Parallel Programming using MPI

Lecture 3

January 13, 2025

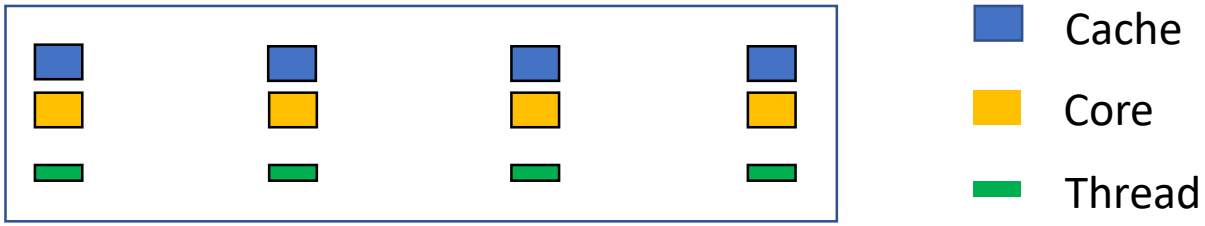
Parallel Programming Models

- Shared memory
- Distributed memory

Parallel Programming Models

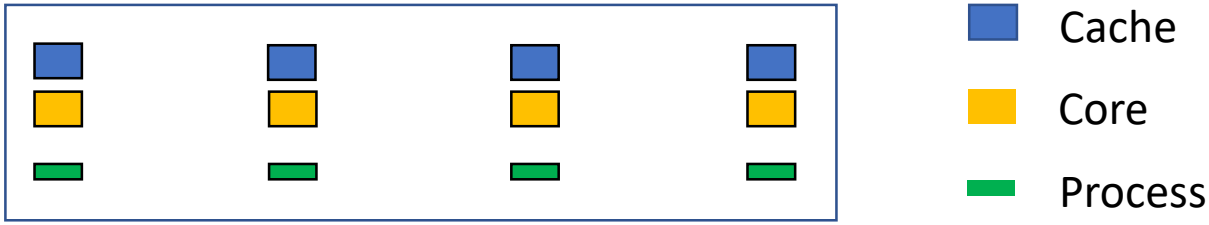
Shared memory programming – OpenMP

- Shared address space
- Implicit communication

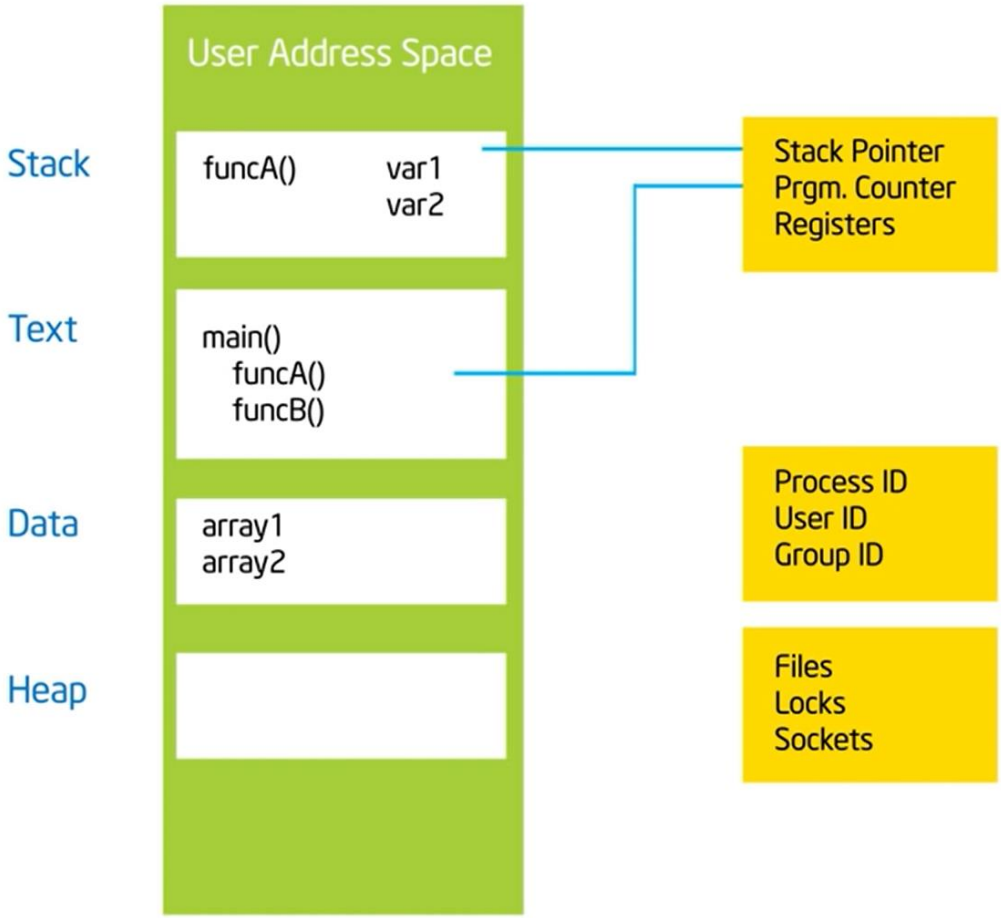


Distributed memory programming – MPI

- Distinct address space
- Explicit communication



Process

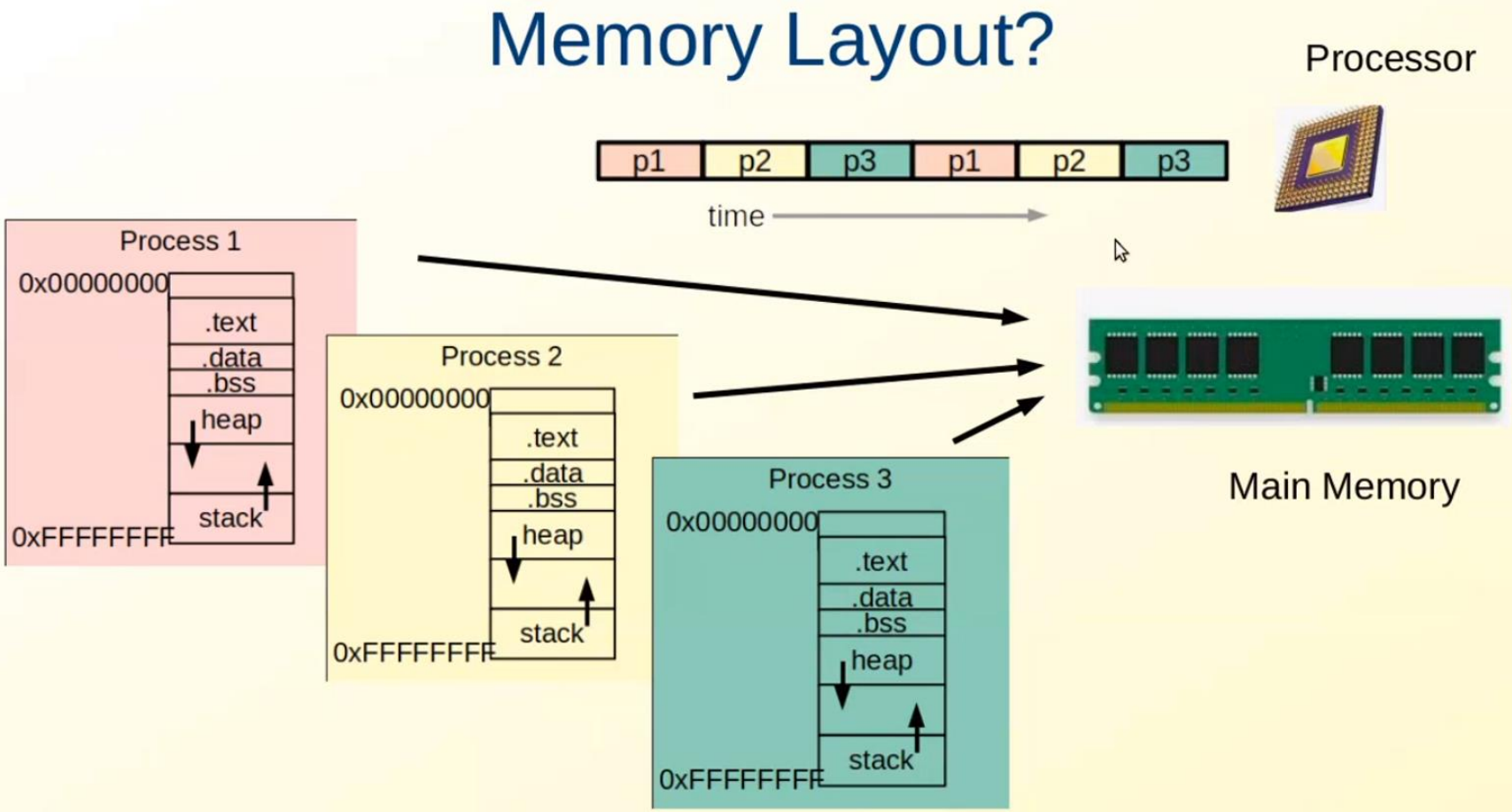


Process:

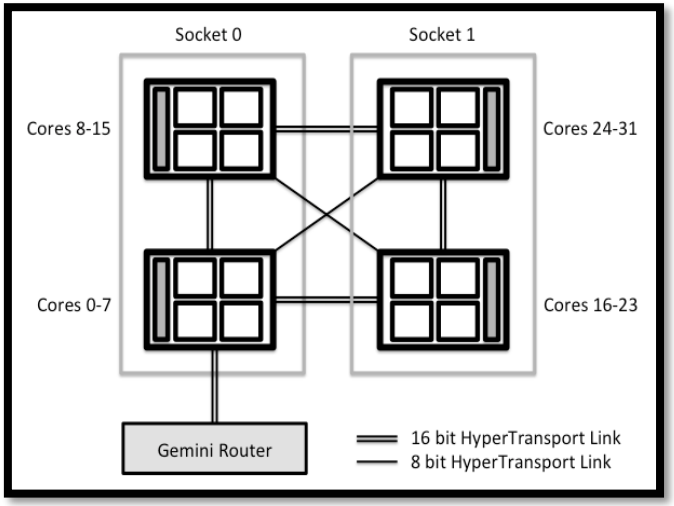
* An instance of a program execution.
* The execution context of a running program... i.e. the resources associated with a program's execution.

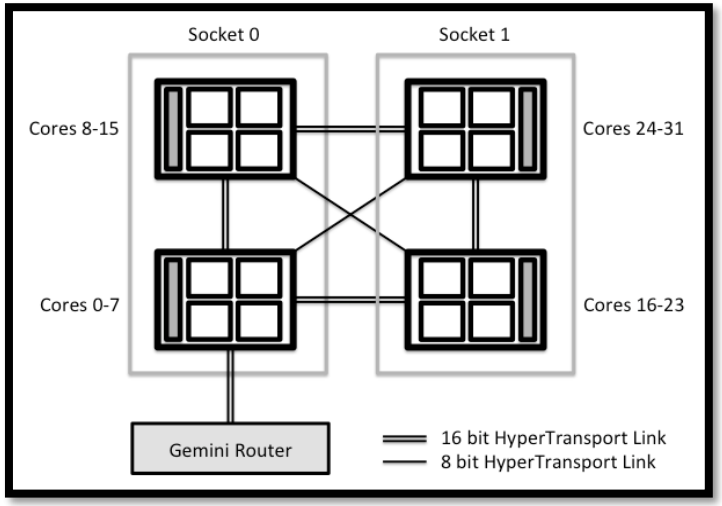
Adapted from Tim Mattson's slides

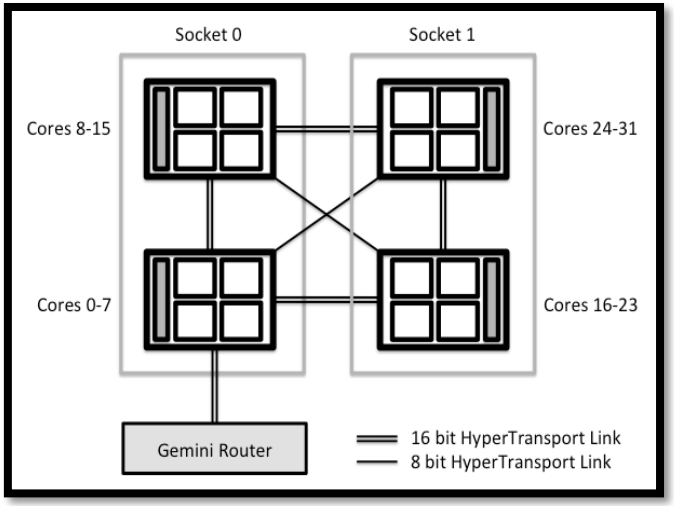
Memory Layout?

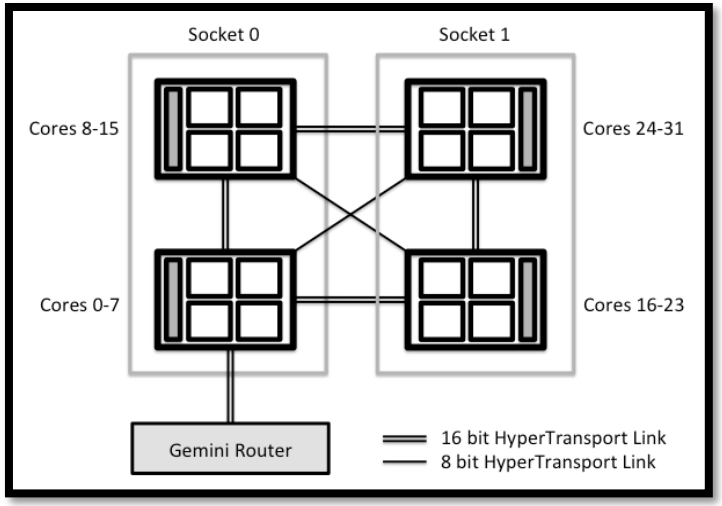


Adapted from Neha Karanjkar's slides









Process Placement

mpirun -np 6 ./program.x

Beowulf PC Clusters: Breaking the Cost Barrier to High End Application Computing

Thomas Sterling, PhD

"Leader of NASA BEOWULF Project"
Senior Staff Scientist, HPC Systems Group, Jet Propulsion Lab
Visiting Professor, California Institute of Technology

This exciting presentation emphasizes the economical opportunity of networking low cost personal computers (PCs) to solve science and engineering problems. Such PC networks permit parallel or distributed computing in an effective and efficient manner. Various high performance computing applications using personal computer networks are addressed in terms of science performed and performance achieved. Performance metrics discussed include speedup vs. costs vs. scalability. Such experience has evolved with Dr. Sterling's NASA BEOWULF project and associated BEOWULF-class computing. Come and discuss your applications using a PC network for high performance computation!

FROM TOYS TO TERAFLOPS: BRIDGING THE BEOWULF GAP

Thomas Sterling
Daniel F. Savarese

CENTER FOR ADVANCED COMPUTING RESEARCH, CALIFORNIA INSTITUTE OF TECHNOLOGY, PASADENA, CALIFORNIA, U.S.A.

Summary

Do-it-yourself supercomputing has emerged as a solution to cost-effectively sustain the computational demands of the scientific research community. Despite some of the successes of this approach, represented by Beowulf-class computing, it has limitations that need to be recognized as well as problems that need to be resolved in order to extend its scope of applicability. While the performance of hardware incorporated into these systems has continued to improve at a remarkable rate, enabling the execution of steadily larger and more compute-intensive applications, the software environment of the machines has seen little to no improvement or evolution. The authors find

1 A Crossroads of Computing

Over the past 5 years, the do-it-yourself supercomputing phenomenon has grown in popularity as the ranks of high performance computer vendors have winnowed and the performance of commodity microprocessors has steadily improved. With the maturation of no-cost operating systems and compilers into high performance commercial-quality software, in addition to the establishment of standard parallel programming models, the case has become more compelling than ever for research organizations to assemble their own parallel computing platforms. The high cost of commercial supercomputers and the oversubscription of resources at supercomputing centers have forced many researchers to resort to their own means in order to run computational simulations. Ensembles of networked desktop computers have become a liberating force for computational science, enabling supercomputing to be performed in the figurative backyard.

We have entered a crossroads, where the notion of supercomputing is losing, or at least redefining, its meaning, as it becomes subsumed by the more general notion of parallel computing. The prefix "super" is applied to a noun to highlight its extraordinary nature, elevating it above the norm. Commodity-clustered computing is anything but extraordinary. Yet, there is little to distinguish

IJHPCA 1999 (*Informative paper)*

csews*

lscpu

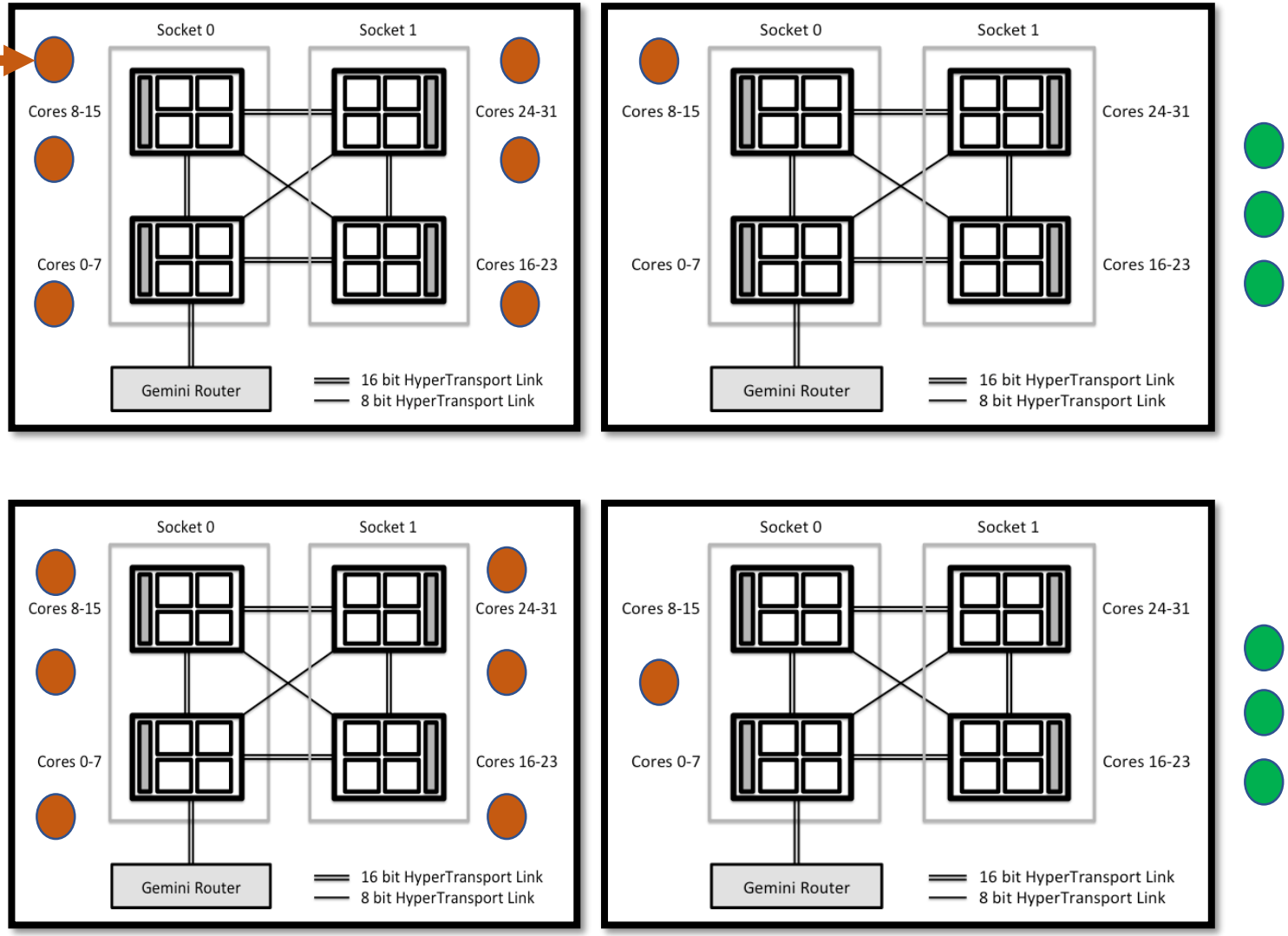
```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              12
On-line CPU(s) list: 0-11
Thread(s) per core:  2
Core(s) per socket:  6
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               158
Model name:          Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
Stepping:            10
CPU MHz:             900.353
CPU max MHz:         4600.0000
CPU min MHz:         800.0000
BogoMIPS:            6384.00
Virtualization:      VT-x
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            12288K
NUMA node0 CPU(s):   0-11
```

```
processor       : 0
processor       : 1
processor       : 2
processor       : 3
processor       : 4
processor       : 5
processor       : 6
processor       : 7
processor       : 8
processor       : 9
processor       : 10
processor       : 11
```

Beowulf Cluster

mpirun -np 6 ./program.x

Running program



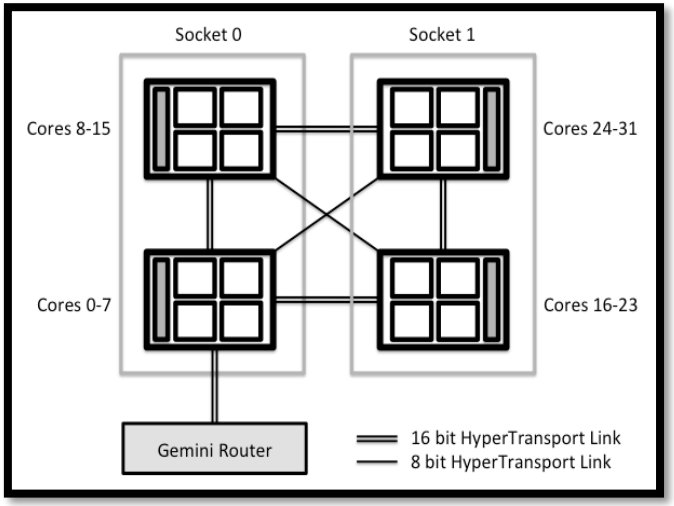
Execution Parameters

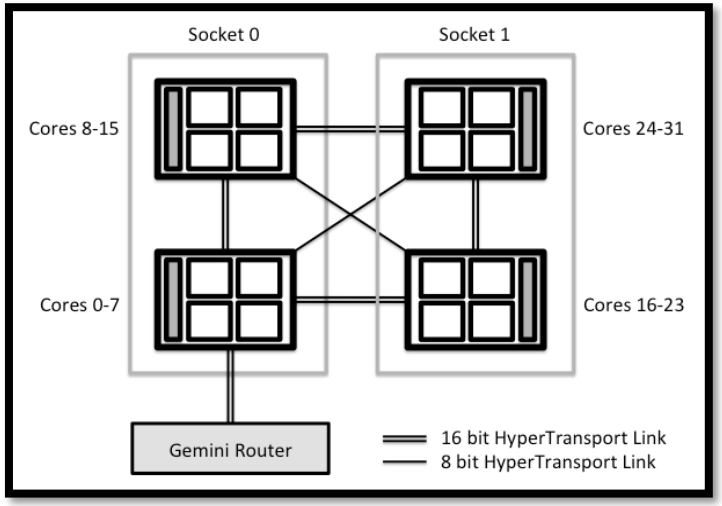
MPI process
MPI process

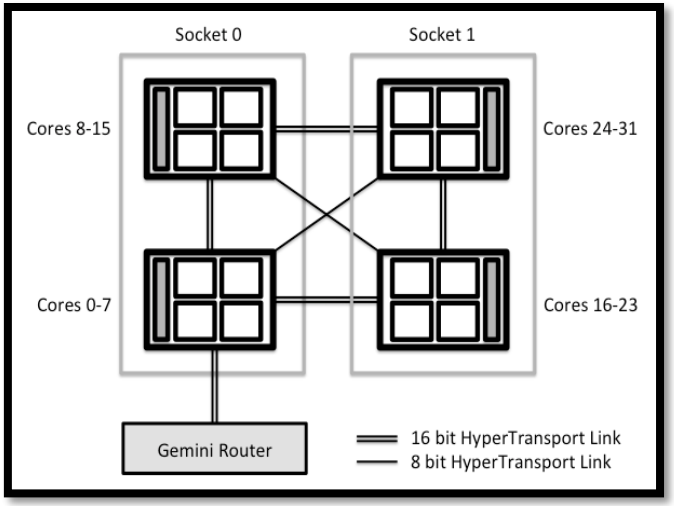
Number of nodes = 4
Processes per node (ppn) = 2

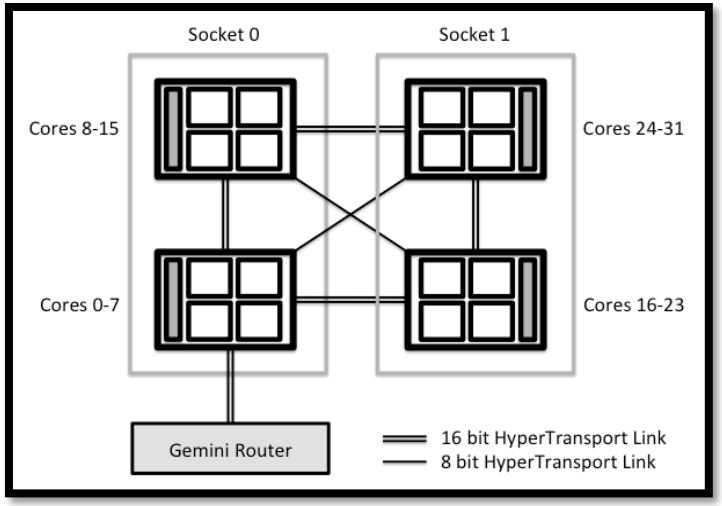
Process Placement on Unmanaged Cluster

```
mpirun –np 8 –f hostfile ./program.x
```



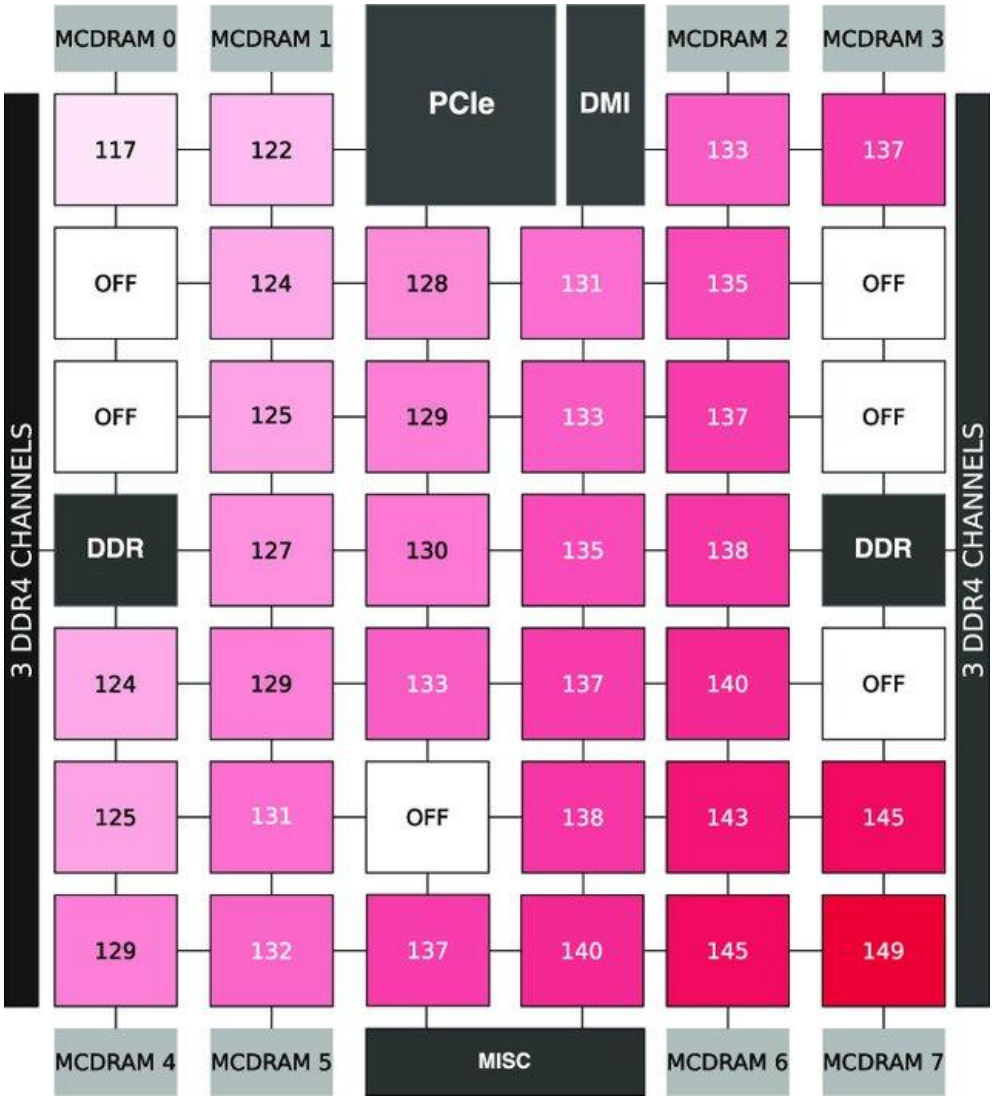






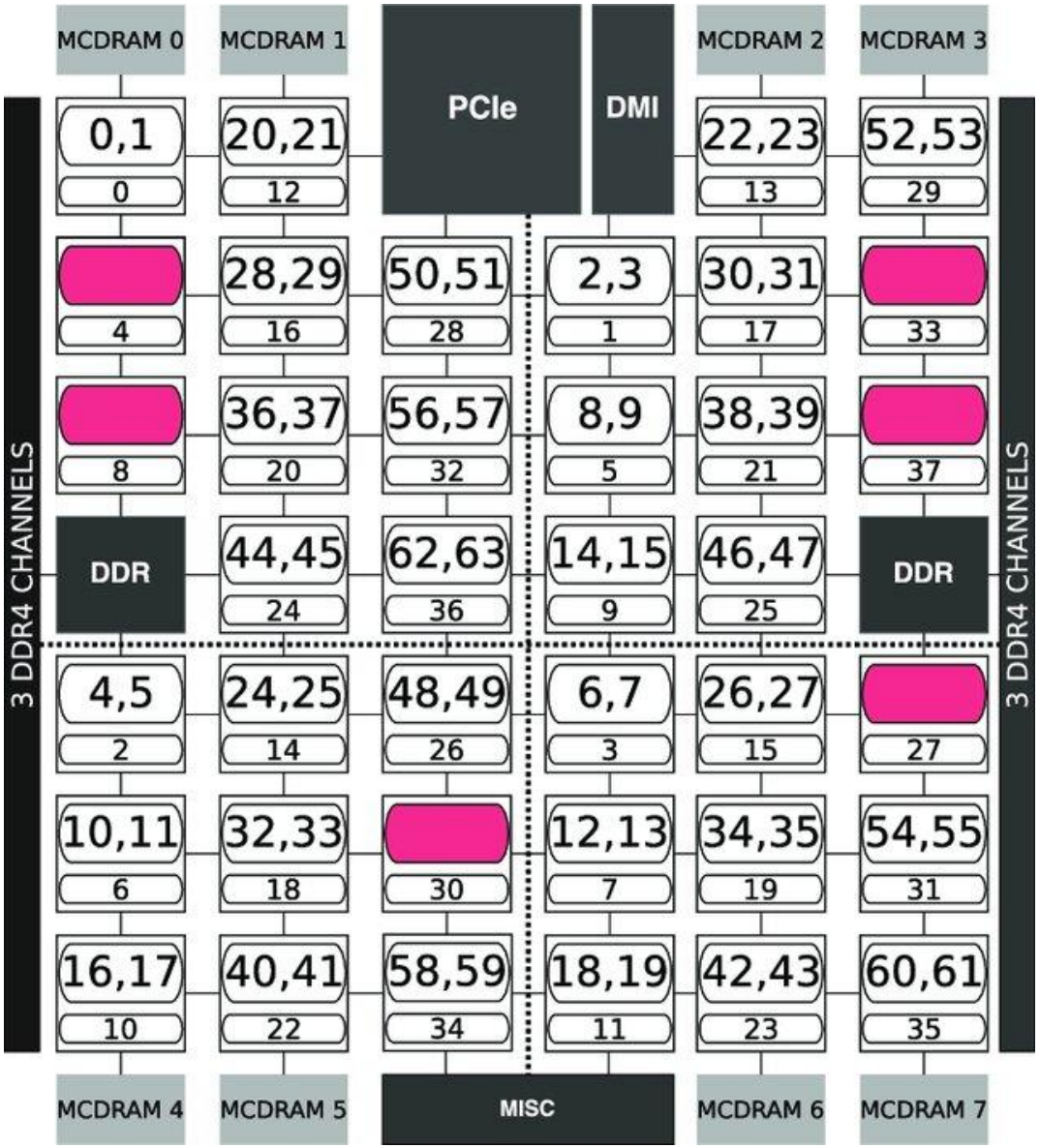
Multicore Node (Intel KNL)

Figure shows heatmap of the measured access latency (in CPU cycles) from each tile in the mesh of an Intel Xeon Phi x200 7210 to a single block of memory associated to MCDRAM #0 and its adjacent CHA.



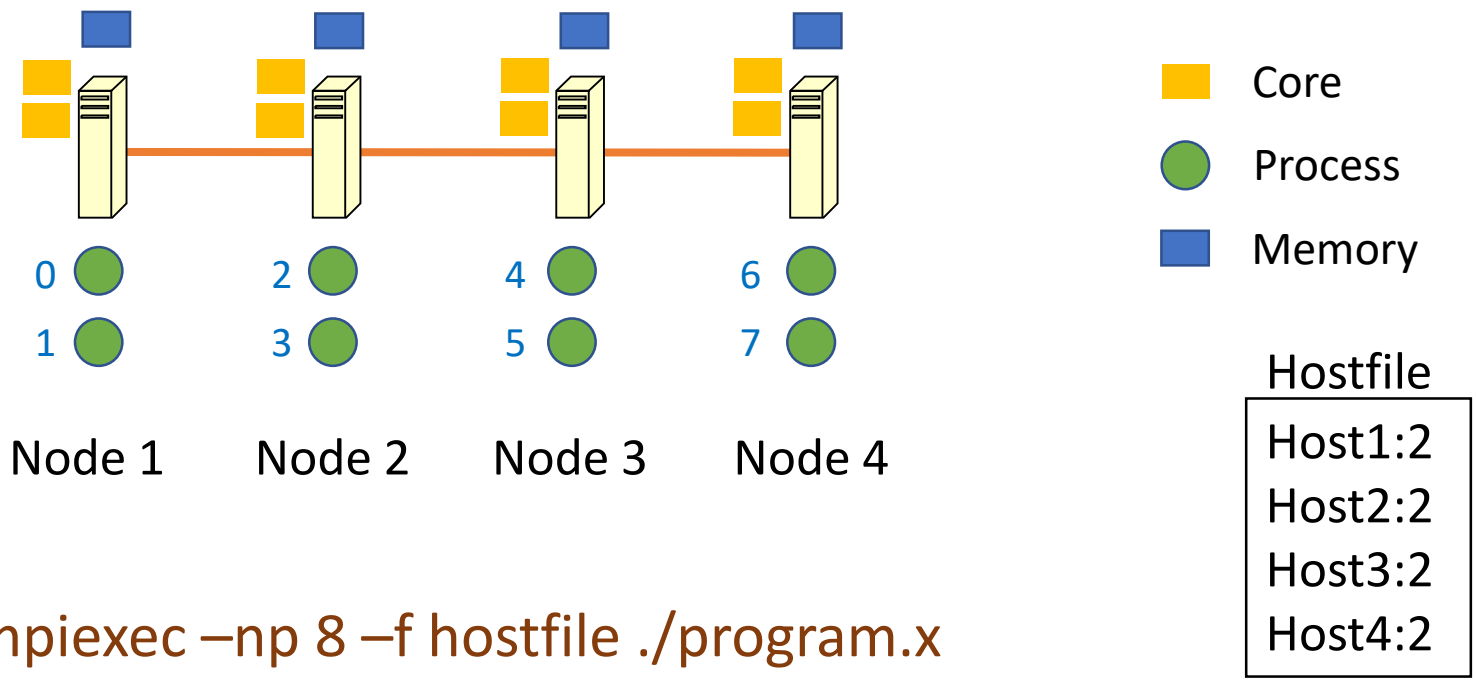
Simulating the Network Activity of Modern Manycores, IEEE Access 2019

Cores



Simulating the Network Activity of Modern Manycores, IEEE Access 2019

Multiple Processes Per Node



mpiexec –np 8 –f hostfile ./program.x

Run examples/cpi (mpirun –np 8 <path to examples/cpi>)

Hostfile

172.27.19.1	172.27.19.1:1	172.27.19.1:4
172.27.19.2	172.27.19.2:1	172.27.19.2:4
172.27.19.3	172.27.19.3:1	172.27.19.3:4
172.27.19.4	172.27.19.4:1	172.27.19.4:4

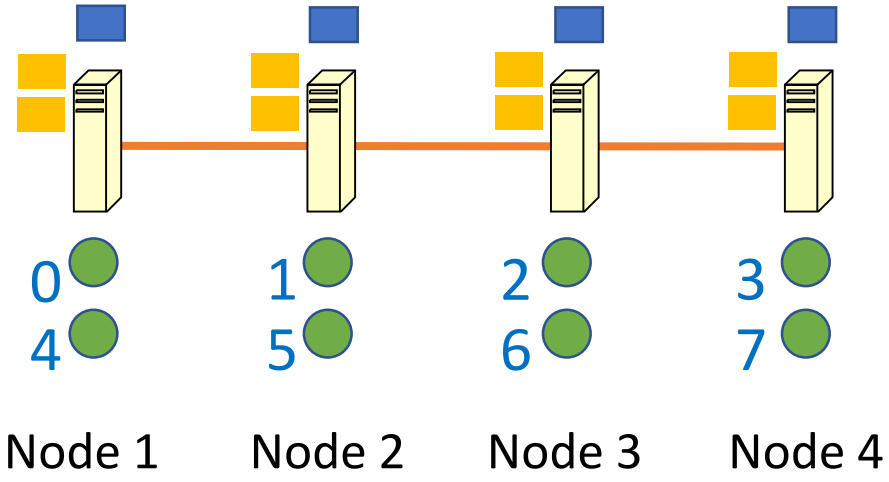
Round-robin Placement

172.27.19.1

172.27.19.2

172.27.19.3

172.27.19.4



mpiexec –np 8 –f hostfile ./program.x

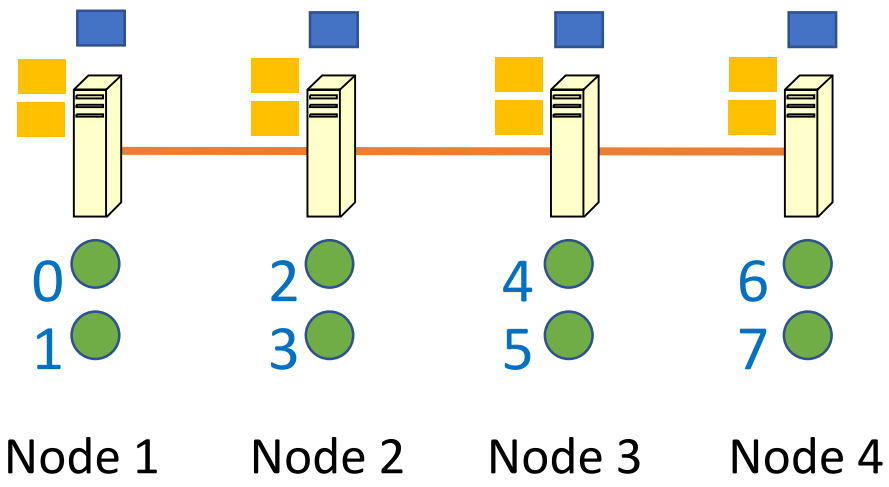
Sequential Placement

172.27.19.1:2

172.27.19.2:2

172.27.19.3:2

172.27.19.4:2



mpiexec –np 8 –f hostfile ./program.x

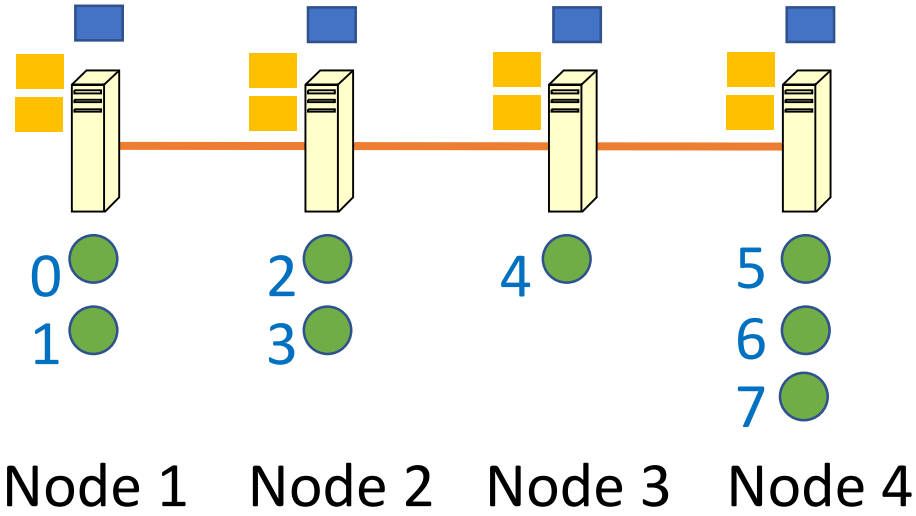
Contiguous Placement

172.27.19.1:2

172.27.19.2:2

172.27.19.3:1

172.27.19.4:3



mpiexec –np 8 –f hostfile ./program.x

MPI Code Execution Options

Same host

- mpirun -np 6 ./program.x

Multiple hosts

- mpirun -np 6 –f hostfile ./program.x
  - Round robin process placement
    - mpirun -np 6 –hosts csews1,csews2 ./program.x
  - Sequential/contiguous placement
    - mpirun -np 6 –hosts csews1:3,csews2:3 ./program.x

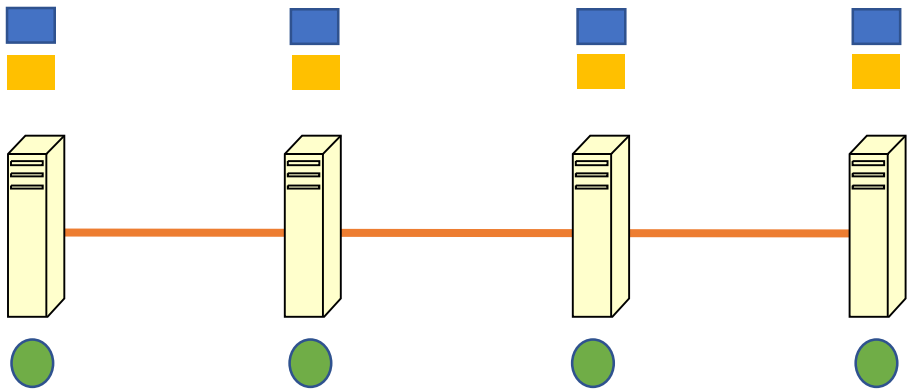
Disassembled Code Example

```
950:   48 89 d6                mov    %rdx,%rsi
953:   48 89 c7                mov    %rax,%rdi
956:   e8 75 fe ff ff          call   7d0 <MPI_Init@plt>
95b:   48 8d 45 d4             lea    -0x2c(%rbp),%rax
95f:   48 89 c6                mov    %rax,%rsi
962:   bf 00 00 00 44          mov    $0x44000000,%edi
967:   e8 34 fe ff ff          call   7a0 <MPI_Comm_rank@plt>
96c:   48 8d 45 d8             lea    -0x28(%rbp),%rax
970:   48 89 c6                mov    %rax,%rsi
973:   bf 00 00 00 44          mov    $0x44000000,%edi
978:   e8 63 fe ff ff          call   7e0 <MPI_Comm_size@plt>
97d:   8b 45 d4                mov    -0x2c(%rbp),%eax
980:   83 c0 02                add    $0x2,%eax
983:   89 45 e0                mov    %eax,-0x20(%rbp)
986:   8b 45 e0                mov    -0x20(%rbp),%eax
989:   89 45 dc                mov    %eax,-0x24(%rbp)
98c:   48 8d 45 e0             lea    -0x20(%rbp),%rax
990:   41 b8 00 00 00 44       mov    $0x44000000,%r8d
996:   b9 00 00 00 00          mov    $0x0,%ecx
99b:   ba 05 04 00 4c          mov    $0x4c000405,%edx
9a0:   be 01 00 00 00          mov    $0x1,%esi
9a5:   48 89 c7                mov    %rax,%rdi
9a8:   e8 43 fe ff ff          call   7f0 <MPI_Bcast@plt>
9ad:   8b 4d e0                mov    -0x20(%rbp),%ecx
9b0:   8b 45 d4                mov    -0x2c(%rbp),%eax
9b3:   8b 55 dc                mov    -0x24(%rbp),%edx
9b6:   89 c6                   mov    %eax,%esi
9b8:   48 8d 3d b5 00 00 00    lea    0xb5(%rip),%rdi        # a74 <_IO_stdin_used+0x4>
9bf:   b8 00 00 00 00          mov    $0x0,%eax
9c4:   e8 f7 fd ff ff          call   7c0 <printf@plt>
9c9:   e8 32 fe ff ff          call   800 <MPI_Finalize@plt>
9ce:   b8 00 00 00 00          mov    $0x0,%eax
9d3:   48 8b 4d f8             mov    -0x8(%rbp),%rcx
9d7:   64 48 33 0c 25 28 00    xor    %fs:0x28,%rcx
```

Parallel Execution (ppn=1)

```
$ ps (on each host)

parallel.x
```

SIMD



Core
Process
Memory

Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …

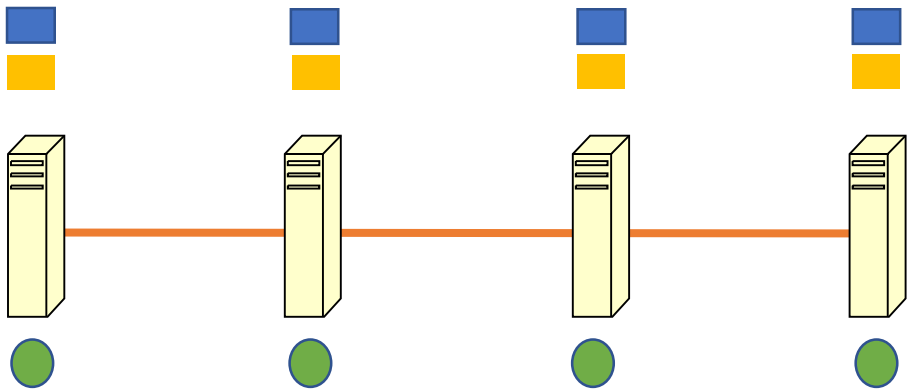
parallel.x parallel.x parallel.x parallel.x

mpirun –np 4 –f hostfile ./parallel.x

Parallel Execution (ppn=2)

```
$ ps (on each host)

parallel.x
parallel.x
```

SIMD



Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …
parallel.x parallel.x	parallel.x parallel.x	parallel.x parallel.x	parallel.x parallel.x

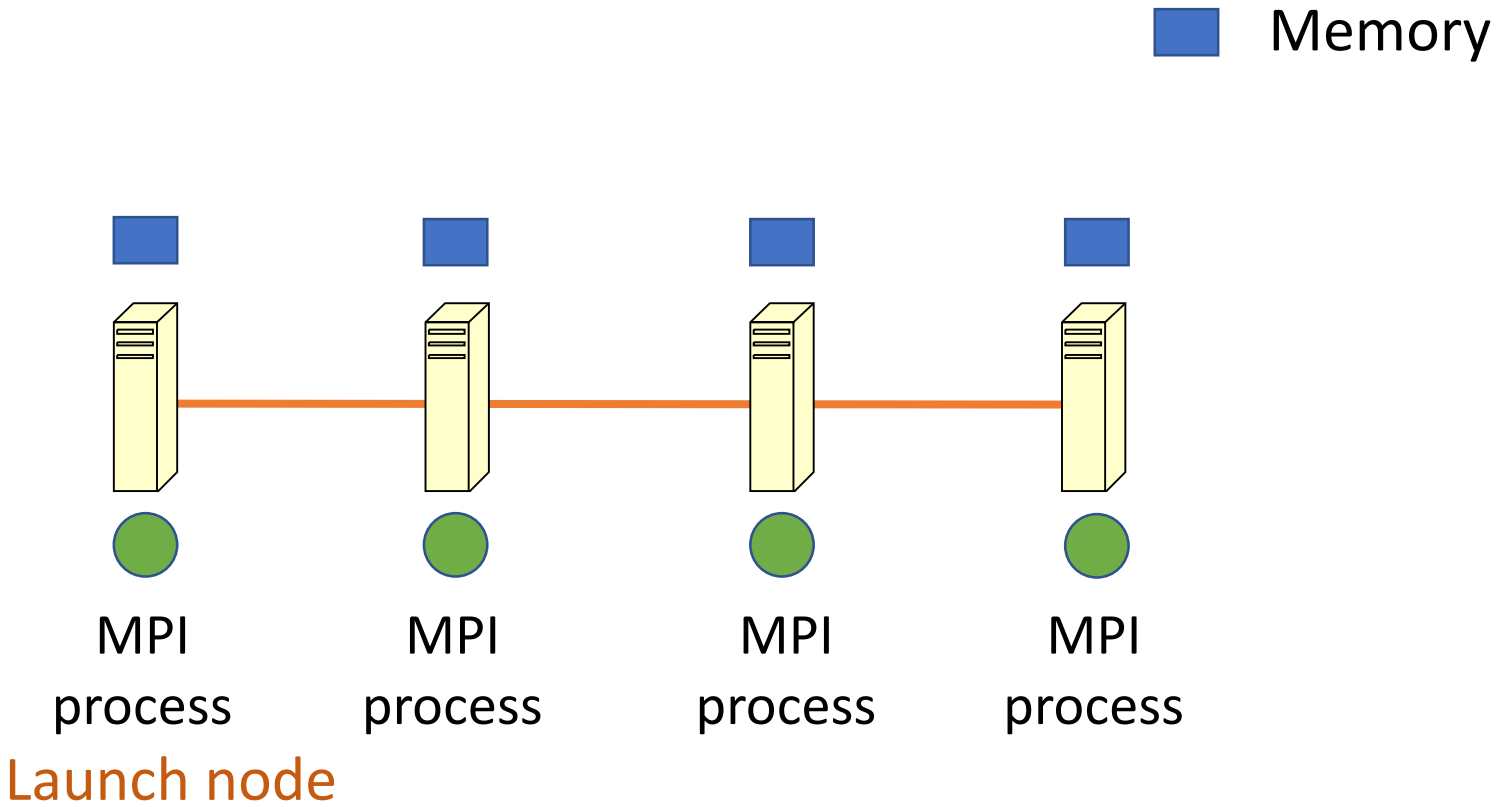
Core

Process

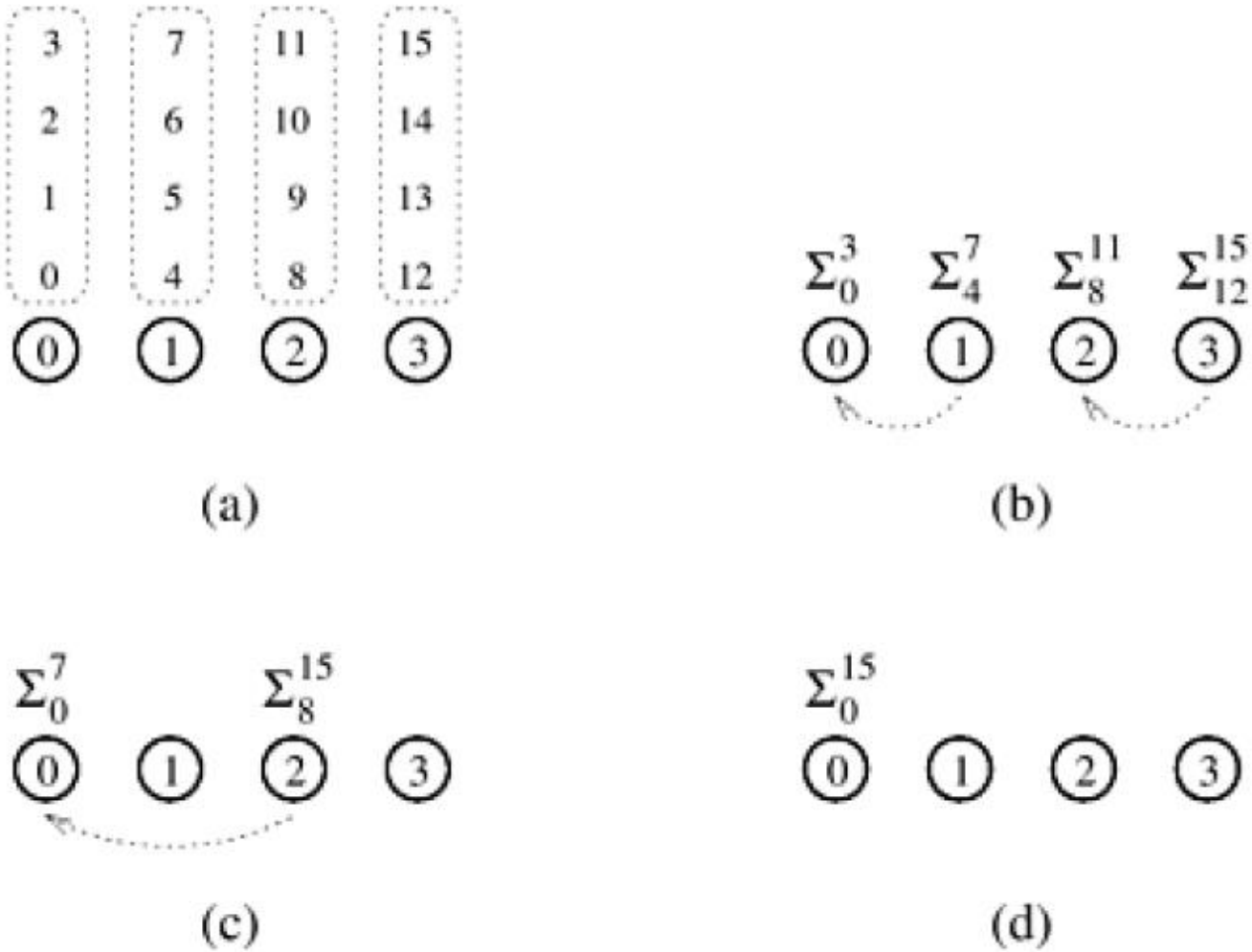
Memory

mpirun –np 8 –f hostfile ./parallel.x

Process Launch



Parallel Sum (Optimized)



Source: GGKK Chapter 5

Homework

Analyze the communication endpoints for the optimized algorithm of parallel sum for round-robin and sequential placement of 8 and 16 processes.

Getting Started with MPI

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
  // initialize MPI
  MPI_Init (&argc, &argv);

  printf ("Hello, world!\n");

  // done with MPI
  MPI_Finalize();
}

~
~
```

- gather information about the parallel job
- set up internal library state
- prepare for communication

Initialization

Finalization

MPI Code Execution Steps

- Compile
  - mpicc -o program.x program.c
- Execute
  - mpirun -np 1 ./program.x (mpiexec -np 1 ./program.x)
    - Runs 1 process on the launch/login node
  - mpirun -np 6 ./program.x
    - Runs 6 processes on the launch/login node

Output – Hello World

mpirun –np 20 ./program.x

```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!Hello, world!
Hello, world!
Hello, world!
Hello, world!

Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

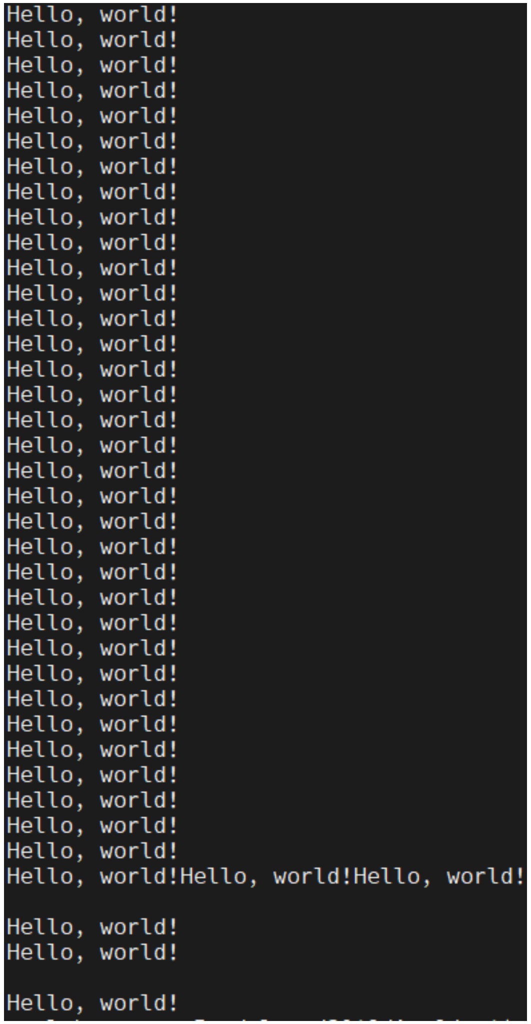
```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!

Hello, world!
```

```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

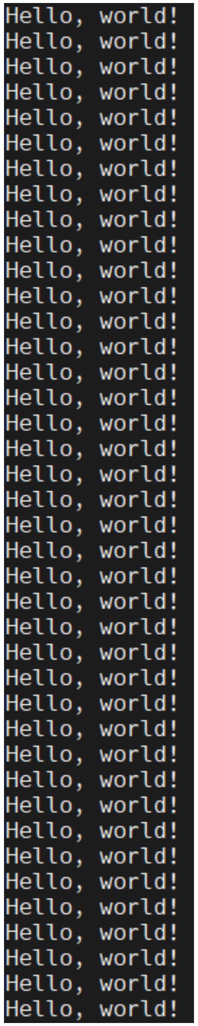
Output on Multiple Hosts

mpirun –np 40 –hosts host1,host2 ./program.x

```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!Hello, world!
Hello, world!


Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

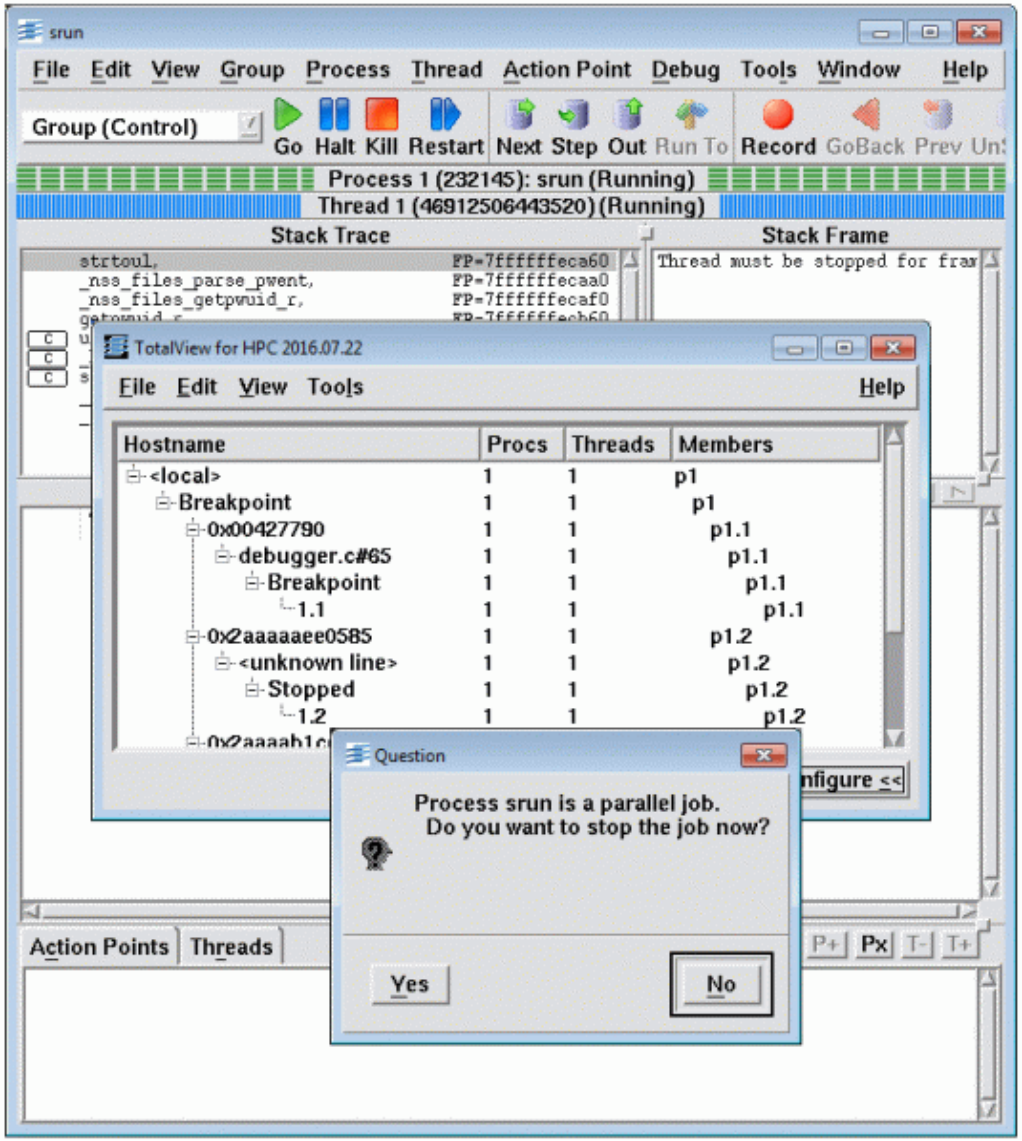


```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!Hello, world!Hello, world!

Hello, world!
Hello, world!

Hello, world!
```



```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

TotalView Debugger



MPI Process Identification

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

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

    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);



    // Print off a hello world message
    printf("Hello I am rank %d out of %d processes\n", rank, size);

    // Finalize the MPI environment.
    MPI_Finalize();
}

~
~
```

Global communicator

Total number of processes

Rank of a process

Entities

Process

- Belongs to a group
- Identified by a rank within a group

Identification

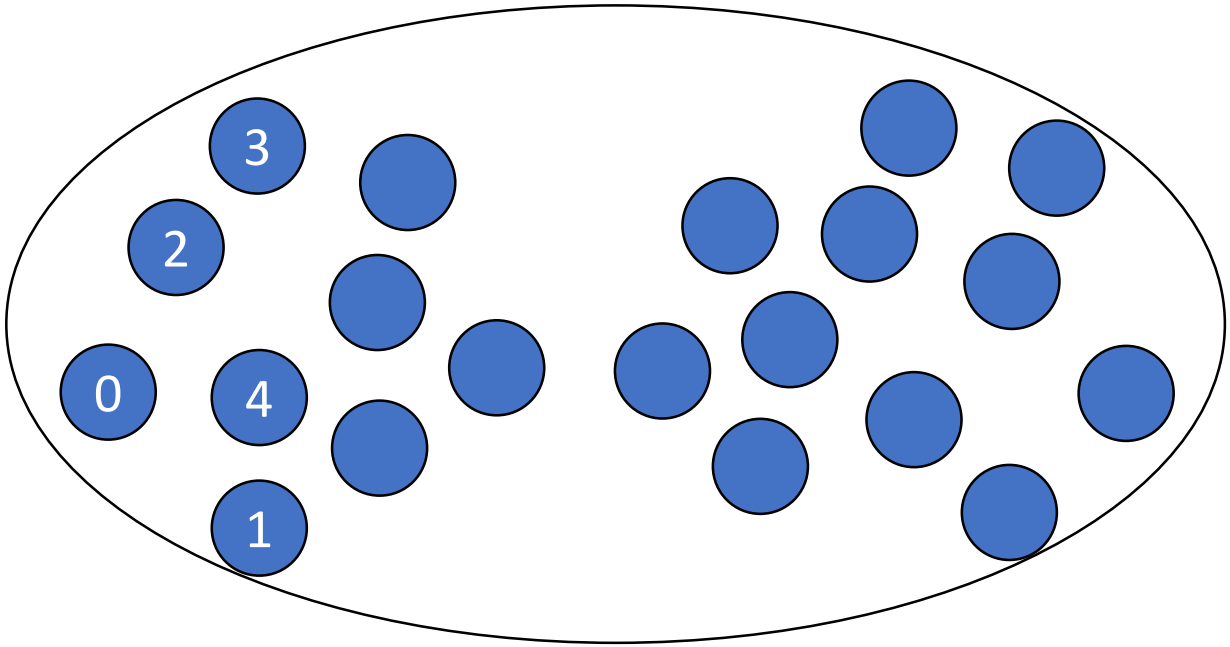
- MPI_Comm_size – total number of processes in communicator
- MPI_Comm_rank – rank in the communicator

Communicator (communication handle)

- Defines the scope
- Specifies communication context

MPI_COMM_WORLD

- Required in every MPI communication
- Process identified by rank/id



Communicator

- Communication handle among a group/collection of processes
- Representative of communication domain
- Associated with a context ID (in MPICH)
- Predefined:
  - MPI_COMM_WORLD
  - MPI_COMM_SELF

Output

```
// get number of tasks
 MPI_Comm_size (MPI_COMM_WORLD, &numtasks);

 // get my rank
 MPI_Comm_rank (MPI_COMM_WORLD, &rank);

printf ("Hello I'm rank %d of %d processes\n", rank, numtasks);
```

RR mpiexec –n 4 –hosts host1,host2,host3,host4 ./exe

mpiexec –n 4 –hosts host1:2,host2:2,host3:2,host4:2 ./exe

RR mpiexec –n 8 –hosts host1,host2,host3,host4 ./exe

mpiexec –n 8 –hosts host1:1,host2:2,host3:3,host4:4 ./exe

Host

pmalakar@172.27.19.10

Terminal Sessions View X server Tools Games Settings Macros Help

2. pmalakar@172.27.19.10

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

int main(int argc, char *argv[])
{
  int numtasks, rank, len;
  char hostname[MPI_MAX_PROCESSOR_NAME];

  // initialize MPI
  MPI_Init (&argc, &argv);

  // get number of tasks
  MPI_Comm_size (MPI_COMM_WORLD, &numtasks);

  // get my rank
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);

  // this one is obvious
  MPI_Get_processor_name (hostname, &len);

  printf ("Number of tasks=%d My rank=%d Running on %s\n", numtasks, rank, hostname);

  // done with MPI
  MPI_Finalize();
}
```

csews*

lscpu

```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              12
On-line CPU(s) list: 0-11
Thread(s) per core:  2
Core(s) per socket:  6
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               158
Model name:          Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
Stepping:            10
CPU MHz:             900.353
CPU max MHz:         4600.0000
CPU min MHz:         800.0000
BogoMIPS:            6384.00
Virtualization:      VT-x
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            12288K
NUMA node0 CPU(s):   0-11
```

```
processor       : 0
processor       : 1
processor       : 2
processor       : 3
processor       : 4
processor       : 5
processor       : 6
processor       : 7
processor       : 8
processor       : 9
processor       : 10
processor       : 11
```

Multiple Tasks Core ID

```c
#include <stdio.h>
#include <sched.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
  int numtasks, rank, len, coreID;
  char hostname[MPI_MAX_PROCESSOR_NAME];

  // initialize MPI
  MPI_Init (&argc, &argv);

  // get number of tasks
  MPI_Comm_size (MPI_COMM_WORLD, &numtasks);

  // get my rank
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);

  // get hostname
  MPI_Get_processor_name (hostname, &len);

  // core ID
  coreID = sched_getcpu();

  printf ("%d of %d Running on %s:%d\n", rank, numtasks, hostname, coreID);

  // done with MPI
  MPI_Finalize();
}
```

Get the CPU core ID

Process Placement

pmalakar@172.27.19.10

Terminal Sessions View X server Tools Games Settings Macros Help

2. pmalakar@172.27.19.10 | 3. pmalakar@172.27.19.10

```
cs633 mpirun -np 12 ./multipletaskscoreID
0 of 12 Running on csews10:0
1 of 12 Running on csews10:3
2 of 12 Running on csews10:6
6 of 12 Running on csews10:9
7 of 12 Running on csews10:1
8 of 12 Running on csews10:8
11 of 12 Running on csews10:7
3 of 12 Running on csews10:5
4 of 12 Running on csews10:11
5 of 12 Running on csews10:4
9 of 12 Running on csews10:10
10 of 12 Running on csews10:2
cs633
```

Process Placement Output

```
pmalakar@csews2:~/class/          mpirun -np 4 -hosts csews1,csews2 ./3.multipletaskscoreID | sort -k1n
0 of 4 Running on csews1:2
1 of 4 Running on csews2:6
2 of 4 Running on csews1:10
3 of 4 Running on csews2:10
pmalakar@csews2:~/class/          mpirun -np 4 -hosts csews1,csews2 ./3.multipletaskscoreID | sort -k1n
0 of 4 Running on csews1:2
1 of 4 Running on csews2:3
2 of 4 Running on csews1:7
3 of 4 Running on csews2:5
pmalakar@csews2:~/class/          mpirun -np 4 -hosts csews1:2,csews2:2 ./3.multipletaskscoreID | sort -k1n
0 of 4 Running on csews1:8
1 of 4 Running on csews1:10
2 of 4 Running on csews2:2
3 of 4 Running on csews2:5
pmalakar@csews2:~/class/          mpirun -np 4 -hosts csews1:2,csews2:2 ./3.multipletaskscoreID | sort -k1n
0 of 4 Running on csews1:4
1 of 4 Running on csews1:2
2 of 4 Running on csews2:3
3 of 4 Running on csews2:6
```

Message Passing Paradigm

- Message sends and receives
- Explicit communication

Communication patterns

- Point-to-point
- Collective

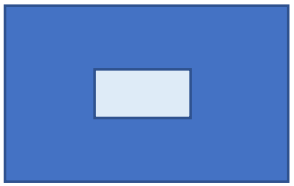
MPI Data Types

- MPI_BYTE
- MPI_CHAR
- MPI_INT
- MPI_FLOAT
- MPI_DOUBLE

Point-to-point Communication

Blocking send and receive

MPI_Send

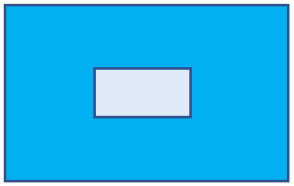


SENDER

int MPI_Send (const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Tags should match

MPI_Recv



RECEIVER

int MPI_Recv (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)