Optimum utilization of clusters

Pushan Majumdar

Indian Association for the Cultivation of Sciences, Jadavpur, Kolkata

Outline

- Introduction: What is a cluster?
- Do I really need a cluster
- Getting to know the characteristics of your cluster
- Modern compilers and SSE
- Before launching a job
- Summary

What is a cluster?

A cluster has three main components:

- The individual computing node
- the network which connects these nodes together
- the software which lets the different nodes work as a single system

speed of the cluster depends mostly on the speed of the network

eg. for 3 GHz Pentium PC's connected together by normal 100Mbps ethernet, one gets typically only 8-9% of the peak performance. Upgrading the network to Gigabit ethernet improves performance to about 30% of the peak assuming all other things are optimal.

Specialized High speed interconnects : Myrinet, Dolphinics, Infiniband

Clusters at IACS

- Aneesh-ur-Rahman Centre

- IBM server p-590

Do I really need a cluster

Important : different programs run with different efficiencies on a cluster.

What kind of programs am I going to run on the cluster ?

– Program runs on a single machine, but many such runs have to be done for say different input parameters

– Memory requirement is such that program can run on a single machine, but run time is unrealistically large (10 years).

– Memory requirement is such that program cannot run on a single machine.

Condor

`http://www.cs.wisc.edu/condor/`

Condor is a batch processing system

Has a server, launching machines, machines in the LAN to run jobs.

user submits jobs to Condor along with resource (CPU speed, memory) requirements

Condor then looks around the network to see which machines are free at that time and submits the job there. As soon as that machine becomes busy, (say because the owner of that machine

is using the machine) condor stops the job on that machine and transfers it to the next available machine.

Once the job is over, Condor brings the output back to the launching machine.

All this goes on in the background without any user noticing anything.

really efficient as it uses the idle time of the machines

Getting to know the characteristics of your cluster

compute nodes :
Useful files (under linux):
/proc/cpuinfo : contains information on CPU
/proc/meminfo : contains information on memory

Useful command :
dmesg | grep eth : will give you information on network cards

Other useful information : What kind of switch are you using?
Ask your system administrator or look up the HPC website.

Modern compilers and SSE

Output of cat /proc/cpuinfo : ..... sse sse2 sse3 ......

SSE : Streaming SIMD (Single Instruction Multiple data) Extensions

PC processors come equipped with some extra registers which the user can access directly.

This feature enables quite a bit of speeding up of programs.

SSE originally added eight new 128-bit registers known as XMM0 through XMM7. Each register packs together four 32-bit single-precision floating point numbers. Operations can be operated

in parallel with SSE operations offering further performance increases in some situations

add two single precision, 4-component vectors

```
vec_res.x = v1.x + v2.x;
vec_res.y = v1.y + v2.y;
vec_res.z = v1.z + v2.z;
vec_res.w = v1.w + v2.w;
```

```
movaps xmm0,address-of-v1 ;xmm0=v1.w | v1.z | v1.y | v1.x
addps xmm0,address-of-v2;xmm0=v1.w+v2.w|v1.z+v2.z|v1.y+v2.y|v1.x+v2.x
movaps address-of-vec_res,xmm0
```

Modern compilers use SSE instructions while generating code.
Use capabilities of modern compilers as much as possible.

Example - Program for multiplication of matrix with vector

```
program main

  use ranlxd_generator              ! A random number generator
  implicit none
  integer, parameter :: n=7000      ! Size of the matrix

  real(8) :: mat(n,n), ran1(n), ran2(n)
  integer :: i,j,k

  call rlxd_init(1,100)             ! Initializing the random no. generator

  do i=1,n
  call ranlxd(mat(i,:))             ! Getting n sets of n random numbers
  enddo

  call ranlxd(ran1)                 ! Getting the size n random vector
!-------------------------------------------------------------
!Method 1
```

```
 do i=1,10                          ! Repeats the multiplication 10 times.
  ran2=0
  do j=1,n                          ! Usual way of multiplication of
  do k=1,n                          ! matrices using nested loops
  ran2(j)=ran2(j) + mat(k,j)*ran1(k)
  enddo
  enddo
 enddo
!-------------------------------------------------------------
!Method 2

 do i=1,10
  do j=1,n                          ! Using the dot product function
  ran2(j)=dot_product(mat(:,j),ran1)
  enddo
 enddo
!-------------------------------------------------------------
!Method 3

 do i=1,10                          ! Using the intrinsic matrix multiplication
  ran2=matmul(mat,ran1)
```

```
 enddo
!-----------------------------------------------------------------
  stop
  end
```

Timing:

Method 1:	real 0m10.226s	user 0m9.841s	sys 0m0.376s
Method 2:	real 0m8.950s	user 0m8.581s	sys 0m0.352s
Method 3:	real 0m9.974s	user 0m9.649s	sys 0m0.316s

Most efficient : Method 2

Before launching a job

Learn about the batch queues.

If there is no batch queue then:
for small clusters log onto nodes and do a top.
For large clusters use a shell script.

Usually all clusters have passwordless ssh or rsh installed.

```
for i in 1 2 3 4 5 6 7 8 9 10
do
echo $i
rsh node$i "cat /proc/loadavg"
done
```

Summary

- Evaluate your problem carefully
- Know your cluster well
- Test your program thoroughly for optimizations
- Have consideration for other users. Use existing batch systems or check the load factor on compute nodes.