
- Checking the codelets list

CTI Codelet Manager

- Checking the codelet files



CTI MAQAO Integration



1) Codelet to Binary

The codelet is sent to the cluster for complitation

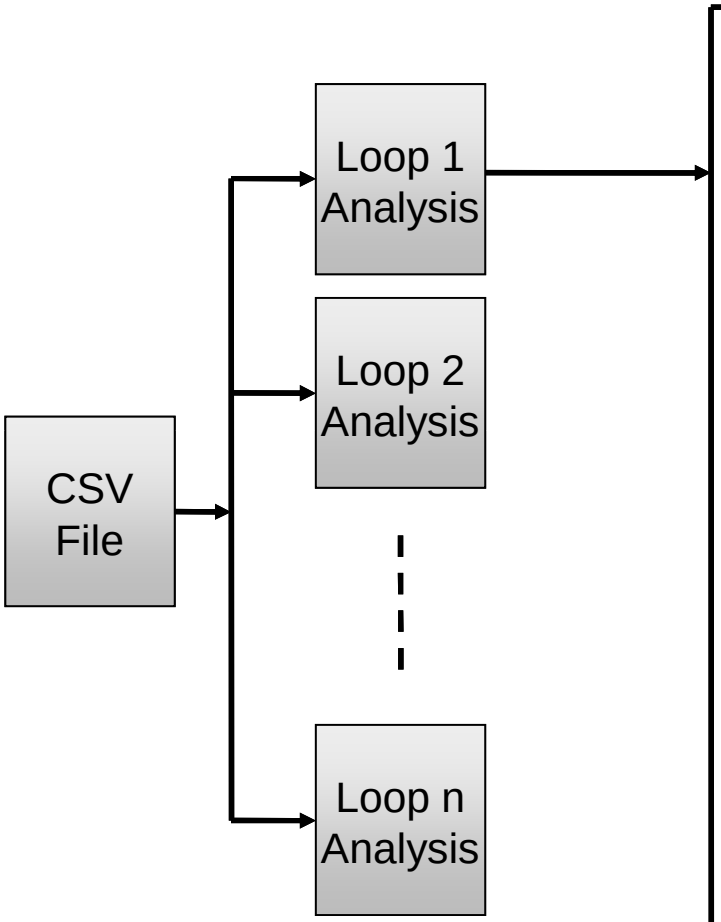
2) Binary to CSV

Binary is analyzed using MAQAO and a CSV file with results is produced

3) CSV to CTI entry

CSV file is loaded into CTI repository and saved in an entry

CTI MAQAO Integration



+ Number of instructions
+ Number of micro operations
+ Size (in bytes) of the loop
+ Number of XMM or YMM used registers
+ Does the loop fit in cache?
+ Number of bytes loaded and stored
+ Number of micro-operations per iteration for each execution port
+ Number of cycle per iteration for each execution port
+ Vectorization ratios for:
  - Loads
  - Store
  - Add/Sub
  - Mul
  - Others
+ Predictions if data fit in L1 cache:
  - Number of cycles
  - FLOP/cycle
  - Bytes loaded/cycle
  - Bytes stored/cycle
  - Number of cycle if fully vectorized

Overall Conclusions

- Studying a large application is difficult
  - Dividing the application into codelets
  - Using tools such as MAQAO and DECAN help understand the codelet's performance and behavior
  - Understanding the underlying architecture with MicroTools and the MDL help detect hardware bottlenecks
  - Analyzing all the data is only possible with automatic tools and infrastructures such as CTI

Acknowledgements

- Pablo Oliviera, UVSQ
- José Noudohouenou, UVSQ
- Franck Talbart, UVSQ
- William Jalby, CTO, UVSQ
- Jean-Thomas Acquaviva, Lead of the Performance Tools
- Bettina Krammer, Head of Software Tools, UVSQ
- Marie-Christine Sawley, Lab Director, Intel

ECR Contacts

- Address

  UVSQ, 45 Av. des Etats-Unis, Buffon building, 5th floor
  78 000 Versailles, France

- Web site: www.exascale-computing.eu

- Team

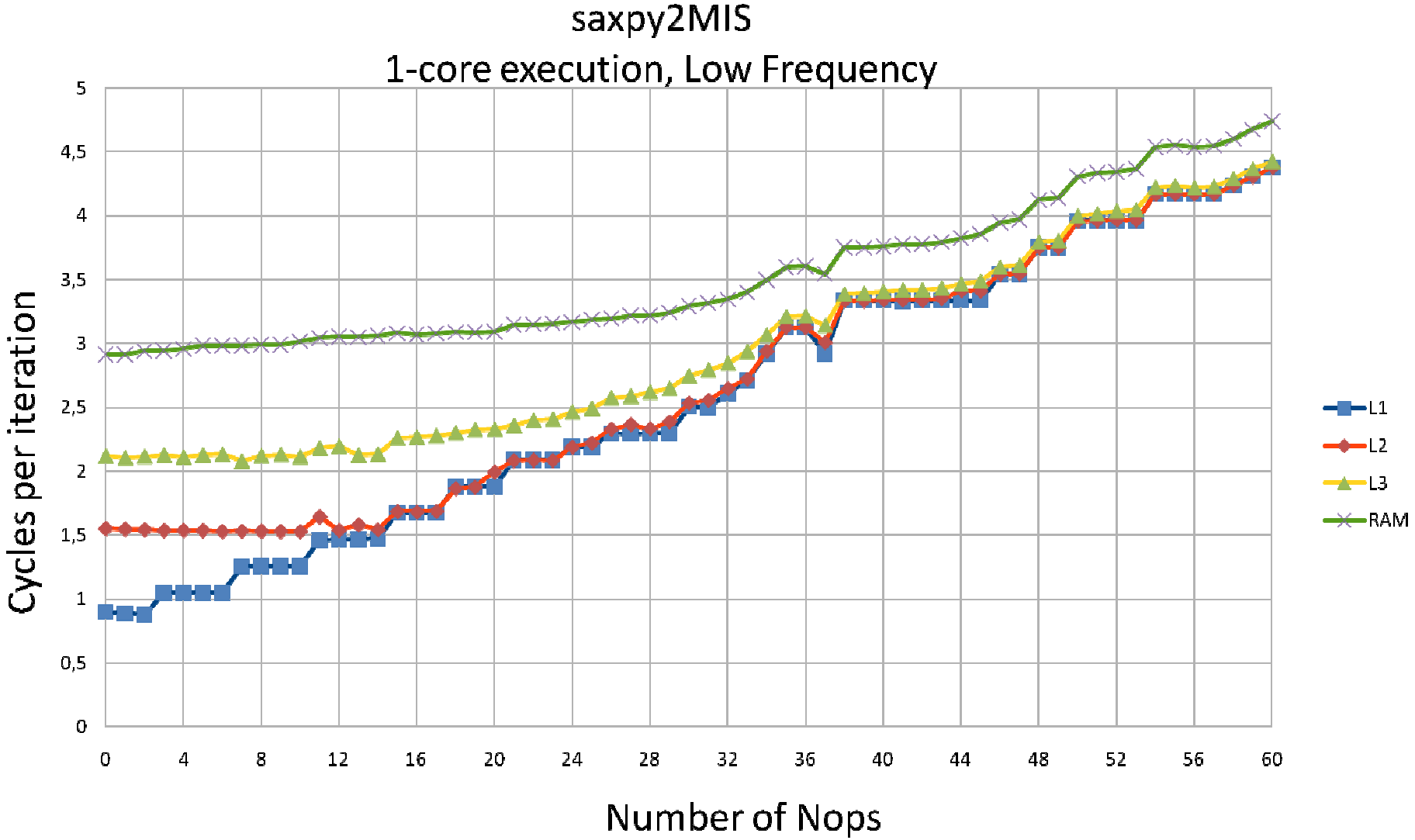
  William Jalby, CT , william.jalby@uvsq.fr

  Jean Christophe Beyler, Lead of Application Characterization, jean.christophe.beyler@intel.com
  Marie-Christine Sawley, Co-design, marie-christine.sawley@intel.com
  Bettina Krammer, Tools, bettina.krammer@uvsq.fr

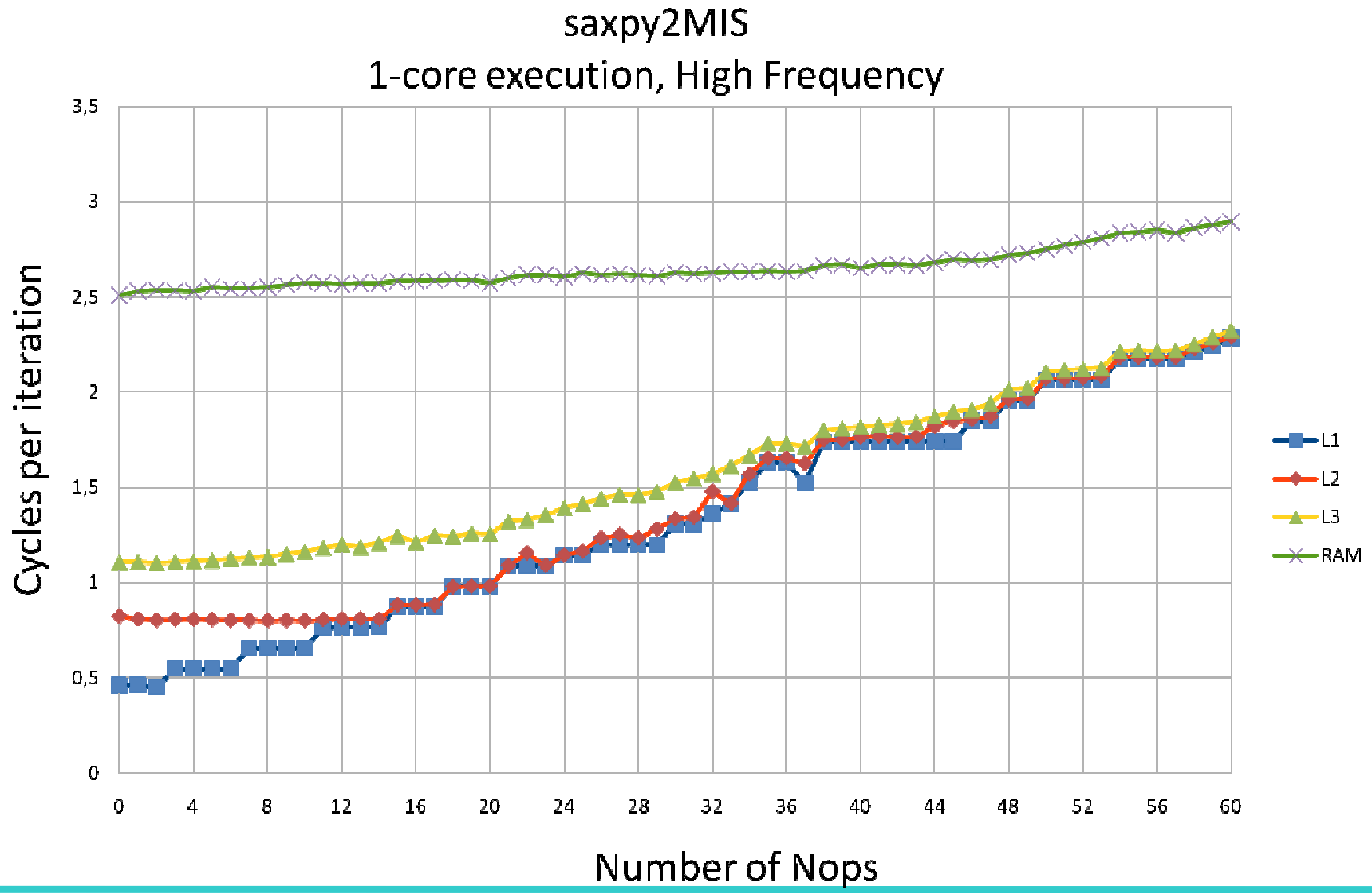
- Collaboration partners

Thank you

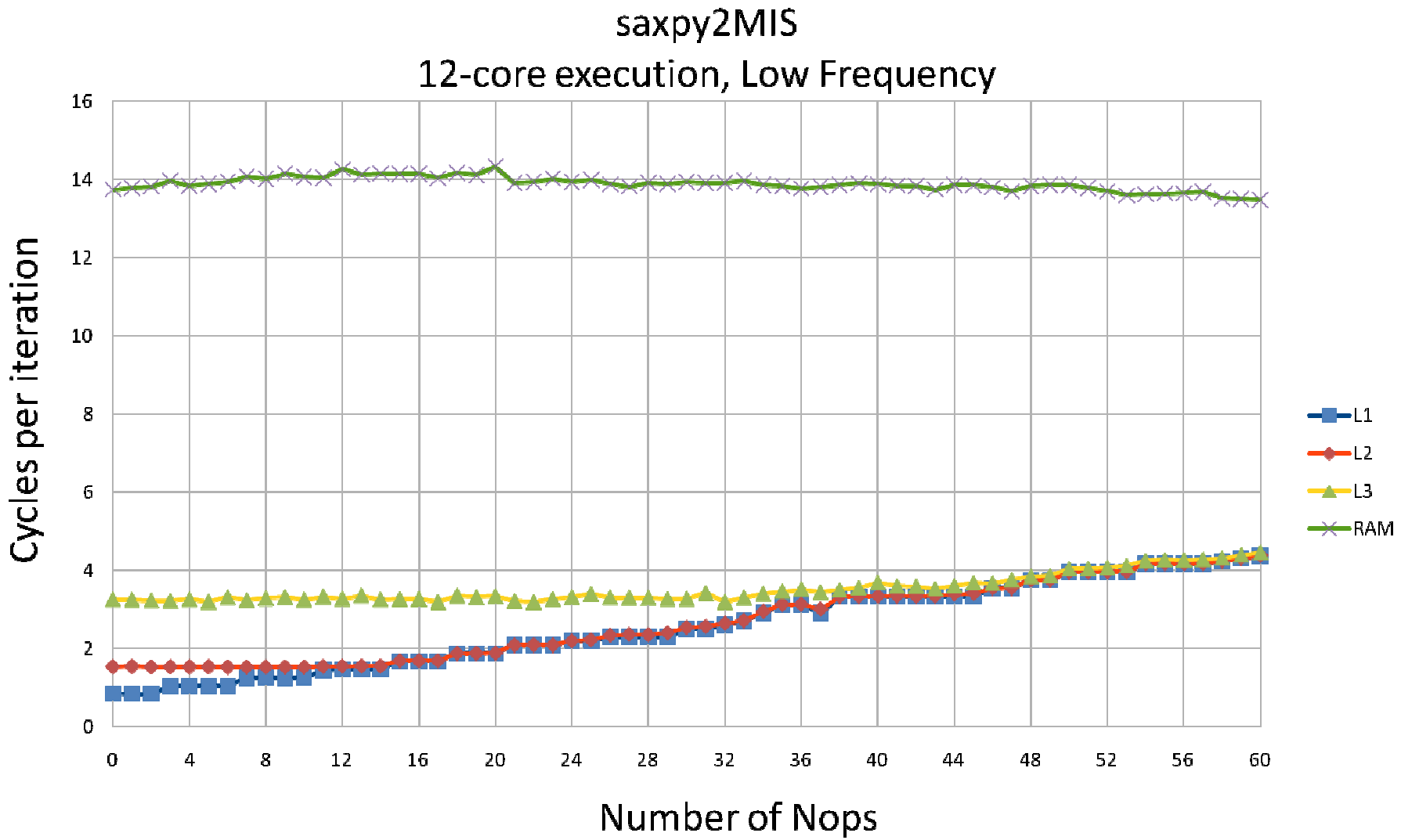
DECAN + MICROTOOLS



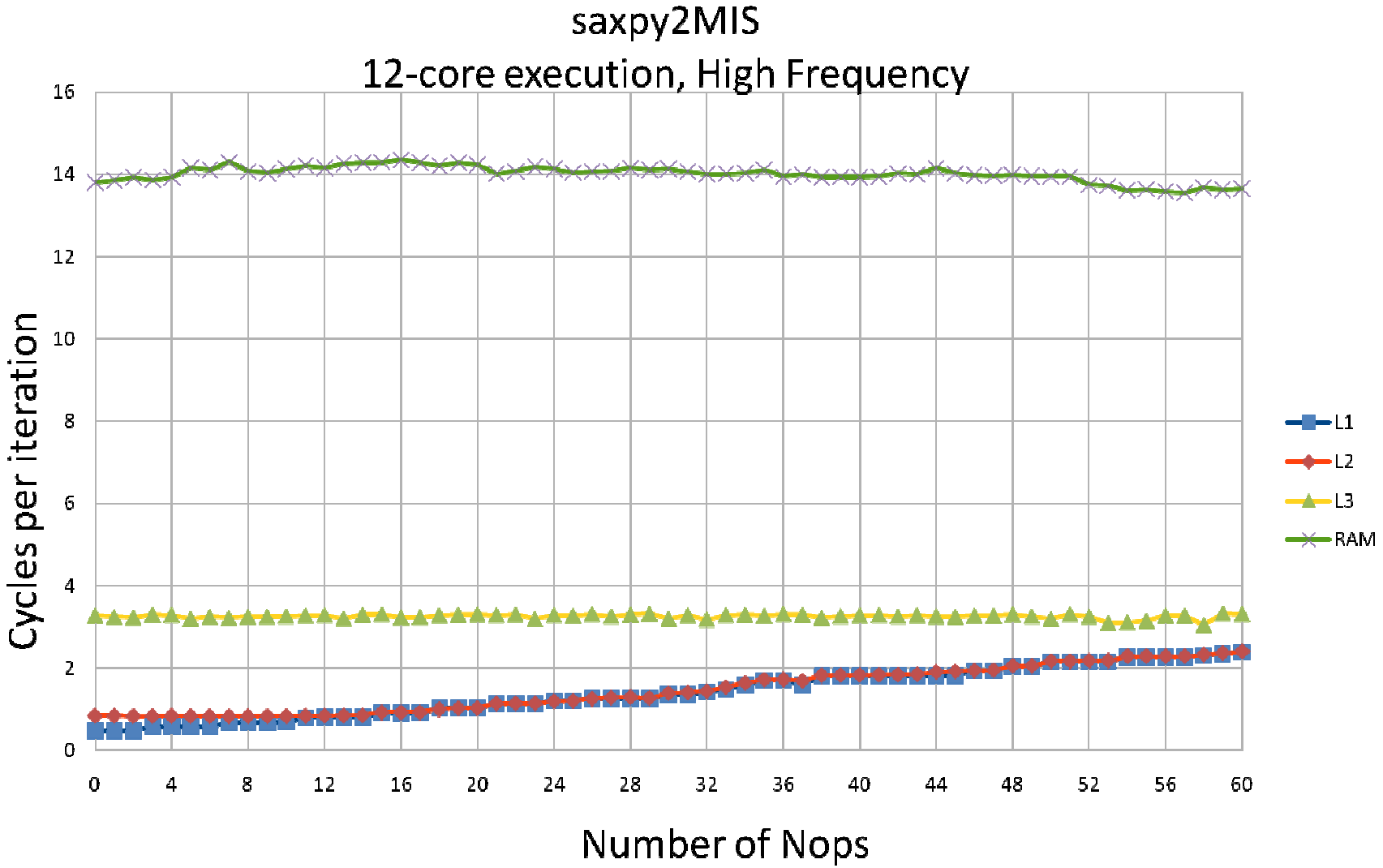
DECAN + MICROTOOLS



DECAN + MICROTOOLS



DECAN + MICROTOOLS



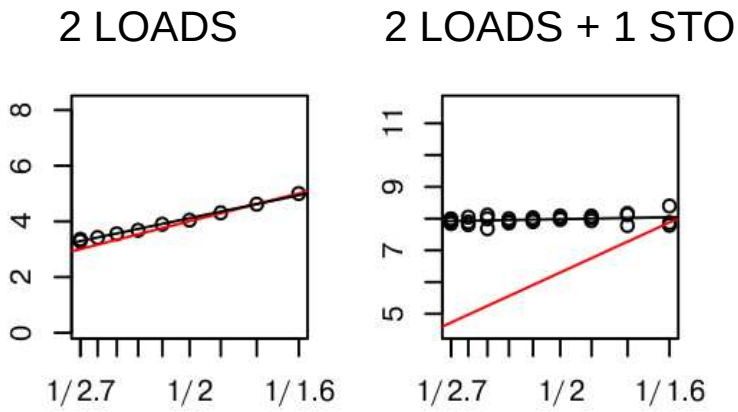
Handling Stores

- L3 Nehalem
  - Pure load patterns scale better with frequency

Handling Stores

- L3 Nehalem
  - Pure load patterns scale better with frequency

MOVSD

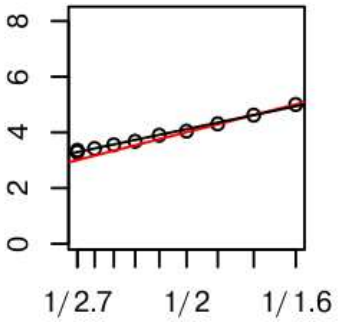


Handling Stores

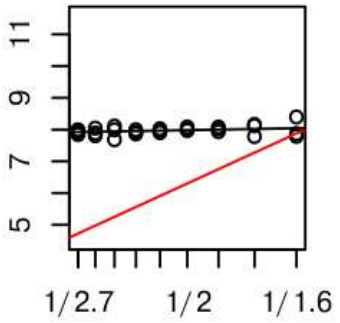
- L3 Nehalem
  - Pure load patterns scale better with frequency

MOVSD

2 LOADS

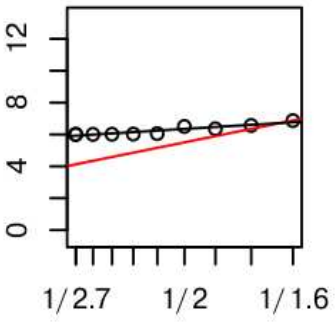


2 LOADS + 1 STORE

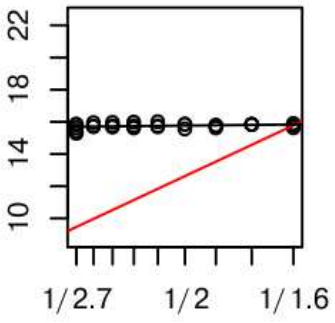


MOVAPS

2 LOADS



2 LOADS + 1 STORE



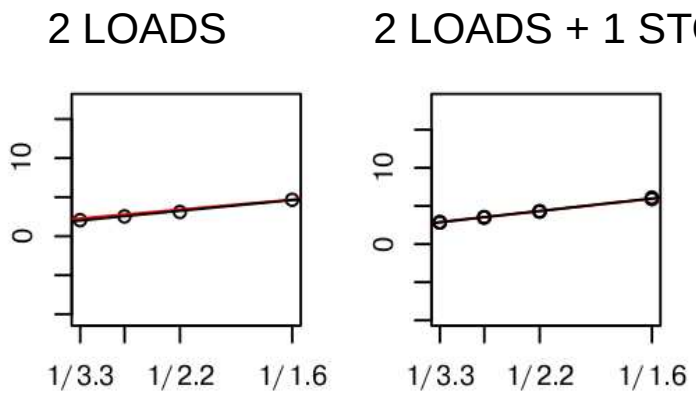
Handling Stores

- L3 Sandy Bridge cache scales much better
- Perfect scaling for all the store and load configurations

Handling Stores

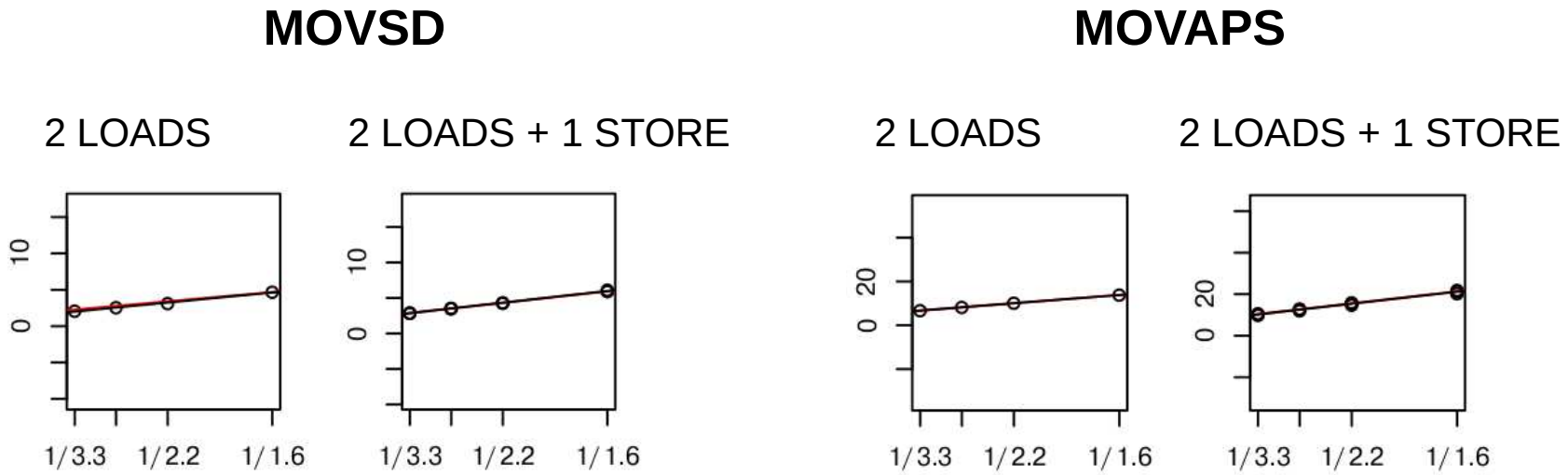
- L3 Sandy Bridge cache scales much better
- Perfect scaling for all the store and load configurations

MOVSD



Handling Stores

- L3 Sandy Bridge cache scales much better
- Perfect scaling for all the store and load configurations



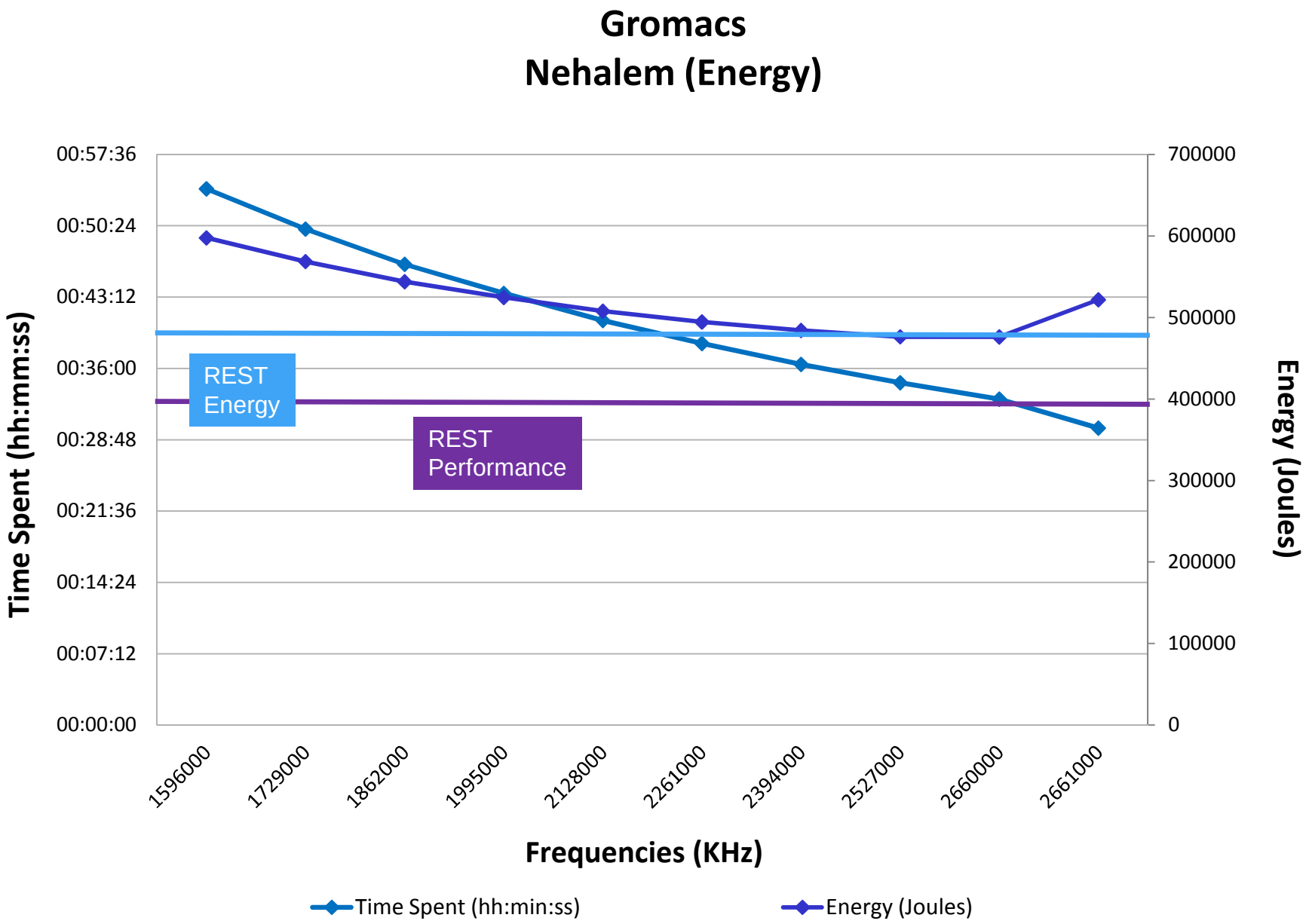
Energy

Results

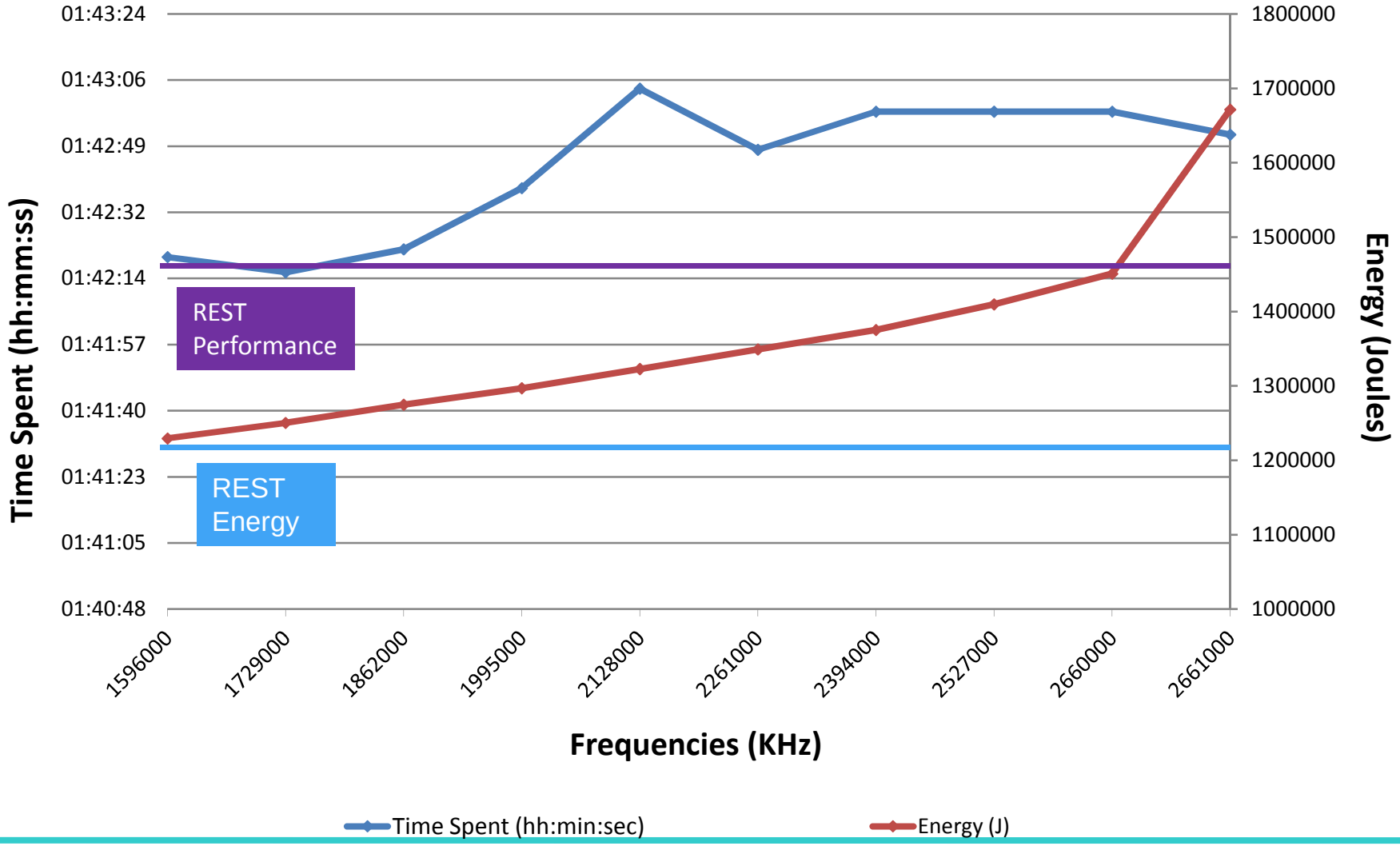
Used a dual-Nehalem, 6-cores each, nine possible frequencies

Used a Sandy Bridge Quad-core, sixteen frequencies

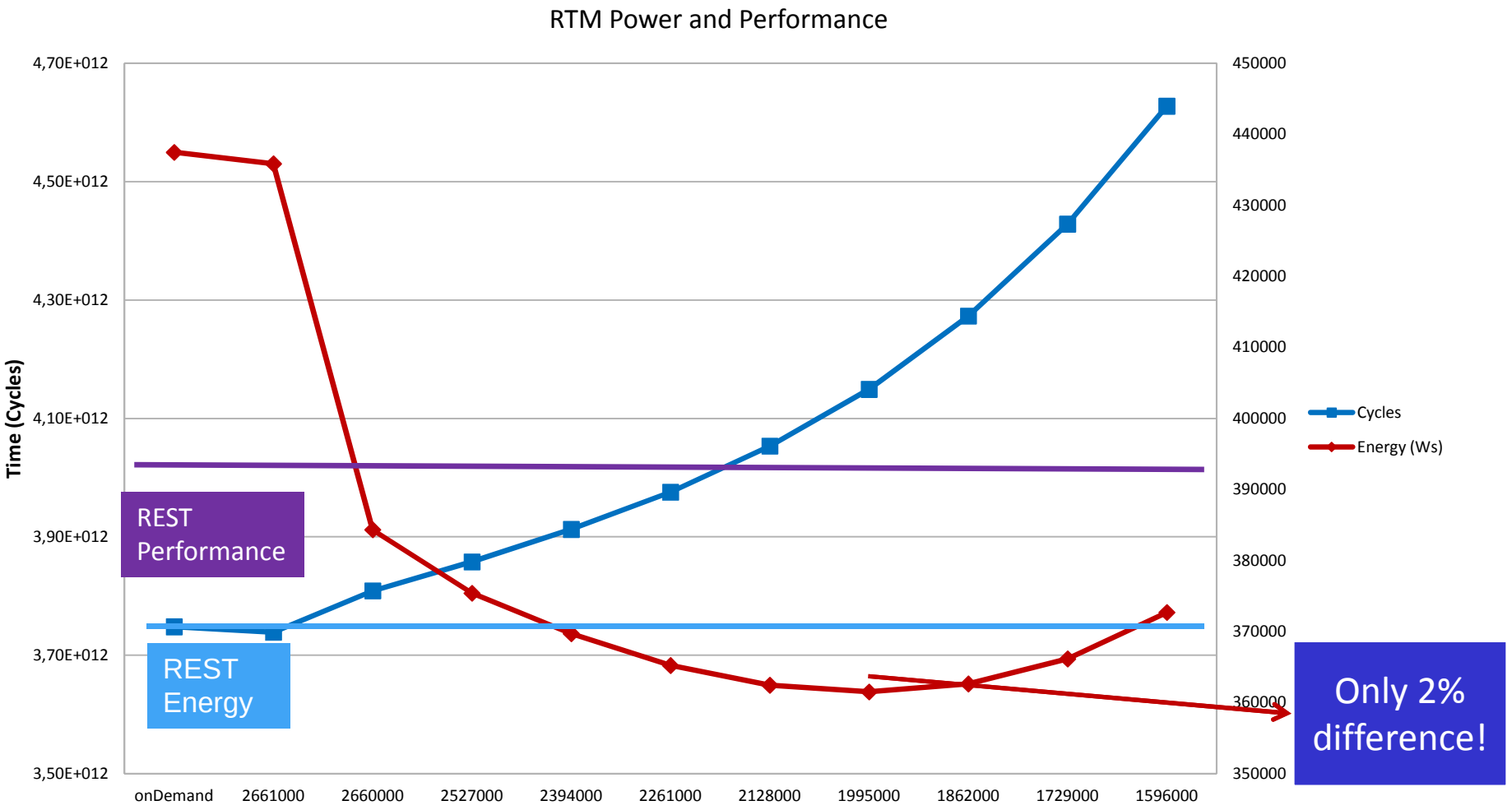
Compared with O3 execution, the Linux governors and static frequencies


Gromacs
Nehalem (Energy)
Time Spent (hh:mm:ss)
00:57:36
00:50:24
00:43:12
00:36:00
00:28:48
00:21:36
00:14:24
00:07:12
00:00:00
Energy (Joules)
700000
600000
500000
400000
300000
200000
100000
0
REST Energy
REST Performance
1596000
1729000
1862000
1995000
2128000
2261000
2394000
2527000
2660000
2661000
Frequencies (KHz)
Time Spent (hh:min:ss)
Energy (Joules)

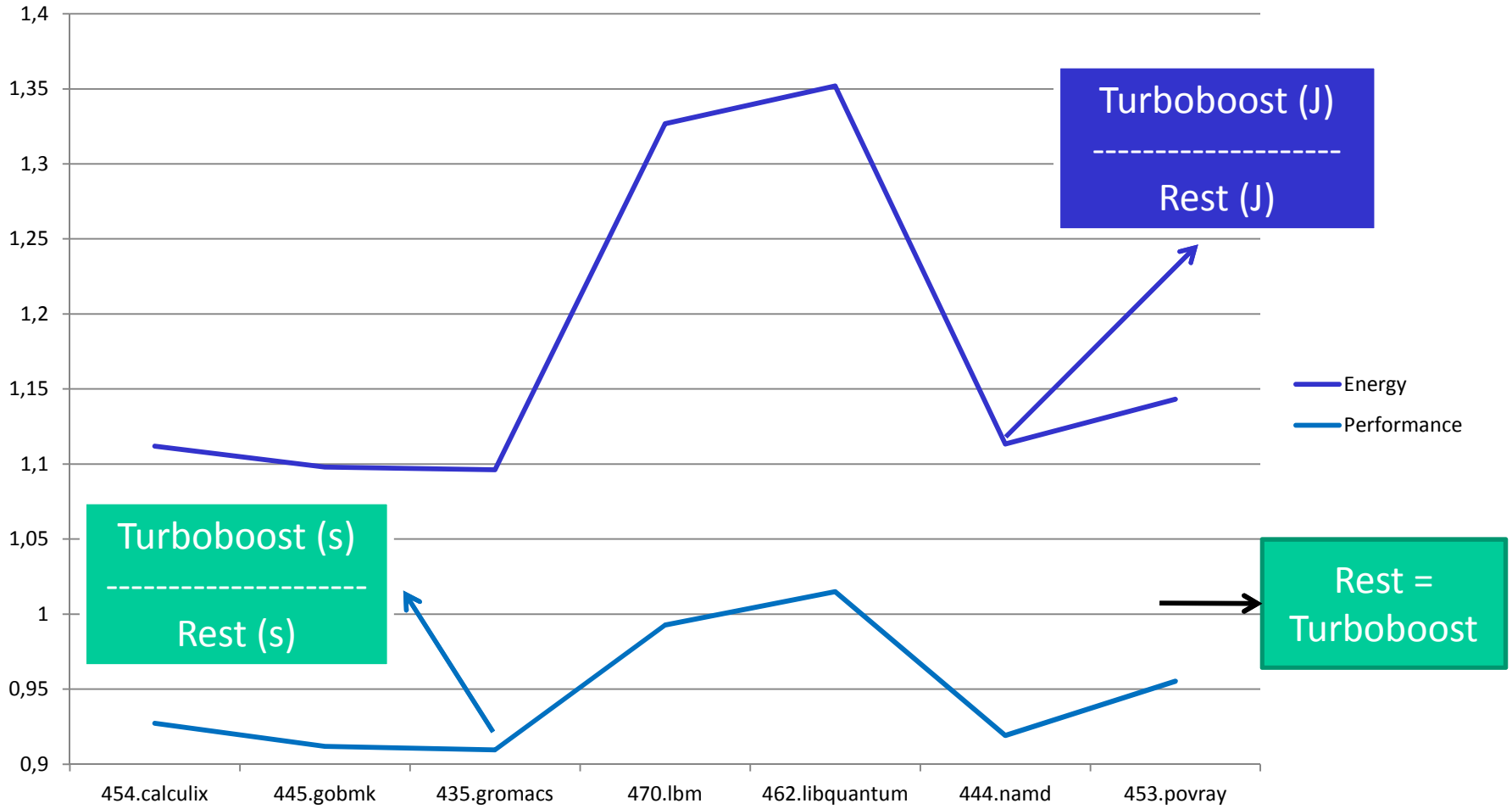
Libquantum
Nehalem Energy



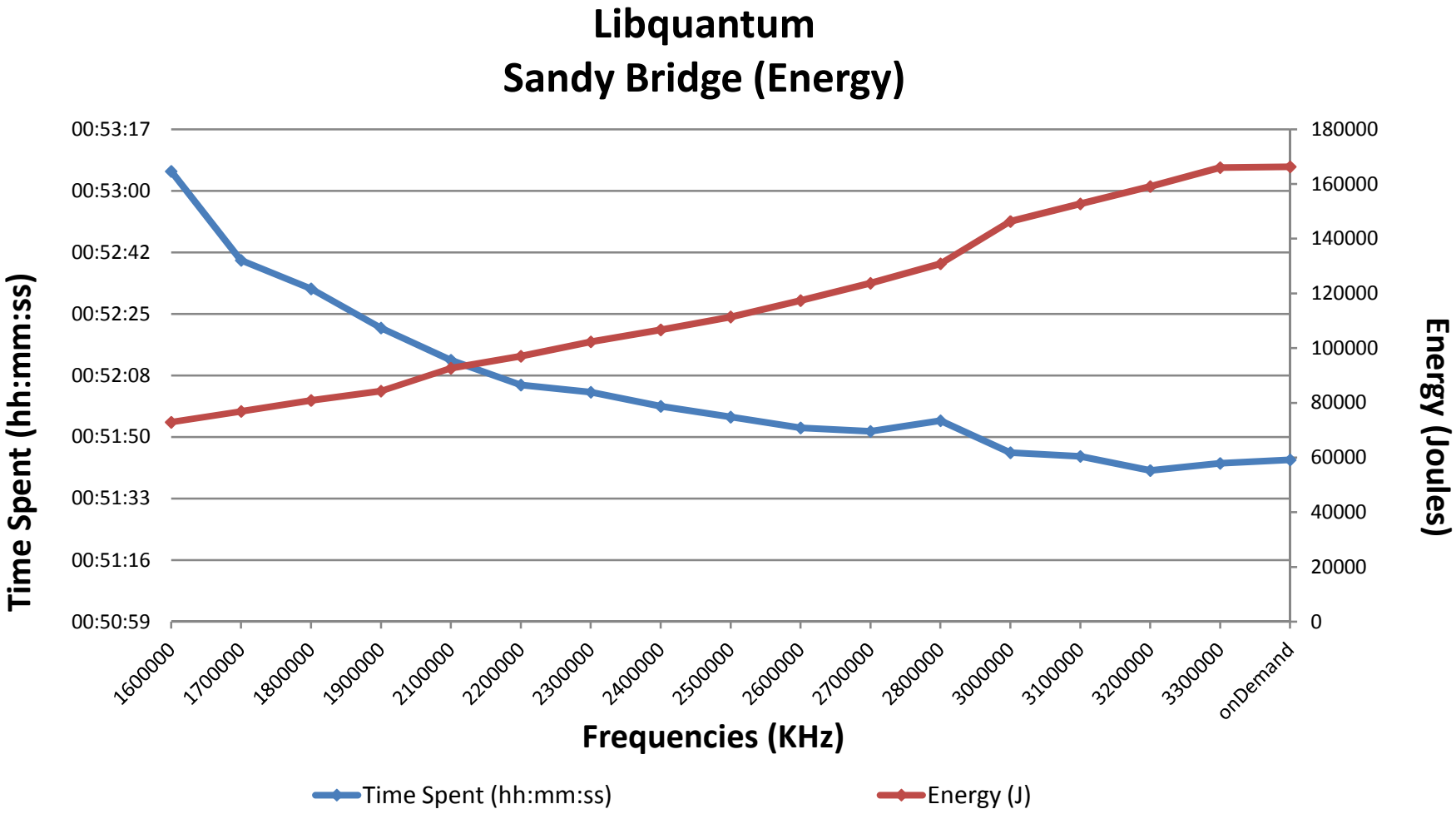
RTM on Nehalem



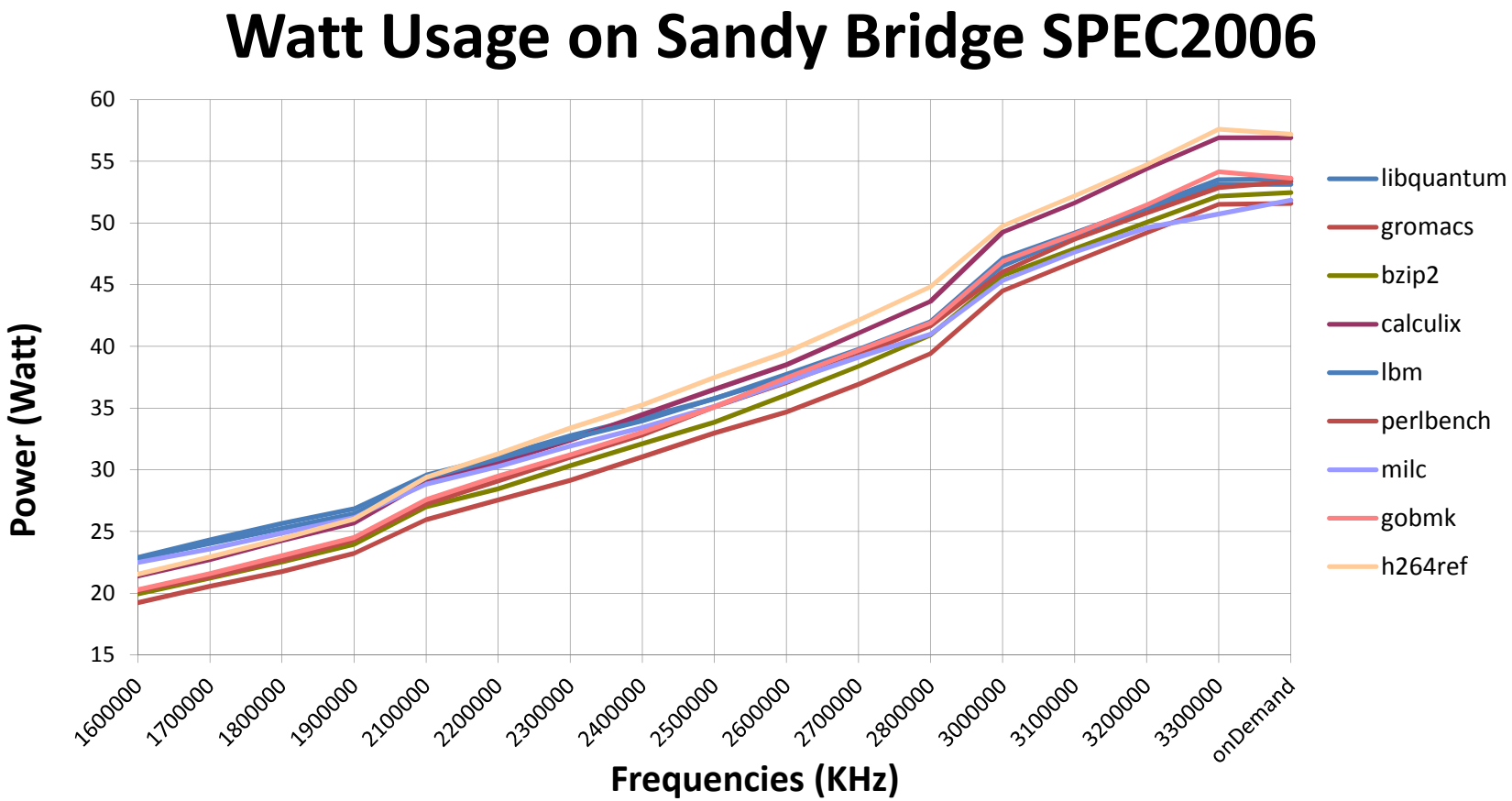
Ratio Turboboost on REST



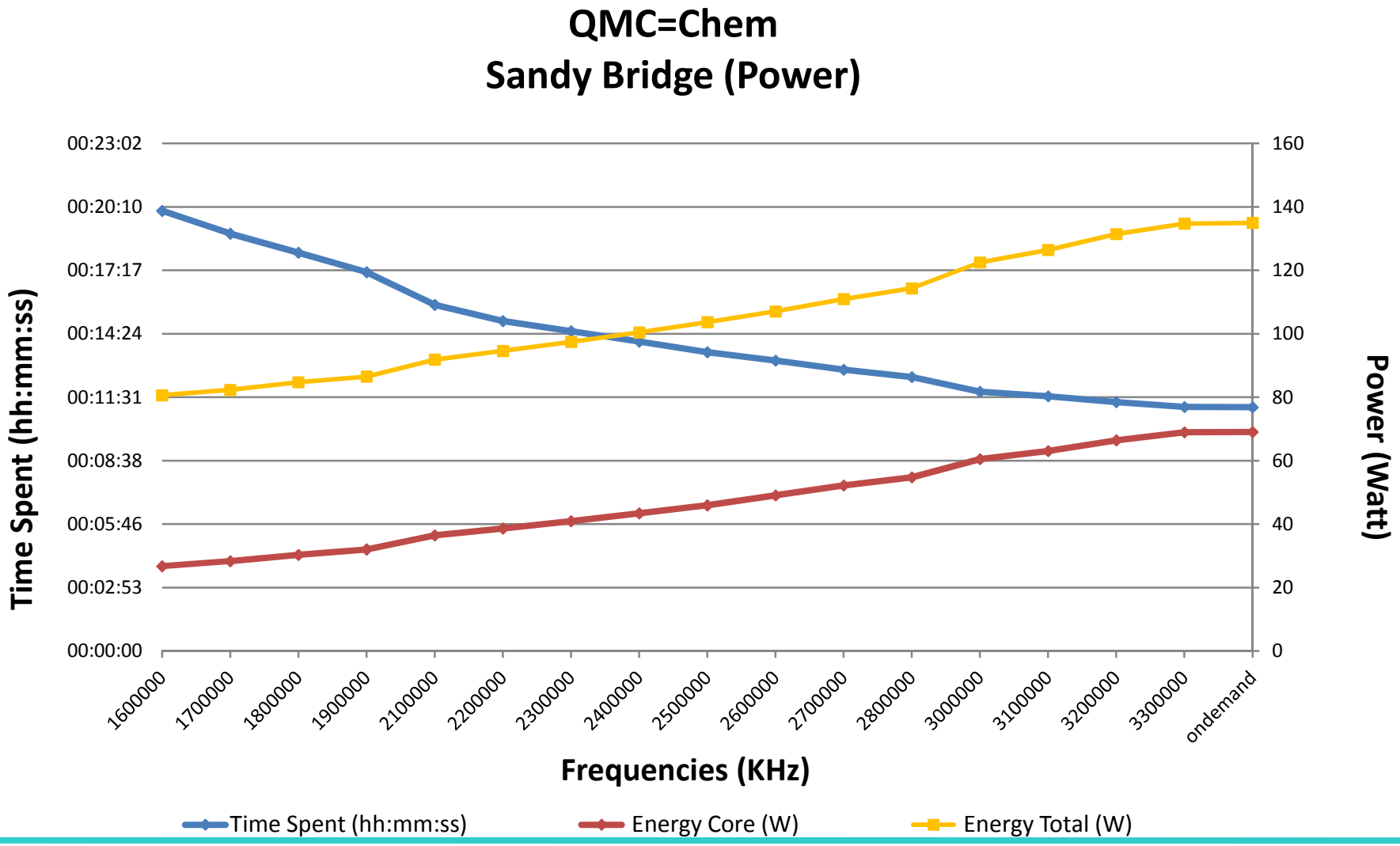
What about Sandy bridge?



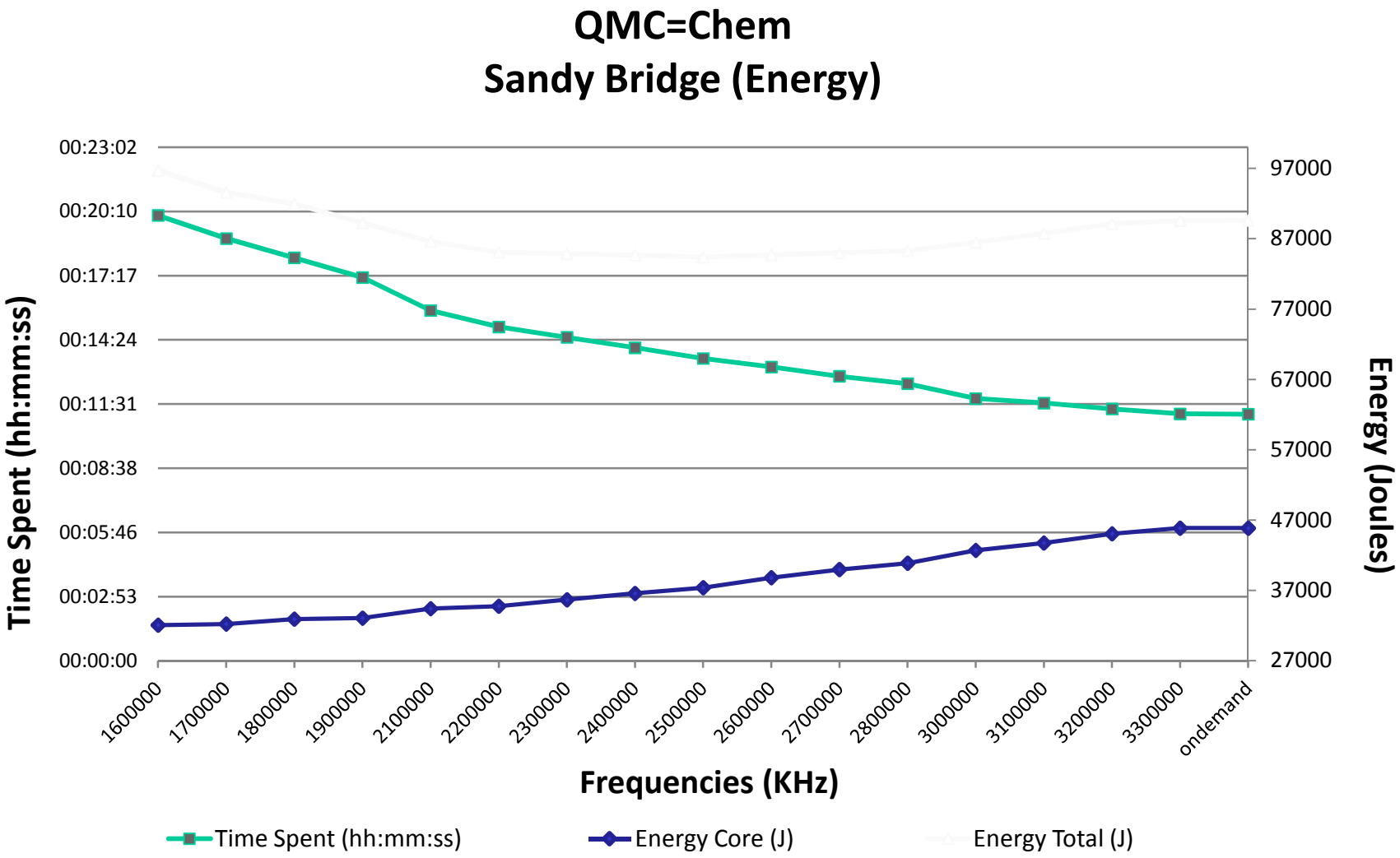
What about Watts?



Sandy Bridge's Hardware counters 1/2



Sandy Bridge's Hardware counters 2/2



UpDownbench profiled with Eprof (Hardware counter from PAPI)