TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

TEXAS

The University of Texas at Austin

# Hybrid Computing

PCSE 2018

4/10/18

**PRESENTED BY:**

Lars Koesterke

Charlie Dey

# MPI & OpenMP == Hybrid Programs

MPI and OpenMP in a single code

MPI process acts as a container for OpenMP threads.*

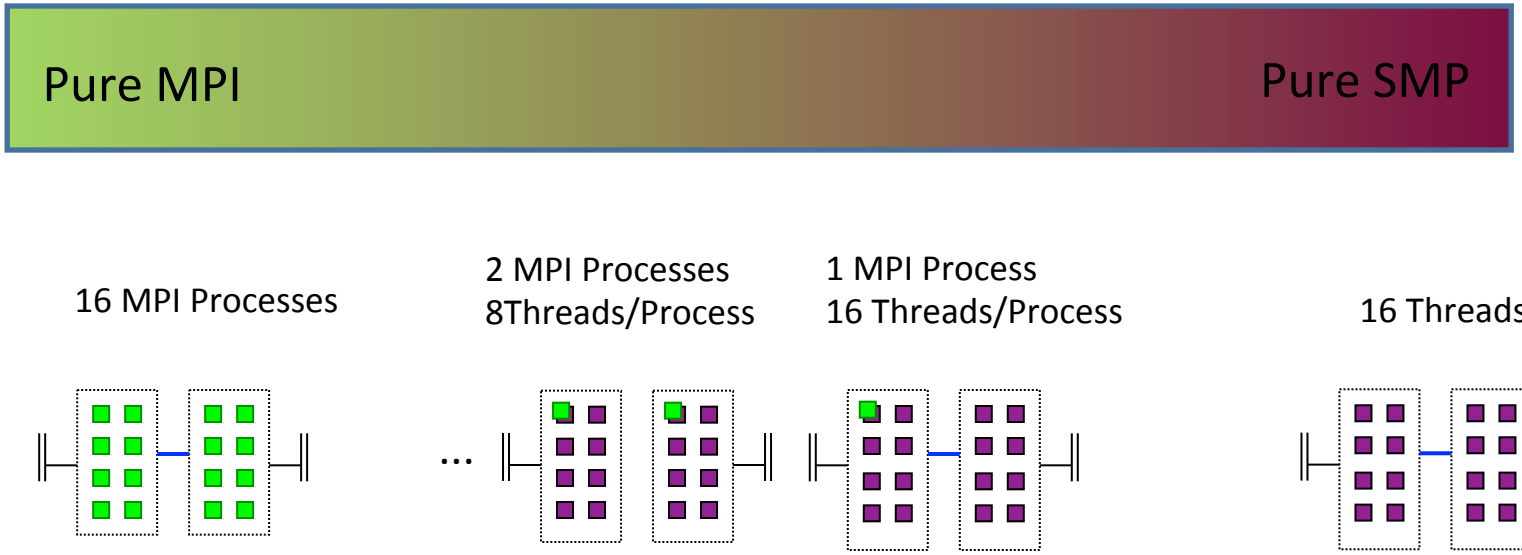
- MPI processes for communication between nodes.
- OpenMP threads can access all memory within MPI process on node.
- Use MPI parallelism across Nodes and OpenMP within a Nodes.

- But, you may see multiple MPI processes on a node!
- Threads with MPI process only see memory of its MPI process

*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 Stampede Sandy Bridge node:

- 2 sockets
- 8 cores per socket

Stampede2 differs

- 2 sockets
- 16 cores per socket
- Hyperthreading enabled

Master Thread of MPI Task
MPI Process on Core (a.out)
Master Process-Thread of MPI Task
Spawned Thread of MPI Task
Memory of socket
Socket + Core

# MPI – Program Model
## **Usually** a **Container for Threading**

Start with MPI_Init

MPI Calls

End with MPI_Finalize

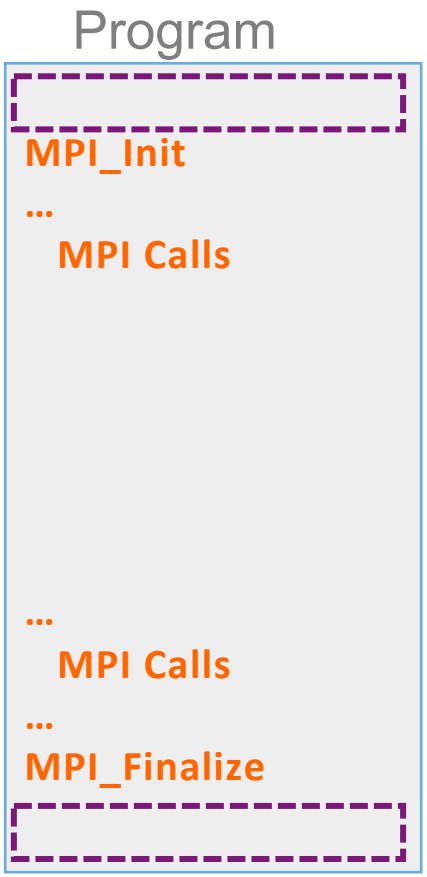

OpenMP Parallel regions most often appear between MPI Initialize and Finalize.

# MPI – Program Model
## **<u>Unusual</u> 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
# 1) 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

Program

```
MPI_Init
…
  MPI Calls
      OMP Parallel
      …

      …
      end Parallel
…
  MPI Calls
…
MPI_Finalize
```

# Hybrid – Program Model
# 2) 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

Program

```
MPI_Init_thread…
…
  MPI Calls
      OMP Parallel
      …
      MPI Calls
      …
      end Parallel
…
  MPI Calls
…
MPI_Finalize
```

Thread Safe MPI comes in 2 flavors
- With ‘master’ (2a)
- General (2b)

# Hybrid Coding – **2a) MPI with Master**

Fortran

```
use  mpi     ! or include 'mpif.h'
use omp_lib
program single_thread

call MPI_Init(ierr)

!$OMP parallel
!$OMP master
call MPI_Whatever()
!$OMP end master

!$OMP 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);

#pragma omp parallel {
#pragma omp master {
MPI_Whatever()}

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

// MPI with Master (or here)

ierr= MPI_Finalize();
}
```

NO MPI HERE

# Hybrid Coding – **2b) MPI in parallel region**

Fortran

```
use  mpi     ! or 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

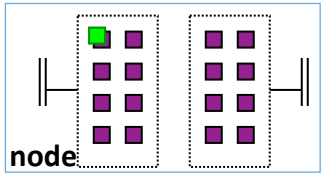
```
#include <mpi.h>
#include <omp.h>
int main(){

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)

How to run a hybrid code, example:

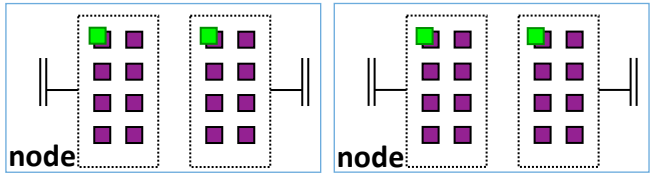
```
-N Nodes and -n Tot_tasks  are SLURM OPTIONS
          tasks_per_node  = Tot_tasks/Nodes
export OMP_NUM_THREADS=Nthreads
       OMP_NUM_THREADS will be threads_per_mpitask
       usually have 1 or 2 (HT) threads per core
       OMP_NUM_THREADS=cores_per_node/(tasks_per_node)
```

**-N = 1**
**-n = 1**
**OMP_NUM_THREADS=16**




**-N = 1**
**-n = 2**
**OMP_NUM_THREADS=8**


**-N = 2** **-- Nodes**
**-n = 4** **-- total tasks**
**OMP_NUM_THREADS=8** **-- threads/task**

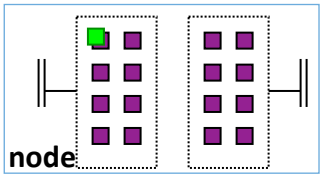



Master Thread of MPI Task
MPI Process on Core (a.out)
Master Process-Thread of MPI Task
Spawned Thread of MPI Task

# Hybrid Computing (setup & run)

How to run a hybrid code, example:

**-N = 1**
**-n = 1**
**OMP_NUM_THREADS=16**




```
#SBATCH -N 1
#SBATCH -n 1

export OMP_NUM_THREADS=16
ibrun ./a.out
```

```
idev -N 1 -n 1
...

export OMP_NUM_THREADS=16
ibrun ./a.out
```

Master Thread of MPI Task
MPI Process on Core (a.out)
Master Process-Thread of MPI Task
Spawned Thread of MPI Task

# Hybrid Computing (setup & run)

How to run a hybrid code, example:

**-N = 1**
**-n = 2**
**OMP_NUM_THREADS=8**


Need to make sure 1 MPI process ends up on each socket.

```
#SBATCH -N 1
#SBATCH -n 2

export OMP_NUM_THREADS=8
ibrun tacc_affinity ./a.out
```

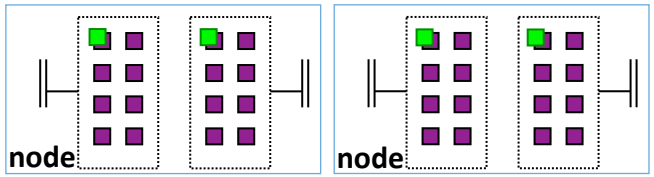
```
idev -N 1 -n 2
...

export OMP_NUM_THREADS=8
ibrun tacc_affinity ./a.out
```

Master Thread of MPI Task
MPI Process on Core (a.out)
Master Process-Thread of MPI Task
Spawned Thread of MPI Task

# Hybrid Computing (setup & run)

How to run a hybrid code, example:

**-N = 2** **-- Nodes**
**-n = 4** **-- total tasks**
**OMP_NUM_THREADS=8** **-- threads/task**




Does this work across nodes?
What if –n = 3?

```
#SBATCH -N 2
#SBATCH -n 4

export OMP_NUM_THREADS=8
ibrun tacc_affinity ./a.out
```

```
idev -N 2 -n 4
...

export OMP_NUM_THREADS=8
ibrun tacc_affinity ./a.out
```

Master Thread of MPI Task
MPI Process on Core (a.out)
Master Process-Thread of MPI Task
Spawned Thread of MPI Task

# Hybrid Computing (setup & run)

How are MPI tasks and thread positioned?
--That's what Affinity is all about. --

## How do you know where procs/threads will run:

Watch the load in `top`

`export I_MPI_DEBUG=4` ← will list assigned proc-ids for MPI procs.

`export OMP_DISPLAY_ENV=TRUE` ← limited OMP affinity info.

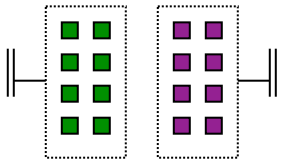
Use **maskeraid** (git clone https://github.com/tacc/maskeraid) utility to see assignment in parallel region:

```
e.g.  ibrun tacc_affinity ./hybrid_whereami
```

hybrid_whereami
mpi_whereami
omp_whereami

# Process IDs on Stampede

2 sockets
8 cores/socket
1 HW thread/core



| | | |
|---|---|---|
| **proc-id** | 0-7 | 8-15 |
| **core-#** | 0-7 | 8-15 |

```
$ grep -E 'processor|physical id' /proc/cpuifo
processor      : 0 physical id  : 0
processor      : 1 physical id  : 0
processor      : 2 physical id  : 0
processor      : 3 physical id  : 0
processor      : 4 physical id  : 0
processor      : 5 physical id  : 0
processor      : 6 physical id  : 0
processor      : 7 physical id  : 0
processor      : 8 physical id  : 1
processor      : 9 physical id  : 1
processor      : 10 physical id: 1
processor      : 11 physical id: 1
processor      : 12 physical id: 1
processor      : 13 physical id: 1
processor      : 14 physical id: 1
processor      : 15 physical id: 1
```

```
c557-201.stampede(15)$ lscpu

Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                16
On-line CPU(s) list:   0-15
Thread(s) per core:    1
Core(s) per socket:    8
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 45
Model name:            Intel(R) Xeon(R)
CPU E5-2680 0 @ 2.70GHz
Stepping:              7
CPU MHz:               2701.000
BogoMIPS:              5399.22
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              20480K
NUMA node0 CPU(s):     0-7
NUMA node1 CPU(s):     8-15
```

# Hybrid Computing (MPI masks)

## Intel MPI (IMPI)

```
idev -n 2 -N 1
export I_MPI_DEBUG=4
ibrun  my_mpi_a.out
```

[0] MPI rank: 0  c567-001
{8,9,10,11,12,13,14,15}
[0] MPI rank: 1  c567-001
{0,1,2,3,4,5,6,7}

Rank 0 can execute on any of these proc-ids.

Rank 1 can execute on any of these proc-ids.

```
idev -n 2 -N 1
ibrun  mpi_whereami
```

Each row of matrix is an Affinity mask.
A set mask bit = matrix digit + column # in |...|

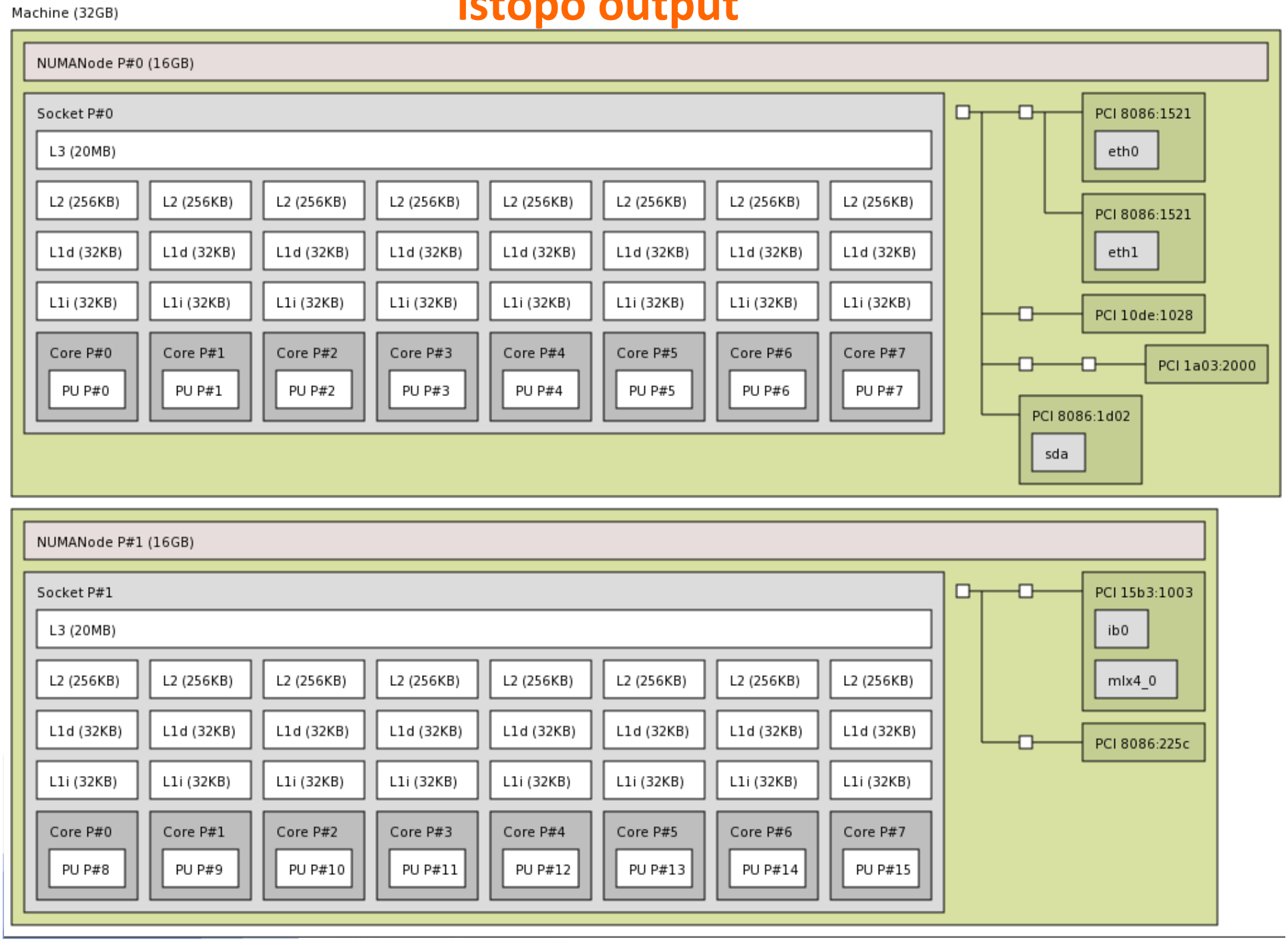
```
rank |          |    10     |
0000 --------89012345
0001 01234567--------
```

```
idev -n 2 -N 1
ibrun  tacc_affinity \
       mpi_whereami
```

Each row of matrix is an Affinity mask.
A set mask bit = matrix digit + column # in |...|

```
rank |          |    10     |
0000 01234567--------
0001 --------89012345
```

# lstopo output



# Hybrid Computing (MPI masks)

**MVAPICH2 MPI (default @ TACC)**

```
idev -n 2 -N 1
# no tools for affinity
ibrun  my_mpi_a.out
```

```
idev -n 2 -N 1
ibrun  mpi_whereami
```

Each row of matrix is an Affinity mask.
A set mask bit = matrix digit + column # in |...|

```
rank |          |    10     |
0000 0---------------
0001 -1--------------
```

Problem:
In hybrid runs
All threads on
proc-id 0 & 1

```
idev -n 2 -N 1
ibrun  tacc_affinity \
       mpi_whereami
```

Each row of matrix is an Affinity mask.
A set mask bit = matrix digit + column # in |...|

```
rank |          |    10     |
0000 01234567--------
0001 --------89012345
```

# Hybrid Computing (thread masks)

```
idev -n 2 -N 1
export OMP_NUM_THREADS=8
ibrun  tacc_affinity hybrid_whereami
```

Each row of matrix is an Affinity mask.
A set mask bit = matrix digit + column # in |...|

```
rank thrd |          |   10     |
0000 0000 01234567--------
0000 0001 01234567--------
0000 0002 01234567--------
0000 0003 01234567--------
0000 0004 01234567--------
0000 0005 01234567--------
0000 0006 01234567--------
0000 0007 01234567--------
0001 0000 --------89012345
0001 0001 --------89012345
0001 0002 --------89012345
0001 0003 --------89012345
0001 0004 --------89012345
0001 0005 --------89012345
0001 0006 --------89012345
0001 0007 --------89012345
```