Introduction to Parallel Programming and MPI

Paul Edmon
FAS Research Computing
Harvard University
Outline

• What is parallel computing?

• Theory

• Message Passing Interface
Parallel vs. Serial

- Serial: A logically sequential execution of steps. The result of next step depends on the previous step.

- Parallel: Steps can be contemporaneously and are not immediately interdependent or are mutually exclusive.
Microprocessor Transistor Counts 1971-2011 & Moore’s Law

The curve shows transistor count doubling every two years.
High Performance Computing (HPC)

- Goal: Leverage as much computer power as possible with as much efficiency as possible to solve problems that cannot be solve by conventional means

- Sub Types
  - Algorithm and Single Chip Efficiency
  - High Throughput Computing
  - High I/O Computing
  - Tightly Coupled Parallel Computing
Scaling

• **Weak Scaling**
  – Keep the size of the problem per core the same, but keep increasing the number of cores.
  – Ideal: Amount of time to solution should not change

• **Strong Scaling**
  – Keep the total size of the problem the same but keep increasing the number of cores.
  – Ideal: Time to completion should scale linearly with the number of cores

• **Reasons for Deviation**
  – Communications Latency
  – Blocking Communications
  – Non-overlapped communications and computation.
  – Not enough computational work
Amdahl’s Law

• The maximum you can speed up any code is limited by the amount that can be effectively parallelized.

• In other words: You are limited by the mandatory serial portions of your code.
Types of Parallelization

• SIMD

• Thread

• Multinode
SIMD

• Single Instruction Multiple Data

• Vectorization
  – \( A(:,)=B(:,)+C(:,) \)

• Processors natively do this, compilers optimize for it.
  – SSE (Streaming SIMD Extensions): 128 bit register, \( a=a+b \)
  – AVX (Advanced Vector Extensions): 128 bit register, \( a=a+b \rightarrow 256 \text{ bit register } a=b+c \)

• Note on Optimization Flags:
  – -O0: No optimization
  – -O1: Safe optimization
  – -O2: Mostly Safe optimization
  – -O3: Aggressive optimization

• Always check your answers after your optimize to make sure that you get the same answer back. This is true for any time you recompile or build on a new system. If there are differences make sure they are minor with respect to your expected code outcome.
Thread

- Single Node, program is broken up into threads

- Libraries: OpenMP, pThreads, Cilk

- SMP: Symmetric multiprocessing

- Threads have access to the same memory pool and thus do not have to communicate
Multinode

- Program is broken up into ranks, each rank runs a part of the code
- Ranks run on multiple nodes
- Ranks do not share memory so they must communicate with each to share information
- Libraries: MPI
Is my code parallelizable?

• Does it have large loops that repeat the same commands?

• Does your code do multiple tasks that are not dependent one another? If so is the dependency weak?

• Can any dependencies or information sharing be overlapped with computation? If not is the amount communications small?

• Do multiple tasks depend on the same data?

• Does the order of operations matter? If so how strict does it have to be?
Examples

• Computational Fluid Dynamics
• N-Body and NAMD
• Radiative Transfer and Image Processing
• Markov Chain Monte Carlo
• Embarrassingly Parallel Work
General Guidelines for Parallelization

• Is it even worth parallelizing my code?
  – Does your code take an intractably long amount of time to complete?
  – Do you run single large models or do statistics on multiple small runs?
  – Would the amount of time it take to parallelize your code be worth the gain in speed?

• Parallelizing Established Code vs. Starting from Scratch
  – Established Code: May be easier/faster to do, but may not give good performance or scaling
  – Start from Scratch: Takes longer but will give better performance, accuracy, and gives opportunity to turn a black box code into a code you understand

• Test, test, test, etc.

• Use Nonblocking Communications as often as possible

• Overlap Communications with Computation

• Limit synchronization barriers
General Guidelines for Parallelization

- Limit Collective Communications
- Make messages small
  - Only send essential information
- Make sure messages are well packaged
  - Do one large send with data in a buffer rather than multiple sends
- Use MPI_Iprobe to grease the wheels of nonblocking communications
- Always post nonblocking receives before sends
- Watch out for communications deadlocks
- Be careful of your memory overhead
- Be careful of I/O
  - Avoid having all the cores write to disk at once
  - Alternately don’t have all I/O go through one rank.
General Guidelines for Parallelization

• Do as much as is possible asynchronously

• See if some one has parallelized a code similar to yours and look at what they did

• Beware of portions of the code that depend on order of operations

• Avoid gratuitous IF statements

• Do not use GOTO unless absolutely necessary

• KISS: Keep it simple stupid.

• Print statements are your friend for debugging

• So is replicating the problem on a small number of ranks

• Think at scale
Message Passing Interface

- MPI standard: Set by MPI Forum
- Current full standard is MPI-2
  - MPI-3 is in the works which includes nonblocking collectives
- MPI allows the user to control passing data between processes through well defined subroutines
- API: C, C++, Fortran
- Libraries: C#, Java, Python, R
- MPI is “agnostic” about network architecture, all it cares is that the location that is being run on can be addressed by whatever transport method you are using
MPI Nomenclature

- **Rank**: The ID of a process, starts counting from 0
- **Handle**: The unique ID for the communication that is being done
- **Buffer**: An array or string, either controlled by the user or MPI, which is being transported
- **Core**: An individual compute element
- **Node**: A collection of compute elements that share the same network address, share memory, and are typically on the same main board
- **Hostfile**: The list of hosts you will be running on
- **MPI Fabric**: The communications network MPI constructs either by itself or using a daemon
- **Blocking**: Means the communications subroutine waits for the completion of the routine before moving on.
- **Collective**: All ranks talk to everyone else to solve some problem.
Available MPI Compilers on Odyssey

- **OpenMPI**
  - Open Source project
  - No daemon required
  - Supports MPI-2
  - Even releases are stable, odd releases are development

- **MVAPICH2**
  - Ohio State University project
  - Old versions require daemon, Latest version does not require daemon
  - MPI-2.2 support as well as some support for MPI-3

- **Intel MPI**
  - Version of MVAPICH2 optimized by Intel
  - Requires daemon

- All compile for C, C++ and Fortran
MPI Hello World (Fortran/C)

PROGRAM hello

    !!! Need to include this to be able to hook into the MPI API !!!
    INCLUDE 'mpif.h'

    INTEGER*4 :: numprocs, rank, ierr

    !!! Initializes MPI !!!
    CALL MPI_Init(ierr)

    !!! Figures out the number of processors I am asking for !!!
    CALL MPI_Comm_size(MPI_COMM_WORLD, numprocs, ierr)

    !!! Figures out which rank we are !!!
    CALL MPI_Comm_rank(MPI_COMM_WORLD, rank, ierr)

    write(*,*) 'Process', rank, 'out of', numprocs

    !!! Need this to shutdown MPI !!!
    CALL MPI_Finalize(ierr)

END PROGRAM hello

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

int main(int argc, char *argv[]) {
    int numprocs, rank;

    /* Initializes MPI */
    MPI_Init(&argc, &argv);

    /* Figures out the number of processors I am asking for */
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);

    /* Figures out which rank we are */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    printf("Process \%d out of \%d\n", rank, numprocs);

    /* Need this to shutdown MPI */
    MPI_Finalize();
}

Compiling and Running OpenMPI

```
[pedmon@itc011 ~]$ module load hpc/openmpi-intel-latest
Loading module hpc/openmpi-intel-latest.
[pedmon@itc011 ~]$ mpif90 hello.f90
/n/sw/intel_cluster_studio-2013/lib/intel64/libimf.so: warning: warning: feupdateenv is not implemented and will always fail
[pedmon@itc011 ~]$ cat hostfile
itc011 slots=8
itc012 slots=8
[pedmon@itc011 ~]$ mpirun --np 16 --hostfile hostfile ./a.out
Process  10 out of  16
Process  14 out of  16
Process   2 out of  16
Process   6 out of  16
Process   7 out of  16
Process   1 out of  16
Process   9 out of  16
Process   4 out of  16
Process  15 out of  16
Process   5 out of  16
Process   8 out of  16
Process   3 out of  16
Process  12 out of  16
Process  10 out of  16
Process  13 out of  16
Process  11 out of  16
Process   0 out of  16
[pedmon@itc011 ~]$ exit
```
Compiling and Running in other versions of MPI

- **MVAPICH2**: Same as OpenMPI but hostfile is different
  - OpenMPI: hostname slots=8
  - MVAPICH: hostname:8

- **Intel MPI**: Same as MVAPICH2 but you first need to start the daemon using the following line
  - mpdboot –f hostfile –n 2
  - mpirun –np 16 ./a.out
  - Where n in this is the number of nodes
Stay tuned

• Next presentation by Plamen will cover more complex topics such as:
  – MPI Collectives
  – Point to Point Communications
  – Asynchronous Communications
  – MPI and non-C and non-Fortran codes
  – I/O in Parallel Environments