TACC
TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

Hybrid Computing

TACC OpenMP Tutorial

PRESENTED BY:

Lars Koesterke

Kent Milfeld

MPI & OpenMP == Hybrid Programs

MPI and OpenMP in a single code

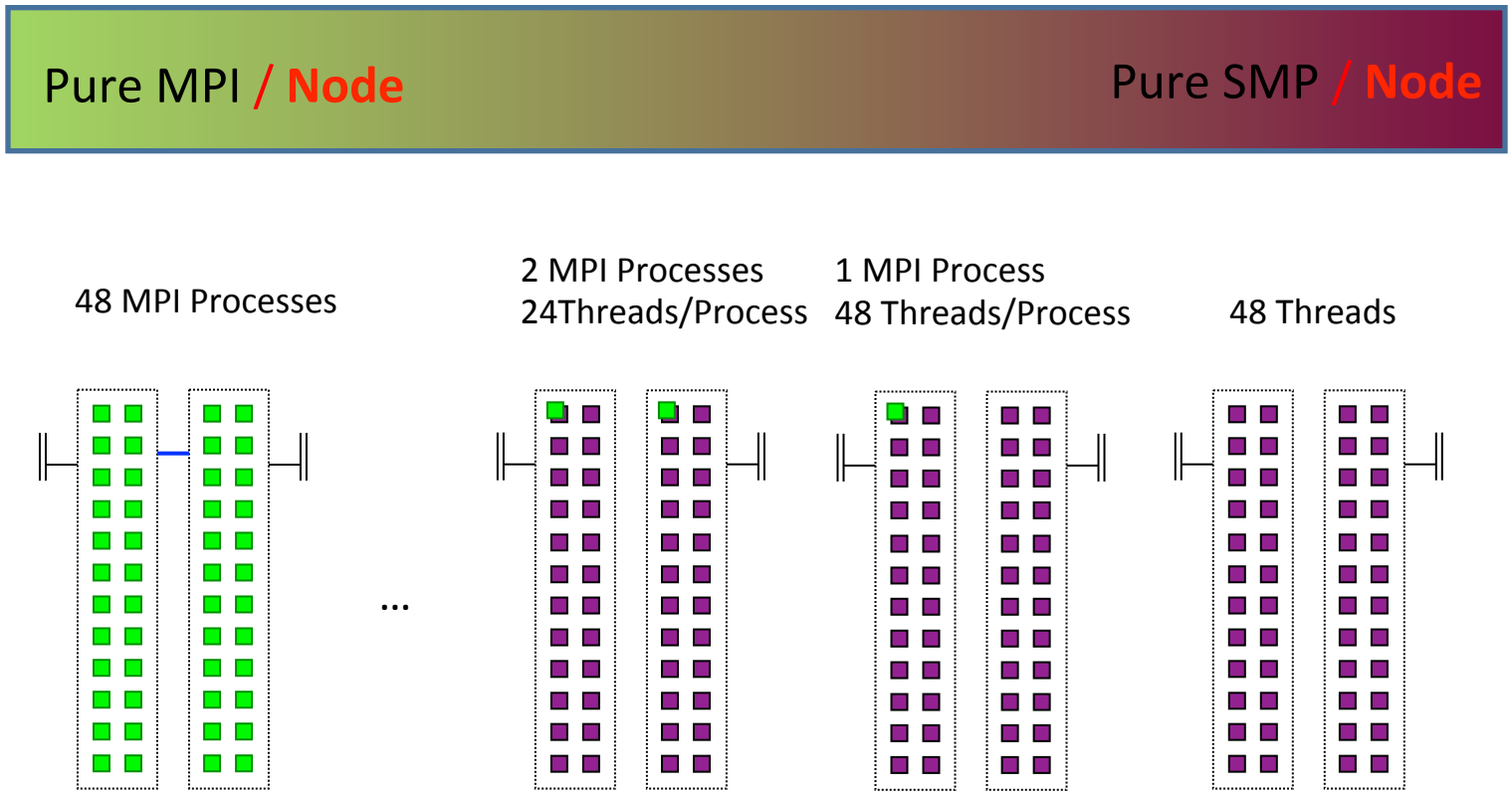
MPI process acts as a container for OpenMP threads*.

- Makes sense since MPI processes don't share memory.
  - OpenMP threads can access all of memory within a MPI process. Threads of one MPI process cannot access memory in another MPI process.
- Use MPI parallelism across Nodes and OpenMP within a Nodes.
- But, you may see multiple MPI processes on a node!

*Actually you can initiate MPI within a parallel region–
but there are restrictions and there is no reasonable use case.

Modes of MPI/OpenMP Operation

Hardware View



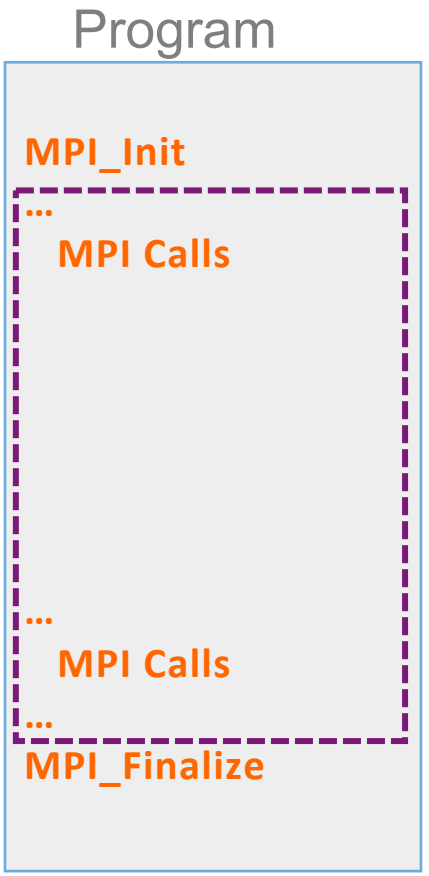
Model based on Stampede2 Skylake node:

MPI – Program Model
Usually a Container for Threading

Start with MPI_Init

MPI Calls

End with MPI_Finalize



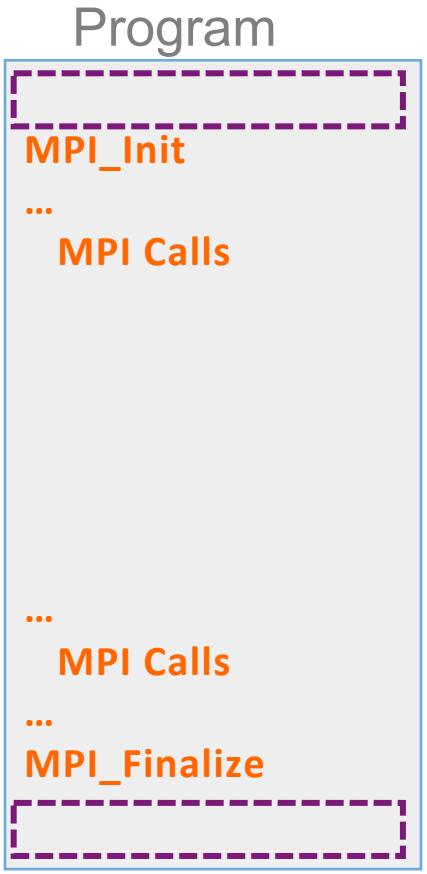
OpenMP Parallel regions most often appear between MPI Initialize and Finalize.

MPI – Program Model
Unusual for threading outside of MPI

Start with MPI_Init

MPI Calls

End with MPI_Finalize



Unusual -- impractical to have OpenMP without availability of MPI Communications.

Hybrid – Program Model In Serial region only

Start with MPI_Init

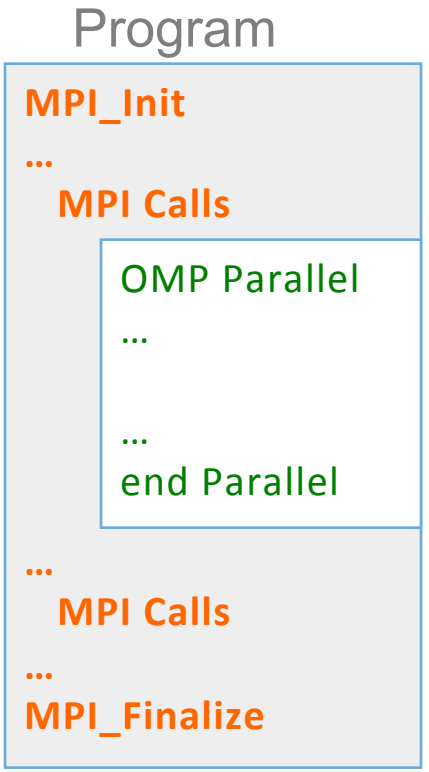
MPI Calls in Serial regions ONLY

- Serial regions MPI calls use the “master thread”.

OMP parallel regions

- No MPI Calls within parallel region

End with MPI_Finalize



Hybrid – Program Model
Thread Safe MPI

Start with MPI_Init_thread

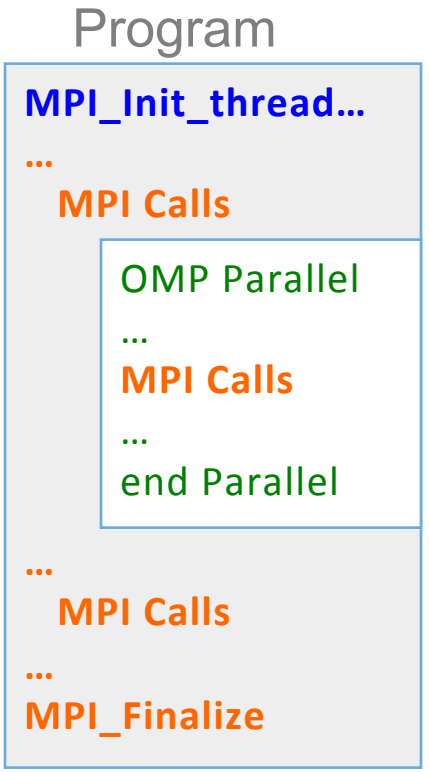
MPI Calls in Serial regions

- Serial regions MPI calls use the “master thread”.

OMP parallel regions

- MPI Calls within parallel region
  In parallel region MPI rank is known to all threads

End with MPI_Finalize



Hybrid – Program Model
Thread Safe MPI

MPI_THREAD_SINGLE

Only one thread will execute.

MPI_THREAD_FUNNELED

The process may be multi-threaded, but only the main thread will make MPI calls (all MPI calls are funneled to the main thread).

MPI_THREAD_SERIALIZED

The process may be multi-threaded, and multiple threads may make MPI calls, but only one at a time: MPI calls are not made concurrently from two distinct threads (all MPI calls are serialized).

MPI_THREAD_MULTIPLE

Multiple threads may call MPI, with no restrictions.

~

Hybrid Coding – MPI with Master

Fortran

```
include 'mpif.h'
use omp_lib
program single_thread

call MPI_Init(ierr)

! MPI with Master thread (here)

!$OMP parallel do
   do i=1,n
     <work>
   enddo

! MPI with Master thread (or here)

call MPI_Finalize(ierr)
end
```

NO MPI HERE

C

```
#include <mpi.h>
#include <omp.h>
int main(){

ierr= MPI_Init(NULL, NULL);

//MPI with Master (here)

#pragma omp parallel for
   for(i=0; i<n; i++){
     <work>
   }

// MPI with Master (or here)

 ierr= MPI_Finalize();
}
```

NO MPI HERE

Hybrid Coding – MPI with threads

Fortran

```
include 'mpif.h'
use omp_lib
program multi_thread

call MPI_Init_thread(MPI_THREAD_MULTIPLE,  &
                     iprovided,ierr)

!  MPI  with Master thread

!$OMP parallel

  !$OMP barrier      !may be necessary

   call MPI_<Whatever>(…,ierr)  ! {thread(s)}

!$OMP end parallel

!  MPI  with Master thread

end
```

C

```
#include <mpi.h>
#include <omp.h>
int main(){

MPI_Init_thread(…,MPI_THREAD_MULTIPLE,  \
                &iprovided)

//MPI with Master

 #pragma omp parallel
 {
  #pragma omp barrier   //maybe

    ierr=MPI_<Whatever>(…)  // {thead(s)}

 }

//MPI with Master

}
```

Hybrid Computing (setup & run)

- Use MPI compiler and openmp option.
- Set # of nodes and tasks per node.
- Set number of threads
- Set MPI and OpenMP Affinity (defaults often good)

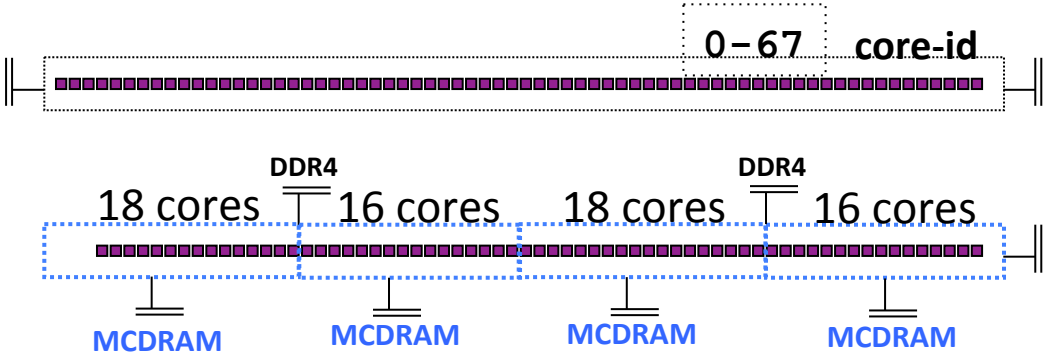
```
mpif90 -qopenmp -xMIC-AVX512 p.f90 -o hybrid_a.out

idev -N 2 -tpn 4            #SBATCH -N 2 -tasks-per-node 4

export OMP_NUM_THREADS=17 OMP_PROC_BIND=spread
export I_MPI_PIN=1 I_MPI_PIN_DOMAIN=auto:compact
ibrun  hybrid_a.out         #4 tasks/node 68 threads/node
```

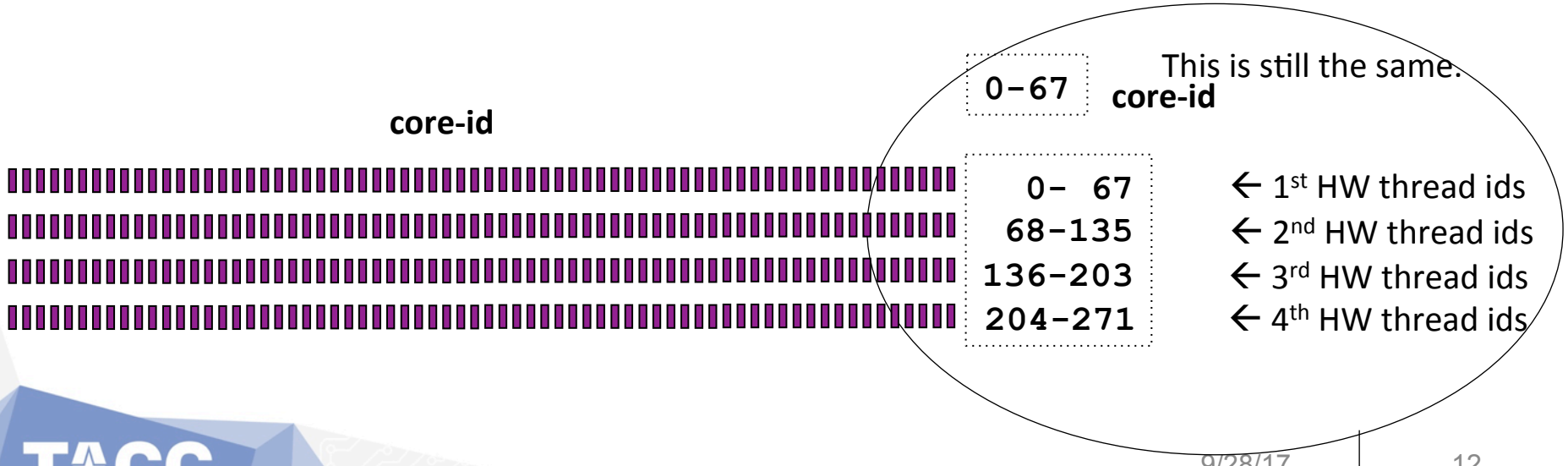
Memory Nodes on KNL

1 socket
68 cores
4 HW threads/core

4 MCDRAM NUMA "nodes"

Now it's complicated!
Would expect 17, 17, 17, 17.
cores per numa node.
Cannot split across tiles→
Some sort of 18, 16 combination.





Hybrid Computing (proc/thrd placement)

Where does the system put MPI Processes and OMP Threads?

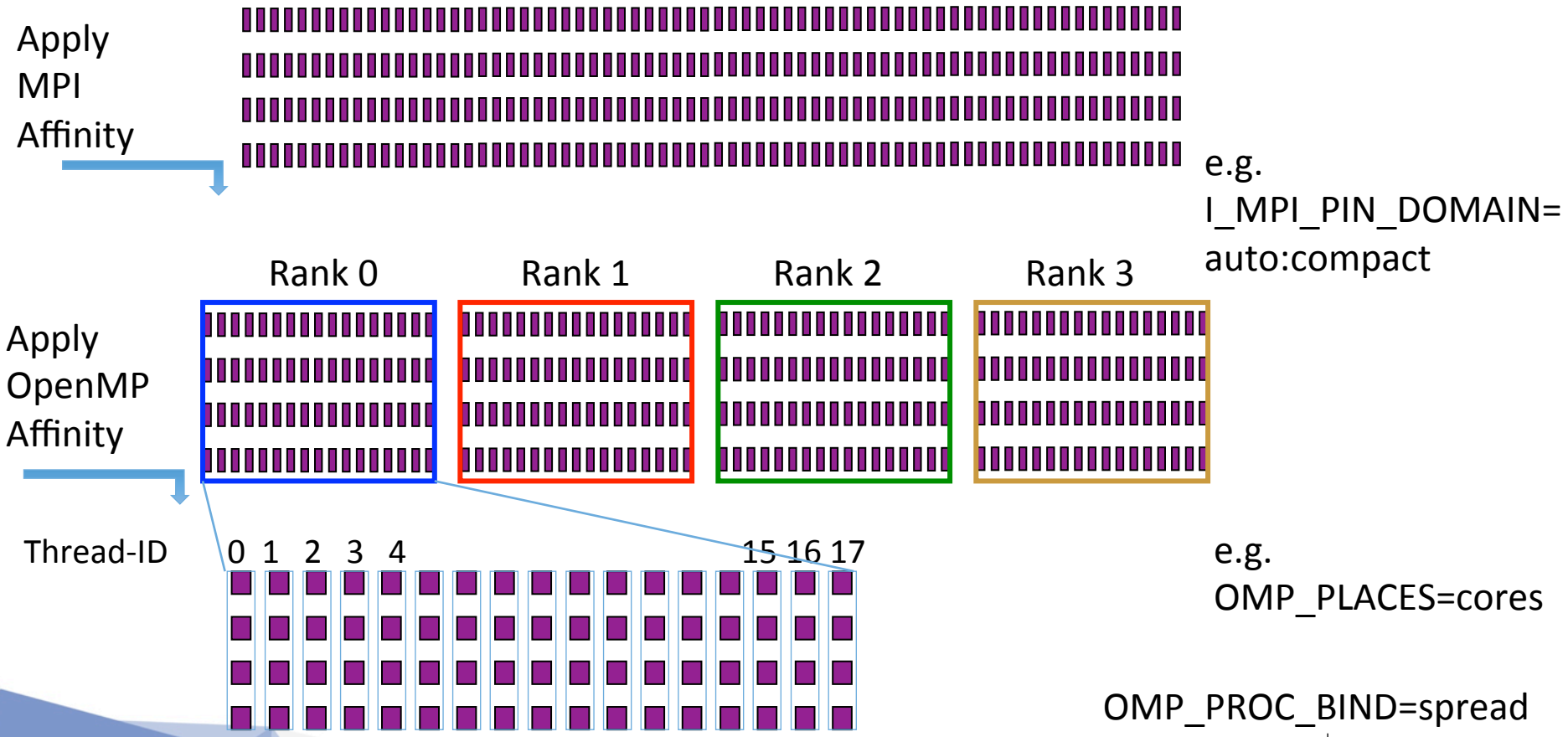
1. An MPI mask for each MPI process is created
2. Using the MPI mask, at the beginning of each parallel region the thread runtime creates a new mask for each thread of an MPI rank.

- Placement for common cases works fine.
  Use Affinity setting for special cases.

Hybrid Computing MPI MASK

Hybrid: 4 tasks, 17 threads

MPI: Create 4 masks: masking off separate 17-core sections in the kernel mask, one for each task.



Hybrid Computing

Suggested MPI process and thread count combinations

MPI Rank Count	Use Case	Thread Count – OMP_NUM_THREADS (for 1-4 threads/core)
1	1 communicator	(68, 136, 204,272)
2	numa DDR	(34, 68, 102, 136)
4	numa MCDRAM	(17, 34, 51, 68)
8		(8,16,24,32) (not using all cores, 68/8 != integer)
16		(4,8,12,16) (not using all cores, 68/8 != integer)
17		(4, 8, 12, 16)
34	1 proc/tile	(2, 4, 6, 8)
68	1 proc/core	(1, 2, 3, 4)

TASKs Per Node

OMP_NUM_THREADS

Be careful, for 8 and 16 MPI processes– USE OMP_PROC_BIND=spread (not KMP_KMP_AFFINITY=balanced); for MPI ranks/threads = 8/25 and 16/12 some cores have 4 threads and others only 2.

How do you set up the MPI mask?

- This varies for each MPI package
  Intel, OS-HPC, Cray Microsoft, OSU, …
  IMPI, Open MPI, Cray MPI, MS-MPI, OSU MVAPICH2,...

- IMPI
  - Set up DOMAINs:
    <name> = logical computing units related to Hardware
    <n> = number of domains [sequential sets of logical Units]
  - Explicitly specify:
    a set of hardware entities

Domains

export I_MPI_PIN_DOMAIN=<shape|name>: layout

Shape	Domains
core	68 cores (SMT0-SMT3 physical packing)
cache1	68 cores (SMT0-SMT3 cache associate)
cache2	34 tiles (2 cores paired to cache, 6 SMTs)
numa	4 numa nodes (cores associated with MCDRAM Controller, only with SNC-4 mode)
node	1 node: all 272 SMTs

Domains

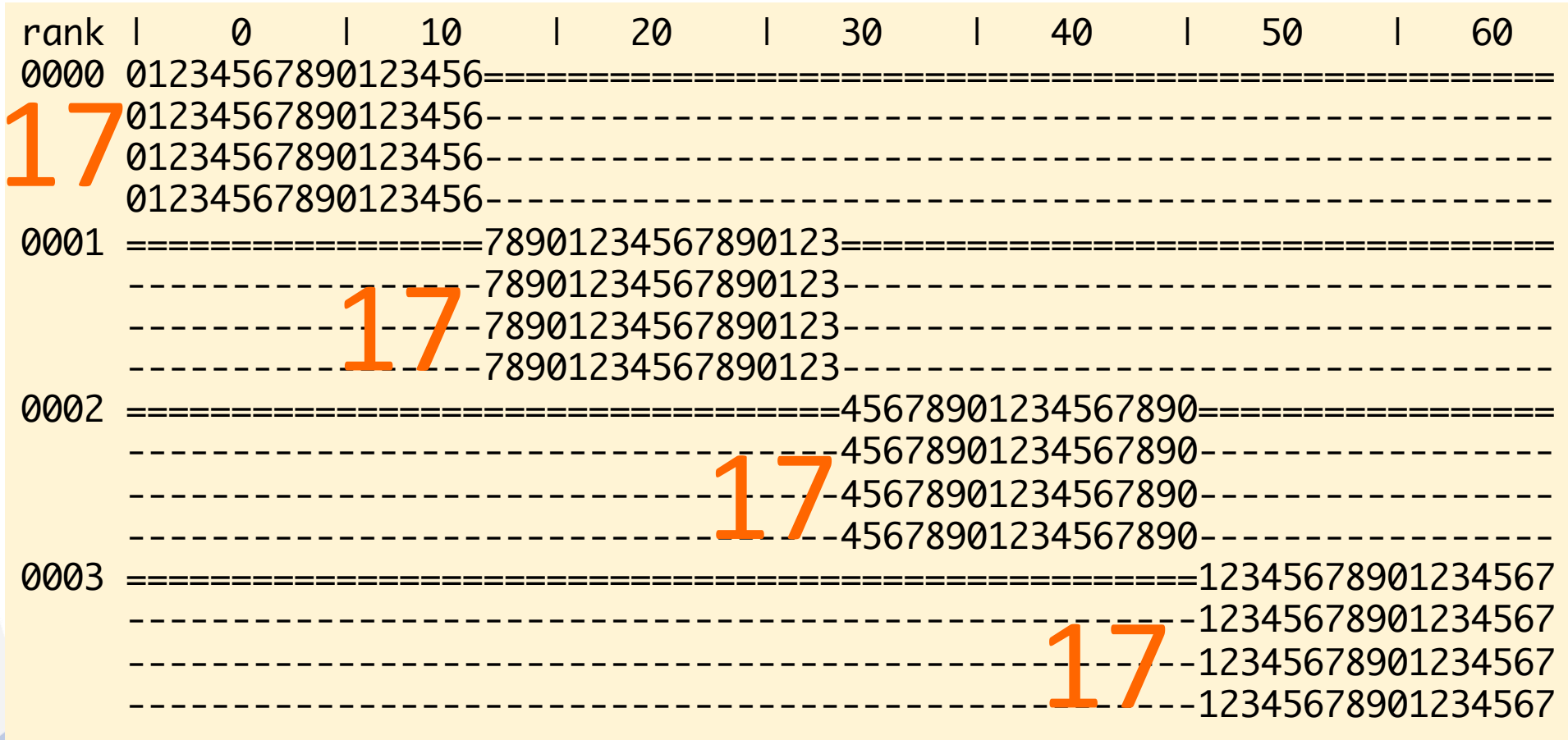
mpirun –np<#>	defaults	DOMAIN=<#>	DOMAIN=
68	68 cores		core, cache1
34	34 tiles		cache2
17	17 blocks of 4 cores		
8	8 blocks of 8.5 cores	34 (34/4)	
4	4 blocks of 17 cores	68 (68/4)	
2	2 blocks of 34 cores	136 (136/4)	
1	1 block of all cores	272 (272/4)	node

Note the difference between even blocks and numa blocks!

4	None 4 blks: 18,18,16,16 cores		numa (on SNC-4)

Viewing Affinity mask

```
$ module load amask
$ ibrun -np 4 amask_mpi  # MPI mask
```



Viewing Affinity mask

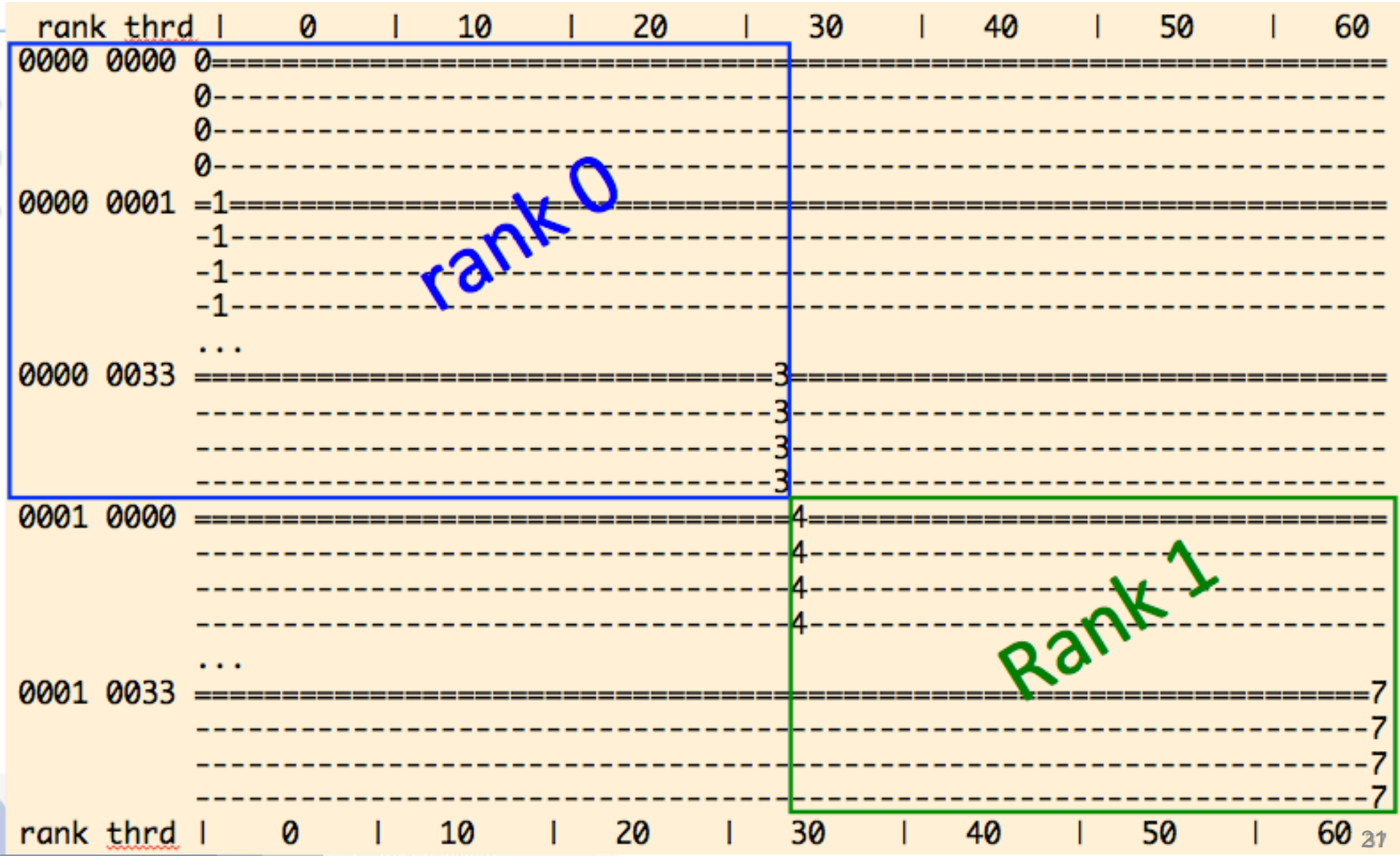
```
$ ibrun -np 2 amask_mpi
```

```
Each row of matrix is a mask for a SMT-id -- 0, 1, 2, or 3.
    CORE ID  = matrix digit + column group # in |...|
    A set mask bit (proc-id) = core id + add 68 to each additional row.

rank |    0    |   10    |   20    |   30    |   40    |   50    |   60
0000 01234567890123456789012345678901234=====================
     01234567890123456789012345678901234-
     01234567890123456789012345678901234-
     01234567890123456789012345678901234-
0001 =====================56789012345678901234567890123456789
     ---------------------56789012345678901234567890123456789
     ---------------------56789012345678901234567890123456789
     ---------------------56789012345678901234567890123456789
```

MPI process Rank 0 allowed anyplace on 1st 34 cores.

MPI process Rank 1 allowed anyplace on 2nd set of 34 cores.

```
$ export OMP_NUM_THREADS=34 OMP_PLACES=cores
$ ibrun -np 2 amask_hybrid
```



Summary – Hybrid Computing

Determine best ratios of tasks/node and threads/task

Make sure MPI process mask can accommodate OMP thread count

Set Appropriate MPI Affinity (subset of proc-ids for each task)

Set OpenMP/KMP (thread) Affinity (operates on MPI subset of proc-id)

Use MPI launcher: `ibrun`/mpirun/`mpiexec.hydra a.out*`

Use utils to observe mask: `I_MPI_DEBUG, AMASK`

*Use a script instead of a.out to specify numactl and explicit affinity lists (settings) for each rank.

Use numactl –H to see NUMA node assignments or lscpu.

Summary example

Set the number of MPI Processes & Nodes (e.g. 8 total ranks, 2 nodes):

in batch: `#SBATCH -N 2 --tasks-per-node 4`

idev interactive: `idev -N 2 -tpn 4`

Determine an appropriate number of Threads PER MPI Process

(usually 1, 2, 3 or 4 threads per core)

`export OMP_NUM_THREADS=17` or {34, 51, 68}

Set OpenMP distribution (or KMP, but not both):

`export OMP_PROC_BIND=spread`

Launch with `ibrun` (or `mpiexec.hydra` for special control*)

Use amask utils: ml amask; ibrun amask_hybrid to see Affinity

*Use a script instead of a.out to specify numactl and affinity settings for each rank. Use numactl –H to see NUMA node assignments, or lscpu.