TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

Hybrid MPI/OpenMP Programming

IHPCSS
July 7-12, 2019

PRESENTED BY:
Kent Milfeld
milfeld@tacc.utexas.edu

Lecture and Lab slides available at: tinyurl.com/tacc-2019-ihpcss

What is Hybrid Computing?

Large HPC systems = Many Compute Nodes
Distributed-memory across nodes
Shared-memory on nodes

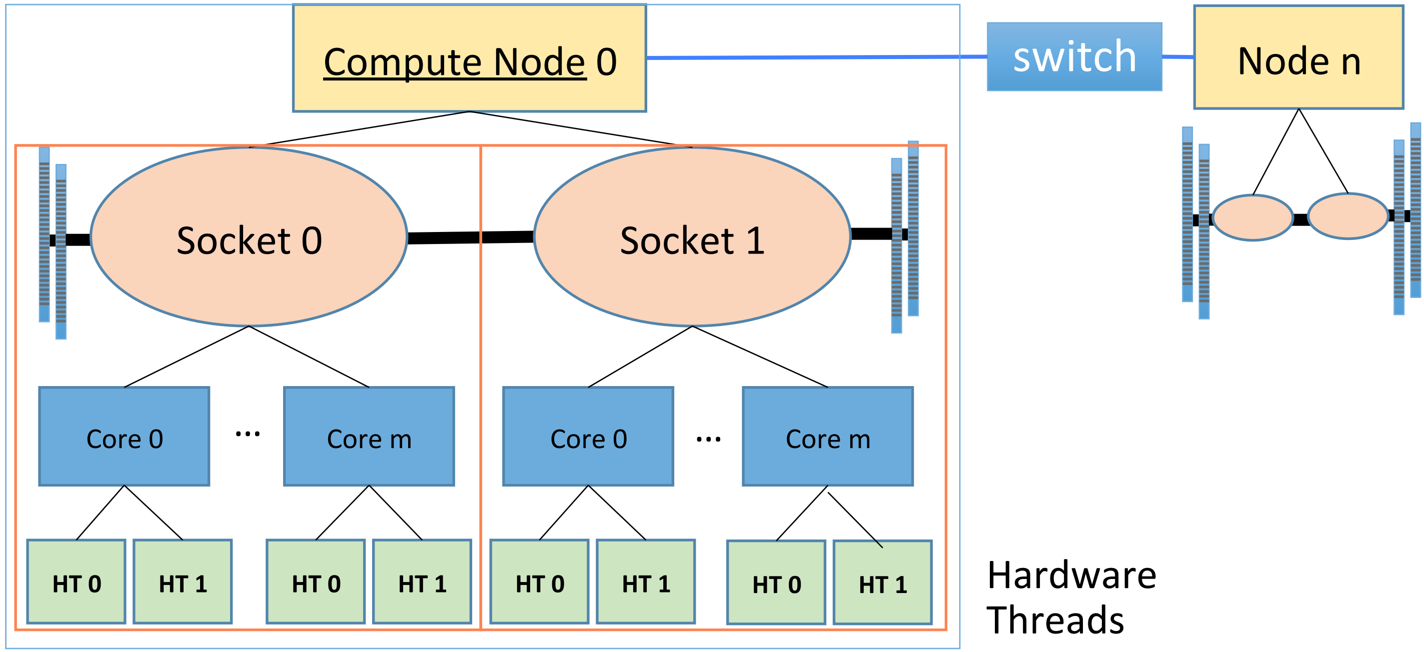
Hybrid: MPI for Process-Distributed +
OpenMP for Thread-Shared

Hybrid MPI+X programming

Parallelism	Process	Thread
Memory type	Distributed	Shared
Method	MPI	OpenMP, pthreads, etc.

Others	Process-Shared: PGAS, SHMEM

Hierarchical system layout



In a compute node, cores that are “local” to memory form a NUMA node.
Use command “`numactl -H`” to show available numa nodes

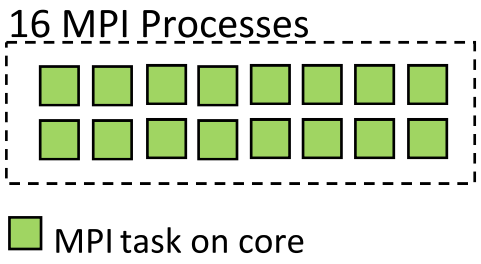
Pure MPI/OpenMP vs Hybrid

System: 16 cores/node

Which approach is the best for your application?

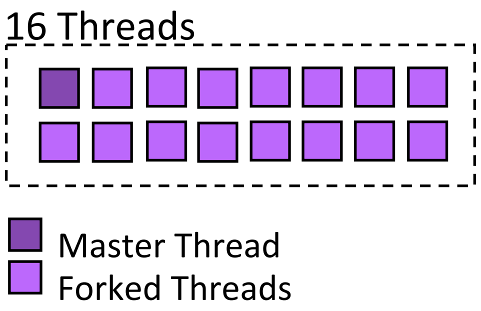
Pure MPI:

- High scalability and portability.
- Scalability beyond 1 node.
- Hard to ensure load balancing.

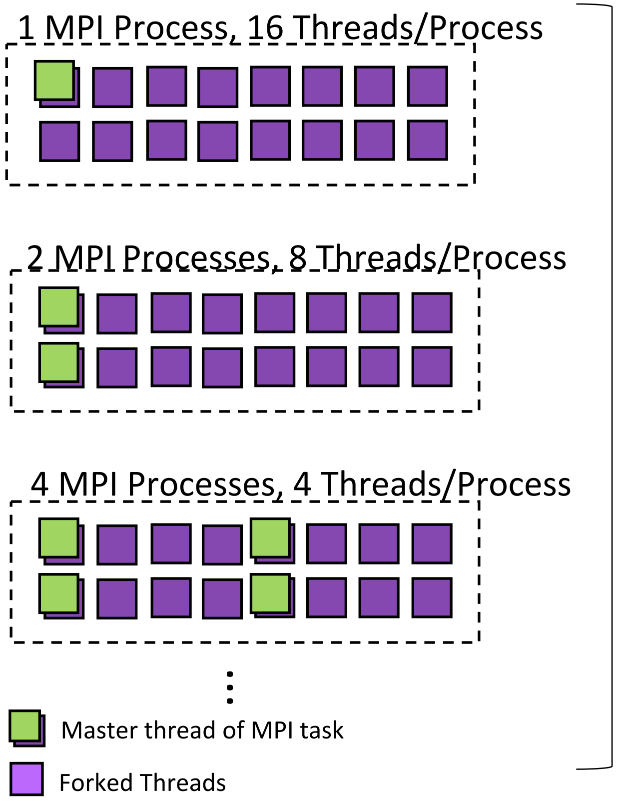


Pure OpenMP:

- Easy to write.
- Lower scalability. Low latency.
- Dynamic load balancing.
- Fine granularity.
- Limited to 1 node.



Hybrid MPI + OpenMP



Why MPI+OpenMP?

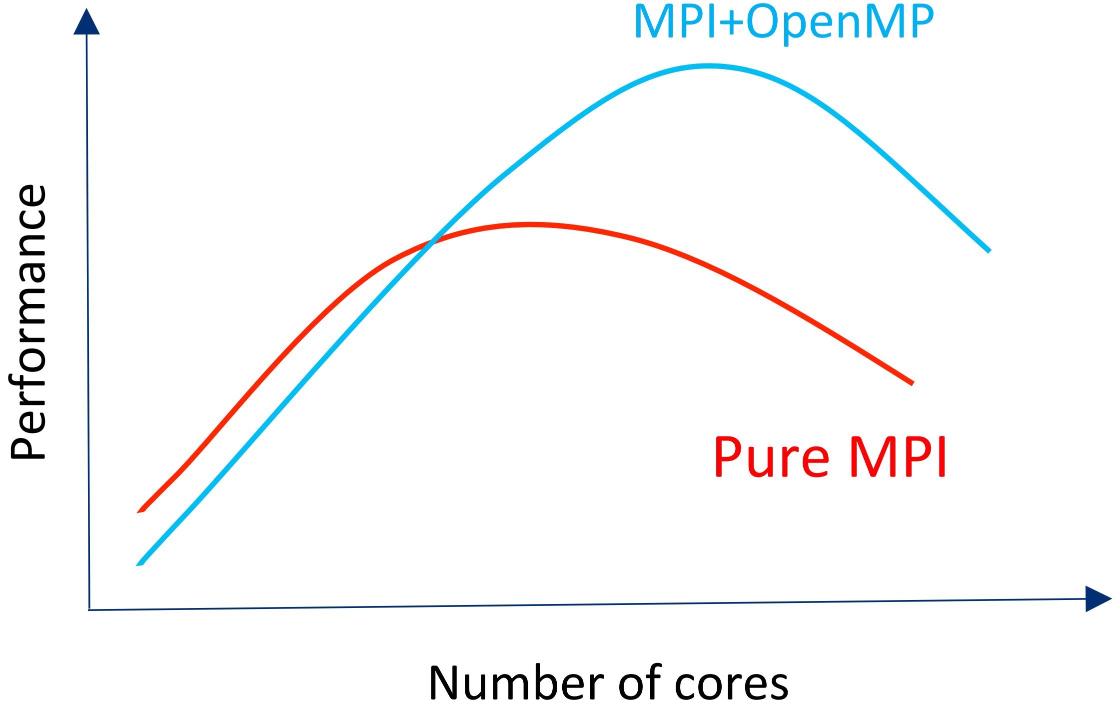
In a well balanced MPI application all cores are busy all the time, so using threading can give no immediate improvement.

There are two main motivations for using MPI+OpenMP

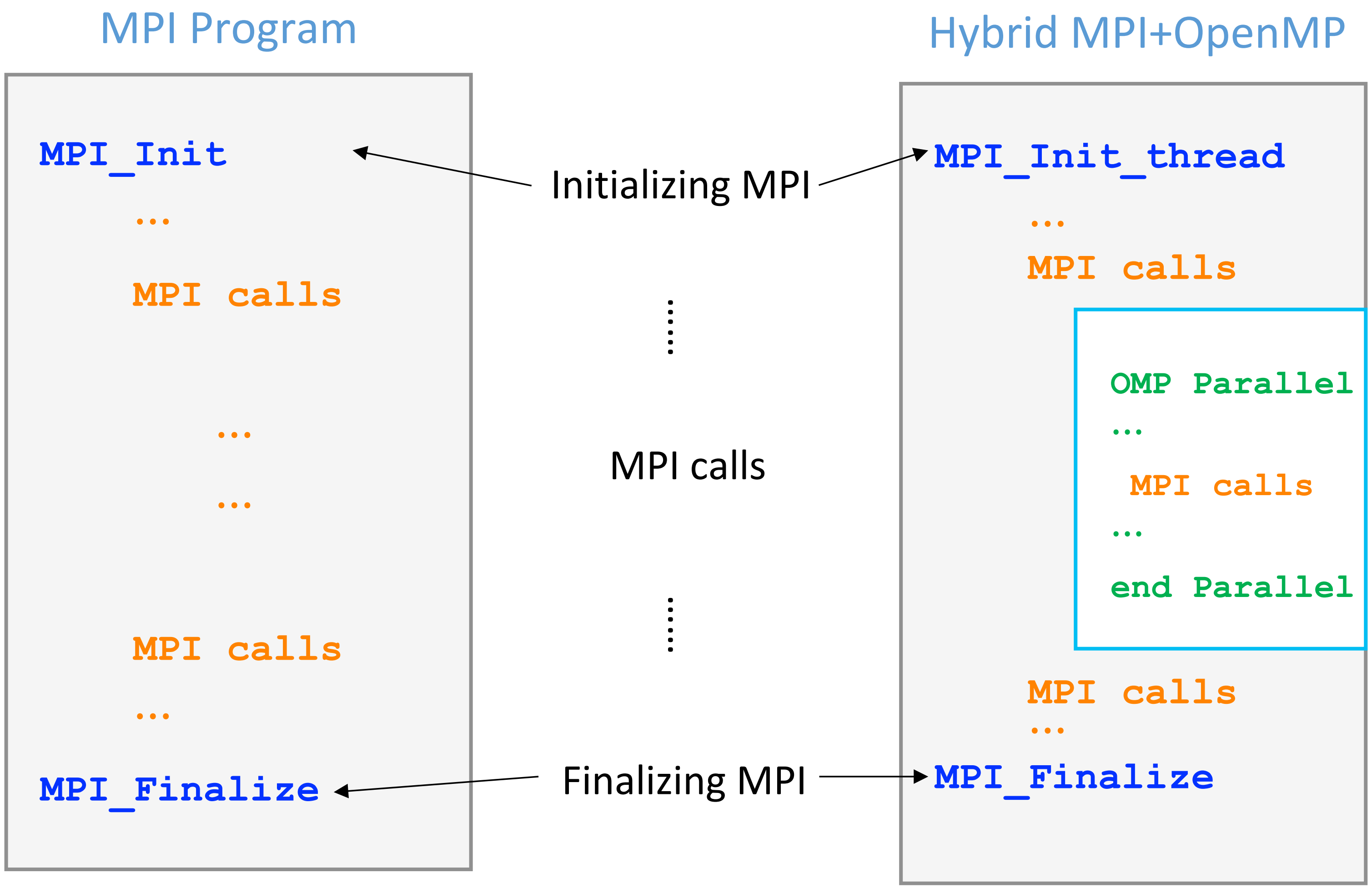
1. Reducing memory footprint
   (Replicated data, buffers in MPI codes)

2. Improving performance when the pure MPI scalability is running out
   (In places where load balancing in MPI is difficult or when MPI process count is too large for MPI to handle adequately)

Typical performance curves



Program Structure MPI processes act as "containers" for threads



Use `MPI_Init_thread` instead of `MPI_Init`

C/C++

```
int required, provided;
...
MPI_Init_thread(NULL,NULL, required, &provided);
```

Fortran

```
integer :: required, provided, ierr
...
MPI_Init_thread(          required, provided, ierr)
```

Level of Support (required)

`MPI_THREAD_SINGLE < MPI_THREAD_FUNNELED < MPI_THREAD_SERIALIZED < MPI_THREAD_MULTIPLE`

If MPI cannot support a requested level, it returns the highest level it can provide

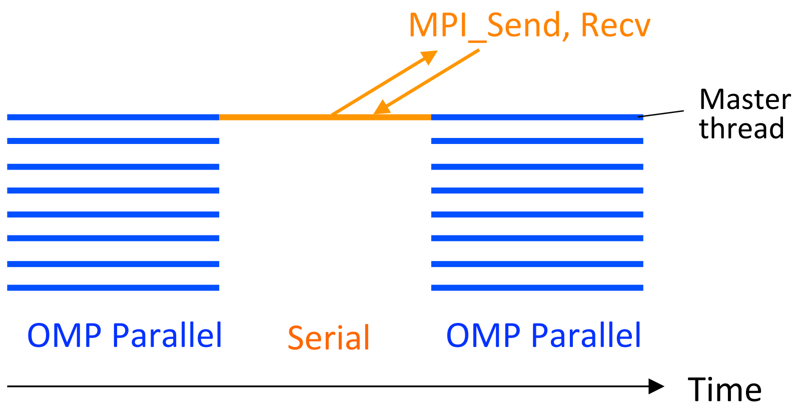
Check

```
MPI_Init_thread(NULL,NULL, MPI_THREAD_FUNNELED, &provided)
if (provided < MPI_THREAD_FUNNELED) MPI_Abort(MPI_COMM_WORLD, 1);
```

Hybrid Programming Styles

Model	Description	Required Support Level
Master-only	All MPI communication takes place in the sequential part of the OpenMP program	`MPI_THREAD_FUNNELED`
Funneled	MPI communication takes place through the same master thread	`MPI_THREAD_FUNNELED`
Serialized	MPI calls can be made by any thread, but only one at a time.	`MPI_THREAD_SERIALIZED`
Multiple	Multiple threads can make MPI calls simultaneously	`MPI_THREAD_MULTIPLE`

Hybrid Style : Master-Only



Fortran

```
!$OMP parallel
       work...
!$OMP end parallel

call MPI_send(...)

!$OMP parallel
       work...
!$OMP end parallel
```

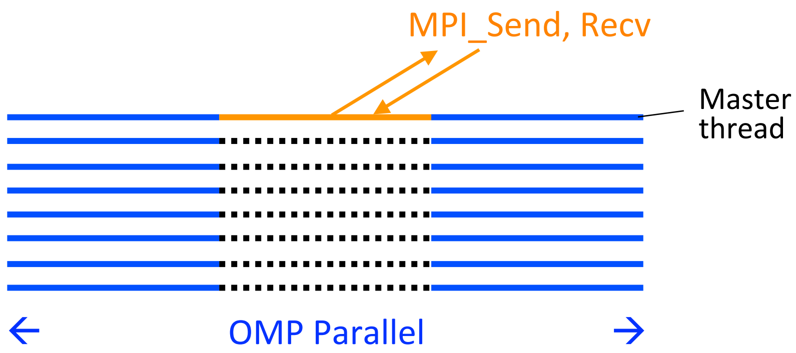
C

```
#pragma omp parallel
{
       work...
}

ierror=MPI_send(...);

#pragma omp parallel
{
       work...
}
```

- **Simple to write and maintain**
- Synchronized before/after MPI call

- Other threads are idle during MPI call
- **Data locality is bad – data must go through the cache where master thread is executing(“funneled”)**
- OpenMP parallel construct overhead can be comparable to MPI message latency (μs)

Hybrid Style : Funneled



Fortran

```
!$OMP parallel
 work...

   !$OMP barrier
   !$OMP master
     call MPI_send(...)
   !$OMP end master
   !$OMP barrier

 work...
!$OMP end parallel
```

C

```
#pragma omp parallel
{
 work...

  #pragma omp barrier
  #pragma omp master
  {
    ierror=MPI_send(...);
  }
  #pragma omp barrier
 work...
}
```

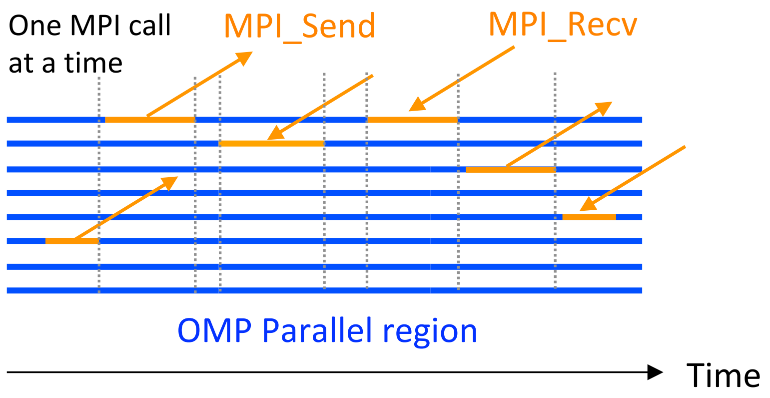
🙂

- **Relatively simple to write and maintain**
- **Cost of thread synchronization before/after MPI calls is less expensive**

🙁

- **Data locality is still bad**
- Might need load balancing between the master thread and the other threads

Hybrid Style : Serialized



Fortran

```
!$OMP parallel

   work…

   !$OMP critical
     call MPI_Send(...)
   !$OMP end critical

   work...
!$OMP end parallel
```

C

```
#pragma omp parallel
{
   work…

   #pragma omp critical
   {
     ierror=MPI_Send(...);
   }

   work...
}
```

🙂

- **Improved data locality: threads send own data, not funneled through Master**
- Works well with imbalanced work.

🙁

- May be harder to write and maintain
- Blocking on entry to a critical region may result in idle threads
- **Use tags to distinguish messages from/ to different threads with same MPI rank**

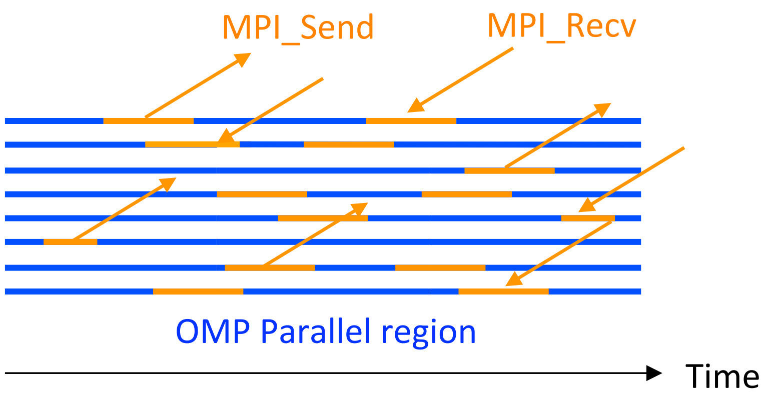
Hybrid Style : Multiple

Fortran

```
!$OMP parallel
  work…
    call MPI_Send(...)
  work…
!$OMP end parallel
```

C

```
#pragma omp parallel
{
  work…
    ierror=MPI_Send(...);
  work…
}
```



- **Good data locality**
- **Inter- and intra-node communication can be overlapped**
- **Data preparation can be done on multiple treads concurrently**
- Fewer concerns about synchronizing treads correctly

- Trickier to write and maintain
- MPI implementations work better or worse in this style.

Thread-Rank Communication

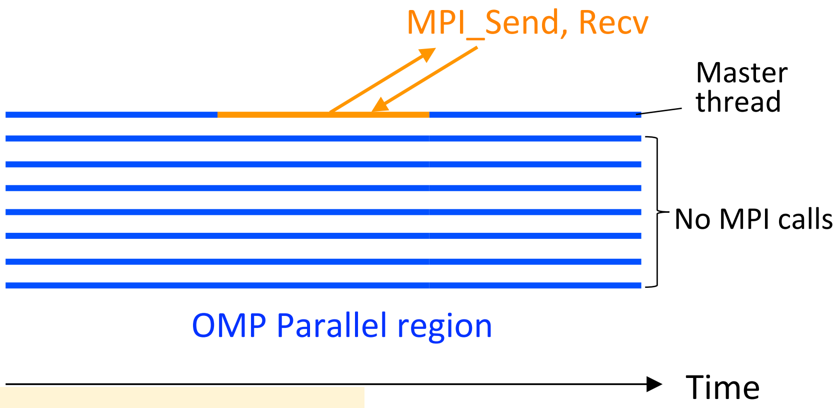
C/C++

```
int tid = omp_get_thread_num();

if (rank == 0) {
        MPI_Send(tid, MPI_INT, 1, tid, MPI_COMM_WORLD)
} else {
        MPI_Recv(buf, MPI_INT, 0, tid, MPI_COMM_WORLD, Status)
        printf("Rank%d-thread %d: Received from %d\n", rank, tid, buf)
}
```

Use thread ID as tag
to differentiate threads

Communicate
between rank 0 and 1

Overlapping Communication and Work



Fortran

```
!$OMP parallel private(tid)

  tid = omp_get_thread_num()

  if (tid.eq.0) then
    call MPI_<whatever>(......)
  else
    work
  endif
!$OMP end parallel
```

C/C++

```
#pragma omp parallel private(tid)
{
  tid = omp_get_thread_num();

  if (tid == 0) {
    MPI_<whatever>(......)
  } else {
    work
  }
}
```

Setup and Run

- Use MPI compiler and openmp option to compile the code

```
mpif90 -qopenmp p.f90 -o hybrid_a.out    [ Fortran ]
mpicc  -qopenmp p.c   -o hybrid_a.out    [ C ]
```

- Set number of nodes and tasks per node.

```
idev     -N 2 -tpn 4                [ idev ]
#SBATCH -N 2 -tasks-per-node 4     [ slurm ]
```

- Set number of threads

```
export OMP_NUM_THREADS=8
```

- Set MPI and OpenMP affinity (defaults often good)

```
export OMP_PROC_BIND=spread OMP_PLACES=cores
export I_MPI_PIN=1
export I_MPI_PIN_DOMAIN=auto:compact
```

- Run `ibrun hybrid_a.out # 2 nodes, 4 tasks/node, 8 threads/task`

Set Affinity

Without affinity threads can switch from one core to another, loosing data locality.

- Cache Thrashing may occur
- Local data may become REMOTE after the switch

It is difficult for the runtime libraries to optimize affinity settings, especially for hybrid codes. Set affinity! Some common case work well, though.

MPI Process / Thread 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 for its rank, at the beginning of each parallel region the thread runtime creates a new mask for each thread MPI rank.

Rank 0	1 1 1 1 0 0 0 0	Rank 1	0 0 0 0 1 1 1 1	MPI MASKS
0 thrd 0	1 0 0 0 0 0 0 0	1 thrd 0	0 0 0 0 1 0 0 0	
0 thrd 1	0 1 0 0 0 0 0 0	1 thrd 1	0 0 0 0 0 1 0 0	Thread MASKS
0 thrd 2	0 0 1 0 0 0 0 0	1 thrd 2	0 0 0 0 0 0 1 0	
0 thrd 3	0 0 0 1 0 0 0 0	1 thrd 3	0 0 0 0 0 0 0 1	

Intel MPI Affinity

Syntax: `I_MPI_PIN_DOMAIN=`*`size:layout`*

Default: `I_MPI_PIN_DOMAIN=auto:compact`

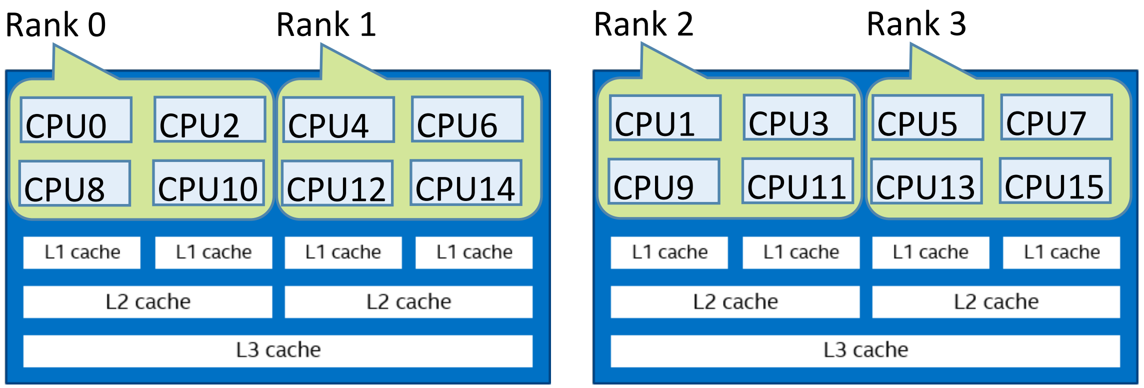
e.g.: `I_MPI_PIN_DOMAIN=4:compact`

auto : domain size = #logical processors/#tasks

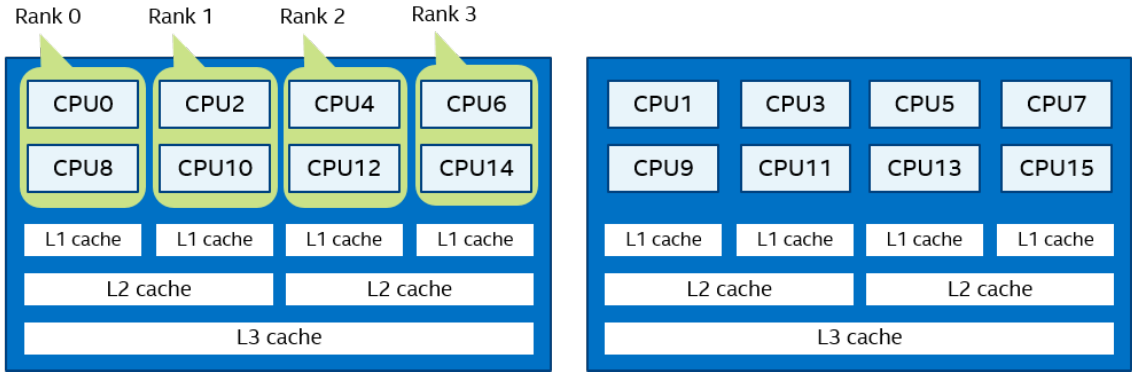
compact : domain members are ordered close

spread : domain members are spread out

mpirun -np 4 a.out. # size =16 logical procs/4 tasks



export I_MPI_PIN_DOMAIN=2:compact # size = 2 logical procs
mpirun -np 4 a.out



Ref: https://software.intel.com/en-us/mpi-developer-reference-linux-process-pinning
https://software.intel.com/en-us/mpi-developer-reference-linux-interoperability-with-openmp-api

Setting Up Thread Affinity

- Multiple ways of doing this from the environment
  - Intel KMP_* environmental variables [1]
  - OpenMP environmental variables
- OpenMP is more portable
- OpenMP uses two variables:
  - PROC_BIND policy : OMP_PROC_BIND=close | spread
  - PLACES list* | abstract set: OMP_PLACES=threads | cores | sockets

*Doesn't work for hybrid codes. Not able to assign different values for different MPI processes. Set OMP_PLACES=a<bstract_set> and/or OMP_PROC_BIN=<policy>

Ref1: https://software.intel.com/en-us/cpp-compiler-developer-guide-and-reference-thread-affinity-interface-linux-and-windows#KMP_AFFINITY_ENVIRONMENT_VARIABLE

Checking MPI Process Binding

- Use the following to get Intel MPI to report process bindings:

```
export I_MPI_DEBUG=4
```

- As part of your standard output you will see something like:

```
[0] MPI startup(): Rank    Pid      Node name       Pin cpu
[0] MPI startup(): 0       105228   test1         {0,1,2,3,4,5, … 31, … }
[0] MPI startup(): 1       105229   test1         {32,33,34,35, … 63, … }
```

Checking Thread Binding with Amask

- This is a tool developed by Kent Milfeld at TACC to look at large sets of processor bindings (like you see in KNL)
- Available at https://github.com/TACC/amask
- On TACC Systems:

```
$module load amask
```

- Execute one of the stand-alone executables, `amask_omp`, `amask_mpi`, or `amask_hybrid` in an OpenMP, MPI or hybrid environment, respectively, to obtain the expected affinity mask for each process/thread for your program in the same environment.
- Refer the user guide at `$TACC_AMASK_DOC`

Amask

The default affinity

```
export OMP_NUM_THREADS=4
ibrun -np 2 amask_hybrid
```

Rank=0 floating among 0-33 cores (not shown)

The 4 threads are on processors 0, 1, 2, 3,
i.e. On the cores 0, 1, 2, 3 (1st HW thread)

Rank=1 floating amoung 34-68 cores

The 4 threads are on processors 30+4, 5, 6, 7,
i.e. On the cores 34, 35, 36, 37 (1st HW thread)

Rank=0

Rank=1

```
rank thrd |    0    |   10    |   20    |   30    |   40    |   50    |   60    |
0000 0000 0===================================================================
          --------------------------------------------------------------------
          --------------------------------------------------------------------
          --------------------------------------------------------------------
0000 0001 =1==================================================================
          --------------------------------------------------------------------
          --------------------------------------------------------------------
          --------------------------------------------------------------------
0000 0002 ==2=================================================================
          --------------------------------------------------------------------
          --------------------------------------------------------------------
          --------------------------------------------------------------------
0000 0003 ===3================================================================
          --------------------------------------------------------------------
          --------------------------------------------------------------------
          --------------------------------------------------------------------
0001 0000 ==================================4=================================
          --------------------------------------------------------------------
          --------------------------------------------------------------------
          --------------------------------------------------------------------
0001 0001 ===================================5================================
          --------------------------------------------------------------------
          --------------------------------------------------------------------
          --------------------------------------------------------------------
0001 0002 ====================================6===============================
          --------------------------------------------------------------------
          --------------------------------------------------------------------
          --------------------------------------------------------------------
0001 0003 =====================================7==============================
          --------------------------------------------------------------------
          --------------------------------------------------------------------
          --------------------------------------------------------------------
```

default

Amask

Close

```
export OMP_PROC_BIND=close
export OMP_NUM_THREADS=4
ibrun -np 2 amask_hybrid
```

Rank=0 floating among 0-33 cores

The 4 threads are on processors
0, 0+68, 0+68*2, 0+68*3,
i.e. On the 4 hardware threads of core 0

Rank=1 floating amoung 34-68 cores

The 4 threads are on processors
34, 34+68, 34+68*2, 34+68*3,
i.e. On the 4 hardware threds of core 34

Rank=0

Rank=1

close

```
rank thrd |    0    |   10    |   20    |   30    |   40    |   50    |   60    |
0000 0000 0===================================================================
          --------------------------------------------------------------------
          --------------------------------------------------------------------
          --------------------------------------------------------------------
0000 0001 ====================================================================
          0-------------------------------------------------------------------
          --------------------------------------------------------------------
          --------------------------------------------------------------------
0000 0002 ====================================================================
          --------------------------------------------------------------------
          0-------------------------------------------------------------------
          --------------------------------------------------------------------
0000 0003 ====================================================================
          --------------------------------------------------------------------
          --------------------------------------------------------------------
          0-------------------------------------------------------------------
0001 0000 ==================================4=================================
          --------------------------------------------------------------------
          --------------------------------------------------------------------
          --------------------------------------------------------------------
0001 0001 ====================================================================
          ----------------------------------4---------------------------------
          --------------------------------------------------------------------
          --------------------------------------------------------------------
0001 0002 ====================================================================
          --------------------------------------------------------------------
          ----------------------------------4---------------------------------
          --------------------------------------------------------------------
0001 0003 ====================================================================
          --------------------------------------------------------------------
          --------------------------------------------------------------------
          ----------------------------------4---------------------------------
```

Amask

Spread

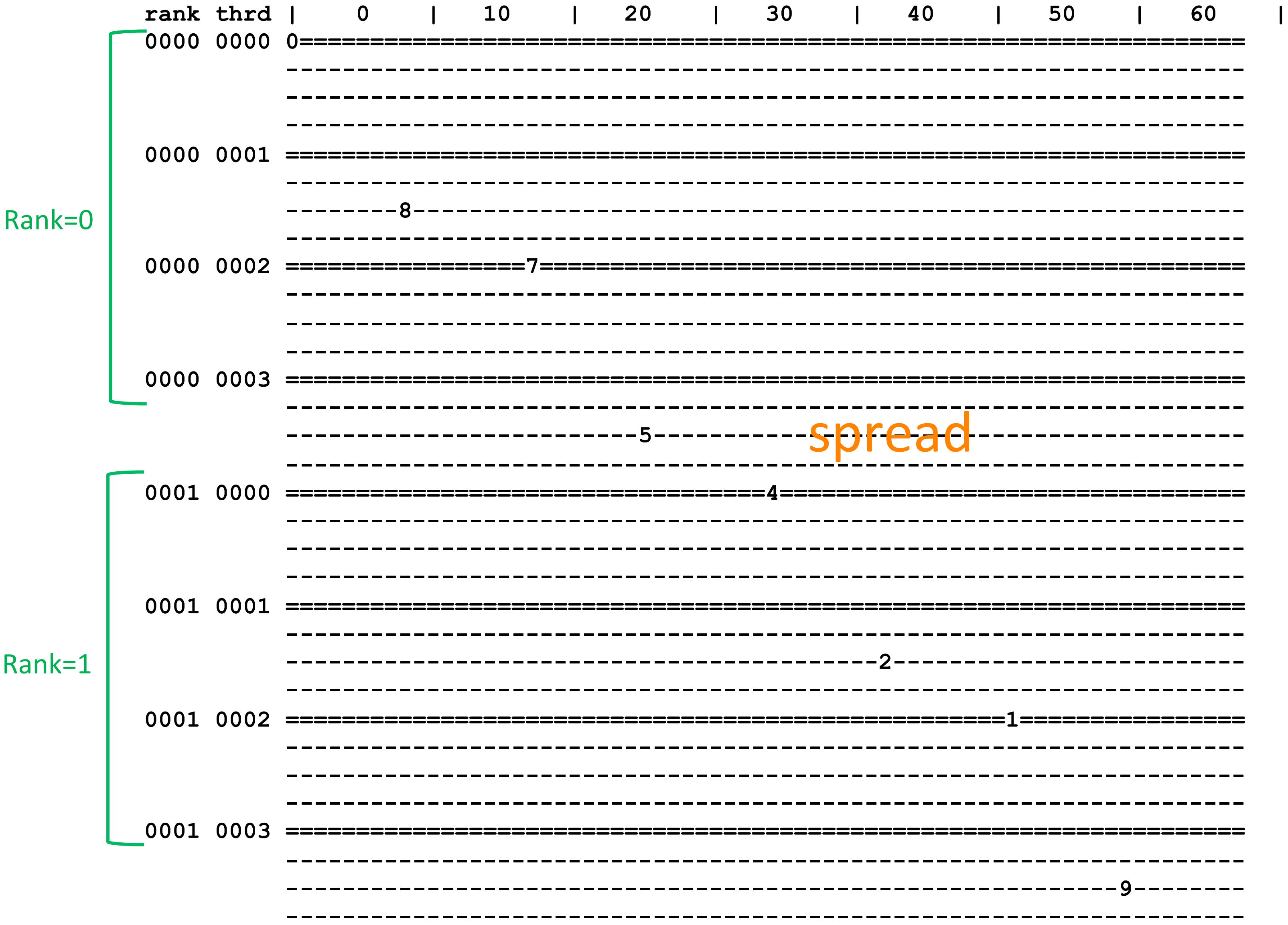
```
export OMP_PROC_BIND=spread
export OMP_NUM_THREADS=4
ibrun -np 2 amask_hybrid
```

Rank=0 floating among 0-33 cores

The 4 threads are on processors
0, 8+68*2, 10+7, 20+5+68*2,
i.e. On the cores 0,8,17,25

Rank=1 floating amoung 34-68 cores

The 4 threads are on processors
30+4, 40+2+68*2, 50+1, 50+9+68*2,
i.e. On the cores 34,42,51 59



Amask

Spread, cores

```
export OMP_PROC_BIND=spread
export OMP_PLACES=cores
export OMP_NUM_THREADS=4
ibrun -np 2 amask_hybrid
```

Rank=0 floating among 0-33 cores

The 4 threads are on processors
0, 8+68*2, 10+7, 20+5+68*2,
i.e. On the cores 0,8,17,25

Rank=1 floating amoung 34-68 cores

The 4 threads are on processors
30+4, 40+2+68*2, 50+1, 50+9+68*2,
i.e. On the cores 34,42,51 59

Rank=0

Rank=1

spread

```
rank thrd |    0    |   10    |   20    |   30    |   40    |   50    |   60    |
0000 0000 0===================================================================
          0-------------------------------------------------------------------
          0-------------------------------------------------------------------
          0-------------------------------------------------------------------
0000 0001 ========8===========================================================
          --------8-----------------------------------------------------------
          --------8-----------------------------------------------------------
          --------8-----------------------------------------------------------
0000 0002 =================7==================================================
          -----------------7--------------------------------------------------
          -----------------7--------------------------------------------------
          -----------------7--------------------------------------------------
0000 0003 =========================5==========================================
          -------------------------5------------------------------------------
          -------------------------5------------------------------------------
          -------------------------5------------------------------------------
0001 0000 ==================================4=================================
          ----------------------------------4---------------------------------
          ----------------------------------4---------------------------------
          ----------------------------------4---------------------------------
0001 0001 ==========================================2=========================
          ------------------------------------------3-------------------------
          ------------------------------------------2-------------------------
          ------------------------------------------3-------------------------
0001 0002 ===================================================1================
          ---------------------------------------------------1----------------
          ---------------------------------------------------1----------------
          ---------------------------------------------------1----------------
0001 0003 ===========================================================9========
          -----------------------------------------------------------9--------
          -----------------------------------------------------------9--------
          -----------------------------------------------------------9--------
```

Summary

- Use hybrid MPI+OpenMP when you want to (1) reduce memory footprint and/or (2) Improve performance when the pure MPI scalability is running out (3) fine tune imbalances
- Levels of MPI thread support: MPI_THREAD_SINGLE < MPI_THREAD_FUNNELED < MPI_THREAD_SERIALIZED < MPI_THREAD_MULTIPLE
- Four hybrid styles: master-only, funneled, serialized, multiple
- Process and thread affinity are important to the performance of hybrid execution on multi- and many-core architectures
- The general rule for setting the affinity is to populate the processors evenly with threads, to minimize the resource contention, and to maximize the hardware utilization.
- Default setting on Stampede2 is optimal for most cases. You should still fine tune for your applications if needed. Always check out what the default setting are doing with your application threads/processes.