www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

On the Instrumentation of OpenMP and OmpSS Tasking Constructs

H.Servat, X.Teruel, G. Llort, A.Duran, J.Giménez, X.Martorell, E. Ayguadé and J. Labarta

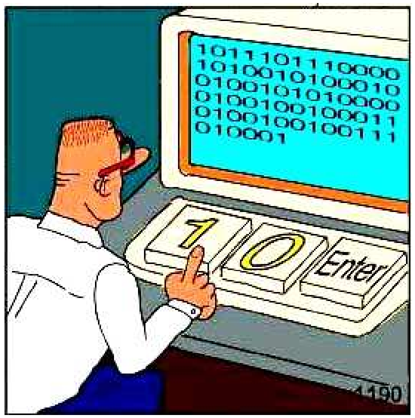
Supercomputers today

Top500 (June 2012)

Rank	Site	Computer/Year Vendor	Cores
1	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom / 2011 IBM	1572864
2	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu	705024
3	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM	786432
4	Leibniz Rechenzentrum Germany	**SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR / 2012 IBM	147456
5	National Supercomputing Center in Tianjin China	**Tianhe-1A** - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 / 2010 NUDT	186368
6	DOE/SC/Oak Ridge National Laboratory United States	**Jaguar** - Cray XK6, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA 2090 / 2009 Cray Inc.	298592
7	CINECA Italy	**Fermi** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM	163840
8	Forschungszentrum Juelich (FZJ) Germany	**JuQUEEN** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM	131072
9	CEA/TGCC-GENCI France	**Curie thin nodes** - Bullx B510, Xeon E5-2680 8C 2.700GHz, Infiniband QDR / 2012 Bull	77184
10	National Supercomputing Centre in Shenzhen (NSCS) China	**Nebulae** - Dawning TC3600 Blade System, Xeon X5650 6C 2.66GHz, Infiniband QDR, NVIDIA 2050 / 2010 Dawning	120640

Parallel programming is getting (even) harder

- More parallelism needed because of large core count
- Heterogenity brings an additional complexity layer



Think in serial

Execute in parallel

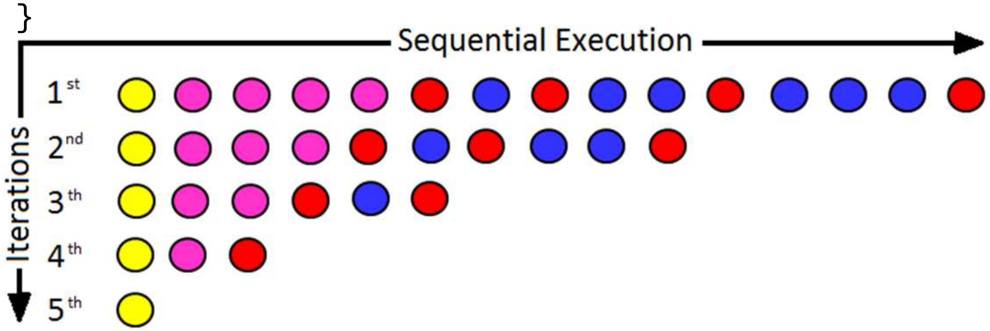
Cholesky (Serial)

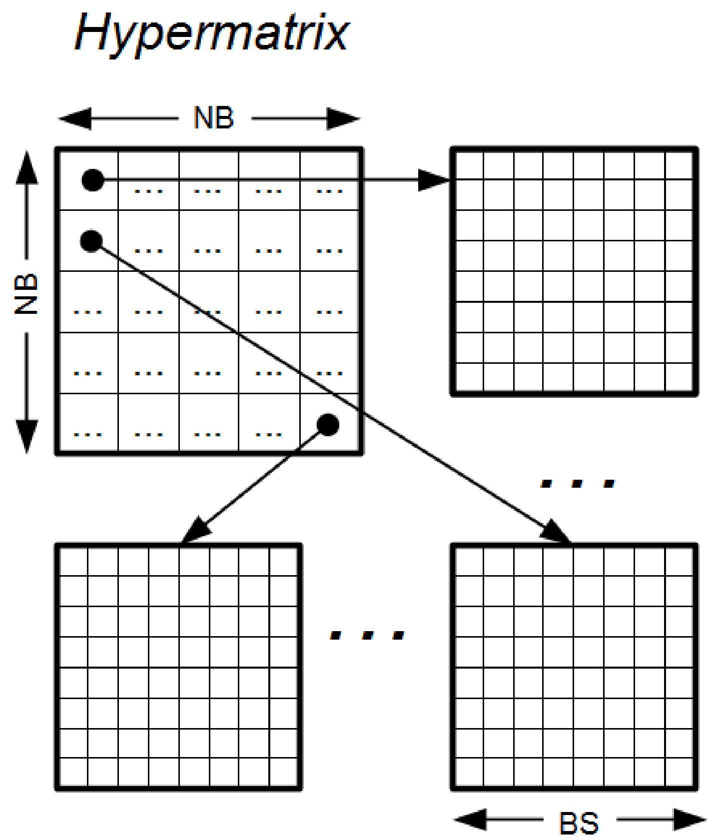
```
void Cholesky( float *A, int NB) {
  int i, j, k;
  for (k=0; k<NB; k++) {
    ● spotrf (A[k*NB+k]);
    for (i=k+1; i<NB; i++)
      ● strsm (A[k*NB+k], A[k*NB+i]);
    for (i=k+1; i<NB; i++) {
      for (j=k+1; j<i; j++)
        ● sgemm( A[k*NB+i], A[k*NB+j], A[j*NB+i]);
      ● ssyrk (A[k*NB+i],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

OUT: A[0]

IN: A[0]

OUT: A[1] … A[4]

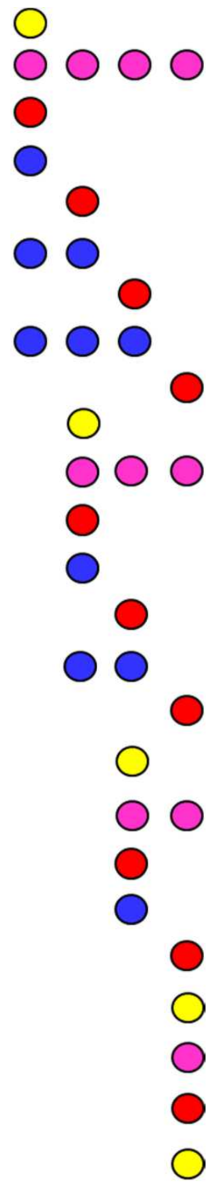




- Iters = 5, Critical path = 35

Cholesky (OpenMP 2.5)

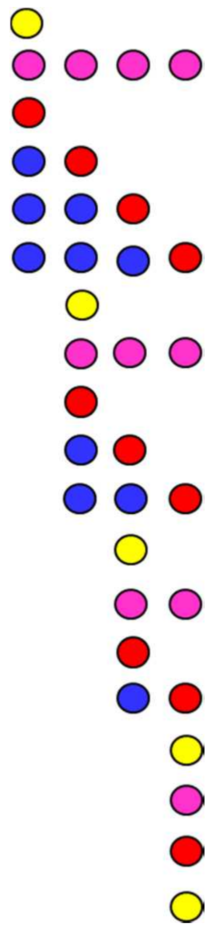
```
void Cholesky( float *A ) {
  int NB, i, j, k;
  for (k=0; k<NB; k++) {
    • spotrf (A[k*NB+k]);
#pragma omp parallel for
    for (i=k+1; i<NB; i++)
      • strsm (A[k*NB+k], A[k*NB+i]);
    for (i=k+1; i<NB; i++) {
#pragma omp parallel for
      for (j=k+1; j<i; j++)
        • sgemm( A[k*NB+i], A[k*NB+j], A[j*NB+i]);
      • ssyrk (A[k*NB+i], A[i*NB+i]);
    }
  }
}
```



- Iters = 5, Critical path = 25

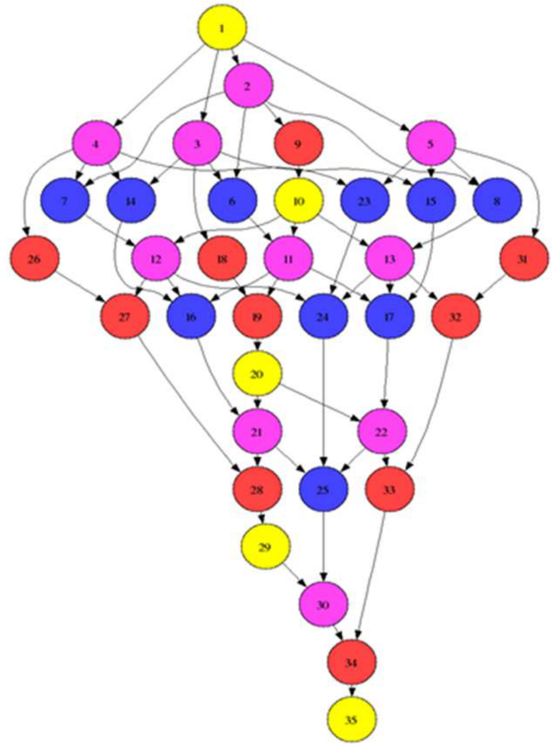
Cholesky (OpenMP 3.0)

```
void Cholesky( float *A ) {
  int NB, i, j, k;

  for (k=0; k<NB; k++) {
    ● spotrf (A[k*NB+k]);
#pragma omp parallel for
    for (i=k+1; i<NB; i++)
      ● strsm (A[k*NB+k], A[k*NB+i]);
    for (i=k+1; i<NB; i++) {
      for (j=k+1; j<i; j++)
#pragma omp task
        ● sgemm( A[k*NB+i], A[k*NB+j], A[j*NB+i]);
#pragma omp task
    ● ssyrk (A[k*NB+i], A[i*NB+i]);
#pragma omp taskwait
    }
  }
}
```



- Iters = 5, Critical path = 19

```
void Cholesky( float *A ) {
  int NB, i, j, k;

  for (k=0; k<NB; k++) {
#pragma omp task inout(A[k*NB+k])
    • spotrf (A[k*NB+k]);
    for (i=k+1; i<NB; i++)
#pragma omp task input(A[k*NB+k]) inout(A[k*NB+i])
      • strsm (A[k*NB+k], A[k*NB+i]);
    for (i=k+1; i<NB; i++) {
      for (j=k+1; j<i; j++)
#pragma omp task input(A[k*NB+i], A[k*NB+j]) \
                 inout(A[j*NB+i])
        • sgemm( A[k*NB+i], A[k*NB+j], A[j*NB+i]);
#pragma omp task input (A[k*NB+i]) inout(A[i*NB+i])
      • ssyrk (A[k*NB+i], A[i*NB+i]);
    }
  }
}
```



- Iters = 5, Critical path = **13** (33% shorter than OpenMP)

OpenMP vs. OmpSs

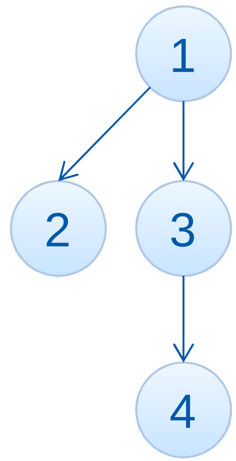
OpenMP – You say how to parallelize

- Based in compiler directives
- Worksharing constructs (<= 2.5): do/for loops, sections
- Tasking constructs (>= 3.0): task, taskwait, taskyield
  - While, linked lists, recursivity

OmpSs – You say how data is used ➔ System does the rest

- Extends OpenMP
- Focus on tasks
- Heterogeneity
  - SMP, GPU
- Data dependences

```
#pragma omp task out(x)        // 1
x = 5;
#pragma omp task in(x)         // 2
printf("%d\n" , x ) ;
#pragma omp task in(x) out(y)  // 3
y = x + 1;
#pragma omp task in(y)         // 4
printf ("%d\n" , y ) ;
```



Performance analysis – CEPBA Tools

One image worths a thousand words.

Do not speculate about your code perfomance.

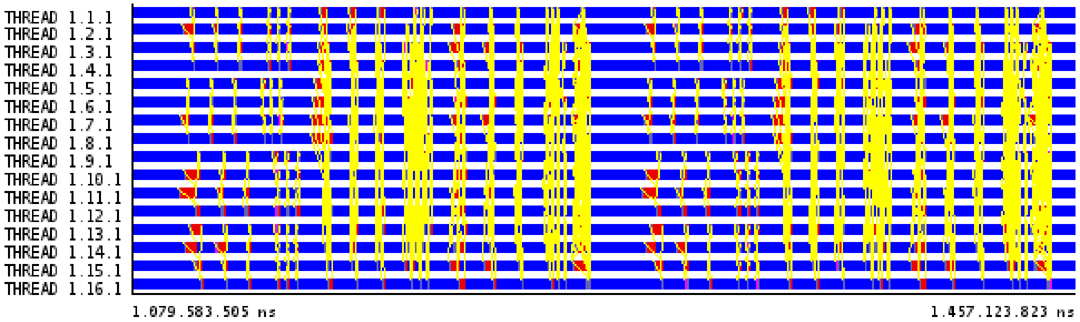
Look at it.

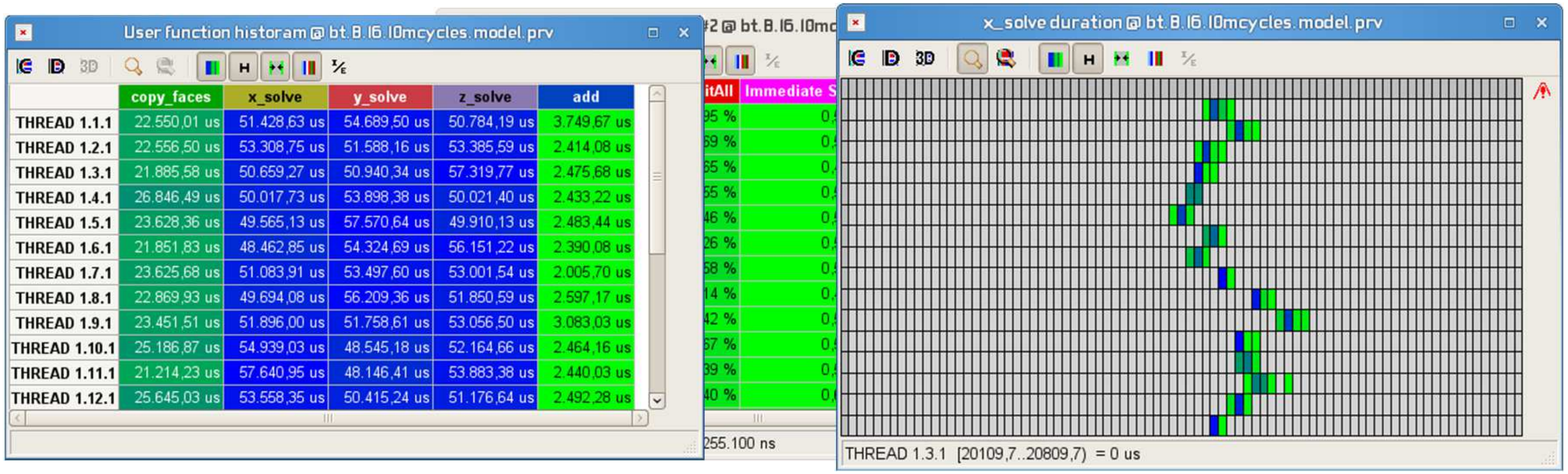
- Since 1991
- Based on traces
- Open-source
- Core Tools
  - Extrae – Trace generation
  - Paraver – Trace analyzer
  - Dimemas– Message passing simulator
- Detail and intelligence

Extrae – Trace generation

- Parallel programming model runtime
  - MPI, OpenMP, Pthreads, CUDA…
- Counters
  - CPU – PAPI and PMAPI
  - Network – Myrinet (GM and MX)
  - OS – Memory allocation, resource usage
- Links to source
  - Callstack at MPI calls
  - User functions selected (default none)
- Periodic samples
  - PAPI counters + callstack
- User events

Paraver – Trace analyzer





	copy_faces	x_solve	y_solve	z_solve	add
THREAD 1.1.1	22.550,01 us	51.428,63 us	54.689,50 us	50.784,19 us	3.749,67 us
THREAD 1.2.1	22.556,50 us	53.308,75 us	51.588,16 us	53.385,59 us	2.414,08 us
THREAD 1.3.1	21.885,58 us	50.659,27 us	50.940,34 us	57.319,77 us	2.475,68 us
THREAD 1.4.1	26.846,49 us	50.017,73 us	53.898,38 us	50.021,40 us	2.433,22 us
THREAD 1.5.1	23.628,36 us	49.565,13 us	57.570,64 us	49.910,13 us	2.483,44 us
THREAD 1.6.1	21.851,83 us	48.462,85 us	54.324,69 us	56.151,22 us	2.390,08 us
THREAD 1.7.1	23.625,68 us	51.083,91 us	53.497,60 us	53.001,54 us	2.005,70 us
THREAD 1.8.1	22.869,93 us	49.694,08 us	56.209,36 us	51.850,59 us	2.597,17 us
THREAD 1.9.1	23.451,51 us	51.896,00 us	51.758,61 us	53.056,50 us	3.083,03 us
THREAD 1.10.1	25.186,87 us	54.939,03 us	48.545,18 us	52.164,66 us	2.464,16 us
THREAD 1.11.1	21.214,23 us	57.640,95 us	48.146,41 us	53.883,38 us	2.440,03 us
THREAD 1.12.1	25.645,03 us	53.558,35 us	50.415,24 us	51.176,64 us	2.492,28 us

Synergy

What?

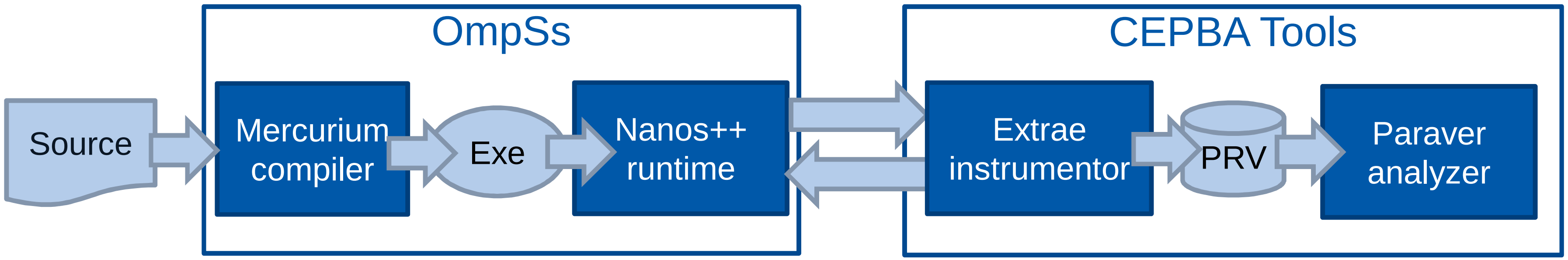
- OmpSs + CEPBA Tools

Why?

- Improve the analysis adding data only known by the parallel runtime
  - Ready task, blocked task, spins, sleeps, yields, etc.
- Facilitate the development – analysis – optimization cycle

How?

- Implement services to communicate both suites



Code transformations

OmpSs in several flavours

- Performance
  - Production runs
- Instrumentation
  - Logs information at compile and run stages

```
{
  code 1
  #pragma omp task
  {
    code 2
  }
  code 3
}
```

User source code

```
code2OL
{
  code 2
}

{
  code 1
  t = nanos_create_task (code2OL);
  nanos_submit_task (t);
  code 3
}
```

Mercurium translation (Performance)

Mercurium translation (Instrumented)

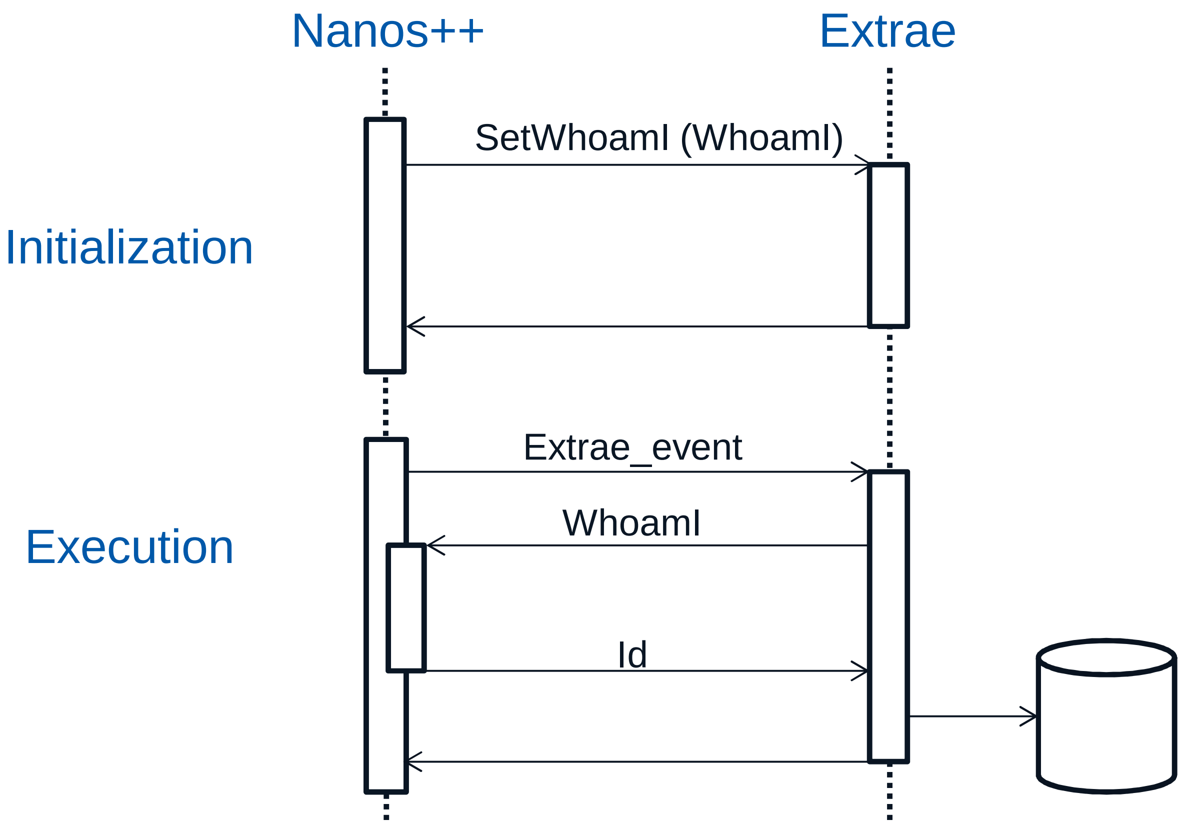
```
nanos_create_task
{
  // Task creation
}

nanos_submit_task
{
  // Queue task into "run" queue
}
```

Nanos++ RTL (Performance)

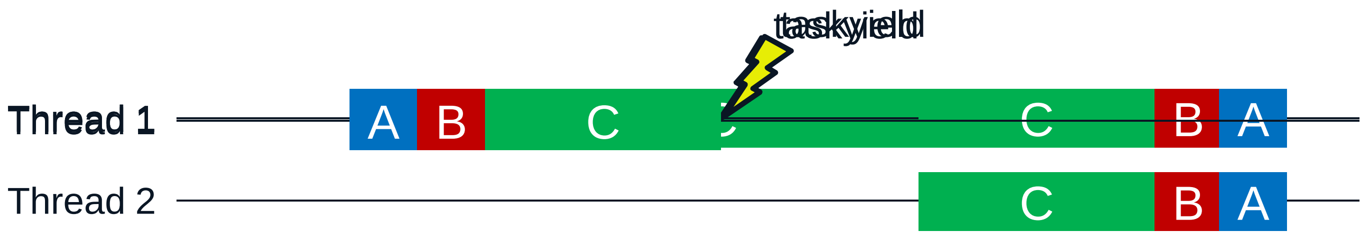
Nanos++ RTL (Instrumented)

Thread identification

- Extrae needs to know which thread is emitting the event
  - Leveraged to Nanos++ via callback

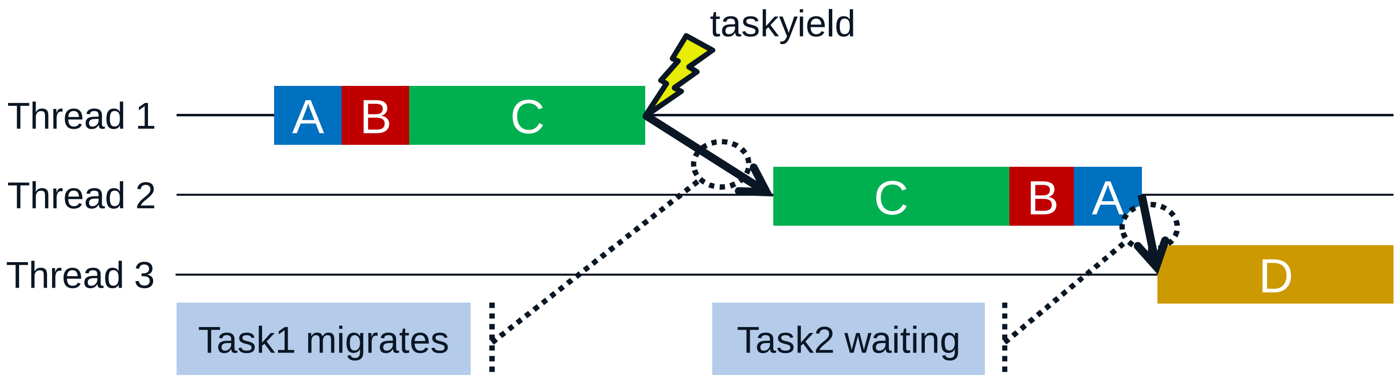


Task suspension, resumption & migration



- Each task keeps an instrumentation context
  - Includes information to be backed up and restored at schedule points
    - Routine being executed, task identifier, task state…
- When the task migrates, also does the context
  - Enables support for untied tasks

Tasks relations



Draw lines relating

- Task migrations
- Data dependences
- Others: taskwait dependences, data transfers…

Useful to study

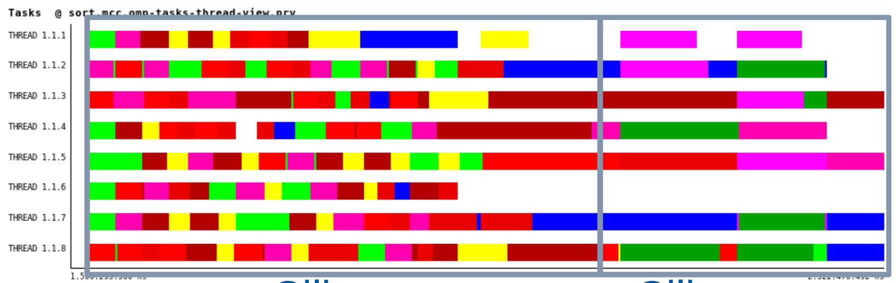
- Execution/Critical path
- Scheduling policies

Visualization

Thread-view (standard)

- Determine thread usage
- Shows task migrations

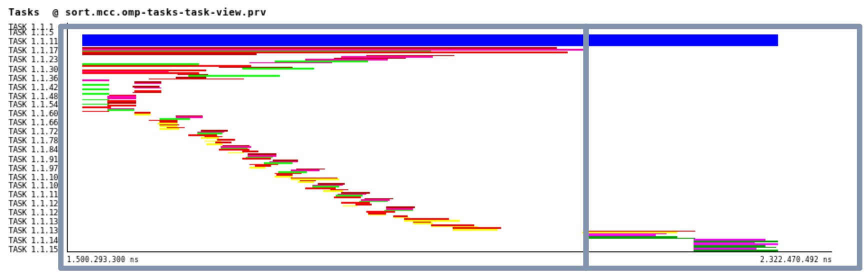
Recursive multisort



Cilk_sort Cilk_merge

Task-view (new)

- Displays task liveness, logical order and dependences (critical path)

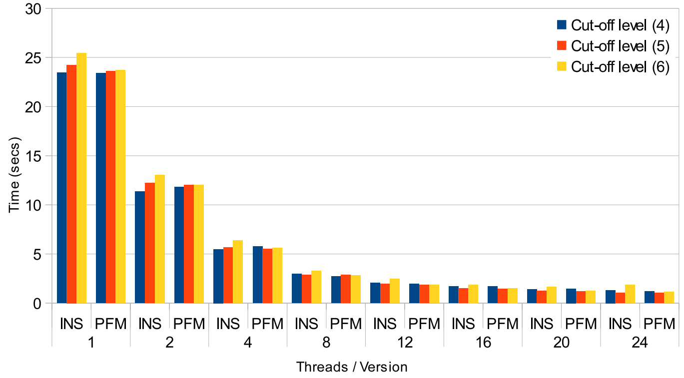


Cilk_sort Cilk_merge

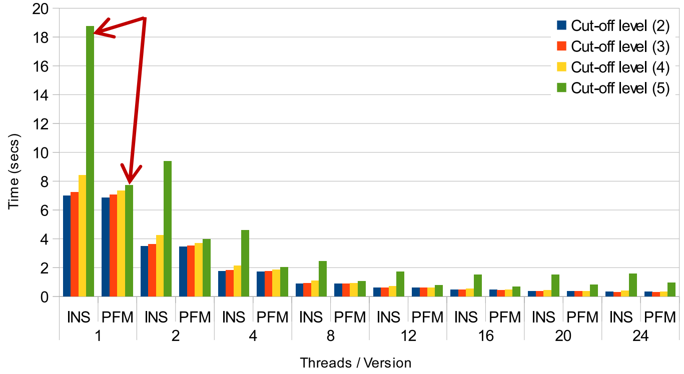
Complementary views

BOTS Benchmarks

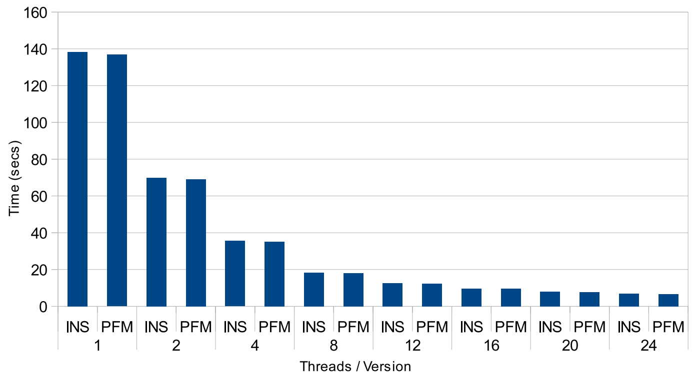
Floorplan



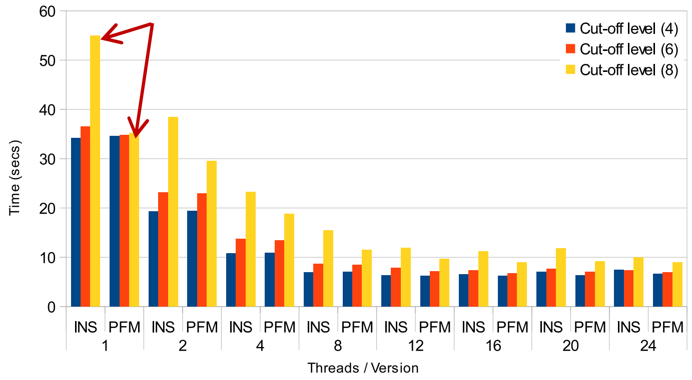
N Queens



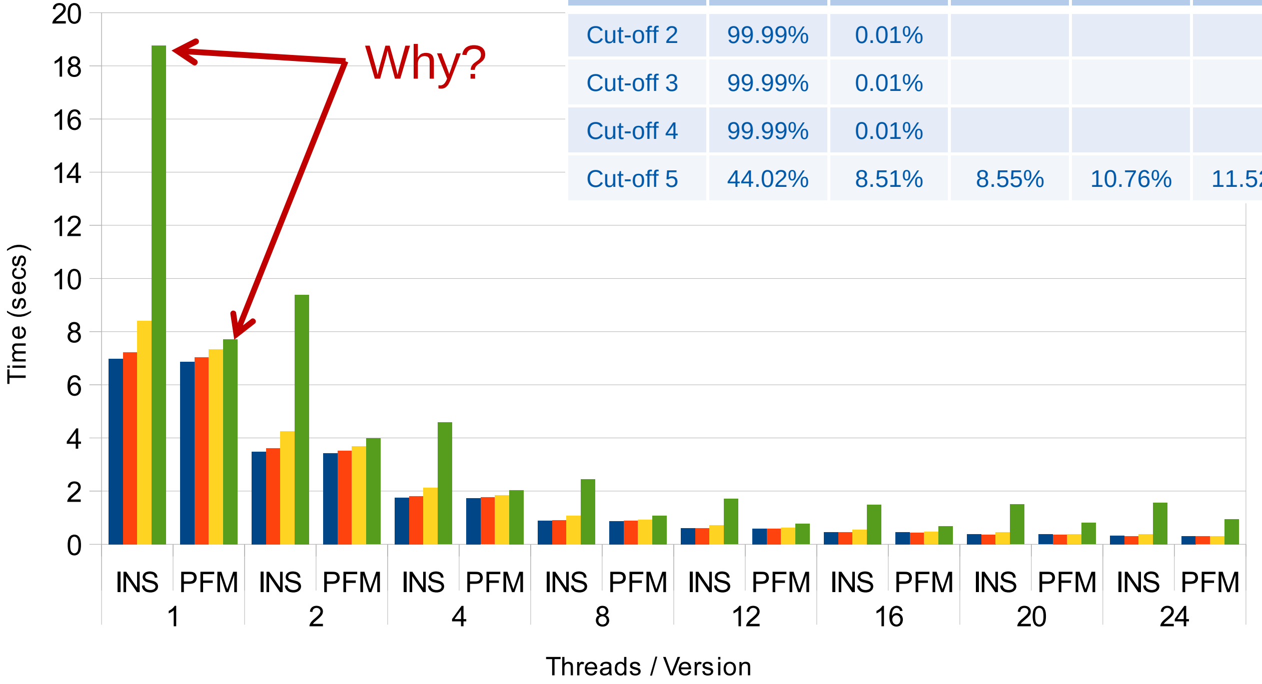
Sparse LU



Strassen



N Queens overhead study

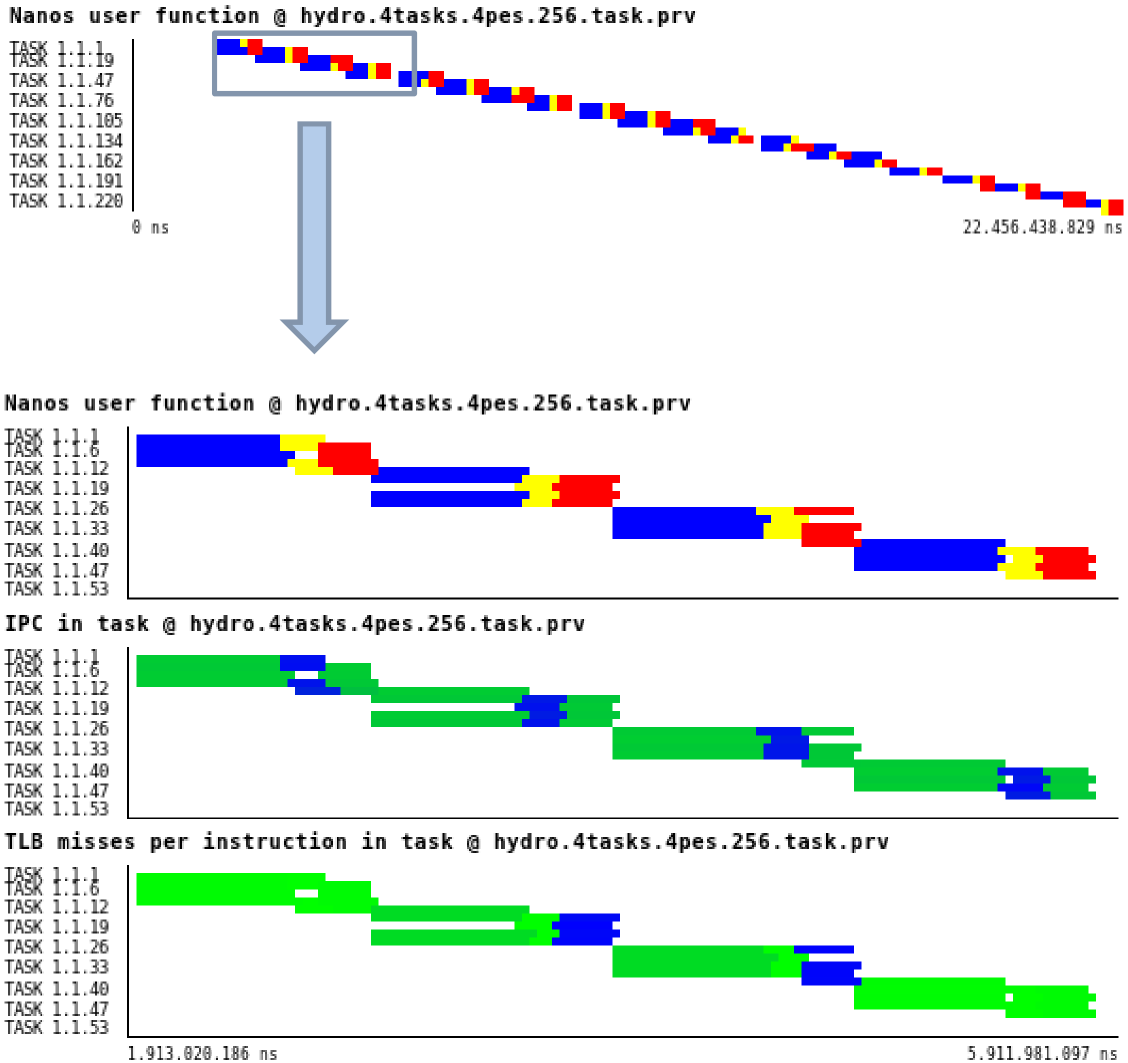


	.00-.01	.01-.02	.02-.03	.03-.04	.04-.05	.05-.06	.06-.07	.07-.08	.08-.09
Cut-off 2	99.99%	0.01%							
Cut-off 3	99.99%	0.01%							
Cut-off 4	99.99%	0.01%							
Cut-off 5	44.02%	8.51%	8.55%	10.76%	11.52%	6.94%	4.30%	2.75%	1.69%

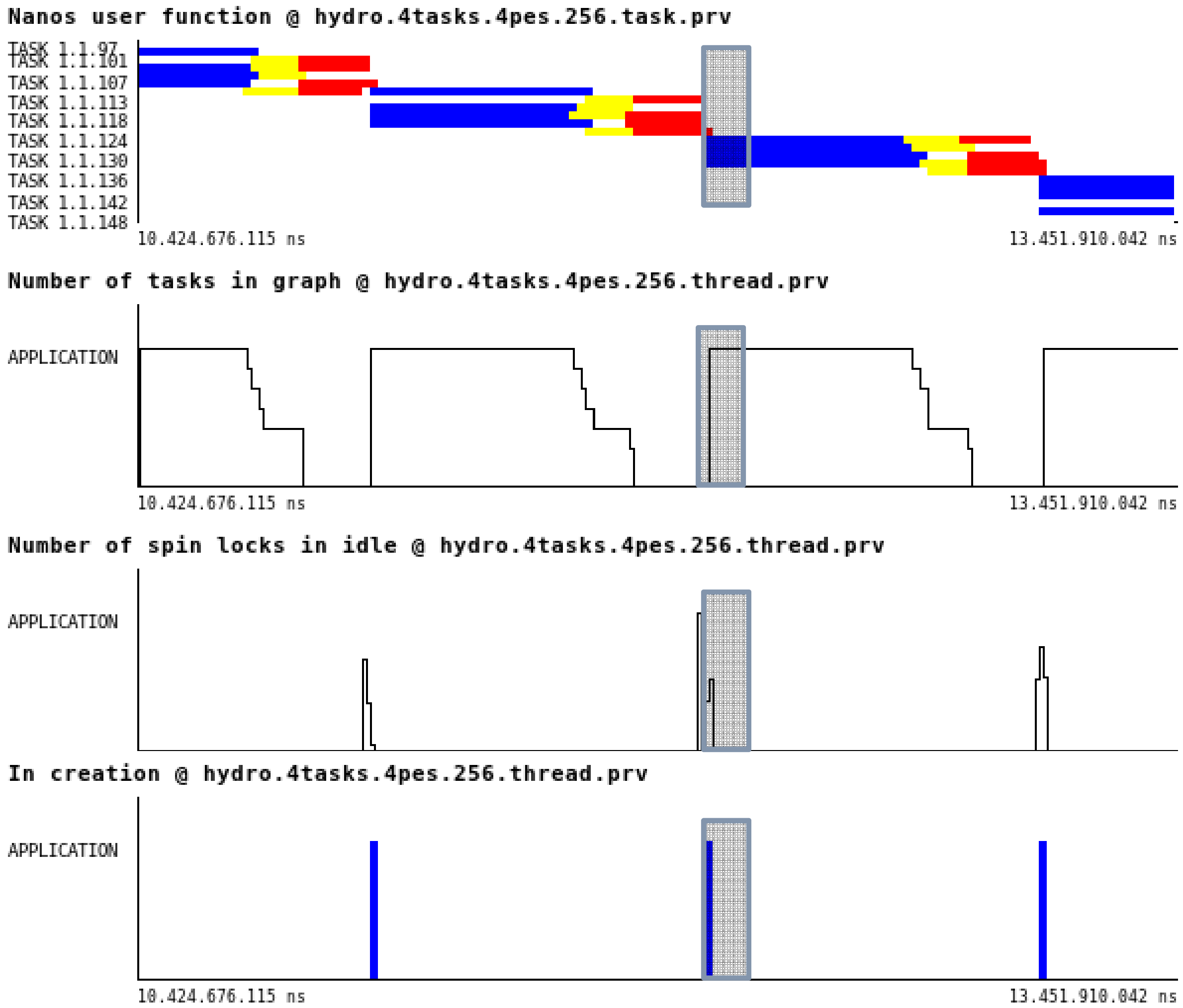
- TLB misses rocket
  - Cut-off ↑ ➔ tasks ↑↑ ➔ traced events ↑↑ ➔ memory usage ↑↑

HydroC analysis (4 threads)

```
for ( ; ; ) {

#pragma omp task output(*qleft,*qright)
 {
  gatherConservativeVars();
  constoprim();
  equation_of_state();
  if (H.iorder != 1) slope();
  trace();
  qleftright();
 }

#pragma omp task input(*qleft,*qright)\
                 output(*qgdnv)
 {
  riemann()
 }

#pragma omp task input(*qgdnv)
 {
  cmpflx();
  updateConservativeVars();
 }

}
#pragma omp taskwait
```



HydroC analysis (4 threads) – Runtime internals



Conclusions

- Integrated environment for productive programming and analysis
  - OpenMP and OmpSs codes
  - Support for tasking constructs (either tied or untied)
  - Extra information: Application performance + Runtime internals
- No need for “magic tricks” to instrument
  - The runtime itself produces the data automatically
- New displays
  - Thread-view + Task-view
  - Track task relations (migrations, dependences, etc.)
  - Correlate with hardware counters
- Tool for the application and the programming model developer

Future Work

- Critical path studies
- Scalability
  - Reduce the information gathered from the runtime
- Support for distributed memory systems and accelerators

www.bsc.es

Barcelona Supercomputing Center

Centro Nacional de Supercomputación

Thank you!