Impact of sequential performance in parallel codes

Jesus Labarta (BSC)

EU H2020 Center of Excellence (CoE)

POP Webinar 6

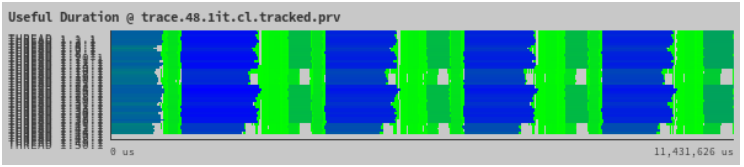
March 28th 2018

Agenda

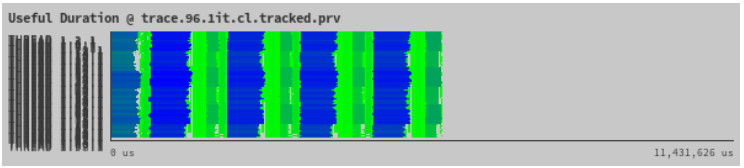
- Reviewing parallel efficiencies and scaling behavior
- Characterizing computational efficiencies
- Application Structure
  - Clustering
- Tracking the scaling behavior of computational phases
- What's next

Scaling behavior of parallel codes

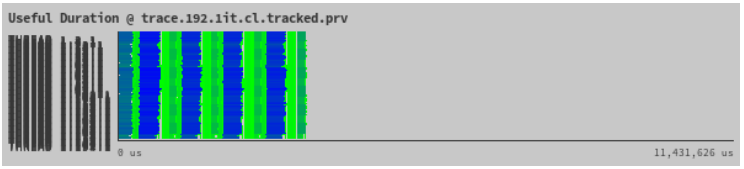
48



96

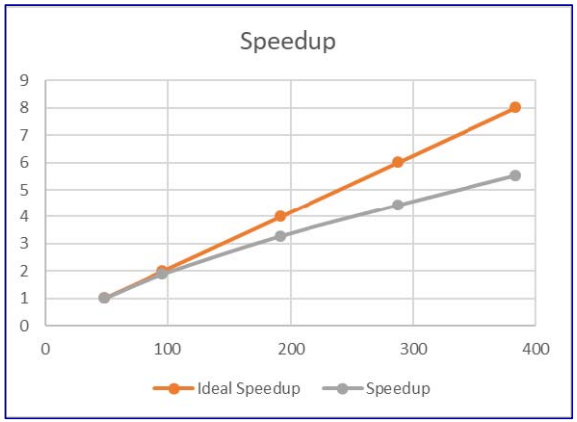


192

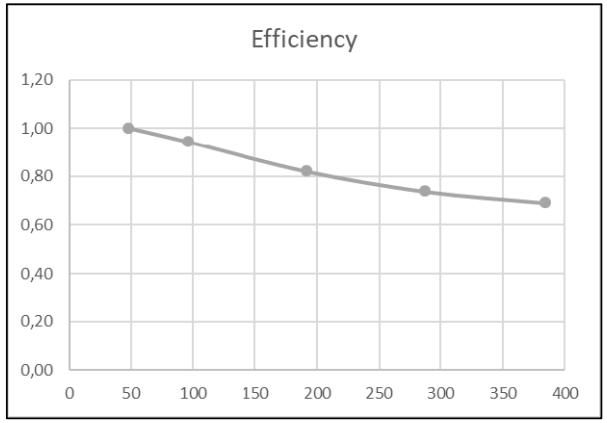


288

384

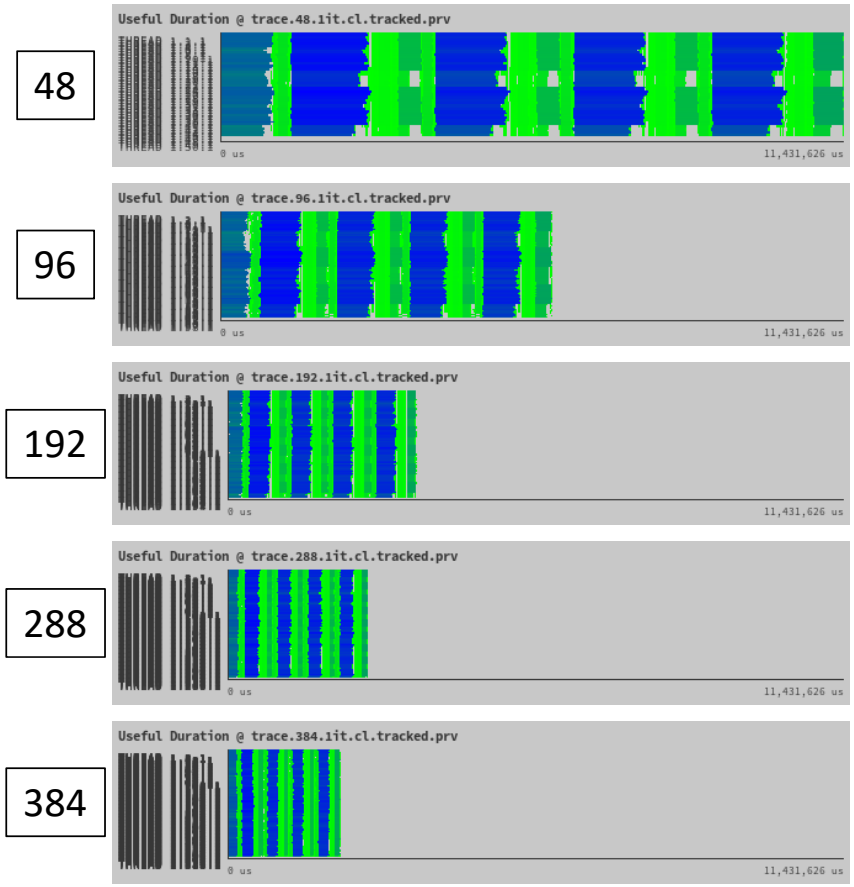


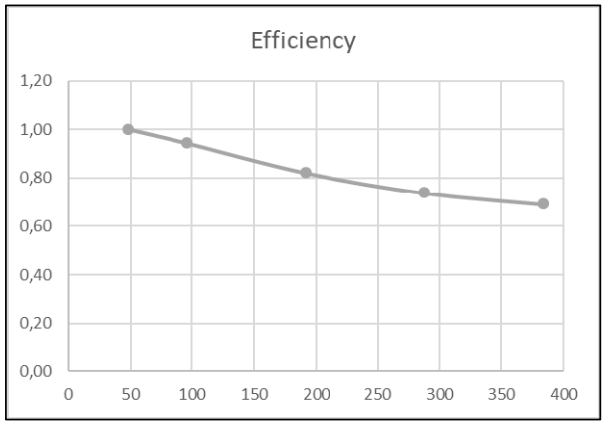
$$Sup = \frac{T_{ref}}{T_P}$$

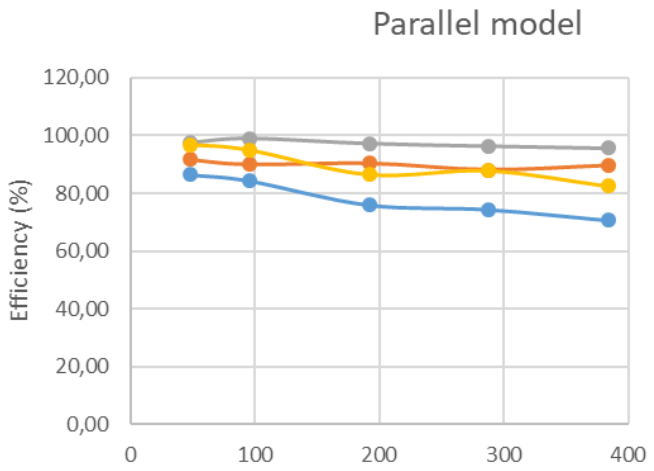


$$Eff = \frac{Sup}{P \ / \ ref}$$

Scaling behavior of parallel codes

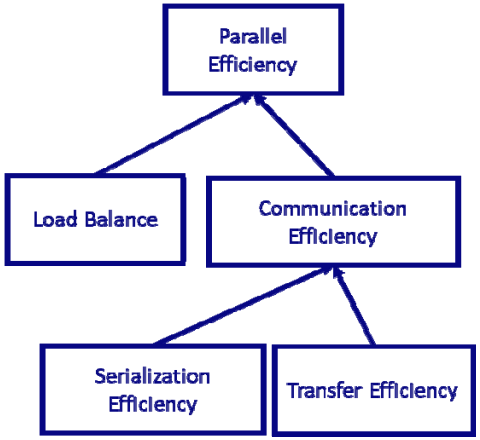






Subject of POP Webinar #3

$$ParEff = LB * Ser * Trf$$



Todays topic

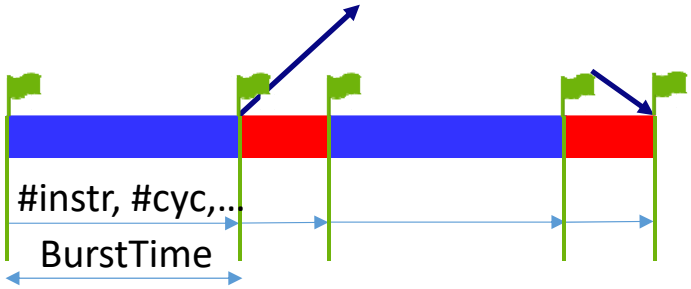
- Concurrency issues are not the only potential cause of efficiency/scaling degradations
- Sequential computation is also important !!
- How to quantify it in a high level way, with few numbers
- How to then dig down into details

Hardware counters

- Mechanisms to count hardware events
  - Program
    - PAPI_TOT_INS, …
  - Architecture
    - PAPI_L1_DCM, PAPI_L2_DCM, PAPI_L3_TCM, PAPI_BR_MSP, …
  - Performance
    - PAPI_TOT_CYC
  - Power
- API configure choice and read: PAPI
  - Typically a few at a time
  - Multiplexing

Reading hardware counters in Extrae

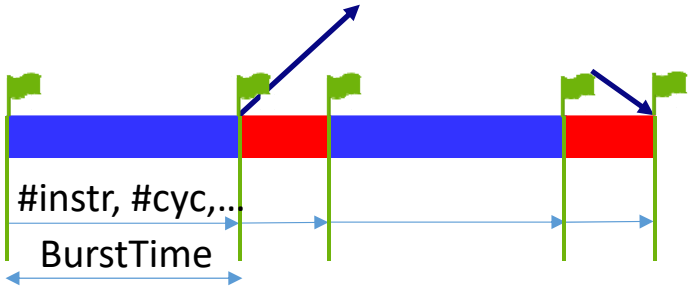
- Reads
  - At each runtime call
  - Emit count since previous read



- xml
  - Specification of counters to measure (groups)
    - Important counters: instructions, cycles
  - Specification of coarse grain multiplexing between groups
    - Time based
    - Based on collectives count

Reading hardware counters in Extrae

- Reads
  - At each runtime call
  - Emit count since previous read



- xml
  - Specification of counters to measure (groups)
    - Important counters: instructions, cycles
  - Specification of coarse grain multiplexing between groups
    - Time based
    - Based on collectives count

job.sh

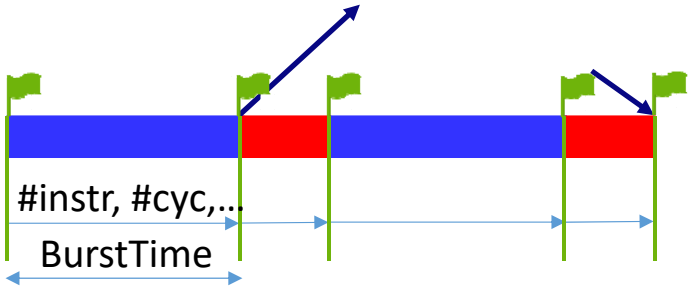
```
#!/bin/bash
mpirun -np 8  trace.sh ../my_production_binary
```

trace.sh

```
#!/bin/bash
module load Extrae
export EXTRAE_CONFIG_FILE=extrae.xml
export LD_PRELOAD=${EXTRAE_HOME}/lib/libmpitrace.so
$*
```

Reading hardware counters in Extrae

- Reads
  - At each runtime call
  - Emit count since previous read



- xml
  - Specification of counters to measure (groups)
    - Important counters: instructions, cycles
  - Specification of coarse grain multiplexing between groups
    - Time based
    - Based on collectives count

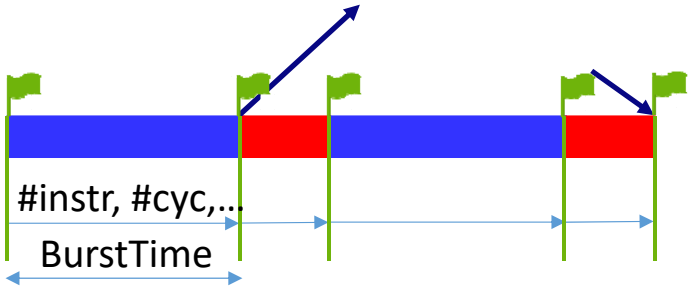
extrae.xml

```
...
<counters enabled="yes">
  <cpu enabled="yes" starting-set-distribution="1">
    <set enabled="yes" domain="all" changeat-time="0">
      PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L1_DCM,PAPI_L2_DCM,
      PAPI_L3_TCM,PAPI_FP_INS,PAPI_BR_MSP
    </set>
    <set enabled="yes" domain="all" changeat-time="0">
      PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_LD_INS,PAPI_SR_INS,
      PAPI_BR_UCN,PAPI_BR_CN,PAPI_VEC_SP,RESOURCE_STALLS
    </set>
  </cpu>
...
```

Hardware counter metrics

- Base and derived metrics

- Metrics
  - Program
    - **Instruction count**, Instruction mix
  - Architecture
    - Misses, Miss ratios, miss prediction ratios, …
  - Performance
    - **Instructions per cycle (IPC)**
    - **Frequency**
  - Models

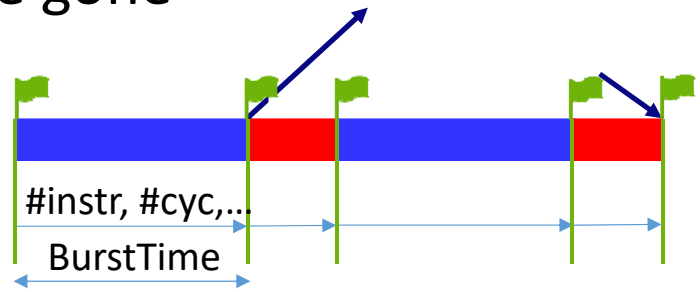


$$IPC = \frac{\#\ instr}{\#\ cyc}$$

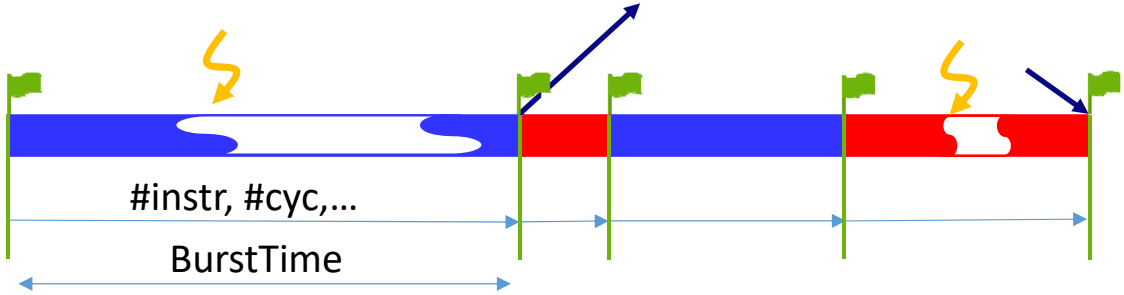
$$Freq = \frac{\#\ cyc}{BurstTime}$$

A comment on frequency

- Good old times where frequency was known are gone
  - Turbo
  - DVFS
  - Power capping, governors
  - Device variability



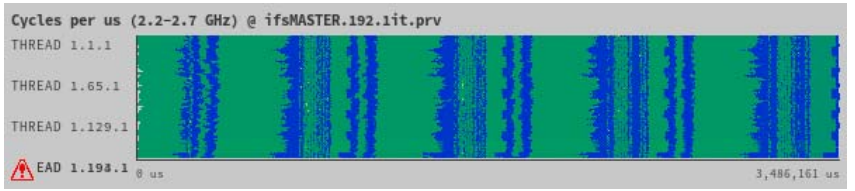
- PAPI counters virtualized
  - OS activity, noise, yields



$$Freq = \frac{\#\ cyc}{BurstTime}$$

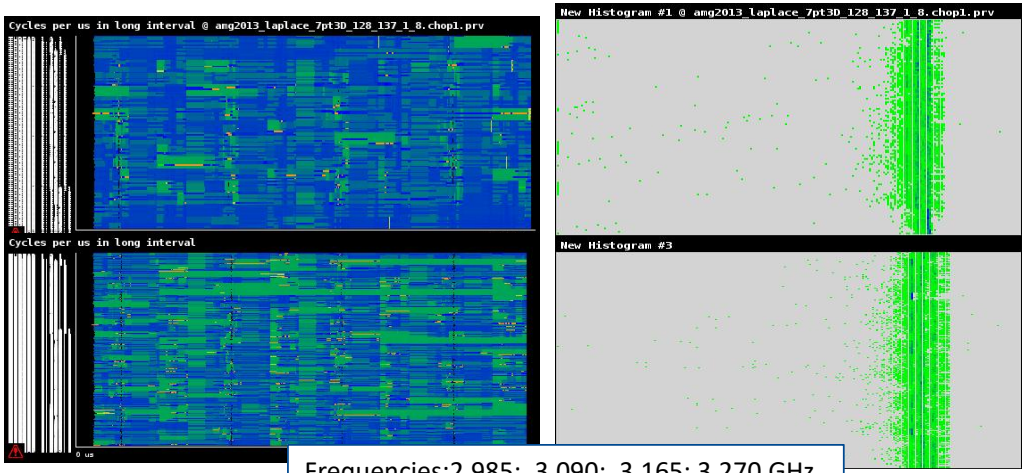
A comment on frequency

- Good old times where frequency was known are gone
  - Turbo
  - DVFS
  - Power capping, governors
  - Device variability



Towards unpredictable core performance

- PAPI counters virtualized
  - OS activity, noise, yields



Frequencies:2.985; 3.090; 3.165; 3.270 GHz

Characterizing sequential performance

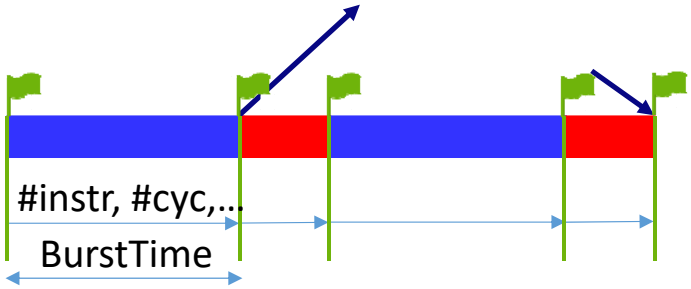
- Computation efficiency model
  - Performance scaling factor over reference case
    - 0 .. 1 .. X
  - Multiplicative model

$$CompEff = Ieff * IPCeff * Feff$$

- Efficiency factors
  - Instruction scaling efficiency
    - Total amount of work. Code replication?
  - IPC scaling efficiency
    - How fast are instructions executed by architecture
  - Frequency efficiency
    - Frequency changes with load

Characterizing sequential performance

- Computation efficiency model
  - Performance scaling factor over reference case
    - 0 .. 1 .. X
  - Multiplicative model



- Efficiency factors
  - Instruction scaling efficiency
    - Total amount of work. Code replication?
  - IPC scaling efficiency
    - How fast are instructions executed by architecture
  - Frequency efficiency
    - Frequency changes with load

$$InstrEff = \frac{\#\ instr_{ref}}{\#\ instr_{P}}$$

$$IPCEff = \frac{IPC_{P}}{IPC_{ref}}$$

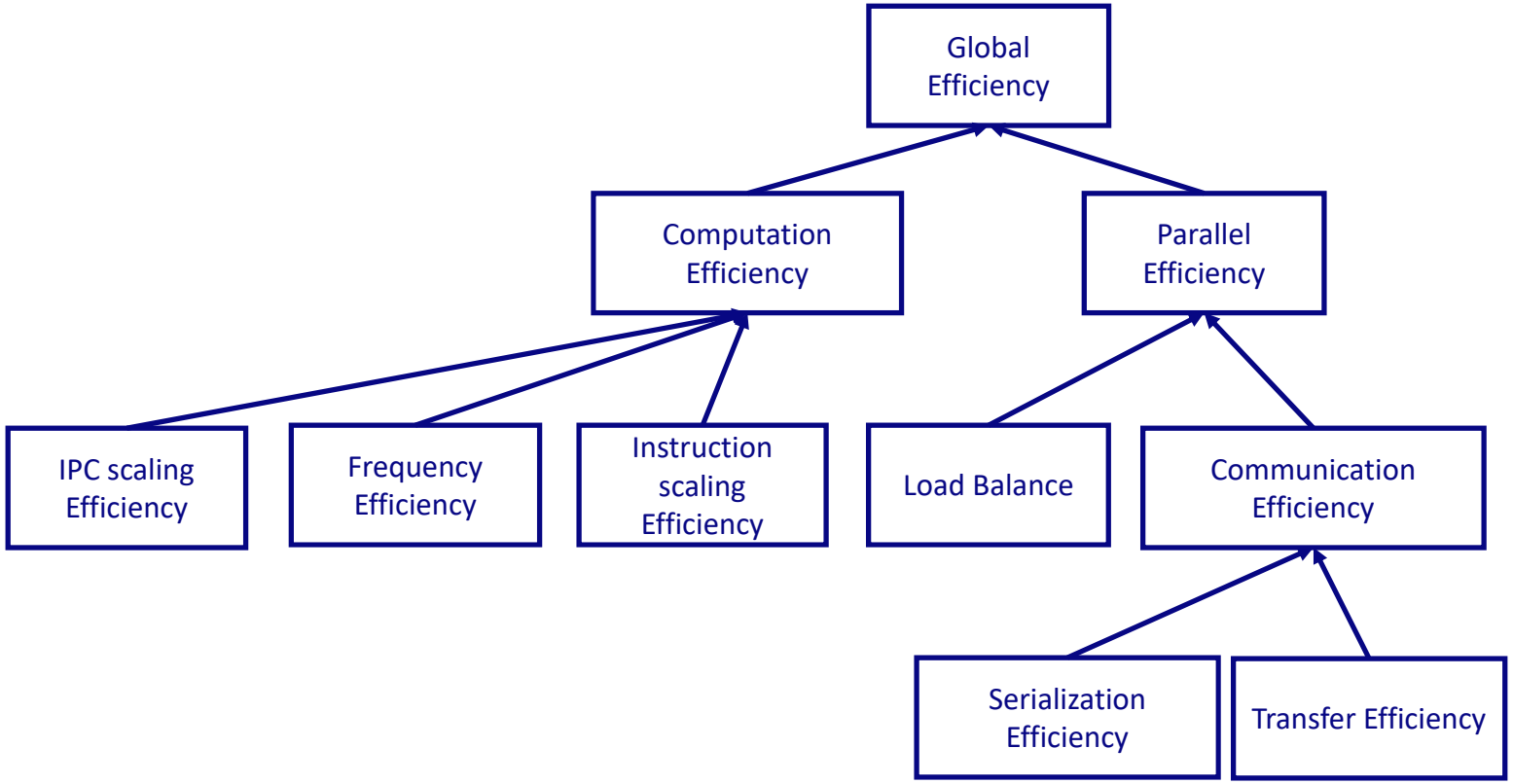
$$Feff = \frac{F_{P}}{F_{ref}}$$

Reference core count

In user level code "Useful"

Application characterization

Efficiencies: ~ (0,1]
Multiplicative model




```
$ model_factors.py –sc strong -t 8.prv 16.prv 32.prv 64.prv
```

Examples

	2	4	8	16
Parallel Efficiency	0.983	0.944	0.898	0.848
Load Balance	0.987	0.969	0.910	0.918
Serialization efficiency	0.998	0.977	0.994	0.940
Transfer Efficiency	0.998	0.997	0.993	0.983
Computation Efficiency	1.000	0.959	0.868	0.695
IPC scalability	1.000	0.993	0.959	0.842
Instruction scalability	1.000	0.972	0.939	0.908
Frequency scalability	1.000	0.993	0.964	0.910
Global efficiency	0.983	0.905	0.780	0.589

	8	16	32	40
Parallel Efficiency	0.377	0.348	0.222	0.181
Load Balance	0.382	0.360	0.233	0.189
Serialization efficiency	0.981	0.967	0.957	0.959
Transfer Efficiency	1.000	1.000	0.999	0.999
Computation Efficiency	1.000	0.840	0.796	0.774
IPC scalability	1.000	0.944	0.894	0.870
Instruction scalability	1.000	1.000	1.000	0.999
Frequency scalability	1.000	0.890	0.890	0.890
Global efficiency	0.377	0.292	0.177	0.141

	2	4	8
Parallel Efficiency	0.985	0.914	0.931
Load Balance	0.985	0.914	0.939
Serialization efficiency	1.000	1.000	0.911
Transfer Efficiency	1.000	1.000	1.088
Computation Efficiency	1.000	0.814	0.633
IPC scalability	1.000	0.961	0.594
Instruction scalability	1.000	0.873	1.106
Frequency scalability	1.000	0.970	0.964
Global efficiency	0.985	0.744	0.590

	32	48	64	96	128	256
Parallel Efficiency	0.917	0.906	0.887	0.847	0.864	0.790
Load Balance	0.946	0.925	0.934	0.858	0.871	0.813
Serialization efficiency	0.970	0.980	0.951	0.987	0.994	0.976
Transfer Efficiency	1.000	1.000	1.000	0.999	0.999	0.995
Computation Efficiency	1.000	1.025	1.026	1.036	1.012	0.956
IPC scalability	1.000	1.013	1.013	1.013	1.004	0.982
Instruction scalability	1.000	1.013	1.020	1.019	1.009	0.977
Frequency scalability						
Global efficiency	0.917	0.928	0.911	0.877	0.874	0.755

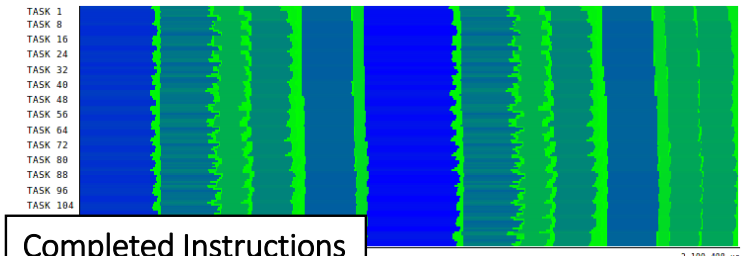
	48	96	192	288	384
Parallel Efficiency	0.865	0.843	0.760	0.744	0.707
Load Balance	0.917	0.900	0.904	0.880	0.896
Serialization efficiency	0.975	0.989	0.972	0.963	0.956
Transfer Efficiency	0.967	0.948	0.866	0.878	0.826
Computation Efficiency	1.000	0.966	0.932	0.856	0.843
IPC scalability	1.000	0.974	0.955	0.896	0.891
Instruction scalability	1.000	0.993	0.976	0.950	0.943
Frequency scalability	1.000	0.999	1.000	1.006	1.003
Global efficiency	0.865	0.815	0.709	0.637	0.596

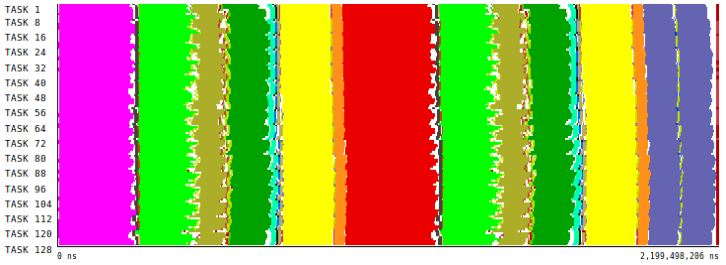
Coloring	1.000	0.850	0.849	0.750	0.749	0.650	0.649	0.000

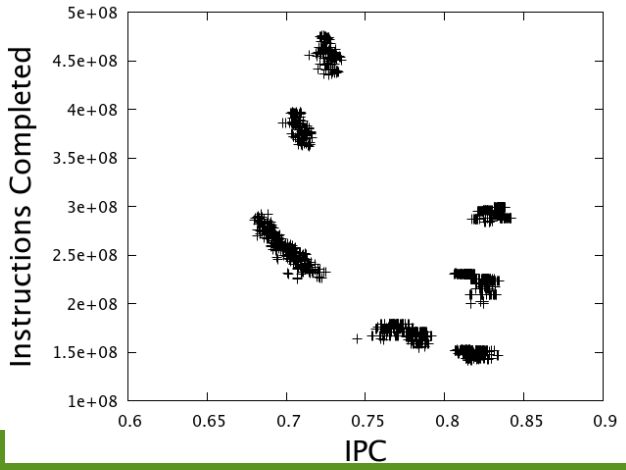
Agenda

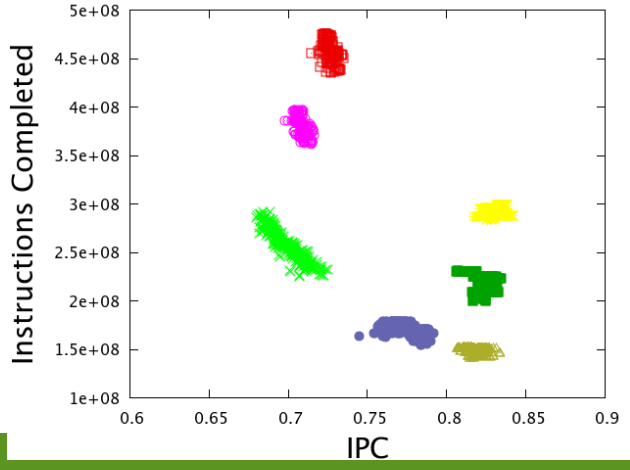
- Reviewing parallel efficiencies and scaling behavior
- Characterizing computational efficiencies
- Application Structure
  - Clustering
- Tracking the scaling behavior of computational phases
- What's next

Clustering → fine grain structure





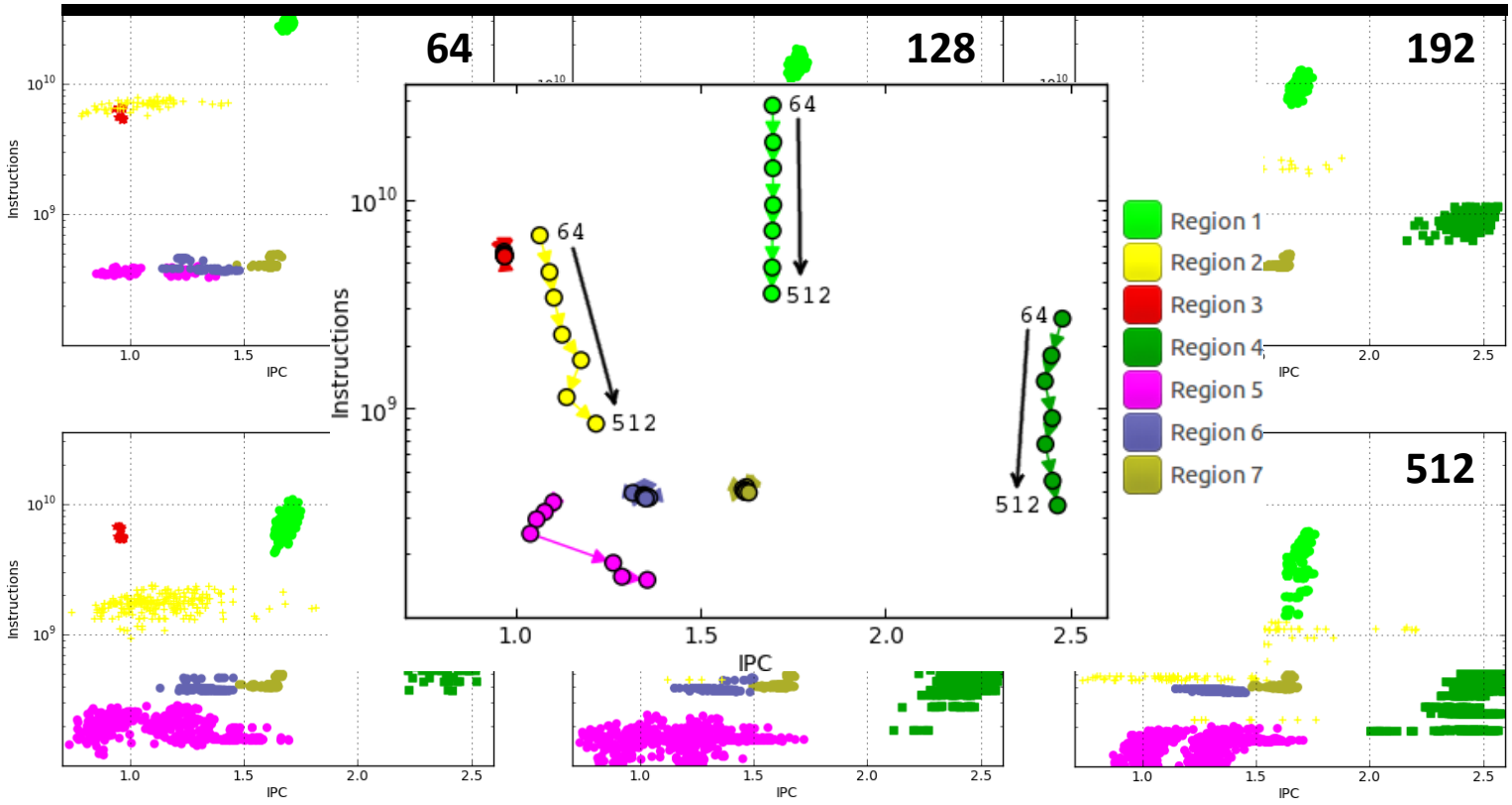




J. Gonzalez et al, "Automatic Detection of Parallel Applications Computation" Phases. (IPDPS 2009)

Tracking structural evolution

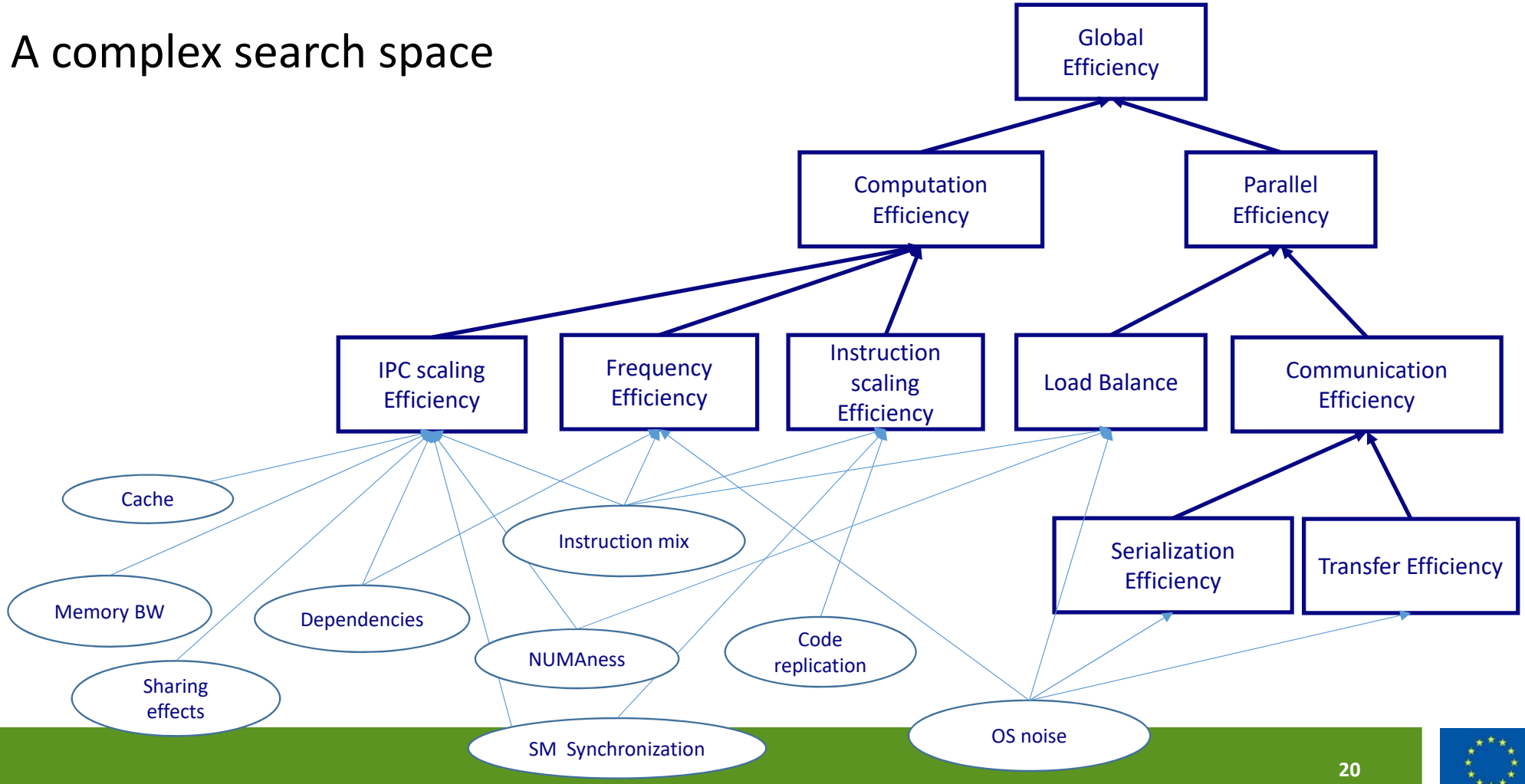
- Frame sequence: clustered scatterplot as core counts increases



G.Llort et all, "On the Usefulness of Object Tracking Techniques in Performance Analysis", SC 2013

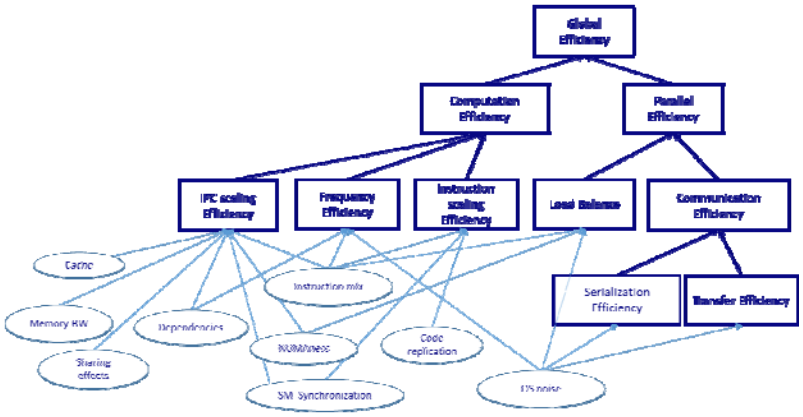
Further analysis

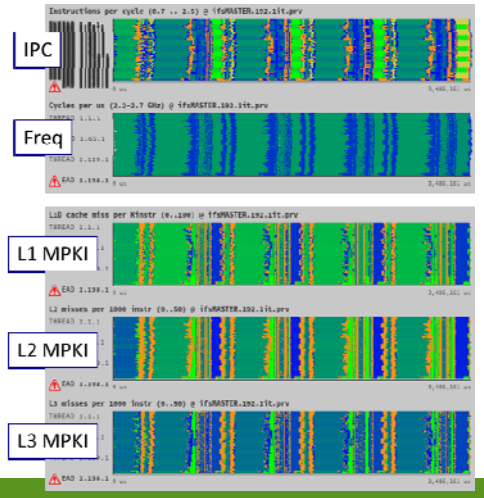
- A complex search space

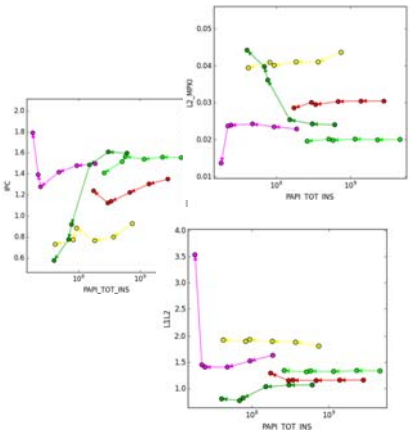


Further analysis

- A complex search space …
  - Can always dig down into detailed analysis of traces and source code
- … but would recommend progressive top down approach
  - **Track the evolution of different metrics**
  - Further analytics techniques

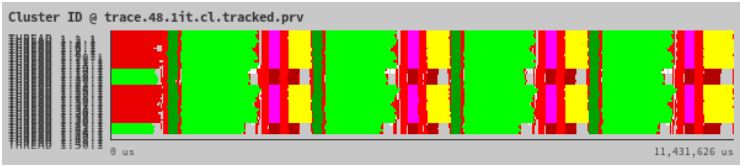




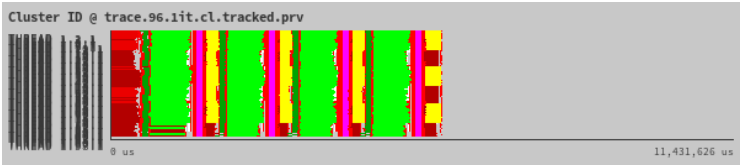


Example: MPI strong scaling

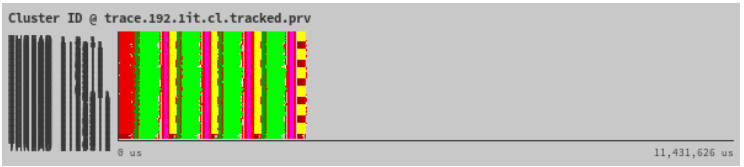
48



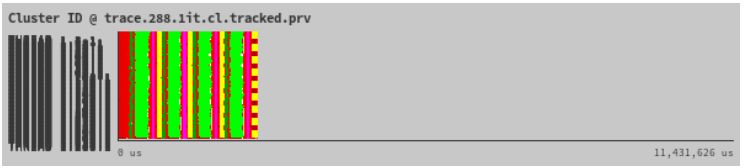
96



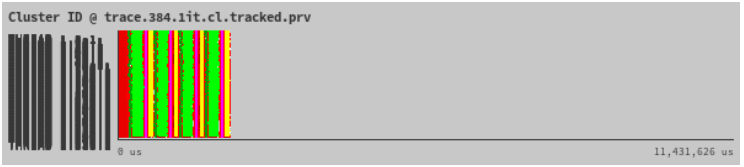
192

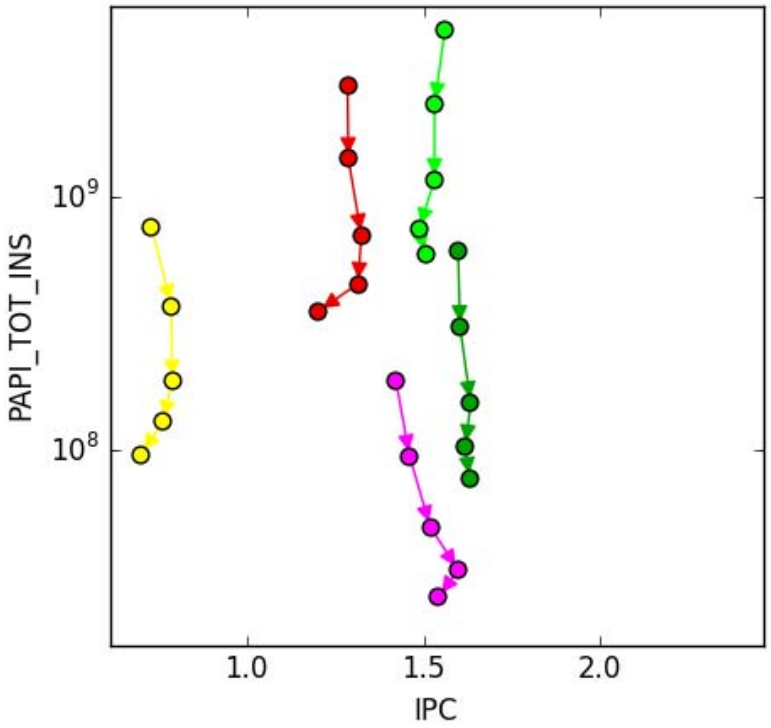


288

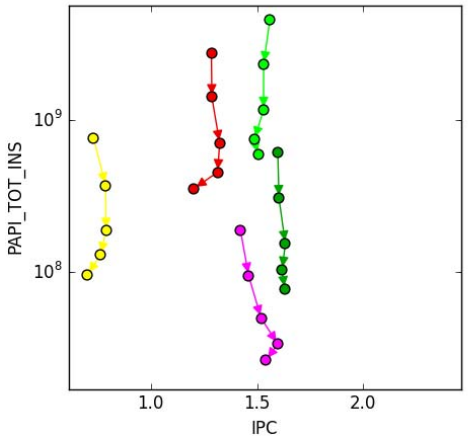


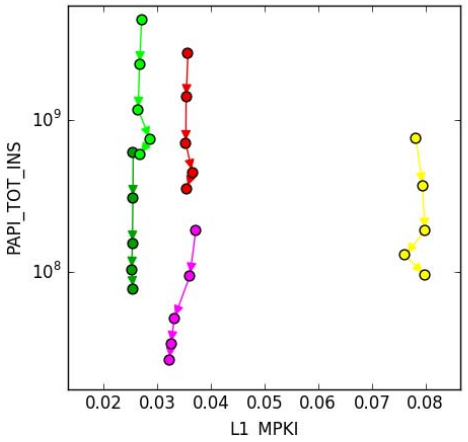
384

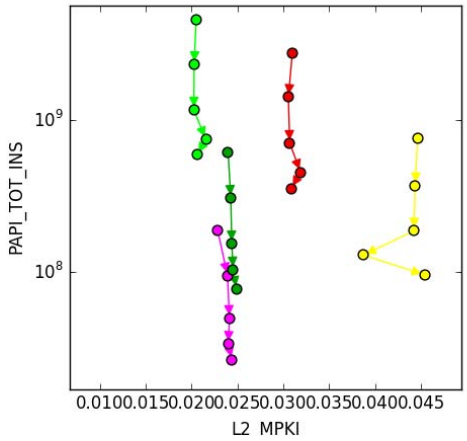


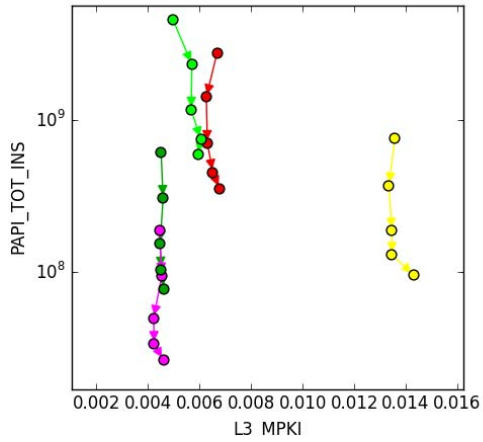


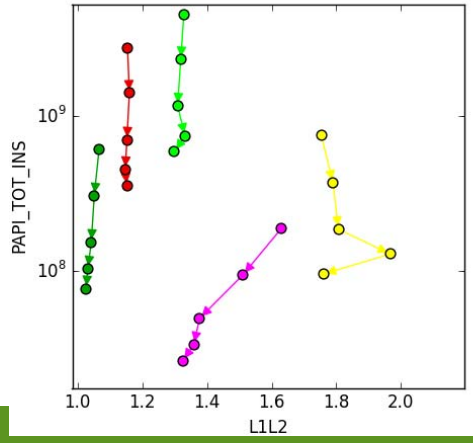
Example: MPI strong scaling

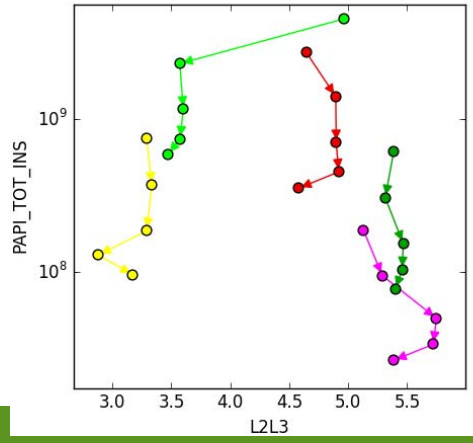












Example: MPI+OMP strong scaling

48x1

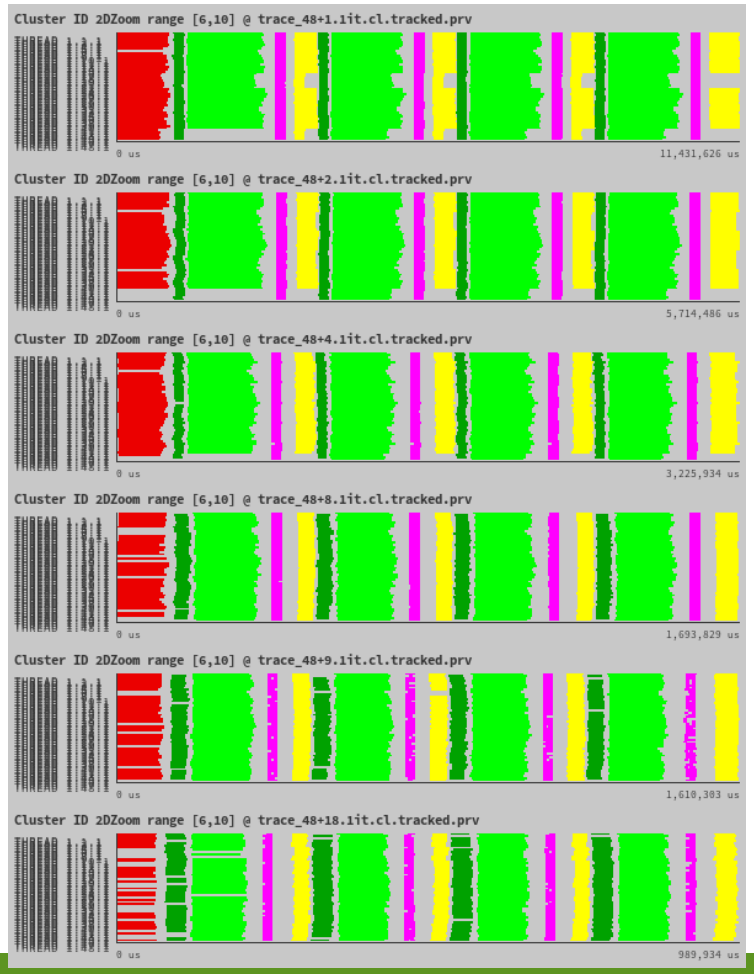
48x2

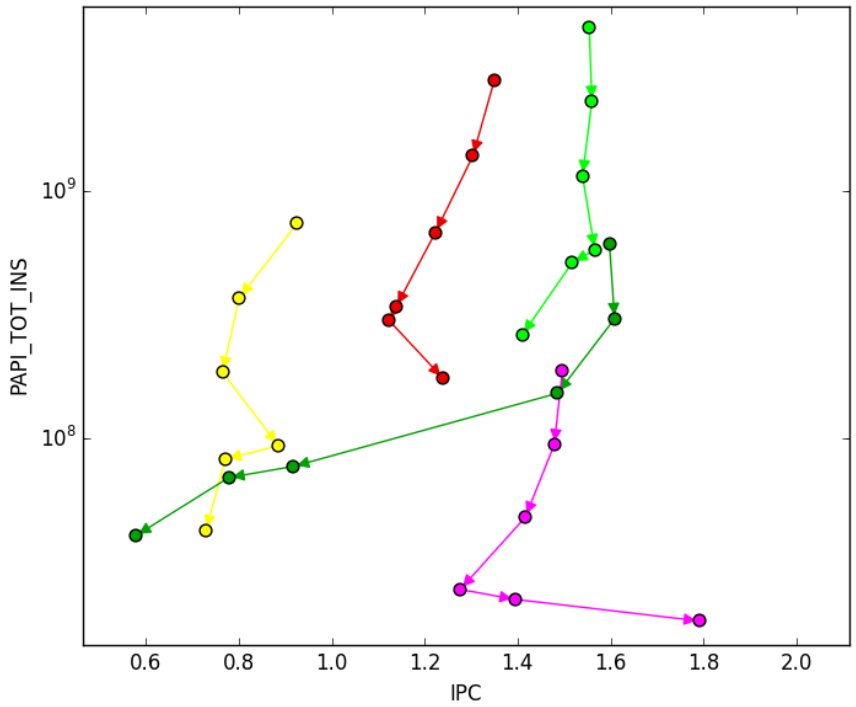
48x4

48x8

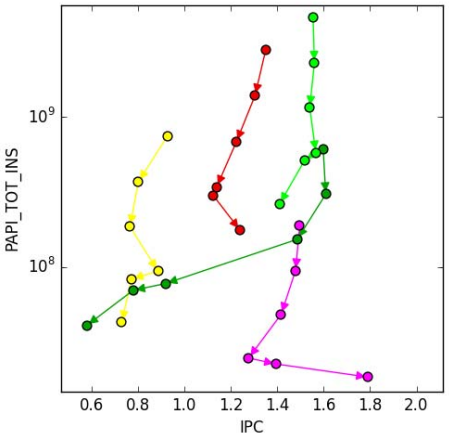
48x9

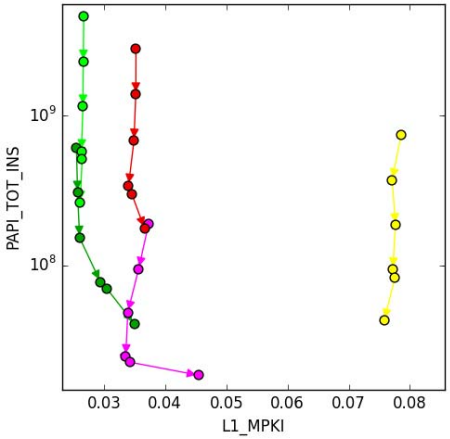
48x18

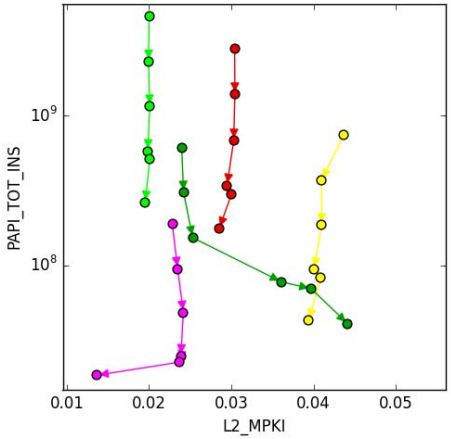


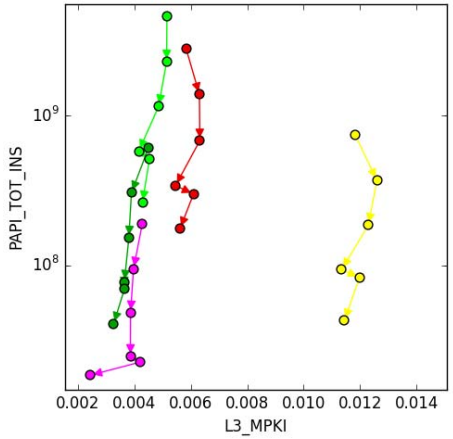


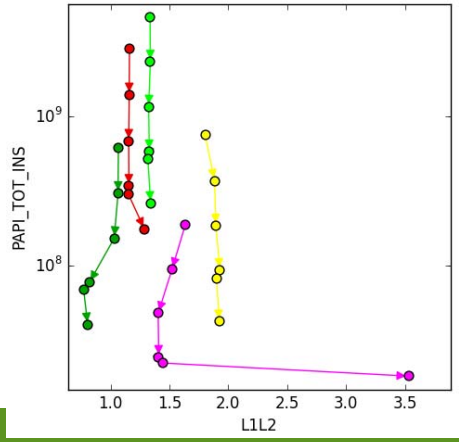
Example: MPI+OMP strong scaling

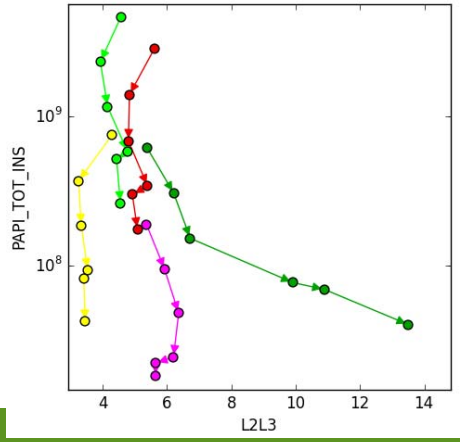






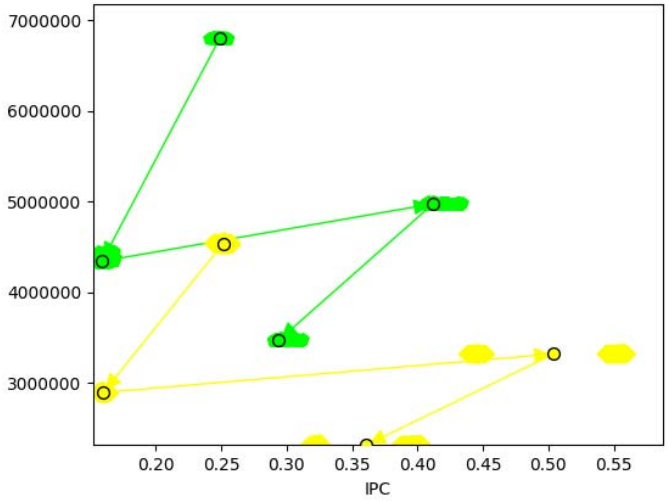






Track other factors

- Track behaviour when changing compiler and hardware?
  (A) Machine A / Compiler A
  (B) Machine A / Compiler B
  (C) Machine B / Compiler A
  (D) Machine B / Compiler C



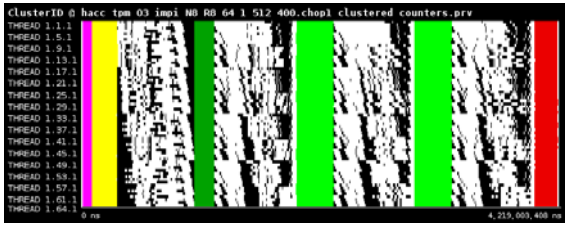
- Open vs. Specialized compiler
  - Reduction of instructions
  - Equivalent IPC loss
  - Not big difference in execution time

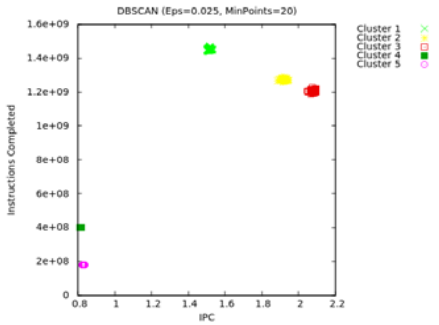
Agenda

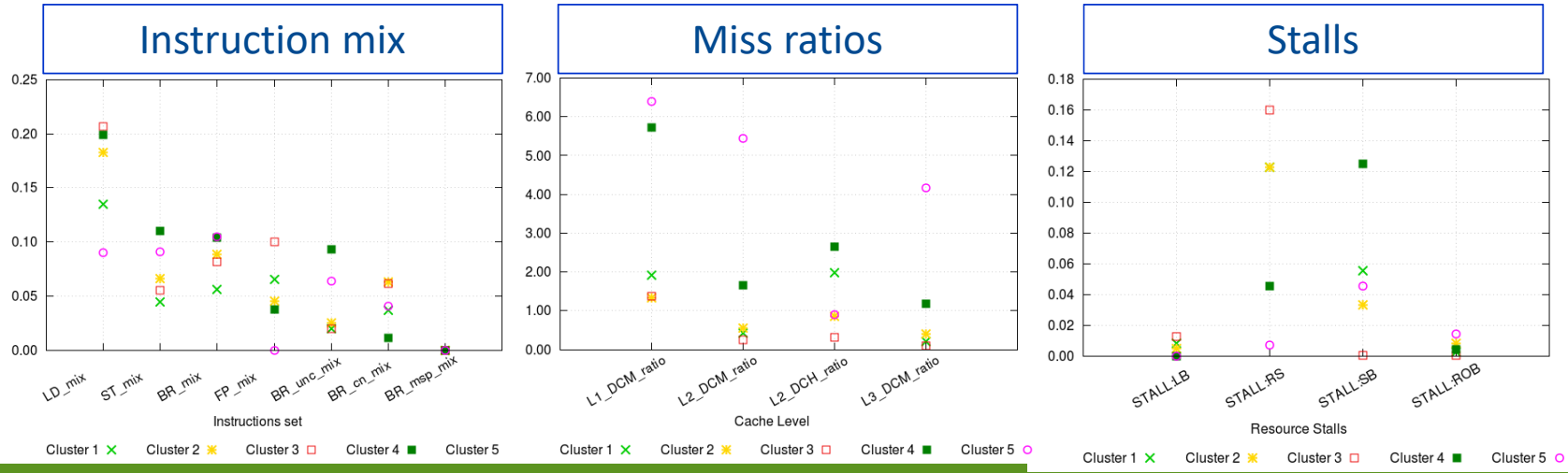
- Reviewing parallel efficiencies and scaling behavior
- Characterizing computational efficiencies
- Application Structure
  - Clustering
- Tracking the scaling behavior of computational phases
- What's next

Other Performance analytics

- “All” hardware counters from a single run



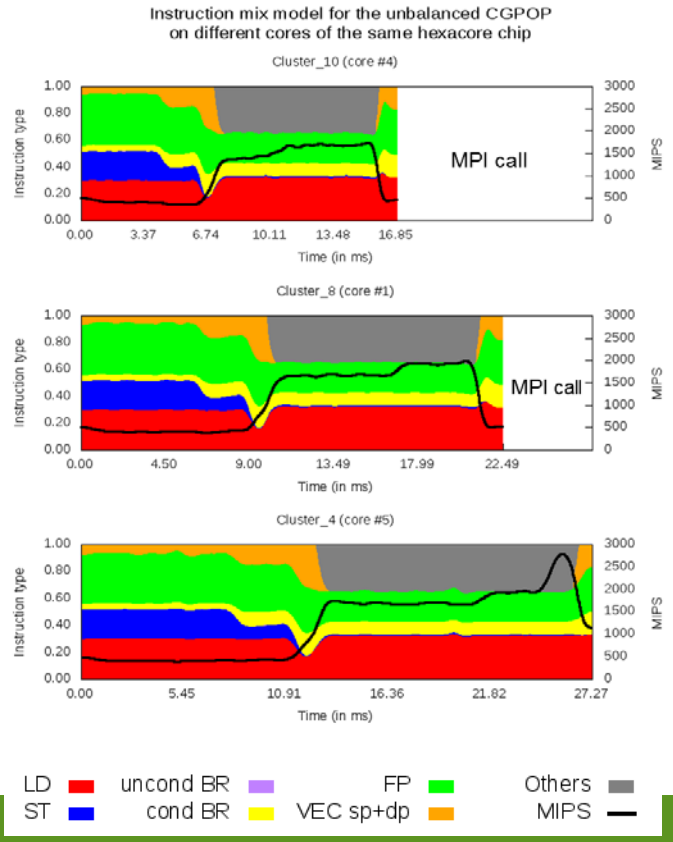


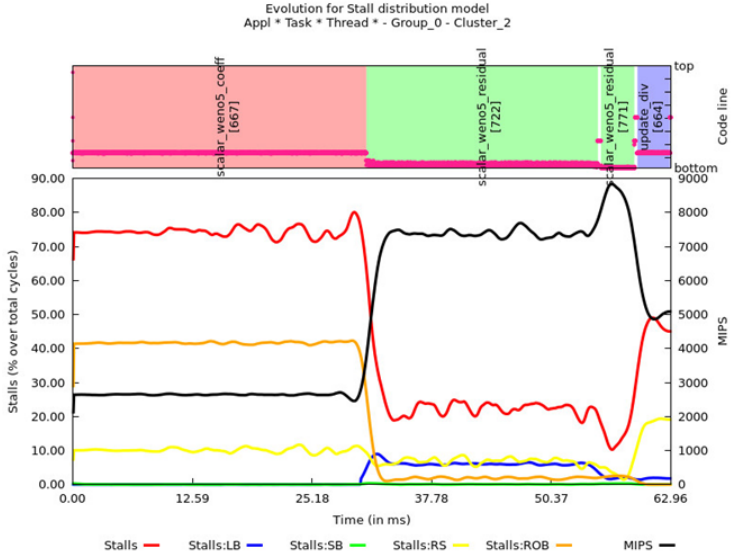


González J. et al , “Performance Data Extrapolation in Parallel Codes”. ICPADS '10

Other Performance analytics

- Instantaneous metrics at “no” cost





Further material

- Follow the “Learning material” link within our web page

  https://www.pop-coe.eu

Performance Optimisation and Productivity

A Centre of Excellence in Computing Applications

Contact:

https://www.pop-coe.eu

mailto:pop@bsc.es

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676553.