HPC(High Performance Computer) Linux Cluster HowTo

이진호
System Engineer
System Programer
DBA

jin_sm (at) hotmail.com

HPC(High Performance Computer) Linux Cluster HowTo

지은이 이진호

펴냄 v0.8, 2003/05/12

이 문서는 동기종(homogeneous system) 간의 클러스터를 통하여 병렬처리 방식의 슈퍼컴퓨팅을 제작하는 방식을 기술하였습니다. 기본적으로 beowolf 방식 을 따라하고 있고, 응용을 하였으나, 한글로 된 관련 문서가 많이 없다는데 착안, 문서화 하게 되었습니다.

고친 과정

고침 0.82003-05-12고친이 ljn
오탈자 교정
고침 0.72003-05-12고친이 ljn
최초 작성

차례

표 목록

1장. 서론

이 문서는 GPL(GNU PUBLIC LICENSE) 에 의해 자유로이 배포될수 있습니다. 라이센스에 관한 설명은 http://www.fsf.org/copyleft/gpl.html에서 참고할수 있습니다.

1.1. 들어가기 전에

이 문서는 클러스터 에 사용될 모든 Machine 이 동일한 기종 하에서 구동한다고 가정합니다. 각자의 시스템 들은 독립된 boot 영역으로 구동되며 각각의 통신은 FastEthernet 으로 (10/100 or 1000 Mbps) 연결되어 있음을 가정합니다. 그리고 대규모 클러스터(32 Node 이상) 구축시 본문서의 내용대로 적용이 안될수도 있습니다. 이것은 대부분 유닉스 NFS(Network File System) 의 I/O 문제와 밀접한 관련이 있습니다. Beowolf 와 관련된 많은 문서들이 있기 때문에 여기서는 많은 내용의 주석이나 설명을 하지 않고 당장 실무에서 사용할수 있는 수준의 문서를 간결하고 가능한한 명확하게 만드는 것이 주 목적 입니다.

1.2. 감사의 글

이 문서를 작성하는데 도움을 주신분, 특히 한국 외국어대 컴퓨터 공학과 DB Lab 의 김경원 님<kimkyungmin (at) msn.com> 과 한희정 님<powercgi (at) hananet.net> 에게 감사의 말을 전합니다. 클러스터 관련 자료, 혹은 문서를 찾는데 많은시간을 절약하게 해주었습니다.

1.3. 피드백

이 문서에 대한 발전적인 제안이나 수정사항, 문제점 등에 대한 피드백은 언제든지 환영합니다. <jin_sm (at) hotmail.com >로 메일을 보내 주십시오.

2장. 개요

2.1. 하드웨어 의 선택

클러스터 를 구축하기 위해서는 어떠한 하드웨어를 사용하여 어느정도 규모의 클러스터를 구축할 것 인지 H/W 적인 선택을 해야 한다. 다음은 H/W 선택에 대한 Alpha-11 프로젝트 팀(http://www.alpha11.com) 의 몇가지 조언이다.

2.1.1. 프로세서(CPU)

클러스터를 구축하여 원하는 컴퓨팅 파워를 얻기위한 두가지 고려사항이 있다. 저성능의 컴퓨터를 다수 연결할것인가 고성능의 컴퓨터를 소수 연결할 것인가를 놓고 고려해야 한다. 일반적으로 노드의 수가 증가할수록 프로세서간의 통신이 증가하므로 네트워크 비용이 증가하게 되고 동기화에 필요한 시간이 증가하므로 병렬화 효율 역시 줄어들게 된다. 따라서 고성능 컴퓨터를 소수 연결하는 것이 효율면에서는 좋다. 그러나 고성능 컴퓨터의 가격은 그 성능에 비해 비싸기 때문에 타협을 해야 한다. Compaq 의 ALPHA 프로세서는 Floting 계산능력에 있어서 동일 clock 의 intel Pentium 머신에 비해 약 2.5배 정도 빠르며 대부분의 수치모델들이 floating 연산이 주를 이루는 것을 감안하면 수치모델을 위한 클러스터 구축에 적합하다. 또한 64bit 아키텍처 이므로 그에 따른 IO 관련 속도 향상에 도움을 줄수 있다. AMD 와 INTEL 의 ix86 으로 대변되는 프로세서들은 저가에 클러스터를 구성할수 있으며 다양한 H/W 와 S/W 의 지원을 받을수 있는 장점이 있다.

2.1.2. 네트워크 장치(NIC), 스위칭 허브

가장 일반적인 경우는 100Mbps 스위칭 허브와 랜카드를 사용하여 구성된다. 100Mbps 네트워크 장치를 사용하여 제작한 클러스터는 많은 사용자들에 의해 충분히 사용할 만한 장비라는 것이 입증되었다. 만약 더 좋은 네트워크 장치를 사용하기를 원한다면 미리넷(MyriNet) 과 기가비트이더넷(Gigabit) 을 고려할수 있으나 이들 장비는 고가이다. 특히 미리넷은 전용 프로토콜을 사용할경우 패킷전송시 지연시간(Latency)이 매우 작은 클러스터 전용 네트웍 장치 이지만 TCP 가 지원되지 않는 문제점은 있다. 최근 TCP 를 에뮬레이션 할수 있는 방법이 나와 있으나 아직 안정화 되어 있지 않은 걸로 알고있다.

2.1.3. 기타 주변장치

메모리는 1 bit 에러 보정능력이 있는 ECC 램과 일반램이 고려될수 있다. 좀더 신뢰성 있는 시스템을 구축하기 위해서는 ECC 램을 사용하는 것이 좋으나 가격은 좀더 비싸다. 메인보드의 선택에 있어서는 성능보다는 안정성에 중점을 두어서 선택하는 것이 바람직하다. 클러스터는 다수의 컴퓨터로 구성되므로 하드웨어가 문제를 일으킬 확률이 그만큼 증가하게 되기 때문이다. Storage(저장장치) 의 경우 일반적으로 IDE 방식과 SCSI 방식의 HDD 가 고려될수

있다. 이는 전적으로 어떤 프로그램을 수행할 것인지에 따라 결정될수 있다. I/O 가 빈번이 발생하는 프로그램을 수행한다거나 동시에 다수의 프로세스에 의해서 I/O 가 이루어 진다면 SCSI 를 고려해야 할것이다. 그리고 더 나은 안정성과 속도를 원한다면 고가의 RAID 역시 고려될수 있다. 한편 다수의 하드디스크를 이용하여 소프트웨어적으로 RAID 를 구성하는 방법도 있다.

2.2. 소프트웨어 의 선택

클러스터를 위한 소프트웨어를 크게 3가지로 나눈다면 클러스터에 사용될 OS, 클러스터를 구축하는데 필요 한 Middleware, 마지막으로 클러스터를 구성한후 사용할 프로그램으로 나눌수 있다. 다음은 이들에 대한 간단한 조언이다.

2.2.1. OS 에 대한 고려사항

가장 널리 사용되는 OS 로는 역시 리눅스를 들수 있다. 현재 리눅스를 이용하여 클러스터를 구축하기 위한 많은 기술문서와 라이브러리, 모니터링 툴, 큐잉시스템 등이 공개되어 있으며 심지어는 클러스터링 을 위한 커널이미지, 배포판 등도 나와 있다. 물론 리눅스뿐 아니라 UNIX 와 MS 의 Window 로도 클러스터를 구축할수 있다.

2.2.2. MiddleWare 에 대한 고려사항

여기서 말하는 미들웨어란 병렬 라이브러리, 모니터링 툴, 큐잉시스템 등 클러스터 관련 소프트웨어를 지칭한다. 먼저 병렬 라이브러리는 무수히 많이 있으며 이들중 원하는 라이브러리를 하나 또는 여러 개 선택해서 설치할수 있다. 모니터링 툴과 큐잉시스템 역시 다수가 공개되어 있으며 상용툴들도 나와 있다. 이들에 대한 자세한 정보는 Reference Sites 에 나와 있는 각 소프트웨어의 홈페이지를 참고하기 바란다.

2.2.3. 프로그램 병렬화 에 대한 고려사항

기존의 워크스테이션 에서 돌아가고 있는 프로그램을 클러스터 에서 실행한다고 해서 성능이 좋아지는 것은 아니다. 클러스터를 비롯한 병렬처리 컴퓨터는 병렬화 된 코드를 요구한다. 즉 시리얼 프로그램을 병렬화 한후 이를 클러스터에서 실행해야 한다. C, C++, Fortran 으로 만든 프로그램은 전통적으로 병렬화가 가능하다.

2.3. 요구 사항

여기서는 7대의 컴퓨터를 클러스터링 할것이며 이후 필요한 장비는 다음과 같다. 컴퓨터 7대, 각 컴퓨터는 CPU, RAM, BOARD, HDD 등의 기본적인 장치를 가지고 있다. 컴퓨터간 각각 Lan Card 2개 씩(필자는 NFS 와 Gateway 용 통신용으로 하나, 나머지 1개의 랜카드는 노드간의 실질적인 계산작업 용 통신으로 사용하기 위해 노드당 2개의 랜카드를 설치 했다. 물론 1개의 랜카드 만으로도 서비스 하는데 문제될 것은 없다.) 스위치 허브 1개, 모니터, 키보드, 마우스 등이 필요하다.

클러스터를 만들고 모니터링 하는데는 기본적인 리눅스 기술이 요구된다. 이러한 기술들은 이 문서의 범위를 넘어서기 때문에 다루지 않는다. 이러한 내용들은 http://www.kldp.org 나 http://www.tldp.org 에서 많은 자료를 얻을수 있다.

3장. 설치하기

3.1. 기본 사항

일반적으로 클러스터링 시스템에는 1 대의 관리노드와 n 대의 컴퓨팅 노드로 구성되어 있다. 관리노드는 말그대로 클러스터 시스템을 관리하는 노드 이고 컴퓨팅 노드는 주로 연산을 담당하게 되는 노드 이다. 관리노드와 컴퓨팅 노드는 물리적으로 분리시키는 것이 좋으나 여기선 특정 노드를 관리노드와 컴퓨팅 노드도 함께 세팅 할것이다. 클러스터 시스템이 구축된 후에 사용자 및 관리자는 는 관리노드에 접속하여 전체 클러스터 시스템을 사용, 관리하게 될것이다. 하드디스크는 모든 클러스터 노드가 각각 독립적인 디스크를 보유하고 있고 각각의 물리적인 부트 시스템을 갖추고 있다.

3.2. OS Install

여러종류의 리눅스 버전, 제품들이 있으나 여기서 설치할 제품은 Redhat Linux 7.3 (Valhalla) 를 설치 하기로 한다. 본 HowTo 에서 제공하는 클러스터 제작 기법은 대규모 클러스터 (32 Node 이상) 에서는 그대로 적용이 안될수도 있다. 이유는 기본적으로 Beowolf Family 에서 클러스터 끼리의 File 공유는NFS (Network File System) 를 사용하고 있는데, 대량의 클러스터 를 NFS 로 묶을경우 심각한 I/O 병목현상이 일어나게 된다. 파일시스템과 관련하여 현재 많은 연구가 이뤄지고 있으며 좋은 결과물 들이 많이 나오고 있다. 그러나 여기서는 파일시스템의 세부 관련내용을 다루지 않을것이다. 준비가 되었으면 각 Node 에 Redhat 리눅스를 설치한다. 관리노드가 될 컴퓨터는 설치화면 에서 NFS Server 패키지를 반드시 선택 하도록 한다. 설치에 대해 궁금한것이 있다면 관련문서를 참고하기 바란다. 잘 모르겠으면 Full Install 후 쓸모없는 패키지를 하나씩 제거 하는것을 권장한다.

3.2.1. Bug patching, Software Upgrade

불행하게도 어떤 OS 든지 취약점이나 결함이 없는 것은 없다. Redhat Linux 도 예외는 아니며, 관련 버그와 업데이트 에 대해서 http://www.redhat.com/apps/support/errata/ 에 errata 문서로 상세히 기술해 놓았다. 필자가 반드시 권장하는 것은 gcc 관련 패키지를 change 하는 것이다. Redhat 7.x 대에 들어가서 gcc 관련 버그 리포팅은 이루 헤아릴수 없이 많이 나오고 있다. Redhat 7.3 의 기본 gcc 패키지는 놔두고 anonymous 계정으로 다음의 url 에서 gcc 2.95 버전을 다운 받는다. ftp:// mirror.oops.org/pub/Linux/Redhat/RPMS/7.3/gcc/ 해당 디렉토리의 파일들을 전부 다운 받아서 설치 해야 한다. 또는 http://gcc.gnu.org 에서 gcc 2.95 버전의 RPM 파일들을 다운받아서 설치해도 좋다. 설치하려는 gcc 버전이 현재것 보다 낮기 때문에 설치시 의존성 에러나 버전 에러가 날수 있다. 무시하고 설치하도록 하자.

```
[root@master root]# rpm -Uvh --force --nodeps gcc-2.95.4-3.ix86.rpm
```

명령으로 설치한다.

3.2.2. Tuning

필자의 경험상 서버 OS 의 커널튜닝 이나 최적화 세팅은 기본세팅 되어진 서버 의 성능보다 커다란 성능향상을 가져오지 못한다. (최고의 성능차이를 보인 것이 약 15% 정도였다 그것도 특정기능 에서만..) 물론 어플리케이션의 잘못된 설계나 DB 서버의 불안정한 모델링 등은 심각한 성능장애를 초래하기도 하는데 이런것들은 어디까지나 잘못된 것을 바로잡는 것이지, 튜닝이 아니다. 그걸 바로잡는 것은 여러분 몫이다. 리눅스 커널 튜닝이나 Device 튜닝에 대해서는 내용이 너무 방대하고 또 하드웨어에 특화된 내용들이 있을 수밖에 없기 때문에 여기서는 다루지 않을것이다. 별도의 좋은 문서들이나 책들을 참고하기 바란다.

3.3. Networking

3.3.1. 사설 네트워크 구축

사설 네트워크를 구축하는 이유는 다음과 같다.

- 전체 클러스터 시스템 관리의 편리성 (모든 사용자가 관리노드 에서 작업을 할것 이기 때문에 클러스터 관리가 편이)
- 보안의 편리성 (관리노드 만 보안에 신경쓰면 된다 나머지는 사설망으로 묶여 있어 외부 패킷등에 전혀 영향을 받지 않음)

3.3.2. 관리노드 셋업

OS 설치가 끝났다면 모든 노드에서 작업할수 있는 일반 유저 계정을 하나 만들자. 실제 계정 이름은 어떠한 것이든 상관없다.

```
[root@master root]# useradd -d /home/share micro
```

```
[root@master root]# passwd micro
```

클러스터 노드간 네트워크 세팅을 한다. 필자의 연구실 내부 네트워은 사설망으로 이루어져 있다. 따라서 관리노드 의 IP 주소는 접속가능한 (내부 네트워에서 관리노드로) ip 주소여야 한다. 여기서 주어지는 ip 주소는 필자의 환경에 맞게 세팅된 것 이며 실제로는 어떠한 값이 주어져도 상관없다.

```
[root@master root]# /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:E0:81:20:07:84
inet addr:10.1.1.1  Bcast:10.1.1.255  Mask:255.255.255.0
eth0:0    Link encap:Ethernet  HWaddr 00:E0:81:20:07:85
inet addr:192.168.0.64 Bcast:192.168.0.255 Mask:255.255.255.0
eth1      Link encap:Ethernet  HWaddr 00:E0:81:20:07:85
```

```
inet addr:10.1.2.1  Bcast:10.1.2.255  Mask:255.255.255.0
```

eth0 의 10.1.1.0~254 C class 사설 IP 대역은 노드간 NFS 를 위한 네트웍 인터페이스 이다. 모든 노드는 eh0 를 통해 NFS 서비스를 주고 받을 것이다. eth0:0 은 필자가 임의로 마스커레이딩을 통해 가상 IP 를 준것이며 내부 혹은 외부 사용자 네트웍 환경에서 접속할 IP 주소를 세팅한다. 모든 사용자 및 관리자는 관리노드의 eth0:0 의 네트웍 인터페이스 로 접속 할것이다. eth1 의 10.1.2.0 ~ 254 C class 사설 IP 대역은 계산작업 용 통신을 위한 네트웍 인터페이스 이다. 관리용, NFS 용 통신망과 논리적으로 분리를 시켜 독립적인 인터페이스를 사용하도록 한다. 한편 라우팅은 아래와 같이 설정해 주어야 한다. netconfg 명령이나 xwindow 상에서 netcfg 명령으로 설정할수 있다.

```
[root@master root]# /sbin/route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Met-
ric Ref    Use Iface
192.168.0.0     *               255.255.255.0   U     0      0         0 eth0
10.1.1.0        *               255.255.255.0   U     0      0         0 eth0
10.1.2.0        *               255.255.255.0   U     0      0         0 eth1
127.0.0.0       *               255.0.0.0       U     0      0         0 lo
default         your.gateway    0.0.0.0         UG    0      0         0 eth0
```

/etc/hosts 파일을 편집 한다. 클러스터를 이루고 있는 각노드의 주소와 이름을 정 의 해준다. 이것은 관리노드와 모든 컴퓨팅 노드에 동일하게 설정해 준다.

```
[root@master root]# vi /etc/hosts
# 각 node 용 ip 대역
10.1.2.1        node00          master
10.1.2.2        node01
10.1.2.3        node02
10.1.2.4        node03
10.1.2.5        node04
10.1.2.6        node05
10.1.2.7        node06
127.0.0.1       localhost
# NFS 용 IP 대역
10.1.1.1 nfsserver
10.1.1.2 nfs01
10.1.1.3 nfs02
10.1.1.4 nfs03
10.1.1.5 nfs04
10.1.1.6 nfs05
10.1.1.7 nfs06
```

공유디렉토리 를 설정 한다. 여기선 /home/share 를 공유디렉토리로 설정한다. 다음과 같이 /etc/exports 파일을 편집하자.

```
[root@master root]# cat /etc/exports
/home/share     *(rw)
```

그리고 다음의 명령을 사용하여 nfs 서버를 다시 시작해주자.

```
[root@master root]# /etc/rc.d/init.d/nfs restart
```

마스터 노드에 nfs 데몬이 제대로 떠있는지 확인해 보자. 아래와 비슷하게 나와야 한다.

```
[root@master root]# ps -auex | grep nfs
root      4633  0.0  0.0     0    0 ?        SW   May01   0:00 [nfsd]
root      4634  0.0  0.0     0    0 ?        SW   May01   0:00 [nfsd]
root      4635  0.0  0.0     0    0 ?        SW   May01   0:00 [nfsd]
root      4636  0.0  0.0     0    0 ?        SW   May01   0:00 [nfsd]
root      4637  0.0  0.0     0    0 ?        SW   May01   0:00 [nfsd]
root      4638  0.0  0.0     0    0 ?        SW   May01   0:00 [nfsd]
root      4639  0.0  0.0     0    0 ?        SW   May01   0:00 [nfsd]
root      4640  0.0  0.0     0    0 ?        SW   May01   0:00 [nfsd]
```

nfs 데몬이 떠있지 않다면 nfs 는 RPC(Remote Procedure Call) 을 사용하기 때문에 포트매퍼 (Port mapper) 라는 데몬이 먼저 떠있어야 한다. 이를 확인해 보자.

```
[root@master root]# rpcinfo -p
program vers proto   port
100000    2   tcp    111  portmapper
100000    2   udp    111  portmapper
………
```

이 데몬이 떠 있지 않다면 다음과 같은 명령을 사용하여 portmapper 를 먼저 띄우고 nfs 를 재 시작 해보자.

```
[root@master root]# /etc/rc.d/init.d/portmap start
```

위 두가지 모두 정상적으로 작동한다면 nfs 가 제대로 동작 할것이다. 그러나 그렇 지 않다면 nfs 가 커널에서 지원되지 않는 경우일수 있다. 이를 위해서는 커널 컴 파일을 해주어야 한다. 커널 컴파일시 nfs 관련 옵션들을 모두 커널에 포함 시켜 주어야 한다. 이것은 커널 컴파일에 익숙한 사용자 라면 별 어려움이 없겠지만 그 렇지 않은 사용자라면 KLDP homepage (http://kldp.org)에서 "kernel compile" 이라는 검색어로 검색하면 많은 문서를 찾을수 있을것이다. 이를 참고하자. 커널 컴파일 옵션에서 FileSystem -> Network File Systems a NFS 관련된 옵션은 모두 선택을 하고 컴파일 을 한다. 커널 컴파일을 마쳤으면 커널과 모듈들을 적재하고 재부팅 한후 nfs 를 다시 시작 해 보자.

이상이 없으면 rsh 을 허가한다. 클러스터 내의 모든 노드들간에 rsh 이 가능하도 록 세팅해야 한다. 이것은 병렬 라이브러리 의 하나인 MPI (Message Passing Interface) 가 정상작동하기 위

해 필요하다. rsh 관련 세팅은 관리 노드와 컴퓨팅 노드에 동일하게 설정해 줘야만 한다. /etc/xinetd.d 디렉토리 에서 rsh , rlogin, rexec 등의 파일을 편집한다.

```
[root@master root]# vi /etc/xinetd.d/rsh
service shell
{
        disable = no
socket_type = stream
...
```

위와 같이 disable 란을 no 로 바꿔주자. 해당 파일을 모두 수정했으면 xinetd 데 몬을 재시작 하면 적용이 된다.

```
[root@master root]# /etc/rc.d/init.d/xinetd restart
```

이제 어떤 컴퓨터 들이 자신에게 rsh 이 가능하도록 허가할지를 정의해 줘야 한다. 관련 파일은 .rhosts 파일에 적용하는 법과 /etc/hosts.equiv 파일에 적용하는법 이 있는데 여기서는 /etc/hosts.equiv 에 적용하도록 하겠다.

```
[root@master root]# cat /etc/hosts.equiv
master node00
node01
node02
node03
node04
node05
node06
```

/etc/hosts.equiv 파일이 없으면 만들고 위와 같이 호스트 네임을 적어주자. 다음은 root 유저가 rsh 을 이용하여 로그인 할수 있도록 /etc/securetty 파일을 수정해 주도록 한다.

```
[root@master root]# vi /etc/securetty
vc/1
vc/2
vc/3
……
tty1
tty2
……
rsh
rexec
rlogin
```

위와 같이 마지막 라인에 rsh 관련 명령을 추가 시킨다. 다음으로 노드간 rsh , rlogin 시 패스워드 질의 절차를 생략하기 위해서 /etc/pam.d 디렉토리의 rsh, rlogin, rexec 파일을 수정해주자.

```
  [root@master root]# vi /etc/pam.d/rsh
#%PAM-1.0
auth       sufficient   /lib/security/pam_nologin.so
auth       sufficient   /lib/security/pam_securetty.so
auth       sufficient     /lib/security/pam_env.so
auth       sufficient     /lib/security/pam_rhosts_auth.so
account    sufficient   /lib/security/pam_stack.so service=system-auth
session    sufficient   /lib/security/pam_stack.so service=system-auth
[root@master root]# vi /etc/pam.d/rlogin
#%PAM-1.0
# For root login to succeed here with pam_securetty, "rlogin" must be
# listed in /etc/securetty.
auth       sufficient   /lib/security/pam_nologin.so
auth       sufficient   /lib/security/pam_securetty.so
auth       sufficient     /lib/security/pam_env.so
auth       sufficient   /lib/security/pam_rhosts_auth.so
auth       sufficient   /lib/security/pam_stack.so service=system-auth
account    sufficient   /lib/security/pam_stack.so service=system-auth
password   sufficient   /lib/security/pam_stack.so service=system-auth
session    sufficient   /lib/security/pam_stack.so service=system-auth
[root@master root]# vi /etc/pam.d/rexec
#%PAM-1.0
# For root login to succeed here with pam_securetty, "rexec" must be
# listed in /etc/securetty.
auth       sufficient   /lib/security/pam_nologin.so
auth       sufficient   /lib/security/pam_securetty.so
auth       sufficient     /lib/security/pam_env.so
auth       sufficient   /lib/security/pam_stack.so service=system-auth
account    sufficient   /lib/security/pam_stack.so service=system-auth
session    sufficient   /lib/security/pam_stack.so service=system-auth
```

수정을 끝마쳤으면 xinetd 데몬을 재시작 한다.

```
[root@master root]# /etc/rc.d/init.d/xinetd restart
```

3.3.3. 컴퓨팅노드 셋업

관리노드와 마찬가지로 리눅스를 설치하도록 하자 그리고 각 사용자는 모든 노드에 동일한 계정을 가지고 있어야 한다. 권장하는 방법은 관리노드의 /etc/passwd, /etc/passwd- 파일과 /etc/shadow, /etc/shadow- 파일들, /etc/group , /etc/group- 파일들을 모든 노드에 카피 하는 것이다. 이로써 관리노드에서 추가한 유저계정 들을 모든 노드에서 사용할수 있게 되었다. 다음은 네트워크 설정에 들어가보자. 전체적인 설정은 관리노드와 크게 다를바가 없다.

```
           [root@node01 root]# /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:E0:81:20:07:84
```

```
inet addr:10.1.1.2 Bcast:10.1.1.255  Mask:255.255.255.0
eth1      Link encap:Ethernet  HWaddr 00:E0:81:20:07:85
inet addr:10.1.2.2  Bcast:10.1.2.255  Mask:255.255.255.0
```

eth0 의 10.1.1.0~254 C class 사설 IP 대역은 노드간 NFS 를 위한 네트웍 인터페이스 이다. 모든 노드는 eh0 를 통해 NFS 서비스를 주고 받을 것이다. eth1 의 10.1.2.0 ~ 254 C class 사설 IP 대역은 계산작업 용 통신을 위한 네트웍 인터페이스 이다. 관리용 과 NFS 용 통신망과 논리적으로 분리를 시켜 독립적인 인터페이스를 사용한다. 한편 라우팅은 아래와 같이 설정해 주어야 한다. 기본 게이트웨이 는 관리 노드가 된다. netconfg 명령이나 xwindow 상에서 netcfg 명령으로 설정할수 있다.

```
   [micro@node01 share]$ /sbin/route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Met-
ric Ref    Use Iface
10.1.1.0        *               255.255.255.0   U     0      0        0 eth0
10.1.2.0        *               255.255.255.0   U     0      0        0 eth1
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
default         node00          0.0.0.0         UG    0      0        0 eth1
```

etc/hosts 파일을 편집 한다. 클러스터를 이루고 있는 각노드의 주소와 이름을 정의 해준다. 이것은 관리노드와 모든 컴퓨팅 노드에 동일하게 설정해 준다.

```
[root@master root]# vi /etc/hosts
# 각 node 용 ip 대역
10.1.2.1        node00          master
10.1.2.2        node01
10.1.2.3        node02
10.1.2.4        node03
10.1.2.5        node04
10.1.2.6        node05
10.1.2.7        node06
127.0.0.1       localhost
# NFS 용 IP 대역
10.1.1.1 nfsserver
10.1.1.2 nfs01
10.1.1.3 nfs02
10.1.1.4 nfs03
10.1.1.5 nfs04
10.1.1.6 nfs05
10.1.1.7 nfs06
```

공유디렉토리 를 설정 한다. 여기선 /home/share 를 공유디렉토리로 설정한다. 이미 관리노드에서는 모든 node 가 관리노드 자신의 /home/share 를 마운트 하는것을 허락했다. 따라서 컴퓨팅 노드 에서는 관리노드의 /home/share 공간을 /etc/fstab 에 추가해 주도록 하자.

```
   [root@node01 /]# vi /etc/fstab
…………
node00:/home/share      /home/share             nfs     bg,intr,noac    0 0
```

관리노드 node00 의 /home/share 를 자신의 /home/share 로 마운트 하 는 명령이다. 파일시스템은 nfs 이고 nfs 마운팅 옵션은 bg, intr, noac 으 로 세팅 해준다. bg 는 만약 첫번째 NFS 마운트 시도가 타임아웃에 걸리면 백그라운드 에서 실행을 계속한다. noac 옵션은 반드시 설정해줘야 한다. MPI 라이브러리 (특히 MPICH) 에서 NFS V3 와 I/O 관련 충돌 문제가 리포팅 되었는데, 관련 버그를 막기 위해 noac 옵션이 필요하다. 이것은 NFS 의 속성 캐쉬를 해제하는 것으로 서버 효율을 떨어드리기는 하지만 두개의 다른 NFS 클라이언트로 하여금 서버상의 공통 파일 시스템에 쓰기 작업을 할때 좋은 효율을 얻을수 있게 해준다. 이제 /home/share 공유디렉토리 를 마운팅 해보자.

```
   [root@node01 /]# mount -t nfs node00:/home/share /home/share
```

이상없이 마운팅이 되었으면 rsh 셋업에 들어가도록 한다. rsh 세팅은 기 본적으로 관리노드와 다르지 않다. rsh 관련 셋업이 완료 되었으면 이제 각 호스트 간에는 rsh 명령이 가능해 질것이다. 다음은 rsh 이 정상적으로 동작 하는지 확인해 보도록 한다. 관리 노드에서 컴퓨팅 노드로 rsh 이 되는지 확인해 보자.

```
   [micro@master micro]$ rsh node01
Last login: ………
[micro@node01 micro]$
```

컴퓨팅 노드에서 관리노드로 rsh 이 되는지 확인해 보자.

```
   [micro@node01 micro]$ rsh master
Last login: ………
[micro@master micro]$
```

3.3.4. 시간 동기화

클러스터 간 시간을 동기화 한다는건 매우 중요한 일이다. 어떠한 이유가 있더라도 클러스터간에 시간을 맞춰줘야 한다. 모든 시스템의 시각을 현재시각으로 맞춘다. 그 다음 그 시각을 CMOS 실시간 시간으로 "hwclock -w" 을 이용 변경한다. 시간동기화의 좋은 패키지중 하나는 수학자 출신인 D.J.Bernstein 이 만든 clockspeed 가 있다. 자세한 내용은 다음에서 확인할수 있다. http://cr.yp.to/clockspeed.html 혹은 http://kldp.org/~eunjea/clockspeed.php 위의 url 에서 clockspeed 소스를 다운로드 한다.

```
[root@master opt]# wget http://cr.yp.to/clockspeed/clockspeed-0.62.tar.gz
………………
[root@master opt]# tar xzf clockspeed-0.62.tar.gz
```

```
[root@master opt]# cd clockspeed-0.62
[root@master opt]# make
[root@master opt]# make setup check
```

파일들은 /usr/local/clockspeed 에 설치된다. 실행파일들은 /usr/local/clockspeed/bin 에 설치되므로 사용자의 $PATH 에 추가해 주도록 한다. 다음은 실제 동기화 하는 과정이다. 먼저 ntp 서버와 클러스터서버 와의 시간 동기화를 한다. 여기서는 ntp 서버로 ntp.nasa.gov 를 이용했다.

```
[root@master /]# nslookup
…………………
> ntp.nasa.gov
…………………
Non-authoritative answer:
ntp.nasa.gov    canonical name = ntp-nasa.arc.nasa.gov.
Name:   ntp-nasa.arc.nasa.gov
Address: 198.123.30.132   庠雰---- ntp 서버 주소
[root@master /]# sntpclock 198.123.30.132 | clockadd
```

이제 ntp 서버와 클러스터 와의 시간을 동기화 했다. 이제는 좀전에 ntp 서버와 동기화를 맞춘 master 서버와 나머지 클러스터들 간의 동기화 작업을 하도록 하자. 동기화의 기준이 되는 서버는 master 서버 이다. 기본적인 실행은 master 서버에서 taiclockd 서버 데몬을 실행시켜 주고 나머지 서버에서 taiclock 클라이언트 를 사용하여 동기화 를 시켜주는 방식이다.

```
[micro@master /]$ taiclockd &   庠雰-- master 서버 (root 권한으로 실행하지 않아도 됨)
[micro@node01 /]$ taiclock 10.1.2.1 > /tmp/adjust   庠雰-- 클라이언트 서버
[micro@node01 /]$ clockview < /tmp/adjust  庠雰- 시간차 확인
before: 2003-05-06 10:50:05.824066000000000000
after:  2003-05-06 10:50:32.290600000000000000
```

약 23초 정도 차이가 나고 있다. 시간을 맞춰주자.

```
[micro@node01 /]$ clockadd < /tmp/adjust
[root@node01 /]# /sbin/hwclock -w
```

위의 과정을 클러스터 시스템 에서 정기적으로 실행하면 된다. 필자는 master 서버 에 다음과 같이 shell script 를 만들어 cron 을 사용하여 정기적으로 클러스터간 시간을 동기화 하고 있다.

```
[root@master bin]# vi /root/bin/sync_nodes_time.sh
#!/bin/sh

#
# First, correct time of this machine
#
/usr/local/clockspeed/bin/sntpclock 198.123.30.132 | \
```

```
/usr/local/clockspeed/bin/clockadd
/sbin/hwclock -w
#
# Next, sync time of each nodes
#

echo Start taiclockd ...
/usr/local/clockspeed/bin/taiclockd &

echo Correct time of node01
rsh node01 '/usr/local/clockspeed/bin/taiclock 10.1.3.1 > /tmp/adjust ; \
                /usr/local/clockspeed/bin/clockadd < /tmp/adjust; \
                /sbin/hwclock -w'
echo Correct time of node02
rsh node02 '/usr/local/clockspeed/bin/taiclock 10.1.3.1 > /tmp/adjust ; \
                /usr/local/clockspeed/bin/clockadd < /tmp/adjust; \
                /sbin/hwclock -w'
echo Correct time of node03
rsh node03 '/usr/local/clockspeed/bin/taiclock 10.1.3.1 > /tmp/adjust ; \
                /usr/local/clockspeed/bin/clockadd < /tmp/adjust; \
                /sbin/hwclock -w'
echo Correct time of node04
rsh node04 '/usr/local/clockspeed/bin/taiclock 10.1.3.1 > /tmp/adjust ; \
                /usr/local/clockspeed/bin/clockadd < /tmp/adjust; \
                /sbin/hwclock -w'
echo Correct time of node05
rsh node05 '/usr/local/clockspeed/bin/taiclock 10.1.3.1 > /tmp/adjust ; \
                /usr/local/clockspeed/bin/clockadd < /tmp/adjust; \
                /sbin/hwclock -w'
echo Correct time of node06
rsh node06 '/usr/local/clockspeed/bin/taiclock 10.1.3.1 > /tmp/adjust ; \
                /usr/local/clockspeed/bin/clockadd < /tmp/adjust; \
                /sbin/hwclock -w'

echo Stop taiclockd ...
killall taiclockd

echo done.
# end of file
```

3.4. Parallel Library

3.4.1. 설치하기에 앞서

병렬 라이브러리 에 사용되는 Middle ware 는 클러스터 시스템일 경우 크게 PVM(Parallel Virtual Machine) 과 MPI (Message Passing Interface) 로 나뉠수 있다. 양쪽 모두 관련문서가 상당히 많이 있으며 한가지 차이점을 꼽으면 PVM 은 이기종간의 시스템에서 돌아간다는 점과 MPI 는 동기종간의 클러스터 에서 고려된 라이브러리 라는 점이다. PVM VS MPI 에 관련된 많은 논쟁거리가 꽤 오래전부터 진행되어 왔고 아직도 논쟁이 벌어지는 곳도 있다. 여기서는 동기종 간의 클러스터 시스템으로 꾸밀것 이므로 MPI 를 설치 할것이다. MPI 는 병렬 라이브러리의 사실상의 표준으로 자리잡았다. 대표적인 MPI 라이브러리 에는 LAM-MPI 와 MPICH 가 많이 사용된다. 기본적으로 MPI 의 통신은 rsh 을 이용하며 병렬 프로세서간 점대점 (Point-to-Point) 통신을 한다. MPI 를 이용하여 실행할 병렬 프로그램은 논리적, 혹은 물리적으로 클러스터 각 노드간 동일한 디렉토리에 존재해야 한다.

3.4.2. LAM-MPI

LAM-MPI 를 설치해 보자. http://www.lam-mpi.org 에서 lam 최신버전을 받아오자. 이글을 쓰는 시점에서는 6.5.9 가 최신버전 이다. 참고로 새로운 버전이 나올때 설치 방법이 변경될수 있으므로 다운받은 파일에 있는 설치문서를 참고하기 바란다.

```
[root@master root]# wget http://www.lam-mpi.org/download/files/lam-
6.5.9.tar.gz
--15:07:35--  http://www.lam-mpi.org/download/files/lam-6.5.9.tar.gz
           => 'lam-6.5.9.tar.gz'
Resolving www.lam-mpi.org... done.
Connecting to www.lam-mpi.org[129.79.245.239]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2,184,991 [application/x-gzip]

100%[===========================================>] 2,184,991    197.43K/s    ETA 00:00
15:07:52 (197.43 KB/s) - 'lam-6.5.9.tar.gz' saved [2184991/2184991]
[root@master root]# tar xzf lam-6.5.9.tar.gz
[root@master root]# cd lam-6.5.9
[root@master root]# ./configure --prefix=/usr/local/lam
………………
[root@master root]# make
………………
[root@master root]# make install
………………
```

이제 lam-mpi 설치가 끝났다. 그러나 실행을 위해서 몇가지 설정이 남아 있다. 사용자가 LamMPI 를 사용하기 위해서는 먼저 몇가지 환경변수를 설 정해 줘야 한다. 모든 사용자가 동일하게 적용 받도록 /etc/bashrc 나 /etc/profile 에 다음을 추가해 주자.

```
[root@master root]# vi /etc/bashrc
………………
LAMHOME=/usr/local/lam
PATH=$PATH:/usr/local/lam/bin
export LAMHOME PATH
```

이제 모든 사용자가 로그인 할때 LAMHOME 과 PATH 가 적당하게 설정 될 것이다. 이제 Lam 을 실행해 보도록 하자. Lam team 에서 권장하는 방법은 머신네임 이 기술되어 있는 파일을 만들어서 lamd 데몬과 동시에 부팅 시키는 것이다.

```
[root@master root]# vi /etc/lamhosts
node00
node01
………
node06
```

위와 같이 lamhosts 파일을 만들어서 lamd 의 시동 프로그램인 lamboot 에 파라메터를 넘겨주도록 하자. Lamd 데몬의 실행방법은 다음과 같다. Root 계정이 아닌 일반 계정으로 실행해야 한다.

```
[micro@master micro]$ lamboot -v -b /etc/lamhosts
LAM 6.5.9/MPI 2 C++/ROMIO - Indiana University

Executing hboot on n0 (node00 - 1 CPU)...
Executing hboot on n1 (node01 - 1 CPU)...
Executing hboot on n2 (node02 - 1 CPU)...
Executing hboot on n3 (node03 - 1 CPU)...
Executing hboot on n4 (node04 - 1 CPU)...
Executing hboot on n5 (node05 - 1 CPU)...
Executing hboot on n6 (node06 - 1 CPU)...
topology done
```

위와 같은 메시지가 나와야 한다. 그리고 실행되어 있는 lamd 의 상태를 보려면 lamnodes 로 확인해 보면 된다.

```
[micro@master micro]$ lamnodes
n0      node00:1
n1      node01:1
n2      node02:1
n3      node03:1
n4      node04:1
n5      node05:1
n6      node06:1
```

lamd 데몬을 실행해 봤으니 shutdown 도 해보도록 하자. Lamd 데몬의 중지는 wipe 의 실행으로 가능하다.

```
[micro@master micro]$ wipe -v -b /etc/lamhosts
LAM 6.5.9/MPI 2 C++/ROMIO - Indiana University

Executing tkill on n0 (node00)...
Executing tkill on n1 (node01)...
Executing tkill on n2 (node02)...
Executing tkill on n3 (node03)...
Executing tkill on n4 (node04)...
Executing tkill on n5 (node05)...
Executing tkill on n6 (node06)...
```

Lam-mpi 를 이용하여 병렬 프로그램을 실행시켜 보자. Lamd 데몬을 띄운다. 필자는 간단한 shell script 를 만들어서 /etc/rc.d/init.d/ 디렉토 에 위치시키고 runlevel 이나 command 상에서 lamd 데몬을 start, stop 할수 있도록 했다. 다음은 해당 script 파일이다. 주의할 점은 여러분이 실 행하는 클러스터 환경과 필자가 구현한 환경이 서로 다를수 있으니 사용자 환경에 맞게 약간의 수정을 가해야 할것이다.

```
[micro@master micro]$ vi /etc/rc.d/init.d/lamd
#! /bin/sh
#

# PATH your lam-mpi
#export PATH=$PATH:/usr/local/lam/bin

# Source function library.
. /etc/init.d/functions

# See how we were called.

case "$1" in
  start)
        echo -n "Starting lamd"
        if [ "$UID" == "500" ];then   # lam 을 실행할 일반 유저계정의 UID
                lamboot -v -b /etc/lamhosts
        else
                su - micro -c 'lamboot -v -b /etc/lamhosts'
        fi
        echo
        ;;
  stop)
        echo -n "Stopping lamd"
        if [ "$UID" == "500" ];then
                wipe -v -b /etc/lamhosts
        else
                su - micro -c 'wipe -v -b /etc/lamhosts'
```

```
        fi
        echo
        ;;
  status)
        echo -n "shows status nodes..."
        if [ "$UID" == "500" ];then
                lamnodes
        else
                su - micro -c 'lamnodes'
        fi
        echo
        ;;
  restart)
        $0 stop
        $0 start
        ;;
  *)
        echo $"Usage: $0 {start|stop|status|restart}"
        exit 0
esac

exit $?
[micro@master micro]$ /etc/rc.d/init.d/lamd start
Starting lamd
LAM 6.5.9/MPI 2 C++/ROMIO - Indiana University

Executing hboot on n0 (node00 - 1 CPU)...
Executing hboot on n1 (node01 - 1 CPU)...
Executing hboot on n2 (node02 - 1 CPU)...
Executing hboot on n3 (node03 - 1 CPU)...
……………………
```

다음은 클러스터 프로세서 간에 간단한 메시지 전달을 주고받는 C 로 만든 MPI 프로그램 이다

```
/*
 * Laboratory for Scientific Computing
 * http://www.lam-mpi.org/tutorials/
 * University of Notre Dame
 *
 * MPI Tutorial
 * The cannonical ring program
 *
 * Mail questions regarding tutorial material to mpi@lam-mpi.org
 */

#include <stdio.h>
```

```
#include "mpi.h"    /* MPI Header 를 인클루드 해야함 */

int main(int argc, char *argv[])
{
  MPI_Status status;
  int num, rank, size, tag, next, from;

  /* Start up MPI */

  MPI_Init ( ,  );
  MPI_Comm_rank(MPI_COMM_WORLD, );
  MPI_Comm_size(MPI_COMM_WORLD, );

  /* Arbitrarily choose 201 to be our tag.  Calculate the */
  /* rank of the next process in the ring.  Use the modulus */
  /* operator so that the last process "wraps around" to rank */
  /* zero. */

  tag = 201;
  next = (rank + 1) % size;
  from = (rank + size - 1) % size;

  /* If we are the "console" process, get a integer from the */
  /* user to specify how many times we want to go around the */
  /* ring */

  if (rank == 0) {
    printf("Enter the number of times around the ring: ");
    scanf("%d", #) ;

    printf("Process %d sending %d to %d\n", rank, num, next);
    MPI_Send(#, 1, MPI_INT, next, tag, MPI_COMM_WORLD);
  }

  /* Pass the message around the ring.  The exit mechanism works */
  /* as follows: the message (a positive integer) is passed */
  /* around the ring.  Each time is passes rank 0, it is decremented. */
  /* When each processes receives the 0 message, it passes it on */
  /* to the next process and then quits.  By passing the 0 first, */
  /* every process gets the 0 message and can quit normally. */

  while (1) {

    MPI_Recv(#, 1, MPI_INT, from, tag, MPI_COMM_WORLD, );
    printf("Process %d received %d\n", rank, num);

    if (rank == 0) {
      num--;
```

```
      printf("Process 0 decremented num\n");
    }

    printf("Process %d sending %d to %d\n", rank, num, next);
    MPI_Send(#, 1, MPI_INT, next, tag, MPI_COMM_WORLD);

    if (num == 0) {
      printf("Process %d exiting\n", rank);
      break;
    }
  }

  /* The last process does one extra send to process 0, which needs */
  /* to be received before the program can exit */

  if (rank == 0)
    MPI_Recv(#, 1, MPI_INT, from, tag, MPI_COMM_WORLD, );

  /* Quit */

  MPI_Finalize();
  return 0;
}
```

앞서도 얘기했듯이 병렬로 실행할 프로그램은 논리적 이던 물리적이던 프로그램을 실행할 각각 클러스터 간에 동일한 위치(Path) 에 존재해야 한다. 클러스터간에 NFS, 혹은 다른 가상 파일시스템 으로 공유중인 위치에 프로그램을 위치시키거나 혹은 특정 디렉토리에서 실행할 경우 모든 노드간에 동일한 위치에 카피를 한후 실행을 하면된다. 이제 컴파일을 해보자. gcc 나 egcs 대신 lam-mpi 의 mpicc 를 이용해야 한다.

```
[micro@master micro]$ mpicc -o ring MPI_C_SAMPLE.c
```

이제 컴파일한 병렬프로그램을 돌려보도록 하자. 프로그램 실행은 mpirun 명령으로 실행하며 mpirun 에는 여러가지 옵션이 있다. 관련 내용은 man mpirun 이나 mpirun --help 명령으로 알수가 있다.

```
[micro@master micro]$ mpirun -O -np 2 ./ring
Enter the number of times around the ring: 150
……………………
Process 1 received 3
Process 1 sending 3 to 0
Process 0 received 3
Process 0 decremented num
Process 0 sending 2 to 1
Process 1 received 2
Process 1 sending 2 to 0
Process 0 received 2
```

```
Process 0 decremented num
Process 0 sending 1 to 1
Process 1 received 1
Process 1 sending 1 to 0
Process 0 received 1
Process 0 decremented num
Process 0 sending 0 to 1
Process 0 exiting
Process 1 received 0
Process 1 sending 0 to 0
Process 1 exiting
```

lam-mpi 의 mpirun 의 -O 옵션은 프로그램을 실행할 클러스터를 기본적으로 homogeneous system (동기종) 으로 가정한다. 따라서 메시지 패싱하는데 있어서 별도의 transaction 절차를 거치지 않는다. Low type 으로 직접 통신하는 것이다. -np 2 는 프로세서의 수를 2개로 실행한다는 것이다. 해당프로그램을 2 node 시스템(CPU 가 1개 일경우) 에서 실행한다는 것을 뜻한다. 이상없이 프로그램이 돌아갔다면 기본적인 클러스터 시스템이 구성된것이다. 같은 프로그램을 여러분의 클러스터 노드 개수에 맞게도 돌려보도록 하자.

마지막으로 lam-mpi 의 관련된 여러정보는 man 을 이용 mpirun , lamclean , lamboot 등의 필요한 정보를 얻을수 있으며 또한 http://www.lam-mpi.org 에서 무수히 많은 정보를 얻을수 있다. (FAQ 나 Tutorial 등은 한번씩 읽어보기를 권장한다)

3.4.3. MPICH

MPICH (MPI Chameleon) 을 설치해 보자. 앞서 얘기했듯이 가장 많이 사용되고 있는 MPI 라이브러리 중 하나가 바로 MPICH 이다. 다양한 하드웨어를 지원하며, 각종 시스템 퍼포먼스 벤치마크 프로그램의 척도로 도 많이 사용되고 있다. 많은 분량의 유저 Guide 와 매뉴얼이 있으며 MPICH 를 사용하게 된다면 최소한 적어도 CH_P4MPD(http://www.mcs.anl.gov/mpi/mpich/docs/mpichman-chp4mpd/mpichman-chp4mpd.htm) Device 에 관련된 Manual (대부분의 ix86 시스템에 해당한다. 이 문서에서는 ch_p4mpd device 를 선택하여 설치 할것이다.) 과 FAQ (http://www-unix.mcs.anl.gov/mpi/mpich/docs/faq.htm) 를 읽어보기를 권장한다. 준비 되었으면 소스파일을 다운로드 하자.

```
[micro@master share]$ wget ftp://ftp.mcs.anl.gov/pub/mpi/mpich.tar.gz
```

앞축을 풀고 설치를 한다. 여기서는 /usr/local/mpich-1.2.5 에 설치를 할것이다.

```
[micro@master share]$ tar xzf mpich.tar.gz
[micro@master share]$ cd mpich-1.2.5
[micro@master mpich-1.2.5]$ ./configure --with-device=ch_p4mpd --
prefix=/usr/local/mpich-1.2.5
[micro@master mpich-1.2.5]$ make; make install
```

이상없이 설치가 되었다면 /usr/local/mpich-1.2.5/share/machines.LINUX 에 사용할 호스트네임 을 적어준다.

```
[micro@master mpich-1.2.5]$ vi /usr/local/mpich-1.2.5/share/machines.LINUX
node00
node01
node03
………
node06
```

나머지 클러스터 에서도 같은 옵션으로 설치를 한다. 모든 클러스터에 설치가 되었다면 간단한 sample 프로그램을 돌려보도록 하자. Mpich 를 이용하여 병렬프로그램을 실행하려면 LAM-MPI 와 마찬가지로 MPICH 전용 데몬인 MPD 가 먼저 떠있어야 한다. MPD 를 실행시키자. 일반계정으로 실행시켜도 된다.

```
[micro@master share]$ /usr/local/mpich-1.2.5/bin/mpd &
```

마스터 서버에서 mpd 를 실행시키고 나머지 서버에서는 실행방법이 약간 다르다. 먼저 mpd 데몬의 소켓 번호를 알아야 한다. 다음의 명령으로 알 수가 있다.

```
[micro@master share]$ /usr/local/mpich-1.2.5/bin/mpdtrace
mpdtrace: mas-
ter_33253: ihs=node00_33253  rhs=node00_33253  rhs2=node00_33253 gen=2
```

위에서 보면 mpd 데몬이 33253 소켓으로 바인딩 되었다는 것을 알수있다. 나머지 클러스터에서도 mpd 데몬을 실행시켜 준다. Mpd 데몬끼리의 소 켓 통신을 위해 마스터 서버와 연동한다.

```
[micro@master share]$ rsh node01
[micro@node01 share]$ /usr/local/mpich/bin/mpd -h master -p 33253 &
```

-h 옵션은 클러스터 마스터 서버의 hostname 을 적어주면 되며, -p 옵션 으로 소켓번호를 적어주면 된다. Node01 뿐 아니라 나머지 클러스터에서 도 mpd 를 실행시킨후 예제로된 병렬프로그램을 실행해보도록 하자. 예제 프로그램은 mpich examples 에 있는 원주율을 구하는 간단한 프로그램 이다.

```
[micro@master share]$ cd mpich-1.2.5/examples/basic
[micro@master basic]$ make cpi
[[micro@master basic]$ /usr/local/mpich-1.2.5/bin/mpirun -np 4 ./cpi
Process 0 of 4 on node00
Process 1 of 4 on node06
Process 2 of 4 on node05
Process 3 of 4 on node04
pi is approximately 3.1415926544231239, Error is 0.0000000008333307
wall clock time = 0.002665
```

위와 비슷한 결과가 나와야 한다. 만일 올바른 결과가 나오지 않거나 에러 가 발생했다면, mpich 매뉴얼을 참고를 하거나, Case of Trouble (http://www-unix.mcs.anl.gov/mpi/mpich/docs/mpichman-chp4mpd/node59.htm#Node59) 을 참고하도록 하자.

4장. Benchmark

4.1. 소개

현재 만들어진 시스템 상에서 각종 기준이 될수있는 퍼포먼스를 측정하는 것은 매우 중요한 일이다. Benchmark 는 시스템의 성능을 체크해본다는 중요성도 있지만 더 나아가 향후 System 을 Upgrade 해 나가는 과정중에 성능이 실제로 향상되었느냐는 관점에서 중요한 지침이 되는것이다. 물론 자신이 사용하고 있는 병렬 Application 을 돌려보는 일은 우선적이다. 그러나 그밖에 일반적인 공개/표준 Benchmark Program 을 돌려본다면 다른 Cluster 시스템과 상호 성능비교를 할 수 있는 중요 자료를 확보 하는것이다. 흔히 쉽게구할수 있는 대표적인 병렬 Benchmark Program 들은 다음과 같다.

4.2. NetPIPE (http://www.scl.ameslab.gov/Projects/NetPIPE/) (Network Protocol Independent Performance Evaluator)

네트웍 퍼포먼스 측정 벤치마크 프로그램. Ethernet 카드, 혹은 네트웍 인터페이스 간에 간단한 Ping-Pong 테스트를 하여 메시지 Size 를 변경해가면서 다양한 측정을 한다. 여러종류의 하드웨어를 지원하며, PVM, MPI , MPI-2 등 병렬 소프트웨어와 네트웍 퍼포먼스 의 병합측정도 가능하다.

4.3. NPB (http://www.nas.nasa.gov/Software/NPB/) (Nas Parallel Benchmark)

CFD(Computational Fluid Dynamics) 어플리케이션 에 관련한 8개의 다양한 Benchmark Test. 네트웍 성능뿐 아니라 Computational 성능 등 Cluster 의 다양한 성능들을 종합적으로 가늠해 볼수 있는 의미있는 Benchmark Program.

4.4. SCALAPACK (http://www.netlib.org/scalapack/) (Scalable LAPACK)

SCALAPACK 은 여러 선형대수 Math Library 들로 이루어진 벤치마크 프로그램이다. 특히 이것은 전세계 슈퍼컴퓨터 랭킹(http://www.top500.org) 을 측정할 때 쓰여지는 벤치마크 로써

Network 성능보다는 Computing 성능이 주로 반영된다.

4.5. Install & Run

이제 여러분의 클러스터 시스템에 벤치마크 프로그램을 설치해보고 직접 성능을 체크 해보자. 이 문서에서는 NetPIPE 와 SCALAPACK 을 설치, 테스트 해볼것이다.

4.5.1. 테스트 사양

- CPU : AMD MP 1900+ 1.5Ghz 256KB Cash (SINGLE)
- RAM : 2Giga PC 2100 ECC Memory
- HDD : 10000 RPM SCSI
- NIC : 3Com Gigabit Card, 3Com 10/100 Lancard

4.5.2. NetPIPE

NetPIPE 는 머신간에 혹은 에더넷 카드간에 네트웍 성능을 측정할수 있는 벤치마크 프로그램 이다. PVM 과 MPI 둘다 지원하고 있으며 여러 기종의 하드웨어 도 지원하고 있다. 인스톨을 해보자. 다음의 url 에서 소스를 다운받는다. 이글을 쓰는 현재 NetPIPE 의 최신버전은 3.3 이다. http://www.scl.ameslab.gov/netpipe/ 클러스터간 공유디렉토리 (필자의 환경에서는 /home/share) 에서 작업을 하는 것이 좀더 편리하다.

```
[micro@master share]$ tar xzf NetPIPE_3.3.tar.gz
[micro@master share]$ cd NetPIPE_3.3
```

Makefile 을 편집하도록 하자. 대부분 손댈부분은 거의 없고 MPI 의 설정 부분만 편집해주면 된다. MPICC 컴파일러 를 해당 사용자의 시스템 에 맞 는 컴파일러를 지정해주면 된다. 즉 LAM-MPI 로 NetPIPE 를 벤치마킹 할경우는 LAM 의 mpicc 를 지정해주면 되고 MPICH 의 성능을 테스트 해보려면 MPICH 의 mpicc 를 지정해주면 된다.

```
[micro@master NetPIPE_3.3]$ vi makefile
# For MPI, mpicc will set up the proper include and library paths
MPICC       = /usr/local/mpich/bin/mpicc     # 庠  이부분
…………………………………….
MPI2CC    = /usr/local/mpich/bin/mpicc # 庠  이부분
```

수정을 했으면 컴파일을 해보도록 한다. 단순히 tcp 의 성능을 테스팅 해 보기 위해선 make tcp 로 컴파일 하면 된다.

```
[micro@master NetPIPE_3.3]$ make tcp
```

기본적으로 NetPIPE 의 퍼포먼스 측정은 양방향 ping-pong 테스트 이다. Tcp 성능 벤치마킹을 하는경우 한쪽에선 receiver 가 되고 다른 한쪽에 선 sender 로 실행시켜 주면 된다.

```
[micro@master NetPIPE_3.3]$ ./NPtcp -r &
[micro@master NetPIPE_3.3]$ rsh node01
[micro@node01 NetPIPE_3.3]$ ./NPtcp -t -h master
Send and receive buffers are 512000 and 512000 bytes
(A bug in Linux doubles the requested buffer sizes)
Now starting the main loop
  0:       1 bytes    500 times -->      0.10 Mbps in      78.87 usec
  1:       2 bytes   1267 times -->      0.19 Mbps in      78.89 usec
  2:       3 bytes   1267 times -->      0.29 Mbps in      78.84 usec
  3:       4 bytes    845 times -->      0.39 Mbps in      79.14 usec
4:       6 bytes    947 times -->      0.58 Mbps in      79.00 usec
……………………………………
123: 8388611 bytes      3 times -->    505.81 Mbps in  126529.34 usec
```

필자의 환경에서는 PEAK 결과가 다음과 같이 나왔다.

표 **4-1. NetPIPE** 벤치마크 결과 **(tcp)**

Message Size	Bandwidth	Latency (Usec)
8.3 Mbytes	505 Mbps	126529

이제 MPI 를 이용한 네트웍 성능을 측정해 보도록 하자. 참고로 이 문서에서는 lam-mpi 의 mpi 성능을 측정해 볼것이다. 앞서 makefile 에 mpicc 를 수정해 줬으면 컴파일을 하도록 한다.

```
[micro@master NetPIPE_3.3]$ make mpi
```

Npmpi 프로그램이 컴파일 되었을 것이다. Lam-mpi 의 mpirun 으로 실행 해 보도록 하자.

```
[micro@master NetPIPE_3.3]$ mpirun -O -np 2 ./Npmpi
0: master
1: node01
0:       1 bytes    500 times --> 0.09 Mbps in      81.29 usec
1:       2 bytes   1230 times -->      0.19 Mbps in      81.40 usec
2:       3 bytes   1228 times -->      0.28 Mbps in      81.38 usec
3:       4 bytes    819 times -->      0.38 Mbps in      81.16 usec
……………………………………
123: 8388611 bytes      3 times -->    504.27 Mbps in  126917.33 usec
```

lam-mpi 의 mpi 네트웍 최대 성능은 다음과 같이 나왔다.

표 **4-2. NetPIPE** 벤치마크 결과 **(mpi)**

Message Size	Bandwidth	Latency (Usec)
8.3 Mbytes	508 Mbps	94347

마찬가지로 mpich 의 mpi 네트웍 퍼포먼스도 한번 테스트 해보기 바란다.

4.5.3. SCALAPACK (LINPACK Benchmark)

SCALAPACK 은 선형대수 의 해를 구하는 패키지로 SCALAPACK 에 기본적으로 포함되어 있으며 많은 부분이 부동소수점 연산으로 구성되어 있다. LINPACK 벤치마크 에서 중점적으로 사용되는 루틴들은 Gauss 소거법을 이용한 N 개의 선형방정식 의 해를 구하는 것으로 BLAS (Basic Linear Algebra Subprograms) 에 포함되어 있다. BLAS 는 LINPACK 벤치마크 에서 가장 기본이 되는 라이브러리 로써 기본적인 선형대수 연산함수 들을 구현해놓은 집합이다. 이것은 Fortran 으로 짜여져 있으며 BLAS 라이브러리 내의 각 함수들은 연산자와 연산결과가 Vector 냐 Matrix 냐 에 따라 계산 레벨이 나뉘어 진다. 이 BLAS 를 이용해 벤치마킹을 할수도 있지만 ATLAS (Automatically Tuned Linear Algebra Software) 를 이용하여 해당 플랫폼에 최적화된 루틴 라이브러리 를 생성할수도 있다. 이 문서에서는 BLAS, BLACS, ATLAS, SCALAPACK 을 이용하여 벤치마킹을 할것이다. 여기서는 설치 및 실행의 간결한 Guide 만을 제시할 예정이며 이 문서 외의 기타 자세한 내용은 SCALAPACK 의 홈페이지 http://www.netlib.org/scalapack/ 이나 한국클러스터 기술센터(http://www.cluster.or.kr/board/read.php?table=benchmark) 의 벤치마크 Guide 를 참고하기 바란다.

4.5.3.1. LINPACK

http://www.netlib.org/blas/blas.tgz 에서 BLAS 를 다운 받아서 설치한다.

```
[micro@master share]$ mkdir BLAS
[micro@master share]$ cd BLAS
[micro@master share]$ tar xzf blas.tgz
```

컴파일 한다. 참고로 해당 프로세서에 최적화된 컴파일러 를 사용하면 성능향상을 볼수 있다 (Intel 의 pgcc 나 Compaq 의 ccc 등등..)

```
[micro@master share]$ f77 -c *.f
```

생성된 오브젝트 파일들 (확장자가 *.o) 을 라이브러리 로 만든다.

```
[micro@master share]$ ar cr blas_LINUX.a *.o
```

BLACS (Basic Linear Algebra Communication Subprograms) 설치. BLACS 는 다양한 분산메모리 환경에서 프로세서간 메시지 통신을 위 한 선형대수 라이브러리 이다. PVM 과 MPI 용이 따로 있으므로 필요 한 파일을 다운받는다. 여기서는 MPI 를 사용하므로 http://www.netlib.org/blacs/ 에서 mpiblacs.tgz 와 blacstester.tgz 를 다운 받는다. 다음 mpiblacs.tgz 를 압축을 풀면 BLACS 디렉토리가 생긴다.

```
[micro@master share]$ tar xzf mpiblacs.tgz
[micro@master share]$ tar xzf blacstester.tgz BLACS/TESTING/*
[micro@master share]$ cd BLACS
```

BMAKES 디렉토리에서 머신에 맞는 Bmake 파일을 BLACS 디렉토리 로 복사한다.

```
[micro@master BLACS]$ cp BMAKES/Bmake.MPI-LINUX ./Bmake.inc
```

Bmake.inc 파일을 편집한다. Bmake.inc 파일은 3가지 Section 으로 정의되어 있으며 각 섹션은 컴파일 과정에서 필요한 여러 매크로를 정 의 한다. 섹션 1 에서는 라이브러리와 실행파일의 위치를 지정하고 make 결과로 생성되는 파일의 이름을 지정할 때 이용하는 매크로를 정의한다. Section 2 에서는 BLACS 에서 이용하는 C Preprocessor 값 을 정의한다. Section 3 에선 컴파일러와 링커/로더 를 설정하는 매크로 를 정의한다.

```
[micro@master BLACS]$ vi Bmake.inc

#============ SECTION 1: PATHS AND LIBRARIES ======================
SHELL = /bin/sh <- 사용할 쉘의 종류

BTOPdir = $(HOME)/BLACS   <- BLACS 의 Top Level 디렉토리

COMMLIB = MPI <- 사용할 communication 라이브러
리 CMMD,  MPI, PVM, MPL, NX 중 하나

PLAT = LINUX                <- 플렛폼

BLACSdir    = $(BTOPdir)/LIB <- BLACS 라이브러리의 위치
BLACSDBGLVL = 1 <- 디버깅 레벨 (0 = NO, 1 = YES)
BLACSFINIT  = $(BLACSdir)/blacsF77init_$(COMMLIB)-$(PLAT)-$(BLACSDBGLVL).a
BLACSCINIT  = $(BLACSdir)/blacsCinit_$(COMMLIB)-$(PLAT)-$(BLACSDBGLVL).a
BLACSLIB    = $(BLACSdir)/blacs_$(COMMLIB)-$(PLAT)-$(BLACSDBGLVL).a
<- 라이브러리 이름들

MPIdir = /usr/local/mpich <- MPICH 의 위치
MPIdev = ch_p4mpd <- MPICH Device 의 종류
MPIplat = LINUX
MPILIBdir = $(MPIdir)/$(MPIdev)/lib <- MPICH 라이브러리 위치
MPIINCdir = $(MPIdir)/$(MPIdev)/include <- MPICH 헤더 파일 위치
MPILIB = $(MPILIBdir)/libmpich.a <- MPICH 라이브러리 파일

BTLIBS = $(BLACSFINIT) $(BLACSLIB) $(BLACSFINIT) $(MPILIB)
```

```
<- 테스팅에 필요한 라이브러리 들.
INSTdir = $(BTOPdir)/INSTALL/EXE
TESTdir = $(BTOPdir)/TESTING/EXE
FTESTexe = $(TESTdir)/xFbtest_$(COMMLIB)-$(PLAT)-$(BLACSDBGLVL)
CTESTexe = $(TESTdir)/xCbtest_$(COMMLIB)-$(PLAT)-$(BLACSDBGLVL)
#================= End SECTION 1================================
#============== SECTION 2: BLACS INTERNALS ========================
SYSINC = -I$(MPIINCdir)
INTFACE = -Df77IsF2C <- Fortran77 에서 C 로 인터페이스 방
법 Add_, NoChange, UpCase, 또는 f77IsF2C 확실하지 않을 경
우 INSTALL/EXE/xintface 프로그램을 수행해 볼것
SENDIS =            <- -DSndIsLocBlk 로 정의하면 MPI_Send 가
                    locally-blocking 루틴으로 수행된어 더 효율적이다.
   비워둘 경우 globally-blocking으로 가정한다.
BUF =
TRANSCOMM = -DuseMpich <- 여기에 정의 된 파라미터는 플랫폼 마다 틀리다. 주석에서
는 BLACS/INSTALL/xtc_CsameF77 과 BLACS/INSTALL/xtc_UseMpich 를 실행하도록 지시
하고 있다. xtc_CsameF77 과 xtc_UseMpich 를 생성하는 방법은 다음과 같다.
$ BLACS/INSTALL/make xtc_CsameF77
$ BLACS/INSTALL/make xtc_UseMpich
이 파일을 실행시키면 설정할 값이 출력된다.
$ BLACS/INSTALL/EXE/mpirun -np 2 xtc_CsameF77
.............
Set TRANSCOMM = -DUseMpich
$ BLACS/INSTALL/EXE/xtc_UseMpich
Set TRANSCOMM = -DuseMpich
WHATMPI =
SYSERRORS =
DEBUGLVL = -DBlacsDebugLvl=$(BLACSDBGLVL)
DEFS1 = -
DSYSINC $(SYSINC) $(INTFACE) $(DEFBSTOP) $(DEFCOMBTOP) $(DEBUGLVL)
BLACSDEFS = $(DEFS1) $(SENDIS) $(BUFF) $(TRANSCOMM) $(WHATMPI) $(SYSERRORS)
#================= End SECTION 2================================
#================= SECTION 3: COMPILERS ============================
F77            = f77            <- fortran 컴파일러
#F77NO_OPTFLAGS = -Nx400
F77FLAGS       = $(F77NO_OPTFLAGS) -O
F77LOADER      = $(F77)
F77LOADFLAGS   =
CC             = gcc            <- C 컴파일러
CCFLAGS        = -O4
CCLOADER       = $(CC)
CCLOADFLAGS    =

ARCH      = ar
ARCHFLAGS = r
RANLIB    = ranlib
#================= End SECTION 3 ================================
```

컴파일 한다

```
[micro@master BLACS]$ make mpi
```

LIB/blacs_MPI-LINUX-1.a 화일이 생성되어야 한다. SRC/ 디렉토리에는 사용자가 호출할 수 있는 루틴들이 들어 있고, 모두 C와 Fortran77 인터페이스를 가지고 있다. 모든 non-communication 루틴들은 blacs_ 라는 접두어로 시작된다. BLACS 내부 루틴과 전역 변수(global variables)들은 모두 BI_ 라는 접두어를 가지고 있다.

ATLAS (Automatically Tuned Linear Algebra Software) 설치. ATLAS 홈(http://math-atlas.sourceforge.net) 에서 관련 파일을 다운 받아 설치를 한다. 이글을 쓰는 시점에서의 ATLAS 최신버전은 3.5.2 이다.

```
[micro@master share]$ tar xzf atlas3.5.2.tar.gz
[micro@master share]$ cd ATLAS
[micro@master ATLAS]$ make config CC=gcc  #CC 를 지정하지 않으면 gcc 가 사용된
다.
[micro@master ATLAS]$ make config
gcc -o xconfig config.c
./xconfig
ATLAS configure started.
160
159
……
001
Enter number at top left of screen [0]: 160 # 화면에 보이는 가장 큰 수를 적는
다.
====================================================================
IMPORTANT
====================================================================
Before going any further, check
http://math-atlas.sourceforge.net/errata.html.
This is the ATLAS errata file, which keeps a running count of all known
ATLAS bugs and system problems, with associated workarounds or fixes.
IF YOU DO NOT CHECK THIS FILE, YOU MAY BE COMPILING A LI-
BRARY WITH KNOWN BUGS.

Have you scoped the errata file? [y]: y  # errata 문서를 읽어보도록 한다.
Configure will ask a series of ques-
tions, in one of two forms. The first form of ques-
tion is a menu of choices. One option in almost all menus is
'Other/UNKNOWN'. If you are unsure of the answer, al-
ways choose this option.
…………..(생략)……….
Are you ready to continue? [y]: y
I need to know if you are using a cross-compiler (i.e., you are compil-
ing on a different architecture than you want the library built for).
```

```
Are you using a cross-compiler? [n]: n
Probing to make operating system determination:
Operating system configured as Linux # 맞는지 확인

Probing for architecture:
Architecture is set to ATHLON # 맞는지 확인

Probing for supported ISA extensions:
make[2]: *** [atlas_run] Error 132
make[1]: *** [IRun_SSE1] Error 2
SSE2: NO.
SSE1: DETECTED!
Number of CPUs: 1
Required cache flush detected as : 524288 bytes
Looking for compilers:

F77 = /usr/bin/g77 -funroll-all-loops -O3
CC = /usr/bin/gcc -fomit-frame-pointer -O3 -funroll-all-loops
MCC = /usr/bin/gcc -fomit-frame-pointer -O

Looking for BLAS (this may take a while):
Unable to find usable BLAS, BLASlib left blank.
FINDING tar, gzip, AND gunzip
tar : /bin/tar
gzip : /bin/gzip
gunzip : /bin/gunzip


ATLAS has default parameters for OS='Linux' and system='ATHLON'.
If you want to just trust these default values, you can use express setup,
drastically reducing the amount of questions you are required to answer

use express setup? [y]: y
……………
Enter Architecture name (ARCH) [Linux_ATHLONSSE1]: Enter

[micro@master ATLAS]$ make install arch=< arch>
```

arch 는 아키텍쳐 이름이다. config 과정에서 마지막에 출력된다. 위의 config 과정에서 출력된 대로 make install arch=Linux_ATHLONSSE1 를 입력한다.

```
[micro@master ATLAS]$ make install arch=Linux_ATHLONSSE1
.........
(생략 1시간 이상 소요)
ATLAS install complete. Examine
ATLAS/bin//INSTALL_LOG/SUMMARY.LOG for details.
```

다음은 SCALAPACK 을 설치한다. MPICH, BLAS, BLACS 가 설치되어 있어야 한다. http://www.netlib.org/scalapack/ 에서 최신버전의 scalapack 을 다운받은후 압축을 풀면 SCALAPACK 디렉토리가 생긴다.

```
[micro@master share]$ tar xzf scalapack.tgz
```

Slmake.inc 파일을 편집한다. 이 파일은 모든 Makefile 에서 include 되어 사용 되며 설치에 필요한 매크로를 정의한다. INSTALL 디렉토리를 참조 하여 자신에게 맞는 Slmake.inc 파일을 복사하여 편집한다.

```
[micro@master share]$ cd SCALAPACK
[micro@master SCALAPACK]$ cp INSTALL/SLmake.LINUX ./SLmake.inc
[micro@master SCALAPACK]$ vi SLmake.inc
```

대부분의 값들은 기본값을 그대로 이용하고, 앞서 ATLAS 를 이용 플랫폼 에 최적화된 BLAS 를 생성했으니 관련설정을 맞추어 주도록 한다.

```
############################################################################
#
#  Program:         ScaLAPACK
#
#  Module:          SLmake.inc
#
#  Purpose:         Top-level Definitions
#
#  Creation date:   February 15, 2000
#
#  Modified:
#
#  Send bug reports, comments or suggestions to scalapack@cs.utk.edu
#
############################################################################
#
SHELL         = /bin/sh
#
#  The complete path to the top level of ScaLAPACK directory, usually
#  $(HOME)/SCALAPACK
#
home          = $(HOME)/SCALAPACK
#
#  The platform identifier to suffix to the end of library names
#
PLAT          = LINUX
#
#  BLACS setup.  All version need the debug level (0 or 1),
#  and the directory where the BLACS libraries are
#
```

```
BLACSDBGLVL   = 1
BLACSdir      = $(HOME)/BLACS/LIB
#
#  MPI setup; tailor to your system if using MPIBLACS
#  Will need to comment out these 6 lines if using PVM
#
USEMPI        = -DUsingMpiBlacs
#SMPLIB        = /usr/lib/mpi/build/LINUX/ch_p4/lib/libmpich.a
SMPLIB        = /usr/local/mpich/lib/libmpich.a
BLACSFINIT    = $(BLACSdir)/blacsF77init_MPI-$(PLAT)-$(BLACSDBGLVL).a
BLACSCINIT    = $(BLACSdir)/blacsCinit_MPI-$(PLAT)-$(BLACSDBGLVL).a
BLACSLIB      = $(BLACSdir)/blacs_MPI-$(PLAT)-$(BLACSDBGLVL).a
TESTINGdir    = $(home)/TESTING

#
#  PVMBLACS setup, uncomment next 6 lines if using PVM
#
#USEMPI        =
#SMPLIB        = $(PVM_ROOT)/lib/$(PLAT)/libpvm3.a
#BLACSFINIT    =
#BLACSCINIT    =
#BLACSLIB      = $(BLACSdir)/blacs_PVM-$(PLAT)-$(BLACSDBGLVL).a
#TESTINGdir    = $(HOME)/pvm3/bin/$(PLAT)

CBLACSLIB     = $(BLACSCINIT) $(BLACSLIB) $(BLACSCINIT)
FBLACSLIB     = $(BLACSFINIT) $(BLACSLIB) $(BLACSFINIT)

#
#  The directories to find the various pieces of ScaLapack
#
PBLASdir      = $(home)/PBLAS
SRCdir        = $(home)/SRC
TESTdir       = $(home)/TESTING
PBLASTSTdir   = $(TESTINGdir)
TOOLSdir      = $(home)/TOOLS
REDISTdir     = $(home)/REDIST
REDISTTSTdir  = $(TESTINGdir)
#
#  The fortran and C compilers, loaders, and their flags
#
F77           = /usr/local/mpich/bin/mpif77
CC            = /usr/local/mpich/bin/mpicc
NOOPT        =
F77FLAGS     =  -funroll-all-loops -O3 $(NOOPT)
DRVOPTS      = $(F77FLAGS)
CCFLAGS      = -O4
SRCFLAG       =
#F77LOADER     = $(F77)
F77LOADER     = $(F77)
```

```
CCLOADER      = $(CC)
F77LOADFLAGS  =
CCLOADFLAGS   =
#
#  C preprocessor defs for compilation
#  (-DNoChange, -DAdd_, -DUpCase, or -Df77IsF2C)
#
CDEFS         = -Df77IsF2C -DNO_IEEE $(USEMPI)
#
#  The archiver and the flag(s) to use when building archive (library)
#  Also the ranlib routine.  If your system has no ranlib, set RAN-
LIB = echo
#
ARCH          = ar
ARCHFLAGS     = cr
RANLIB        = ranlib
#
#  The name of the libraries to be created/linked to
#
SCALAPACKLIB  = $(home)/libscalapack.a
#BLASLIB       = $(HOME)/BLAS/blas_LINUX.a
# ATLAS 의 BLAS 라이브러리를 지정해 준다.
BLASLIB       = -L$(HOME)/ATLAS/lib/Linux_ATHLONSSE1 -lf77blas -latlas
#
PBLIBS        = $(SCALAPACKLIB) $(FBLACSLIB) $(BLASLIB) $(SMPLIB)
PRLIBS        = $(SCALAPACKLIB) $(CBLACSLIB) $(SMPLIB)
RLIBS         = $(SCALAPACKLIB) $(FBLACSLIB) $(CBLAC-
SLIB) $(BLASLIB) $(SMPLIB)
LIBS          = $(PBLIBS)
############################################################################
```

컴파일 한다. 컴파일 과정에서 에러가 생기면 SLmake.inc 파일을 수정하 고 다시 컴파일 한다.

```
[micro@master SCALAPACK]$ make lib
```

SCALAPACK 디렉토리 아래에 libscalapack.a 파일이 만들어 진다. 여기 까지 이상이 없다면 간단한 테스트 프로그램을 돌려보자.

```
[micro@master SCALAPACK]$ cd TESTING
[micro@master TESTING]$ cd LIN
[micro@master LIN]$ make double
[micro@master LIN]$ cd ..
[micro@master TESTING]$ /usr/local/mpich/bin/mpirun -np [프로세서 개
수] ./xdlu
ScaLAPACK Ax=b by LU factorization.
'MPI Machine'
```

```
Tests of the parallel real double precision LU factorization and solve.
The following scaled residual checks will be computed:
 Solve residual         = ||Ax - b|| / (||x|| * ||A|| * eps * N)
 Factorization residual = ||A - LU|| / (||A|| * eps * N)
The matrix A is randomly generated for each test.

An explanation of the input/output parameters follows:
TIME    : Indicates whether WALL or CPU time was used.
M       : The number of rows in the matrix A.
N       : The number of columns in the matrix A.
NB      : The size of the square blocks the matrix A is split into.
NRHS    : The total number of RHS to solve for.
NBRHS   : The number of RHS to be put on a column of processes be-
fore going
          on to the next column of processes.
P       : The number of process rows.
Q       : The number of process columns.
THRESH  : If a resid-
ual value is less than THRESH, CHECK is flagged as PASSED
LU time : Time in seconds to factor the matrix
Sol Time: Time in seconds to solve the system.
MFLOPS  : Rate of execution for factor and solve.

The following parameter values will be used:
  M       :         10000
  N       :         10000
  NB      :            36
  NRHS    :             3
  NBRHS   :             3
  P       :             1
  Q       :             7

Relative machine precision (eps) is taken to be       0.111022E-15
Routines pass computational tests if scaled residual is less than   1.0000

TIME     M     N  NB NRHS NBRHS    P    Q  LU Time Sol Time  MFLOPS  CHECK
---- ----- ----- --- ---- ----- ---- ---- -------- -------- -------- -----
-
WALL 10000 10000  36    3    3    1    7   100.52     0.46  6607.53 PASSED
Finished      1 tests, with the following results:
    1 tests completed and passed residual checks.
    0 tests completed and failed residual checks.
    0 tests skipped because of illegal input values.

END OF TESTS.
```

위와 비슷한 결과가 나와야 한다. 다음은 SCALAPACK이 설치되었다고 가정하고, 단일 노드에서 LINPACK Benchmark를 수행하는 방법을 설명한다. ATLAS 최적화 루틴을 이용하려면 SLmake.inc에서 ATLAS에서 제공하는 BLAS 루틴을 이용하도록 설정한다.

```
[micro@master SCALAPACK]$ vi SLmake.inc
......(생략)
BLASLIB       = -L$(HOME)/ATLAS/lib/Linux_ATHLONSSE1 -lf77blas -latlas
......(생략)
```

SCALAPACK/TESTING 디렉토리에는 기본적으로 13개의 .dat 화일과 LIN, EIG 디렉토리가 설치된다. LIN, EIG 디렉토리에는 벤치마크를 수행하는 Fortran으로 작성된 소스코드와 Makefile이 들어 있다. LIN 디렉토리는 Linear Equations Testing 을 위한 프로그램들이 들어 있다. EIG 디렉토리에는 Eigenroutine Testing을 위한 프로그램이 들어 있다. TESTING 디렉토리에 포함된 .dat 화일과 용도는 다음과 같다.

```
BLLT.dat 'ScaLAPACK, Version 1.2, banded linear systems input file'
BLU.dat  'ScaLAPACK, Version 1.2, banded linear systems input file'
BRD.dat  'ScaLAPACK BRD input file'
HRD.dat  'ScaLAPACK HRD input file'
INV.dat  'ScaLAPACK, Version 1.0, Matrix Inversion Testing input file'
LLT.dat  'ScaLAPACK, LLt factorization input file'
LS.dat   'ScaLAPACK LS solve input file'
LU.dat   'SCALAPACK, LU factorization input file'
NEP.dat  'SCALAPACK NEP (Nonsymmetric Eigenvalue Problem) input file'
QR.dat   'ScaLAPACK, Orthogonal factorizations input file'
SEP.dat  'ScaLAPACK Symmetric Eigensolver Test File'
SVD.dat  'ScaLAPACK Singular Value Decomposition  input file'
TRD.dat  'ScaLAPACK TRD computation input file'
```

.dat 파일에는 각각의 벤치마크에 필요한 변수들이 저장된다. 테스트 에 필요한 LU.dat 파일을 살펴보자.

```
[micro@master TESTING]$ vi LU.dat
-- LU.dat --
'SCALAPACK, LU factorization input file'
'MPI Machine'
'LU.out'                output file name (if any)
6                       device out
4                       number of problems sizes
4 10 17 13 23 31 57     values of M
4 12 13 13 23 31 50     values of N
3                       number of NB's
2 3 4 5                 values of NB
3                       number of NRHS's
1 3 9 28                values of NRHS
3                       Number of NBRHS's
1 3 5 7                 values of NBRHS
```

```
4                       number of process grids (ordered pairs of P & Q)
1 2 1 4 2 3 8           values of P
1 2 4 1 3 2 1           values of Q
1.0                     threshold
T                       (T or F) Test Cond. Est. and Iter. Ref. Routines
-- LU.dat --
```

대략적인 형식은 각 벤치마크에 필요한 값들과 해당 값들의 숫자로 정의 되어 있다. 예를 들어 number of problems sizes 가 4 이므로 M 과 N 의 벤치마크 테스팅 에 사용될 값은 M = 4,10,17,13 이고 N = 4,12,13,23 이다. 그리고 values of P,Q 는 프로세서의 Grid 를 뜻한다. P 는 프로세서의 row 를 뜻하며 Q 는 column 을 뜻한다. 위의 경우 P = 1 , Q = 1 이면 1 X 1 = 1 이니 1 개의 프로세서 에서 실행된다는 것을 뜻한다. 두번째 의 경우인 P = 2 , Q = 2 의 경우 2 X 2 = 4 이니 4 개의 프로세서(Node) 에서 실행된다는 것을 뜻한다. 해당 벤치마킹 프로그램을 생성하려면 LIN 이나 EIG 디렉토리 에서 make [type] 을 실행하면 된다. 해당 type 은 single, double, complex, complex16 4 가지 가 있다. 예를 들어 make single 은 single precision floating point 를 위한 벤치마킹 프로그램이 생성된다. make all 은 4가지 형에 대한 모든 벤치마킹 프로그램을 한꺼번에 컴파일 한다. 해당 type 에 대한 파일명과 개수는 다음과 같다.

```
[micro@master LIN]$ make single
[micro@master LIN]$ ls ../xs*
xsdblu*  xsdtlu*  xsgblu*  xsinv*  xsllt*  xsls*  xslu*  xsp-
bllt*  xsptllt*  xsqr*
[micro@master LIN]$ make double
[micro@master LIN]$ ls ../xd*
xddblu*  xddtlu*  xdgblu*  xdinv*  xdllt*  xdls*  xdlu*  xdp-
bllt*  xdptllt*  xdqr*
[micro@master LIN]$ make complex
[micro@master LIN]$ ls ../xc*
xcdblu*  xcdtlu*  xcgblu*  xcinv*  xcllt*  xcls*  xclu*  xcp-
bllt*  xcptllt*  xcqr*
[micro@master LIN]$ make complex16
[micro@master LIN]$ ls ../xz*
xzdblu*  xzdtlu*  xzgblu*  xzinv*  xzllt*  xzls*  xzlu*  xzp-
bllt*  xzptllt*  xzqr*
```

이와 같이 총 40개의 실행파일 들이 생성되는데 마찬가지로 EIG 디렉토리 에서도 같은 형식으로 make 를 하면 프로그램이 생성된다. 각각의 프로그 램을 실행하여 보자. MPI 를 이용하여 실행하려면 mpirun -np N program 을 실행하면 된다. LINPACK 의 벤치마크 에서는 LU.dat 파일의 파라메터 설정을 이용하여 벤치마크 할수 있다. 기본적으로 ScaLAPACK은 블럭 단위로 연산을 수행하며, 클러스터와 같은 병렬 컴퓨터에서 최대 성능을 얻기 위해서는 주어진 컴퓨터에 적절한 블록의 크기를 구하여야 한다. 이는 계산으로 대략적인 값을 구한 후에, 많은 실행을 거쳐 경험적으로 얻을 수 있다. 그리고 컴퓨터가 수행할 수 있는 최대 문제 크기(Nmax)를 얻기 위해서는, 하나의 프로세서에서 문제 크기를 점차로 늘리면서 주어진 메모리에 노드가 포용할 수 있는 최대 크기를 알아내야 한다. 마지막으로 이를 바탕으로 많은 노

드를 가진 병렬 컴퓨터에서 수행할 수 있는 최대의 크기로 LU 인수분해 루팅을 수행시켜서 최대의 성능(Rmax)을 얻을 수 있다.

Fortran 으로 짜여진 LIN/pdludriver.f 파일에서 TOTMEM 의 값을 변화시키면서 Segmentation fault 가 발생하는 범위를 조사해 보자. 이것은 주 메모리 뿐만 아니라 스왑영역의 크기에 영향을 받는다. 스왑 영역의 크기보다 TOTMEM 값을 크게 하면 Segmentation fault 를 일으킬 것이다. 물론 소스를 수정하고 나면 컴파일을 다시 해야 한다. 2GB 메모리, 500MB 스왑영역의 주어진 조건에서 TOTMEM 을 500000000 로 정하였다. TESTING 디렉토리의 LU.dat 을 다음과 같이 수정하고 xdlu 를 실행시켜 보자. 계산에 필요한 메모리양이 메인 메모리의 크기보다 크면 스왑영역의 억세스를 위해서 하드디스크가 동작하는 것을 볼수 있을것이다.

```
-- LU.dat --
'SCALAPACK, LU factorization input file'
'MPI Machine'
'LU.out'                 output file name (if any)
6                        device out
6                        number of problems sizes
1000 1200 1400 1600 1800 2000 values of M
1000 1200 1400 1600 1800 2000 values of N
1                        number of NB's
60          values of NB
1                        number of NRHS's
1                  values of NRHS
1                        Number of NBRHS's
1                   values of NBRHS
1                        number of process grids (ordered pairs of P & Q)
1        values of P
1        values of Q
1.0                      threshold
T                        (T or F) Test Cond. Est. and Iter. Ref. Routines
-- LU.dat --
```

이것을 2번 실행한 결과는 다음과 같다. 첫번째 실행한 결과

```
TIME     M     N  NB NRHS NBRHS    P    Q  LU Time Sol Time   MFLOPS  CHECK
---- ----- ----- --- ---- ----- ---- ---- -------- -------- -------- -----
-
WALL  1000  1000  60     1     1    1    1     0.64     0.01  1026.42 PASSED
WALL  1200  1200  60     1     1    1    1     1.05     0.02  1078.62 PASSED
WALL  1400  1400  60     1     1    1    1     1.67     0.02  1083.34 PASSED
WALL  1600  1600  60     1     1    1    1     2.29     0.03  1177.45 PASSED
WALL  1800  1800  60     1     1    1    1     3.13     0.04  1227.84 PASSED
WALL  2000  2000  60     1     1    1    1     4.37     0.05  1207.76 PASSED
```

두번째 실행한 결과

```
TIME     M     N  NB NRHS NBRHS    P    Q  LU Time Sol Time   MFLOPS  CHECK
```

```
---- ----- ----- --- ---- ----- ---- ---- -------- -------- -------- -----
-

WALL  1000  1000  60     1     1     1     1     0.63      0.01  1032.06 PASSED
WALL  1200  1200  60     1     1     1     1     1.05      0.02  1079.69 PASSED
WALL  1400  1400  60     1     1     1     1     1.59      0.02  1134.49 PASSED
WALL  1600  1600  60     1     1     1     1     2.28      0.03  1184.27 PASSED
WALL  1800  1800  60     1     1     1     1     3.12      0.04  1231.93 PASSED
WALL  2000  2000  60     1     1     1     1     4.37      0.05  1207.30 PASSED
```

문제의 크기가 커질수록 MFLOPS 가 증가하다가 SWAP 을 사용할 정도가 되면 성능이 떨어지는 것을 알수 있다. 다음은 NB를 바꾸면서 수행을 해보자. M 과 N 을 각자의 시스템에 맞게 수정하면서 NB 를 28 에서 60 까지 변화 시킨다.

```
TIME      M      N  NB NRHS NBRHS     P     Q   LU Time Sol Time   MFLOPS  CHECK
---- ----- ----- --- ---- ----- ---- ---- -------- -------- -------- -----
-

WALL  5000  5000  28     1     1     1     7    19.85      0.14  4170.73 PASSED
WALL  5000  5000  30     1     1     1     7    14.85      0.14  5562.10 PASSED
WALL  5000  5000  32     1     1     1     7    15.40      0.13  5367.77 PASSED
WALL  5000  5000  34     1     1     1     7    15.89      0.15  5198.10 PASSED
WALL  7000  7000  28     1     1     1     7    49.39      0.24  4608.81 PASSED
WALL  7000  7000  30     1     1     1     7    37.77      0.27  6013.41 PASSED
WALL  7000  7000  32     1     1     1     7    38.96      0.25  5833.21 PASSED
WALL  7000  7000  34     1     1     1     7    39.07      0.26  5816.04 PASSED
WALL 10000 10000  28     1     1     1     7   133.66      0.41  4973.60 PASSED
WALL 10000 10000  30     1     1     1     7    99.69      0.45  6659.18 PASSED
WALL 10000 10000  32     1     1     1     7   102.15      0.43  6500.25 PASSED
WALL 10000 10000  34     1     1     1     7   101.73      0.40  6529.03 PASSED
```

위의 실험에서 측정된 최고 성능은 M=N=10000 NB=30 일때 6659.18 MFLOPS 이다. M=N 값을 통일하고 NB 의 최적화 값을 찾아보도록 하자.

```
TIME      M      N  NB NRHS NBRHS     P     Q   LU Time Sol Time   MFLOPS  CHECK
---- ----- ----- --- ---- ----- ---- ---- -------- -------- -------- -----
-

WALL  1000  1000  28     1     1     1     7     1.09      0.03   595.45 PASSED
WALL  1000  1000  30     1     1     1     7     0.33      0.02  1939.09 PASSED
WALL  1000  1000  32     1     1     1     7     0.33      0.02  1911.82 PASSED
WALL  1000  1000  34     1     1     1     7     0.37      0.02  1699.22 PASSED
WALL  1000  1000  36     1     1     1     7     0.38      0.02  1695.32 PASSED
WALL  1000  1000  38     1     1     1     7     0.40      0.02  1606.20 PASSED
```

M=N=1000 으로 통일시키고 측정했을경우 NB 가 30 일경우 가장 좋은 성능을 보임.

4.5.3.2. HPL (High-Performance Linpack Benchmark)

다음은 대용량 메모리 시스템 을 벤치마크 하는데 쓰이는 (전세계 슈퍼컴퓨팅 순위를 매기는 TOP 500 Site 에서 사용하는 프로그램) HPL 을 이용하여 벤치마킹 을 해보자. HPL 을 설치하기 전에 BLAS , MPICH CBLAS 등이 설치되어 있어야 한다. 여기서는 ATLAS 의 BLAS 루틴을 이용할 것 이기 때문에 ATLAS 도 설치되어 있어야 한다. CBLAS 설치는 한국 클러스터 기술센터 http://www.cluster.or.kr/board/read.php?table=benchmark=3 를 참고하거나 여기를 참고하도록 한다. http://www.netlib.org/blas/ Hpl 을 다운받아서 압축을 푼다. http://www.netlib.org/benchmark/hpl/

```
[micro@master share]$ tar xzf hpl.tgz
```

hpl 디렉토리 안의 setup 디렉토리 에서 해당플랫폼에 맞는 make 파일을 hpl Top 디렉토리 안으로 복사한다. 여기서는 Linux 의 Athlon 칩, 그리 고 BLAS 의 C 인터페이스인 CBLAS 를 사용할 것 이므로 파일명은 다음 과 같다.

```
[micro@master share]$ cd hpl
[micro@master hpl]$ cp setup/Make.Linux_ATHLON_CBLAS .
```

해당파일을 수정하도록 한다.

```
[micro@master hpl]$ vi Make.Linux_ATHLON_CBLAS
------ Make.Linux_ATHLON_CBLAS -------
SHELL        = /bin/sh
#
CD           = cd
CP           = cp
LN_S         = ln -s
MKDIR        = mkdir
RM           = /bin/rm -f
TOUCH        = touch

ARCH         = Linux_ATHLON_CBLAS

TOPdir       = $(HOME)/hpl
INCdir       = $(TOPdir)/include
BINdir       = $(TOPdir)/bin/$(ARCH)
LIBdir       = $(TOPdir)/lib/$(ARCH)
#
HPLlib       = $(LIBdir)/libhpl.a

#CC           = gcc
CC           = /usr/local/mpich/bin/mpicc  <- MPICH 의 C 컴파일러
NOOPT        =
#CCFLAGS       = -fomit-frame-pointer -O3 -funroll-loops -W -Wall
CCFLAGS      = -fomit-frame-pointer -O3 -funroll-loops
#
```

```
#LINKER        = gcc
LINKER        = /usr/local/mpich/bin/mpicc
LINKFLAGS     = $(CCFLAGS)
#
ARCHIVER      = ar
ARFLAGS       = r
RANLIB        = echo

MPdir         = /usr/local/mpich
MPinc         = -I$(MPdir)/include
MPlib         = $(MPdir)/lib/libmpich.a

F2CDEFS       =
NOOPT         =
F77           = /usr/local/mpich/bin/mpif77
F77LOADER     = /usr/local/mpich/bin/mpif77
F77FLAGS      = -O $(NOOPT)

LAdir         = $(HOME)/ATLAS/lib/Linux_ATHLONSSE1
LAinc         = $(HOME)/ATLAS/include/Linux_ATHLONSSE1
LAlib         = $(LAdir)/libcblas.a $(LAdir)/libatlas.a

#HPL_INCLUDES = -I$(INCdir) -I$(INCdir)/$(ARCH) $(LAinc) $(MPinc)
<- 윗부분을 아래와 같이 수정한다.
HPL_INCLUDES = -I$(INCdir) -I$(INCdir)/$(ARCH) -I$(LAinc) $(MPinc)
HPL_LIBS      = $(HPLlib) $(LAlib) $(MPlib)

HPL_OPTS      = -DHPL_CALL_CBLAS
HPL_DEFS      = $(F2CDEFS) $(HPL_OPTS) $(HPL_INCLUDES)
------ Make.Linux_ATHLON_CBLAS -------
```

다음 Make.top 파일과 Makefile 의 arch 부분을 수정해 준다.

```
#
arch              = Linux_ATHLON_CBLAS
#
```

컴파일 한다. Make arch=[해당시스템] 을 입력하자. 그럼 bin 디렉토리 아 래에 [해당시스템] 디렉토리가 생성됐을것이다.

```
[micro@master hpl]$ make arch=Linux_ATHLON_CBLAS
[micro@master hpl]$ cd bin/Linux_ATHLON_CBLAS
```

bin/Linux_ATHLON_CBLAS 디렉토리에 가보면 HPL.dat 파일과 xhpl 파 일이 보일것이다. HPL.dat 파일은 앞서 LINPACK 벤치마킹 에 환경설정 파일처럼 여러가지 벤치마킹에 필요한

파라미터 들을 설정하는 곳이고, xhpl 실행파일은 실질적으로 벤치마킹에 돌리는 프로그램이다. 그럼 HPL.dat 파일의 포맷을 살펴보자.

```
[micro@master hpl]$ vi HPL.dat
----- HPL.dat -----
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
10000
1           # of NBs
85           NBs
1            # of process grids (P x Q)
1            Ps
7            Qs
16.0         threshold
1            # of panel fact
1            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64          swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
----- HPL.dat -----
```

기본적으로 LINPACK 벤치마크의 LU.dat 과 크게 다르지 않다는 것을 알수가 있다. 몇가지 차이점은 Problem size 가 1차원 으로 바뀐것과 Swapping threshold 를 지정할수 있다는 것 등인데 자세한 내용은 해당 튜닝 페이지 http://www.netlib.org/benchmark/hpl/tuning.html 를 참고하도록 하자. xhpl 을 실행해 보도록 한다.

```
[micro@master Linux_ATHLON_CBLAS]$ mpirun -np 7 xhpl
====================================================================
HPLinpack 1.0  --  High-Performance Linpack benchmark  --
September 27, 2000
Written by A. Petitet and R. Clint Whaley,  Innovative Comput-
ing Labs.,  UTK
```

```
============================================================================
An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N      :   10000
NB     :    85
P      :       1
Q      :       7
PFACT  :   Crout
NBMIN  :       4
NDIV   :       2
RFACT  :   Right
BCAST  :  1ringM
DEPTH  :       1
SWAP   : Mix (threshold = 64)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words
============================================================================
T/V                N    NB     P     Q               Time             Gflops
----------------------------------------------------------------------------
--
W11R2C4        10000    85     1     7              70.49          9.460e+00
----------------------------------------------------------------------------
--
||Ax-
b||_oo / ( eps * ||A||_1  * N        ) =        0.0646673 ...... PASSED
||Ax-
b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0153022 ...... PASSED
||Ax-
b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0034203 ...... PASSED
============================================================================
```

위의 결과는 N = 10000 , NB = 85 일때 9.46Gflops 가 나왔다. LINPACK 과 마찬가지로 여러분의 시스템 환경에 맞게 problem size 와 NB 를 적절히 수정해 가면서 시스템이 수행할수 있는 최고성능을 이끌어 내보자. HPL.dat 파일을 수정한다음 컴파일을 다시 한다.

```
[micro@master Linux_ATHLON_CBLAS]$ rm -f ./xhpl
```

```
[micro@master Linux_ATHLON_CBLAS]$ cd ../../
[micro@master Linux_ATHLON_CBLAS]$ make clean
[micro@master Linux_ATHLON_CBLAS]$ make all
```

5장. Reference sites

- Internet Performance Tuning http://www-iepm.slac.stanford.edu/
- System Tuning for Linux http://people.redhat.com/alikins/system_tuning.html
- Linux Performance tuning http://linuxperf.nl.linux.org/
- MPICH http://www-unix.mcs.anl.gov/mpi/mpich/
- Linux Kernel Guide http://kldp.org/KoreanDoc/html/Kernel-KLDP/
- Linux Kernel Hacker's Guide http://tldp.org/LDP/khg/HyperNews/get/khg.html
- NetPIPE Benchmark http://www.scl.ameslab.gov/
- Rocks Cluster http://www.rocksclusters.org/Rocks/
- Top500 SuperComputer http://www.top500.org
- LAM-MPI http://www.lam-mpi.org
- Beowolf Project http://www.beowulf.org
- POSDATA http://cluster.posdata.co.kr
- Samsung Alpha-11 Cluster http://www.alpha11.com
- Cluster References http://www.cluster.or.kr
- SCALAPACK http://www.netlib.org/scalapack