Introduction to Parallel Computing using Cluster

Agenda

- What is parallel computing?
- Why parallel computing?
- What is cluster?
- How to use cluster for your work?
- Application example

Why we need High Performance Computer?

Hypothesis → Experiment

Compute-Intensive Applications

- Simulation and Modeling problems:
  - Based on successive approximations. More calculations, better results
  - Optimization problems
- Problems that dependent on computations and manipulations of large amounts of data
- Example
  - Weather Prediction
  - Image and Signal Processing, Graphics
  - Database and Data Mining

CFD for Clean room

- Analyzing behaviour of air flow in clean room for electronics industry
- Collaboration project
  - Suranaree University of Technology
  - Kasetsart University
  - Funded by NECTEC

CFD Software

- CAMETA version 3.0 (SUT)
  - Time-independent (steady-state) solution
  - Three-dimensional domain
  - Cartesian coordinate system
- Physical quantities of interest:
  - Temperature distribution
  - Relative humidity distribution
  - Particle concentration distribution
Molecular Dynamic Simulation

- Drug Discovery using molecular docking
  - Avian Flu
  - HIV
- Analyzing property of Chemical compound

SWAP Model Parameter identification - Data Assimilation using RS and GA – (KU/AIT)

RS Observation

Input Parameters
- sowing date, soil property, Water management, and etc.

Crop Growth Model

Assimilation by finding Optimized parameters By GA

RS

LAI, Evapotranspiration

Day Of Year

Challenges

- The calculation time for identify SWAP parameters only for 1 pixel (1 sq.km) takes several minutes to 30 minutes.
- Thus, a RS image of 1000 x 1000 sq.km of 1000x1000 pixels will take more than 50 years (30min x 1000 x 1000) is not acceptable.
- Solutions
  - Parallel Computing

Graphics Rendering and Special Effect

- Rendering
  - Generating 3D image from Model
- Problem
  - Rendering is a time consuming process especially for complex and realistic scene
  - Massive number of rendering job needed to be done to create a movie

Top500 Fastest Installed Computers

- www.top500.org
- List the top 500 supercomputer at sites worldwide.
- Provides a snapshot of the SC installed around the world.
- Began in 1993, published every 6 months.
- Measures the performance for the TPP Linpack Benchmark
- Provides a way to measure trends

24th List:

1. BlueGene/L
   - DOE/IBM
   - Rochester, USA
   - BlueGene/L, 1024
   - Rank: 1
   - TFlops: 1.11TFlops

2. Columbia
   - NAS, Alumni
   - Mountain View, USA
   - SGI Altix/Altix XT
   - Rank: 31
   - TFlops: 9.47TFlops

3. Earth Simulator
   - Earth Simulator Center
   - Yokohama, Japan
   - NEC
   - Rank: 35
   - TFlops: 10TFlops

4. MareNostrum
   - Barcelona Supercomputer Center
   - Barcelona, Spany
   - IBM BlueGene/L, 1024
   - Rank: 20
   - TFlops: 1.11TFlops

5. Thunder
   - Lawrence Livermore National Lab
   - Livermore, USA
   - 1024/1024/1280/640/8 Quadrics
   - Rank: 19
   - TFlops: 1.11TFlops
Parallel Processing

- Solving large problem by breaking it into a number of small problems, then solve them on multiple processors at the same time
- Real life example
  - building a house using many workers working on separate parts
  - Assembly line in factory

Parallel Computer

- Parallel computer is a special computer that is designed for parallel processing
  - Multiple processors
  - High-speed interconnection network that link these processors together
  - Hardware and software that help coordinate the computing tasks
- Sometimes this type of computer are called MPP (Massively Parallel Processor) Computer

Shared Memory MPP

- Multiple Processors
- Global Shared Memory
- Processors communicate using shared memory
**Distributed Memory MPP**

- Multiple processors
- Each processor has local memory
- No global memory, use message passing to communicate
- Also called Multicomputer

**Introduction**

- Cluster computing is a technology related to the building of high performance scalable computing system from a collection of small scalable computing system and high speed interconnection network

**Cluster System Structure**

**MAEKA System**

*Massive Adaptable Environment for Kasetsart Applications*

- 64 bit Opteron Cluster in Collaboration between
  - Kasetsart University
  - AMD Inc
  - ATEC
- Main Compute Facility in Kasetsart University
- 32 nodes 64 Opteron processors

**How to use cluster system for your work**

- High Throughput Computing
  - Use cluster as a collection of processors
  - Run sequential applications
  - Using job scheduler to control the execution
- High Performance Computing
  - Use cluster as a parallel machine
  - Develop parallel application to harness the computing power from the cluster
High Throughput Computing
- High throughout, not high performance
  - Complete most number of jobs in shortest amount of time
- Serial, parametric (usually), non-parallelized code
  - Solve them on multiple processors at the same time, varying input parameters
- Example
  - BLAST, Monte Carlo simulation
- Use of Load Schedulers
  - Condor, Codine, LSF, SunGridEngine
- Pros and Cons
  - Easy to get started. Use the sequential code in C or Fortran.
  - Excellent for many type of applications such as
    - Parametric computing: Running the same computation with multiple data set
    - Distributed application such as massive rendering in animation industry
  - Excellent when model can fit well in memory of a single computer
  - No communication at all

High Performance Computing
- Maximum performance, not maximum throughput
- Use of specialised codes, libraries
  - MPI (Message Passing Interface)
  - Parallel Maths Libraries (ScaLapack)
- Solve large problem by breaking it in to a number of small problems (data or task partitioning), then solve them on distributed, multiple processors at the same time.
- Pros and Cons
  - Difficult since a parallel program must be developed
  - Good when
    - Problem is larger than memory size of a single machines
    - Speedup for a single instance of problem is needed

Steps in Developing Parallel Application
- Develop sequential application first
- Identify
  - Most time consuming part using profiling tool
  - Parallelism inherent in that part
- Choose strategy for data partitioning and task partitioning
- Choose the development tools on target parallel machine
- Add code for task control and communication
  - Compile, test , debug, measure performance, improve

Program decomposition
- Functional Decomposition
  - Decompose program to multiple functions , distribute these functions as tasks that run on multiple processors concurrently
- Domain decomposition
  - Partition problem based on data domain and distribute both data and computation to multiple processors to run concurrently

Programming using Message Passing
- Partition task into multiple concurrent tasks that communicate by passing message
- MPI (message passing interface)
  - a de facto standard that is now supported by all platform.
  - Free implementations are also available (MPICH, LAM)

Introduction to MPI
- Standard for parallel programming on cluster and supercomputer
- MPI is implementation and machine independent
  - Works for parallel computers, clusters, and heterogeneous networks
- Many implementation exists
  - MPICH from Argonne National Laboratory
  - LAM from Notre-Dam University
  - MPI-PRO from MPI software technology
A Minimal MPI Program (C)

```c
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    MPI_Init( &argc, &argv );
    printf( "Hello, world!\n" );
    MPI_Finalize();
    return 0;
}
```

Running MPI program (with MPICH)

```
% mpicc -o hello hello.c
% mpirun -np 4 hello
```

Example: Finding PI

Value of PI can be estimate from the equation

\[
\int_{1}^{\infty}\frac{1}{1+x^2} \, dx = \frac{\pi}{4}
\]

- Integrate this function from 0 to 1 numerically and multiply by 4 to get the value of PI
- How to do it in MPI?

Finding PI

- Divided the integration into many small regions
- Each processors calculate the area of different regions
- Sum it together

Example: PI in C - 1

```c
#include "mpi.h"
#include <math.h>

int main(int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i, rc;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x, a;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    while (!done)  {
        if (myid == 0) {
            printf("Enter the number of intervals: (0 quits) ");
            scanf("%d",&n);
        }
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
        if (n == 0) break;
        h = 1.0 / (double) n;
        sum = 0.0;
        for (i = myid + 1; i <= n; i += numprocs) {
            x = h * ((double)i - 0.5);
            sum += 4.0 / (1.0 + x*x);
        }
        mypi = h * sum;
        MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
                   MPI_COMM_WORLD);
        if (myid == 0)
            printf("pi is approximately %.16f, Error is
                   %.16f\n", pi, fabs(pi - PI25DT));
    }
    MPI_Finalize();
    return 0;
}
```

Example: PI in C - 2

```c
h = 1.0 / (double) n;
sum = 0.0;
for (i = myid + 1; i <= n; i += numprocs) {
    x = h * (i - 0.5);
    sum += 4.0 / (1.0 + x*x);
}
pi = h * sum;
MPI_Reduce(&pi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
           MPI_COMM_WORLD);
if (myid == 0)
    printf("pi is approximately %.16f, Error is
           %.16f\n", pi, fabs(pi - PI25DT));
}```
Performance Metric for Parallel System

- **Runtime**
  - Sequential runtime ($T_s$) is an elapse time between the beginning and the end of execution of a program on sequential computer.
  - Parallel runtime ($T_p$) is the elapse time from the moment that first processors start the parallel execution until the last processor finish its execution.

**SPEED UP**
- Ratio between sequential runtime and parallel runtime.
- Measure the gain obtained from parallelizing the code.
- For system with $P$ processors, perfect speedup is $S = \frac{T_s}{T_p}$.

**Efficiency**
- A measure of the fraction of time for which a processor is used.
- For parallel system with $P$ processors, the efficiency $E$ is defined as:
  $$E = \frac{S}{P}$$
- Perfect efficiency is when $S=P$, $E=1$.

Performance of Parallel code

- Performance depends on several factors:
  - How good you balance the load.
  - Communication.
  - Selecting right machine for task granularity.

Load Balancing

- Parallel computation stop when last task terminate execution.
- If the computation time on each processors is not equal, faster processors have to wait for slow one. Performance loss.

Causes of Work Load Imbalance

- Heterogeneity:
  - Processors have different speed but received the same amount of work.
- Synchronization:
  - Processors trying to synchronize the operation but certain processors have a problem.
    - Interfered by other running process.
    - Wait for I/O.
    - Network is busy.
Communication

- Parallel task generate communication in many way
  - Explicit communication
    - Data exchange between task
  - Synchronization
  - Implicit Communication
    - Moving data, loading file
- Optimizing communication can result in a big improvement

Some frequently used technique

- Preloading data to destination and use local data access through out the computation
- Send small number of large message
  - Capitalize on property of TCP/IP network
- Decompose problem in the form that reduce communication
- Reduce frequency of synchronization
  - Implicit synchronization using data message

Task Granularity

- Granularity is a ratio between computation and communication
  - Fine-grain parallelism
    - Short execution time between cycles of communication
    - High overhead, need special hardware support
  - Coarse grain parallelism
    - Low computation between communication cycle
    - Lower overhead
    - Less parallelism, less potential for load balancing
    - Good for machine with low speed interconnection network

Running parallel matrix multiplication

- Compare the performance of parallel matrix multiplication
  - Very embarrassingly parallel code
- Running on 2 cluster
  - GASS 6 node 12 processors Athlon MP1800+ and fast ethernet
  - MAGI 4 node of Athlon 2200+ and Gigabit ethernet

Running on GASS

```
time_t t2,t1;
fill_matrix(A);
fill_matrix(B);
t1=time(NULL);
for (i=0; i<SIZE; i++)
  for (j=0; j<SIZE; j++) {
    C[i][j]=0;
    for (k=0; k<SIZE; k++)
      C[i][j] += A[i][k]*B[k][j];
  }
t2=time(NULL);
printf("1,%10.3f\n",(float) (t2-t1));
```

```
$ gcc -o mmx mmx.c
$ ./mmx
1,    40.000
$
```

```
$ mpicc -o pmx pmx.c
$ mpirun -np 2 pmx
2,21.90
$ mpirun -np 4 pmx
4,11.84
$ mpirun -np 8 pmx
8, 7.55
```

```
$ iperf
Usage: iperf [-s|-c host] [options]
Try `iperf --help' for more information.
```

```
$ rlogin gass2
Last login: Fri Sep  5 14:23:43 from gass1.cpe.ku.ac.th
```

```
$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 65.5 KBytes (default)
------------------------------------------------------------
```

```
$ bg
[1]+  Stopped                 iperf -s
```

```
$ rsh gass3
```

```
$ iperf -c gass2
[  4] local 158.108.35.38 port 5001 connected with 158.108.35.39 port 38600
------------------------------------------------------------
Client connecting to gass2, TCP port 5001
TCP window size: 65.5 KBytes (default)
------------------------------------------------------------
[  3] local 158.108.35.38 port 38600 connected with 158.108.35.39 port 5001
  0.0-10.0 sec    112 MBytes 94.2 Mbits/sec
[  4]  0.0-10.0 sec    112 MBytes 94.0 Mbits/sec
```
GASS Results

<table>
<thead>
<tr>
<th>P</th>
<th>Time(GASS)</th>
<th>SP</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>21.9</td>
<td>1.83</td>
<td>0.91</td>
</tr>
<tr>
<td>4</td>
<td>11.84</td>
<td>1.38</td>
<td>0.84</td>
</tr>
<tr>
<td>8</td>
<td>7.53</td>
<td>5.30</td>
<td>0.58</td>
</tr>
</tbody>
</table>

Thank you