www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

OpenMP Hands-on: Exercises & Laboratory

Parallel Programming Workshop

Xavier Teruel and Xavier Martorell

Montevideo, October 21-22, 2019

Software requirements and system access

SSH: Secure shell (to connect the HPC system)

- Linux: has native support of secure shell "ssh user@host"
- Windows: need to install a ssh program
  - PuTTY http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
  - MobaXterm http://mobaxterm.mobatek.net/download.html

X Server (for wxparaver or .pdf readers)

- Linux: has native support (remember to connect with "ssh -X user@host"
- Windows: need to install a X server program
  - Xming https://sourceforge.net/projects/xming/
  - MobaXterm already includes a X server within the package

Pool of accounts in Minotauro: @mt1.bsc.es @mt2.bsc.es @mt3.bsc.es

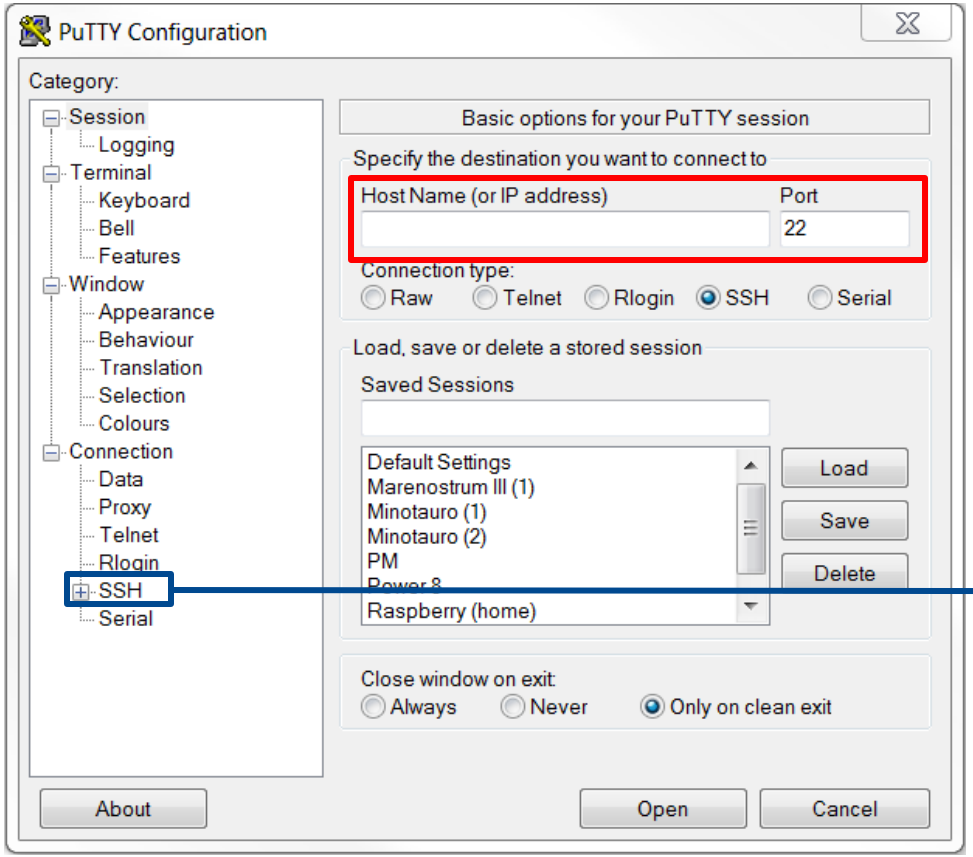
```
username: nct01YYY
password: 699rjs.YYY

Where YYY is in range [131-170]
```

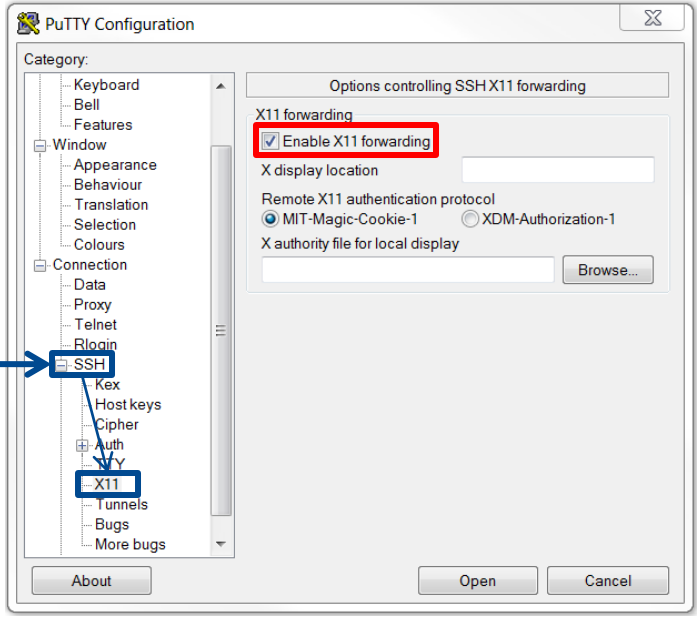
Example (YYY = "042"):

```
$ ssh -X nct01042@mt1.bsc.es
```

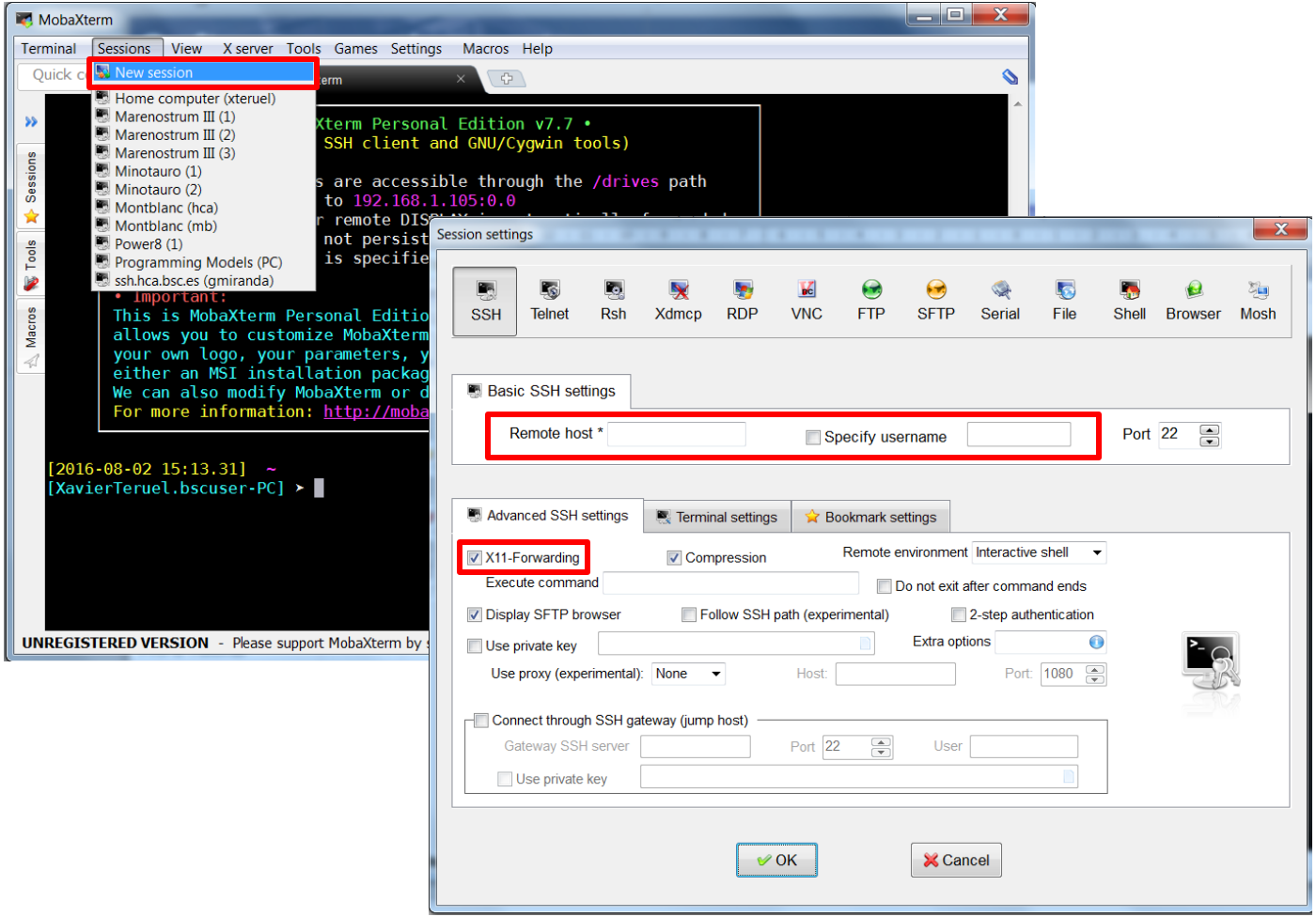
& type the password: **699rjs.042**

PuTTY: ssh configuration





MobaXterm: ssh configuration



Extract and compile exercises & laboratory

(1) Extracting the sources

```
$ tar –xzvf openmp-ee.tar.gz
<list of extracted files>
```

(2) Main directory contents

```
$ cd openmp-ee
$ ls
01-benchmark
02-data-env
03-benchmark
<...exercises...>
```

(3) Configure your system

```
$ source configure.sh
<configure output>
```

(4) Exercise directory (e.g.)

```
$ cd 01-benchmark
$ ls
Makefile        matrix_init.c  matrix_list.c
matrix_loop.c   matrix_main.c  matrix-run.sh  support.h
matrix_head.c   matrix_init.h
matrix_list.h   matrix_loop.h  matrix_main.h
support.c
```

(5) Build application (make)

```
$ make
```

Synthetic benchmark: introduction

Sequential execution of several phases

matrix_main.c

```
int main (int argc, char* argv[] ) {
   double ref_time = get_time();
   // Program initialization
   synthetic_init();

   // (3) Independent phases
   synthetic_phase1();
   synthetic_phase2();
   synthetic_phase3();

   // Program verification
   synthetic_check();
}
```

support.h

```
#define VERSION "orig"
```

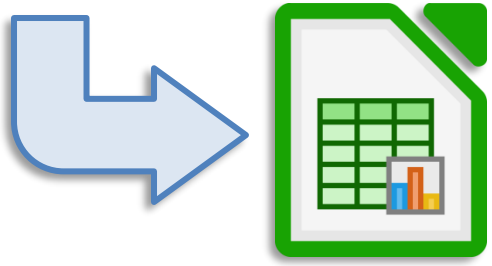
```
$ make
$ ./matrix-run.sh
```

```
program, threads, version, time-0, time-1, time-2, time-3, total, check
./matrix-par,1,orig,0.006966,1.160599,0.869602,0.754891,2.795729,pass(4)
./matrix-par,2,orig,0.006144,1.165501,0.871603,0.777414,2.824604,pass(4)
./matrix-par,3,orig,0.006638,1.156533,0.866852,0.743644,2.773355,pass(4)
./matrix-par,4,orig,0.006509,1.161360,0.869006,0.749328,2.790883,pass(4)
./matrix-par,5,orig,0.006222,1.159360,0.868245,0.759814,2.793526,pass(4)
./matrix-par,6,orig,0.006072,1.158124,0.872231,0.743791,2.783974,pass(4)
./matrix-par,7,orig,0.006353,1.195458,0.900220,0.750928,2.856615,pass(4)
./matrix-par,8,orig,0.005744,1.201392,0.887900,0.750698,2.849589,pass(4)
./matrix-seq,1,orig,0.007484,1.183664,0.873954,0.759545,2.828253,pass(4)
```

VERIFICATION

TOTAL

PHASES



www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Fundamentals

Parallel Programming Workshop

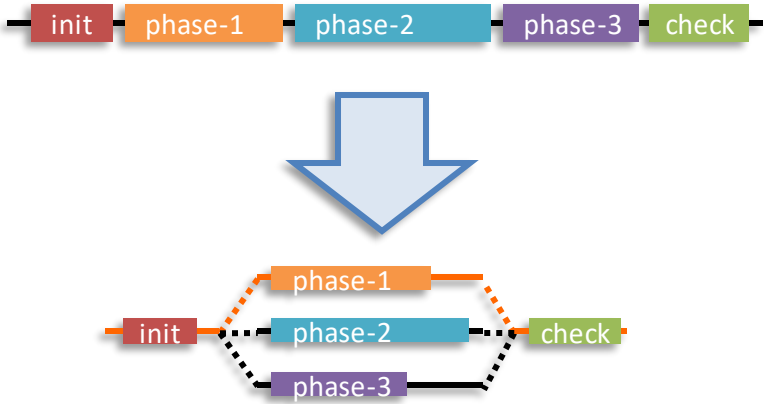
Montevideo, October 21-22, 2019

01-benchmark: distribute work among threads

Edit the matrix_main.c file and parallelize synthetic phases:

```
int main (int argc, char* argv[] ) {
   double ref_time = get_time();
   // Program initialization
   synthetic_init();

   // (3) Independent phases
   synthetic_phase1();
   synthetic_phase2();
   synthetic_phase3();
   ...
}
```

```
$ make
$ ./matrix-run.sh
```



Using thread-id and num-threads distribute work among threads

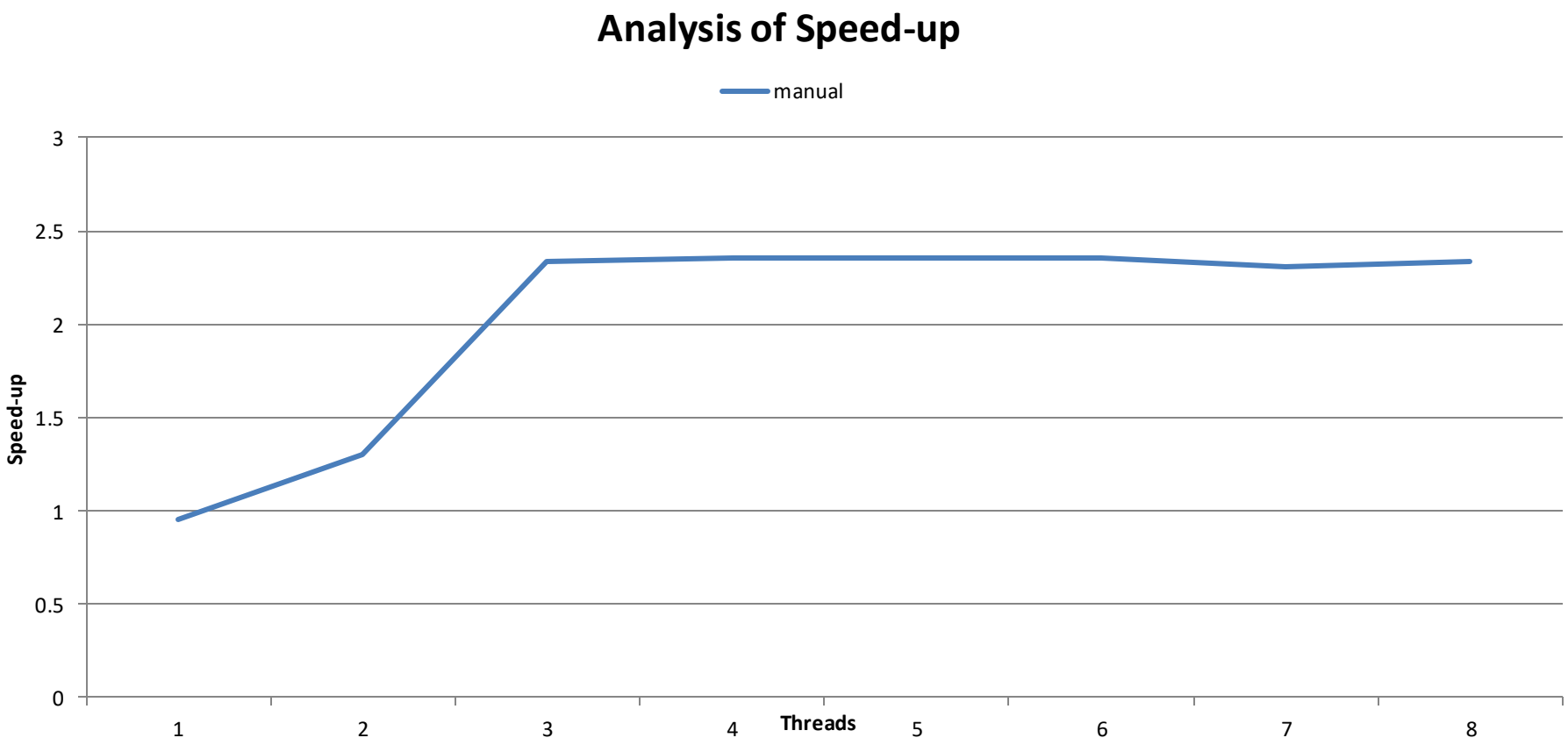
- DO NOT break the sequential code (matrix-seq)
- DO NOT override the OMP_NUM_THREADS variable

Performance analysis (compute speed-up *wrt* matrix-seq)

01-benchmark: manual distribution hints

01-benchmark: proposed solution

01-benchmark: analysis of speed-up



02-data-env: data-sharing attributes

Edit the data_sharing.c file and fix the problems (example)

```
// ---------------------------------------------
// Each thread in the parallel region should
// have their own private 'a' variable
// ---------------------------------------------
char data_parallel_01 ( void )
{
   int a = 0;
   #pragma omp parallel
   {
      a = 1;
   }
   return ( a == 0 );
}
```

```
$ make

$ ./test-data.sh
data_parallel_01.............. fail
data_parallel_02.............. fail
data_parallel_03.............. fail
data_parallel_04.............. fail
data_parallel_05.............. fail
data_parallel_06.............. fail
```

02-data-env: data-sharing attributes (solution: 1)

02-data-env: data-sharing attributes (solution: 2)

02-data-env: data-sharing attributes (solution: 3)

02-data-env: data-sharing attributes (solution: 4)

02-data-env: data-sharing attributes (solution: 5)

02-data-env: data-sharing attributes (solution: 6)

Homework: better distribution of loop iterations

Parallel loop initial approach

```
#include <omp.h>
#define SIZE 1204
double A[SIZE];
void main (void)
{
  #pragma omp parallel
  {
    int id = omp_get_thread_num();
    int nt = omp_get_num_threads();
    int lb = id * (SIZE/nt);
    int ub = (id+1)*(SIZE/nt) + ( (id==nt-1)? (SIZE%nt) : 0 );
    for (int i = lb; i < ub; i++) {
      A [ i ] = 0;
    }
  }
}
```

Example of distribution:

- 23 iterations
- 4 threads

Iteration space (per thread)

- thread-0: [00..04] → 5 iters
- thread-1: [05..09] → 5 iters
- thread-2: [10..14] → 5 iters
- thread-3: [15..23] → 8 iters*

*60% more iterations!!!

Could you think in a better algorithm?

www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Worksharing

Parallel Programming Workshop

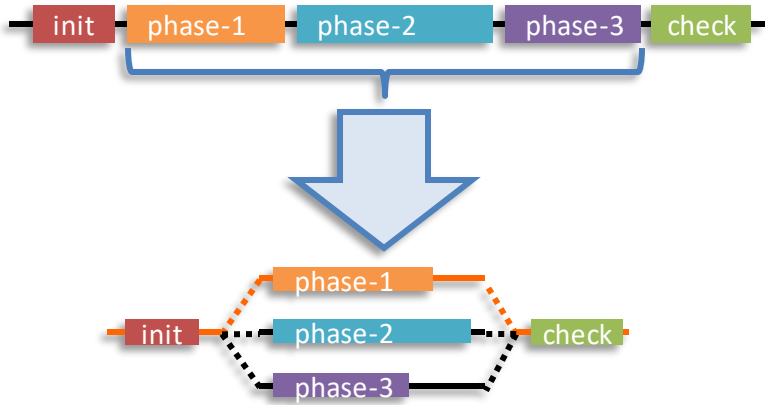
Montevideo, October 21-22, 2019

03-benchmark: distribute work among threads

[AGAIN] Edit the matrix_main.c file and parallelize phases:

```
int main (int argc, char* argv[] ) {
   double ref_time = get_time();
   // Program initialization
   synthetic_init();

   // (3) Independant phases
   synthetic_phase1();
   synthetic_phase2();
   synthetic_phase3();
   ...
}
```

```
$ make
$ ./matrix-run.sh
```



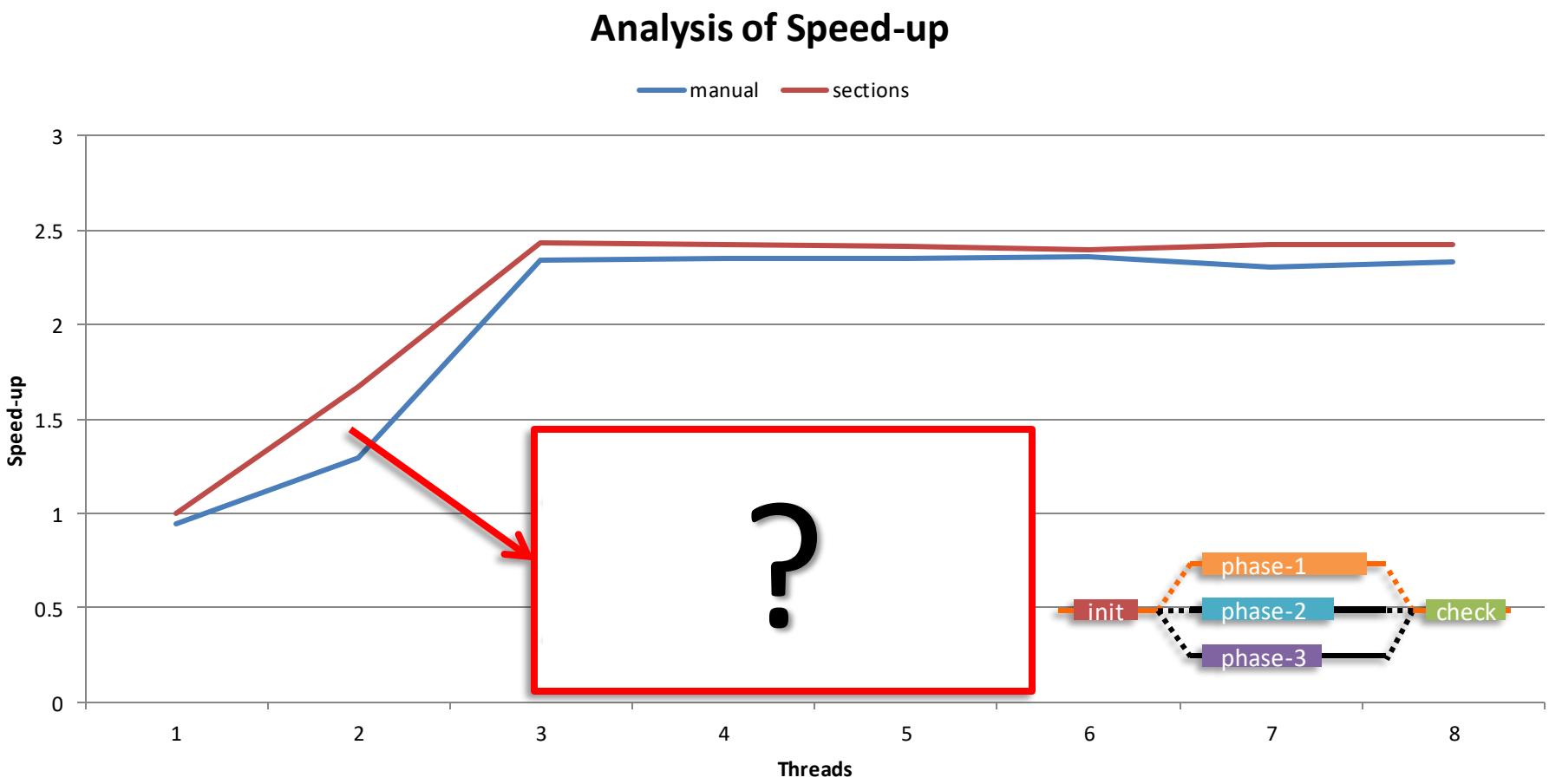
Using parallel sections distribute work among threads

- DO NOT break the sequential code (matrix-seq)
- DO NOT override the OMP_NUM_THREADS variable

Performance analysis: compute speed-up wrt matrix-seq + compare w/ previous version

03-benchmark: proposed solution

03-benchmark: analysis of speed-up

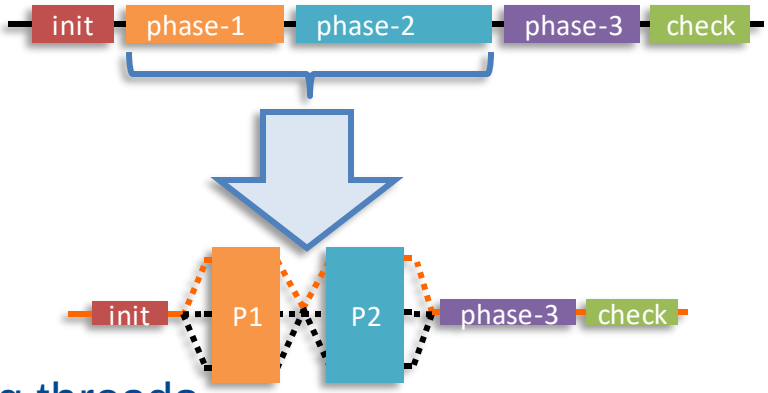


04-benchmark: distribute loop among threads

Edit the matrix_loop.c file and parallelize the loops

```
void matrix_multiply ( double *A, double *B,
                       double *C, int size )  {
 int i,j,k;
 for(i=0;i<size;i++) {
   ...
}
void matrix_compute ( double *A, double *B,
                       double *C, int size )  {
 int i,j,k;
 for(i=0;i<size;i++) {
   ...
}
```

```
$ make
$ ./matrix-run.sh
```



Using worksharing constructs distribute work among threads

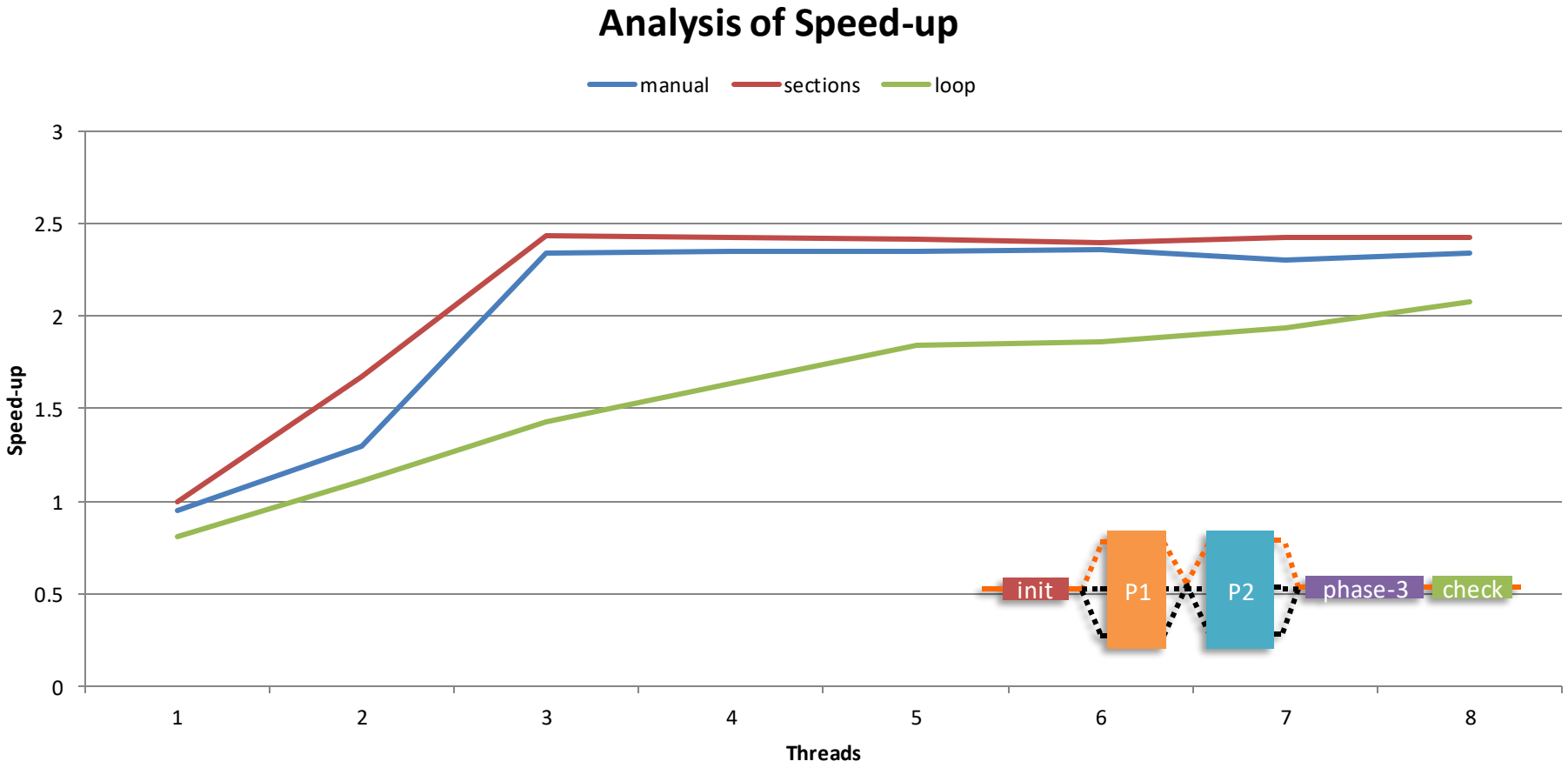
- Function matrix_multiply() → loop parallelization, schedulers, options
- Function matrix_compute() → loop parallelization, schedulers, options

Performance analysis (special care to time-1 and time-2)

04-benchmark: proposed solution (1st phase)

04-benchmark: proposed solution (2nd phase)

04-benchmark: analysis of speed-up



www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Tasking

Parallel Programming Workshop

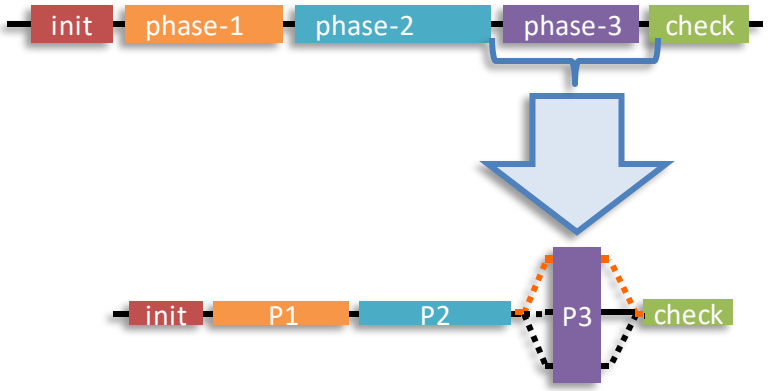
Montevideo, October 21-22, 2019

05-benchmark: distribute tasks among threads

Edit the matrix_list.c file and parallelize nested while loops

```
// Matrix List: compute matrix multiplication
  i = 0;
  while ( i < NT ) {
    j = 0;
    while ( j < NT ) {
      mlist_t *tmpA = matrix_2D_list_find(mlist_A, i, j);
      double *tileA = tmpA->tile;
      k = 0;
      while ( k < NT ) {
        mlist_t *tmpB = matrix_2D_list_find(mlist_B, i, k);
        double *tileB = tmpB->tile;
        mlist_t *tmpC = matrix_2D_list_find(mlist_C, k, j);
        double *tileC = tmpC->tile;
        matrix_2D_tile_multiply(tileA, tileB, tileC, TS);
        k++;
      }
      j++;
    }
    i++;
  }
```

```
$ make
$ ./matrix-run.sh
```

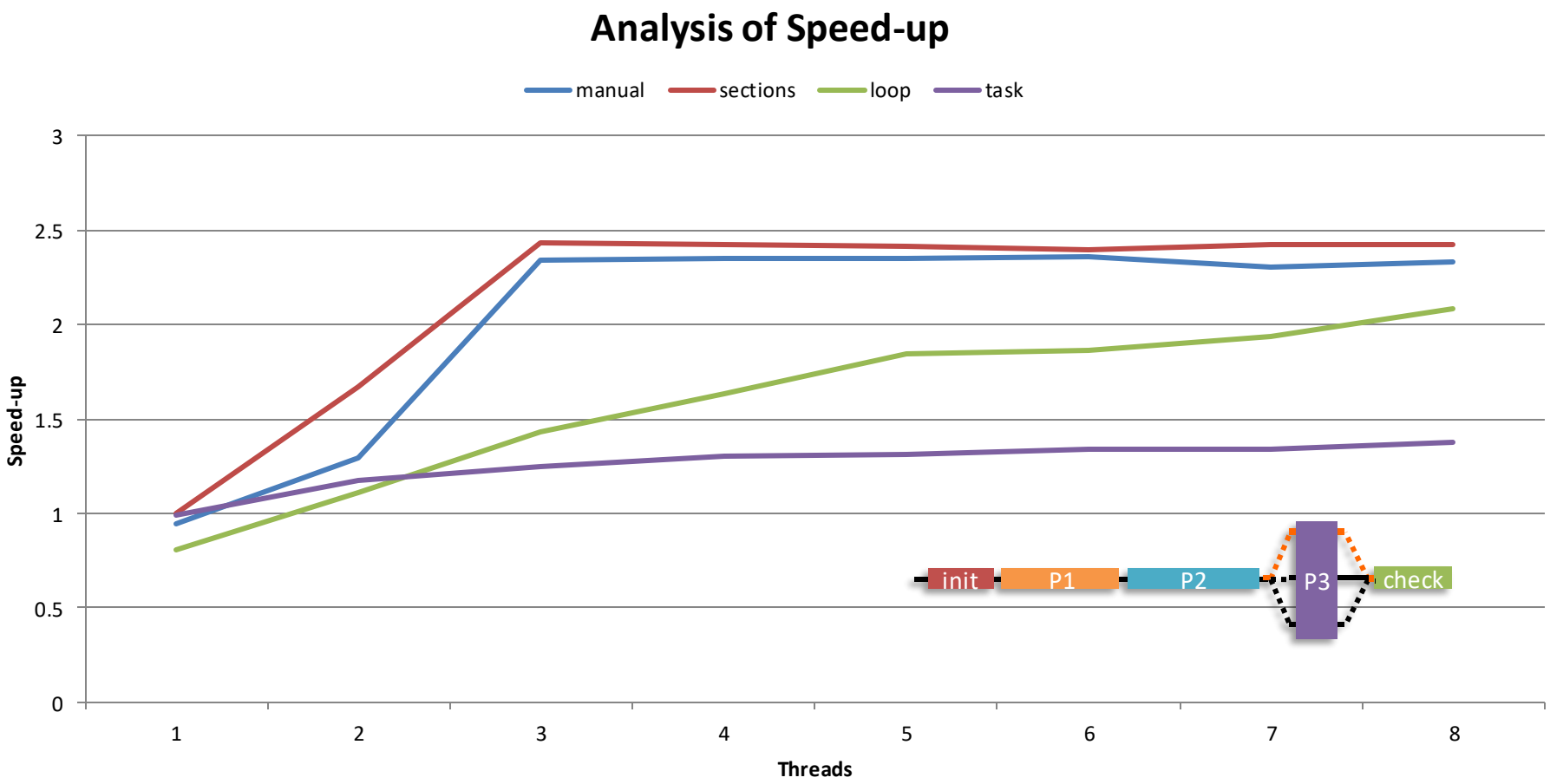


Using tasking constructs distribute work among threads
Explore other places in which you can use (also) tasks: copies?

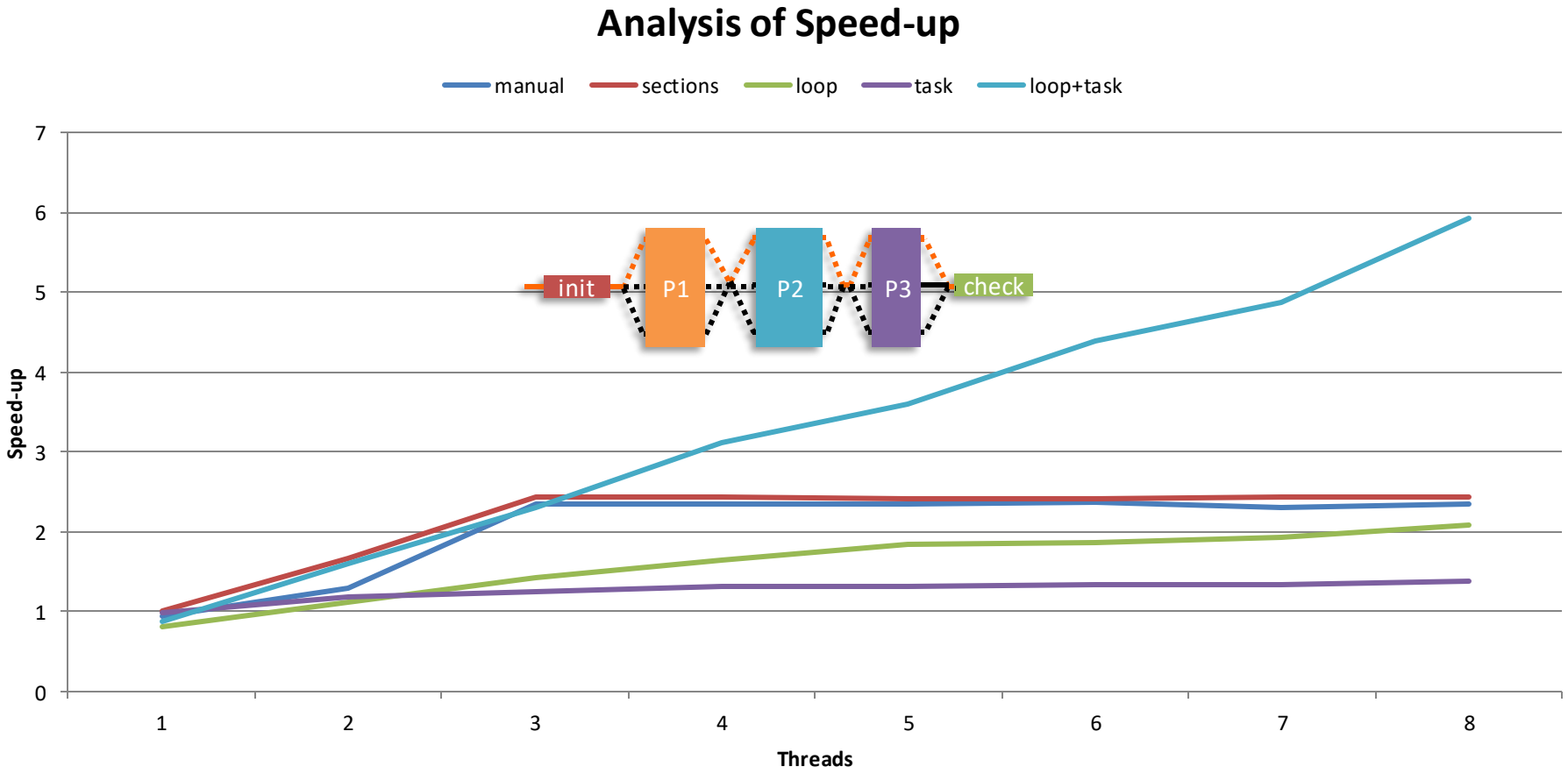
05-benchmark: proposed solution (compute)

05-benchmark: proposed solution (copy)

05-benchmark: analysis of speed-up (only tasks)



0x-benchmark: analysis of speed-up (all)



www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Laboratory

Parallel Programming Workshop

Montevideo, October 21-22, 2019

Laboratory contents

Continue with previous hands-on exercises

- New parallelization approaches, loop policies, locality, etc.
- Getting paraver traces, performance analysis,

New available apps (lab-nn-xxxxx)

- Matrix multiplication (blocked version)
- Heat diffusion (jacobi, redblack, gauss-seidel)

Explore your own applications

- Application's knowledge before starting parallelization
- Verification mechanism is highly recommended
- Incremental approach (not too many changes at a time)

Matrix multiplication

List of source files:

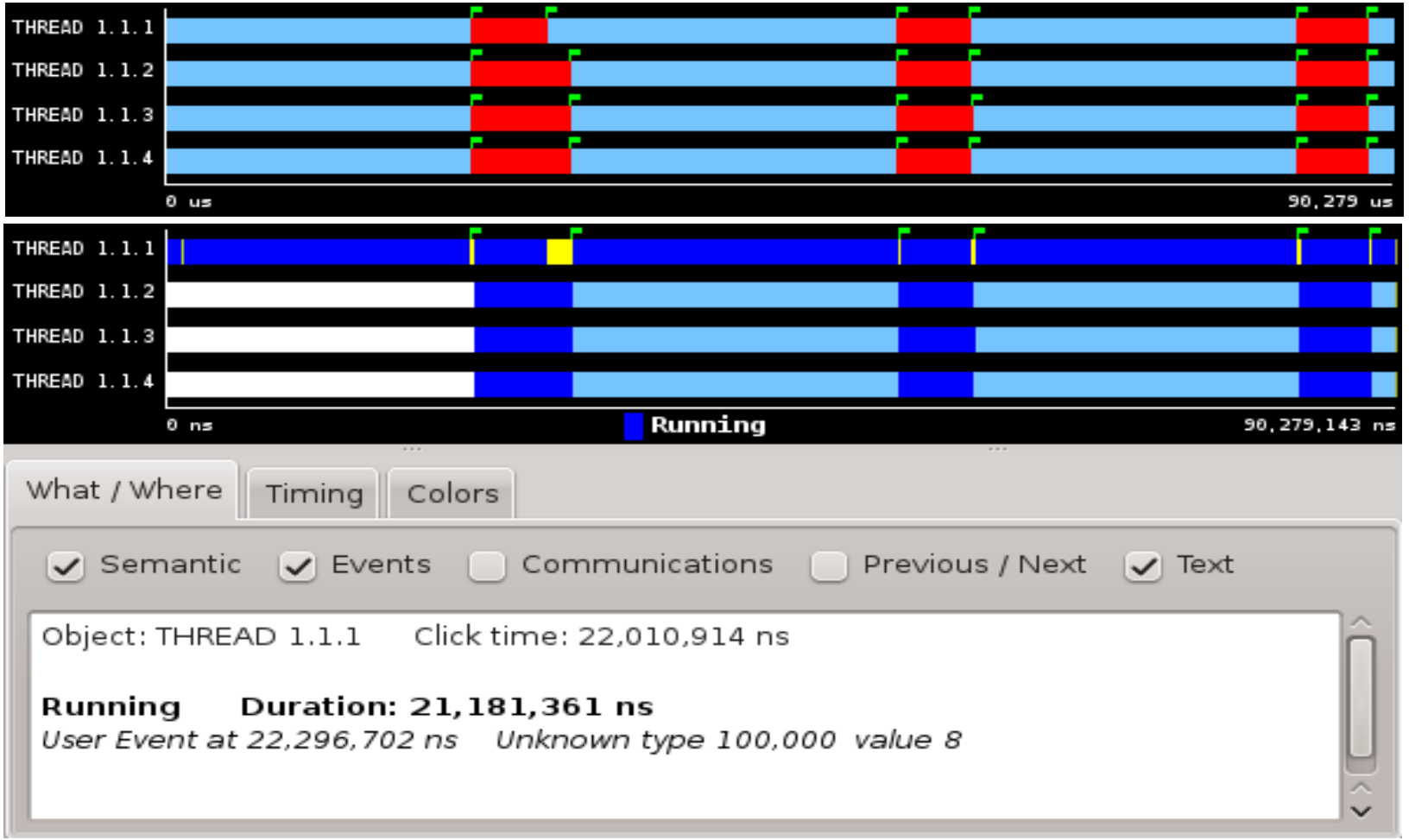
– driver.c	main program
– matmul.c	matmul implementation → parallelism here!
– check.c	routine to check for errors
– config.h	configuration file defining the block size
– test.in	input file defining matrix sizes
– run-extrae.sh	shell script to run generating a Paraver trace
– run.sh	shell script to run (non-instrumented version)

Find where you can parallelize the application

- Add the proper OpenMP directive, make sure all data-sharing attributes are properly set
- Run and evaluate performance, compute and compare speed-up with respect to the sequential execution
- Get paraver traces to better understand execution

Matrix multiplication: proposed solution

Matrix multiplication: trace (user events)



Heat diffusion (1)

Heat implements 3 algorithms for computing heat propagation

- Solver.c contains three different algorithms
  - Jacobi (function relax_jacobi, solver.c)
  - Red/Black (function relax_redblack, solver.c)
  - Gauss Seidel (function relax_gauss, solver.c)

Heat.c contains the main program

- For Jacobi, it implements the copy from “uhelp” field to “u” (can be parallel)

Input file test*.dat

- defines number of iterations to perform, resolution, and algorithm selected

Compilation will generate two binnary versions

- heat → OpenMP plain version
- heat.extrae → OpenMP instrumented version

After running the application (run.sh) an image is generated

- test512-1.ppm → image generated by the app

Heat diffusion (2)

Find where you can parallelize the application

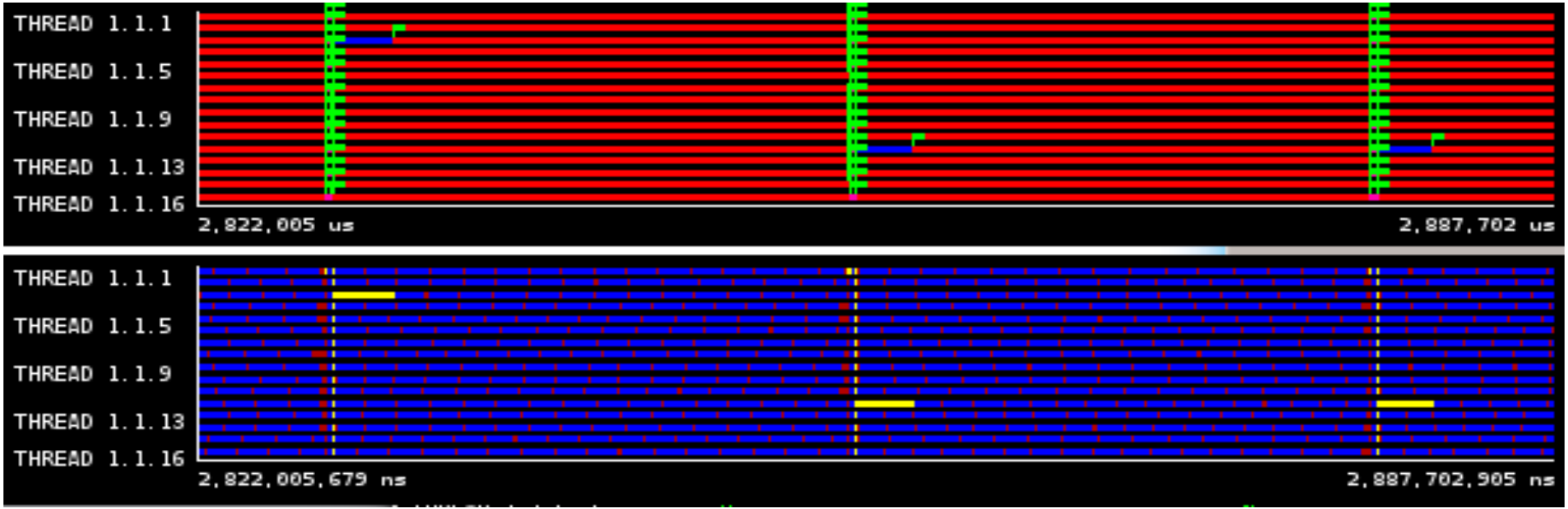
- Add the proper OpenMP directive, make sure all data-sharing attributes are properly set
- Run and evaluate performance, compute and compare speed-up with respect to the sequential execution
- Get paraver traces to better understand execution

Heat diffusion: Jacobi (parallel for)

Heat diffusion: Jacobi (tasks)

Heat diffusion: Jacobi trace (parallel for)

Heat diffusion: Jacobi trace (tasks)



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Thank you!

For further information please visit/contact

http://www.linkedin.com/in/xteruel

xavier.teruel@bsc.es

Synthetic benchmark: introduction (2)

Building the synthetic benchmark → generate binaries

```
$ make
Building: matrix-par and matrix-seq
```

Execute the matrix runner script (multiple executions) → CSV

```
program, threads, version, time-0, time-1, time-2, time-3, total, check
./matrix-par,1,orig,0.006966,1.160599,0.869602,0.754891,2.795729,pass(4)
./matrix-par,2,orig,0.006144,1.165501,0.871603,0.777414,2.824604,pass(4)
./matrix-par,3,orig,0.006638,1.156533,0.866852,0.743644,2.777355,pass(4)
./matrix-par,4,orig,0.007509,1.161360,0.869006,0.749328,2.790883,pass(4)
./matrix-par,5,orig,0.006222,1.159301,0.872452,0.752814,2.794526,pass(4)
./matrix-par,6,orig,0.006072,1.158124,0.872231,0.743791,2.783974,pass(4)
./matrix-par,7,orig,0.006353,1.195458,0.900220,0.750928,2.856615,pass(4)
./matrix-par,8,orig,0.005744,1.201392,0.887900,0.750698,2.849589,pass(4)
./matrix-seq,1,orig,0.007484,1.183664,0.873954,0.759545,2.828253,pass(4)
```

INITIAL PHASES TOTAL