Research Computing
UNIVERSITY OF COLORADO

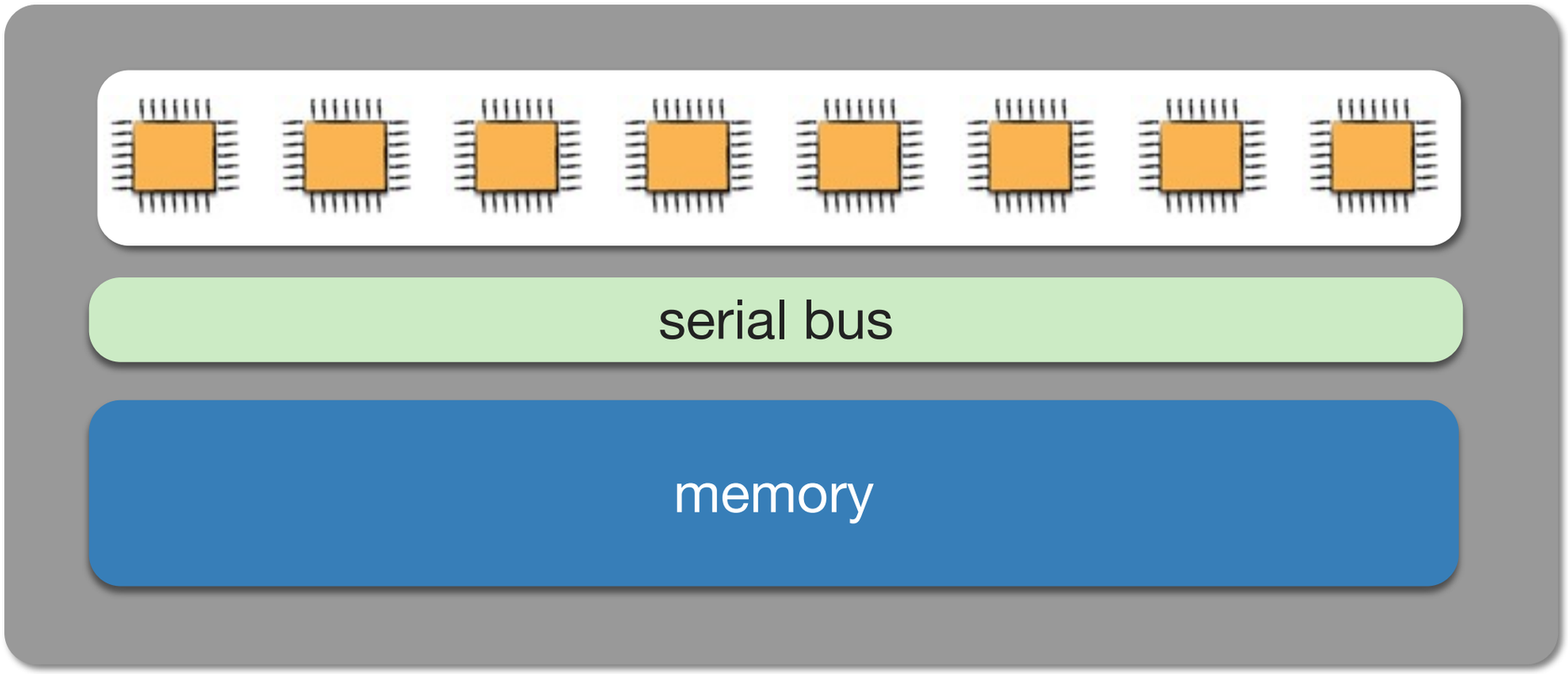
NUMA Architecture and Hybrid OpenMP-MPI

NUMA Architecture

Symmetric Multiprocessing (SMP)

Each CPU can access any memory location in the same amount of time as any other CPU in the system.

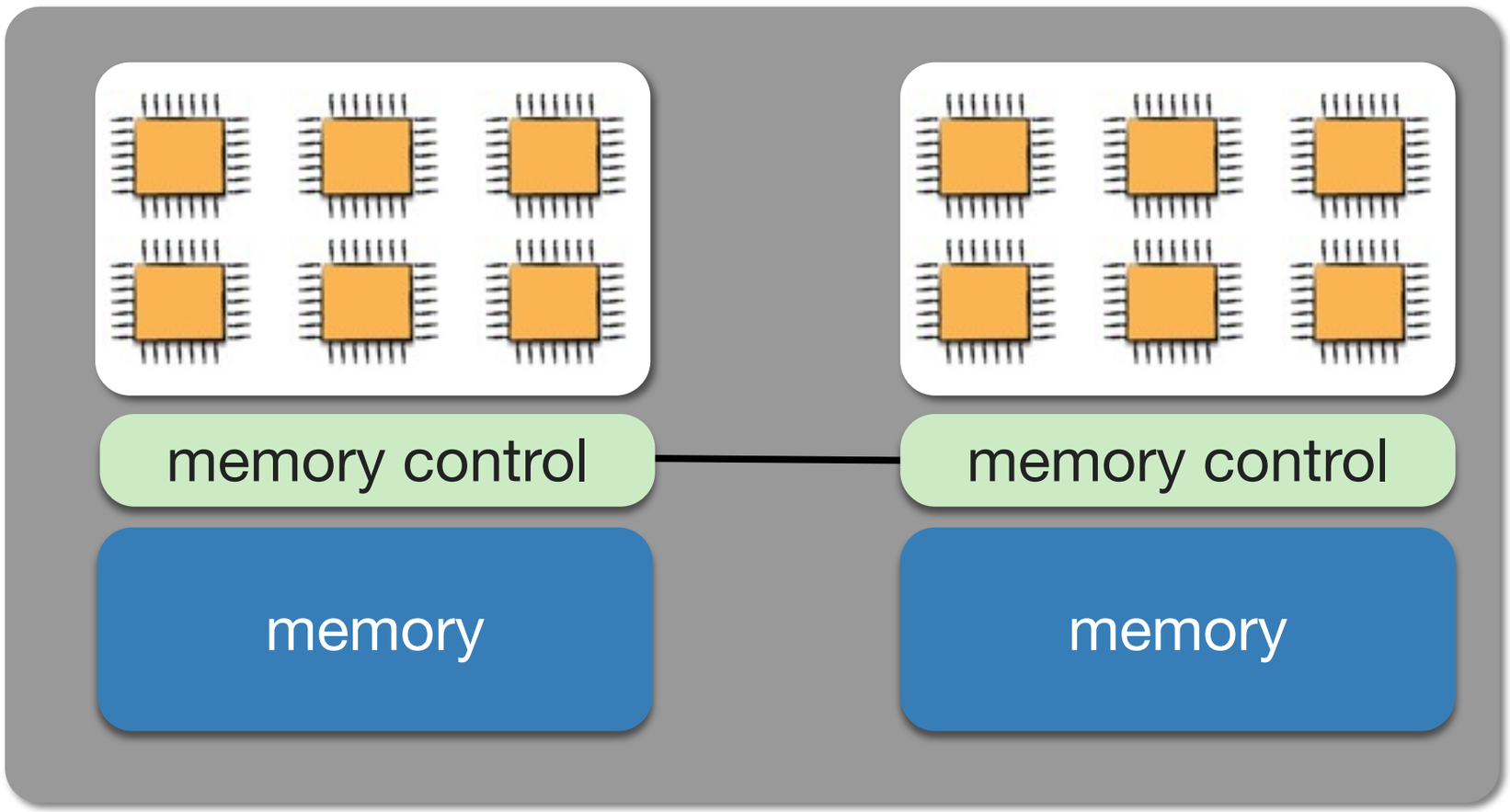
Fine for a small number of CPUs (< ~ 8 - 16)



Non-Uniform Memory Access (NUMA)

Each socket has a dedicated memory area for high speed access

Also has an interconnect to other sockets for slower access to the other sockets' memory



Affinity

Keeping threads close to the data they access

Problem:

- CPU on socket 0 can take twice as long to access memory on socket 1
- Avoid regularly accessing remote memory in a NUMA topology system

How do we ensure that information travels the shortest path?

- Bind thread to processor (so OS doesn't move them)
- Bind thread to memory near processor

Stream

Simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s)

Creates three arrays, called a, b, and c, each of a fixed size N with double-precision elements

Initializes these arrays

Performs the following sequence of streaming benchmarks

Copy: c[k] = a[k] (one read and one write)

Scale: b[k] = alpha*c[k] (one read and one write)

Add: c[k] = a[k] + b[k] (two reads and one write)

Triad: a[k] = b[k] + alpha*c[k] (two reads and one write)

Single job on a node

```
#PBS -N example_1
#PBS -q janus-debug
#PBS -l walltime=0:00:30
#PBS -l nodes=1:ppn=12

cd $PBS_O_WORKDIR

./stream
```

```
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       10092.7727       0.0318       0.0317       0.0319
Scale:       8246.1557       0.0388       0.0388       0.0390
Add:        11107.5515       0.0432       0.0432       0.0433
Triad:      11122.2173       0.0432       0.0432       0.0432
```

Multiple jobs on a node

```
#PBS -N example_1
#PBS -q janus-debug
#PBS -l walltime=0:00:30
#PBS -l nodes=1:ppn=12

cd $PBS_O_WORKDIR

./stream &
./stream &
./stream &
./stream &
./stream &
./stream &
```

```
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:        4109.7208       0.0781       0.0779       0.0786
Scale:       4983.1156       0.0660       0.0642       0.0668
Add:         6205.3184       0.0784       0.0774       0.0788
Triad:       6148.8038       0.0782       0.0781       0.0784
```

Making it worse...

```
#PBS -N example_1
#PBS -q janus-debug
#PBS -l walltime=0:00:30
#PBS -l nodes=1:ppn=12

cd $PBS_O_WORKDIR

numactl --cpunodebind=0 --membind=0 ./stream &
numactl --cpunodebind=0 --membind=0 ./stream &
numactl --cpunodebind=0 --membind=0 ./stream &
numactl --cpunodebind=0 --membind=0 ./stream &
numactl --cpunodebind=0 --membind=0 ./stream &
numactl --cpunodebind=0 --membind=0 ./stream &
```

```
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:        2373.2332       0.1351       0.1348       0.1358
Scale:       3428.9513       0.0974       0.0933       0.1024
Add:         3693.3611       0.1346       0.1300       0.1363
Triad:       4098.7435       0.1327       0.1171       0.1353
```

OpenMP Stream

```
export OMP_NUM_THREADS=6

./stream_omp

Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:        24510.1768       0.0033       0.0033       0.0033
Scale:       24330.6736       0.0033       0.0033       0.0033
Add:         25141.9392       0.0048       0.0048       0.0049
Triad:       25423.8763       0.0047       0.0047       0.0047
```

OpenMP Stream

```
export OMP_NUM_THREADS=6

numactl --membind=0 ./stream_omp

Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       13867.7600       0.0058       0.0058       0.0058
Scale:      13500.0732       0.0060       0.0059       0.0061
Add:        14450.6598       0.0083       0.0083       0.0084
Triad:      14857.1739       0.0081       0.0081       0.0082
```

When is this important?

Intel MKL DGEMM (matrix multiply)

> if the data are not distributed, `numactl --interleave=all` a higher performance compared to default mode
>
> Distributing matrices in the default mode has almost the same effect as applying NUMA interleave memory policy.

Similar findings for Linpack

> NUMA helps whenthe whole matrix is initialized (and previously allocated) in the master thread

Hybrid OpenMP and MPI

Motivation

Introducing MPI into OpenMP applications can help scale across multiple nodes

Introducing OpenMP into MPI applications can help make more efficient use of the shared memory on nodes mitigating the need for explicit intra-node communication

Introducing MPI and OpenMP during the design/coding of a new application can help maximize efficiency, performance, and scaling

Details

OpenMP

- Launch one process per node
- Have each process fork one thread per core
- Share data using shared memory

MPI

- Launch one process per core
- Pass messages among processes without concern for location

Hybrid OpenMP/MPI

- we want each MPI process to launch multiple OpenMP threads that can share local memory

Configurations

Treat each node as an SMP

launch a single MPI process per node

create parallel threads sharing full-node memory

12 threads per node on JANUS

Treat each socket as an SMP

launch one MPI process on each socket

create parallel threads sharing same-socket memory

6 threads per socket on JANUS

Creating Configurations

To achieve configurations like these, we must be able to:

- Assign to each process/thread an affinity for some set of cores
- Make sure the allocation of memory is appropriately matched

Hello World!

```
    #include <iostream>
#include <mpi.h>
#include <omp.h>

int main(int argc, char ** argv)
{
    int provided;
    MPI_Init_thread(&argc, &argv, MPI_THREAD_SINGLE, &provided);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);

    int thread_id, thread_size;
    #pragma omp parallel private(thread_id, thread_size)
    {
      thread_id = omp_get_thread_num();
      thread_size = omp_get_num_threads();
      #pragma omp critical
      std::cout << rank << " " << thread_id << " " << thread_size << std::endl;
    }
```

MPI

```
export OMP_NUM_THREADS=1

mpirun -np 24 ./hello_world
```

```
0 0 1
3 0 1
5 0 1
7 0 1
10 0 1
20 0 1
14 0 1
...
```

MPI per node

```
export OMP_NUM_THREADS=12

mpirun --bind-to-core --bynode --npernode 1 ./hello_world
```

```
0 1 12
0 2 12
0 0 12
0 3 12
...
1 1 12
1 2 12
...
```

MPI per socket

```
export OMP_NUM_THREADS=6

mpirun --bind-to-socket --bysocket --npersocket 1 ./
hello_world
```

```
0 0 6
0 2 6
...
1 0 6
1 4 6
1 1 6
...
```