Computational Resources, first steps, Parallel Programming etc
Outline

1. Introduction
2. Kraken Basics
3. HowTo and more
4. Parallel Programming Basics
5. MPI programming
6. Q&A
ORNL is the U.S. Department of Energy’s largest science and energy laboratory

National Institute for Computational Sciences (NCCS)

$70M Operating budget to deploy and operate the computational resources required to tackle global challenges

• $1.3B budget
• 4,250 employees
• 3,900 research guests annually
• $350 million invested in modernization

• World’s most powerful computing facility
• Nation’s largest concentration of open source materials research

• Nation’s most diverse energy portfolio
• The $1.4B Spallation Neutron Source in operation
• Managing the billion-dollar U.S. ITER project
NICS is a collaboration between UT and ORNL
Awarded the NSF Track 2B ($65M)
Phased deployment of Cray XT systems
Staffed with 25 FTEs, funding for 15 more
Total JICS funding ~$92M
Computational Sciences

- Computer Hardware Software
- Mathematics Algorithms Numerics
- Sciences Theories Verifications Predictions
**Need of Parallel Computer**

- Requirement of computational capacity depends on applications and formulations and what you want to achieve
- **Length Scale** - resolution of the dimension, e.g. number of grid points
- **Time Scale** - resolution of duration, e.g. number of time step

**2D problem:**
- grid points $100 \times 100 = 10000$ pts
- a vector of 10000 elements $\sim 80$ KB
- need 10 such vectors $\sim 800$ KB
- Steady State in seconds

**3D problem:**
- grid points $10000 \times 10000 \times 100 = 10e10$ pts
- $10e10$ unknowns $\sim 80$ GB
- need 10 such vectors $\sim 800$ GB MEMORY
- 100 years simulation !!

NEED MULTIPLE WORKERS and RESOURCES – PARALLEL COMPUTER
Parallel Computing

Division of work into smaller tasks
Multiple computers work on smaller tasks simultaneously

>> Reduce Wall Clock Time <<
Issues of Parallel Computing

• Pros :
  – decrease wallclock time
  – deliver huge amount of memory
  – Allow realistic simulation

• Cons :
  – Difficult to construct
  – Efficient parallel algorithm may need some thought
  – Cost of program development

KEYS:
1) **LOAD BALANCE** - same amount of work for every processor
2) **LOCALITY** - minimize communications among processors
3) **PORTABILITY** - work well on different platforms of computers
4) **SCALABILITY** - can solve larger problem efficiently
Parallel Computers

Shared Memory Systems (SMP) (SGI, HP, PC, IBM) (USE OPENMP)

- P
- P
- P

bus or switch

shared memory

Distributed Memory Systems (MPP) (IBM SP, Cray XT or PC Cluster) (USE MESSAGE PASSING)

- M
- M
- M

P

P

P

Communication Network
Simple Parallel Computer

Many computers connected by a COS interconnect
The Hardware Schematic

Schematic of the SSD PC Cluster

y2k.ssd.ornl.gov
128.219.23.248

Switch to Outside

y2k01.ssd.ornl.gov
128.219.37.25

Master Server

UPS

3COM SuperStack II 3300 10/100 24-Port Switch

2 Dells  5 Fat Workers  14 Thin Workers

NIS Private Domain Name
y2kssd3
192.168.3.XX
**Jaguar: World’s Most Powerful Computer**

www.nccs.gov

<table>
<thead>
<tr>
<th></th>
<th>jaguar XT4</th>
<th>jaguarpf XT5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak Performance</td>
<td>263.16 TFLOPS</td>
<td>2.33 PFLOPS</td>
</tr>
<tr>
<td>System Memory</td>
<td>61 TB</td>
<td>292 TB</td>
</tr>
<tr>
<td>Disk Space</td>
<td>750 TB</td>
<td>10,000 TB</td>
</tr>
<tr>
<td>Disk Bandwidth</td>
<td>44 GB/s</td>
<td>240 GB/s</td>
</tr>
<tr>
<td>Interconnect Bandwidth</td>
<td>157 TB/s</td>
<td>374 TB/s</td>
</tr>
</tbody>
</table>
Kraken: 1st Academic PetaFLOPS Computer

- Cray XT5 – 1.028 PetaFLOPS (Peak)
- 88 cabinets in 4 rows
- 8256 compute nodes (99,072 cores)
- Each node has 12 cores - 2.6 GHz AMD (Istanbul) Processor
- 16 GB RAM per node
- 129TB of compute memory
- Scratch disk space, with 2.4PB of usable space
- 22x16x24 3D torus topology interconnect
- www.nics.utk.edu
## TOP500 List - November 2009 (1-100)

*R_{max} and R_{peak} values are in TFlops. For more details about other fields, check the **TOP500** description.*

Power data in KW for entire system

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Computer/Year Vendor</th>
<th>Cores</th>
<th>R_{max}</th>
<th>R_{peak}</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Oak Ridge National Laboratory United States</td>
<td>Jaguar - Cray XT5-HE Opteron Six Core 2.6 GHz / 2009 Cray Inc.</td>
<td>224162</td>
<td>1759.00</td>
<td>2331.00</td>
<td>6950.60</td>
</tr>
<tr>
<td>2</td>
<td>DOE/NNSA/LANL United States</td>
<td>Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband / 2009 IBM</td>
<td>122400</td>
<td>1042.00</td>
<td>1375.78</td>
<td>2345.50</td>
</tr>
<tr>
<td>3</td>
<td>National Institute for Computational Sciences/University of Tennessee United States</td>
<td>Kraken XT5 - Cray XT5-HE Opteron Six Core 2.6 GHz / 2009 Cray Inc.</td>
<td>98928</td>
<td>831.70</td>
<td>1028.85</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Forschungszentrum Juelpich (FZJ) Germany</td>
<td>JUGENE - Blue Gene/P Solution / 2009 IBM</td>
<td>294912</td>
<td>825.50</td>
<td>1002.70</td>
<td>2268.00</td>
</tr>
</tbody>
</table>
HPL (High Performance Linpack): Solving $Ax = b$

http://wiki.math.msu.edu/index.php/Gaussian_Elimination

\[
\begin{align*}
2x_1 + 2x_2 + 2x_3 &= 1 \\
3x_1 + 4x_2 + 5x_3 &= 2 \\
4x_1 + 6x_2 + 7x_3 &= 3.
\end{align*}
\]
\[
A = \begin{bmatrix}
2 & 2 & 2 \\
3 & 4 & 5 \\
4 & 6 & 7
\end{bmatrix}, \quad b = \begin{bmatrix}
1 \\
1/2 \\
3
\end{bmatrix}
\]
\[
A = \begin{bmatrix}
2 & 2 & 2 \\
0 & 1 & 2 \\
4 & 6 & 7
\end{bmatrix}, \quad b = \begin{bmatrix}
1 \\
1/2 \\
3
\end{bmatrix}
\]
\[
A = \begin{bmatrix}
2 & 2 & 2 \\
0 & 1 & 2 \\
0 & 0 & -1
\end{bmatrix}, \quad b = \begin{bmatrix}
1 \\
1/2 \\
0
\end{bmatrix}
\]

\[
x_3 = 0, \quad x_2 = 1/2 - x_3 = 1/2, \quad 2x_1 + 2x_2 + 2x_3 = 1 \implies x_1 = 0.
\]

Total operation count for Gaussian elimination with backward substitution:

\[
\frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n.
\]
$HPL - Gaussian Elimination$

Ax = b

change A into A = L U

\[
\begin{array}{c}
\text{A} \\
= \\
\text{L} \\
\text{U}
\end{array}
\]

so \( LUx = b \)

first solve \( Ly = b \) by direct downward solve

then solve \( Ux = y \) by direct upward solve
**LAPACK (LU)**

- The inner loop consists of BLAS1 and one BLAS 2 operations.

\[
\begin{align*}
\text{for } i &= 1 \text{ to } n-1 \\
\text{for } j &= i+1 \text{ to } n \\
A(j,i) &= A(j,i) / A(i,i) \quad \text{<----- BLAS1 (to BLAS2)} \\
\text{for } k &= i+1 \text{ to } n \\
A(j,k) &= A(j,k) - A(j,i) \times A(i,k) \quad \text{<----BLAS2 (to BLAS3)}
\end{align*}
\]
2D Block Cyclic Distribution

- Consider an 8 x 8 system of linear equations using a 2D blocked cyclic data distribution
- Matrix A is first decomposed into 2x2 blocks starting at its upper left corner, bk=2.
- These blocks are then uniformly distributed across a 2x2 processor grid, nprow = npcol = 2.
- There are 4 processes in the 2D process grid, nbrow = nbcol = 2.
**jaguar Rating**: World Fastest Computer

- **FLOPS** – FLoating Point Operation Per Second
- **GFLOPS = 10^9 FLOPS** ; **TFLOPS = 10^12** ; **PFLOPS = 10^15**
- **FLOPS = (clock rate) x (floating point operation in one clock cycle)**
- **Peak Rate = (FLOPS in one CPU) x (no. of CPU)**
- **Cray XT5 one core AMD Opteron** :
  - Rpeak : (2.6 GHz) x (4) x (224162 cores) = 2331284 GFLOPS
  - Rmax : 1759000 GFLOPS \(\Rightarrow 75.4\%\) of peak

**jaguar: What does it do?**

- **Solve a very big system of equations**: \(Ax = b\) using a standard benchmark C program (HPL)
- **Nmax**: Size of \(A\) for HPL (Solve \(Ax=b\)) = 5474272
- **Total Memory needed** = \((Nmax) \times (Nmax) \times (8\text{ Bytes})\) = 239741 GB
- **Memory needed per core** = 1.07 GB
- **Elapse Time**: \(2(Nmax)(Nmax)(Nmax)/3/Rmax \sim = 13\text{ hrs}\)
Earthquake Simulation: “The Big One”

- Performed earthquake simulation on the San Andreas Fault using 96,000 processor cores
Biomedical Simulations

- Performed largest simulation to date with solution of 4 billion degrees of freedom
- Numerical simulations to study fundamental aspects of blood flow including platelets aggregation, malaria-infected red blood cells, and aneurysms
Climate Simulations and Weather (Storms) forecast

SPC4-EF_00 ARW (1000x760x50, dx=4 km)
30 h WRF/ARW Forecast valid 06Z Sat 06 Jun 2009

Composite Ref (dBZ, Shaded)
U-V (m/s, Vector) Umin=-15.47 Umax=20.85 Vmin=-15.59 Vmax=17.77

www.caps.ou.edu
Genome and Protein Simulations

indinavir (Crixivan) natural substrate
1. Kraken Basics
Kraken System Configuration

- Cray XT5 running CNL 2.2.41
- 88 cabinets in 4 rows
- 8256 compute nodes (99,072 cores) & 96 service nodes
- 129TB of compute memory
- Two file systems available
- NFS mounted home areas, 2TB
- Lustre Scratch space, with 2.4PB of usable space
- 22x16x24 3D torus topology interconnect using SeaStar2 chips
**Compute node configuration**

- Two 2.6 Ghz Six-Core AMD (Istanbul) Processors
- Dual socket – 12 cores per node
- 16GB RAM per node
- Diskless nodes
- The ONLY accessible file system is Lustre scratch
- Runs a streamlined version of Linux-like OS called CLE
- Users cannot login to the compute nodes
- You need `qsub` & `aprun` to launch jobs in these nodes
- TORQUE/MOAB & ALPS control these resources
Service node configuration

- One 2.6 Ghz Dual-Core AMD Processors
- One socket – 2 cores per node
- 8GB RAM per node
- Diskless nodes
- Both NFS home areas & Lustre scratch accessible
- Runs a complete Linux-like OS called SLES10
- There are 16 login nodes
- 11 OTP only + 4 GSISSH only + 1 Experimental
- 4 GridFTP only with 10GigE internet connection
- 16 Aprun nodes & 48 I/O nodes
About the XT5
Cray XT5 Cooling

1. Fan pushes air into the rack
2. Inlet evaporator cools air
3. Air traverses through racks/blades
4. Exit evaporators cool down air that goes back to the computer room

Refrigerant R134a pipes

Cray ECOphelex

single axial turbofan
2. HowTo
What do you get with your account

- A Unix account (userid and Project account)
- A One Time Password generator (token) to login via ssh
- Access through Globus Grid tools (gssissh, GridFTP)
- A NFS home area (default 2GB quota)
- Lustre scratch space (<2.4PT)
- HPSS mass storage archival via hsi/htar (OTP only)
- >100 applications ready to run on Kraken
- Up to 99,072 cores
- User Assistance
- Bash as default Unix shell
Getting started: Accounts + Connecting

- Request an account
  Jaguar: http://www.nccs.gov/user-support/access/
  Kraken: http://www.nics.tennessee.edu/user-support/request-an-account
- Use Secure Shell (with passcode or password)
  - `ssh username@kraken.nics.utk.edu`

One time password (OTP)
- RSA SecurID (aka “fob” or “dongle”, or “thingamajig”)
- PASSCODE = PIN + Token Code
- SSH client need keyboard-interactive
HowTo login

Via SSH, BBCP using OTP

% ssh userid@kraken.nics.tennessee.edu
Enter PASSCODE:

PASSCODE = PIN + TokenCode

Via GSISSH (Teragrid portal)

login3$ myproxy-logon
Enter MyProxy pass phrase:
A credential has been received for user userid in /tmp/x509up_u974.
login3$ gsissh kraken-gsi.nics.tennessee.edu
userid@kraken-pwd4(XT5):~>

Via GridFTP, UBERFTP

gsiftp://gridftp.nics.utk.edu:2811
HowTo login – Teragrid style

Via the TeraGrid Portal

Via GSISSH

```
login3$ myproxy-logon
Enter MyProxy pass phrase:
A credential has been received for user userid in /tmp/x509up_u974.
login3$ gsissh kraken-gsi.nics.tennessee.edu
userid@kraken-pwd4(XT5):~>
```

Via GridFTP, UBERFTP

```
gsiftp://gridftp.nics.utk.edu:2811
```
HowTo compile

- Available C, C++, and Fortran compilers: PGI, GNU, Pathscale
- Use the Cray compiler wrappers `cc`, `CC`, and `ftn`, to compile programs for the compute nodes.
- The compiler wrappers know where most of the correct Cray provided libraries and include files are, if the corresponding module is loaded.
- You do not need to know where the MPI libraries are.
- The wrappers automatically add the correct tuning parameters for the Istanbul Processor.
- Use module help `<name>` to learn what you need to manually add for 3rd party modules.
**HowTo compile**

This example shows that a user needs to add `${SUPER_LU}` to the compile line

```bash
lucio@krakenpf2(XT5):~> module help superlu
----------- Module Specific Help for 'superlu/4.0' ----------------
Sets up environment to use parallel SUPERLU 4.0.
Usage:   ftn test.f90 ${SUPERLU_LIB}
or   cc test.c ${SUPERLU_LIB}
```

Example of what the wrappers do for you

```
/sw/altd/bin/ld /usr/lib64/crt1.o /usr/lib64/crti.o /opt/pgi/9.0.3/linux86-64/9.0-3/lib/trace_init.o
/usr/lib64/gcc/x86_64-suse-linux/4.1.2/crtbeginT.o -m elf_x86_64 -dynamic-linker /lib64/ld-linux-x86-64.so.2
-L/opt/mpt/3.5.0/xt/sma/lib -L/opt/mpt/3.5.0/xt/util/lib -L/opt/xt/mpt/3.5.0/xt/pmi/lib -L/opt/xt-pe/2.2.41A/lib
-L/usr/lib64/gcc/x86_64-suse-linux/4.1.2 -Bstatic -rpath=/opt/xt-pe/2.2.41A/lib -rpath=/opt/mpt/3.5.0/xt/mpich2-pgi/lib
-rpath=/opt/xt-libsci/10.3.9/pgi/lib -rpath=/opt/xt-sma/lib -rpath=/opt/xt/util/lib -rpath=/opt/xt/pmi/lib
-lnspgc -lpgc -lrt -lpthread -lm -lgcc -lgcc_eh -lc -lgcc -lgcc_eh -lc /usr/lib64/gcc/x86_64-suse-linux/4.1.2/crtend.o
/usr/lib64/crtin.o
```
Simple Hello World C Program

/* Simple serial Hello World C Example : hello.c */
#include <stdio.h>

int main (int argc, char *argv[])
{
    int rank, size;
    printf( "Hello world from process %d of %d\n", rank, size );
    return 0;
}

/* Parallel Hello World C Example : hello.c */
#include <stdio.h>
#include <mpi.h>

int main (int argc, char *argv[])
{
    int rank, size;
    MPI_Init (&argc, &argv); /* starts MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank); /* get current process id */
    MPI_Comm_size (MPI_COMM_WORLD, &size);  /* get number of processes */
    printf( "Hello world from process %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}

To compile : > cc –o hexe ./hello.c
To run in a computer : > ./hexe
To compile : > cc –o hexe ./hello.c
To run in parallel : > aprun –np 4 ./hexe
Compile and run on a CRAY XT

---compile your code---
kraken> cc hello.c -o hello
kraken> ftn hello.f -o hello

---submit your job---
kraken> qsub myjob.pbs
84628.nid00016

---edit a job script---
## myjob.pbs   helloworld job script
#!/bin/bash
#PBS -A UT-TNEDU002
#PBS -N test
#PBS -j oe
#PBS -l walltime=1:00:00, size=24
cd $PBS_O_WORKDIR
date
aprun -n 4 –N2 –S1 ./hello

---check status of your job---
kraken> showq | grep halloy

---view all cabinets and jobs---
kraken> xtshowcabs

output:
Fri Apr 09 20:12:06 EDT 2010
Hello World
Hello World
Hello World
Hello World
OPENMP Hello World – Fortran Code

C AUTHOR: Blaise Barney 5/99
C http://www.llnl.gov/computing/tutorials/openMP/exercise.html
PROGRAM HELLO
INTEGER  NTHREADS, TID, OMP_GET_NUM_THREADS,
INTEGER  OMP_GET_THREAD_NUM
C   Fork a team of threads giving them their own copies of variables
!$OMP PARALLEL PRIVATE(NTHREADS, TID)
C   Obtain thread number
   TID = OMP_GET_THREAD_NUM_NUM()
   PRINT *, 'Hello World from thread = ', TID
C   Only master thread does this
   IF (TID .EQ. 0) THEN
      NTHREADS = OMP_GET_NUM_THREADS()
      PRINT *, 'Number of threads = ', NTHREADS
   END IF
C   All threads join master thread and disband
!$OMP END PARALLEL
END
OPENMP Hello World – C code

/* AUTHOR: Blaise Barney 5/99 */
/* http://www.llnl.gov/computing/tutorials/openMP/exercise.html */
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {
    int nthreads, tid;
    /* Fork a team of threads giving them their own copies of variables */
    #pragma omp parallel private(nthreads, tid) {
        /* Obtain thread number */
        tid = omp_get_thread_num();
        printf("Hello World from thread = \%d\n", tid);
        /* Only master thread does this */
        if (tid == 0) {
            nthreads = omp_get_num_threads();
            printf("Number of threads = \%d\n", nthreads);
        }
    }
    /* All threads join master thread and disband */
}

Compile and run on the CRAY XT

```
%cc -mp -fast omp_hello.c -o hello
%ftn -mp -fast omp_hello.f -o hello
%qsub submit.pbs
```

```
#!/bin/bash
#PBS -A UT-TNEDU002
#PBS -N test
#PBS -j oe
#PBS -l walltime=1:00:00,size=12

cd $PBS_O_WORKDIR
Date
export OMP_NUM_THREADS=4
aprun -n 1 -S1 -d4 ./hello
```

Hello World from thread = 0
Number of threads = 4
Hello World from thread = 3
Hello World from thread = 1
Hello World from thread = 2
HowTo use modules

• All software/packages are managed via modules
• This allows environment variables, libraries, include paths to be cleanly entered and/or removed from your software environment.
• Conflicts are detected and loads that would cause conflicts are not allowed
• There are a number of basic modules loaded by default

1) modules/3.1.6.5
2) torque/2.4.1b1
3) moab/5.2.5.s12399
4) /opt/cray/xt-asyncpe/default/modulefiles/xtpe-istanbul
5) tusage/3.0-r2
6) DefApps
7) cray/MySQL/5.0.64-1.0000.2342.16.1
8) xtpe-target-cnl
9) xt-service/2.2.41A
10) xt-os/2.2.41A
11) xt-boot/2.2.41A
12) xt-lustre-ss/2.2.41A_1.6.5
13) cray/job/1.5.5-0.1_2.0202.18632.46.1
14) cray/csa/3.0.0-1_2.0202.18623.63.1
15) cray/account/1.0.0-2.0202.18612.42.3
16) cray/projdb/1.0.0-1.0202.18638.45.1
17) Base-opts/2.2.41A
18) pgi/9.0.3
19) totalview-support/1.6.3
20) xt-totalview/8.4.1b
21) xt-libsci/10.3.9
22) xt-mpt/3.5.0
23) xt-pe/2.2.41A
24) xt-asyncpe/3.3
25) PrgEnv-pgi/2.2.41A
26) /sw/altd/modulefiles/altd
HowTo use modules

The complete list of all available modules can be viewed with the command `module avail`. The 3rd party list of software can also be viewed from our website at:

http://www.nics.tennessee.edu/user-support/software/Kraken

<table>
<thead>
<tr>
<th>Loading commands</th>
<th>Informational commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>module [load</td>
<td>unload] &lt;my_module&gt;</td>
</tr>
<tr>
<td>Loads/unloads module</td>
<td>Lists available commands and usage</td>
</tr>
<tr>
<td>module swap &lt;module1&gt;&lt;module2&gt;</td>
<td>module show &lt;my_module&gt;</td>
</tr>
<tr>
<td>Replaces &lt;module1&gt; with &lt;module2&gt;</td>
<td>Displays the actions upon loading the module &lt;my_module&gt;</td>
</tr>
</tbody>
</table>

> module swap PrgEnv-pgi PrgEnv-gnu

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>module list</td>
<td>Displays all currently loaded modules</td>
</tr>
<tr>
<td>module avail &lt;name&gt;</td>
<td>Lists all modules (beginning with name)</td>
</tr>
</tbody>
</table>
How to compile

MPI Hello World example with another compiler

> module swap PrgEnv-pgi PrgEnv-gnu
> cc -o hello hello.c

MPI Hello World example with an older compiler version

> module swap pgi/9.0.3 pgi/7.2.5
> module swap xtpe-istanbul xtpe-barcelona
> cc -o hello hello.c

What compiler flags to use?

-fast -Mipa=fast -Minfo=all -Mneginfo=all
HowTo run an Interactive job
Sprng example

$ cd/lustre/scratch/lucio/sprngExample
$ showusage
Usage on kraken

<table>
<thead>
<tr>
<th>Project</th>
<th>StartDateEndDate</th>
<th>Allocation</th>
<th>Remaining</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>TG-STA080002N</td>
<td>2009-06-01 2010-06-01</td>
<td>200000.00</td>
<td>22827.92</td>
<td>177172.08</td>
</tr>
<tr>
<td>UT-ACCEPT</td>
<td>2008-09-01 2010-09-30</td>
<td>7000000.00</td>
<td>1259246.04</td>
<td>5740753.96</td>
</tr>
<tr>
<td>UT-SUPPORT</td>
<td>2009-03-01 2010-09-30</td>
<td>3000000.00</td>
<td>924565.75</td>
<td>2075434.25</td>
</tr>
</tbody>
</table>

NOTE - For more detailed job charges, use 'glsjob -u {username}

$ qsub -I -A UT-SUPPORT -lwalltime=00:05:00,size=12
qsub: waiting for job 504380.nid00016 to start
qsub: job 504380.nid00016 ready
$
HowTo run an Interactive job
Sprng example (cont)

$ pwd
/nics/a/home/lucio

$ cd/lustre/scratch/lucio/sprngExample/

$ aprun -n 8 ./sprng_mpi

Available generators; use corresponding numeral:
lfg     --- 0
lcg     --- 1
lcg64    --- 2
cmrg    --- 3
mlfg    --- 4
pmlcg   --- 5

3

Type in a generator type (integers: 0,1,2,3,4,5):

Process 7, random number 3: 0.01232133043670

Process 0, print information about stream:
Application 2041147 resources: utime 0, stime 0
**HowTo use Lustre**

Interesting facts:
- Kraken has 1.73PB of data with ~300M files
- Jaguar has 1.43PB of data with ~201M files

Configuration:
- 2.4PB of total space
- 48 OSS servers
- **7 OST per OSS (336 OSTs total)**
- Peak sustained bandwidth: ~30GB/s
- Defaults: Stripe count 4, Stripe size 1MB
- Location: /lustre/scratch(userid)
**HowTo use Lustre**

Best practices:

- Change your default stripe count to one! Specially if doing one file per process.
- Use stripe count of more than one only when needed.
- Use single/multiple shared files, and stripe counts multiple of 48 to get the best bandwidth
- Avoid using in Lustre: `ls -lt`
- If you need to monitor the progress, you want to use instead something like: `ls -t1 $destination | head`
- Learn to use the `lfs` command
- Visit our I/O page for more information

http://www.nics.tennessee.edu/io-tips
**HowTo debug and profile**

The following tools are available on Kraken for debugging, profiling and analysis parallel programs

<table>
<thead>
<tr>
<th>Debugging</th>
<th>Profiling and Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Totalview, lgdb, atp, gdb, pgdbg</td>
<td>Cray PAT, pgprof, TAU, FPMDI, PAPI, Scalasca</td>
</tr>
</tbody>
</table>

Always check the compatibility of the compiler options you want to use. For example, the following PGI compiler options are not supported:

-Mprof=mpi, -Mmpi, and -Mscalapack
HowTo transfer & archive

There are five ways to transfer files to/from Kraken:

- **globus-url-copy (GridFTP)**: The fastest way to transfer files to/from other (TG) systems. Can only be used with Kraken’s GridFTP nodes. Requires GSI authentication. Use gsiftp://gridftp.nics.utk.edu:2811

- **Uberftp**: Convenient FTP/SFTP like client, that uses the GridFTP protocol. Same requirements as ‘globus-url-copy’.

- **BBCP**: Yields much better performance than standard scp. Recommended if GridFTP is not available. *The right transfer parameters are critical for getting the best transfer rates!*

- **SCP (HPN)**: A modified version of scp that uses dynamic flow control buffers which yields better transfer rates than the vanilla version that comes with OpenSSH. Available at all OTP/GSI nodes.

- **HSI/HTAR**: Used to archive files and extract to/from the mass storage HPSS system. Only available at the OTP login nodes. *Highly desirable to bundle files together when archiving files!***
**NICS survival kit**

This is a list of Unix commands available on Kraken that all users should be aware of:

- `module <command>`
  All software packages like compilers, libraries and applications, are handled via modules

- `qsub<jobscript>`
  All jobs are submitted with the `qsub` command

- `aprun --{n|N|S|d|cc}`
  Programs get executed on the compute nodes with this command

- `showq --noblock [-r]`
  Shows the state of the queue
**NICS survival kit**

qstat\(<jid>\) Shows the status of a job with job id \(<jid>\)

glsjob\(<jid>\) It can be used to query information about a previous job or all previous jobs with -u\(<uid>\)

showstart\(<jid>\) Shows approximate start time for job with job id \(<jid>\)

showusage Displays the current balance of all the project accounts on Kraken a user has access to

showbf Shows what resources are available for “immediate” use.

Other commands include: checkjob, apstat, glsuser, glsproject
**NICS survival kit**

A better/faster way to work with files in Lustre, can be done with the help of the `lfs` instead of the standard Unix commands: `ls`, `find`, `df`.

---

**lfs<command> [options]**

- **setstripe/getstripe**  Used to manipulate the striping of files and directories in Lustre

- **find**  A much faster way to find files in Lustre. Example:
  
  ```
  lfs find /lustre/scratch/challoy --name *.c
  ```

- **df**  Shows how much space is left in Lustre

- **quota**  Shows how much space I am using in Lustre. Example:
  
  ```
  lfs quota -u challoy/lustre/scratch | sed -n 3p
  ```
**Important Policies**

- No production jobs should be run on the login nodes
- Large core count (i.e. capability) jobs have more priority
- Dedicated mode of the whole system is possible on Wednesdays
- Jobs using an account with negative balance will run only as backfill jobs
- Refunds can be provided for jobs that failed due to a system failure
- When Lustre gets 70% full we contact users to ask them to delete files. When 80% full, we will start deleting oldest files as an emergency procedure
More information

Cray offers most of their documentation online at
http://docs.cray.com/

Two excellent documents for new users are:
• Cray XT System Overview (S-2423-22)
• Cray XT Programming Environment User’s Guide (S-2396-22)
• Using Cray Performance Analysis Tools (S-2376-50)

Kraken’s User Guide developed by:
http://www.nics.tennessee.edu/computing-resources/kraken/user-guide

Cray XT5 workshop slides
http://www.nersc.gov/projects/workshops/CrayXT/agenda.php
Other NICS HPC resources

For more information on other NICS HPC resources, please visit

http://www.nics.tennessee.edu/computing-resources
How to get help

Send your questions via email to

help@teragrid.org

Or contact us by phone

1.865.241.1504

or

to the TG helpdesk
1.866.907.2383 (off hours)
Parallel Programming Basics
Parallel Programming Example: Pi

- Use numerical integration to compute Pi
- Let \( f(x) = \frac{4}{1+x^2} \) then integrate \( f(x) \) from \( x = 0 \) to \( 1 \)
- Using the rectangle rule

\[
R_n(f) = h \sum_{i=1}^{n} f(x_i)
\]

where \( n = \) the number of intervals, \( h = 1/n \) is the rectangle width and \( x_i = h.(n-0.5) \) is the midpoint of each rectangle

\[\text{Pi} = \text{area under } f(x)\]
**Pi Using Rectangles**

- Method: Divide area under curve into rectangles and distribute the rectangles to the processors.
- Suppose there are 3 processors, how should the distribution be done?
Parallel Performance Measure

- Using multiple processors you hope your program will go faster
- Observed Speedup using \( N \) processors to accomplish a task

\[
\text{Speedup} = \frac{T(1)}{T(N)} = \frac{\text{Time taken using 1 processor}}{\text{Time taken using } N \text{ processors}}
\]

- To be fair, should use the “best” serial algorithm on 1 processor, not the parallel algorithm, simply restricted to 1 processor
- Linear speedup:
  - Two processors take 1/2 the time of 1 processor, so speedup = 2
  - \( N \) processors take 1/N the time of 1 processor, so speedup = \( N \)
- Superlinear speedup
  - May be obtained occasionally, usually due to cache and memory improvements

"Parallel Programming with MPI" 61
**Amdahl’s Law**

- Maximum speedup is limited by the serial fraction of a program
- Serial code

  - Time taken: 100

  ![Parallelizable vs Serial](image)

- Parallel code (using num proc $P >> 10$)

  - Time taken =10, maximum speedup=10, regardless of $P$

![Speedup vs Number of Processors](image)
Performance Evaluation of Codes

• Profiling of codes
  – Use profiling tools, e.g. prof, to identify segment of code that requires intensive computing
  – craypat on the XT5

• Timing codes
  – timers depend on machines platform
  – $ time executable ---> cpu time, elapsed time
  – etime, dtime, ctime, itime
  – should provide good timing resolution for section of codes

• MPI timer
  – MPI_Wtime() ---> return seconds of elapse wall-clock time
  – MPI_Wtick() ----> return value of seconds between successive clock ticks
Message Passing Interface

Topics:
• Message Passing Interface (MPI)
• MPI Point to Point Communication
• Collective Communication
• Derived Datatype
• Parallel Code Development
**Message Passing Interface (MPI)**

- A first portable message passing communications standard defined by the MPI Forum which consists of hardware vendors, researchers, academics, software developers, and users, representing over forty different organizations

- The syntax of MPI is standardized
- The functional behavior of MPI calls is standardized
- Vendors have tuned MPI calls to suit the underlying hardware and software, hence efficiency of programs are generally preserved.

- IBM MPI: IBM implementation for the SP
- MPICH, LAM.. Popular implementations
- Cray MPT, OpenMPI ….
Message Passing Programming

- Used primarily on distributed memory computing environment
- Since memory are local, any data stored in remote processor’s memory must be explicitly requested by programmer.
- Each processor runs the SAME program using partitioned data set
- Written in sequential language (FORTRAN, C, C++) + MPI functions

```
Sum.exe (P0) :
  myid = 0, N = 10
  Data A, B (1:10)
  Do I=1,N
    AL(1:10) = A(myid*N+(1:10))
    BL(1:10) = B(myid*N+(1:10))
    CL = AL + BL
  SUM CL (MPI_Allreduce, C)
  Print global_sum C
```

```
Sum.exe (P1) :
  myid = 1, N = 10
  Data A, B(11:20)
  Do I=1,N
    AL(1:10) = A(myid*N+(1:10))
    BL(1:10) = B(myid*N+(1:10))
    CL = AL + BL
  SUM CL (MPI_Allreduce, C)
  Print global_sum C
```
MPI Message Components

- Envelope:
  - sending processor (processor_id)
  - source location (group_id, tag)
  - receiving processor (processor_id)
  - destination location (group_id, tag)

- Data (letter):
  - data type (integer, float, complex, char….)
  - data length (buffer size, count, strides)
Typical Message Passing Subroutines

• Environment Identifiers
  – processor_id, group_id, initialization

• Point to Point Communications
  – blocking operations
  – non-blocking operations

• Collective Communications
  – barriers
  – broadcast
  – reduction operations
Essentials of MPI programs

- **Header file:**
  - C : #include<mpi.h>
  - Fortran : include ‘mpif.h’

- **Initializing MPI:**
  - C : int MPI_Init(int argc, char**argv)
  - Fortran : call MPI_INIT(IERROR)

- **Exiting MPI:**
  - C : int MPI_Finalize()
  - Fortran : call MPI_FINALIZE(IERROR)
MPI Function Format

MPI

- MPI_Xxxx(parameter, ……..)

- C:
  - error = MPI_Xxxxx(parameter, ……..)
  - Case is IMPORTANT

- Fortran:
  - call MPI_XXXXX(parameter,……, IERROR)
  - Case is NOT important
MPI Process Identifiers

• **MPI_COMM_RANK:**
  – Gets a process’s rank (ID) within a process group
  – C : int MPI_Comm_rank(MPI_Comm comm, int *rank)
  – F : call MPI_COMM_RANK(mpi_comm,rank,ierror)

• **MPI_COMM_SIZE:**
  – Gets the number of processes within a process group
  – C : int MPI_Comm_size(MPI_Comm comm, int *size)
  – F : call MPI_COMM_SIZE(mpi_comm,size,ierror)

• **MPI_Comm** : Communicator
MPI Communicator

- A communicator is the communicating space among processes
- All MPI communication calls require a communicator argument and MPI processes can only communicate if they share a communicator.
- Every communicator contains a group of tasks with a system supplied identifier (for message context) as an extra match field.
- MPI_Init initializes all tasks with MPI_COMM_WORLD
- So, the base group is the group that contains all processes, which is associated with the MPI_COMM_WORLD communicator.
- Communicators are particularly important for user supplied libraries
- Communicators are used to create independent “message universe”
**Communicating Group**

ALL : MPI_COMM_WORLD (world)
ROW : comm1, comm2 ; COLUMN : comm3, comm4, comm5
Every process has three communicating groups and a distinct rank associated to it
Sample Program : Hello World

```c
#include <stdio.h>
#include "mpi.h"
main(int argc, char** argv) {
    int my_PE_num;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_PE_num);
    printf("hello from %d. \n", my_PE_num);
    MPI_Finalize();
}
```

```fortran
program Hello_World
  include ‘mpif.h’
  integer my_PE_num, ierror
  call MPI_INIT(ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD,my_PE_num, ierror)
  print *, ‘Hello from’, my_PE_num
  call MPI_FINALIZE(ierr)
end
```
Compiling and Running MPI (MPICH PC Cluster)

- Compile:
  - `mpicc -o fileexe filename.c`
  - `mpif77 -o fileexe filename.f`
- Running:
  - `mpirun -np <num_of_processes> fileexe`
- Options:
  - `-np : number of processes`
  - `-machinefile : list of machines to be used`
  - `-t : testing option`
  - use “man mpirun” to see more details

```
%mpirun –np 3 –machinefile hostfile hello.exe
hello from 0
hello from 2
hello from 1
```
Compiling and Running (Cray XT)

• Compile: `cc -o hello hello.c` or `ftn -o hello hello.f`
• On kraken.nics.utk.edu, jaguar.ccs.ornl.gov
• Use PBS Batch Script, e.g. submit.pbs
• run: `qsub submit.pbs`

```bash
#!/bin/bash
#PBS -A your-project-id
#PBS -N test
#PBS -j oe
#PBS -l walltime=1:00:00,size=12
cd $PBS_O_WORKDIR
date
aprun -n 4 ./hello
```
**SPMD**

- Single Program, Multiple Data Programming Paradigm
- Same program runs on all processors, however, each processor operates on different set of data
- The simplest parallel parallel paradigm
- Generally contains:
  ```
  if ( my_processor_id .eq. designated_id ) then
    -------
  end if
  ```
Hello World Again

```fortran
program Hello_World
include 'mpif.h'
integer me, ierr, ntag, status(MPI_STATUS_SIZE)
character(12) message

call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, me, ierr)
ntag = 100
if ( me .eq . 0) then
    message = 'Hello, World'
call MPI_Send(message, 12, MPI_CHARACTER, 1, ntag, MPI_COMM_WORLD, ierr)
else
    call MPI_Recv(message, 12, MPI_CHARACTER, 0, ntag, MPI_COMM_WORLD, status, ierr)
    print *, 'node',me, ':', message
endif

call MPI_FINALIZE(ierr)
end
```
Parallel Processing

- **RANK ID**: 
P0  
P1

- **Message Passing**
  - message = Hello World
  - message = (empty )
  - Message
  - Hello World

- **Done (exit)**
  - print
  - print
Sending a Message

- C :: Standard send
  
  ```c
  int MPI_Send(&buf, count, datatype, dest, tag, comm)
  ```
  - &buf : pointer of object to be sent
  - count : the number of items to be sent, e.g. 10
  - datatype : the type of object to be sent, e.g. MPI_INT
  - dest : destination of message (rank of receiver), e.g. 6
  - tag : message tag, e.g. 78
  - comm : communicator, e.g. (MPI_COMM_WORLD)

- Fortran ::

  ```fortran
  call MPI_SEND(buf, count, datatype, dest, tag, comm, ierror)
  ```
Receiving a Message

• C :: Blocking receive
  int MPI_Recv(&buf, count, datatype, source, tag, comm, &status)
  – source : the node to receive from, e.g. 0
  – &status : a structure which contains three fields, the source, tag, and error code of the incoming message.

• Fortran ::
  call MPI_RECV(buf, count, datatype, source, tag, comm, status(MPI_STATUS_SIZE), ierror)
  – status : an array of integers of size MPI_STATUS_SIZE
For a Communication to Succeed

• Sender must specify a valid destination rank
• Receiver must specify a valid source rank
  – may use wildcard : MPI_ANY_SOURCE
• The communicator must be the same
• Tags must match
  – may use wildcard : MPI_ANY_TAG
• Message types must match
• Receiver’s buffer must be large enough
<table>
<thead>
<tr>
<th>MPI Datatypes</th>
<th>C Datatypes</th>
<th>MPI Datatypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_CHAR</td>
<td>signed char</td>
<td>MPI_SHORT</td>
</tr>
<tr>
<td>MPI_INT</td>
<td>signed int</td>
<td>MPI_UNSIGNED_INT</td>
</tr>
<tr>
<td>MPI_LONG</td>
<td>signed long int</td>
<td>MPI_UNSIGNED_LONG</td>
</tr>
<tr>
<td>MPI_FLOAT</td>
<td>float</td>
<td>MPI_UNSIGNED_FLOAT</td>
</tr>
</tbody>
</table>
# MPI Basic Datatypes - Fortran

<table>
<thead>
<tr>
<th>MPI Datatypes</th>
<th>Fortran Datatypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>MPI_REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>MPI_DOUBLE_PRECISION</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>MPI_COMPLEX</td>
<td>COMPLEX</td>
</tr>
<tr>
<td>MPI_LOGICAL</td>
<td>LOGICAL</td>
</tr>
</tbody>
</table>
Sample Program: Send and Receive

```
#include "mpi.h"
main(int argc, char ** argv) {
    int my_PE_num, numtorecv, numtosend=42;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_PE_num);

    if ( my_PE_num == 0) {
        MPI_Recv(&numtorecv, 1, MPI_INT, MPI_ANY_SOURCE, 
                  MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        printf("Number received is : %d\n", numtorecv);
    } else MPI_Send ( &numtosend, 1, MPI_INT, 0, 10, MPI_COMM_WORLD); 
    MPI_Finalize();
}
```
MPI Point to Point Communications

- Communication between two processes
- Source process sends message to destination process
- Communication takes place within a communicator
- Blocking vs non-blocking calls
- MPI defines four communication modes for blocking and non-blocking send: synchronous, buffered, ready, and standard
- The receive call does not specify communication mode--it is simply blocking and non-blocking.
- Two messages sent from one process to another will arrive in that relative order.
MPI Send

- MPI has 8 different types of Send
- The nonblocking send has an extra argument of request handle

<table>
<thead>
<tr>
<th></th>
<th>blocking</th>
<th>nonblocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard</td>
<td>MPI_Send</td>
<td>MPI_Isend</td>
</tr>
<tr>
<td>synchronous</td>
<td>MPI_Ssend</td>
<td>MPI_ISsend</td>
</tr>
<tr>
<td>buffer</td>
<td>MPI_Bsend</td>
<td>MPI_Ibsend</td>
</tr>
<tr>
<td>ready</td>
<td>MPI_Rsend</td>
<td>MPI_Iresend</td>
</tr>
</tbody>
</table>
**Blocking Calls**

- A blocking send or receive call suspends execution of user’s program until the message buffer being sent/received is safe to use.
- In case of a blocking send, this means the data to be sent have been copied out of the send buffer, but they have not necessarily been received in the receiving task. The contents of the send buffer can be modified without affecting the message that was sent.
- The blocking receive implies that the data in the receive buffer are valid.
Non-Blocking Calls

- Non-blocking calls return immediately after initiating the communication.
- In order to reuse the send message buffer, the programmer must check for its status.
- The programmer can choose to block before the message buffer is used or test for the status of the message buffer.
- A blocking or non-blocking send can be paired to a blocking or non-blocking receive
### Communication Modes (Blocking)

<table>
<thead>
<tr>
<th>Communication Mode</th>
<th>Description</th>
</tr>
</thead>
</table>
| Synchronous Send         | MPI_SSEND
|                          | Return when the message is safely reused. Send over.             |
| Buffered Send            | MPI_BSEND
|                          | Return when message is in the system buffer.                    |
| Ready Send               | MPI_RSEND
|                          | Complete if message is ready to send or is waiting for receive. |
MPI_Send (Standard Send)

- Implemented by vendor to give good performance for most programs.
- Simple and easy to use
- Either synchronous or buffered
- MPICH : buffered send
- CRAY XT :
  - Based on MPICH2
  - Use a portals device for MPICH2
  - Support MPI2-RMA (one-sided)
  - Full MPI-IO support
  - No dynamic process management (NO Spawn process!!)
  - Man intro_mpi
Non-blocking Communications

• The non-blocking calls have the same syntax as the blocking calls, with two exceptions:
  – Each call has an “I” immediately following the “_”
  – The last argument is a handle to an opaque request object that contains information about the message

• Non-blocking call returns immediately after initiating the communication

• The programmer can block or check for the status of the message buffer: MPI_Wait or MPI_Test.
Non-blocking Send and Receive

- Fortran:
  call MPI_Isend(buf, count, datatype, dest, tag, comm, handle, ierr)
  call MPI_Irecv(buf, count, datatype, src, tag, comm, handle, ierr)
  call MPI_Test(handle, flag, status, ierr)
  call MPI_Wait(handle, status, ierr)

- C:
  MPI_Isend(&buf, count, datatype, dest, tag, comm, &handle)
  MPI_Irecv(&buff, count, datatype, src, tag, comm, &handle)
  MPI_Wait(&handle, &status)
  MPI_Test(&handle, &flag, &status)
Testing Communications for Completion

• MPI_Wait (request, status)
  These routines block until the communication has completed. They are useful when the data from the communication buffer is about to be re-used

• MPI_Test (request, flag, status)
  This routine blocks until the communication specified by the handle request has completed. The request handle will have been returned by an earlier call to a non-blocking communication routine. The routine queries completion of the communication and the result (TRUE of FALSE) is returned in flag
Timer

- C
  
  `double MPI_Wtime(void)`

- Fortran:
  
  `double precision MPI_Wtime()`

- Time is measured in seconds.

- Time to perform a task is measured by consulting the timer before and after

- Modify your program to measure its execution time and print it out
Collective Communications

• Substitutes for a more complex sequence of p-p calls
• Involve all the processes in a process group
• Called by all processes in a communicator
• All routines block until they are locally complete
• Receive buffers must be exactly the right size
• No message tags are needed
• Divided into three subsets:
  – synchronization, data movement, and global computation
Barrier Synchronization Routines

- To synchronize all processes within a communicator
- A node calling it will be blocked until all nodes within the group have called it.
- C:
  
  ierr = MPI_Barrier(comm)

- Fortran:
  
  call MPI_Barrier(comm, ierr)
Broadcast

- one processor sends some data to all processors in a group
- C
  
  ```c
  ierr = MPI_Bcast(buffer, count, datatype, root, comm)
  ```

- Fortran
  
  ```fortran
  call MPI_Bcast(buffer, count, datatype, root, comm, ierr)
  ```

- The MPI_Bcast must be called by each node in a group, specifying the same communicator and root. The message is sent from the root process to all processes in the group, including the root process.
**Scatter**

- Data are distributed into \( n \) equal segments, where the \( i \)th segment is sent to the \( i \)th process in the group which has \( n \) processes.
- **C**:
  
  ```c
  ierr = MPI_Scatter(&sbuff, scount, sdatatype, &rbuf, rcount, rdatatype, root, comm)
  ```
- **Fortran**:
  
  ```fortran
  call MPI_Scatter(sbuff, scount, sdatatype, rbuf, rcount, rdatatype, root, comm, ierr)
  ```
Example: Scatter

real sbuf(12), rbuf(2)
call MPI_Scatter(sbuf, 2, MPI_INT, rbuf, 2, MPI_INT, 3, MPI_COMM_WORLD, ierr)
Scatter and Gather

**Scatter**

<table>
<thead>
<tr>
<th>PE 0</th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PE 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PE 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PE 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PE 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Gather**

<table>
<thead>
<tr>
<th>DATA</th>
<th>scatter</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td></td>
<td>A0</td>
</tr>
<tr>
<td>A1</td>
<td></td>
<td>A1</td>
</tr>
<tr>
<td>A2</td>
<td></td>
<td>A2</td>
</tr>
<tr>
<td>A3</td>
<td></td>
<td>A3</td>
</tr>
<tr>
<td>A4</td>
<td></td>
<td>A4</td>
</tr>
<tr>
<td>A5</td>
<td></td>
<td>A5</td>
</tr>
</tbody>
</table>

Parallel Programming with MPI
Gather

- Data are collected into a specified process in the order of process rank, reverse process of scatter.
- C:
  ```c
  ierr = MPI_Gather(&sbuf, scount, sdatatype, &rbuf, rcount, rdatatype, root, comm)
  ```
- Fortran:
  ```fortran
  call MPI_Gather(sbuff, scount, sdatatype, rbuff, rcount, rdatatype, root, comm, ierr)
  ```
Example: Gather

real sbuf(2), rbuf(12)
call MPI_Gather(sbuf, 2, MPI_INT, rbuf, 2, MPI_INT, 3, MPI_COMM_WORLD, ierr)
Scatterv and Gatherv

- allow varying count of data and flexibility for data placement
- C :
  ```c
  ierr = MPI_Scatterv( &sbuf, &scount, &displace, sdatatype,
                        &rbuf, rcount, rdatatype, root, comm)
  ierr = MPI_Gatherv(&sbuf, scount, sdatatype, &rbuf, 
                     &rcount, &displace, rdatatype, root, comm)
  ```
- Fortran :
  ```fortran
  call MPI_Scatterv(sbuf,scount,displace,sdatatype, rbuf, 
                   rcount, rdatatype, root, comm, ierr)
  ```
Allgather

\[ \text{ ierr } = \text{MPI\_Allgather}(&\text{sbuf}, \text{scount}, \text{stype}, &\text{rbuf}, \text{rcount}, \text{rtype}, \text{comm}) \]

<table>
<thead>
<tr>
<th>PE 0</th>
<th>PE 1</th>
<th>PE 2</th>
<th>PE 3</th>
<th>PE 4</th>
<th>PE 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>B0</td>
<td>C0</td>
<td>D0</td>
<td>E0</td>
<td>F0</td>
</tr>
</tbody>
</table>

Allgather

<table>
<thead>
<tr>
<th>PE 0</th>
<th>PE 1</th>
<th>PE 2</th>
<th>PE 3</th>
<th>PE 4</th>
<th>PE 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>B0</td>
<td>C0</td>
<td>D0</td>
<td>E0</td>
<td>F0</td>
</tr>
</tbody>
</table>

Parallel Programming with MPI
All to All

MPI_Alltoall(sbuf, scount, stype, rbuf, rcount, rtype, comm)

sbuf : starting address of send buffer (*)
scount : number of elements sent to each process
stype : data type of send buffer
rbuf : address of receive buffer (*)
rcount : number of elements received from any process
rtype : data type of receive buffer elements
comm : communicator
**All to All**

<table>
<thead>
<tr>
<th>PE 0</th>
<th>PE 1</th>
<th>PE 2</th>
<th>PE 3</th>
<th>PE 4</th>
<th>PE 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>B0</td>
<td>C0</td>
<td>D0</td>
<td>E0</td>
<td>F0</td>
</tr>
<tr>
<td>A1</td>
<td>B1</td>
<td>C1</td>
<td>D1</td>
<td>E1</td>
<td>F1</td>
</tr>
<tr>
<td>A2</td>
<td>B2</td>
<td>C2</td>
<td>D2</td>
<td>E2</td>
<td>F2</td>
</tr>
<tr>
<td>A3</td>
<td>B3</td>
<td>C3</td>
<td>D3</td>
<td>E3</td>
<td>F3</td>
</tr>
<tr>
<td>A4</td>
<td>B4</td>
<td>C4</td>
<td>D4</td>
<td>E4</td>
<td>F4</td>
</tr>
<tr>
<td>A5</td>
<td>B5</td>
<td>C5</td>
<td>D5</td>
<td>E5</td>
<td>F5</td>
</tr>
</tbody>
</table>

**DATA** → alltoall → **DATA**

<table>
<thead>
<tr>
<th>PE 0</th>
<th>PE 1</th>
<th>PE 2</th>
<th>PE 3</th>
<th>PE 4</th>
<th>PE 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>B0</td>
<td>C0</td>
<td>D0</td>
<td>E0</td>
<td>F0</td>
</tr>
<tr>
<td>A1</td>
<td>B1</td>
<td>C1</td>
<td>D1</td>
<td>E1</td>
<td>F1</td>
</tr>
<tr>
<td>A2</td>
<td>B2</td>
<td>C2</td>
<td>D2</td>
<td>E2</td>
<td>F2</td>
</tr>
<tr>
<td>A3</td>
<td>B3</td>
<td>C3</td>
<td>D3</td>
<td>E3</td>
<td>F3</td>
</tr>
<tr>
<td>A4</td>
<td>B4</td>
<td>C4</td>
<td>D4</td>
<td>E4</td>
<td>F4</td>
</tr>
<tr>
<td>A5</td>
<td>B5</td>
<td>C5</td>
<td>D5</td>
<td>E5</td>
<td>F5</td>
</tr>
</tbody>
</table>

**Parallel Programming with MPI**
Global Reduction Routines

- The partial result in each process in the group is combined together using some desired function.
- The operation function passed to a global computation routine is either a predefined MPI function or a user supplied function.
- Examples:
  - global sum or product
  - global maximum or minimum
  - global user-defined operation
Reduce and Allreduce

- MPI_Reduce(sbuf, rbuf, count, stype, op, root, comm)
- MPI_Allreduce(sbuf, rbuf, count, stype, op, comm)

- sbuf: address of send buffer
- rbuf: address of receive buffer
- count: the number of elements in the send buffer
- stype: the datatype of elements of send buffer
- op: the reduce operation function, predefined or user-defined
- root: the rank of the root process
- comm: communicator

-- mpi_reduce returns results to single process
-- mpi_allreduce returns results to all processes in the group
## Predefined Reduce Operations

<table>
<thead>
<tr>
<th>MPI NAME</th>
<th>FUNCTION</th>
<th>MPI NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_MAX</td>
<td>Maximum</td>
<td>MPI_LOR</td>
</tr>
<tr>
<td>MPI_MIN</td>
<td>Minimum</td>
<td>MPI_BOR</td>
</tr>
<tr>
<td>MPI_SUM</td>
<td>Sum</td>
<td>MPI_LXOR</td>
</tr>
<tr>
<td>MPI_PROD</td>
<td>Product</td>
<td>MPI_BXOR</td>
</tr>
</tbody>
</table>
Example of Reduce w/Sum

A routines that computes the dot product of two vectors that are distributed across a group of processes and returns the answer to node zero

```fortran
subroutine PAR_BLAS1( m , a , b , c , comm)
  real a(m) , b(m) , sum , c
  integer m , comm , I , ierr

  sum = 0.0
  do I = 1 , m
    sum = sum + a( I ) * b ( I )
  end do

  call MPI_Reduce( sum , c , 1 , MPI_REAL , MPI_SUM,0, comm , ierr )
return
```

Parallel Programming with MPI
**Simple Matrix Multiplication Algorithm**

- Matrix A is copied to every processors (FORTRAN)
- Matrix B is divided into blocks and distributed to processors
- Perform matrix multiplication simultaneously
- Output solutions

\[
A \begin{array}{c}
\text{proc} \\
1, 2, 3, 4
\end{array} \quad \ast \quad \begin{array}{cccc}
p^1 & p^2 & p^3 & p^4
\end{array} = \begin{array}{cccc}
p^1 & p^2 & p^3 & p^4
\end{array}
\]
Parallel Processing

Obtain
A(1:8, 1-8)
B(1:8, 1-8)
A(1:8, 1-8)

B(1:8, 7-8)
B(1:8, 5-6)
B(1:8, 3-4)
B(1:8, 1-2)

C(1:8, 7-8)
C(1:8, 5-6)
C(1:8, 3-4)
C(1:8, 1-2)

C = A * B
Matrix Multiplication

Step 1

Step 2
Matrix Multiplication (continued)

step 3

step 4
### Fox’s Algorithm (1)

- Broadcast the diagonal element of block A in rows, perform multiplication.

\[
\begin{array}{ccc}
A(0,0) & A(0,0) & A(0,0) \\
A(1,1) & A(1,1) & A(1,1) \\
A(2,2) & A(2,2) & A(2,2)
\end{array}
\times
\begin{array}{ccc}
B(0,0) & B(0,1) & B(0,2) \\
B(1,0) & B(1,1) & B(1,2) \\
B(2,0) & B(2,1) & B(2,2)
\end{array}
= \begin{array}{ccc}
C(0,0) \\
C(1,1) \\
C(2,2)
\end{array}
\]

- \( C(0,0) = A(0,0) \times B(0,0) + A(0,1) \times B(1,0) + A(0,2) \times B(2,0) \)
- \( C(0,1) = A(0,0) \times B(0,1) + A(0,1) \times B(1,1) + A(0,2) \times B(2,1) \)
- \( C(0,2) = A(0,0) \times B(0,2) + A(0,1) \times B(1,2) + A(0,2) \times B(2,2) \)
- \( C(1,0) = A(1,0) \times B(0,0) + A(1,1) \times B(1,0) + A(1,2) \times B(2,0) \)
- \( C(1,1) = A(1,0) \times B(0,1) + A(1,1) \times B(1,1) + A(1,2) \times B(2,1) \)
- \( C(1,2) = A(1,0) \times B(0,2) + A(1,1) \times B(1,2) + A(1,2) \times B(2,2) \)
- \( C(2,0) = A(2,0) \times B(0,0) + A(2,1) \times B(1,0) + A(2,2) \times B(2,0) \)
- \( C(2,1) = A(2,0) \times B(0,1) + A(2,1) \times B(1,1) + A(2,2) \times B(2,1) \)
- \( C(2,2) = A(2,0) \times B(0,2) + A(2,1) \times B(1,2) + A(2,2) \times B(2,2) \)
Fox’s Algorithm (2)

- Broadcast next element of block A in rows, shift Bij in column, perform multiplication.

\[
\begin{align*}
\begin{array}{ccc}
A(0,1) & A(0,1) & A(0,1) \\
A(1,2) & A(1,2) & A(1,2) \\
A(2,0) & A(2,0) & A(2,0) \\
\end{array}
& \times
\begin{array}{ccc}
B(1,0) & B(1,1) & B(1,2) \\
B(2,0) & B(2,1) & B(2,2) \\
B(0,0) & B(0,1) & B(0,2) \\
\end{array}
= \begin{array}{ccc}
C(0,0) & & \\
C(1,1) & & \\
C(2,2) & & \\
\end{array}
\end{align*}
\]

- \(C(0,0) = A(0,0) \times B(0,0) + A(0,1) \times B(1,0) + A(0,2) \times B(2,0)\)
- \(C(0,1) = A(0,0) \times B(0,1) + A(0,1) \times B(1,1) + A(0,2) \times B(2,1)\)
- \(C(0,2) = A(0,0) \times B(0,2) + A(0,1) \times B(1,2) + A(0,2) \times B(2,2)\)
- \(C(1,0) = A(1,0) \times B(0,0) + A(1,1) \times B(1,0) + A(1,2) \times B(2,0)\)
- \(C(1,1) = A(1,0) \times B(0,1) + A(1,1) \times B(1,1) + A(1,2) \times B(2,1)\)
- \(C(1,2) = A(1,0) \times B(0,2) + A(1,1) \times B(1,2) + A(1,2) \times B(2,2)\)
- \(C(2,0) = A(2,0) \times B(0,0) + A(2,1) \times B(1,0) + A(2,2) \times B(2,0)\)
- \(C(2,1) = A(2,0) \times B(0,1) + A(2,1) \times B(1,1) + A(2,2) \times B(2,1)\)
- \(C(2,2) = A(2,0) \times B(0,2) + A(2,1) \times B(1,2) + A(2,2) \times B(2,2)\)
**Fox’s Algorithm (3)**

- Broadcast next element of block A in rows, shift Bij in column, perform multiplication.

\[
\begin{array}{ccc}
A(0,2) & A(0,2) & A(0,2) \\
A(1,0) & A(1,0) & A(1,0) \\
A(2,1) & A(2,1) & A(2,1) \\
\end{array}
\times
\begin{array}{ccc}
B(2,0) & B(2,1) & B(2,2) \\
B(0,0) & B(0,1) & B(0,2) \\
B(1,0) & B(1,1) & B(1,2) \\
\end{array}
= 
\begin{array}{c}
C(0,0) \\
C(1,1) \\
C(2,2) \\
\end{array}
\]

- \( C(0,0) = A(0,0) \times B(0,0) + A(0,1) \times B(0,1) + A(0,2) \times B(0,2) + A(0,2) \times B(2,0) \)
- \( C(0,1) = A(0,0) \times B(0,1) + A(0,1) \times B(1,1) + A(0,2) \times B(1,2) + A(0,2) \times B(2,1) \)
- \( C(0,2) = A(0,0) \times B(0,2) + A(0,1) \times B(1,2) + A(0,2) \times B(2,2) + A(0,2) \times B(2,2) \)
- \( C(1,0) = A(1,0) \times B(0,0) + A(1,1) \times B(0,1) + A(1,2) \times B(0,2) + A(1,2) \times B(2,0) \)
- \( C(1,1) = A(1,0) \times B(0,1) + A(1,1) \times B(1,1) + A(1,2) \times B(1,2) + A(1,2) \times B(2,1) \)
- \( C(1,2) = A(1,0) \times B(0,2) + A(1,1) \times B(1,2) + A(1,2) \times B(2,2) + A(1,2) \times B(2,2) \)
- \( C(2,0) = A(2,0) \times B(0,0) + A(2,1) \times B(0,1) + A(2,2) \times B(0,2) + A(2,2) \times B(2,0) \)
- \( C(2,1) = A(2,0) \times B(0,1) + A(2,1) \times B(1,1) + A(2,2) \times B(1,2) + A(2,2) \times B(2,1) \)
- \( C(2,2) = A(2,0) \times B(0,2) + A(2,1) \times B(1,2) + A(2,2) \times B(2,2) + A(2,2) \times B(2,2) \)
Acknowledgments

- JICS staff, GRAs, collaborators, and previous workshop participants
- NCCS and NICS staff at UTK/ORNL and their resources
- All contributors to the art and science of parallel computing
The End

• The End!