Methods for performance evaluation on modern HPC systems:

Performance analysis with Paraver

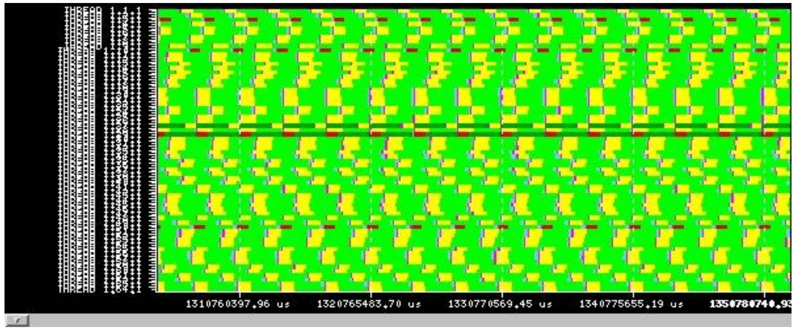
judit@bsc.es

Outline

- The tools
  - Introduction
  - Paraver
    - Applied research: clustering and sampling
    - Analysis examples
  - Dimemas
    - Simulation examples

- Some GPUs examples
  - GPUSs
  - HMPP
  - Extrae support

Our Tools

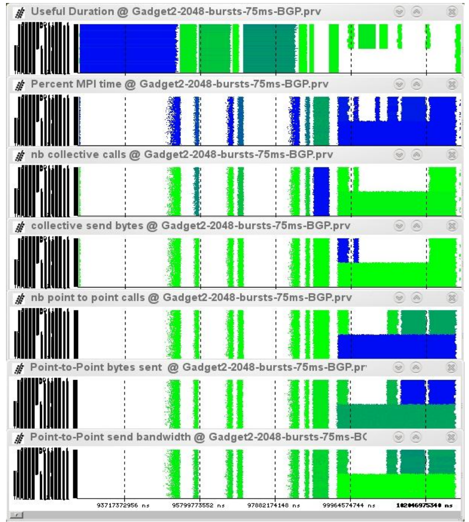


Since 1991

Based on traces

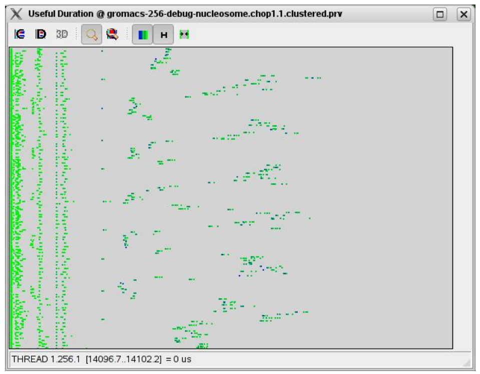
Core tools:

- Paraver – offline trace analysis
- Dimemas – message passing simulator
- Extrae – instrumentation



Detail and intelligence

- analysis of applications at large scale
- intelligent online data reduction
- mixed instrumentation and sampling
- advanced modelling/prediction



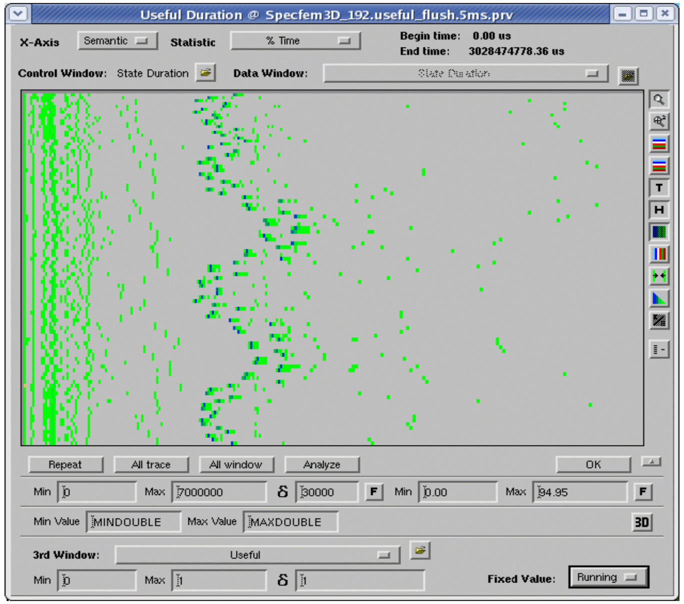
Why traces?

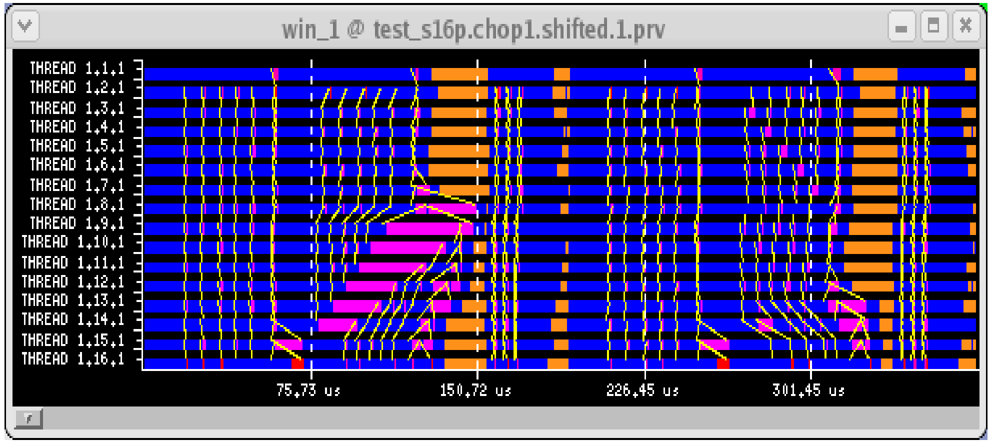
Variability is important

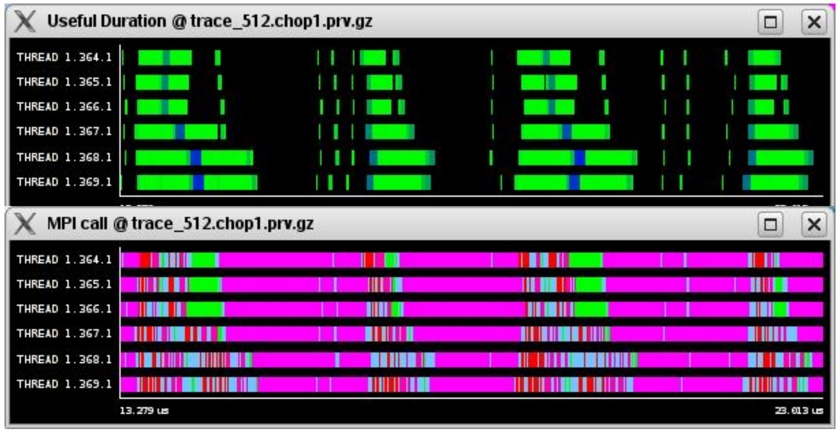
along time, across processors

Higly non-linear systems

microscopic effects may have macroscopic impact

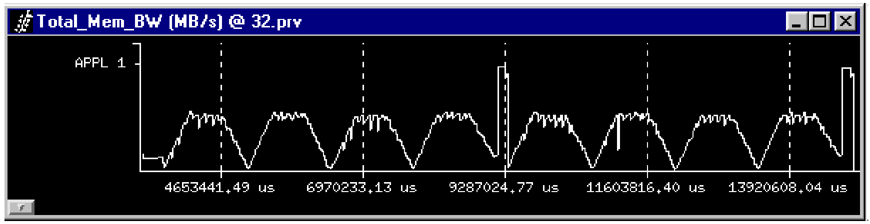






Paraver – Performance data browser

Raw data

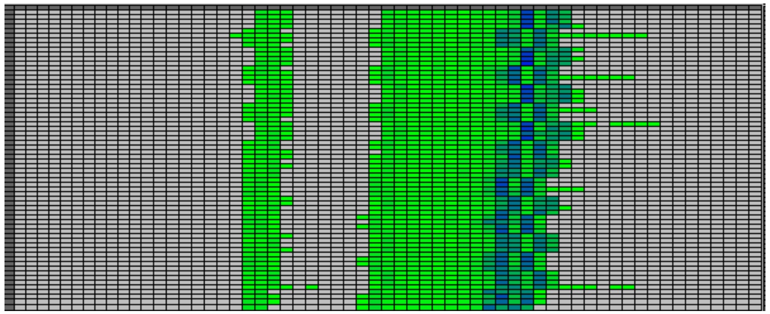


Goal = Flexibility

- No semantics
- Configurable

Timelines

Identifier of function
Performance (IPC, Mflops…)
Routine duration
• • •



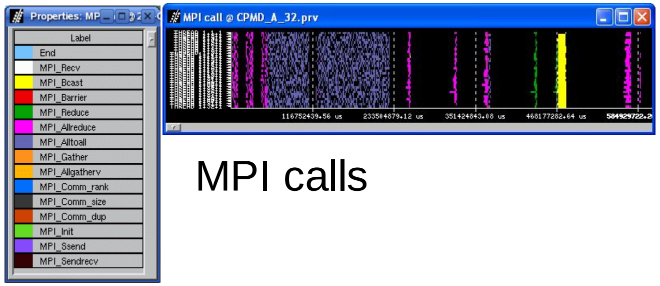
Statistics

Profiles
Average miss ratio per routine
Histogram of routine duration
• • •

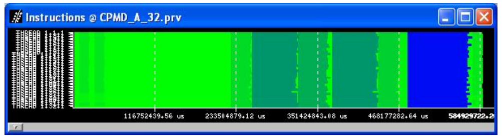
Views: Timelines

Raw events → Piece-wise constant functions of time → plots / colors

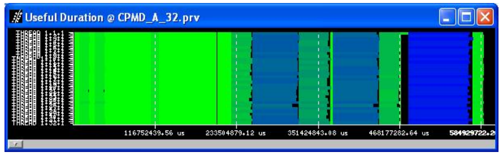
- Basic metrics



MPI calls

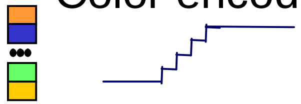


Instructions



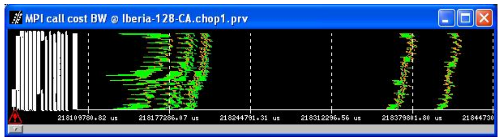
Useful duration

Color encoding

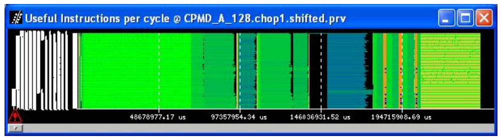


- Derived metrics

$$MPI_{call_{Cost}} = \frac{MPI_{call_{duration}}}{bytes}$$



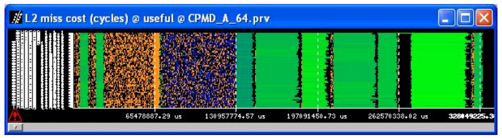
$$useful_{IPC} = \frac{instr}{cycles} \square useful$$



- Models

$$preempted_{time} = elapsed - \frac{cycles}{clock_{freq}}$$

$$L2_{miss_{latency}} = \frac{cycles - instr / idealIPC}{L2misses}$$

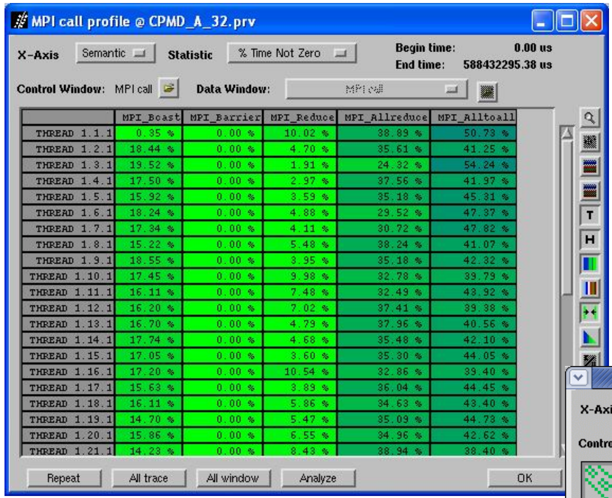


Views: Statistics

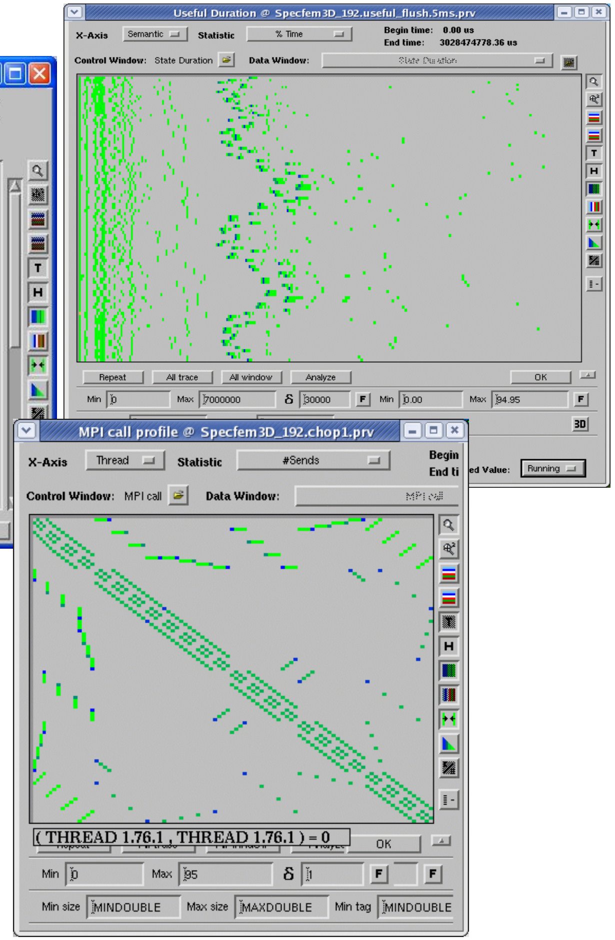
2D

- Profiles
- Histograms
- Correlations
- Communication Patterns

MPI calls profile

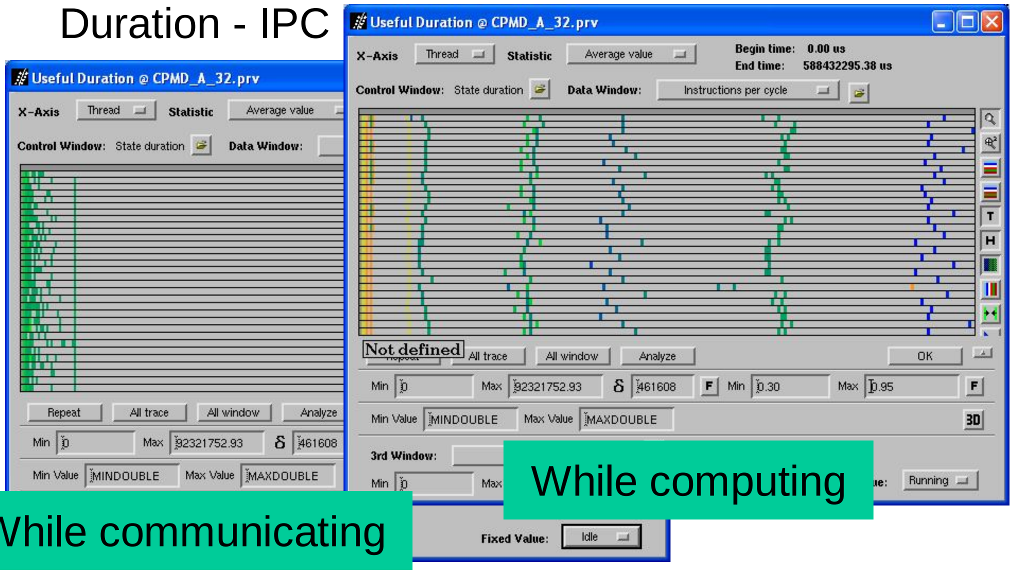


	MPI_Bcast	MPI_Barrier	MPI_Reduce	MPI_Allreduce	MPI_Alltoall
THREAD 1.1.1	0.35 %	0.00 %	10.02 %	38.89 %	50.73 %
THREAD 1.2.1	18.44 %	0.00 %	4.70 %	35.61 %	41.25 %
THREAD 1.3.1	19.52 %	0.00 %	1.91 %	24.32 %	54.24 %
THREAD 1.4.1	17.50 %	0.00 %	2.97 %	37.56 %	41.97 %
THREAD 1.5.1	15.92 %	0.00 %	3.59 %	35.18 %	45.31 %
THREAD 1.6.1	18.24 %	0.00 %	4.88 %	29.52 %	47.37 %
THREAD 1.7.1	17.34 %	0.00 %	4.11 %	30.72 %	47.82 %
THREAD 1.8.1	15.22 %	0.00 %	5.48 %	38.24 %	41.07 %
THREAD 1.9.1	18.55 %	0.00 %	3.95 %	35.18 %	42.32 %
THREAD 1.10.1	17.45 %	0.00 %	9.98 %	32.78 %	39.79 %
THREAD 1.11.1	16.11 %	0.00 %	7.48 %	32.49 %	43.92 %
THREAD 1.12.1	16.20 %	0.00 %	7.02 %	37.41 %	39.38 %
THREAD 1.13.1	16.70 %	0.00 %	4.79 %	37.96 %	40.56 %
THREAD 1.14.1	17.74 %	0.00 %	4.68 %	35.48 %	42.10 %
THREAD 1.15.1	17.05 %	0.00 %	3.60 %	35.30 %	44.05 %
THREAD 1.16.1	17.20 %	0.00 %	10.54 %	32.86 %	39.40 %
THREAD 1.17.1	15.63 %	0.00 %	3.89 %	36.04 %	44.45 %
THREAD 1.18.1	16.11 %	0.00 %	5.86 %	34.63 %	43.40 %
THREAD 1.19.1	14.70 %	0.00 %	5.47 %	35.09 %	44.73 %
THREAD 1.20.1	15.86 %	0.00 %	6.55 %	34.96 %	42.62 %
THREAD 1.21.1	14.23 %	0.00 %	8.43 %	38.94 %	38.40 %



3D

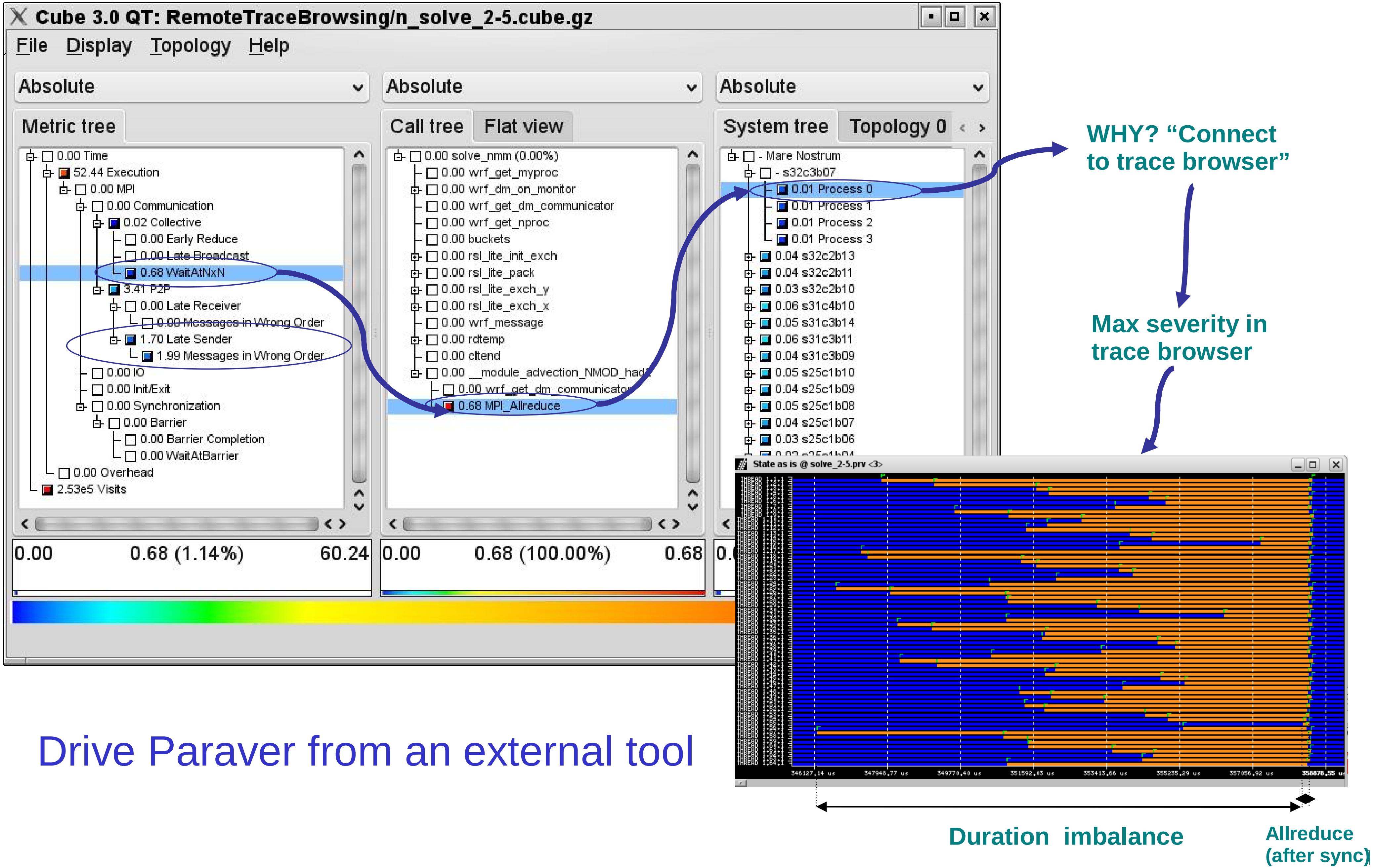
Duration - IPC



While communicating

While computing

Tools integration: CUBE + Paraver



Drive Paraver from an external tool

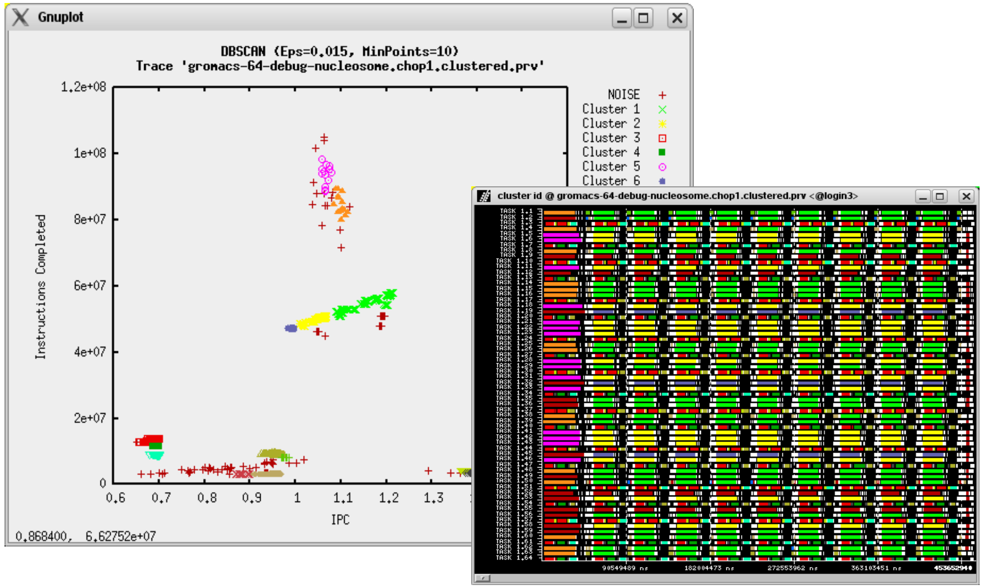
Clustering module – detecting structure

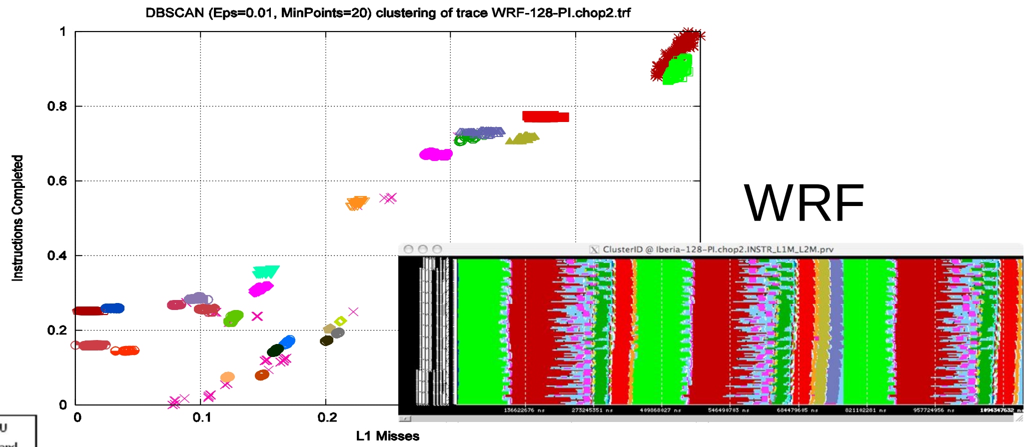
Highlight structure

- Clusters injected in trace
- Phases vs. routines

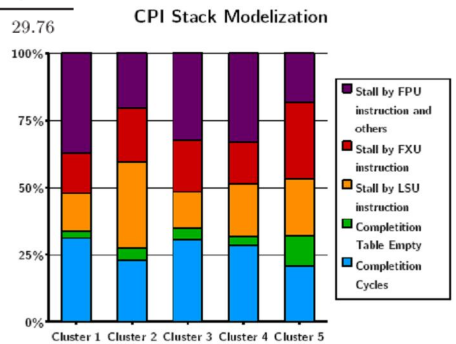
Counters projection

- Statistics
- Models (CPI stack)

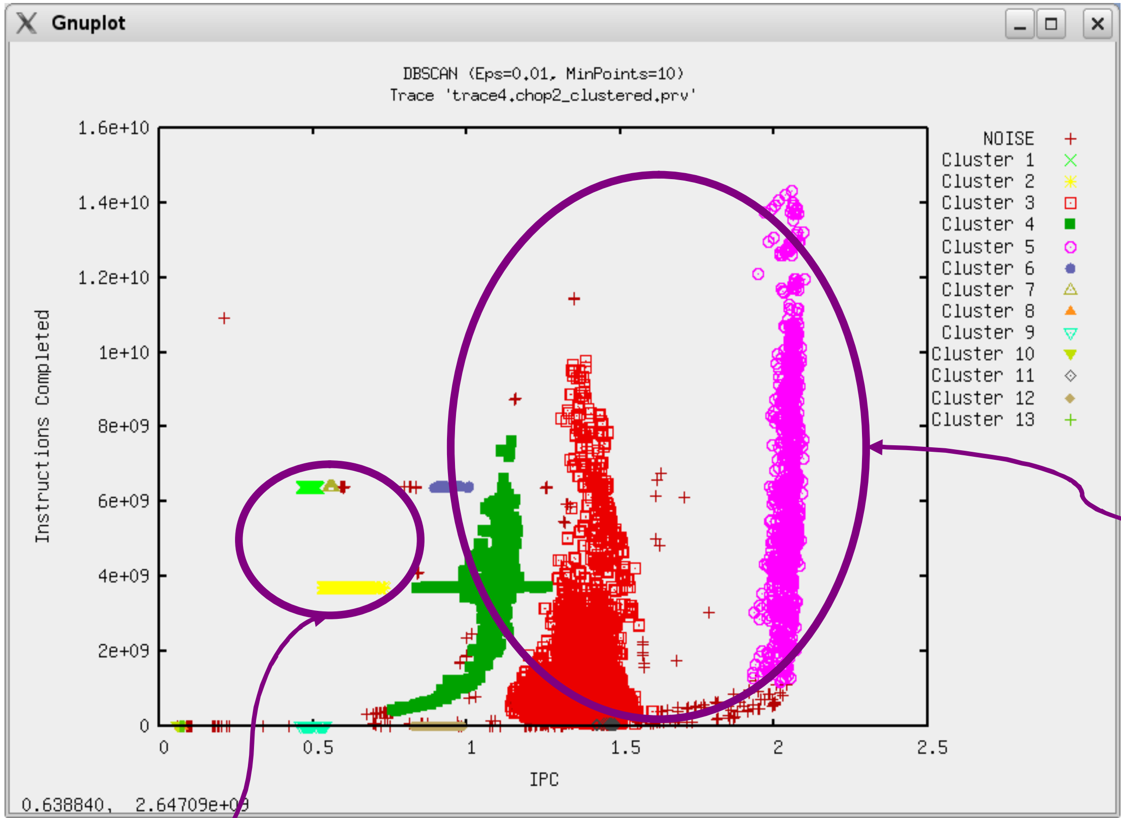




Cluster	1	2	3	4	5
%Time	54.88	17.96	16.90	6.44	1.42
Avg. Burst Dur. (ms)	1.02	0.78	13.14	2.50	1.11
IPC	1.02	0.65	0.89	0.91	0.53
MIPS	2231.8	1423.3	1966.5	2001.8	1163.0
MFLOPS	339.2	46.3	191.6	269.2	23.6
L1M/KInstr	0.92	1.53	1.19	1.17	2.88
L2M/KInstr	0.06	1.26	0.06	0.35	0.21
Mem.BW (MB/s)	16.79	218.47	13.87	85.77	29.76

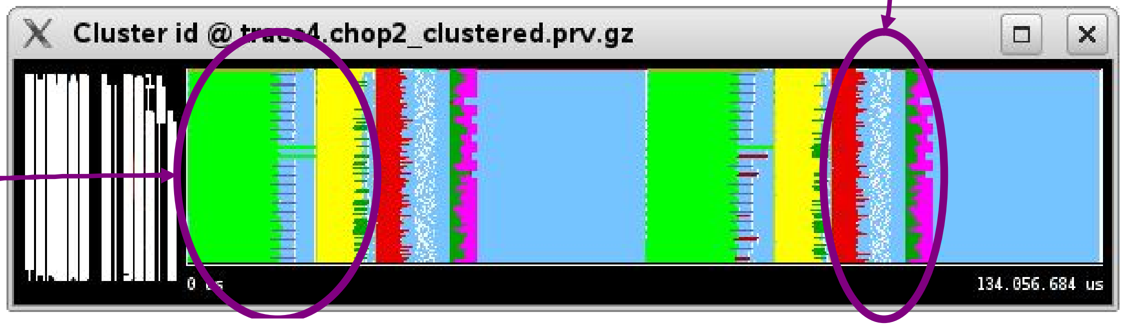


Clustering module – detecting structure



Huge variability on instructions

Most relevant computing regions obtain poor IPC



Unbalance due to different number of chunks

Folding samples – granularity / overhead

Application granularity vs. detailed granularity

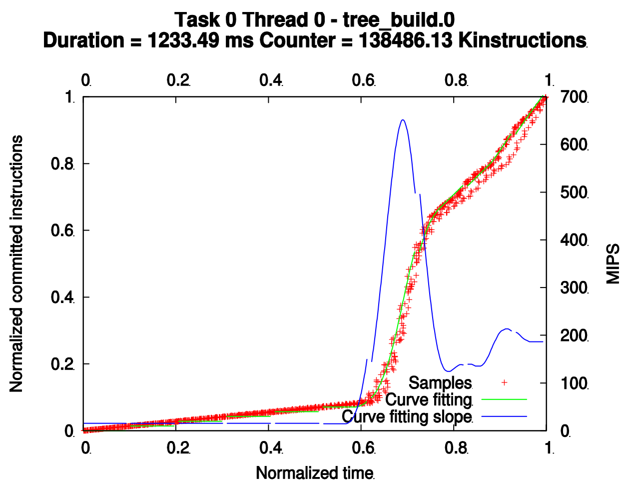
Samples

hardware counters + callstack

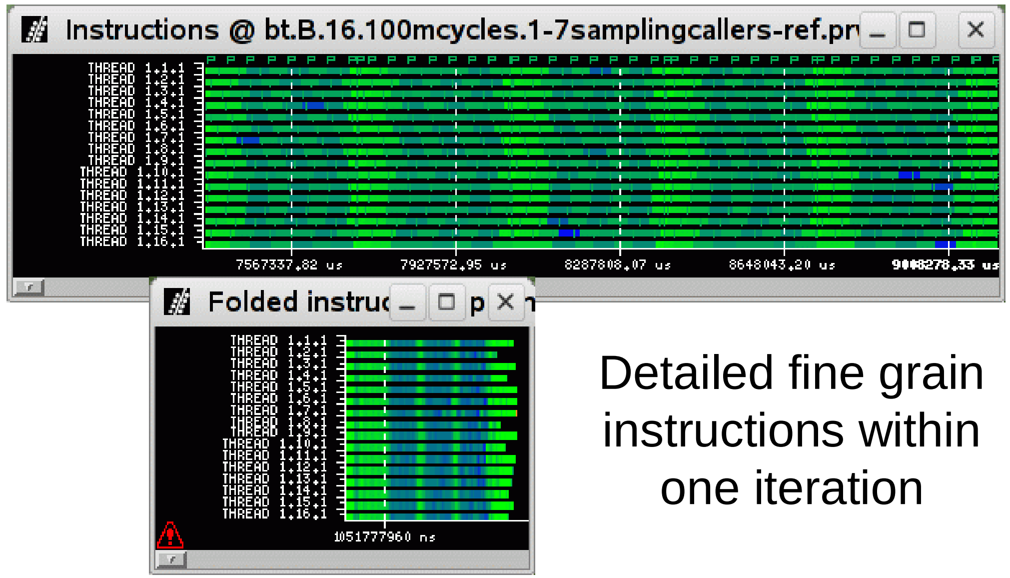
Folding

reduce sampling frequency / overhead

based on known structure: iterations, routines, clusters

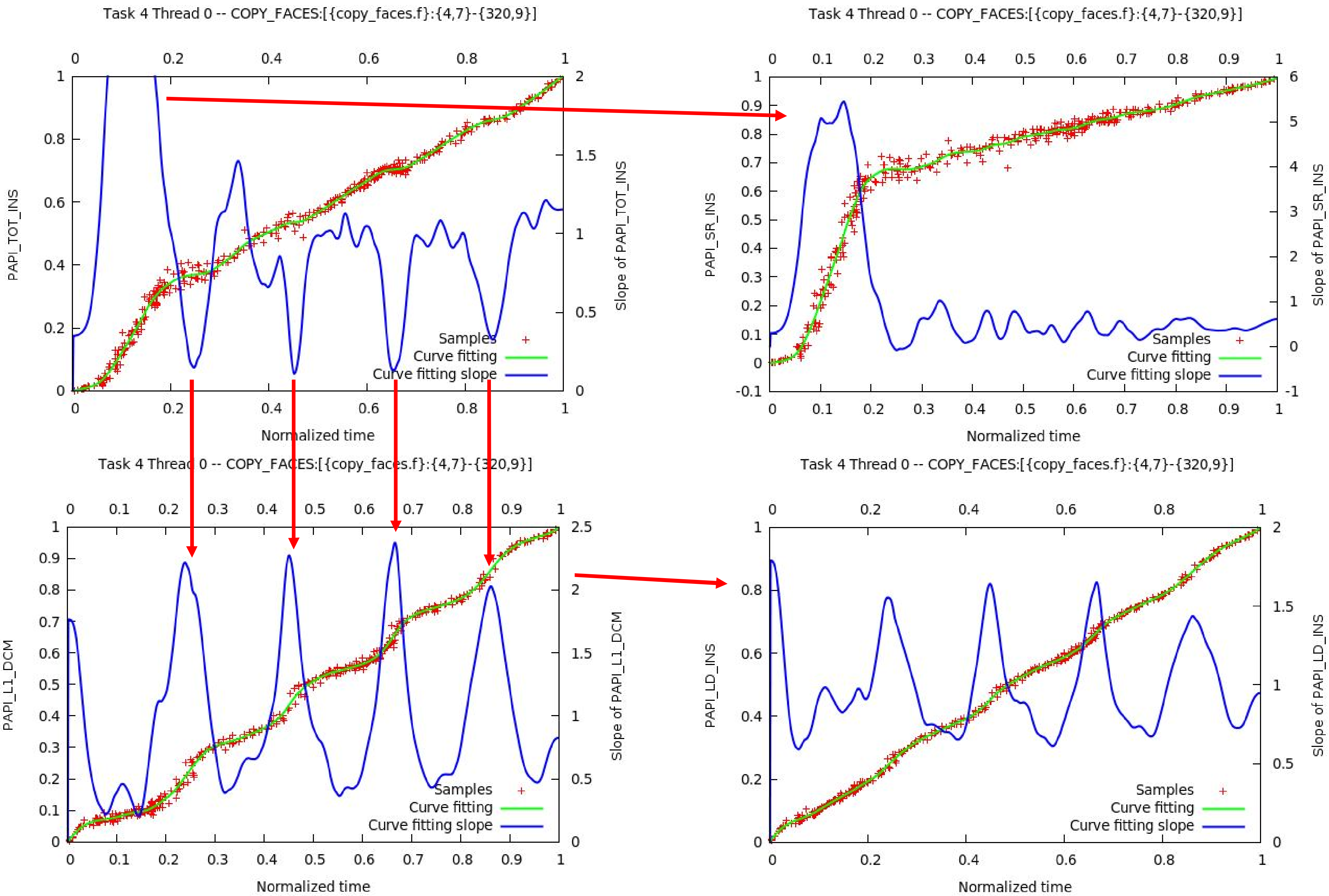


Original sampled instructions



Detailed fine grain instructions within one iteration

Folding samples – granularity / overhead

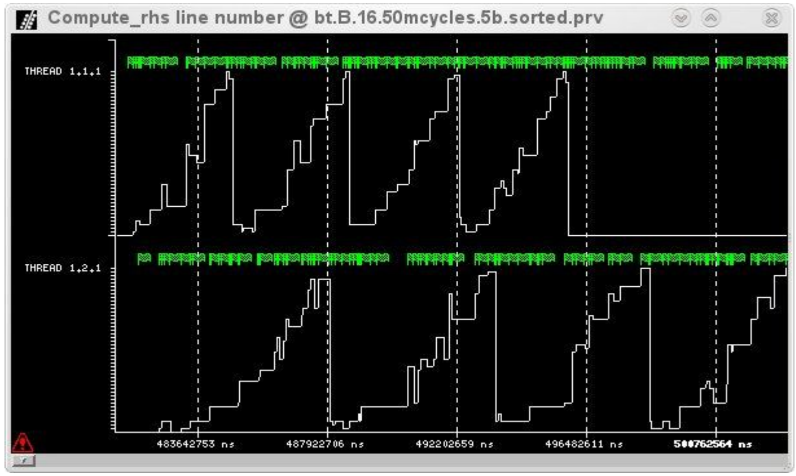


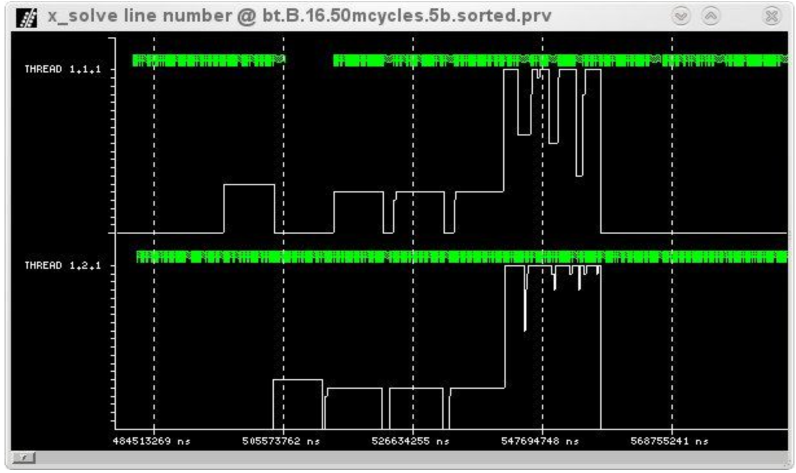
Folding samples – granularity / overhead

Compute_rhs

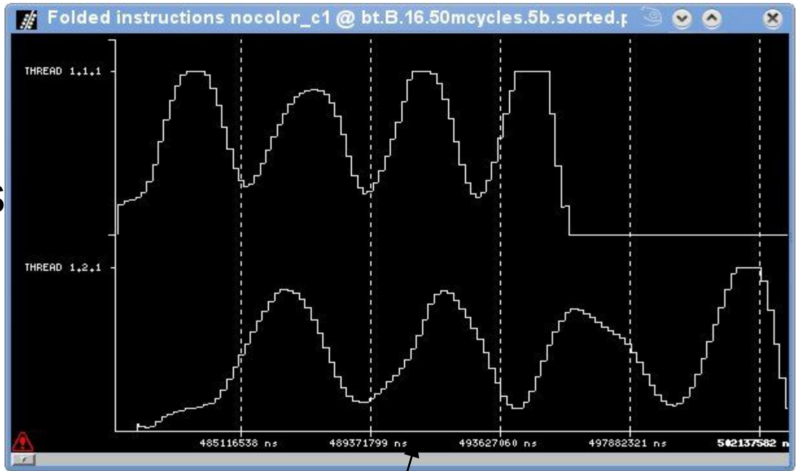
x_solve

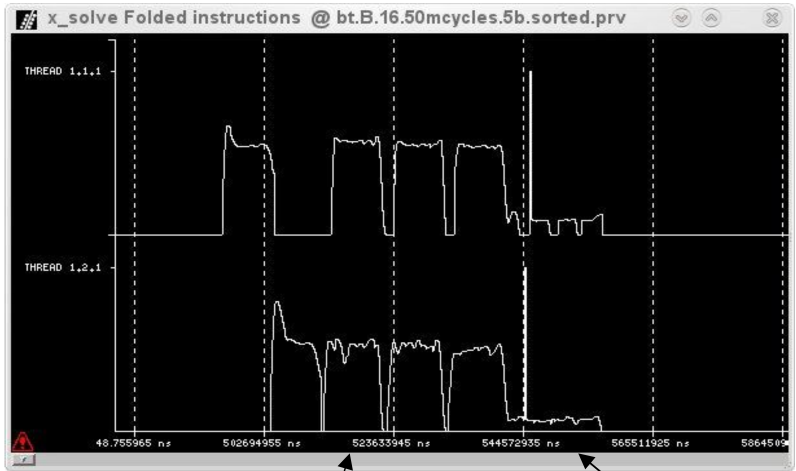
Source line





Instantanous MIPS



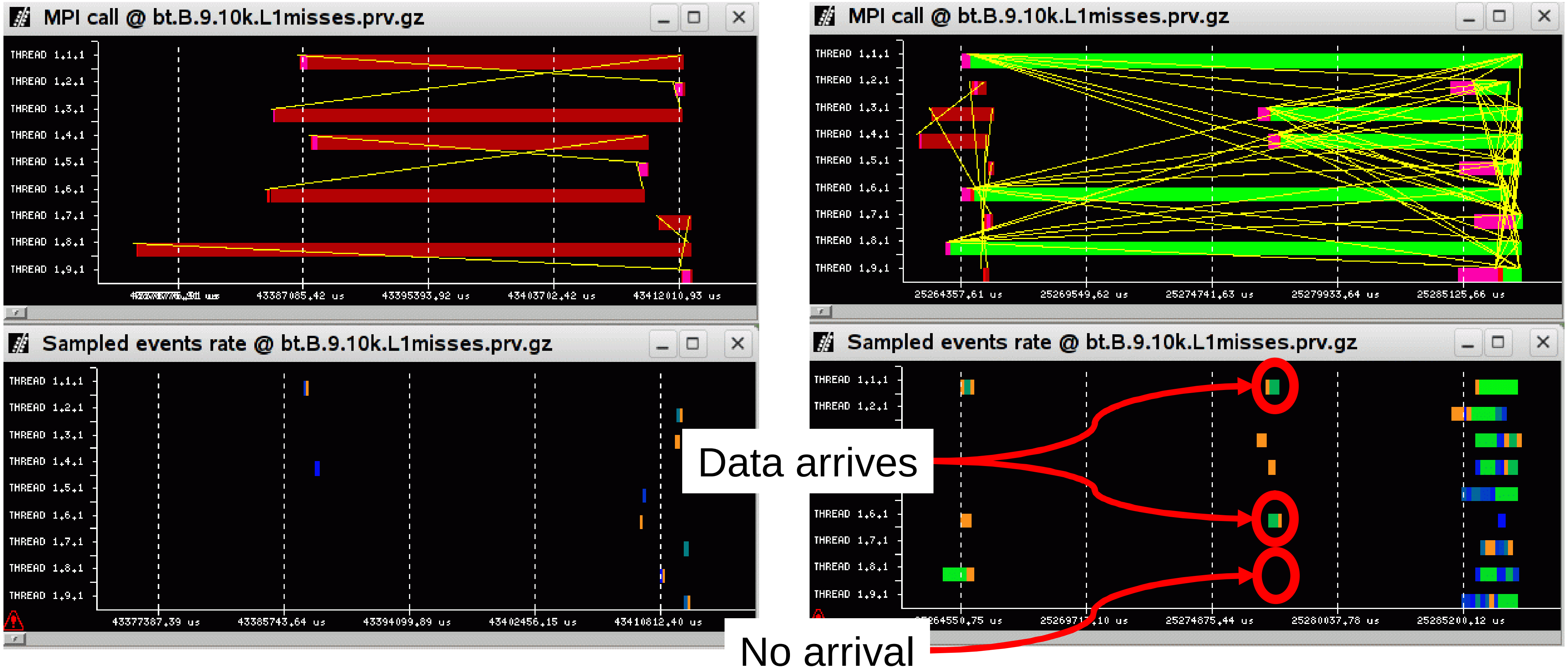


4 iterations outer loop

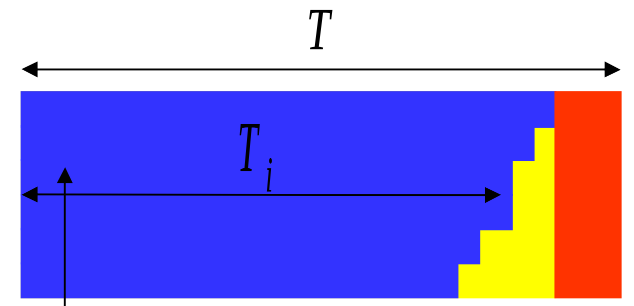
4 x_solve_cell 4 backsubstitute

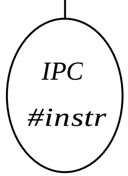
Sampling driven by other HWC

Density of L1 misses within MPI in a SMPs: when the data actually arrives



Scalability model

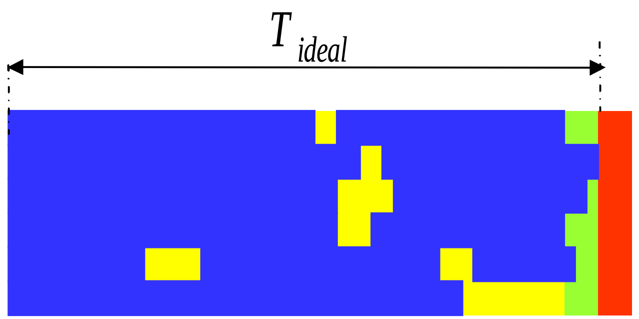




$$Sup=\frac{P}{P_0}\times\frac{LB}{LB_0}\times\frac{CommEff}{CommEff_0}\times\frac{IPC}{IPC_0}\times\frac{instr_0}{instr}$$

Directly from real execution metrics

$$eff_i=\frac{T_i}{T} \qquad LB=\frac{\sum_{i=1}^{P} eff_i}{P*\max(eff_i)} \qquad CommEff=\max(eff_i)$$



$$Sup=\frac{P}{P_0}\times\frac{macroLB}{macroLB_0}\times\frac{microLB}{microLB_0}\times\frac{CommEff}{CommEff_0}\times\frac{IPC}{IPC_0}\times\frac{instr_0}{instr}$$

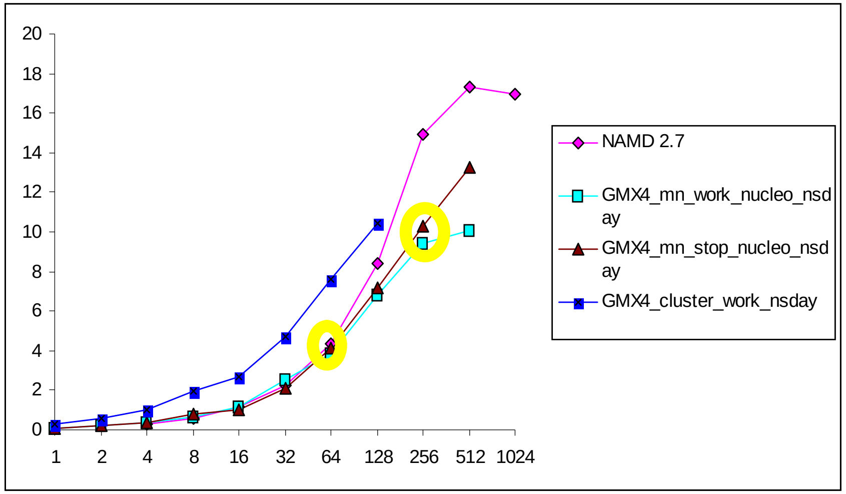
Migrating/local load imbalance
Serialization

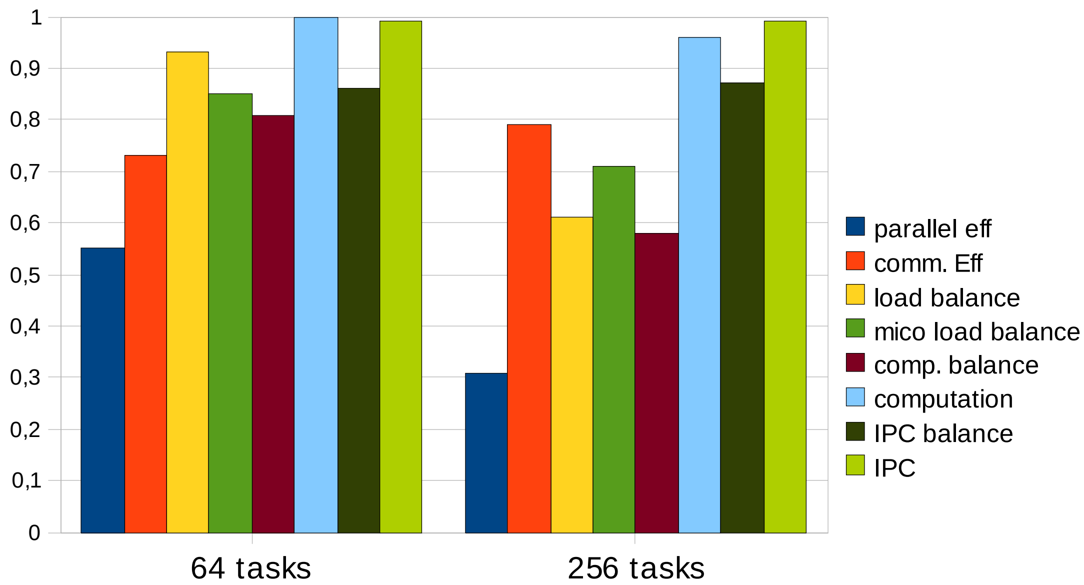
Requires Dimemas simulation

$$microLB=\frac{\max(T_i)}{T_{ideal}} \qquad CommEff=\frac{T_{ideal}}{T}$$

Example: From quick analysis...

GROMACS analysis



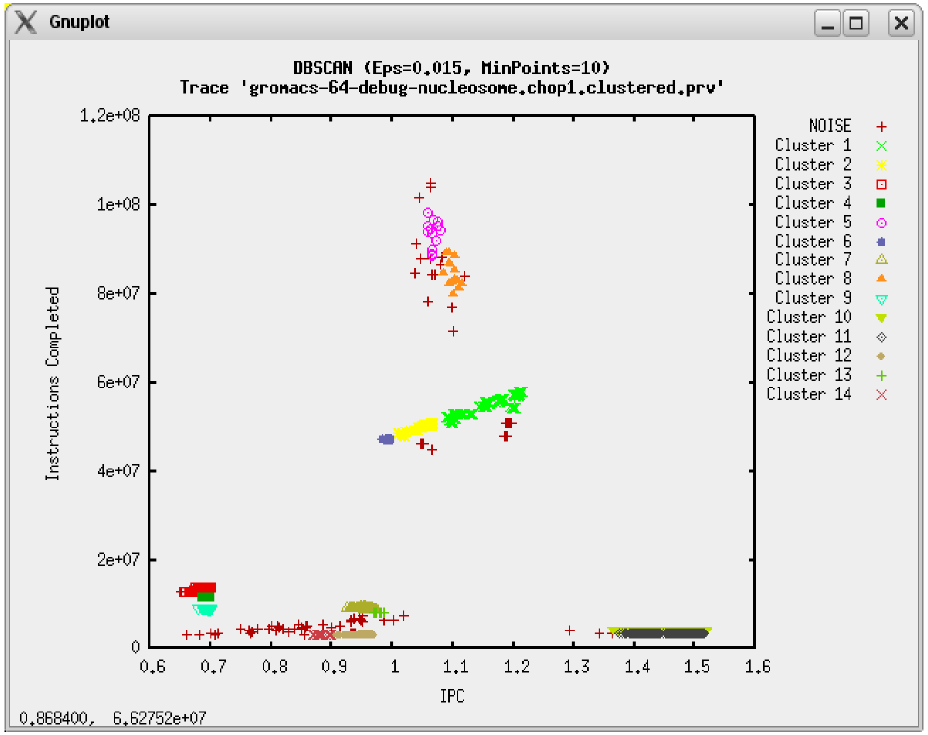


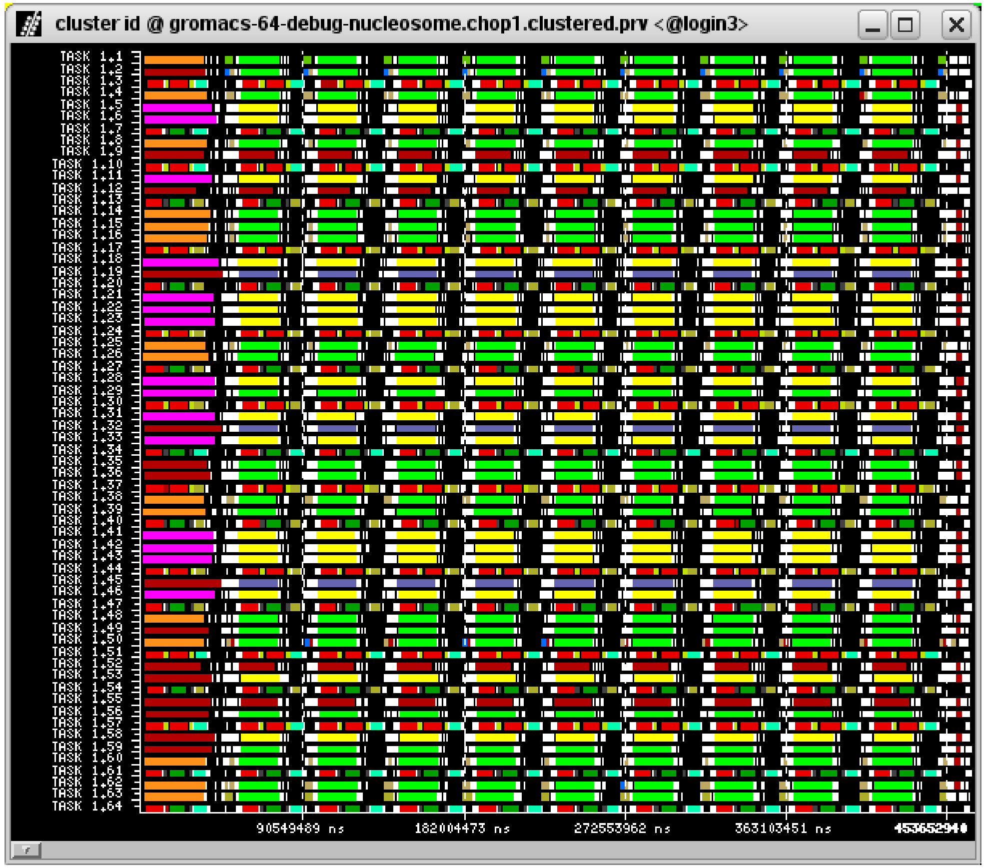
Main scalability problems:
- Load balance
- Computation balance

4% of code replication

64 tasks already poor efficiency!
- Communication problem

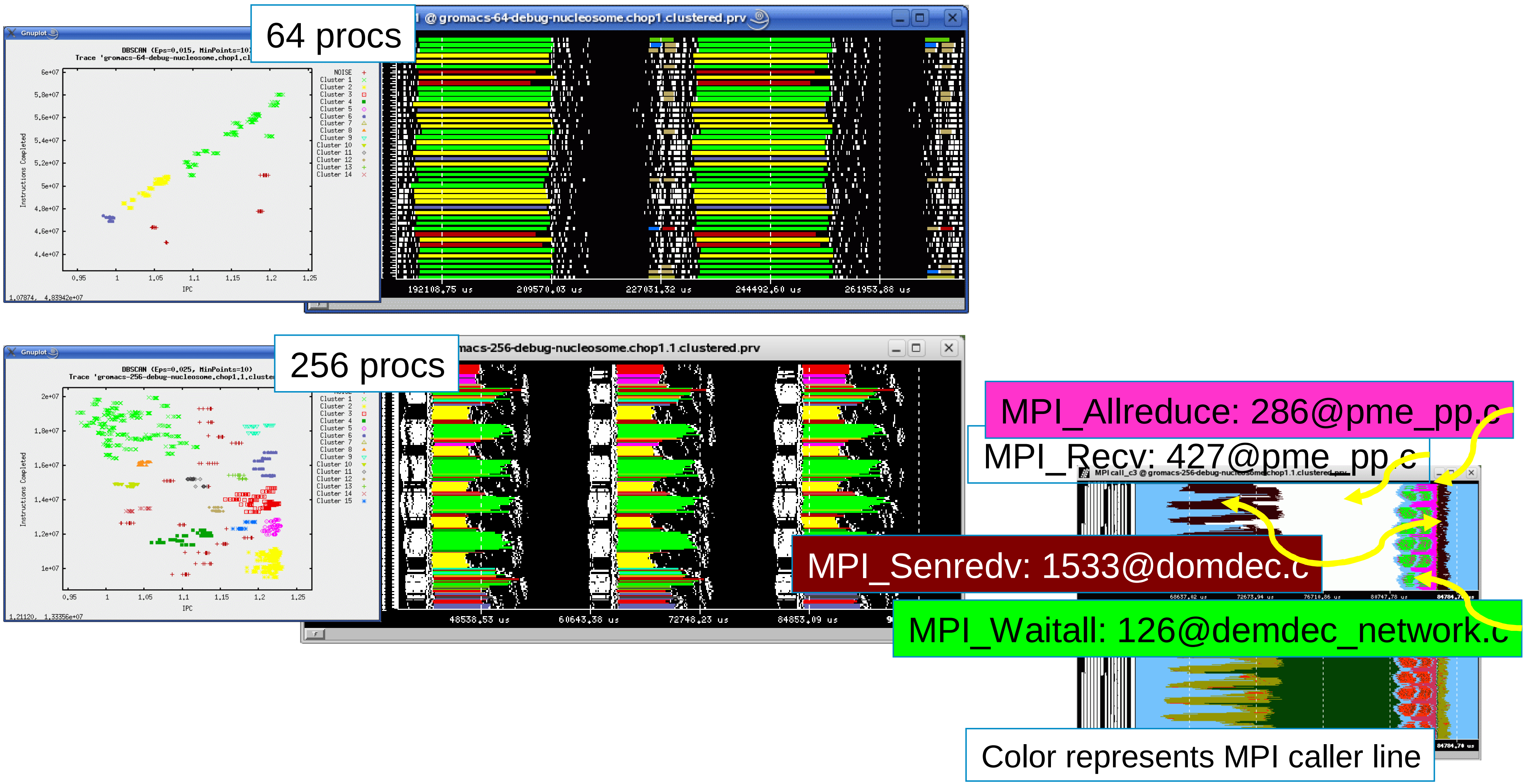
GROMACS structure, 64 tasks





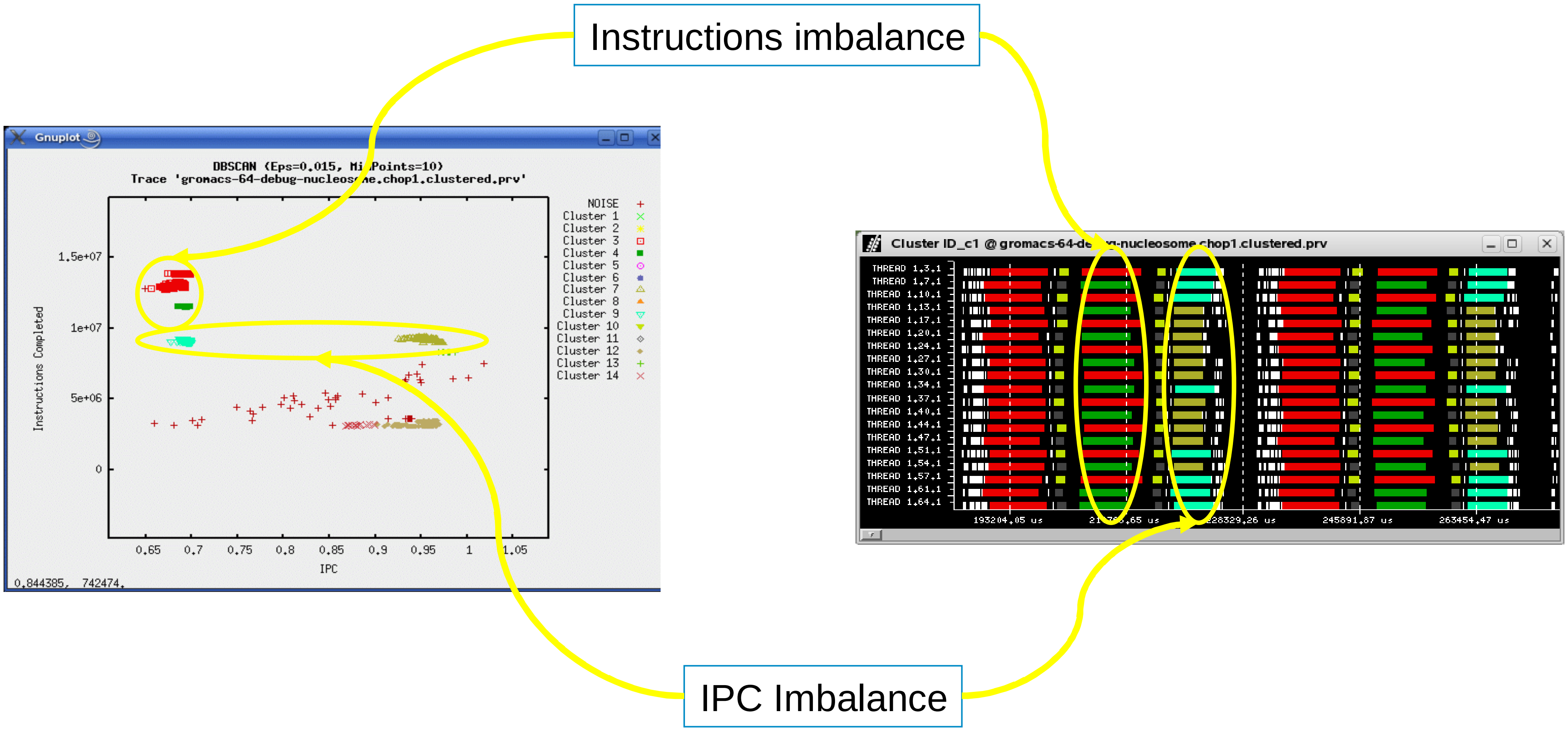
Example: ... to details

GROMACS PMEs balance / scalability



Example: ... to details

GROMACS FFTs balance

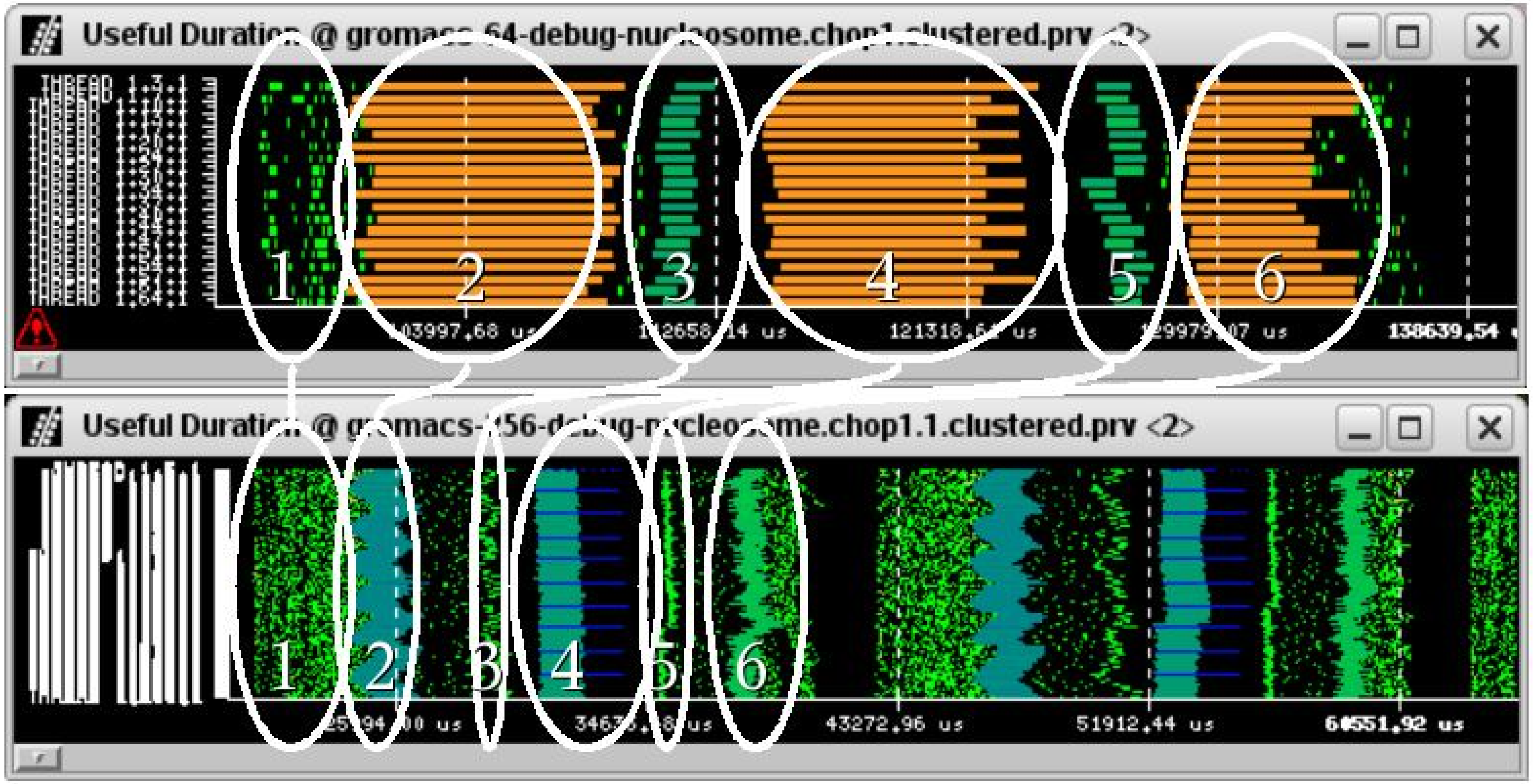


Example: ... to details

GROMACS: Comparing computation time for FFTs

64 tasks
(one iteration)

256 tasks
(2,2 iterations
in the same
time interval,
4 times more
resources)



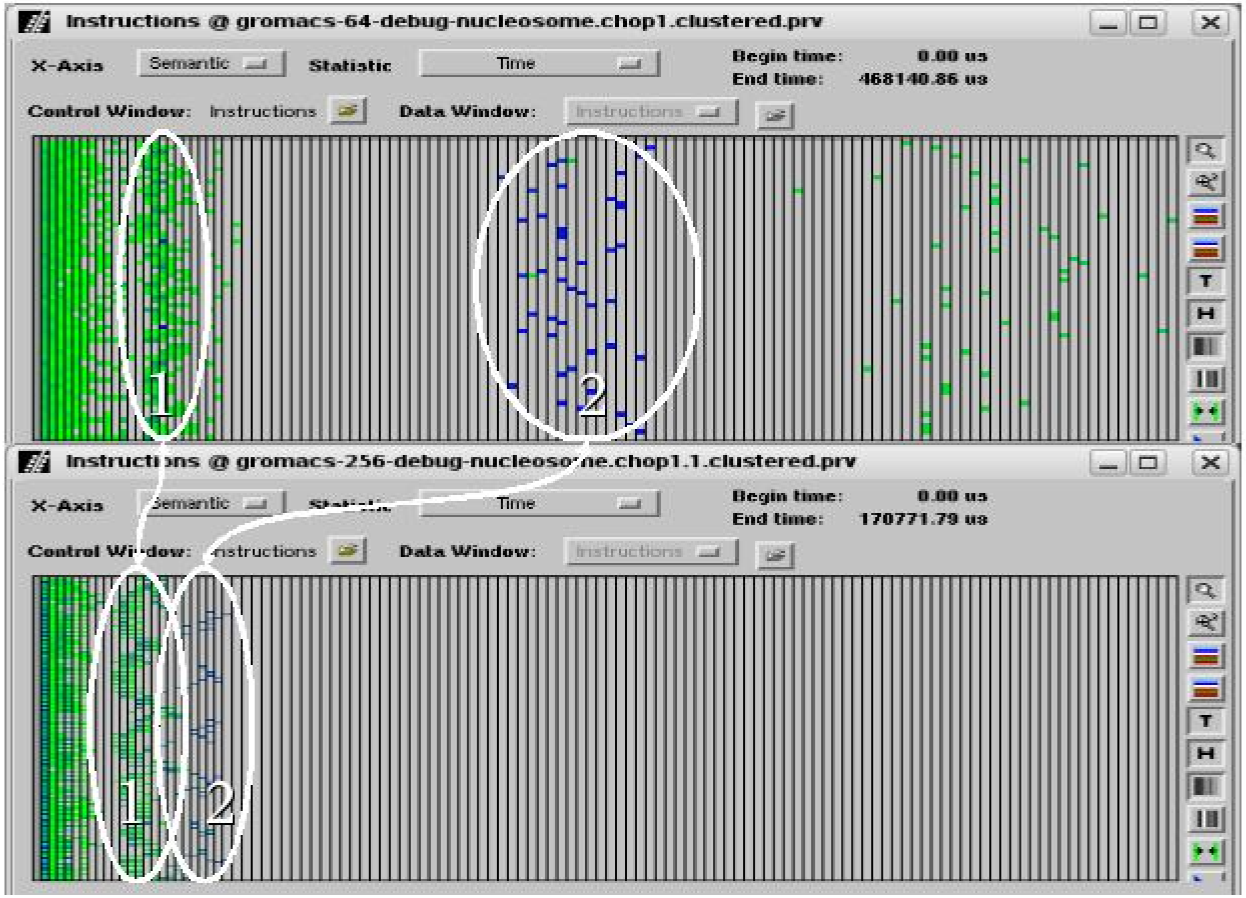
Zone 4 has a potentially structured imbalance with scalability problems (MPI_Alltoallv: 241@fftgrid.c)

Example: ... to details

GROMACS Computational load balance

64 tasks

256 tasks

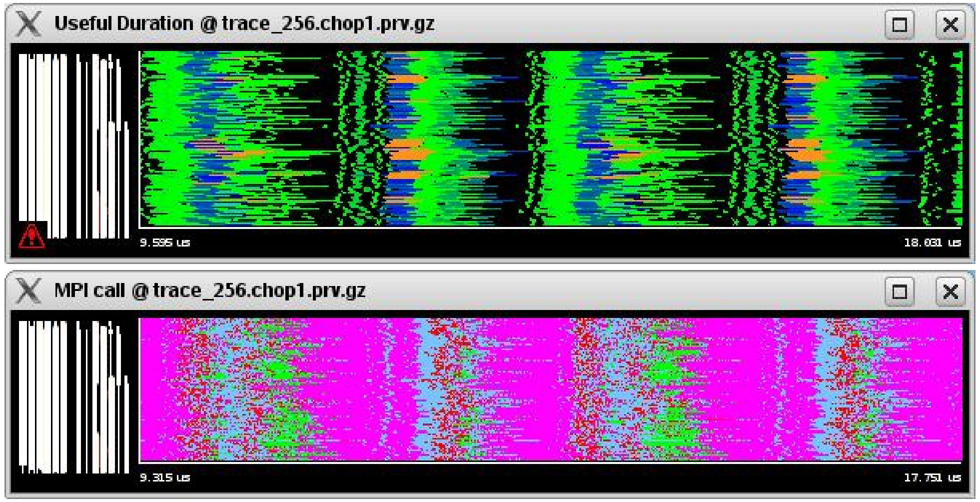


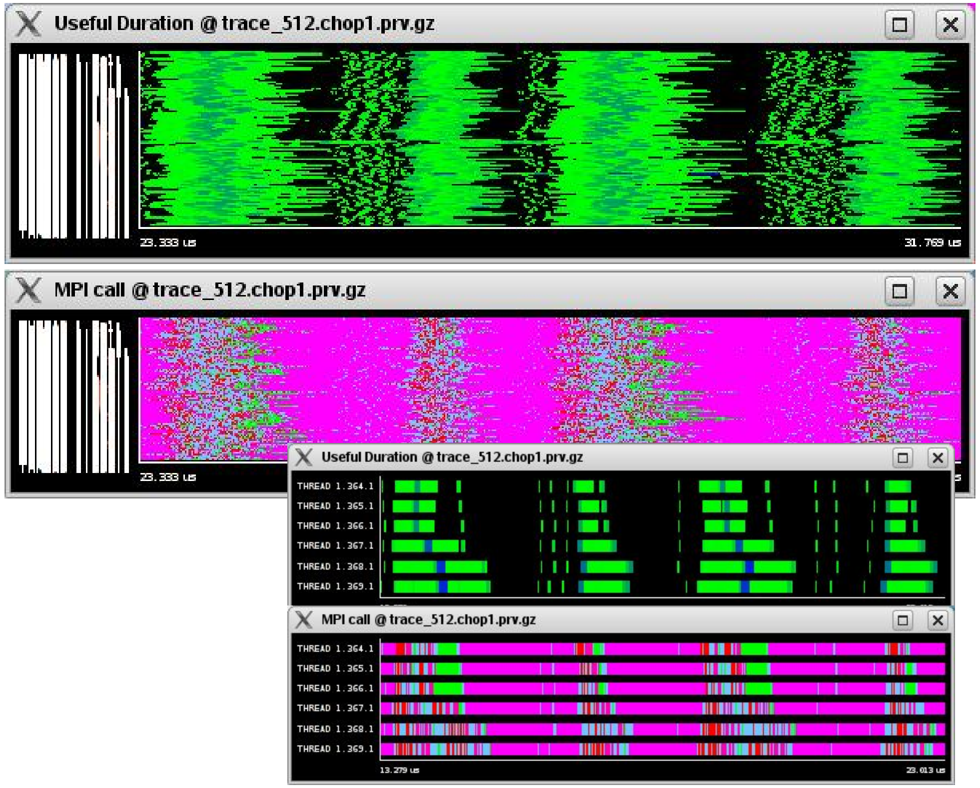
Zone 1 does not scale and increases imbalance!

Example: Code optimization

Short diagnosis: FrontFlowRed – 256 vs 512 tasks

- Unbalance and MPI time increases
- Sequence of MPI_Allreduce calls
- Poor IPC, high cache misses





	256	512
Parallel efficiency	27%	19%
Load balance	56%	48%
% MPI_Allreduce avg time	50%	62%

Example: Code optimization

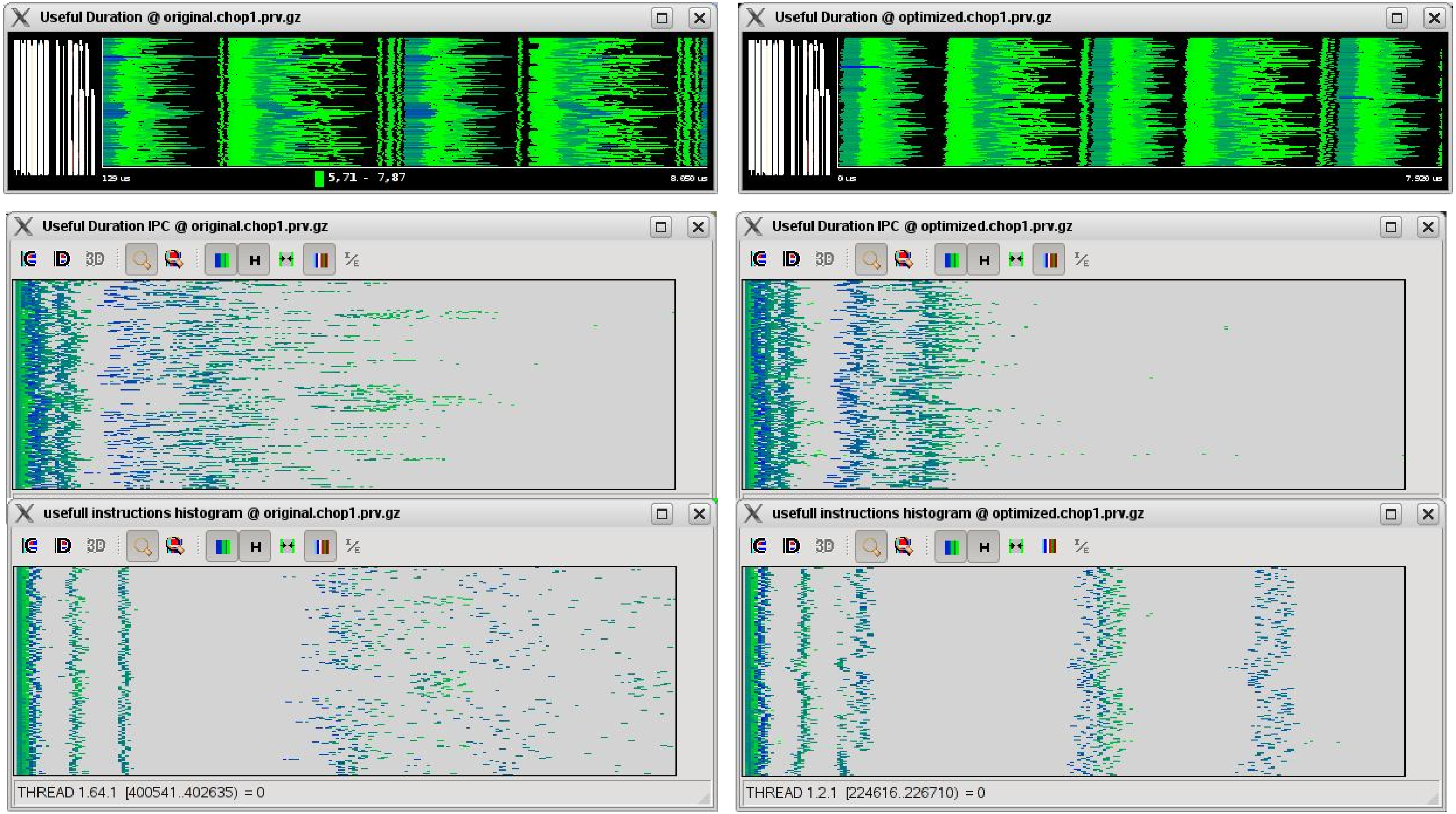
Modifications in the source code

- Improved load balance – weight nodes based on their connectivity, balance total weight
- Reduction of all to all communications – half of MPI_Allreduce cals were eliminated
- Reduce cache misses – reordering node number

Half an hour meeting for the analysis, few days of work to modify code → 25% of improvement

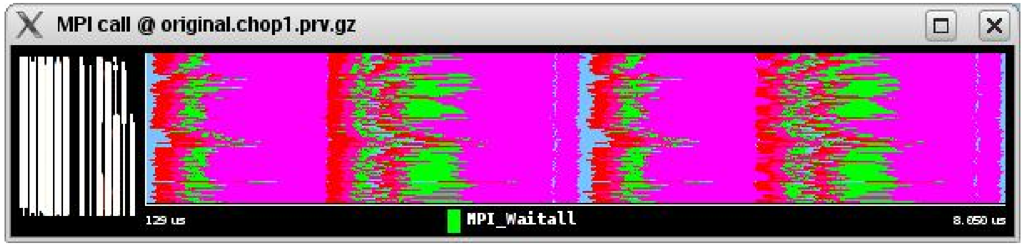
Example: Code optimization

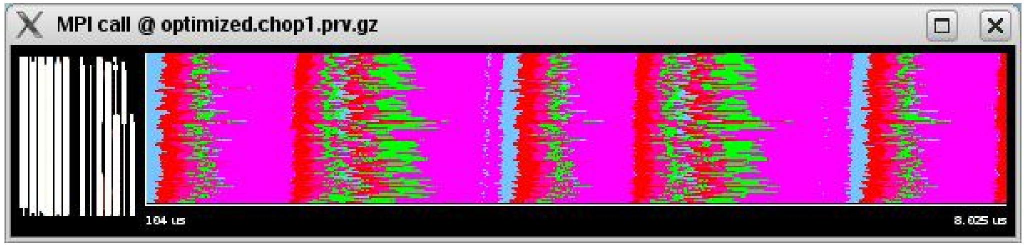
256 tasks – Load balance



Example: Code optimization

256 tasks – MPI time



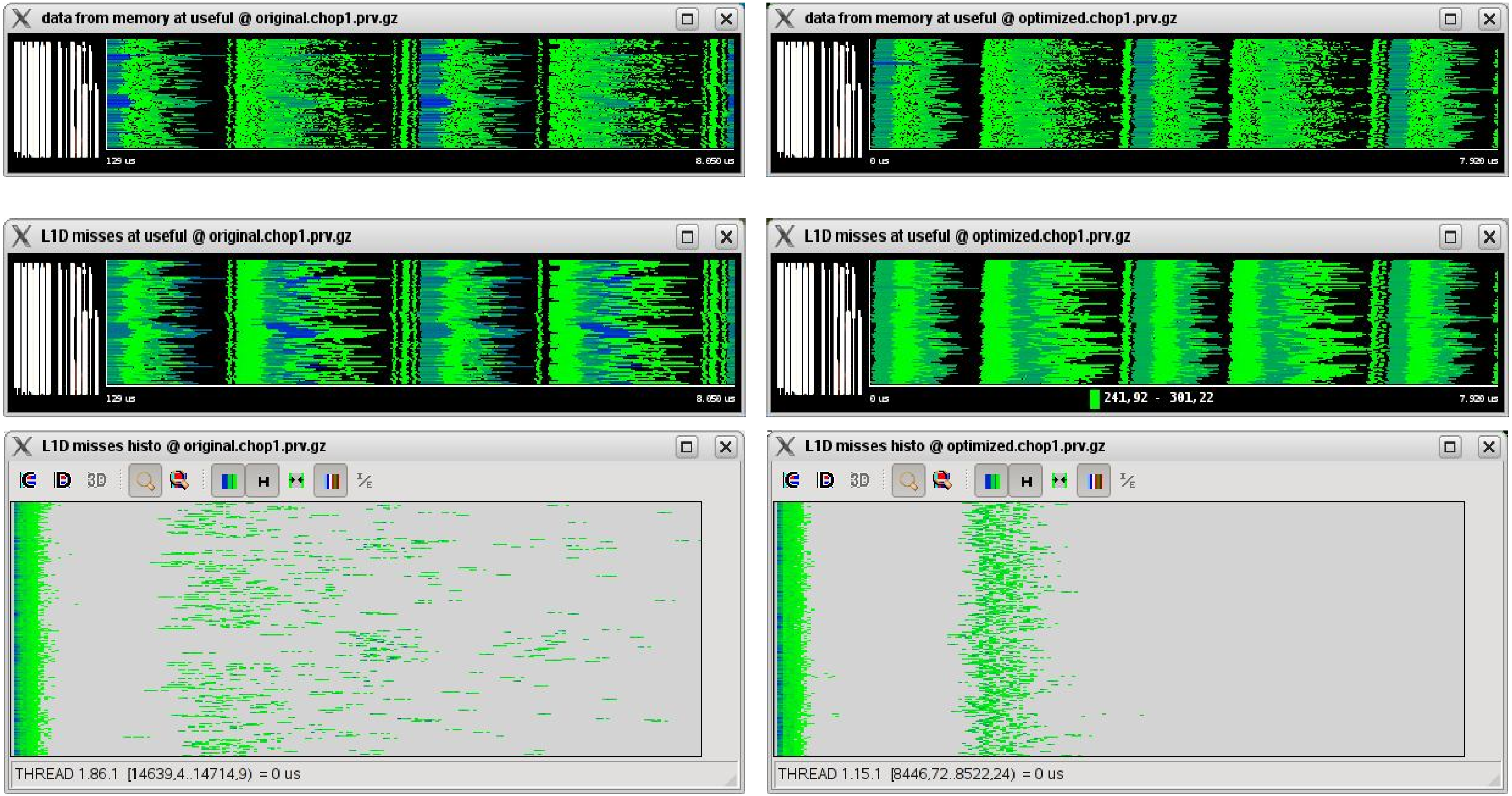


	End	MPI_Isend	MPI_Irecv	MPI_Waitall	MPI_Allreduce
Total	7.617,81 %	2.432,05 %	1.772,21 %	2.245,19 %	11.532,74 %
Average	29,76 %	9,50 %	6,92 %	8,77 %	45,05 %
Maximum	51,85 %	18,11 %	13,00 %	28,59 %	67,69 %
Minimum	17,59 %	1,65 %	1,37 %	1,46 %	14,88 %
StDev	8,04 %	3,33 %	2,39 %	6,05 %	11,03 %
Avg/Max	0,57	0,52	0,53	0,31	0,67

	End	MPI_Isend	MPI_Irecv	MPI_Waitall	MPI_Allreduce
Total	9.089,05 %	3.100,13 %	2.107,96 %	1.511,02 %	9.791,83 %
Average	35,50 %	12,11 %	8,23 %	5,90 %	38,25 %
Maximum	51,05 %	20,77 %	14,59 %	21,31 %	64,34 %
Minimum	22,99 %	3,47 %	3,21 %	1,72 %	10,90 %
StDev	5,92 %	3,62 %	2,61 %	3,89 %	10,13 %
Avg/Max	0,70	0,58	0,56	0,28	0,59

Example: Code optimization

256 tasks – Memory access

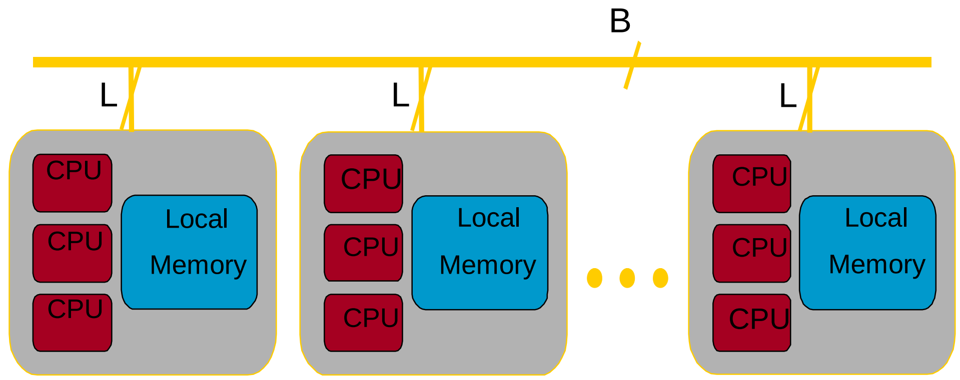


Outline

- The tools
  - Introduction
  - Paraver
    - Applied research: clustering and sampling
    - Analysis examples
  - Dimemas
    - Simulation examples

- Some GPUs examples
  - GPUSs
  - HMPP
  - Extrae support

Dimemas: Coarse grain trace driven simulator



Key factors influencing performance

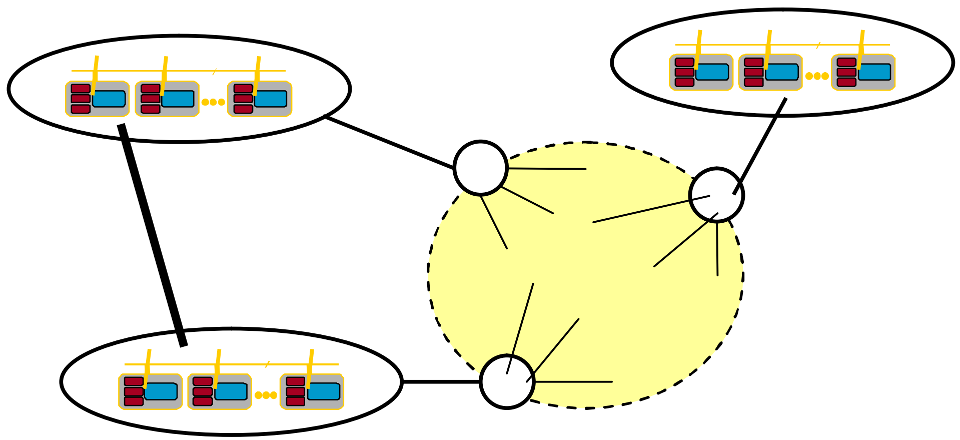
- Abstract architecture
- Basic MPI protocols
- No atempt to model details

$$T = \frac{MessageSize}{BW} + L$$

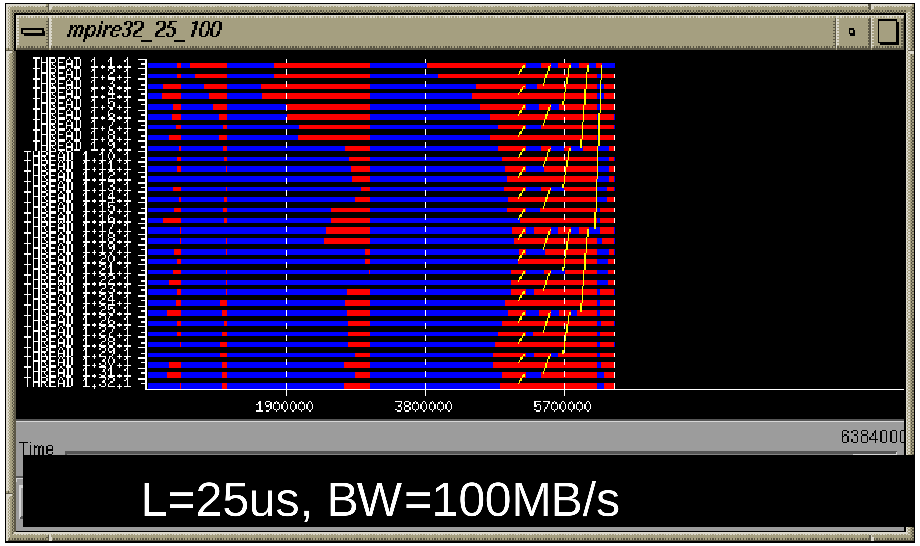
Objectives

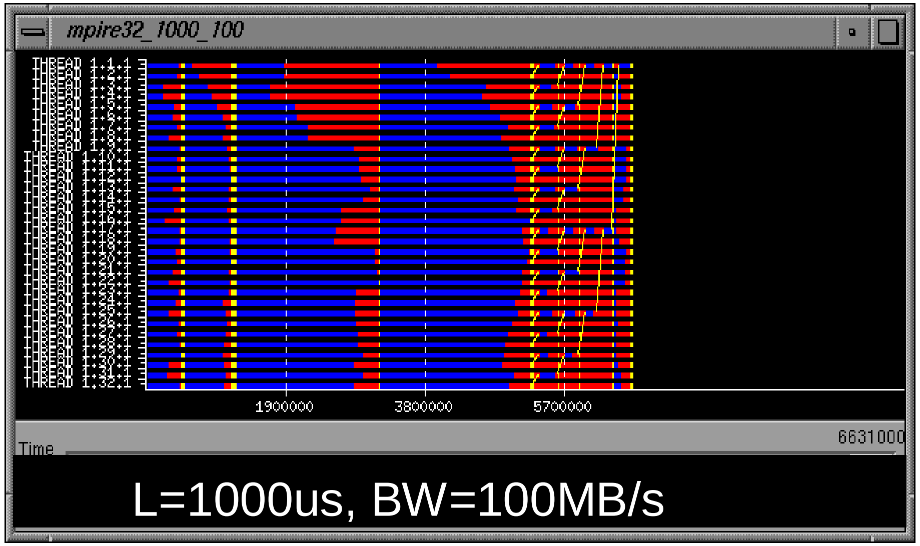
- Simple / general
- Fast simulations

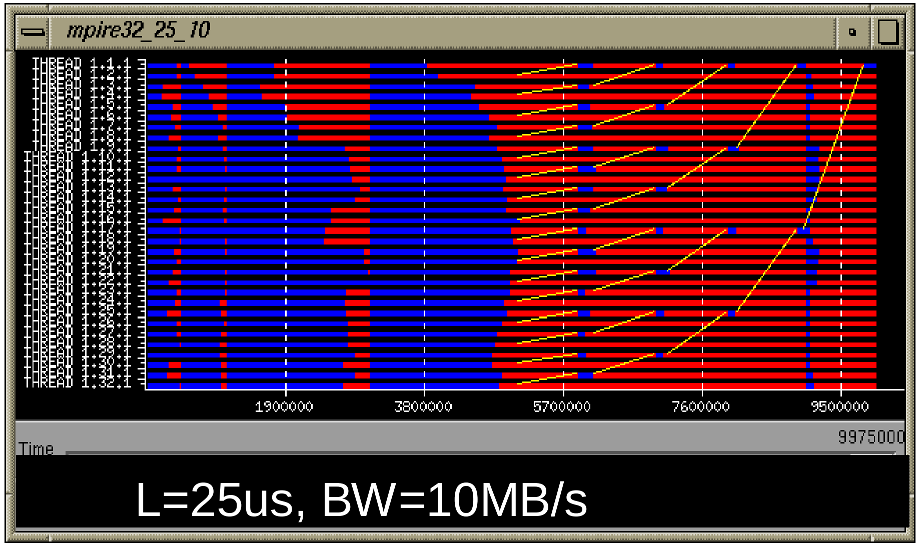
Network of SMPs / GRID



MPIRE 32 tasks, no network contention

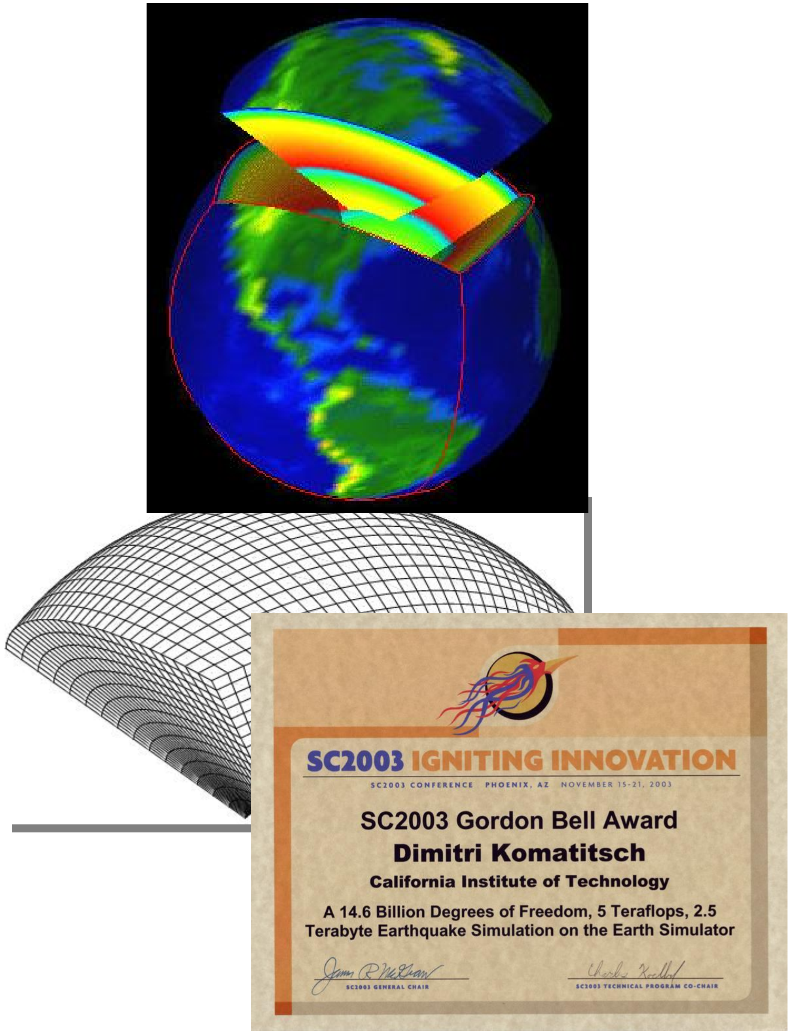


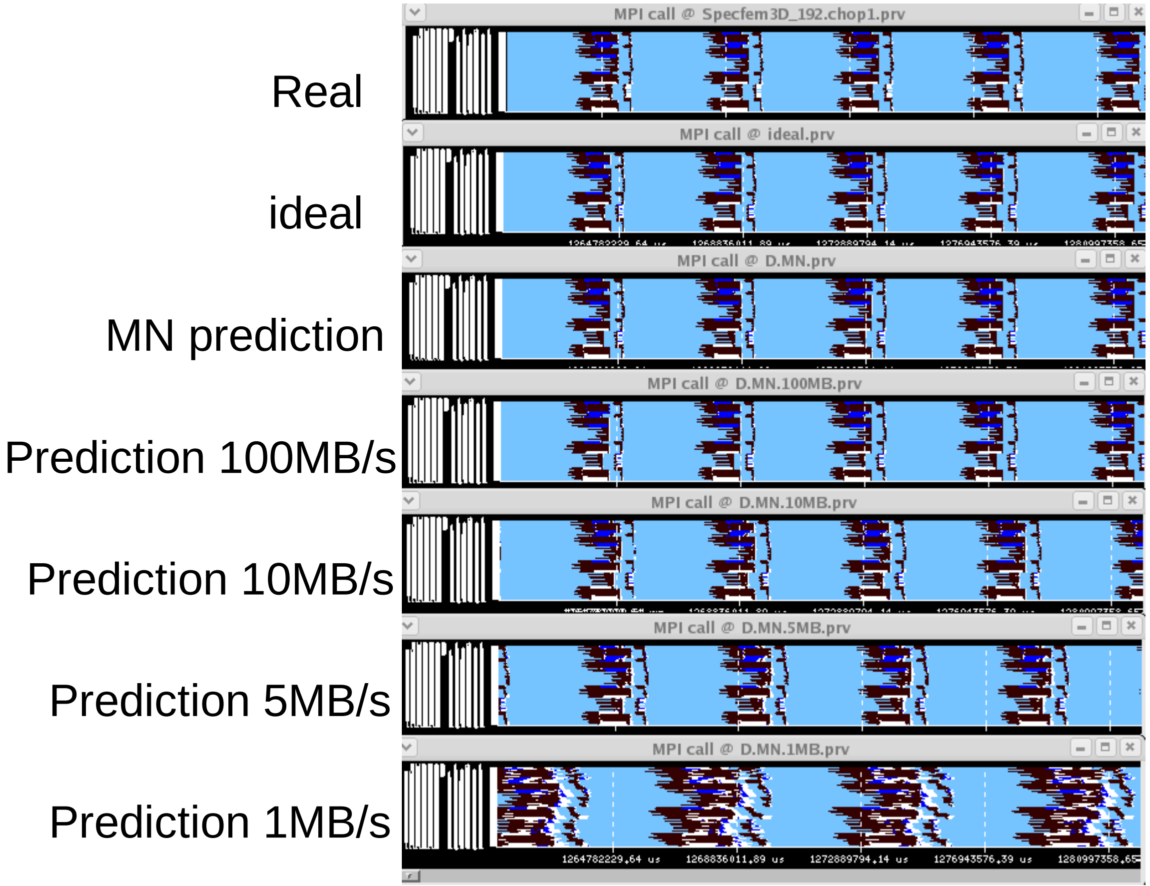




All windows same scale

SPECFEM3D – asynchronous communication?





Courtesy Dimitri Komatitsch

Parametric studies – network sensitivity

WRF, Iberia 4Km, 4 procs/node

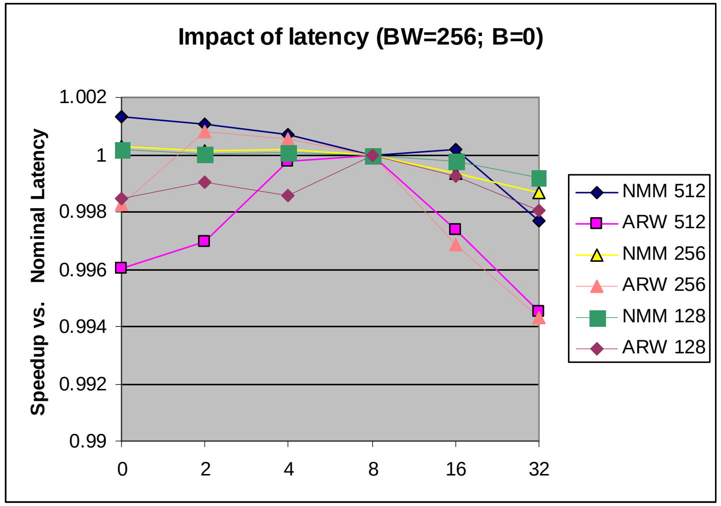
None sensitive to latency

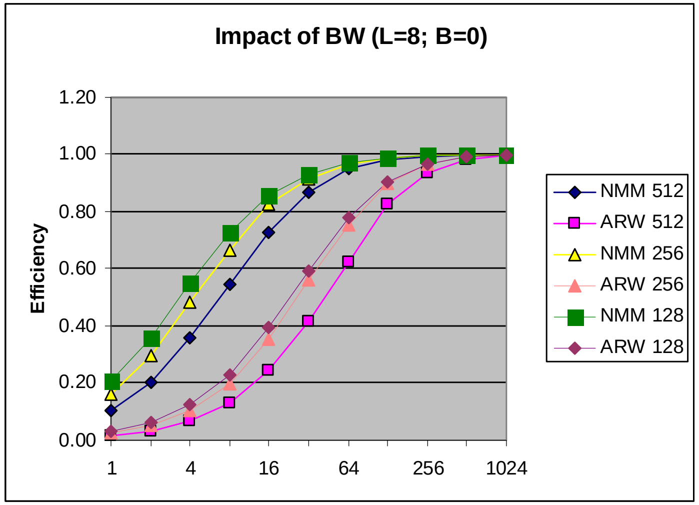
NMM

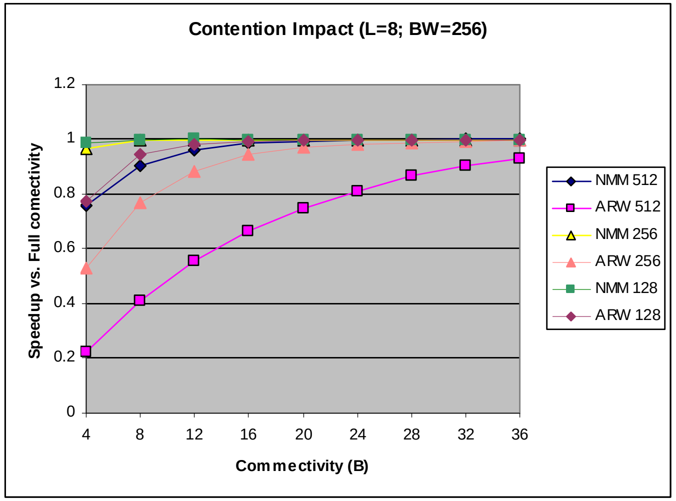
- Bw – 256MB/s
- 512 – sensitive to contention

ARW

- Bw - 1GB/s
- Sensitive to contention







Multiscale simulation: L2 size vs network bw

Left: clusters IPC with different cache sizes

- 64KB - 512MB

Right: execution time with different network bw and cache size

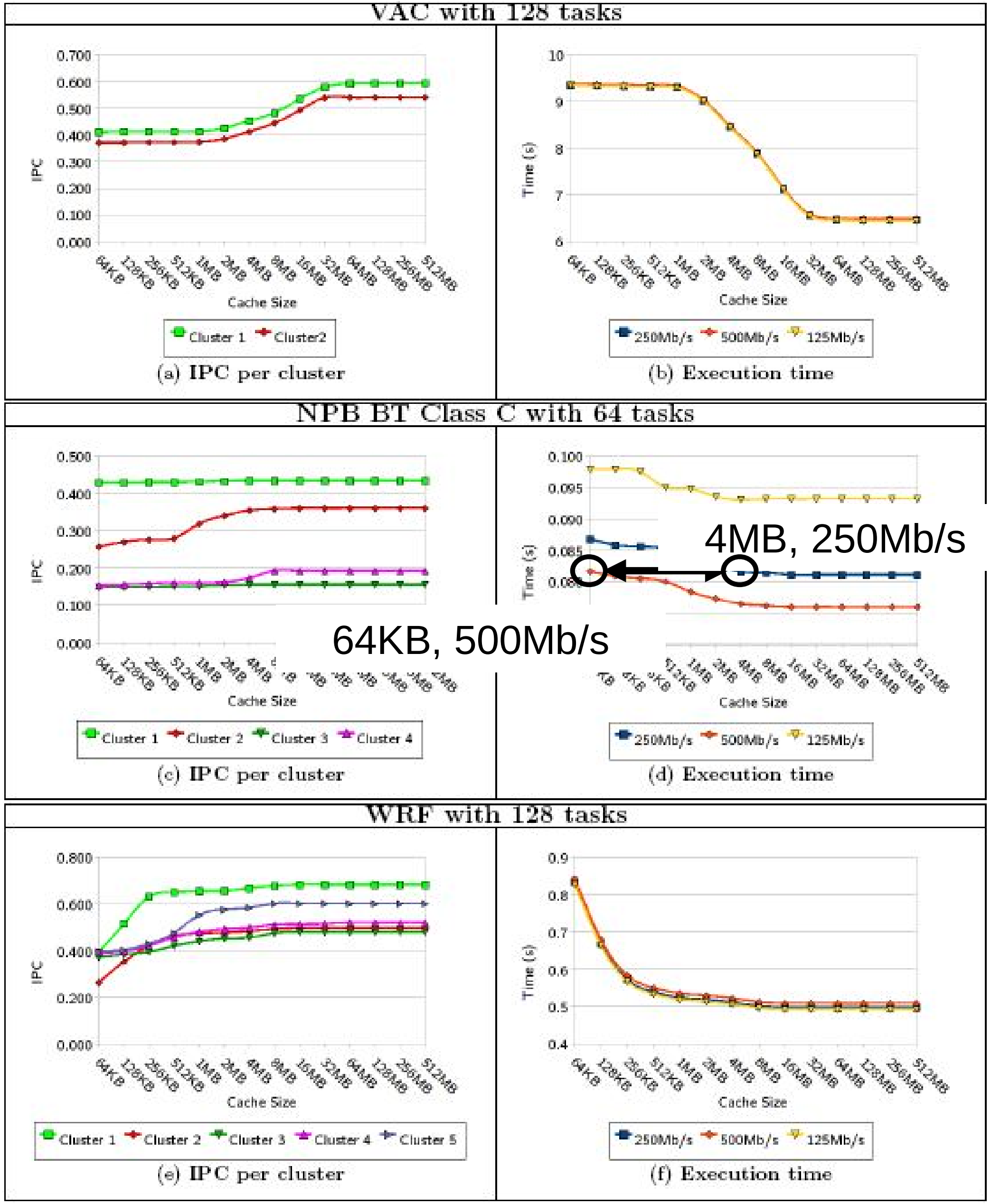
- 125Mb/s - 500Mb/s

NAS BT

- Can compensate cache reduction with more network bw

VAC, WRF

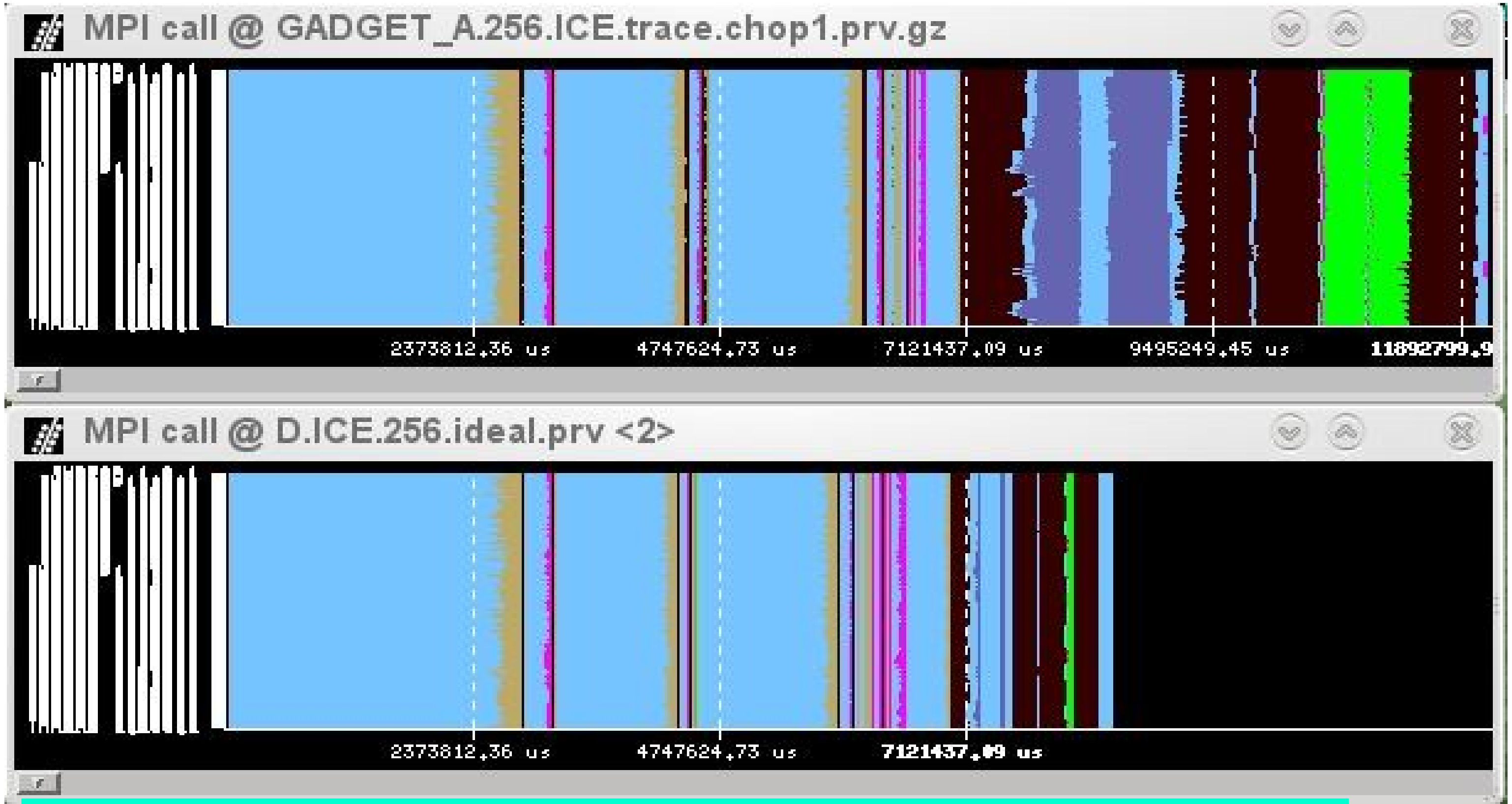
- Dominated by comp.



Fighting Amdahl's law

GADGET behavior: load balance / dependences?

Real run

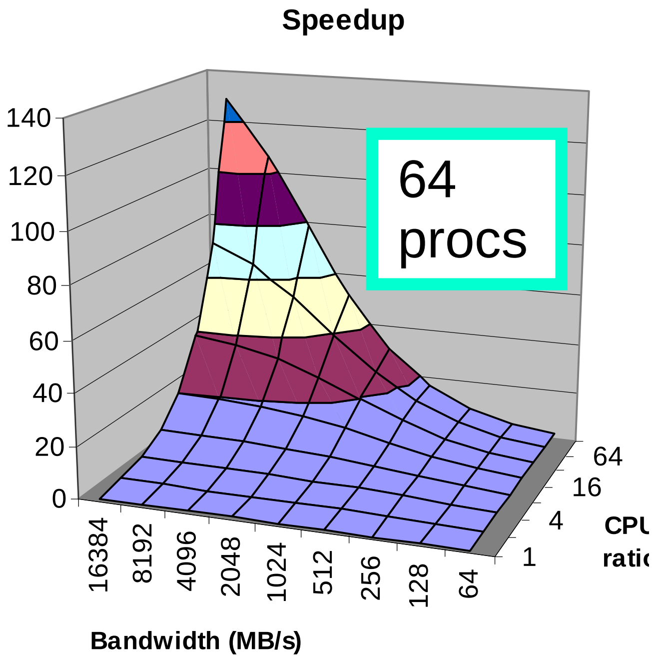


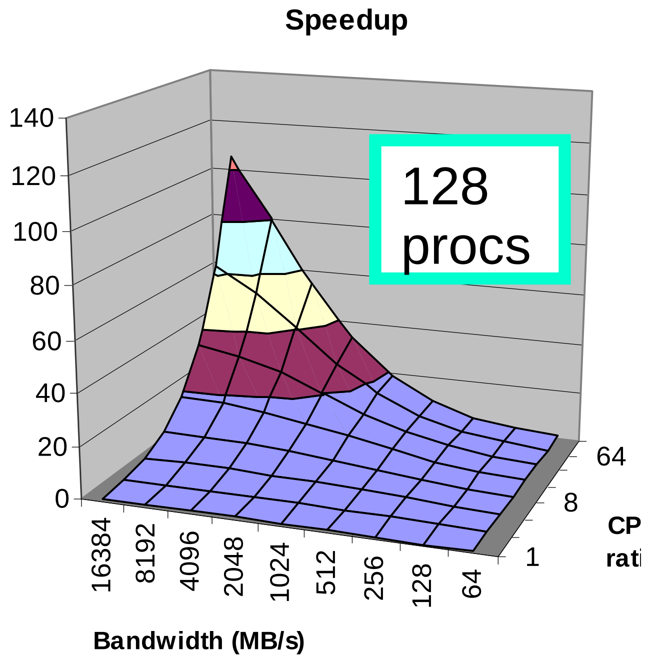
Ideal network: infinite bandwidth, no latency

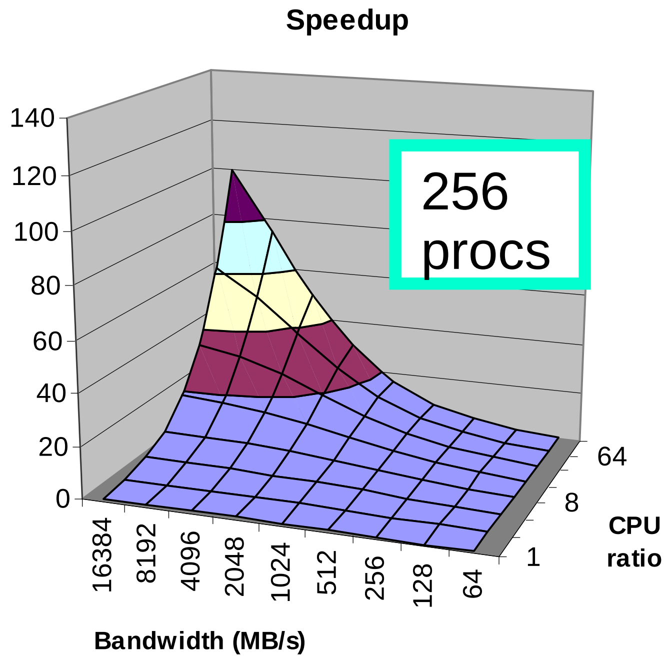
Fighting Amdahl's law

Speeding up **all** computation burst by CPU ratio

- The more processes, less speed up – higher impact of bw limitations



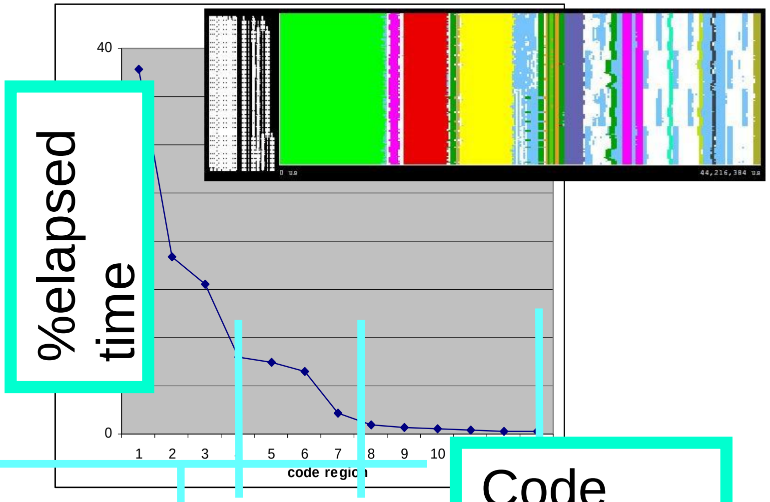


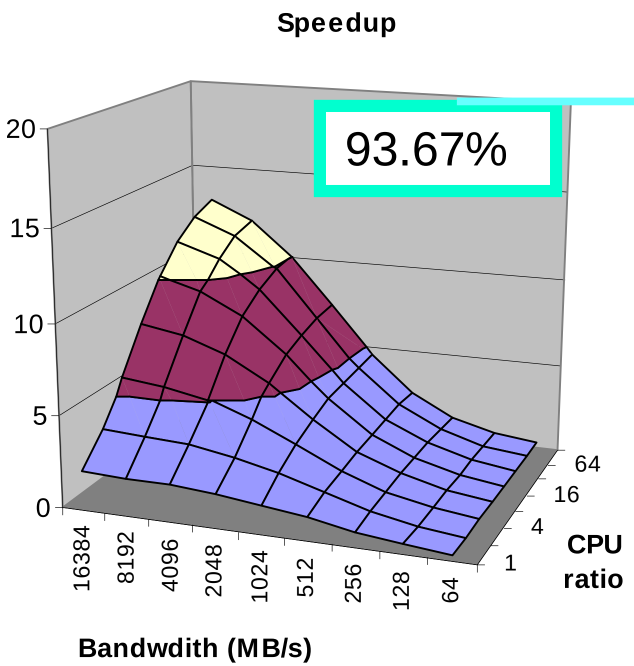


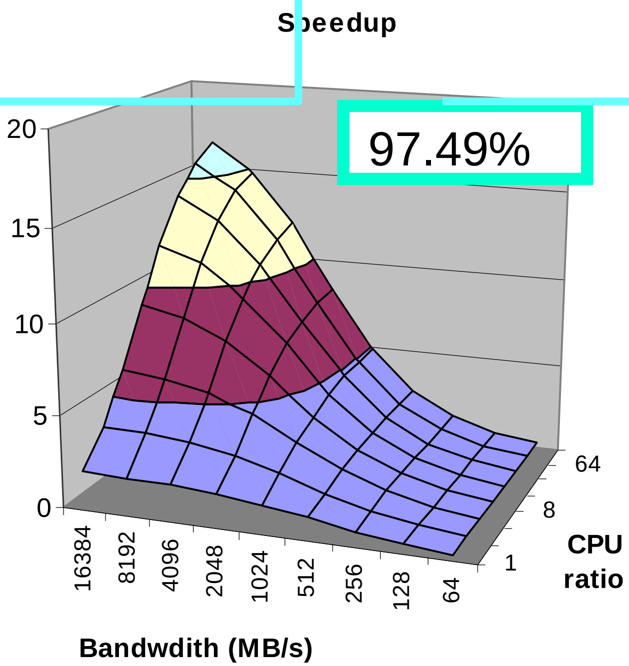
Fighting Amdahl's law

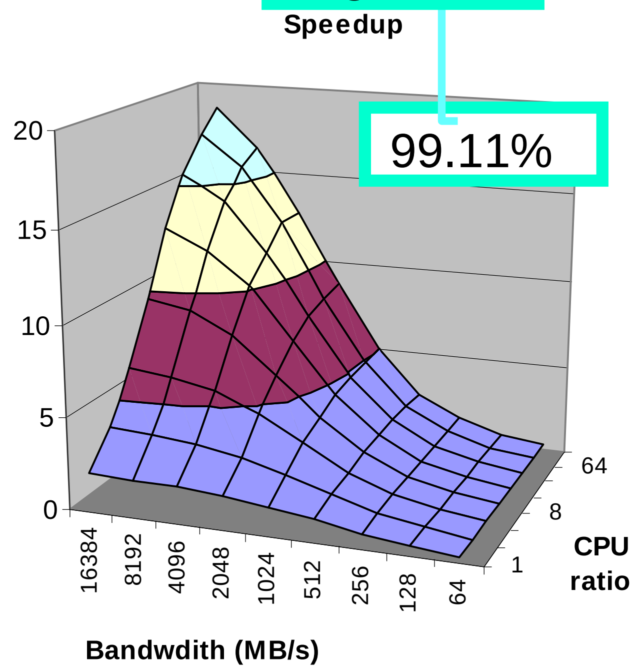
Hybrid parallelization (128 procs)

Speed up only selected regions by CPU ratio









Outline

- The tools
  - Introduction
  - Paraver
    - Applied research: clustering and sampling
    - Analysis examples
  - Dimemas
    - Simulation examples

- Some GPUs examples
  - GPUSs
  - HMPP
  - Extrae support

StarSs: a sequential program...

Understandable to a numerical analyst

```
void Cholesky( float *A ) {
   int i, j, k;
   for (k=0; k<NT; k++) {
      spotrf (A[k*NT+k]) ;
      for (i=k+1; i<NT; i++)
         strsm (A[k*NT+k], A[k*NT+i]);
      // update trailing submatrix
      for (i=k+1; i<NT; i++) {
         for (j=k+1; j<i; j++)
            sgemm( A[k*NT+i], A[k*NT+j], A[j*NT+i]);
         ssyrk (A[k*NT+i], A[i*NT+i]);
   }
}
```

```
void spotrf (float *A);

void ssyrk (float *A, float *C);

void sgemm (float *A, float *B, float *C);

void strsm (float *T, float *B);
```

StarSs: ... taskified...

Compiler translate source to runtime calls

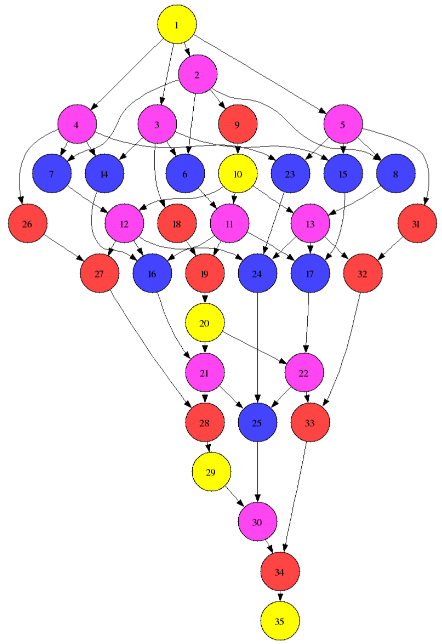
```
void Cholesky( float *A ) {
   int i, j, k;
   for (k=0; k<NT; k++) {
      spotrf (A[k*NT+k]) ;
      for (i=k+1; i<NT; i++)
         strsm (A[k*NT+k], A[k*NT+i]);
      // update trailing submatrix
      for (i=k+1; i<NT; i++) {
         for (j=k+1; j<i; j++)
            sgemm( A[k*NT+i], A[k*NT+j], A[j*NT+i]);
         ssyrk (A[k*NT+i], A[i*NT+i]);
   }
}
```

```
#pragma omp task inout ([TS][TS]A)
void spotrf (float *A);
#pragma omp task input ([TS][TS]A) inout ([TS][TS]C)
void ssyrk (float *A, float *C);
#pragma omp task input ([TS][TS]A,[TS][TS]B) inout ([TS][TS]C)
void sgemm (float *A, float *B, float *C);
#pragma omp task input ([TS][TS]T) inout ([TS][TS]B)
void strsm (float *T, float *B);
```

StarSs: ... and dynamically executed

Data flow model – graph generated at run time

```
void Cholesky( float *A ) {
   int i, j, k;
   for (k=0; k<NT; k++) {
      spotrf (A[k*NT+k]) ;
      for (i=k+1; i<NT; i++)
         strsm (A[k*NT+k], A[k*NT+i]);
      // update trailing submatrix
      for (i=k+1; i<NT; i++) {
         for (j=k+1; j<i; j++)
            sgemm( A[k*NT+i], A[k*NT+j], A[j*NT+i]);
         ssyrk (A[k*NT+i], A[i*NT+i]);
   }
}
```

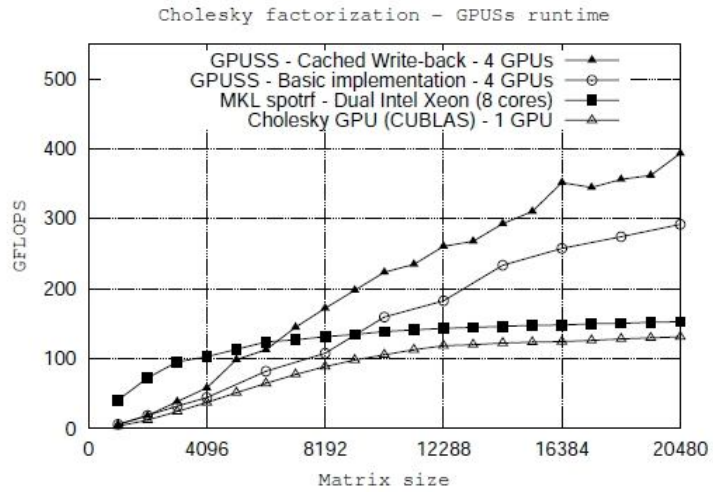


```
#pragma omp task inout ([TS][TS]A)
void spotrf (float *A);
#pragma omp task input ([TS][TS]A) inout ([TS][TS]C)
void ssyrk (float *A, float *C);
#pragma omp task input ([TS][TS]A,[TS][TS]B) inout ([TS][TS]C)
void sgemm (float *A, float *B, float *C);
#pragma omp task input ([TS][TS]T) inout ([TS][TS]B)
void strsm (float *T, float *B);
```

StarSs and accelerators

GPUSs

Cholesky @ 1-4 GPUs



Same source any target

- Possibly optimised tasks
- Different implementations

Transparent data transfer

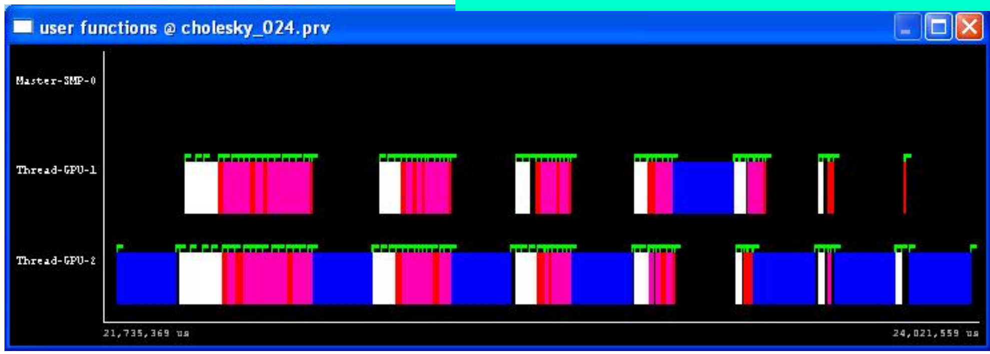
- Prefetch, double buffer
- Locality aware scheduler

```
#pragma css task inout (A[TS][TS])
void chol_spotrf (float *A);
```

```
#pragma css target device (cell, cuda) copy_deps
#pragma css task input (A[TS][TS], B[TS][TS}) inout (C[TS][TS])
void chol_sgemm (float *A, float *B, float *C);
```

Block matrix storage at CPU

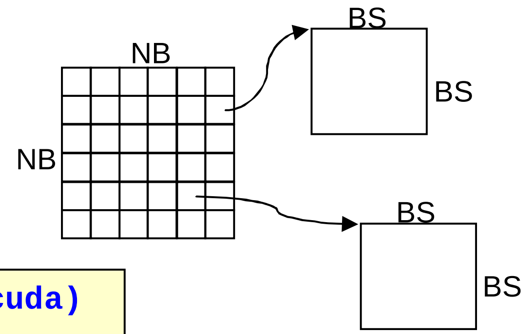
n = 8192; bs =1024



Source code independent of #devices

Spotrf: Slow task @ GPU - In critical path (scheduling problem)

```
void blocked_cholesky( int NT, float *A ) {
   int i, j, k;
   for (k=0; k<NT; k++) {
      spotrf (A[k*NT+k]) ;
      for (i=k+1; i<NT; i++)
         strsm (A[k*NT+k], A[k*NT+i]);
      // update trailing submatrix
      for (i=k+1; i<NT; i++) {
         for (j=k+1; j<i; j++)
            sgemm( A[k*NT+i], A[k*NT+j], A[j*NT+i]);
         ssyrk (A[k*NT+i], A[i*NT+i]);
   }
}
```



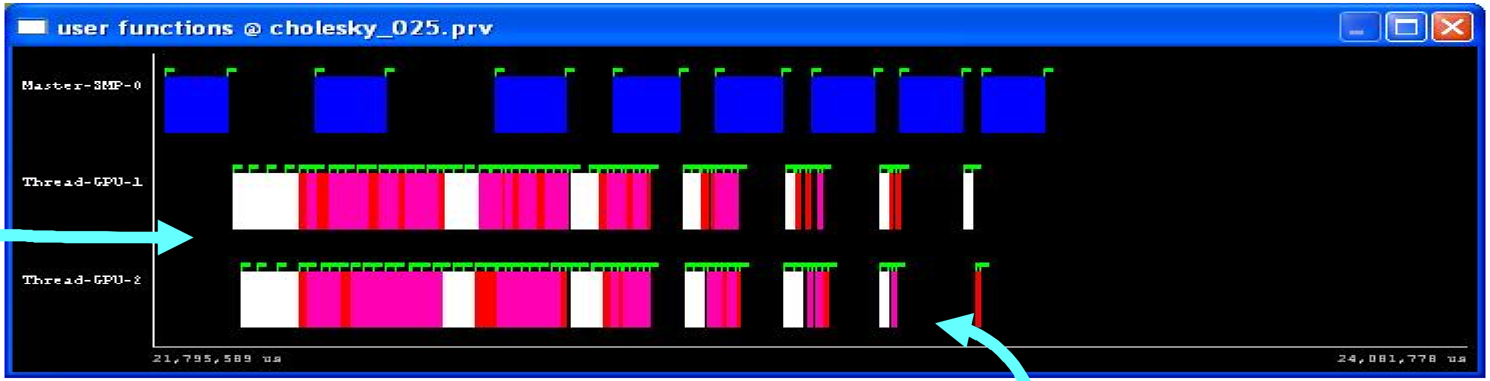

```
#pragma css task input([NB*NB]A, [NB*NB]B) inout([NB*NB]C) target device (cuda)
void sgemm(float  *A, float *B, float *C, unsigned long NB)
{
    unsigned char TR='T', NT='N';
    float DONE=1.0, DMONE=-1.0;
    cublasSgemm( NT, TR, NB, NB, NB, DMONE, A, NB, B, NB, DONE,C, NB );
}
```

Heterogeneus execution

Spotrf more efficient at CPU
Overlap between CPU and GPU

n = 8192; bs =1024

Late start



Lack of overlap

```
#pragma omp task inout([NB*NB]A) target device (smp)
void spotrf_tile(float *A, int NB)
{
    long INFO;
    char L = 'L';

    spotrf_( &L, &NB, A, &NB, &INFO );
}
```

Cholesky performance

Matrix size: 16K x 16K

- Block size: 2K x 2K
- Storage: Blocked / contiguous
- Tasks:
  - spotrf: Magma
  - trsm, syrk, gemm: CUBLAS

Two Intel Xeon E5440, 4 cores
4 Tesla S2050 GPUs

Poor scalability, why?

Comparing 2GPUs and 4GPUs

% running time significally decreases

2DP - Thread state @ cholesky_block_16Kx2K.2GPUs.prv

	Master-SMP-0	Thread-GPU-1	Thread-GPU-2
SHUTDOWN	-	-	-
ERROR	-	-	-
IDLE	66,18 %	4,72 %	4,31 %
RUNTIME	29,45 %	9,72 %	8,83 %
RUNNING	1,13 %	70,48 %	71,55 %
SYNCHRONIZATION	3,17 %	0,02 %	0,01 %
SCHEDULING	0,04 %	0,01 %	0,01 %
CREATION	0,02 %	-	-
DATA TRANSFER TO DEVICE	-	15,04 %	15,27 %
DATA TRANSFER TO HOST	0,01 %	0,00 %	0,00 %
LOCAL DATA TRANSFER IN DEVICE	-	-	-

RUNTIME

2DP - Thread state @ cholesky_block_16Kx2K.4GPUs.prv

	Master-SMP-0	Thread-GPU-1	Thread-GPU-2	Thread-GPU-3	Thread-GPU-4
SHUTDOWN	-	-	-	-	-
ERROR	-	-	-	-	-
IDLE	50,77 %	12,25 %	20,66 %	7,05 %	14,63 %
RUNTIME	45,07 %	13,62 %	11,71 %	14,18 %	11,61 %
RUNNING	0,05 %	46,11 %	45,29 %	50,99 %	51,40 %
SYNCHRONIZATION	4,05 %	0,02 %	0,01 %	0,02 %	0,01 %
SCHEDULING	0,02 %	0,01 %	0,00 %	0,01 %	0,01 %
CREATION	0,03 %	-	-	-	-
DATA TRANSFER TO DEVICE	-	27,97 %	22,31 %	27,73 %	22,32 %
DATA TRANSFER TO HOST	0,01 %	0,00 %	0,00 %	0,00 %	0,00 %
LOCAL DATA TRANSFER IN DEVICE	-	-	-	-	-

while average data transfer increases

2DP - Thread state @ cholesky_block_16Kx2K.2GPUs.prv

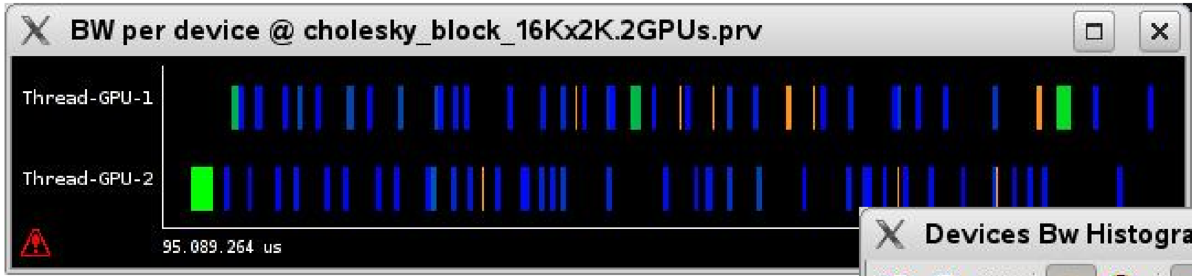
	Master-SMP-0	Thread-GPU-1	Thread-GPU-2
SHUTDOWN	-	-	-
ERROR	-	-	-
IDLE	20.619.067,57 ns	581.564,06 ns	574.129,30 ns
RUNTIME	1.593.852,11 ns	186.670,27 ns	177.580,28 ns
RUNNING	128.155,07 ns	26.207.403,23 ns	28.442.120,69 ns
SYNCHRONIZATION	1.462.820 ns	6.967,74 ns	5.758,62 ns
SCHEDULING	7.714,29 ns	8.259,26 ns	6.805,56 ns
CREATION	3.886,79 ns	-	-
DATA TRANSFER TO DEVICE	-	9.371.324,32 ns	9.029.515,62 ns
DATA TRANSFER TO HOST	5.972,22 ns	4.000 ns	4.000 ns
LOCAL DATA TRANSFER IN DEVICE	-	-	-

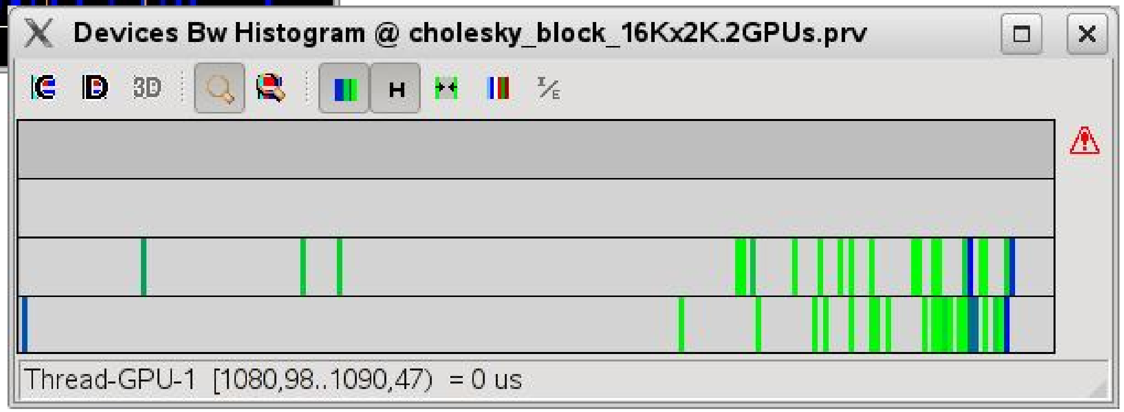
RUNTIME

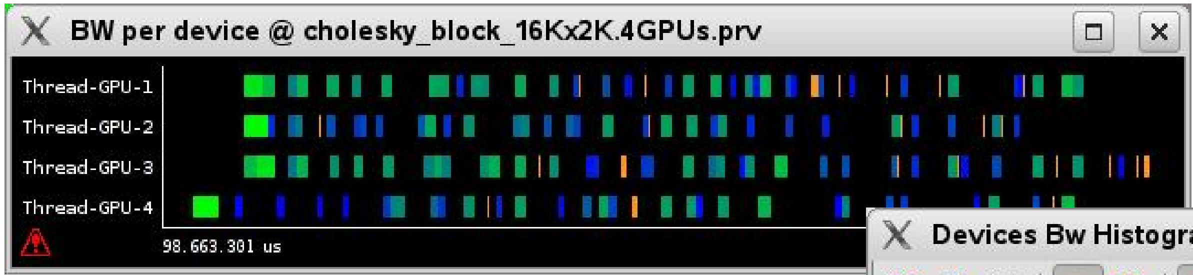
2DP - Thread state @ cholesky_block_16Kx2K.4GPUs.prv

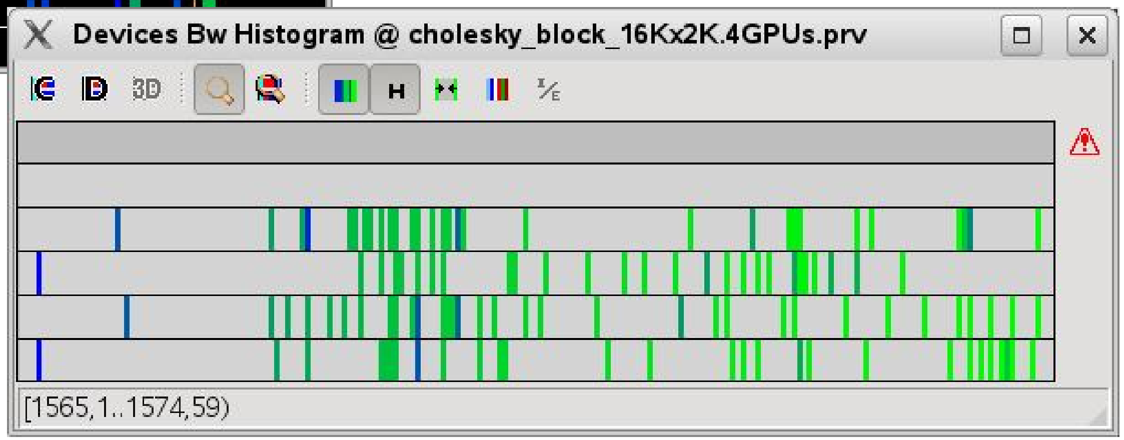
	Master-SMP-0	Thread-GPU-1	Thread-GPU-2	Thread-GPU-3	Thread-GPU-4
SHUTDOWN	-	-	-	-	-
ERROR	-	-	-	-	-
IDLE	12.752.000 ns	2.333.265,43 ns	4.068.342,66 ns	1.342.990,70 ns	2.756.242,79 ns
RUNTIME	2.018.561,59 ns	376.358,85 ns	343.966,10 ns	388.232,23 ns	318.495,25 ns
RUNNING	5.930,96 ns	26.636.066,67 ns	27.066.344,83 ns	29.455.166,67 ns	28.734.258,06 ns
SYNCHRONIZATION	1.798.128,21 ns	9.133,33 ns	6.586,21 ns	10.166,67 ns	7.225,81 ns
SCHEDULING	11.250 ns	9.217,39 ns	5.400 ns	5.409,09 ns	6.833,33 ns
CREATION	7.421,88 ns	-	-	-	-
DATA TRANSFER TO DEVICE	-	13.466.916,67 ns	12.470.903,23 ns	13.350.527,78 ns	12.477.187,66 ns
DATA TRANSFER TO HOST	7.441,18 ns	5.500 ns	5.400 ns	5.000 ns	5.000 ns
LOCAL DATA TRANSFER IN DEVICE	-	-	-	-	-

Comparing 2GPUs and 4GPUs





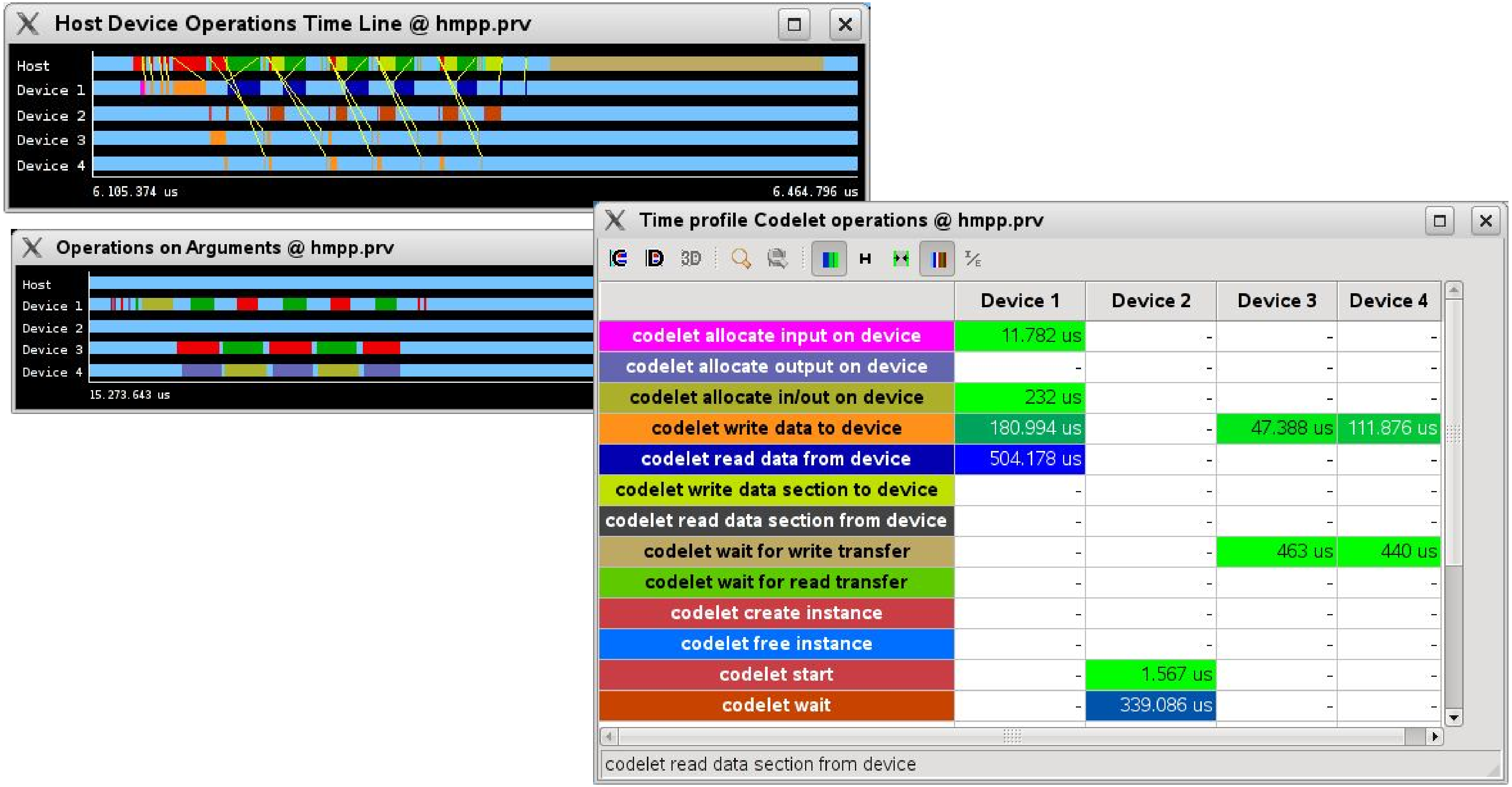




Many data transfers achieve a lower BW with 4 GPUs

HMPP traces

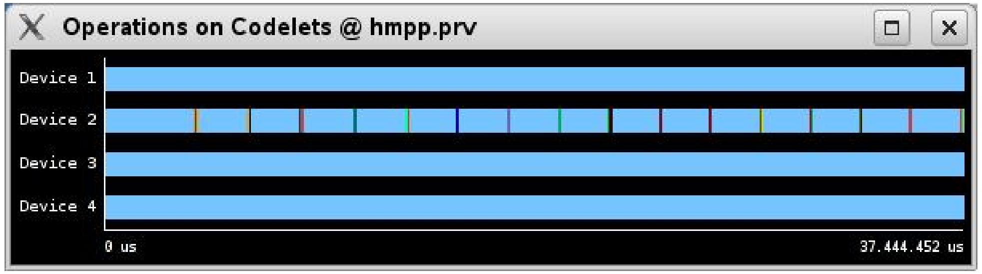
Same analysis tool, different programming model, different concepts

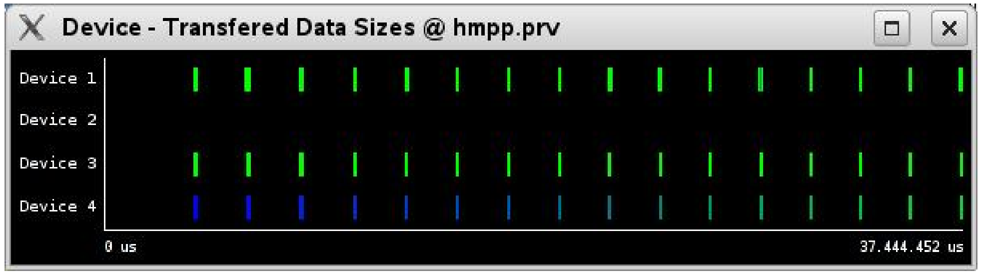


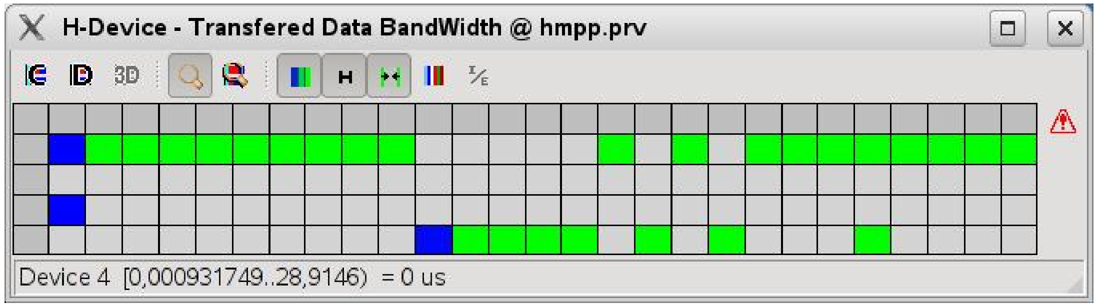
	Device 1	Device 2	Device 3	Device 4
codelet allocate input on device	11.782 us	-	-	-
codelet allocate output on device	-	-	-	-
codelet allocate in/out on device	232 us	-	-	-
codelet write data to device	180.994 us	-	47.388 us	111.876 us
codelet read data from device	504.178 us	-	-	-
codelet write data section to device	-	-	-	-
codelet read data section from device	-	-	-	-
codelet wait for write transfer	-	-	463 us	440 us
codelet wait for read transfer	-	-	-	-
codelet create instance	-	-	-	-
codelet free instance	-	-	-	-
codelet start	-	1.567 us	-	-
codelet wait	-	339.086 us	-	-

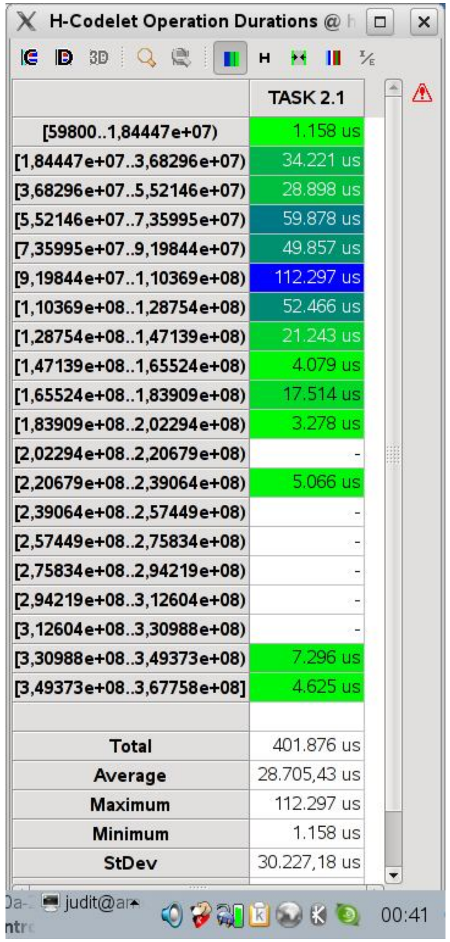
Analysis nethodology predefined

- CAPS distribute it with a complete set of views
  - If sequenced, can be used as an exploratory tree



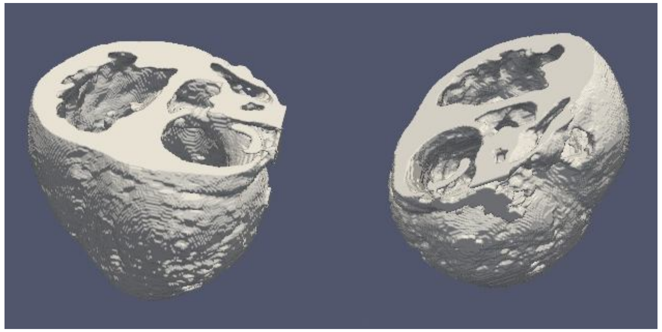


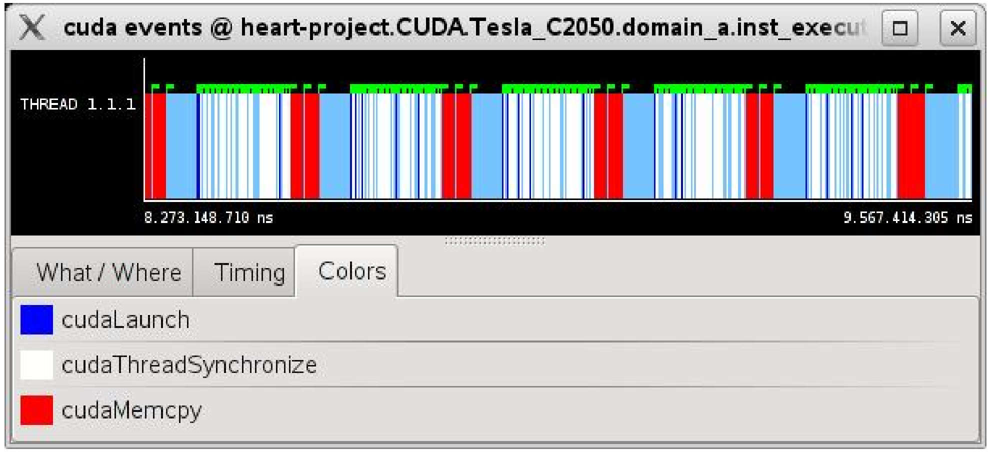


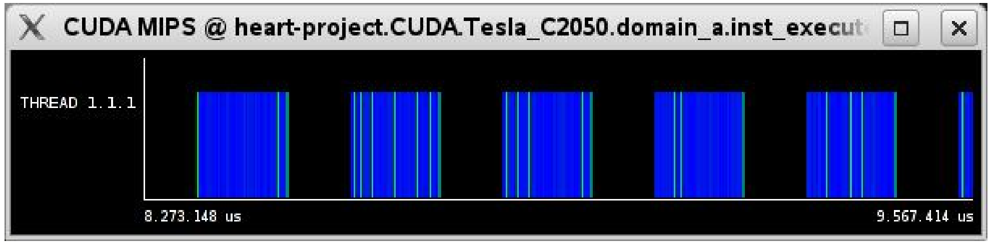


Extrae support to CUDA

- Just starting
- Support to BSC/CASE department
  - Propagation of an electrical shock wave across the cardiac tissue
- Captured information
  - **From CPU perspective**
    - CUDA runtime calls
    - PAPI CUDA counters

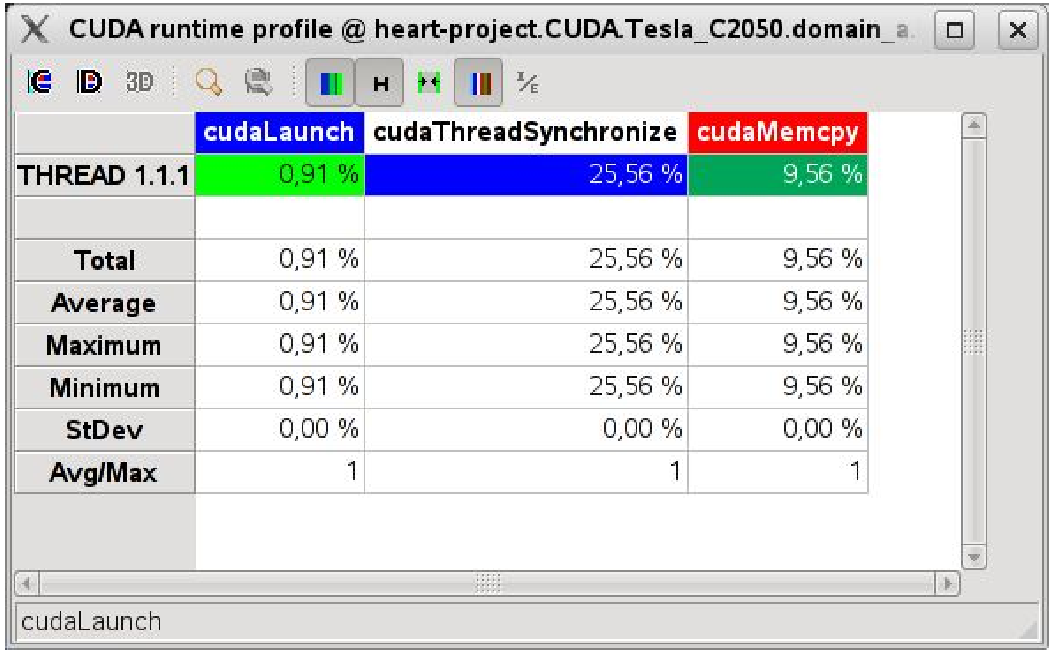






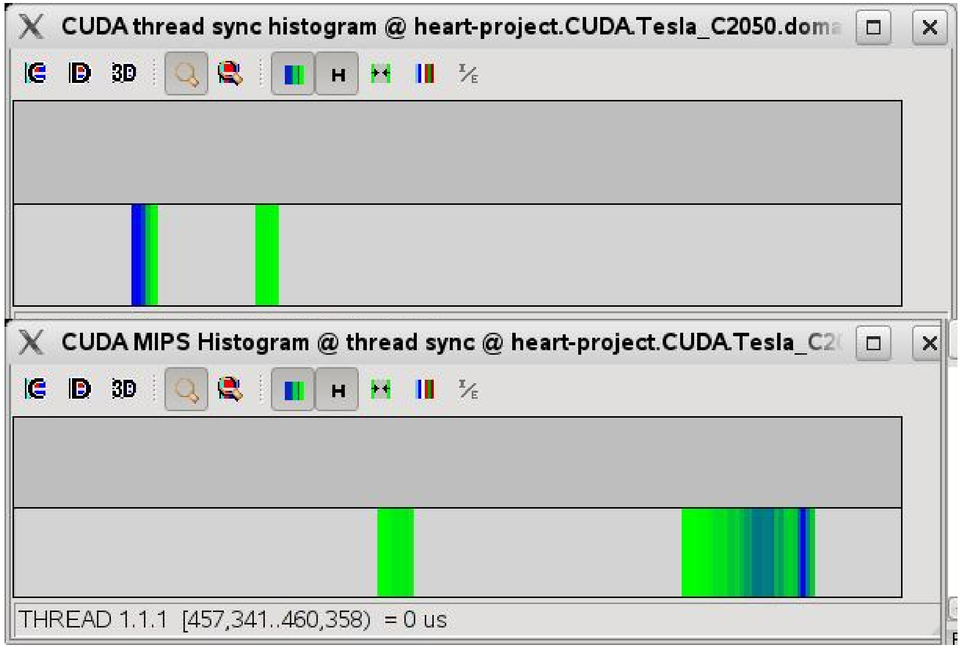
Example of CUDA analysis

CUDA runtime profile

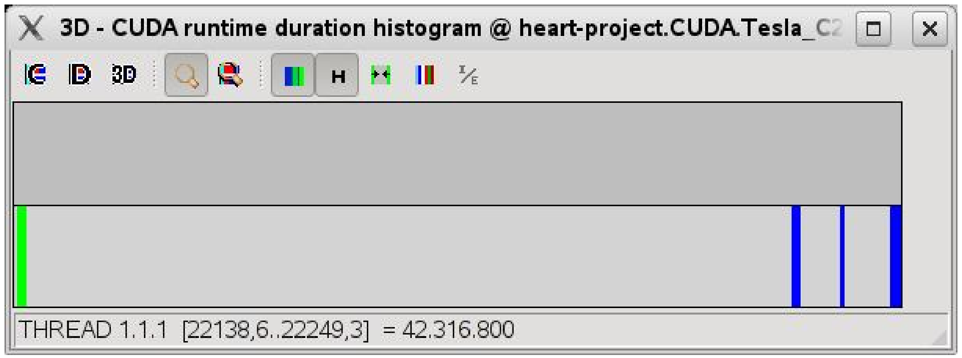


	cudaLaunch	cudaThreadSynchronize	cudaMemcpy
THREAD 1.1.1	0,91 %	25,56 %	9,56 %
Total	0,91 %	25,56 %	9,56 %
Average	0,91 %	25,56 %	9,56 %
Maximum	0,91 %	25,56 %	9,56 %
Minimum	0,91 %	25,56 %	9,56 %
StDev	0,00 %	0,00 %	0,00 %
Avg/Max	1	1	1

CUDA ThreadSync histogram



CUDA Memcpy histogram vs size



Conclusions

- The complexity of heterogeneous systems pushes the need for tools
- Same tools and similar methodology maybe used for very different scenarios
- Do not speculate about your code performance behaviour, look at it

www.bsc.es/paraver