Scientific Computing III. High Performance Scientific Computing

(Phys 2109/Ast 3100H)

Lecture 8: Hybrid OpenMP/MPI Programming

SciNet HPC Consortium, University of Toronto

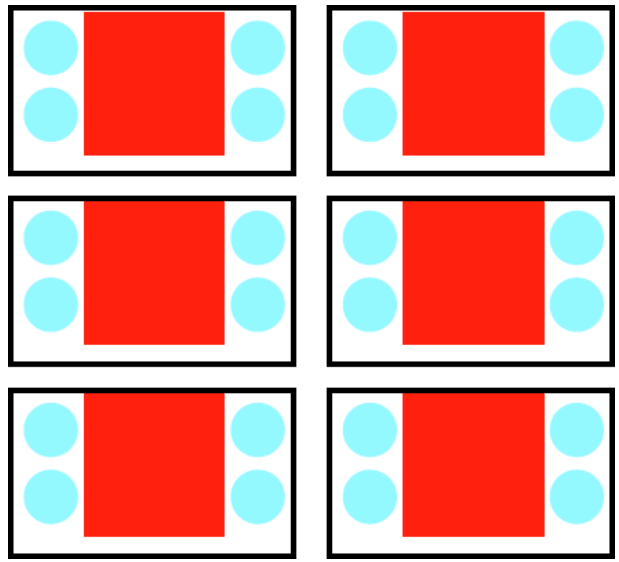
Winter 2013

SciNet compute • calcul CANADA

Shared and distributed memory

Modern clusters have a hybrid architecture.

- Multicore machines linked together with an interconnect
- Machines with GPU or other coprocessors: GPU is multi-core, but the amount of shared memory is limited.



MPI vs OpenMP

We have OpenMP for shared memory programming.

We have MPI to program distributed memory machines

model	memory	latency	mem.overhead	scalable	incremental
OpenMP	shared mem	low	low	limited	yes
MPI	distributed	high(er)	higher	yes	no

- Could we have the best of both worlds?

MPI and OpenMP

Hybrid programming model of using MPI and OpenMP:

- MPI across nodes
- OpenMP within nodes

- Minimizes communication
- Scalable
- Not much more complicated than pure MPI

Hybrid Programming

Pros

- No decomposition on node
- Lower latency, less communication
- Less duplication of data (and perhaps computation)
- OpenMP has load balance capabilities

Cons

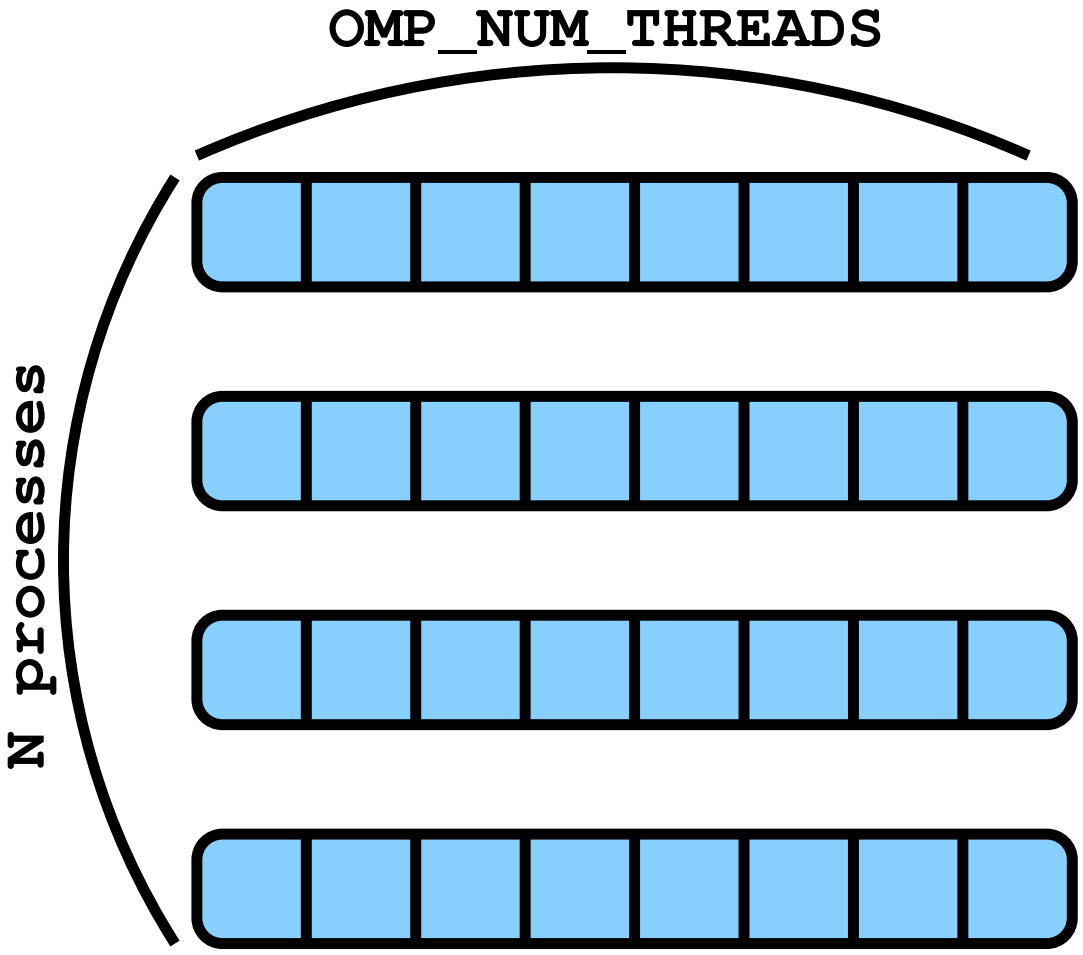
- One more layer to maintain
- OpenMP has more hidden side effects
- May have to worry about NUMA (later)

Hybrid Programming

Example

```
#include <mpi.h>
#include <omp.h>
#include <iostream>
int main(int argc, char ** argv)
{
  int size,rank;
  MPI_Init(&argc,&argv);
  MPI_Comm_get_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_get_size(MPI_COMM_WORLD,&size);
  #pragma omp parallel for
  for (int i=0;i<4;i++)
    std::cout << "Hello world from thread "
              << omp_get_thread_num() << std::endl;
  MPI_Finalize();
}
```

Hybrid Programming



- Memory shared among threads of same process
- Memory not shared among threads of different processes

Hybrid Programming

- Note: OpenMP inside MPI
- Often, one starts with an MPI code and adds in OpenMP.
- Compilation:

```
mpicxx -fopenmp [filename] -o [executable]
```

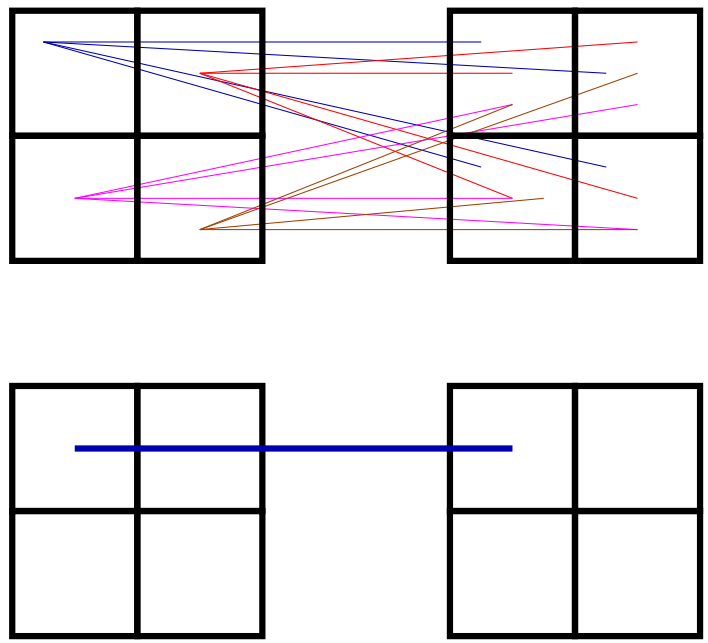
- Run:

```
export OMP_NUM_THREADS=M
mpirun -np N [executable]
```

- This starts N processes
- Between `MPI_Init` and `MPI_Finalize`, each process spawns OMP_NUM_THREADS threads in `#pragma omp parallel` blocks.

Thread Safety

- Some implementations are limited and do not have support for MPI calls within OpenMP parallel blocks
- Thus, may need to do MPI in serial regions
- Not necessarily bad:



Less communication channels
Bigger messages

MPI_Init_thread

. . . is an `MPI_Init` replacement that can check for thread support of the MPI implementation.

```
int MPI_Init_thread(int *argc, char ***argv,
                    int required,
                    int *provided);
```

`required` and `*provided` can take values:

`MPI_THREAD_SINGLE`	Only 1 thread will execute.
`MPI_THREAD_FUNNELED`	1 thread calls MPI.
`MPI_THREAD_SERIALIZED`	1 thread calls MPI at one time.
`MPI_THREAD_MULTIPLE`	Multiple threads may call MPI at once

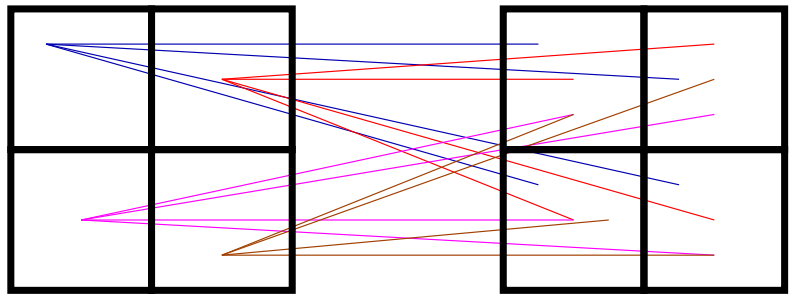
Hybrid Programming

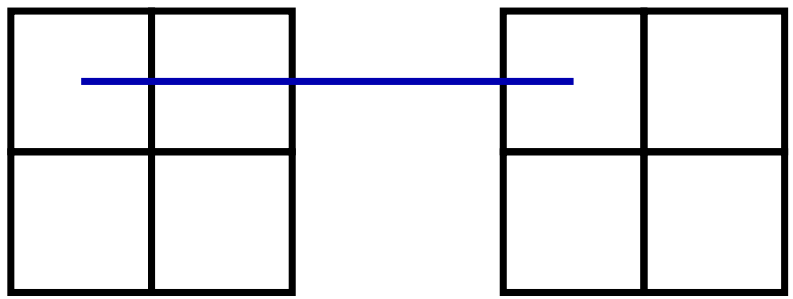
Example

```
#include <mpi.h>
#include <omp.h>
#include <iostream>
int main(int argc, char ** argv)
{
  int size,rank,thread;
  MPI_Init_thread(&argc,&argv,MPI_THREAD_FUNNELED,&thread);
  if (thread>=MPI_THREAD_FUNNELED) {
    MPI_Comm_get_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_get_size(MPI_COMM_WORLD,&size);
    #pragma omp parallel for
    for (int i=0;i<4;i++)
      std::cout << "Hello world from thread "
                << omp_get_thread_num() << std::endl;
  }
  MPI_Finalize();
}
```

Common useful cases

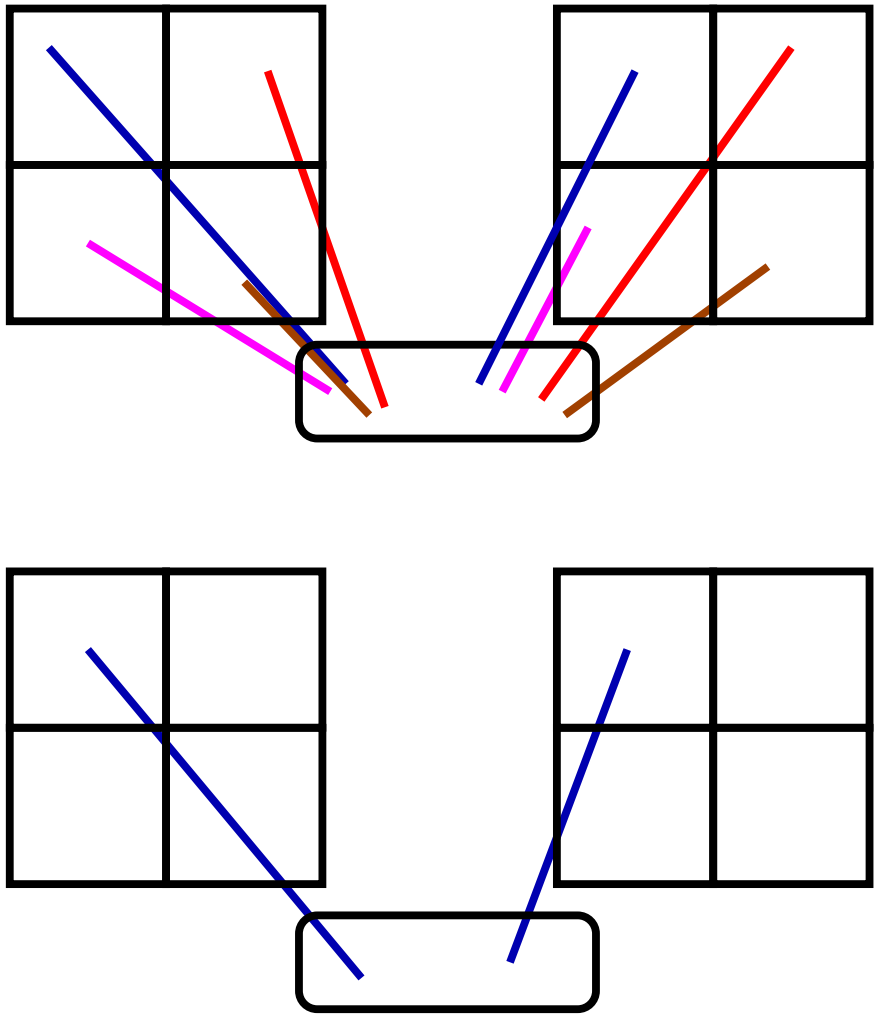
- Memory bound applications
  - each mpi process is a full application
  - openmp requires less memory
- To fit NUMA (later)
- Overlap comm/comp
  - 1 thread for communication
  - rest for work





Common useful cases

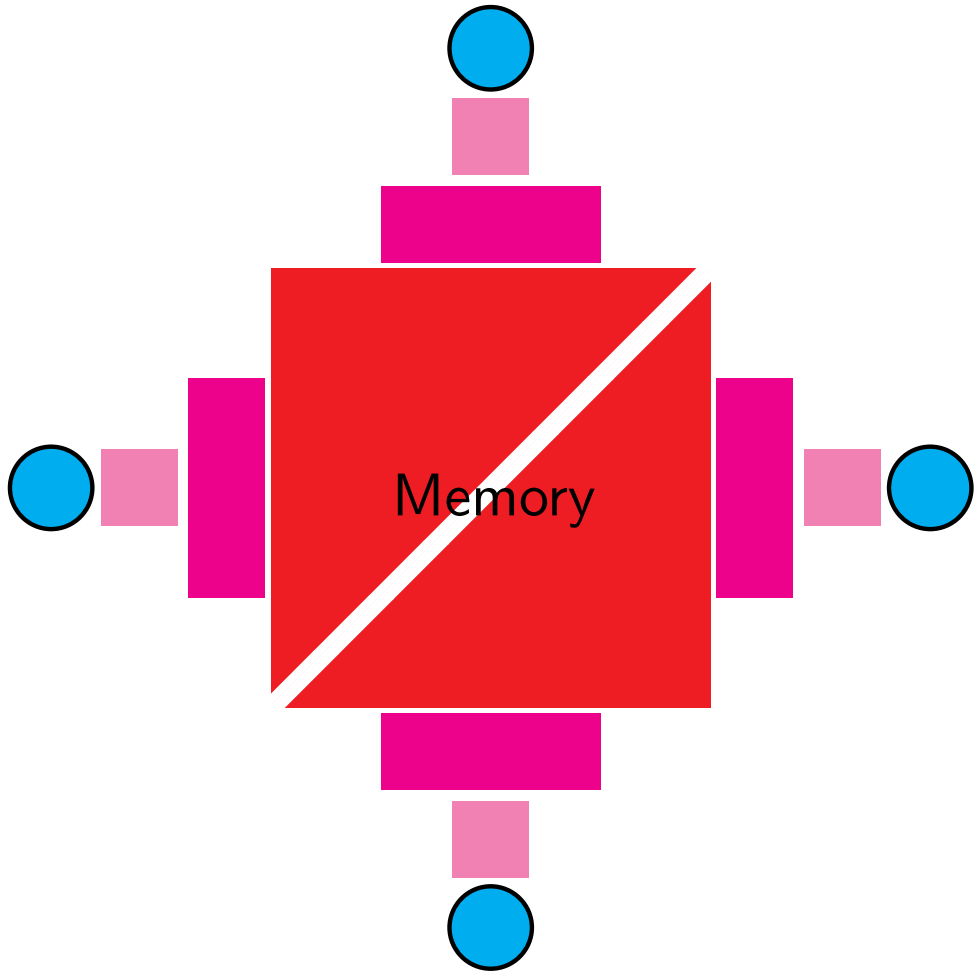
- Overlap IO/comp
  - 1 thread for IO
  - rest for work



Shared Memory: NUMA

NUMA = Non-Uniform Memory Access

- Multiple cores, but often multiple sockets.
- Each socket may have some memory nearby, but can also access the memory of the other socket, at a slower rate.
- Each core typically has some memory/cache of its own.
- Memory locality matters even on a node.



Affinity

Where do processes, threads and memory go?

- Operating system distributes threads and processes over cores, and may migrate them from one core to another.
- Typically, one would want the threads of a process to be on the same core (but not always).
- One would want the memory used by a thread to be close to the core on which it runs.
- In Linux, memory is not physically allocated until used.
- Memory is owned by the first thread that uses it: 'first access'.
- Data initialized in a serial section may be 'far' for some threads.

On most systems, these defaults make sense.

Tuning the process/thread affinity

Process/thread affinity:

- Command-line tools like `numactl`.
- Calls to `sched_setaffinity`.
- Flags to mpirun.
  E.g. OpenMPI has `-bind-to-core` and `-bind-to-socket`.
- `OMP_PROC_BIND=true` and implementation specific environment variables.

This does prevent the OS from load balancing.
Not an issue if you're using all resources of a node.

Tuning the memory affinity

- If a process does not get migrated by OS, memory will remain close to process.
- For threads, in coding, use thread-local variables if you can.

  (sometimes copying into a thread-local variable can help)
- When a part of the data is mainly used by a specific thread, initialize it in that thread (i.e. not in a serial section).

Homework

- Take the diffusion code of last week's homework and add OpenMP to the loops that you expect do most of the computational work.
- Analyze your code's scaling on a single GPC node, by timing 64 cases, varying both OMP_NUM_THREADS and the number of MPI processes from 1 to 8.
- Plot the result and explain what you see.
- Try this on a pair of GPC nodes as well. Let OMP_NUM_THREADS take values 1,2,4,8, while adjusting the number of mpi processes to 16/OMP_NUM_THREADS.
- Plot the timing results and explain what you see.
- Email code, makefile, git log, plots, together with the explanations in a file called explain.txt, by Tuesday April 23.

SciNet
compute • calcul
CANADA