www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

OmpSs Hands-on

Rosa M. Badia, Xavier Martorell, and Xavier Teruel

Barcelona, May 22nd 2019

Software requirements

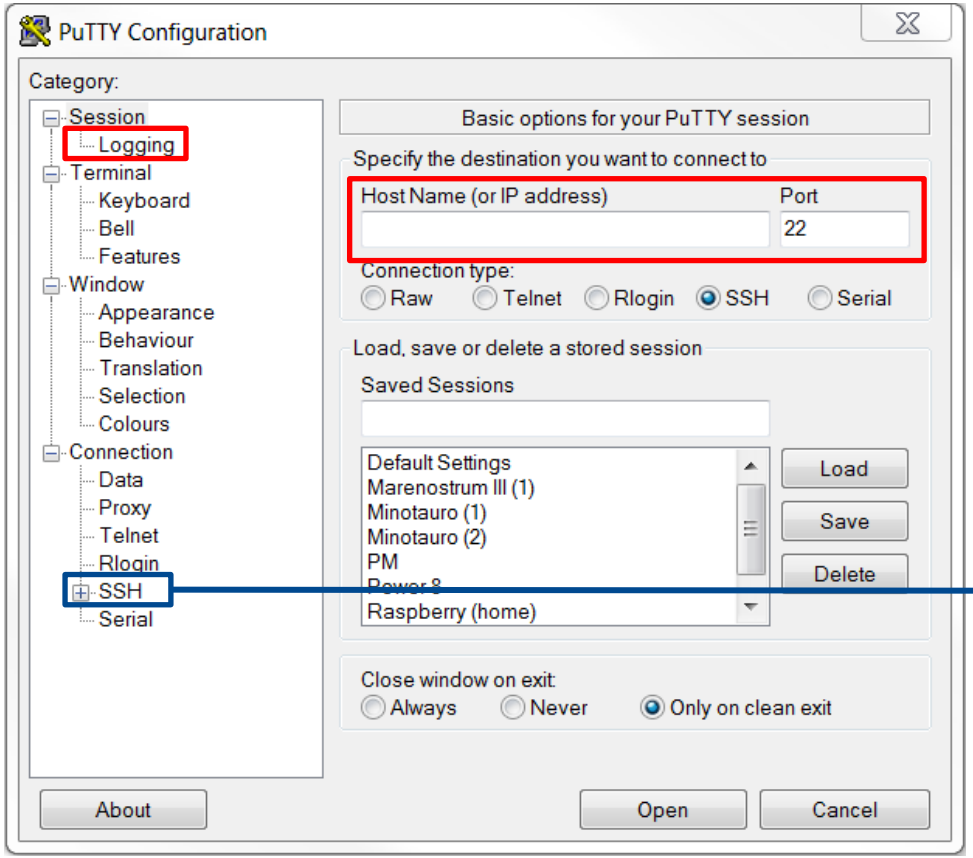
1) SSH: Secure shell (to connect the HPC system)

- 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

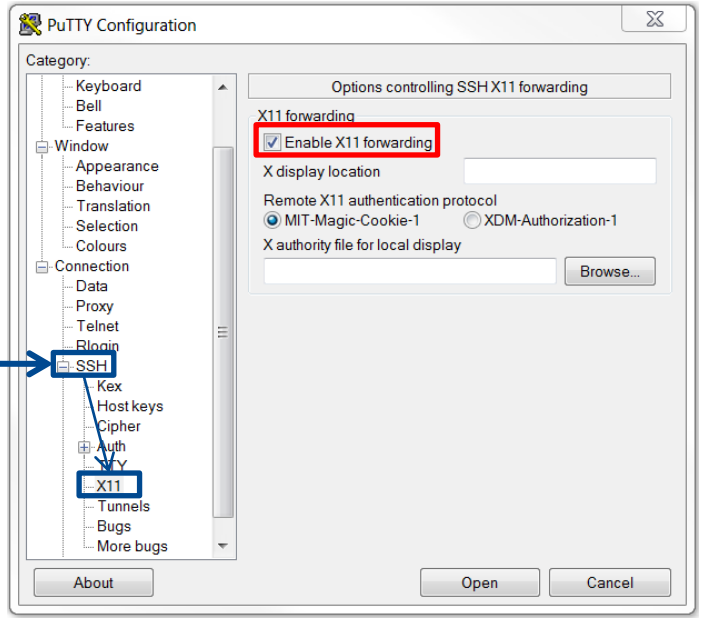
2) X Server / X forwarding (for wxparaver or .pdf readers)

- 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

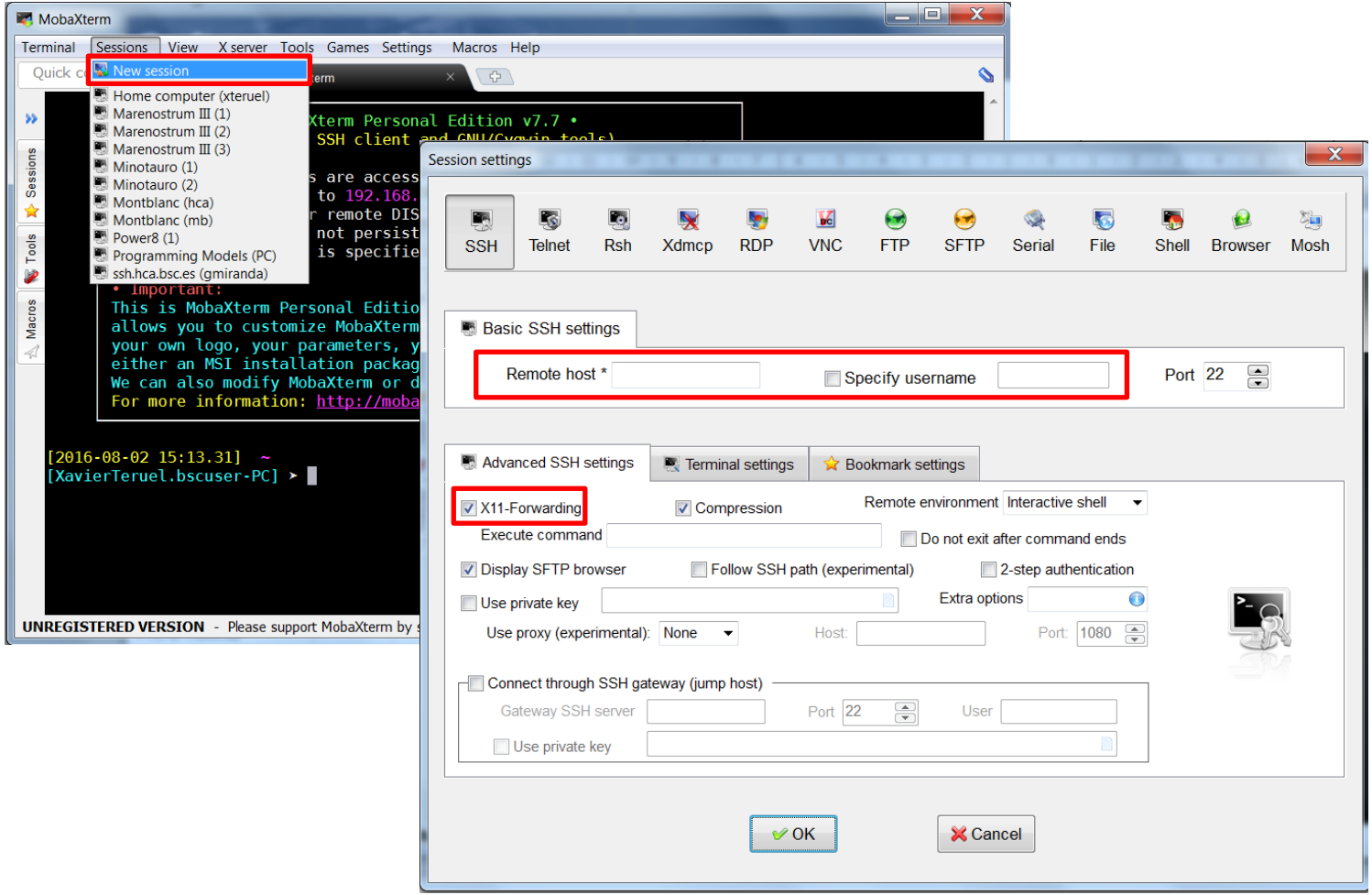
PuTTY: ssh configuration



Before open session make sure…



MobaXterm: ssh configuration



Account and login information [Minotauro]

1) Get your identifier

- Identifiers are three digit numbers [098-122], [125-139] & [141-150]

2) Username and password

- Username: nct01<your_id_here>
- Password: a466mg44.<your_id_here>

Example: for identifier 042, account information would be:

- Username: nct01042
- Password: a466mg44.042

Minotauro (login):

```
$ssh -X    nct01089@mt1.bsc.es
                    @mt2.bsc.es
```

```
$ cp -r /apps/PM/training/PATC-May2019/* .
$ ls
PATC-May2019-00-Xxxxx-xx-XxxXx.pdf
PATC-May2019-01-XxxXx-Xxxxx-Xx.pdf
[...]
ompss-ee.tar.gz
```

System overview (Minotauro)

- 39 nodes
  - » 2 Intel E5649 6C at 2.53 GHz
  - » 2 M2090 NVIDIA GPU Cards (Fermi)
  - » 24 GB of Main memory
  - » Peak Performance: 88.60 Tflop/s
    - *M2090: 81.20 Tflop/s*
    - *E5649: 7.40 Tflop/s*
  - » 250 GB SSD as local storage
  - » 2 Infiniband QDR (40 Gbit each)

- 61 nodes
  - » 2 Intel Xeon E5-2630 v3 (Haswell) 8 core 2.4GHz
  - » 2 K80 NVIDIA GPU Cards
  - » 128 GB of Main memory
  - » Peak Performance: 250.94 Tflops
    - *K80: 226.98 Tflop/s*
    - *E5-2630: 23.96 Tflop/s*
  - » 120 GB SSD as local storage
  - » 1 PCIe 3.0 8GT/s Mellanox ConnectX 3FDR 56 Gbit
  - » 4 Gigabit Ethernet ports

Extract and configure example package

(1) Extracting the sources

```
$ tar –xvzf ompss-ee.tar.gz
<list of extracted files>
```

(2) Main directory

```
$ cd ompss-ee
$ ls
00-introduction
01-examples
02-beginners
03-gpu-devices
04-mpi+ompss
05-ompss+dlb
common-files
configure.sh
paraver-cfgs
README.rst
```

(3) Configure

```
$ source configure.sh
Basic configuration...
  Mercurium compiler at /apps/ompss/bin
  Extrae library at /apps/extrae/bin
  Paraver utility at /apps/wxparaver/bin
  ...
Aditional libraries...
MPI library at /opt/mpi/bullxmpi/.../lib
MKL library at /opt/.../mkl/lib/intel64/
ATLAS library at /opt/.../ATLAS/3.9.51/lib
```

Building the example package

(1) nn-session / exercise

```
$ cd 01-examples/cholesky
$
```

(2) Directory contents

```
$ ls
cholesky.c
cholesky.h
Makefile
README.rst
```

(3) README.rst file (script)

```
$ vi README.rst
$
```

(4) Run “make” will create…

```
$ make
Building: program-d, program-i and program-p
Creating: mutirun.sh, run-once.sh, trace.sh and
extrae.xml
```

» different executable versions (suffixed -d, -i and -p)
» different scripts to run your programs
» a extrae.xml file (as default to get your paraver traces)

```
$ ls
cholesky.c      cholesky-d
cholesky.h      cholesky-i
cholesky-p      extrae.xml
Makefile        multirun.sh
README.rst      run-once.sh
trace.sh
```

Running the example package (Minotauro)

Checking script configuration

```
$ vi run-once.sh
$
```

run-once.sh

```
#!/bin/bash
# @ job_name = ompss-ee
# @ partition = debug
## @ reservation =
. . .
PROGRAM=cholesky-p
export NX_THREADS=4
. . .
./$PROGRAM 4096
```

```
## @ partition = debug
# @ reservation = PATC-TOOLS
```

Submitting the script

```
$ submit run-once.sh
Submitted batch job 1234567
```

```
Check submitted jobs with:
$ queue
```

Exercise directory

```
$ ls
cholesky.c      cholesky-d
cholesky.h      cholesky-i
cholesky-p      extrae.xml
Makefile        multirun.sh
README.rst      run-once.sh
trace.sh
```

Directory after execution

```
$ ls
cholesky.c          cholesky-d
cholesky.h          cholesky-i
cholesky-p          extrae.xml
Makefile            multirun.sh
ompss-ee_nnnnn.err  ompss-ee_nnnnn.out
README.rst          run-once.sh
trace.sh
```

Instrumenting the example package

Checking configuration scripts

```
$ vi run-once.sh trace.sh
$
```

run-once.sh

```
#!/bin/bash
. . .
PROGRAM=cholesky-i
export NX_SMP_WORKERS=4
. . .
./trace.sh ./$PROGRAM 4096 512 1
```

trace.sh

```
#!/bin/bash
. . .
export EXTRAE_CONFIG_FILE=extrae.xml
export NX_INSTRUMENTATION=extrae

$*
```

Exercise directory

```
$ ls
cholesky.c      cholesky-d
cholesky.h      cholesky-i
cholesky-p      extrae.xml
Makefile        multirun.sh
README.rst      run-once.sh
trace.sh
```

Submitting the script

```
$ submit run-once.sh
Creating: cholesky.prv, cholesky.pcf cholesky.raw
```

Visualizing paraver traces [1/5]

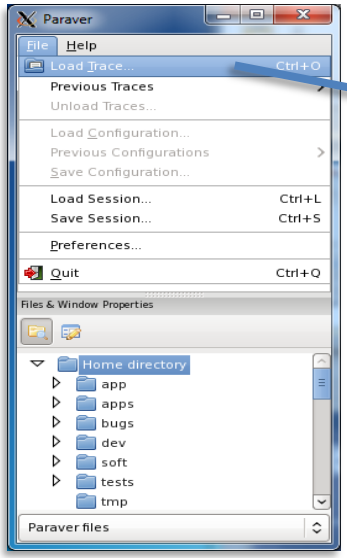
Running paraver tool

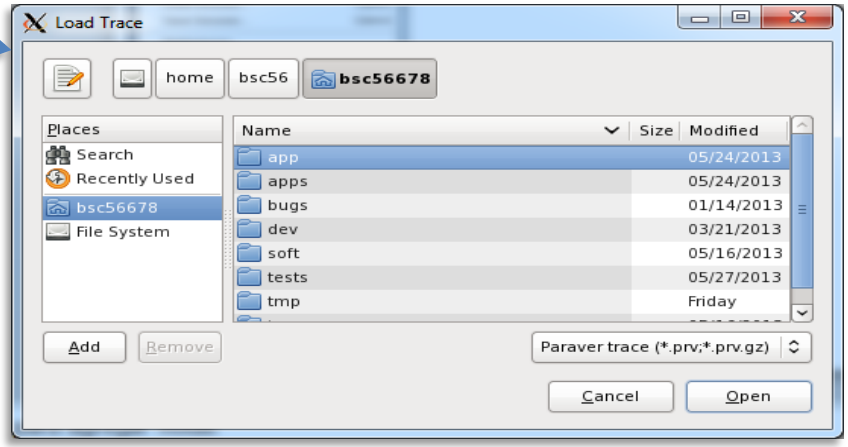
```
$ paraver
$
```

— Load a Paraver trace

Suite exercise directory

```
$ ls
cholesky.c        cholesky-d
cholesky.h        cholesky-i
cholesky-i.pcf    cholesky-i.prv
cholesky-i.raw    cholesky-p
extrae.xml        Makefile
multirun.sh       README.rst
run-once.sh       trace.sh
```





Visualizing paraver traces [2/5]

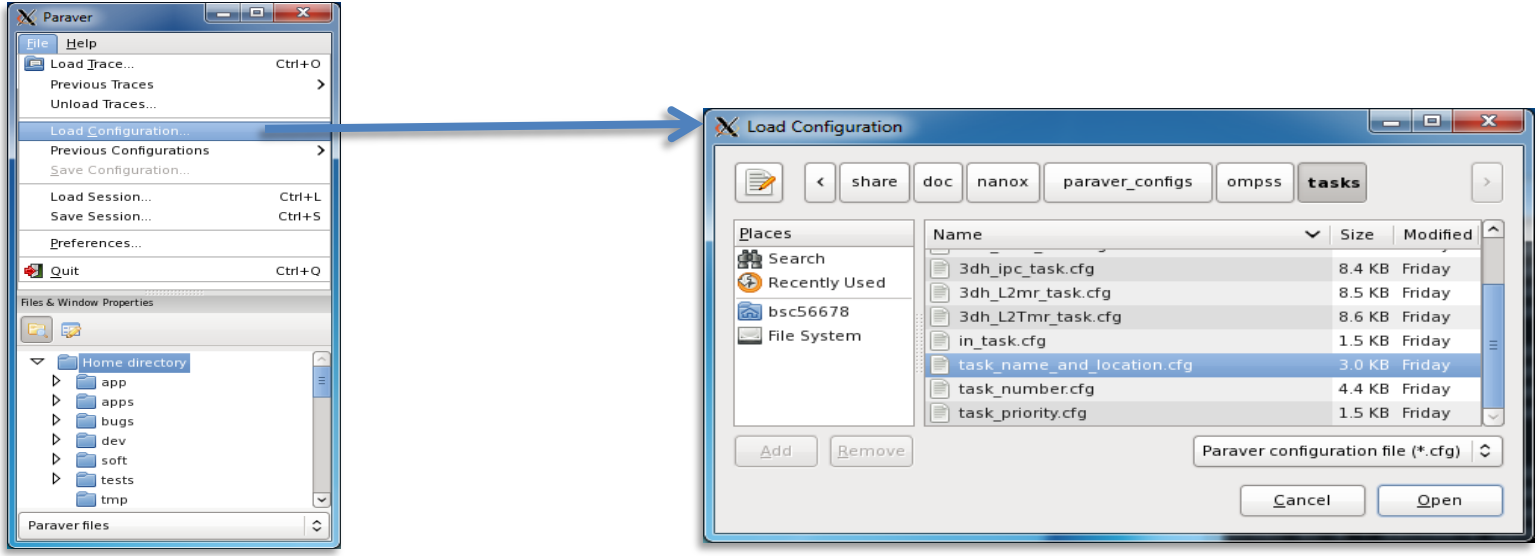
Running paraver tool

```
$ paraver
$
```

— Load a Paraver trace
— Load a configuration file

Suite root directory

```
$ ls
01-examples              02-beginners
03-gpu-devices           04-mpi+ompss
common-files             configure.sh
paraver-cfgs             README.rst
```

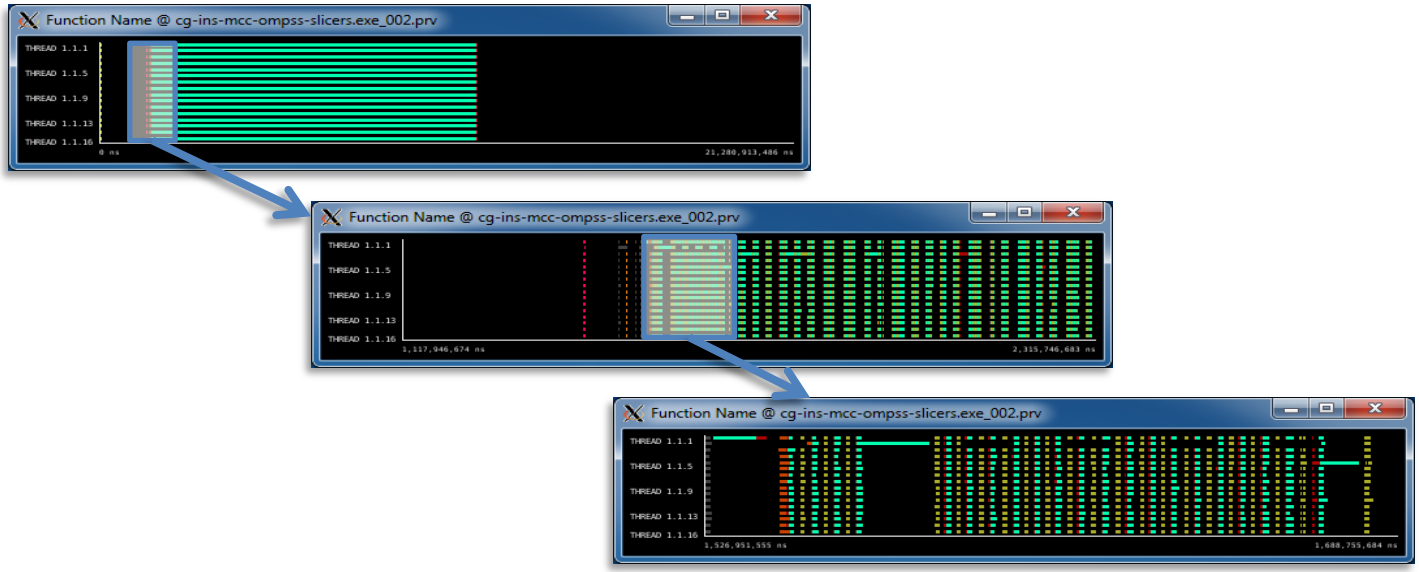


Visualizing paraver traces [3/5]

Running paraver tool

```
$ paraver
$
```

- Load a Paraver trace
- Load a configuration file
- Trace analysis (zoom in, details)



Suite root directory

```
$ ls
01-examples             02-beginners
03-gpu-devices          04-mpi+ompss
common-files            configure.sh
paraver-cfgs            README.rst
```

Visualizing paraver traces [4/5]

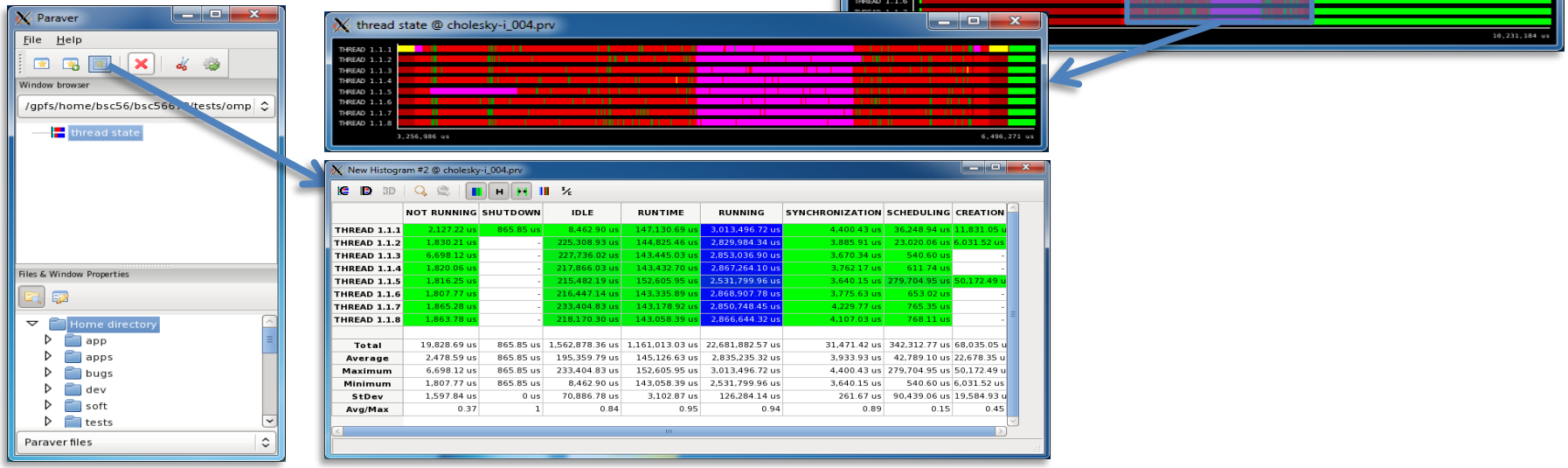
Running paraver tool

```
$ paraver
$
```

- Load a Paraver trace
- Load a configuration file
- Trace analysis (zoom in, details)
- Histograms to summarize traces

Suite root directory

```
$ ls
01-examples              02-beginners
03-gpu-devices           04-mpi+ompss
common-files             configure.sh
paraver-cfgs             README.rst
```



	NOT RUNNING	SHUTDOWN	IDLE	RUNTIME	RUNNING	SYNCHRONIZATION	SCHEDULING	CREATION
THREAD 1.1.1	2,127.22 us	865.85 us	8,462.90 us	147,130.69 us	3,013,496.72 us	4,400.43 us	36,248.94 us	11,831.05 u
THREAD 1.1.2	1,830.21 us	-	225,308.93 us	144,825.46 us	2,829,984.34 us	3,885.91 us	23,020.06 us	6,031.52 us
THREAD 1.1.3	6,698.12 us	-	227,736.02 us	143,445.03 us	2,853,036.90 us	3,670.34 us	540.60 us	-
THREAD 1.1.4	1,820.06 us	-	217,866.03 us	143,432.70 us	2,867,264.10 us	3,762.17 us	611.74 us	-
THREAD 1.1.5	1,816.25 us	-	215,482.19 us	152,605.95 us	2,531,799.96 us	3,640.15 us	279,704.95 us	50,172.49 u
THREAD 1.1.6	1,807.77 us	-	216,447.14 us	143,335.89 us	2,868,907.78 us	3,775.63 us	653.02 us	-
THREAD 1.1.7	1,865.28 us	-	233,404.83 us	143,178.92 us	2,850,748.45 us	4,229.77 us	765.35 us	-
THREAD 1.1.8	1,863.78 us	-	218,170.30 us	143,058.39 us	2,866,644.32 us	4,107.03 us	768.11 us	-
Total	19,828.69 us	865.85 us	1,562,878.36 us	1,161,013.03 us	22,681,882.57 us	31,471.42 us	342,312.77 us	68,035.05 u
Average	2,478.59 us	865.85 us	195,359.79 us	145,126.63 us	2,835,235.32 us	3,933.93 us	42,789.10 us	22,678.35 u
Maximum	6,698.12 us	865.85 us	233,404.83 us	152,605.95 us	3,013,496.72 us	4,400.43 us	279,704.95 us	50,172.49 u
Minimum	1,807.77 us	865.85 us	8,462.90 us	143,058.39 us	2,531,799.96 us	3,640.15 us	540.60 us	6,031.52 us
StDev	1,597.84 us	0 us	70,886.78 us	3,102.87 us	126,284.14 us	261.67 us	90,439.06 us	19,584.93 u
Avg/Max	0.37	1	0.84	0.95	0.94	0.89	0.15	0.45

Visualizing paraver traces [5/5]

Running paraver tool

```
$ paraver
$
```

- Load a Paraver trace
- Load a configuration file
- Trace analysis (zoom in, details)
- Histograms to summarize traces
- Other configuration files
  - » ompss / runtime / thread_state.cfg
  - » ompss / runtime / nanos_API.cfg
  - » ompss / tasks / task_name_and_location.cfg
  - » ompss / cuda / …
  - » hwc / papi / performance / …

Suite root directory

```
$ ls
01-examples             02-beginners
03-gpu-devices          04-mpi+ompss
common-files            configure.sh
paraver-cfgs            README.rst
```

- more info about paraver instrumentation tool

http://pm.bsc.es/ompss-docs/user-guide

www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Compile and Execute

Compile and execute

Exercise's location: 01-examples

```
~ompss-ee:$ ls
01-examples                02-beginners
03-gpu-devices             04-mpi+ompss
...
```

Compile and execute (guidelines)

- Code is completely annotated, you DON'T need to modify it. Review source code, check different directives and their clauses
- Run *corresponding* executable version
  - » program-p → performance's version
  - » program-i → instrumentation's version
  - » program-d → debug's version
- Check scalability → compute speed-up
- Runtime options (schedulers, …)
- Get task dependency graph / paraver traces

Hands-on session's contents

Exercise 1.1: Cholesky kernel
Exercise 1.2: Stream (barr | deps) *b-marks*
Exercise 1.3: Array sum kernel

Remember to configure your system

```
$ source configure.sh
Basic configuration...
```

Check running script (before submit)

```
$ vi run-once.sh
$
```

- Scripts run-once.sh, multi-run.sh or trace.sh
- Executable program's version (-p, -i or -d)
- Job scheduler configuration (queue)
- Other runtime's options used in the script

Documents at http:/pm.bsc.es

Exercise 1.1: Cholesky kernel

Location: 01-examples / cholesky

Source code description

- cholesky.c → main code
- cholesky.h→ headers

Compile and execute the program

```
#pragma omp task inout([ts][ts]A)
void omp_potrf(double * const A, int ts, int ld)
{ ... }
```

Things to do:

- Code is completely annotated, you DON'T need to modify it
- Review source code, check different directives and their clauses
- Check different versions (performance, instrumented & debug)
- Check other runtime options (schedulers, …)
- Check (scalability), execute for different number of thread → compute speed-up
- Get a task dependency graph to analyse dependences
- Get different paraver traces and visualize them: thread state, task name,…

Exercise 1.2: Stream benchmark

Location: 01-examples / stream-xxxx (where xxxx = barr | deps)

Source code description

- stream-xxxx.c → main code (single file)

Compile and execute the programs

```
for (j=0; j<N; j+=BSIZE)
#pragma omp task
...
#pragma omp taskwait
for (j=0; j<N; j+=BSIZE)
```

```
for (j=0; j<N; j+=BSIZE)
#pragma omp task in([bs]a) out([bs]c)
...
for (j=0; j<N; j+=BSIZE)
#pragma omp task in([bs]a) out([bs]c)
```

Things to do:

- Code is completely annotated, you DON'T need to modify it. Review source code, check different directives and their clauses. Compare the two parallelization approaches (one based in the taskwait, the other in dependences)
- Check different versions (performance, instrumented & debug) and other runtimes options
- Check (scalability), execute for different number of thread → compute speed-up
- Get a task dependency graph to analyse dependences
- Get different paraver traces and visualize them: thread state, task name,…

Exercise 1.3: Array sum kernel

Location: 01-examples / array-sum-fortran

Source code description

- array_sum.f90 → main code (single file)

Compile and execute the program

```
!$OMP TASK OUT(VEC1(I:I+BS-1), VEC2(I:I+BS-1), RESULTS(I:I+BS-1))&
!$OMP PRIVATE(J) FIRSTPRIVATE(I, BS) LABEL(INIT_TASK)
   DO J = I, I+BS-1 ...
```

Things to do:

- Code is completely annotated, you DON'T need to modify it
- Review source code, check different directives and their clauses
- Check different versions (performance, instrumented & debug)
- Check other runtime options (schedulers, …)
- Check (scalability), execute for different number of thread → compute speed-up
- Get a task dependency graph to analyse dependences
- Get different paraver traces and visualize them: thread state, task name,…

www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

OmpSs Fundamentals

PATC: Heterogeneous Programming

Barcelona, May 22nd, 2019

OmpSs Fundamentals

Exercise's location: 02-beginners

```
~ompss-ee:$ ls
01-examples                02-beginners
03-gpu-devices             04-mpi+ompss
...
```

OmpSs Fundamentals (guidelines)

- Complete the source code annotation using OmpSs directives (creating tasks)
- Include OmpSs synchronization directives when needed (synchronize tasks)
- Run *corresponding* executable version
  - » program-p → performance's version
  - » program-i → instrumentation's version
  - » program-d → debug's version
- Check scalability → compute speed-up
- Runtime options (schedulers, …)
- Get task dependency graph / paraver traces

Hands-on session's contents

Continue previous session's exercises
Exercise 2.1: Matrix multiply kernel
Exercise 2.2: Dot product kernel
Exercise 2.3: Multisort kernel (rec.)

Remember to configure your system

```
$ source configure.sh
Basic configuration...
```

Check running script (before submit)

```
$ vi run-once.sh
$
```

- Scripts run-once.sh, multi-run.sh or trace.sh
- Executable program's version (-p, -i or -d)
- Job scheduler configuration (queue)
- Other runtime's options used in the script

Documents at http:/pm.bsc.es

Exercise 2.1: Matrix multiplication kernel

Location: 02-beginners / matmul

Source code description

- matmul.c → main code (single file)

Creating tasks

```
for (i = 0; i < DIM; i++)
  for (j = 0; j < DIM; j++)
    for (k = 0; k < DIM; k++)
      matmul ((double *)A[i][k], (double *)B[k][j], (double *)C[i][j], NB);
```

Things to do:

- Look for candidate to become a task
- Don't forget to wait for all task before completion
- Check scalability (for different versions), use runtime options (schedulers, …)
- Get a task dependency graph and/or paraver traces

Exercise 2.2: Dot product kernel

Location: 02-beginners / dot-product

Source code description

- dot-product.c → main code (single file)

Creating tasks

```
for (long i=0; i<N; i+=CHUNK_SIZE) {
  actual_size = (N-CHUNK_SIZE>=CHUNK_SIZE)?CHUNK_SIZE:(N-CHUNK_SIZE);
  C[j]=0;
  for (long ii=0; ii<actual_size; ii++) {
     C[j]+= A[i+ii] * B[i+ii];
  }
  acc += C[j];
  j++;
}
```

Things to do:

- Complete the annotation of tasks. Think how they must be synchronized
- Check different parallelization approaches: concurrent, atomics, commutatives…
- Check scalability (for different versions), use runtime options (schedulers, …)
- Get a task dependency graph and/or paraver traces

Exercise 2.3: Multisort kernel

Location: 02-beginners / multisort

Source code description

- multisort.c → main code (single file)

Creating (nested) tasks

```
multisort(n/4L, &data[0], &tmp[0]);
multisort(n/4L, &data[n/4L], &tmp[n/4L]);
...
merge(n/4L, &data[0], &data[n/4L], &tmp[0], 0, n/2L);
merge(n/4L, &data[n/2L], &data[3L*n/4L], &tmp[n/2L], 0, n/2L);

merge(n/2L, &tmp[0], &tmp[n/2L], &data[0], 0, n);
```

Things to do:

- Think how the tasks must be synchronized
- Check different parallelization approaches: taskwait/dependences
- Check scalability (for different versions), use runtime options (schedulers, …)
- Get a task dependency graph (different domains) and/or paraver traces

www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

OmpSs + CUDA

PATC: Heterogeneous Programming

Barcelona, May 22nd, 2019

OmpSs + CUDA

Exercise's location: 03-gpu-devices

```
~ompss-ee:$ ls
01-examples                 02-beginners
03-gpu-devices              04-mpi+ompss
...
```

OmpSs + CUDA (guidelines)

- Complete existent OmpSs annotation as required (devices, ndrange, copies, dependences,…)
- Complete the source code annotation using OmpSs directives (target directive)
- Include OmpSs synchronization directives when needed (synchronize tasks)
- Run *corresponding* executable version
  - » program-p → performance's version
  - » program-i → instrumentation's version
  - » program-d → debug's version

Documents at http:/pm.bsc.es

Hands-on session's contents

Continue previous session's exercises
Exercise 3.1: SAXPY kernel
Exercise 3.2: Krist kernel
Exercise 3.3: Matrix multiply
Exercise 3.4: Nbody kernel
Exercise 3.5: Cholesky kernel

Remember to configure your system

```
$ source configure.sh
Basic configuration...
```

Check running script (before submit)

```
$ vi run-once.sh
$
```

- Scripts run-once.sh, multi-run.sh or trace.sh
- Executable program's version (-p, -i or -d)
- Job scheduler configuration (queue)
- Other runtime's options used in the script

Exercise 3.1: SAXPY kernel

Location: 03-gpu-devices / saxpy-cuda

Source code description

- File kernel.cu → saxpy CUDA version (definition)
- File saxpy.c → main code, kernel invokation
- File kernel.h → kernel header (partially annotated)

The NDRange clause

```
#pragma target device(cuda) copy_deps ndrange( /*????*/ )
#pragma omp task in([n]x) inout([n]y)
__global__ void saxpy(int n, float a,float* x, float* y);
```

Things to do:

- Complete the OmpSs annotation (NDRange clause)
- Check execution and behaviour for different thread hierarchy configurations
- Check different runtime options (devices, max mem, prefetch, overlap…)

Exercise 3.2: Krist kernel

Location: 03-gpu-devices / krist-cuda

Source code description

- File kernel.cu → cstructface CUDA version
- File krist.c → main code, kernel invokation
- File kernel.h → kernel header (partially annotated)
- File clocks.c → get time support to measure performance (support)

Target and task directives (device support)

```
#pragma omp target device(cuda) ...
#pragma omp task ...
__global__ void cstructfac(int na, int number_of_elements, int nc, float f2, int NA,
                TYPE_A* a, int NH, TYPE_H* h, int NE, TYPE_E* output_array);
```

Things to do:

- Complete the target directive (copies, thread hierarchy,…)
- Complete the task directive (dependences,…)
- Try different complile- (ndrange,…) and run- time options (devices, prefetch,…)

Exercise 3.3: Matrix Multiply

Location: 03-gpu-devices / matmul-cuda

Source code description

- File matmul.c → main code, kernel invokation
- File kernel.cu → Muld CUDA version
- File kernel.h → kernel header (lack of declaration)
- Support files → cclocks, check, driver, gendat and prtspeed

Declare a CUDA task

```
// Kernel declaration as a task should be here
// Remember, we want to multiply two matrices (A*B=C)
// Matrix sizes are NB*NB
```

Things to do:

- Write the target directive (device, copies, thread hierarchy,…)
- Write the task directive (dependences,…)
- Check program execution verification
- Try different complile- (ndrange,…) and run- time options (devices, prefetch,…)

Exercise 3.4: Nbody kernel

Location: 03-gpu-devices / nbody-cuda

Source code description

- File nbody.c → main code, calling SMP version
- File kernel.h → kernel header (lack of declaration)
- File kernel.cu → calculate_force_func CUDA version

Declare and invoke a CUDA task

```
// Kernel declaration as a task should be here
// Remember, we want to declare as a task: calculate_force_func(…)
```

Things to do:

- Write the target directive (device, copies, thread hierarchy,…)
- Write the task directive (dependences,…)
- Invoke the previously created task (instead of the SMP code)
- Check program execution and verification
- Try different complile- (ndrange,…) and run- time options (devices, prefetch,…)
- EXTRA: Use implements clause to allow host/device execution (versionning scheduler)

Exercise 3.5: Cholesky kernel

Location: 03-gpu-devices / cholesky-cuda

Source code description

- File cholesky_hyb.c → hybrid code (SMP,CUDA)
- File cuda_potrf.h → kernel headers
- File cuda_potrf.cu → kernel definitions

Declare and invoke CUDA tasks

- potrf_tile_gpu(…)
- gemm_tile_gpu(…)
- trsfm_tile_gpu(…)
- syrk_tile_gpu(…)

Things to do:

- Write target and task directives (device, copies, thread hierarchy, dependences…)
- Invoke the previously created task (instead of the SMP code)
- Check program execution and verification
- Try different complile- (ndrange,…) and run- time options (devices, prefetch,…)

www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

MPI + OmpSs

PATC: Heterogeneous Programming

Barcelona, May 22nd, 2019

MPI + OmpSs

Exercise's location: 04-gpu-devices

```
~ompss-ee:$ ls
01-examples                 02-beginners
03-gpu-devices              04-mpi+ompss
...
```

MPI + OmpSs (guidelines)

- Review/complete the source code annotation using OmpSs directives
- Run *corresponding* executable version
  - » program-p → performance's version
  - » program-i → instrumentation's version
  - » program-d → debug's version
- Check **`mpirun`** options:
  - » Number of processes
  - » Number of CPUs per process
  - » Binding information (display!!!)

Documents at http:/pm.bsc.es

Hands-on session's contents

Continue previous session's exercises
Exercise 4.1: Matmul
Exercise 4.2: Heat

Remember to configure your system

```
$ source configure.sh
Basic configuration...
```

Check running script (before submit)

```
$ vi run-once.sh
$
```

- Scripts run-once.sh, multi-run.sh or trace.sh
- Executable program's version (-p, -i or -d)
- Job scheduler configuration (queue)
- Other runtime's options used in the script

Exercise 4.1: Matmul benchmark

Location: 04-mpi+ompss / matmul

Source code description

- File matmul.c → communication and computation tasks
- File driver.c → main code
- Multiple files (gendat.c, check.c, cclock.c, pthread.c) → support files

```
#pragma omp task in (a[0:n-1], B[0:m-1]) inout (C[i:i+n-1][0:n-1]) firstprivate (n,m)
cblas_dgemm( <parameters>);
if (it < nodes-1) {
   #pragma omp task in (a[0:n-1]) out (rbuf[0:n-1]) inout(stats) firstprivate \
           (size,m,n,tag,down,up)
   MPI_Sendrecv( <parameters> );
}
```

Things to do:

- Understand, review/modify the OmpSs annotation (taskifying communications)
- Check execution and behaviour for different thread hierarchy configurations
- Check different runtime options (devices, max mem, prefetch, overlap…)
- Check different mpirun options: processes, cpus, binding, etc.

Exercise 4.2: Heat benchmark

Location: 04-mpi+ompss / heat

Source code description

- File heat-mpi-ompss.c → main code, task invokation
- File solver-mpi-ompss.c → jacobi kernel
- File misc.c → some support functions (not annotated)
- File heat.h → source headers
- File test.dat → input data

```
#pragma omp task inout (*uhelp) label (comm) inout (residual)
{
     MPI_Sendrecv(&uhelp[last_row*np], np, MPI_DOUBLE, 1, 0,
                  &uhelp[(last_row+1)*np], np, MPI_DOUBLE, 1, 0,
                  MPI_COMM_WORLD, &status);
}
```

Things to do:

- Understand, review/modify the OmpSs annotation (taskifying communications)
- Check execution and behaviour for different thread hierarchy configurations
- Check different runtime options (devices, max mem, prefetch, overlap…)
- Check different mpirun options: processes, cpus, binding, etc.

www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Thank you!

For further information please contact

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