The Abdus Salam International Centre for Theoretical Physics

2494-21

Workshop on High Performance Computing (HPC) Architecture and Applications in the ICTP

14 - 25 October 2013

Introduction to Parallel Programming, Benchmarking & Profiling

Ivan Girotto
ICTP, Trieste

The Abdus Salam International Centre for Theoretical Physics

Introduction to Parallel Programming, Benchmarking & Profiling

Ivan Girotto – igirotto@ictp.it

Information & Communication Technology Section (ICTS)

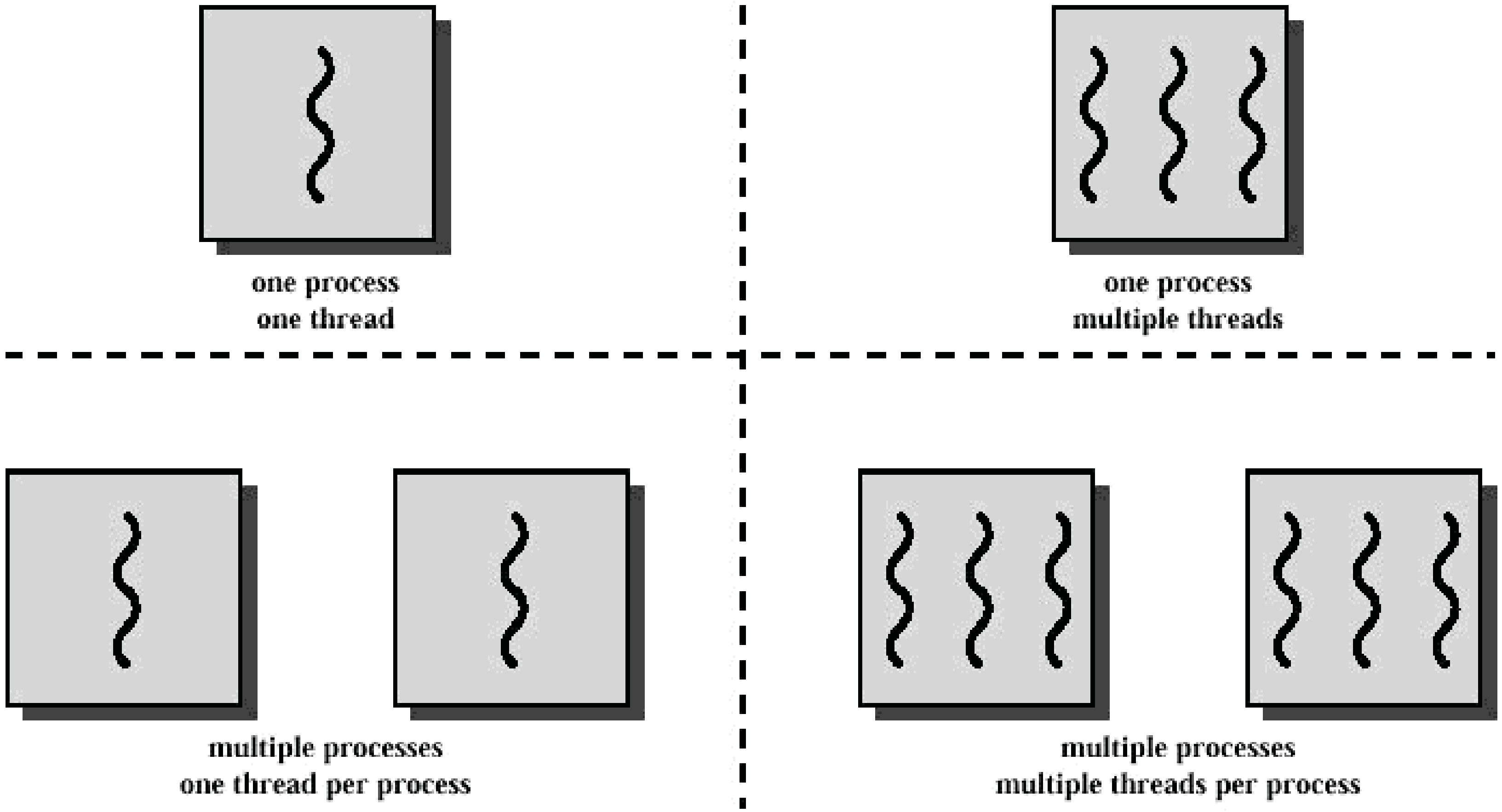
International Centre for Theoretical Physics (ICTP)

Outline

- Introduction to // Programming
- Compiling and Running // Programs
- Benchmarking
- Profiling
- Hands-on

Glossary

- Process: sequence of instructions execute by a given processing unit (core)
  - Unique memory address space
- Thread: sub-entity of a process. Portion of the content of a given process information. Each process can be seen as a single thread.
- Multiple process threads can be replicated to execute single/different instructions on the same/different data. Threads work on SPMD model.


one process
one thread
one process
multiple threads
multiple processes
one thread per process
multiple processes
multiple threads per process

Parallel Programming Models /1

- Shared memory
  - Process threads work together
  - Data are shared among the same address space
  - Communication is performed writing and reading data on a shared portion of the main memory
- Message passing (distribute memory model)
  - Compute processes work together
  - Data are distributed among different address spaces
  - Communication is performed among the exchange of distributed data with operations of send & recv
- On both models the compute units are uniquely identified (ID)

```
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main (int argc, char * argv[]){

 int myId, size, buffer = 0;

#pragma omp parallel private( myId, size )
 {
   myId = omp_get_thread_num();
   size = omp_get_num_threads();

   fprintf( stdout, "\nThread %d of %d. Buffer = %d.", myId, size, buffer);

   if( myId == 0 ) buffer = 1;

#pragma omp barrier

   fprintf( stdout, "\nThread %d of %d. Buffer = %d.", myId, size, buffer);
}

  return 0;
}
```

- Example of an OpenMP program for data exchange

```c
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main (int argc, char * argv[]){

  int myId, size, buffer = 0;

#pragma omp parallel private( myId, size )
  {
    myId = omp_get_thread_num();
    size = omp_get_num_threads();

    fprintf( stdout, "\nThread %d of %d. Buffer = %d.", myId, size, buffer);

    if( myId == 0 ) buffer = 1;

#pragma omp barrier

    fprintf( stdout, "\nThread %d of %d. Buffer = %d.", myId, size, buffer);
}

  return 0;
}
```

- Example of an OpenMP program for data exchange

```c
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>

int main (int argc, char * argv[]){

  int myId, size, buffer = 0;
  MPI_STATUS_SIZE status;

  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &myId );
  MPI_Comm_size( MPI_COMM_WORLD, &size );

  fprintf( stdout, "\nThread %d of %d. Buffer = %d.", myId, size, buffer);

  if( myId == 0 ){
    buffer = 1;
    MPI_Send( &buffer, 1, MPI_INT, 1, 100, MPI_COMM_WORLD );
  }
  else MPI_Recv( &buffer, 1, MPI_INT, 0, 100, MPI_COMM_WORLD, &status )

  fprintf( stdout, "\nThread %d of %d. Buffer = %d.", myId, size, buffer);

  MPI_Finalize();

  return 0;
}
```

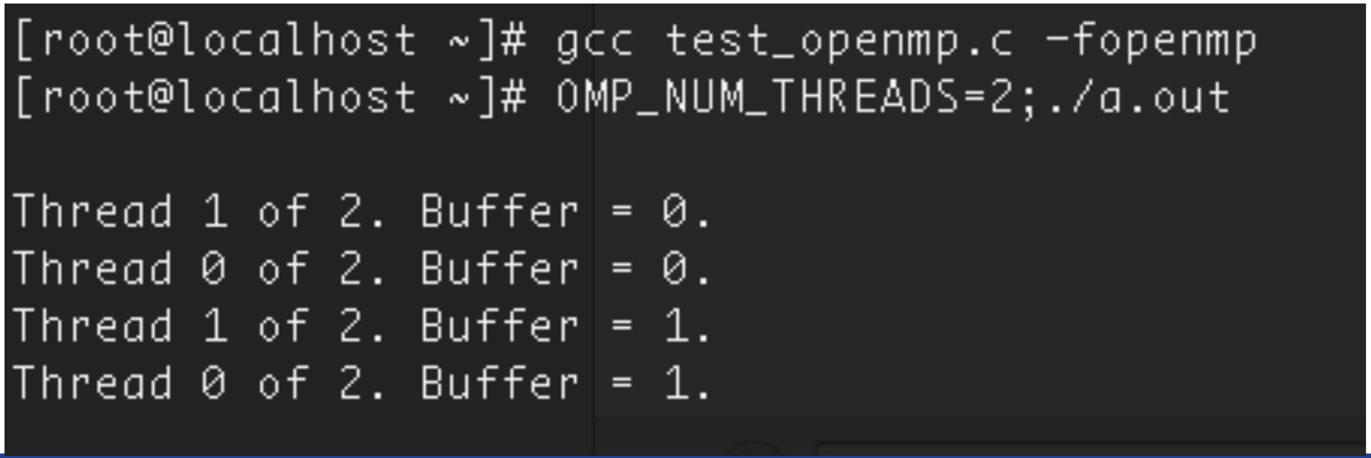
- Example of a MPI program for data exchange

Compile and Run an OpenMP program

gcc -O3 -o my_program.x my_program.c -fopenmp

or

icc -O3 -o my_program.x my_program.c -openmp



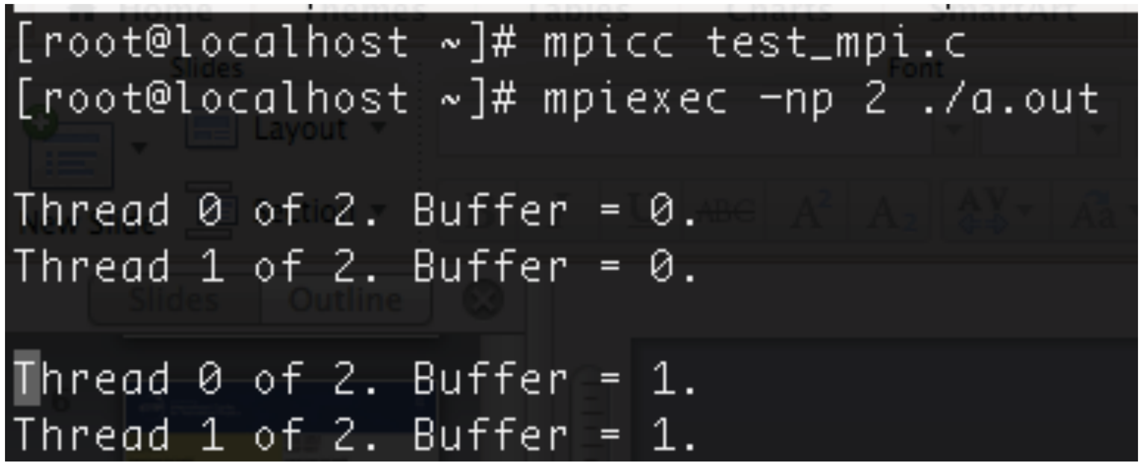
```
[root@localhost ~]# gcc test_openmp.c -fopenmp
[root@localhost ~]# OMP_NUM_THREADS=2;./a.out

Thread 1 of 2. Buffer = 0.
Thread 0 of 2. Buffer = 0.
Thread 1 of 2. Buffer = 1.
Thread 0 of 2. Buffer = 1.
```

Compile and Run an MPI program

mpicc -O3 -o my_program.x my_program.c

or

mpicc -O3 -o my_program.x my_program.c



```
[root@localhost ~]# mpicc test_mpi.c
[root@localhost ~]# mpiexec -np 2 ./a.out

Thread 0 of 2. Buffer = 0.
Thread 1 of 2. Buffer = 0.

Thread 0 of 2. Buffer = 1.
Thread 1 of 2. Buffer = 1.
```

Compile and Run an Hybrid program

mpicc -O3 -o my_program.x my_program.c -fopenmp

or

mpicc -O3 -o my_program.x my_program.c -openmp

```
[root@localhost ~]# mpicc -o test_hybrid.x test_hybrid.c -openmp
[root@localhost ~]# export OMP_NUM_THREADS=2
[root@localhost ~]# mpiexec -np 2 ./test_hybrid.x
```

CPU Architecture Benchmark /1

- Measure of Performance
- Architecture Assessment
- Basic key-points for obtaining useful outcomes:
  - Define what are the measures of interest
  - Define what are the tools
  - Define what is the SW environment
  - Define reasonable workloads
  - Data mining is essential (sooner or later)
- Synthetic Benchmark Vs. Application Benchmark

CPU Architecture Benchmark /2

- DGEMM - double precision Matrix-Matrix Multiplication to measure FLOP/s
  - C := alpha*op(A)*op(B) + beta*C
- DAXPY - adds a scalar multiple of a double precision vector to another double precision vector to measure memory bandwidth
  - y <-- alpha*x + y

Benchmark	DATA	FLOP
DAXPY	O(DIM)	O(DIM)
DGEMM	O(DIM^2)	O(DIM^3)

```
PROGRAM bench_dgemm
  IMPLICIT NONE
  INTEGER, PARAMETER :: dim = 8192
  REAL*8, ALLOCATABLE :: x(:,:), y(:,:), z(:,:)
  ALLOCATE( x( dim, dim ), y( dim, dim ), z( dim, dim ) )
  y = 1.0d0 / ( DBLE( dim ) )
  z = 1.0d0 / ( DBLE( dim ) )
  x = 0.0d0
  t1 = cclock()
  call dgemm('N', 'N', dim, dim, dim, 1.0d0, y, dim, z, dim,  0.0d0, x, dim)
  t2 = cclock()
  [...]
  DEALLOCATE( x, y, z )
END PROGRAM bench_dgemm
```

```
PROGRAM bench_dgemm
  IMPLICIT NONE
  INTEGER, PARAMETER :: dim = 8192
  REAL*8, ALLOCATABLE :: x(:,:), y(:,:), z(:,:)
  ALLOCATE( x( dim, dim ), y( dim, dim ), z( dim, dim ) )
  y = 1.0d0 / ( DBLE( dim ) )
  z = 1.0d0 / ( DBLE( dim ) )
  x = 0.0d0
  t1 = cclock()
  call dgemm('N', 'N', dim, dim, dim, 1.0d0, y, dim, z, dim,  0.0d0, x, dim)
  t2 = cclock()
  [...]
  DEALLOCATE( x, y, z )
END PROGRAM bench_dgemm
```

```
gcc -03 [...] ../LIBS/BLAS/blas_LINUX.a
```

```
PROGRAM bench_dgemm
  IMPLICIT NONE
  INTEGER, PARAMETER :: dim = 8192
  REAL*8, ALLOCATABLE :: x(:,:), y(:,:), z(:,:)
  ALLOCATE( x( dim, dim ), y( dim, dim ), z( dim, dim ) )
  y = 1.0d0 / ( DBLE( dim ) )
  z = 1.0d0 / ( DBLE( dim ) )
  x = 0.0d0
  t1 = cclock()
  call dgemm('N', 'N', dim, dim, dim, 1.0d0, y, dim, z, dim,  0.0d0, x, dim)
  t2 = cclock()
  [...]
  DEALLOCATE( x, y, z )
END PROGRAM bench_dgemm
```

```
gcc -03 [...] -L/opt/intel/mkl/lib/intel64/ -lmkl_gf_lp64  -lmkl_gnu_thread -lmkl_core
```

```
PROGRAM bench_dgemm
 IMPLICIT NONE
 INTEGER, PARAMETER :: dim = 8192
 REAL*8, ALLOCATABLE :: x(:,:), y(:,:), z(:,:)
 ALLOCATE( x( dim, dim ), y( dim, dim ), z( dim, dim ) )
 y = 1.0d0 / ( DBLE( dim ) )
 z = 1.0d0 / ( DBLE( dim ) )
 x = 0.0d0
 t1 = cclock()
 call dgemm('N', 'N', dim, dim, dim, 1.0d0, y, dim, z, dim,  0.0d0, x, dim)
 t2 = cclock()
 [...]
 DEALLOCATE( x, y, z )
END PROGRAM bench_dgemm
```

```
ifort -03 [...] -L/opt/intel/mkl/lib/intel64/ -lmkl_intel_lp64  -lmkl_intel_thread -lmkl_core
```

```
PROGRAM bench_dgemm
 IMPLICIT NONE
 INTEGER, PARAMETER :: dim = 8192
 REAL*8, ALLOCATABLE :: x(:,:), y(:,:), z(:,:)
 ALLOCATE( x( dim, dim ), y( dim, dim ), z( dim, dim ) )
 y = 1.0d0 / ( DBLE( dim ) )
 z = 1.0d0 / ( DBLE( dim ) )
 x = 0.0d0
 t1 = cclock()
 call dgemm('N', 'N', dim, dim, dim, 1.0d0, y, dim, z, dim,  0.0d0, x, dim)
 t2 = cclock()
 [...]
 DEALLOCATE( x, y, z )
END PROGRAM bench_dgemm
```

```
ifort -03 [...] -L/opt/intel/mkl/lib/intel64/ -lmkl_intel_lp64  -lmkl_intel_thread -lmkl_core
gfortran -03 [...] -L${ACML_ROOT}/gfortran64/lib -lacml_mp
```

```
PROGRAM bench_dgemm
 IMPLICIT NONE
 INTEGER, PARAMETER :: dim = 8192
 REAL*8, ALLOCATABLE :: x(:,:), y(:,:), z(:,:)
 ALLOCATE( x( dim, dim ), y( dim, dim ), z( dim, dim ) )
 y = 1.0d0 / ( DBLE( dim ) )
 z = 1.0d0 / ( DBLE( dim ) )
 x = 0.0d0
 t1 = cclock()
 call dgemm('N', 'N', dim, dim, dim, 1.0d0, y, dim, z, dim,  0.0d0, x, dim)
 t2 = cclock()
 [...]
 DEALLOCATE( x, y, z )
END PROGRAM bench_dgemm
```

```
ifort -03 -mmic [...] -L/opt/intel/mkl/lib/mic/ -lmkl_intel_lp64  -lmkl_intel_thread -lmkl_core
```

The Abdus Salam International Centre for Theoretical Physics

```
PROGRAM bench_dgemm
  IMPLICIT NONE
  INTEGER, PARAMETER :: dim = 8192
  REAL*8, ALLOCATABLE :: x(:,:), y(:,:), z(:,:)
  ALLOCATE( x( dim, dim ), y( dim, dim ), z( dim, dim ) )
  y = 1.0d0 / ( DBLE( dim ) )
  z = 1.0d0 / ( DBLE( dim ) )
  x = 0.0d0
  t1 = cclock()
  call dgemm('N', 'N', dim, dim, dim, 1.0d0, y, dim, z, dim,  0.0d0, x, dim)
  t2 = cclock()
  [...]
  DEALLOCATE( x, y, z )
END PROGRAM bench_dgemm
```

```
nvcc -03  [...] -L${CUDA_HOME}/lib64/ -lcublas
```

CPU Architecture Benchmark /2

High Performance Linpack

HPC Challenge

Profiling

- Profiling usually means:
  - Instrumentation of code (e.g. during compilation)
  - Automated collection of timing data during execution
  - Analysis of collected data, breakdown by function
- Example:
  - `$gcc -o some_exe.x -pg some_code.c`
  - `$./some_exe.x`
  - `gprof some_exe.x gmon.out`
- Profiling is often incompatible with code optimization or can be misleading (inlining). But profiling DOES require to be performed with the production setting

PERF – Hardware Assisted Profiling

- Modern x86 CPUs contain performance monitor tools included in their hardware
- Linux kernel versions support this feature which allows for very low overhead profiling without instrumentation of binaries
- The perf command:
  - perf stat ./a.out -> profile summary
  - **perf record ./a.out; perf report**
  - **perf -h; perf [command] -h**

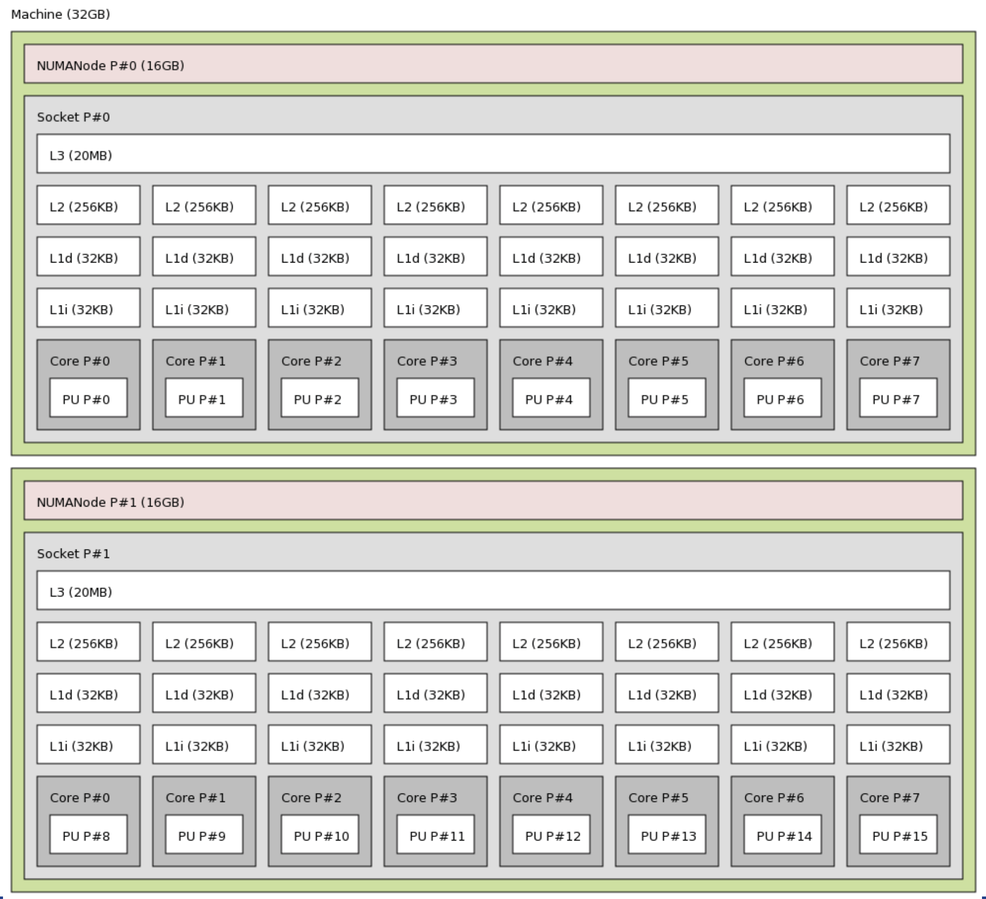
```
# gfortran -pg prog1.f ; ./a.out ; gprof --flat-profile ./a.out gmon.out

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.69      4.30     4.30    10000     0.00     0.00  xaver_
  0.00      4.30     0.00        1     0.00     4.30  MAIN__

# make CFLAGS=-pg mountain ; ./mountain ; gprof -p mountain gmon.out

  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 98.30      5.80     5.80     3206     1.81     1.81  test
  1.87      5.91     0.11        1   110.19   110.19  init_data
  0.00      5.91     0.00     2920     0.00     0.00  access_counter
  0.00      5.91     0.00     1460     0.00     0.00  get_counter
  0.00      5.91     0.00     1460     0.00     0.00  start_counter
  0.00      5.91     0.00     1459     0.00     0.00  add_sample
  0.00      5.91     0.00     1459     0.00     0.00  has_converged
  0.00      5.91     0.00      288     0.00    18.33  fcyc2
  0.00      5.91     0.00      288     0.00    18.33  fcyc2_full
[...]
```

The Abdus Salam International Centre for Theoretical Physics

Hands-on

LAB-SESSION

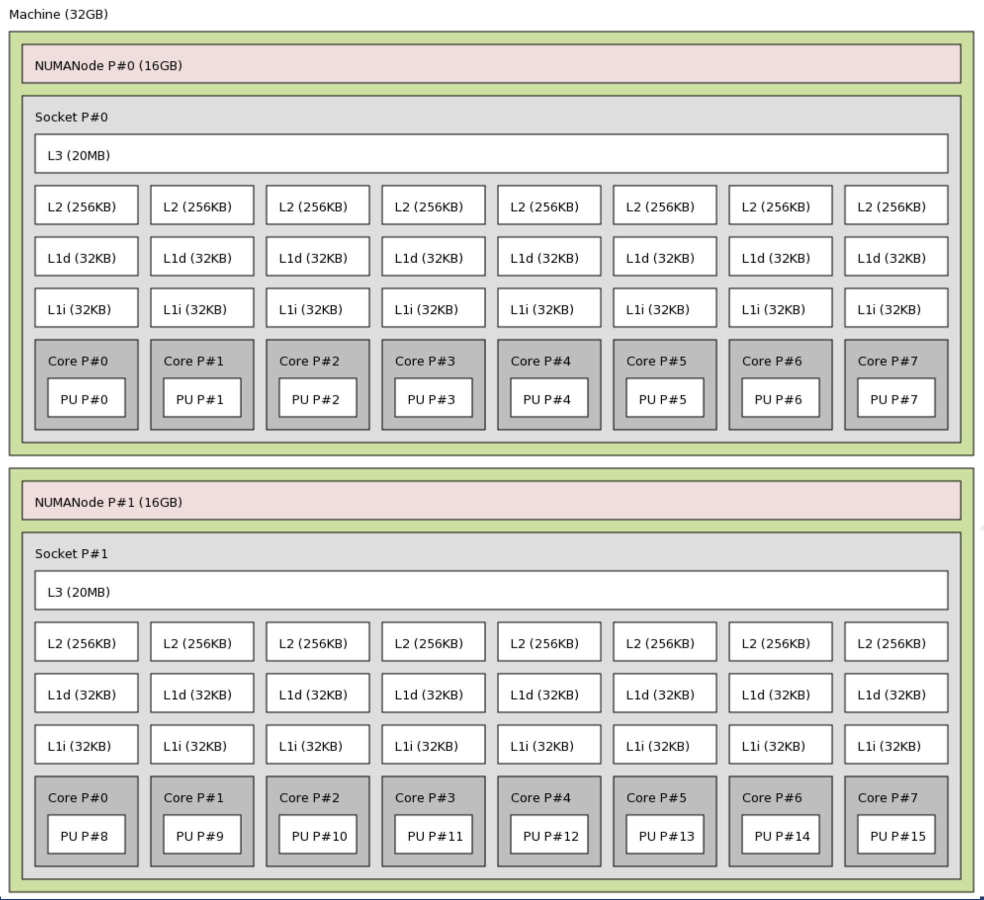
The Intel Xeon E5-2665 Sandy Bridge-EP 2.4GHz

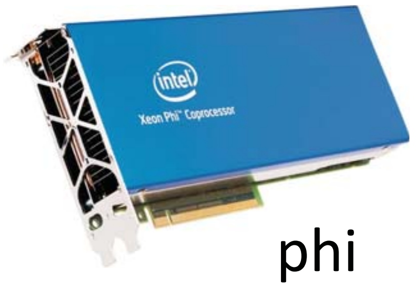


cpu-only

The Intel Xeon E5-2665 Sandy Bridge-EP 2.4GHz

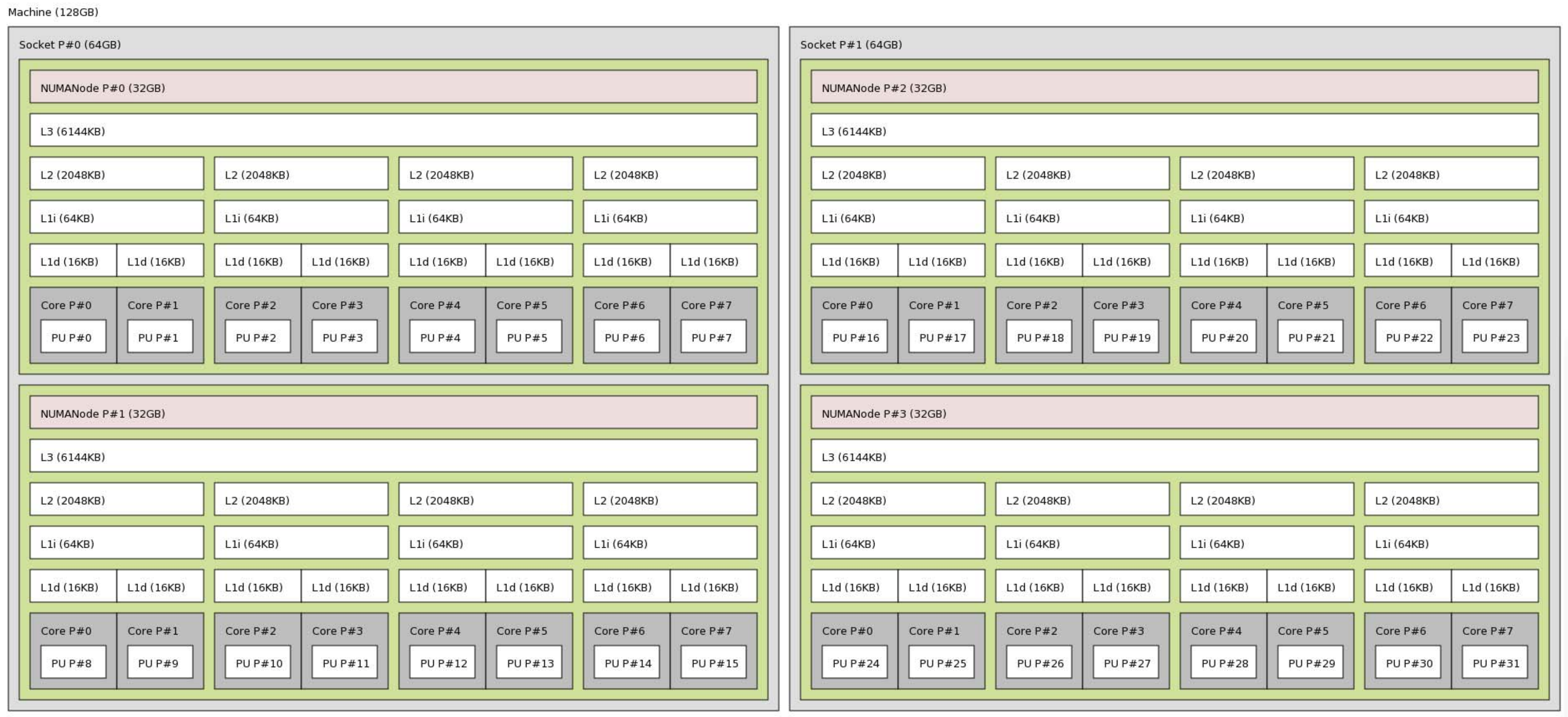
gpu


Machine (32GB)
NUMANode P#0 (16GB)
Socket P#0
L3 (20MB)
L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB)
L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB)
L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB)
Core P#0 PU P#0
Core P#1 PU P#1
Core P#2 PU P#2
Core P#3 PU P#3
Core P#4 PU P#4
Core P#5 PU P#5
Core P#6 PU P#6
Core P#7 PU P#7
NUMANode P#1 (16GB)
Socket P#1
L3 (20MB)
L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB)
L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB)
L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB)
Core P#0 PU P#8
Core P#1 PU P#9
Core P#2 PU P#10
Core P#3 PU P#11
Core P#4 PU P#12
Core P#5 PU P#13
Core P#6 PU P#14
Core P#7 PU P#15

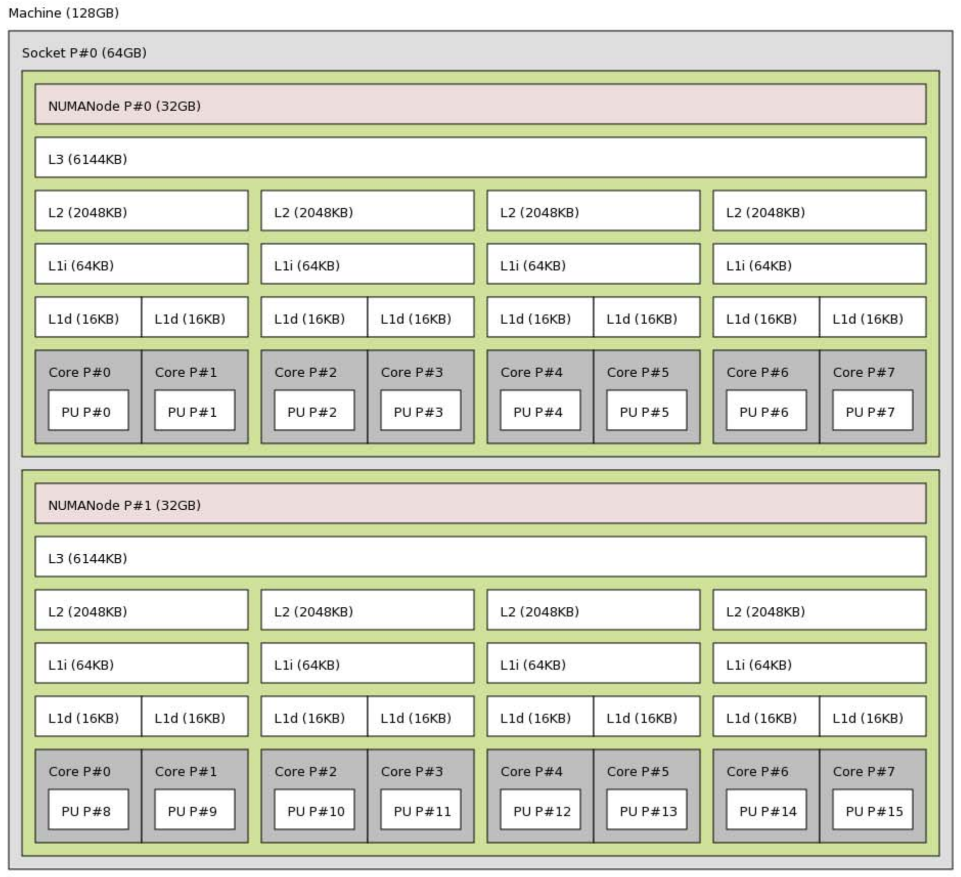


phi

The AMD Opteron 6380 Abu Dhabi 2.5GHz


Machine (128GB)
Socket P#0 (64GB)
NUMANode P#0 (32GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1i (64KB) L1i (64KB) L1i (64KB) L1i (64KB)
L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB)
Core P#0 PU P#0
Core P#1 PU P#1
Core P#2 PU P#2
Core P#3 PU P#3
Core P#4 PU P#4
Core P#5 PU P#5
Core P#6 PU P#6
Core P#7 PU P#7
NUMANode P#1 (32GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1i (64KB) L1i (64KB) L1i (64KB) L1i (64KB)
L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB)
Core P#0 PU P#8
Core P#1 PU P#9
Core P#2 PU P#10
Core P#3 PU P#11
Core P#4 PU P#12
Core P#5 PU P#13
Core P#6 PU P#14
Core P#7 PU P#15
Socket P#1 (64GB)
NUMANode P#2 (32GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1i (64KB) L1i (64KB) L1i (64KB) L1i (64KB)
L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB)
Core P#0 PU P#16
Core P#1 PU P#17
Core P#2 PU P#18
Core P#3 PU P#19
Core P#4 PU P#20
Core P#5 PU P#21
Core P#6 PU P#22
Core P#7 PU P#23
NUMANode P#3 (32GB)
L3 (6144KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1i (64KB) L1i (64KB) L1i (64KB) L1i (64KB)
L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB)
Core P#0 PU P#24
Core P#1 PU P#25
Core P#2 PU P#26
Core P#3 PU P#27
Core P#4 PU P#28
Core P#5 PU P#29
Core P#6 PU P#30
Core P#7 PU P#31

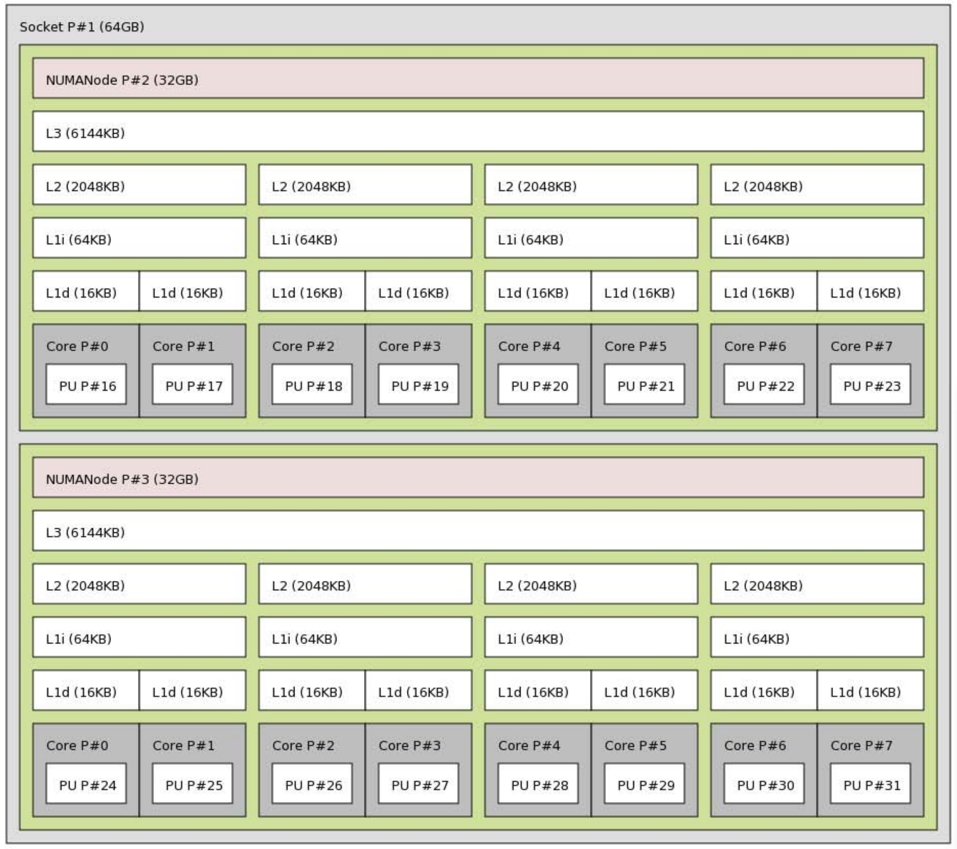
The AMD Opteron 6380 Abu Dhabi 2.5GHz



Socket #0 is used as Login Node

The AMD Opteron 6380 Abu Dhabi 2.5GHz

Socket #1 is used as Compute Node



ssh details

- AMD
  - ssh user[N]@94.82.133.184 -p 55566
- GPU-server
  - ssh user[N]@gpu-server
- password user[N] ... with N ∈ [1,24]

Exercises

1. Make a performance analysis using simple synthetic benchmarks (DGEMM, DAXPY) among the available architecture
2. Plot a scaling curve of the results obtained using DGEMM benchmark on multiple threads among the two computer architectures

Exercises Advanced /1

- Benchmark aggregate memory bandwidth using bench_mpi_daxpy.f90 measuring the scaling of the aggregate bandwidth at the increasing of the number of processes (use -D__MPI preprocessor macro)
- Perform a performance analysis using the pw.x binary of the Quantum-ESPRESSO package - application benchmark. Input provided in ex-lab/QE-Input on the gpu-server
  - mpiexec -np [N] ./pw.x -input input.it | tee output.out

Exercises Advanced /3

- Make a profiling analysis of either the provided synthetic benchmarks or an application benchmark at your choice.
- Use numactl or hwloc to map concurrent execution of processes among different cores. Analyze the increasing of cache-misses (perf command) while increasing the processes/ threads density

Exercises Advanced /4

- Run the benchmark on the INTEL PHI co-processors (no instructions template). See the list of links below:
  - Link1 (.pdf at the bottom of the page)
  - Native Compilation, Automatic Offload, Compiler Assisted Offload

Script Examples

- Interactive session:
  - qsub -I -q [queu] -l walltime=00:10:00 -l nodes=1:ppn=[cores_number_request]

The Abdus Salam International Centre for Theoretical Physics

Best Results Obtained in Class

Benchmark	NVIDIA TITAN GPU	INTEL Xeon PHI	Xeon E5-2665 (2 - sockets)	Opteron 6380 (1 - socket)
DGEMM (GFlop/s)	1.1	766	164 (8 cores)	126 (8 cores)
			320 (16 cores)	104 (16 cores)
DAXPY (GByte/s)			12.8	7
MPI_DAXPY (GByte/s)			41	34

The Abdus Salam
International Centre
for Theoretical Physics

Thanks for your attention!!