

Hands onExample



Process/ Thread Affinity

1. Logical Core layout
2. MPI Example
3. OpenMP Example

Suzanne Parete-Koon



U.S. DEPARTMENT OF
ENERGY



OAK RIDGE NATIONAL LABORATORY

MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY

Motivation

- Motivation: To show you how to run and compile an application by playing with core affinity.
- Core affinity is effectively the layout of the processes/threads on the “cores”.
- Understanding the default layout of threads and process on the cores and how to manipulate core affinity can help avoid performance bottlenecks.
- With Hyper threading there is an extra layer of complication because each physical core becomes two logical cores.
- The following will give you some codes to help illustrate the various options.

XC30 Compute Node

NUMA Node 0

Physical Core 0

L1

L2

Physical Core 1

L1

L2

Physical Core 2

L1

L2

Physical Core 3

L1

L2

Physical Core 4

L1

L2

Physical Core 5

L1

L2

Physical Core 6

L1

L2

Physical Core 7

L1

L2

L3 Cache

NUMA Node 1

Physical Core 8

L1

L2

Physical Core 9

L1

L2

Physical Core 10

L1

L2

Physical Core 11

L1

L2

Physical Core 12

L1

L2

Physical Core 13

L1

L2

Physical Core 14

L1

L2

Physical Core 15

L1

L2

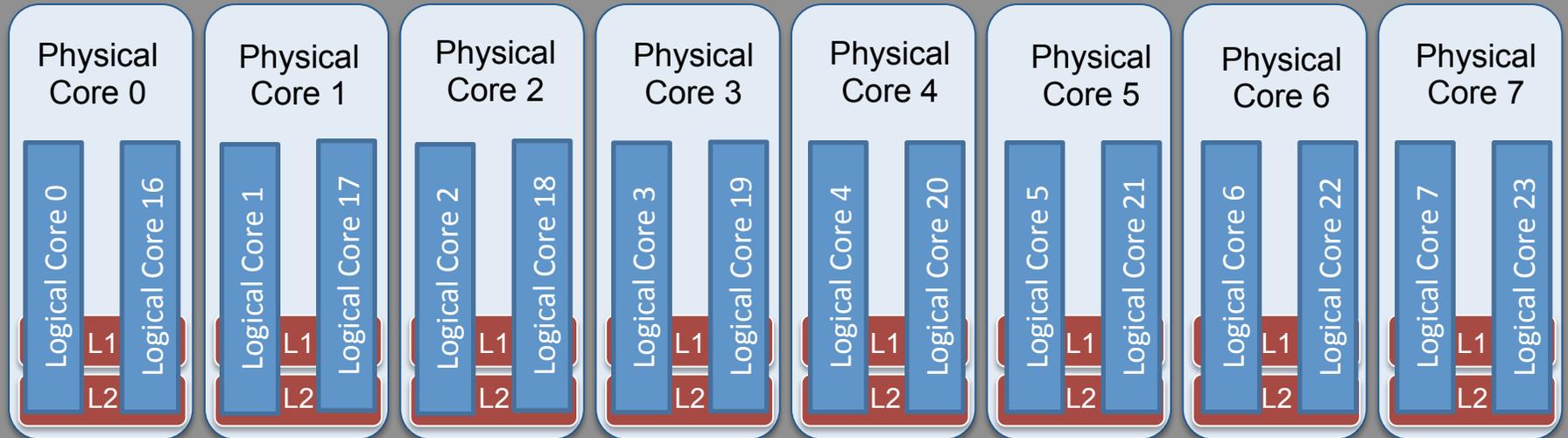
L3 Cache

List of 32 cores via `aprun cat /proc/cpuinfo`

```
processor      : 0 ← Physical Core ID
vendor_id     : GenuineIntel
cpu family    : 6
model         : 45
model name    : Xeon(R) CPU E5-2670 0 @ 2.60GHz
stepping      : 7
cpu MHz       : 2601.000
cache size    : 20480 KB
physical id   : 0 ← NUMA NODE
siblings      : 16
core id       : 0 ← Logical Core ID
cpu cores     : 8
. . .
```

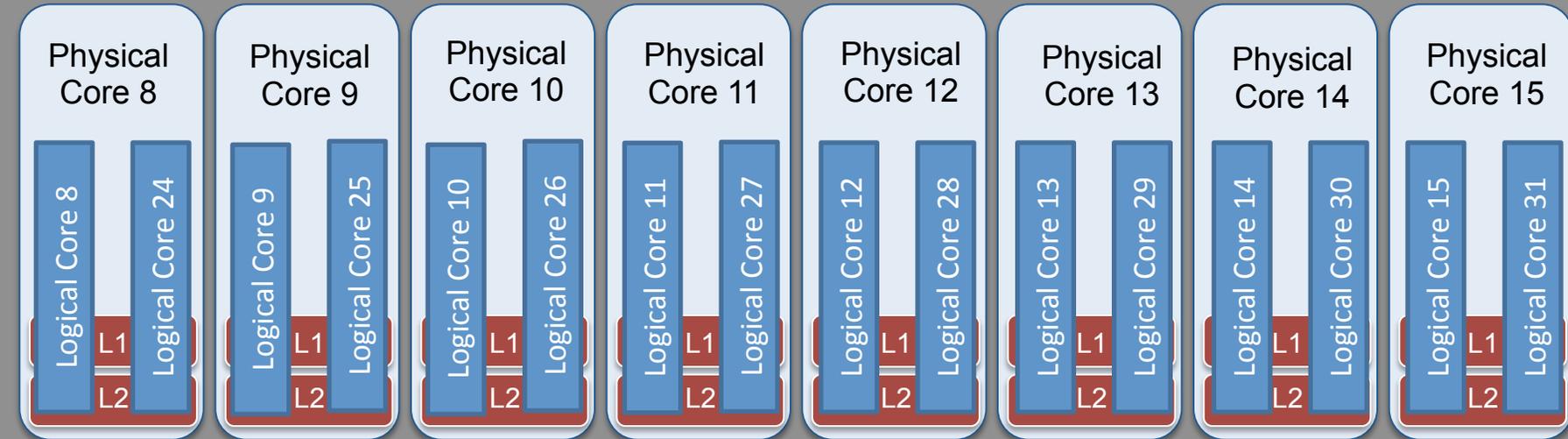
XC30 Compute Node

NUMA Node 0



L3 Cache

NUMA Node 1



L3 Cache

Instructions

The following example is a simple MPI/Open program, Xith.c that shows how processes/threads are placed on the cores. We will use the Intel compiler for most of this. The exercises are designed to allow you to explore the core layout and process placement.

1. Login to Eos/Dater and git files/

```
% module load git.  
% git clone https://github.com/olcf/XC30-Training.git  
% cd XC30-Training
```

Instructions

2. Copy example folder to your scratch area and compile it with the Intel compiler.

Eos:

```
%cp -r affinity $MEMBERWORK/projid  
%cd $MEMBERWORK/projid/affinity  
% cc -openmp Xith.c
```

Darter:

```
% cp -r affinity /lustre/snx/username  
% module swap PrgEnv-Cray PrgEnv-Intel  
% cd /lustre/snx/username/affinity  
% cc -openmp Xith.c
```

Instructions

3. Look at batch script, aff.pbs, and use it to start a job on 1 node.

```
% vi aff.pbs  
% qsub aff.pbs
```

```
#!/bin/bash  
# Begin PBS directives  
#PBS -A STF007  
#PBS -N affinity  
#PBS -j oe  
#PBS -l walltime=00:05:00,nodes=1  
# End PBS directives and begin shell  
commands  
cd $MEMBERWORK/stf007  
aprun -n 16 ./a.out
```

Instructions

The code is a hello world that prints out the node, rank, thread, and “logical” core for all the tasks running. Ranks 0 and 1 have been given some labor- to generate 1000000 random numbers and do some multiplication. All ranks have a timer.

Test1: What do I get with basic hyper threading?

Try no hyper threading : `aprun -n 16`

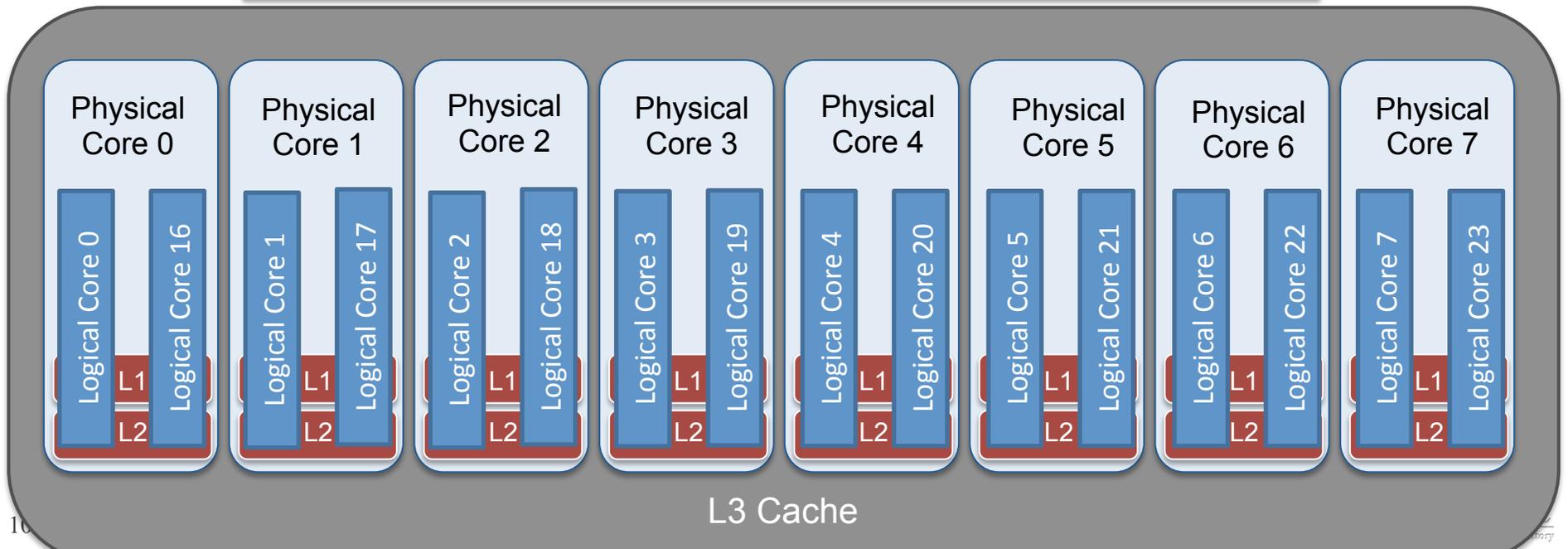
Try `aprun -n 32` (what did we forget?!)

Try `aprun -n 32 -j2`

aprun -n32 -j2 ./a.out

Consecutive ranks fall on the same physical core.

Rank 0, Node 00763, Core 0 ,physical 0
Rank 1, Node 00763, Core 16, physical 0
Rank 2, Node 00763, Core 1, physical 1
Rank 3, Node 00763, Core 17, physical 1
.
.
.
Rank 30, Node 00763, Core 15, physical 31
Rank 31, Node 00763, Core 31, physical 31



Instructions Test 2

What happens if I use hyper threading on an unpacked node?

Here we will look at the effect of the core affinity aprun option, cc.

-cc enables you to bind a processing element (pe) to a particular CPU or a subset of CPUs on a node in a controlled manner.

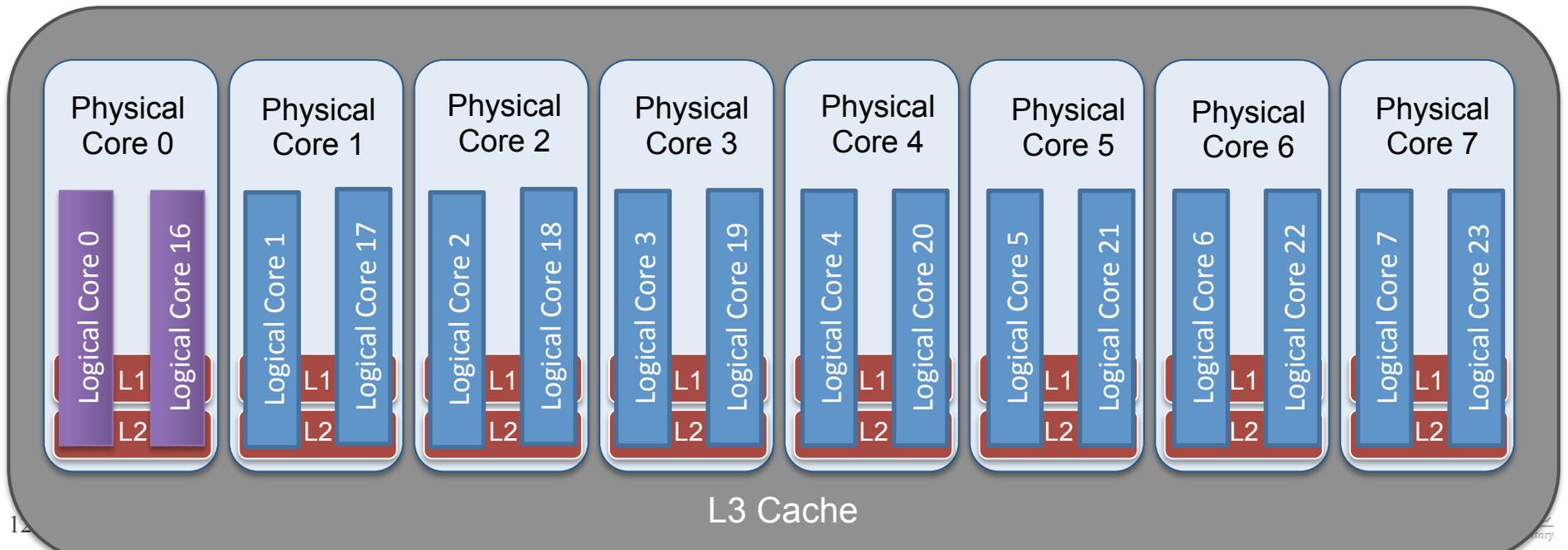
To get the details:

`%man aprun`

Instructions Test 2

Lets look at the default cc behavior

Try `% aprun -n 4 -j2 ./a.out`

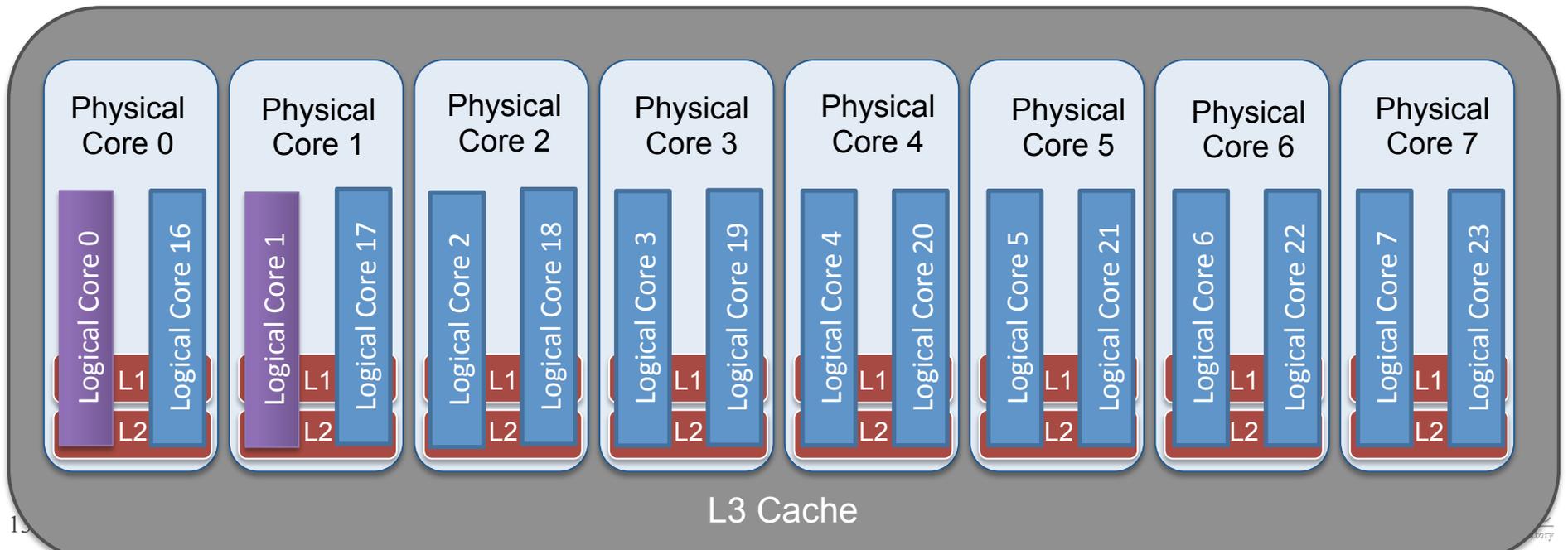


Instructions Test 2

-cc 0-3 will bind the first 4 successive ranks to the first 4 successive cores.

Try `% aprun -n 4 -j2 -cc 0-3 ./a.out`

Try `% aprun -n 4 ./a.out` (Should look familiar)

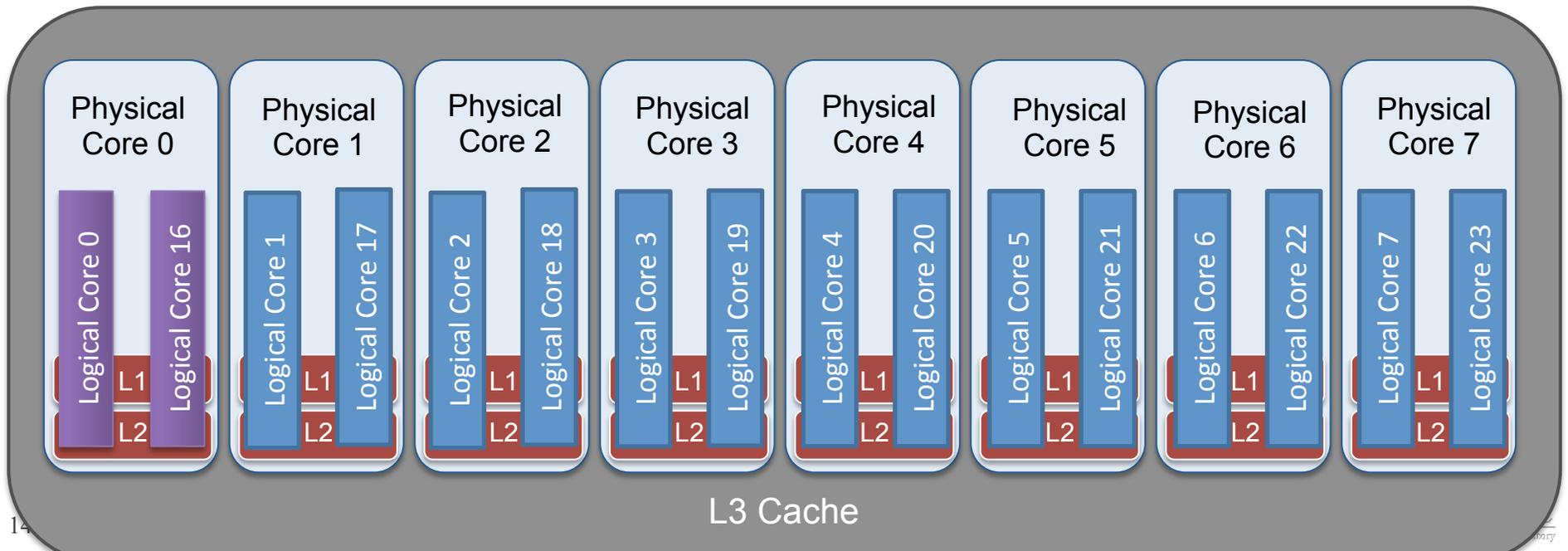


Instructions Test 2

-cc 0-3 will bind the first 4 successive ranks to the first 4 successive cores. The default is `-cc cpu` which binds ranks to each “core” round robin on the node.

Try `% aprun -n 4 ./a.out`

Should look familiar



Test 3 Threading

Modify your batch script:

```
% vi aff.pbs
```

```
#!/bin/bash
# Begin PBS directives
#PBS -A STF007
#PBS -N affinity
#PBS -j oe
#PBS -l walltime=00:05:00,nodes=1
# End PBS directives and begin shell
commands
export OMP_NUM_THREADS=16
cd $MEMBERWORK/stf007
aprun -n 2 -d 16 -j2 ./a.out
```

Test 3 Threading

Modify your batch script:

```
% vi aff.pbs
```

```
#!/bin/bash
# Begin PBS directives
#PBS -A STF007
#PBS -N affinity
#PBS -j oe
#PBS -l walltime=00:05:00,nodes=1
# End PBS directives and begin shell
commands
export OMP_NUM_THREADS=16
cd $MEMBERWORK/stf007
aprun -n 2 -d 16 -j2 ./a.out
```

```
% qsub aff.pbs
```

Instructions Test 3 Threading

The -d option assigns depth; the number of cores per processing element.

In this case our processing element (pe) is an MPI task that spawns 16 threads.

We gave the pe a depth of 16 so, 16 threads use 16 cores.

Test 3 Threading

Modify your batch script:

```
% vi aff.pbs
```

```
#!/bin/bash
# Begin PBS directives
#PBS -A STF007
#PBS -N affinity
#PBS -j oe
#PBS -l walltime=00:05:00,nodes=2
# End PBS directives and begin shell
commands
export OMP_NUM_THREADS=16
cd $MEMBERWORK/stf007
aprun -n 2 -d 16 ./a.out
```

```
% qsub aff.pbs
```

Instructions MPI Example

For many more examples with this code of how to see the man page for `aprun`.

Example 7: Optimizing NUMA-node memory references (`-S` option)

This example uses the `-S` option to restrict placement of PEs to one per NUMA node. Two compute nodes are required, with one PE on NUMA node 0 and one PE on NUMA node 1:

```
% aprun -n 4 -S 1 ./xthi | sort
Application 225117 resources: ~0s, stime ~0s
Hello from rank 0, thread 0, on nid00043. (core affinity = 0)
Hello from rank 1, thread 0, on nid00043. (core affinity = 4)
Hello from rank 2, thread 0, on nid00044. (core affinity = 0)
Hello from rank 3, thread 0, on nid00044. (core affinity = 4)
```

Example 8: Optimizing NUMA-node memory references (`-sl` option)

This example runs all PEs on NUMA node 0; the PEs cannot allocate remote NUMA node memory:

```
% aprun -n 8 -sl 0 ./xthi | sort
Application 225118 resources: utime ~0s, stime ~0s
Hello from rank 0, thread 0, on nid00028. (core affinity = 0)
Hello from rank 1, thread 0, on nid00028. (core affinity = 1)
Hello from rank 2, thread 0, on nid00028. (core affinity = 2)
Hello from rank 3, thread 0, on nid00028. (core affinity = 3)
Hello from rank 4, thread 0, on nid00029. (core affinity = 0)
```

Questions?

Hyper Threading /proc/cpuinfo

Default is to run with one process/thread per physical core. aprun option `-j2` allows two processes per physical core. “Hyper Threading “on”.

To see what that looks like on Eos:

```
eos% qsub -l -A projID -lnodes=1,walltime=01:00:00
eos-login2% cd $MEMBERWORK/projid
[eos-login2% aprun cat /proc/cpuinfo
```

To see what that looks like on Darter

```
Darter% qsub -l -A PrgID 3 -lsize=32,walltime=01:00:00
Darter% cd lustre/snx/username
Darter% aprun cat /proc/cpuinfo
```

aprun -n32 -j2 -cc 0-31 ./a.out

Consecutive ranks do not fall on the same physical core.

Rank 0, Node 00763, Core 0 ,physical core 0
Rank 16, Node 00763, Core 16, physical core 0

Rank 1, Node 00763, Core 1, physical core1
Rank 17, Node 00763, Core 17, physical core1

Rank 2, Node 00763, Core 2, physical core2
Rank 18, Node 00763, Core 18, physical core2

. . .

Rank 15, Node 00763, Core 15,Physical core 16
Rank 31, Node 00763, Core 31, Physical core 16

aprun -n32 -j2 -cc numa_node ./a.out

This allows process to migrate with in a nuam domaine

Rank 0, Node 00757, Core 0-7,16-23

Rank 1, Node 00757, Core 0-7,16-23

Rank 2, Node 00757, Core 0-7,16-23

. . .

Rank 29, Node 00757, Core 8-15,24-31

Rank 30, Node 00757, Core 8-15,24-31

Rank 31, Node 00757, Core 8-15,24-31

You would need to user `-cc numa_node` if you were trying to use Opemp in the Intel compiler enviroment with thread depth that divides evenly in to the nubmer of “cores”.