Introduction to Parallel Programming

Martin Čuma
Center for High Performance Computing
University of Utah
mcuma@chpc.utah.edu
Overview

- Types of parallel computers.
- Parallel programming options.
- How to write parallel applications.
- How to compile.
- How to debug/profile.
- Summary, future expansion.
Parallel architectures

Single processor:
- SISD – single instruction single data.

Multiple processors:
- SIMD - single instruction multiple data.
- MIMD – multiple instruction multiple data.
  - Shared Memory
  - Distributed Memory
Shared memory

- All processors have access to local memory
- Simpler programming
- Concurrent memory access
- More specialized hardware
- CHPC: Linux clusters, 2, 4, 8 core nodes
Distributed memory

- Process has access only to its local memory
- Data between processes must be communicated
- More complex programming
- Cheap commodity hardware
- CHPC: Linux clusters (Arches, Updraft)

8 node cluster (64 cores)
Shared Memory

- Threads – POSIX Pthreads, OpenMP
  - Thread – own execution sequence but shares memory space with the original process
- Message passing – processes
  - Process – entity that executes a program – has its own memory space, execution sequence

Distributed Memory

- Message passing libraries
  - Vendor specific – non portable
  - General – MPI, PVM
OpenMP basics

• Compiler directives to parallelize
  ▪ Fortran – source code comments
    !$omp parallel !$omp end parallel
  ▪ C/C++ - #pragmas
    #pragma omp parallel
• Small set of subroutines
• Degree of parallelism specification
  ▪ OMP_NUM_THREADS or
    omp_set_num_threads(INTEGER n)
MPI Basics

• Communication library
• Language bindings:
  - C/C++ - `int MPI_Init(int argv, char* argc[])`
  - Fortran - `MPI_Init(INTEGER ierr)`
• Quite complex (100+ subroutines) but only small number used frequently
• User defined parallel distribution
MPI vs. OpenMP

- Complex to code
- Slow data communication
- Ported to many architectures
- Many tune-up options for parallel execution

- Easy to code
- Fast data exchange
- Memory access (thread safety)
- Limited usability
- Limited user’s influence on parallel execution
Program example

• saxpy – vector addition: \( \mathbf{z} = a \mathbf{x} + \mathbf{y} \)
• simple loop, no cross-dependence, easy to parallelize

```fortran
subroutine saxpy_serial(z, a, x, y, n)
  integer i, n
  real z(n), a, x(n), y(n)

  do i=1, n
    z(i) = a*x(i) + y(i)
  enddo
return
```

subroutine saxpy_parallel_omp(z, a, x, y, n)
integer i, n
real z(n), a, x(n), y(n)

 !$omp parallel do
do i=1, n
  z(i) = a*x(i) + y(i)
enddo
return

setenv OMP_NUM_THREADS 4
subroutine saxpy_parallel_mpi(z, a, x, y, n)
integer i, n, ierr, my_rank, nodes, i_st, i_end
real z(n), a, x(n), y(n)

call MPI_Init(ierr)
call MPI_Comm_rank(MPI_COMM_WORLD, my_rank, ierr)
call MPI_Comm_size(MPI_COMM_WORLD, nodes, ierr)
i_st = n/nodes*my_rank+1
i_end = n/nodes*(my_rank+1)

do i=i_st, i_end
   z(i) = a*x(i) + y(i)
enddo

z(i) operation on 4 processes

<table>
<thead>
<tr>
<th>z(1)</th>
<th>z(n/4+1)</th>
<th>z(2*n/4+1)</th>
<th>z(3*n/4+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>...n/4</td>
<td>...2*n/4</td>
<td>...3*n/4</td>
<td>...n</td>
</tr>
</tbody>
</table>

return
• **Result on the first CPU**

```fortran
include "mpif.h"
integer status(MPI_STATUS_SIZE)
if (my_rank .eq. 0 ) then
   do j = 1, nodes-1
      do i= n/nodes*j+1, n/nodes*(j+1)
         call MPI_Recv(z(i),1,MPI_REAL,j,0,MPI_COMM_WORLD,
            & status,ierr)
      enddo
   enddo
else
   do i=i_st, i_end
      call MPI_Send(z(i),1,MPI_REAL,0,0,MPI_COMM_WORLD,ierr)
   enddo
endif
```

<table>
<thead>
<tr>
<th>Data</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>N2</td>
</tr>
<tr>
<td>N3</td>
<td>N4</td>
</tr>
</tbody>
</table>

- **Sender**
- **Recipient**
• Collective communication

```fortran
real zi(n)
j = 1
do i=i_st, i_end
   zi(j) = a*x(i) + y(i)
   j = j + 1
enddo
call MPI_Gather(zi,n/nodes,MPI_REAL,z,n/nodes,MPI_REAL,
&                0,MPI_COMM_WORLD,ierr)
```

• Result on all nodes

```fortran
call MPI_AllGather(zi,n/nodes,MPI_REAL,z,n/nodes,
&                   MPI_REAL,MPI_COMM_WORLD,ierr)
```

![Diagram showing data distribution and communication between nodes]
First log into one of the clusters

- ssh delicatearch.chpc.utah.edu - Myrinet
- ssh marchingmen.chpc.utah.edu - Ethernet
- ssh tunnelarch.chpc.utah.edu - Ethernet
- ssh landscapearch.chpc.utah.edu – Ethernet, Myrinet
- ssh sanddunearch.chpc.utah.edu – Ethernet, InfiniBand
- ssh updraft.chpc.utah.edu – Ethernet, InfiniBand

Then submit a job to get compute nodes

qsub -I -l nodes=2:ppn=4,walltime=1:00:00
qsub script.pbs

Useful scheduler commands

qsub – submit a job
qdel – delete a job
showq – show job queue
• No clear text passwords use ssh and scp
• You may not share your account under any circumstances
• Don’t leave your terminal unattended while logged into your account
• Do not introduce classified or sensitive work onto CHPC systems
• Use a good password and protect it
• Do not try to break passwords, tamper with files etc.
• Do not distribute or copy privileged data or software
• Report suspicions to CHPC (security@chpc.utah.edu)
• Please see http://www.chpc.utah.edu/docs/policies/security.html for more details
Arches

- Supported by most compilers, `-openmp`, `-fopenmp` or `-mp` switch
  
  e.g. `pgf77 -mp source.f -o program.exe`

- Dual-processor and quad-processor (Sanddunearch) nodes

- Further references:
  Compilers man page – `man ifort`

Compilers websites

- [http://gcc.cnu.org](http://gcc.cnu.org)
- [http://www.pathscale.com/node/70](http://www.pathscale.com/node/70)
• Three common network interfaces
  – Ethernet, Myrinet, InfiniBand
• Different MPI implementations
  – MPICH – Ethernet, shmem
  – MPICH2 - Ethernet, shmem
  – OpenMPI – Ethernet, Myrinet, IB
  – MVAPICH, MVAPICH2, InfiniPath - InfiniBand
  – MPICH-MX, MPICH2-MX – Myrinet
  – Intel MPI, Scali MPI - commercial
• **Arches** – MPICH, MPICH2, MVAPICH, MVAPICH2

  \[ /\text{MPI-path/bin/mpixx source.x \ -o program.exe} \]

  - **MPI-path** = location of the distribution

    - /uufs/arches/sys/pkg/mpich/std TCP-IP
    - /uufs/$UUFSCELL/sys/pkg/mpich-mx/std Myrinet
    - /uufs/arches/sys/pkg/mpich2/std MPICH2 TCP-IP
    - /uufs/sanddunearch.arches/sys/pkg/mvapich/std MVAPICH InfiniBand
    - /uufs/sanddunearch.arches/sys/pkg/mvapich2/std MVAPICH2 InfiniBand
    - /usr/bin - InfiniPath MPI on Updraft

  = must specify full path to `mpixx` (/MPI-path/bin) or add this path to PATH environment variable
• **MPICH Interactive batch (incl. Updraft)**
  qsub -I -l nodes=2:ppn=2,walltime=1:00:00
  ... wait for prompt ...
  /MPI-path/bin/mpirun -np 4 -machinefile
  $PBS_NODEFILE program.exe - MPICH

  /MPI-path/bin/mpirun_rsh -rsh -np 4
  -machinefile $PBS_NODEFILE program.exe - MVAPICH

• **MPICH Batch**
  #PBS -l nodes=2:ppn=2,walltime=1:00:00

• **OpenMP Batch**
  #PBS -l nodes=1:ppn=2,walltime=1:00:00
  setenv OMP_NUM_THREADS 2
  program.exe
• MPICH2, MVAPICH2 Interactive batch
  qsub -I -l nodes=2:ppn=2,walltime=1:00:00
  ... wait for prompt ...

  mpdboot -n 2 -r /usr/bin/rsh -f $PBS_NODEFILE
  mpdtrace
    da001
    da002
  mpiexec -n 4 ./program.exe

• NOTE  – on sanddunearch, use  ppn=4
         – on updraft, use  ppn=8

http://www.chpc.utah.edu/docs/manuals/software/mpi.html
Debuggers

• Useful for finding bugs in programs
• Several free
  - gdb – GNU, text based, limited parallel
  - ddd – graphical frontend for gdb
• Commercial that come with compilers
  - pgdbg – PGI, graphical, parallel but not intuitive
  - pathdb, idb – Pathscale, Intel, text based
• Specialized commercial
  - totalview – graphical, parallel, CHPC has a license
  - ddt - Distributed Debugging Tool
• How to use:
  - http://www.chpc.utah.edu/docs/manuals/software/par_devel.html
• Parallel debugging more complex due to interaction between processes
• Totalview is the debugger of choice at CHPC
  ▪ Expensive but academia get discount
  ▪ How to run it:
    ▪ compile with \(-g\) flag
    ▪ automatic attachment to OpenMP threads
    ▪ extra flag \((-tv)\) to mpirun/mpiexec
  ▪ Details:
    http://www.chpc.utah.edu/docs/manuals/software/totalview.html

  ▪ Further information
    http://www.totalviewtech.com/alltvdocs.htm
Process view

Source code view

Data inspection

Debuggers – parallel
Profilers

- Measure performance of the code
- Serial profiling
  - discover inefficient programming
  - computer architecture slowdowns
  - compiler optimizations evaluation
  - gprof, pgprof, pathopt2
- Parallel profiling
  - target is inefficient communication
  - Intel Trace Collector and Analyzer
• Serial
  ▪ BLAS, LAPACK – linear algebra routines
  ▪ MKL, ACML – hardware vendor libraries

• Parallel
  ▪ ScaLAPACK, PETSc, NAG, FFTW

http://www.chpc.utah.edu/docs/manuals/software/mat_l.html
• Shared vs. Distributed memory
• OpenMP
  ▪ limited on Arches
  ▪ Simple parallelization
• MPI
  ▪ Arches
  ▪ Must use communication

http://www.chpc.utah.edu/docs/presentations/intro_par
References

- **OpenMP**
  
  http://www.openmp.org/
  
  Chandra, et. al. - Parallel Programming in OpenMP
  
  Chapman, Jost, van der Pas – Using OpenMP

- **MPI**
  
  http://www-unix.mcs.anl.gov/mpi/
  
  Pacheco - Parallel Programming with MPI
  
  Gropp, Lusk, Skjellum - Using MPI 1, 2
Future Presentations

- Introduction to MPI
- Introduction to OpenMP
- Debugging with Totalview
- Profiling with TAU/Vampir
- Intermediate MPI and MPI-IO
- Mathematical Libraries at the CHPC