2020년

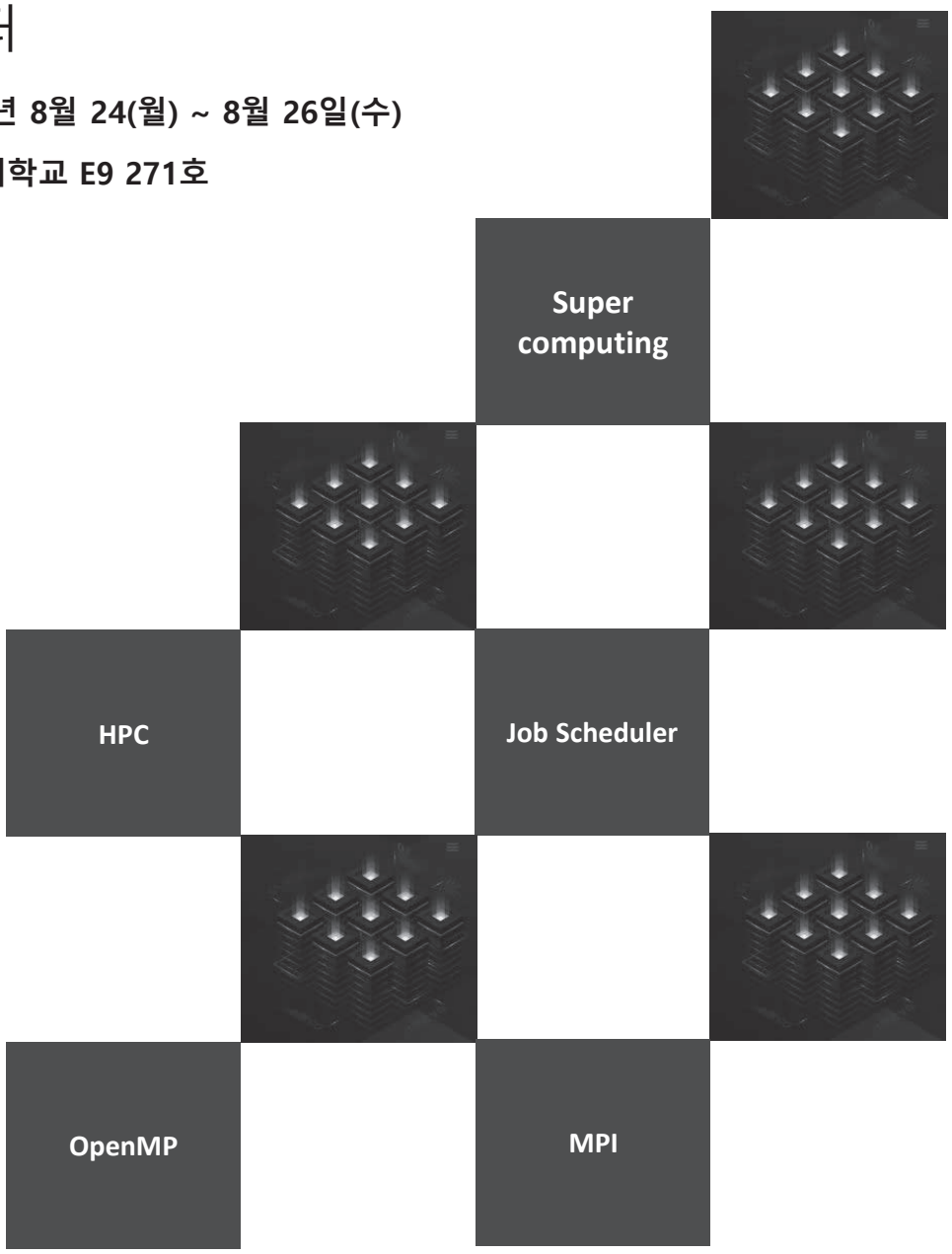
HPC 여름학교

High Performance Computing :

슈퍼컴퓨터

일자 | 2020년 8월 24(월) ~ 8월 26일(수)

장소 | 충북대학교 E9 271호



충북대학교 CHUNGBUK NATIONAL UNIVERSITY

충북대학교 SW중심대학사업단

KISTI 한국과학기술정보연구원 www.kisti.re.kr

KISTI과학데이터스쿨 KISTI SCIENCE DATA SCHOOL

Introduction

Korea Institute of
Science and Technology
Information

KISTI

국가와 국민을 위한
데이터 생태계 중심기관 KISTI

데이터를 공유하고, 가치를 창출하는 역동적인
과학기술 데이터 생태계 중심기관으로 산학연 공동체와 함께
성장하며 국가 혁신 성장의 밑거름이 되겠습니다.

2020년

HPC 여름학교

(High Performance Computing : 슈퍼컴퓨터)

KISTI 과학데이터스쿨과 충북대학교 SW중심대학사업단은 데이터 중심이 되어가는 4차 산업혁명시대에서 더 많은 데이터를, 더 똑똑한 모델로 더 빨리 처리할 수 있는 고성능 컴퓨팅 기술을 이용할 수 있는 HPC여름학교를 개최합니다.

교육 일정

일 자	교육내용
8월 24일(월) Linux & 슈퍼컴퓨터 5호기(누리온)	등록
	인사말 및 소개(사업단, KISTI)
	리눅스 기본 명령어
	리눅스 응용 프로그램 사용법
	점심식사
	슈퍼컴퓨터 5호기(누리온) 소개
	기본 환경 설치 및 사용법
	Job Scheduler 사용법
8월 25일(화) OpenMP	OpenMP 소개 및 병렬 프로그래밍
	점심식사
	스케줄링 작업 및 태스크 관리
8월 26일(수) MPI	MPI 소개 및 기본 API
	점심식사
	MPI를 이용한 P2P 통신방법
	MPI를 이용한 집합 통신방법
	수료식

(※ 사정에 따라 교육내용 및 일정 등은 변동될 수 있음.)

프로그램 안내

- 일시 | 2020년 8월 24일(월) ~ 8월 26일(수)
- 장소 | 충북대학교 E9동 271호
- 공동주관 | 충북대 SW중심대학사업단, KISTI과학데이터스쿨
- 대상 | 충북대 대학(원)생 및 교직원(선착순 30명)

교육 강사진

강사명	소속	이력
안부영	한국과학기술정보연구원	KISTI 과학데이터스쿨 센터장
윤준원	한국과학기술정보연구원	슈퍼컴퓨팅인프라센터 선임연구원
이재국	한국과학기술정보연구원	슈퍼컴퓨팅인프라센터 선임연구원
이승우	㈜모아시스	HPC 분야 전문강사
정진우	㈜모아시스	HPC 분야 전문강사

혜 택

- "한국과학기술정보연구원"장 명의 수료증 발급
- SW중심대학사업단 참여마일리지 부여

접 수

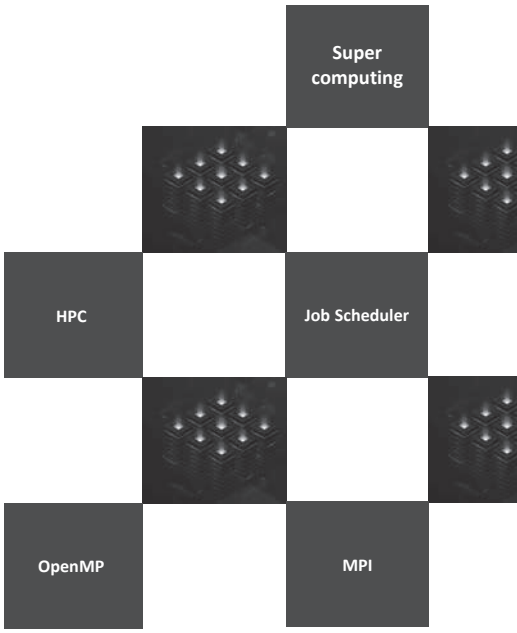
구글 설문지를 통해 신청서 작성
(신청서 링크에 대한 QR 코드 제공)
※온라인 접수: https://forms.gle/Jv8aQtAFmVjUirW47

문 의

충북대학교 E9동 학연산공동기술연구원 274호
(Tel: 043-249-1346 / Email: jby0312@cbnu.ac.kr)

충북대학교 SW중심대학사업단

KISTI 한국과학기술정보연구원 Korea Institute of Science and Technology Information



2020년

HPC 여름학교

High Performance Computing :
슈퍼컴퓨터

리눅스(Linux)

KISTI SUPERCOMPUTING EDUCATION CENTER

리눅스(Linux)

2020

KISTI과학데이터스쿨
KISTI SCIENCE DATA SCHOOL
http://kacademy.kisti.re.kr

KISTI 과학데이터스쿨

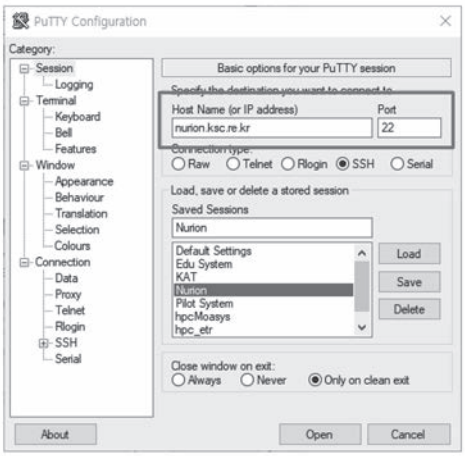
리눅스 개요

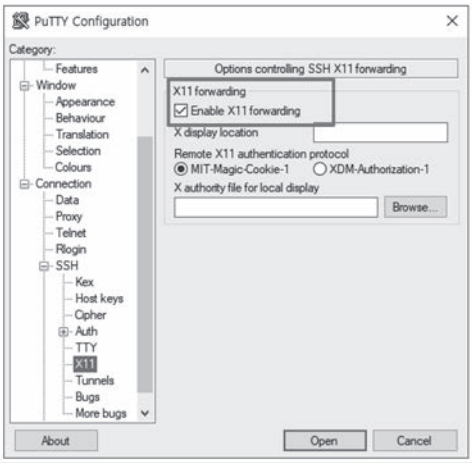
1. 시스템 접속
   - PBS Interactive 작업 제출
2. 유닉스/리눅스 역사
3. 리눅스 구조
4. 명령어 구조
5. **Manual page**

시스템 접속

- Xming
  - X 환경 실행을 위해 필요
- Putty 사용
  - Host Name : nurion.ksc.re.kr(port : 22)

※ Xming 실행 필요





시스템 접속

- 접속 ID & otp
  - sedu##(01~49)
  - Passwd(OTP) : xxxx
  - Passwd :

```
Last login: Mon Jan  7 10:00:35 2019 from xxx.xxx.xxx.xx
================ KISTI 5th NURION System
====================
  * Compute Nodes(node[0001-8305],cpu[0001-0132)
  - KNL(XeonPhi 7250 1.40GHz 68C) / 16GB(MCDRAM),96GB(DDR4)
  - CPU-only(XeonSKL6148 2.40GHz 20C x2) / 192GB(DDR4)

 * Software
  - OS: CentOS 7.4(3.10.0-693.21.1.el7.x86_64)
  - System S/W: BCM v8.1, PBS v14.2, Lustre v2.10

 * Current Configurations
  - All KNL Cluster modes - Quadrant
  - Memory modes
   : Cache-node[0001-7980,8281-8300]/Flat-node[7981-8280]
   : PBS job sharing mode-Exclusive(running 1 job per node)
     (Except just the commercial queue)
   ...
* Policy on User Job
     ....
   (Use the # showq & # pbs_status commands for more queue info.)
```

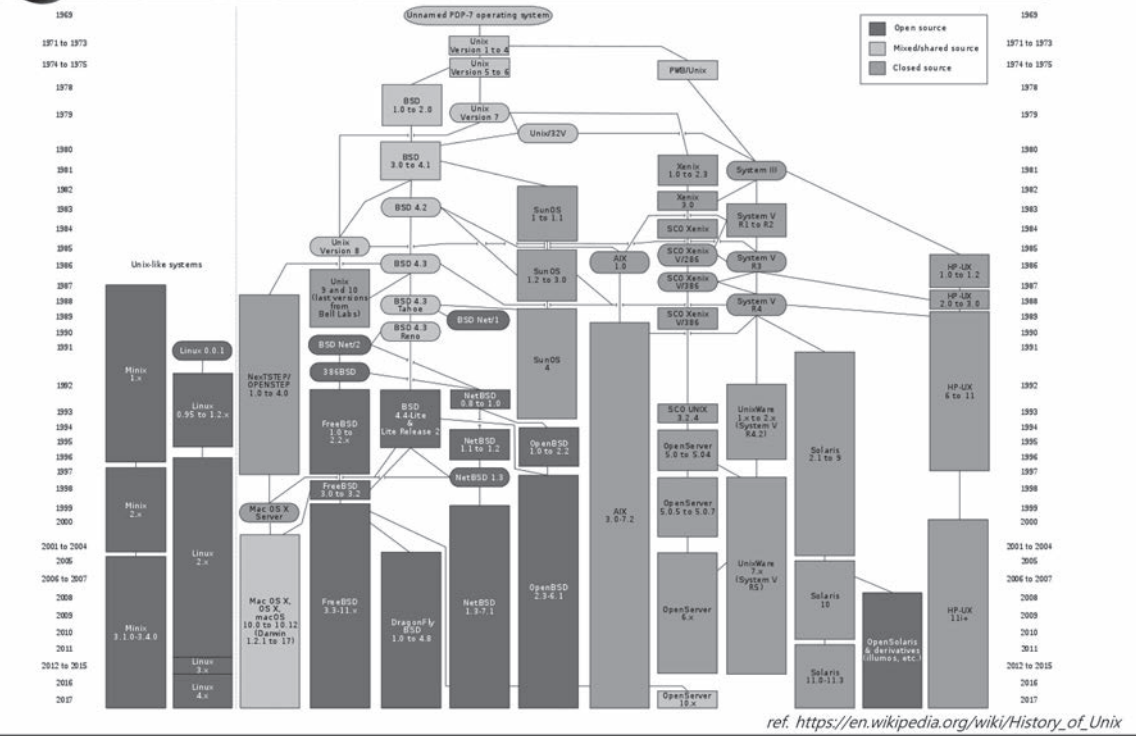
PBS Interactive 작업 제출

- 누리온 시스템은 debug 노드 대신 debug 큐를 제공
- /scratch 디렉터리에서 작업 제출 해야 함
- debug 큐를 이용하여 작업을 제출함으로써 디버깅 수행이 가능
- qsub –I (대문자 i 임)

```
[sedu01@pbcm Pi_Calc]$ qsub -I -V -A etc -l select=1:ncpus=1 -l
walltime=04:00:00 -q debug
qsub: waiting for job 6719.pbcm to start
qsub: job 6719.pbcm ready

Intel(R) Parallel Studio XE 2017 Update 2 for Linux*
Copyright (C) 2009-2017 Intel Corporation. All rights reserved.

[sedu50@node8281 ~]$ cd $PBS_O_WORKDIR
[sedu50@node8281 ~]$ ls
…
[sedu50@node8281 ~]$ exit
```

❖ 보다 자세한 사항은 **슈퍼컴퓨팅센터 홈페이지(https://www.ksc.re.kr) 지침서, 블로그(https://blog.ksc.re.kr) 동영상 지침서** 참조

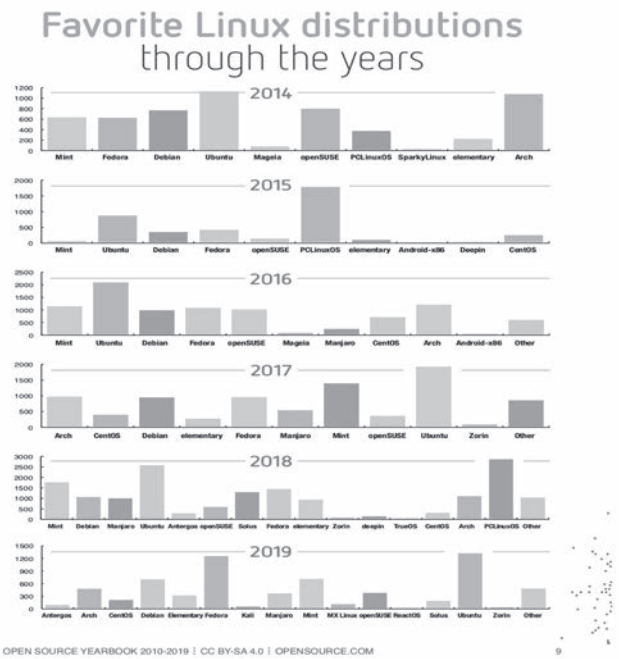
리눅스/유닉스의 역사



ref. https://en.wikipedia.org/wiki/History_of_Unix

리눅스 배포판

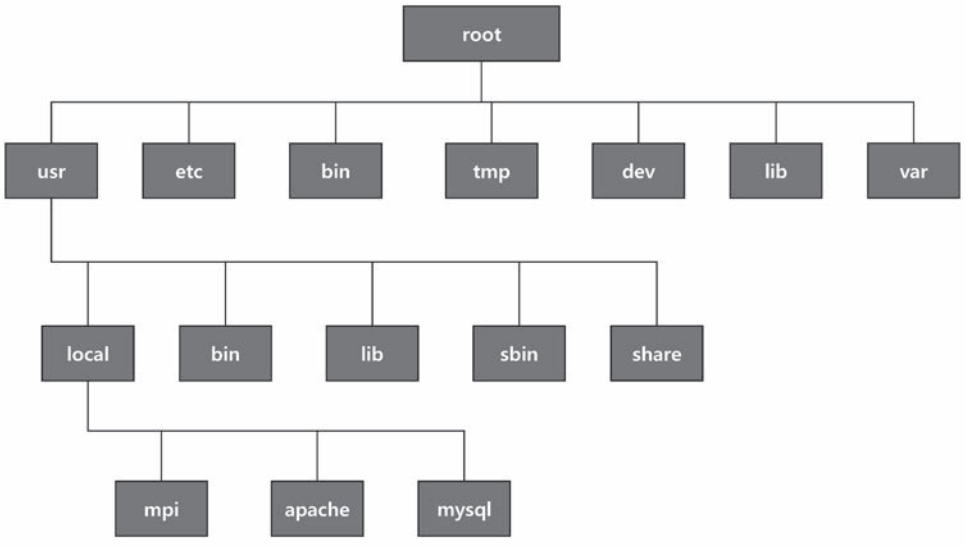
FAVORITE LINUX DISTRIBUTIONS THROUGH THE YEARS



OPEN SOURCE YEARBOOK 2010-2019 ¦ CC BY-SA 4.0 ¦ OPENSOURCE.COM 9

리눅스의 구조

- File Hierarchy



리눅스의 구조

➢ File Hierarchy

디렉터리	기 능
/usr	공유 가능한 파일들을 포함
/etc	시스템에서 사용하는 관리 파일 ▪ init, getty, mknod, motd, passwd, …
/bin	기본적인 실행 가능한 명령 파일을 가지고 있음 ▪ cat, cp, date, echo, mv, pwd, …
/tmp	임시 디렉터리
/dev	장치 파일들이 있는 디렉터리 시스템의 모든 입출력 파일을 가지고 있음
/lib	기본적인 프로그램 모듈이 있는 디렉터리

명령어 – 구조

➢ 명령어 조합

Command + Options + Arguments

➢ Example

```
$ ls
$ ls -a
$ ls -a /home01/sedu50
```

명령어 - 구조

- 명령어 구조
  - 명령어(command)
    - 일반적으로 소문자
    - 무엇을 할 것인가?
  - 옵션(option)
    - 불필요한 경우도 있음
    - 명령어 출력 향상
    - 사용자 필요에 맞추어 출력
    - 하나 이상의 다른 옵션과 함께 조합하여 사용
  - 인자(argument)
    - 사용자의 명령어에 따라 실행
    - 하나 이상의 argument 를 가질 수 있음
    - 불필요한 경우도 있음

```
$ ls
```

```
$ ls -al
```

```
$ ls -alh /home01/sedu50
```


Manual page

- 시스템에서 제공하는 도움말 (man page)
- 기본적으로 command 마다 해당 man page 를 가짐
  - 다음 페이지를 보기 위해서는 space bar 또는 'f' 입력
  - 이전 페이지를 보기 위해서는 'b' 입력
  - 마치려면 'q' 입력

```
$ man  who
WHO(1)                          User Commands                          WHO(1)

NAME
       who - show who is logged on

SYNOPSIS
       who [OPTION]... [ FILE | ARG1 ARG2 ]

DESCRIPTION
       -a, --all
              same as -b -d --login -p -r -t -T -u

       -b, --boot
              time of last system boot

       -d, --dead
              print dead processes
```

기본 명령어

1. 명령어 모음
2. 디렉터리/파일 관련 명령어
3. 절대 경로/상대 경로
4. 파일 속성
5. 리다이렉팅
6. 파일 검색
7. 시스템 모니터링

명령어 모음

- passwd : 패스워드 변경
- du(disk usage) : 디스크 사용량 체크
  - 디스크 파일 사용량을 재귀적으로 보여줌
  - 특별히 디렉터리를 지정하지 않으면 현재 디렉터리에 대하여 동작

  ex) du –h, du –s drectory_name
- df(disk free) : 파일 시스템 사용량을 탭이 들어간 형태로 보여줌

  Ex) df, df -h

명령어 모음

- date
  - 현재 날짜와 시간을 나타냄
  - 사용방법 : date [옵션] [+포맷]

```
$ date                                  $ date -d "1 month 10 days"
Thu Jan 24 10:13:11 KST 2019            Wed Mar  6 10:15:51 KST 2019
$ date -d yesterday                     $ date -d tomorrow
Wed Jan 23 10:13:38 KST 2019            Fri Jan 25 10:16:28 KST 2019
$ date -d "10 days ago"                 $ date +%Y%m%d
Mon Jan 14 10:14:42 KST 2019            20190124
$ date --date="-10 days"                $ date +%Y%m%d%H
Mon Jan 14 10:15:01 KST 2019            2019012410
$ date -d "+10 days"                    $ date +%Y%m%d%H%M
Sun Feb  3 10:15:21 KST 2019            201901241017
$ date --date="+10 days"                $ date +%Y:%m:%d-%H:%M
Sun Feb  3 10:15:35 KST 2019            2019:01:24-10:17
```


명령어 모음

- cal
  - 현재 날짜 및 특정 년, 월의 날짜를 구한다

```
[sedu50@login03 ~]$ cal
    January 2019
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

```
[sedu50@login03 ~]$ cal 4 2019
     April 2019
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
```

- bc [-l]
  - 간단한 계산기, 종료하려면 'quit'
  - 'l' 옵션을 같이 사용하면 함수들 사용 가능$(s(x),\ c(x), l(x), e(x), \ldots)$

```
$ bc -l
e(0)          // exponential
1.00000000000000000000
l(10)         // ln(10)
```

명령어 모음

- who, who am i
  - 시스템에 Login 한 사용자 및 자신이 누구인지 보여줌
  - $ who [am i]

```
[sedu50@login03 EDU]$ who
x1657a05 pts/0        2019-01-22 21:39 (114.70.4.22)
x1685a05 pts/2        2019-01-16 00:26 (114.70.15.204)
systemo  pts/4        2019-01-18 16:05 (150.183.152.106)
systemo  pts/5        2019-01-02 14:42 (150.183.152.106)
systemo  pts/7        2019-01-02 14:39 (150.183.152.103)
x1692a02 pts/8        2019-01-24 03:56 (165.132.36.135)
…
[sedu50@login03 EDU]$ who am i
sedu50   pts/84       2019-01-24 11:22 (210.110.236.29)
```

명령어 모음

- which
  - 명령어의 경로를 보여줌

```
[sedu50@login03 EDU]$ which ls
alias ls='ls --color=auto'
        /usr/bin/ls
[sedu50@login03 EDU]$ which cal
/usr/bin/cal
```

- whereis
  - which와 비슷한 명령어로, 실행파일, 소스, man page의 경로를 보여줌

```
[sedu50@login03 EDU]$ whereis ls
ls: /usr/bin/ls /usr/share/man/man1/ls.1.gz /usr/share/man/man1p/ls.1p.gz
[sedu50@login03 EDU]$ whereis cal
cal: /usr/bin/cal /usr/share/man/man1/cal.1.gz
/usr/share/man/man1p/cal.1p.gz
```

- whatis : 명령어가 무슨 일을 하는지 설명 해줌

```
[sedu50@login03 EDU]$ whatis ls
ls (1)               - list directory contents
ls (1p)              - list directory contents
```

명령어 모음

- id
  - 자신에 대한 uid, gid에 대한 정보
- groups
  - /etc/group 파일을 참고해서 자신이 속한 그룹을 보여줌

```
[sedu50@login03 EDU]$ id
uid=100016349(sedu50) gid=1000163(in0163) groups=1000163(in0163)

[sedu50@login03 EDU]$ groups
in0163

[sedu50@login03 EDU]$ cat /etc/group
root:x:0:
bin:x:1:
daemon:x:2:
sys:x:3:
adm:x:4:
tty:x:5:
disk:x:6:
lp:x:7:
mem:x:8:
…
```


명령어 모음

- passwd
  - 사용자의 암호를 변경
  - 주기적으로 변경해 주는 것이 좋음
  - 해킹되기 쉬운 너무 쉬운 암호는 피하고, 특수문자를 섞어서 쓰는 것이 일반적임

```
[sedu50@login03 khlim]$ passwd
Changing password for user sedu50.
Current Password:   기존 암호 입력
New password:  새로운 암호 입력
Retype new password: 새로운 암호 입력
passwd: all authentication tokens updated successfully.
```

명령어 모음

- free
  - Memory 사용 현황

  $ free -s 초 : 지정한 시간(초)에 지속적으로 정보를 갱신

```
[sedu50@login03 EDU]$ free
              total        used        free      shared  buff/cache   available
Mem:      394875176    33268320   270414420      162968    91192436   335224180
Swap:      16777212           0    16777212
[sedu50@login03 EDU]$ free -h
              total        used        free      shared  buff/cache   available
Mem:           376G         31G        257G        159M         86G        319G
Swap:           15G          0B         15G
[sedu50@login03 EDU]$ free -hs 1
              total        used        free      shared  buff/cache   available
Mem:           376G         31G        257G        159M         86G        319G
Swap:           15G          0B         15G

              total        used        free      shared  buff/cache   available
Mem:           376G         31G        257G        159M         86G        319G
Swap:           15G          0B         15G
…
```


명령어 모음

- history
  - 리눅스는 사용자가 사용했던 명령어들을 기억하고 있으며, 필요할 땐 언제나 다시 불러서 사용할 수 있음
  - 사용자 홈 디렉터리의 .bash_history 파일에 저장돼 있음
  - 방향 Key (Up, Down) 사용(Bash Shell 일 경우)

```
[sedu50@login03 EDU]$ history
 …
  482  19-01-24 11:54:26 id
  483  19-01-24 11:54:38 history 10
  484  19-01-24 11:56:49 history
[sedu50@login03 EDU]$ !482            // 478 번째 수행한 명령어를 수행
id
uid=100016349(sedu50) gid=1000163(in0163) groups=1000163(in0163)
[sedu50@login03 EDU]$ !!              // 가장 최근에 사용한 명령어를 수행
[sedu50@login03 EDU]$ history 5       // 가장 최근에 사용한 명령어 5개를 출력
  482  19-01-24 11:54:26 id
  483  19-01-24 11:54:38 history 10
  484  19-01-24 11:56:49 history
  485  19-01-24 11:57:00 id
  486  19-01-24 11:57:05 history 5
[sedu50@login03 EDU]$ !da             // da로 시작하는 가장 최근의 명령을 수행
date +%Y%m/%d-%H:%M
201901/24-13:12
```

명령어 모음

- ls
  - 디렉터리 내의 파일 목록을 보기 위한 명령

옵션	설 명
-l	파일 및 디렉터리를 list 형식으로 출력
-a	모든 파일, 디렉터리 출력 (숨김 파일 포함)
-R	하위 디렉터리의 파일까지 보여줌
-S	파일 크기가 큰 순서로 출력
-F	파일 뒤에 구분자 표시 (디렉터리는 /, 실행 파일은 *)
-h	용량을 K, M, G와 같은 읽기 쉬운 형태로 보여줌
-r	정렬할 때, 역순으로 보여줌
-t	수정된 시간 순서로 보여줌

명령어 모음

- ls 명령어 기본 사용법

```
[sedu50@login03 etc]$ ls
[sedu50@login03 etc]$ ls -a
[sedu50@login03 etc]$ ls -l
[sedu50@login03 etc]$ ls -la
[sedu50@login03 etc]$ ls –lR
[sedu50@login03 etc]$ ls -F
[sedu50@login03 etc]$ ls –S
[sedu50@login03 etc]$ ls -lrth

[sedu50@login03 etc]$ cd
[sedu50@login03 ~]$ pwd
/home01/sedu50
```

명령어 모음

- alias / unalias
  - 별칭을 설정하고 해제할 수 있음
  - 자주 사용하는 명령어를 입력해두고 간편하게 사용 가능

```
[sedu50@login01 Linux]$ alias
alias cds='cd /scratch/$USER'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias motd='cat /etc/motd'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --
show-tilde'

[sedu50@login01 Linux]$ alias editor='vim'
[sedu50@login01 Linux]$ editor test.txt
[sedu50@login01 Linux]$ unalias editor
[sedu50@login01 Linux]$ editor
-bash: editor: command not found
[sedu50@login01 Linux]$ ]
```

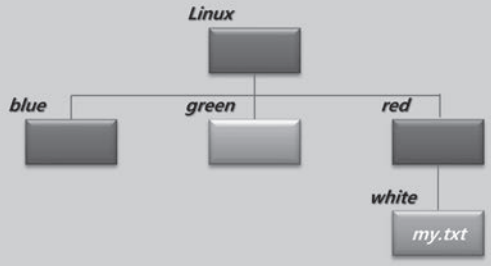

디렉터리 이동/생성/삭제 관련

- cd
  - 디렉터리 이동 명령
- pwd
  - 현재 디렉터리 위치를 보여줌
- mkdir
  - 새로운 디렉터리를 만들 때 사용
- rmdir
  - 디렉터리를 삭제할 때 사용
- touch
  - 파일이 존재할 경우 파일의 수정날짜를 변경
  - 파일이 존재하지 않을 경우 0 kb 파일 생성
- cp
  - 파일 복사 명령
  - 속성을 유지할 경우 -a 옵션 추가

디렉터리 이동/생성/삭제 관련

➢ 기본 사용법 (cd, pwd, mkdir, touch)

```
[sedu50@login03 etc]$ cd
[sedu50@login03 ~]$ pwd
/home01/sedu50
[sedu50@login03 ~]$ cd EDU/
[sedu50@login03 EDU]$ ls
Linux
[sedu50@login03 EDU]$ cd Linux/
[sedu50@login03 Linux]$ mkdir red
[sedu50@login03 Linux]$ mkdir green
[sedu50@login03 Linux]$ mkdir blue
[sedu50@login03 Linux]$ ls
blue  green  red
[sedu50@login03 Linux]$ pwd
/home01/sedu50/EDU/Linux
[sedu50@login03 Linux]$ cd red
[sedu50@login03 red]$ pwd
/home01/sedu50/EDU/Linux/red
[sedu50@login03 red]$ mkdir white
[sedu50@login03 red]$ cd white/
[sedu50@login03 white]$ touch my.txt
[sedu50@login03 white]$ ls -l
total 0
-rw-r--r-- 1 sedu50 in0163 0 Jan 24 14:07 my.txt
```



디렉터리 이동/생성/삭제 관련

➢ 기본 사용법 (cp, clear)

```
[sedu50@login03 white]$ clear
[sedu50@login03 white]$ cp my.txt ../../green/
[sedu50@login03 white]$ cd ../../green/
[sedu50@login03 green]$ pwd
/home01/sedu50/EDU/Linux/green
[sedu50@login03 green]$ ls
my.txt
[sedu50@login03 green]$ ls -l
total 0
-rw-r--r-- 1 sedu50 in0163 0 Jan 24 14:16 my.txt
```

디렉터리 이동/생성/삭제 관련

➢ 기본 사용법 (rm, rmdir)

```
[sedu50@login03 red]$ cd red
[sedu50@login03 red]$ pwd
/home01/sedu50/EDU/Linux/red
[sedu50@login03 red]$ ls
white
[sedu50@login03 red]$ rmdir white/
rmdir: failed to remove ‘white/’: Directory not empty
[sedu50@login03 red]$ cd white/
[sedu50@login03 white]$ ls
my.txt
[sedu50@login03 white]$ rm my.txt
[sedu50@login03 white]$ cd ../
[sedu50@login03 red]$ rmdir white/
[sedu50@login03 red]$ ls
```

디렉터리 안에 파일이 있으면 삭제불가

디렉터리 이동/생성/삭제 관련

➢ 기본 사용법 (rm -rf : 파일이 포함된 디렉터리 지우기)

```
[sedu50@login03 red]$ mkdir white
[sedu50@login03 red]$ cd white/
[sedu50@login03 white]$ touch my.txt
[sedu50@login03 white]$ ls
my.txt
[sedu50@login03 white]$ cd ../
[sedu50@login03 red]$ ls
white
[sedu50@login03 red]$ rm -rf white/
[sedu50@login03 red]$ ls
```

디렉터리 / 파일 관련

- rm
  - 파일이나 디렉터리 (-rf 옵션)를 삭제
  - 옵션
    - -i : 삭제 시 확인
    - -f : 강제 삭제
    - -r : 디렉터리를 삭제할 때 하위 디렉터리와 파일도 모두 삭제
    - -v : rm 명령어 진행 과정 출력
- mv
  - 파일과 디렉터리의 이름을 변경하거나 경로를 옮길 때 사용
  - -b 옵션 사용시 : 같은 파일이 있을 경우는 backup 파일을 생성

디렉터리 / 파일 관련

- 기본 사용법 (rm, mv)

```
[sedu50@login03 Linux]$ mkdir cyan
[sedu50@login03 Linux]$ touch my.txt
[sedu50@login03 Linux]$ ls
blue  cyan  green  my.txt  red
[sedu50@login03 Linux]$ mv my.txt ./cyan/
[sedu50@login03 Linux]$ cd cyan/
[sedu50@login03 cyan]$ ls -l
total 0
-rw-r--r-- 1 sedu50 in0163 0 Jan 24 14:56 my.txt
[sedu50@login03 cyan]$ cp -p my.txt my2.txt
[sedu50@login03 cyan]$ ls -l
total 0
-rw-r--r-- 1 sedu50 in0163 0 Jan 24 14:56 my2.txt
-rw-r--r-- 1 sedu50 in0163 0 Jan 24 14:56 my.txt
[sedu50@login03 cyan]$ mv my.txt ../you.txt
[sedu50@login03 cyan]$ cd ../
[sedu50@login03 Linux]$ ls -l
total 16
drwxr-xr-x 2 sedu50 in0163 4096 Jan 24 14:07 blue
drwxr-xr-x 2 sedu50 in0163 4096 Jan 24 15:00 cyan
drwxr-xr-x 2 sedu50 in0163 4096 Jan 24 14:22 green
drwxr-xr-x 3 sedu50 in0163 4096 Jan 24 14:46 red
-rw-r--r-- 1 sedu50 in0163    0 Jan 24 14:56 you.txt
```

디렉터리 / 파일 관련

- 기본 사용법 (rm, mv)

```
[sedu50@login03 Linux]$ ls -l
total 4
drwxr-xr-x 2 sedu50 in0163 4096 Jan 24 15:00 cyan
-rw-r--r-- 1 sedu50 in0163    0 Jan 24 15:09 you.txt
[sedu50@login03 Linux]$ rm -i you.txt
rm: remove regular empty file ‘you.txt’? y
[sedu50@login03 Linux]$ ls -l cyan/
total 0
-rw-r--r-- 1 sedu50 in0163 0 Jan 24 14:56 my2.txt
[sedu50@login03 Linux]$ rm -rf cyan/
[sedu50@login03 Linux]$ ls -l
total 0
```


디렉터리 / 파일 관련

- cat
  - 간단한 텍스트 파일을 생성 하거나 텍스트 파일 내용을 확인

  ex) cat /etc/passwd

- echo
  - 텍스트를 화면 상에 출력

```
[sedu50@login03 Linux]$ echo "hello"
hello
[sedu50@login03 Linux]$ echo $HOME
/home01/sedu50
[sedu50@login03 Linux]$ echo $PWD
/home01/sedu50/EDU/Linux
```

디렉터리 / 파일 관련

- 기본 사용법 (cat, echo)
  - cat 명령어로 text 파일 작성 후 종료는 ***Ctrl + c*** 를 사용

```
[sedu50@login03 Linux]$ cat >my1.txt
abc
def
^C
[sedu50@login03 Linux]$ cat > my2.txt
abcc
def
^C
[sedu50@login03 Linux]$ ls -l
total 8
-rw-r--r-- 1 sedu50 in0163 8 Jan 24 15:14 my1.txt
-rw-r--r-- 1 sedu50 in0163 9 Jan 24 15:14 my2.txt
[sedu50@login03 Linux]$ echo "Hello, World"
Hello, World
[sedu50@login03 Linux]$ cat my1.txt
abc
def
[sedu50@login03 Linux]$ cat my2.txt
abcc
def
```


디렉터리 / 파일 관련

- head, tail
  - 텍스트 파일의 앞부분, 뒷부분을 특정 라인 수(-n) 만큼 보고 싶을 때
    ex) $ head -10 /etc/passwd
    $ tail -10 /etc/passwd
- more
  - 텍스트 파일 내용을 페이지 단위로 한 화면씩 출력할 때
    ex) $ more /etc/passwd
- ln
  - 파일명이나 경로명을 단순화 시키는 링크 명령
  - 하드링크 : 동일한 inode를 사용하는 파일 생성
  - 심볼릭링크(-s 옵션 사용)
    ex) $ ln –s /etc/passwd passwd
    $ ls –l
    lrwxrwxrwx 1 edun30 edungrp 11 Oct 24 11:12 passwd -> /etc/passwd

디렉터리 / 파일 관련

- wc
  - 줄(-l) 또는 단어(-w) 개수를 출력

ex) $ wc -l /etc/passwd
$ wc -w /etc/passwd

```
[sedu50@login03 Linux]$ cat /etc/passwd
…
[sedu50@login03 Linux]$ wc /etc/passwd
 112  166 5692 /etc/passwd
[sedu50@login03 Linux]$ wc -l /etc/passwd
112 /etc/passwd
[sedu50@login03 Linux]$ wc -w /etc/passwd
166 /etc/passwd
[sedu50@login03 Linux]$ wc -c /etc/passwd
5692 /etc/passwd
[sedu50@login03 Linux]$ wc -lwc /etc/passwd
 112  166 5692 /etc/passwd
```


디렉터리 / 파일 관련

- tar
  - 단순하게 파일을 압축하는 용도가 아닌, 파일이나 디렉터리를 묶는 용도
  - gzip, unzip과 같은 압푹프로그램과 같이 쓰이는 게 일반적

- gzip 압축과 함께 tar로 묶인 .tar.gz 파일 풀기 예
  $ tar -zxf mysrc.tar.gz

- 묶고 압축하기 예
  $ tar -zcf mysrc.tar.gz ./

- 기본적인 Option
  -z : gzip으로 압축 또는 압축해제 할 때 사용
  -f : tar 명령어를 이용할 때 반드시 사용 (default)
  x : tar 파일로 묶여있는 것을 해제할 때 사용 (extract)
  c : tar 파일을 생성할 때 사용 (create)

디렉터리 / 파일 관련

- tar 기본 사용법
  - 실습으로 사용할 filename.tar.gz 형식의 예제파일 선정
  - http://ftp.gnu.org/gnu/autoconf/
  - 실습파일명 : autoconf-2.12.tar.gz

```
[sedu50@login03 Linux]$ cp /tmp/autoconf-2.12.tar.gz .
[sedu50@login03 Linux]$ ls
autoconf-2.12.tar.gz
```


디렉터리 / 파일 관련

- 기본 사용법 (tar) – 첫 번째
  - gzip으로 압축을 풀고, tar로 묶음을 품

```
[sedu50@login03 Linux]$ ls
autoconf-2.12.tar.gz
[sedu50@login03 Linux]$ gzip -d autoconf-2.12.tar.gz
[sedu50@login03 Linux]$ ls
autoconf-2.12.tar
[sedu50@login03 Linux]$ tar xvf autoconf-2.12.tar
[sedu50@login03 Linux]$ ls
autoconf-2.12  autoconf-2.12.tar
```

xvf & -xvf 같은 표현

  - tar로 묶고, gzip으로 압축

```
[sedu50@login03 Linux]$ ls
autoconf-2.12
[sedu50@login03 Linux]$ tar cvf autoconf-2.12.tar autoconf-2.12/
[sedu50@login03 Linux]$ ls
autoconf-2.12  autoconf-2.12.tar
[sedu50@login03 Linux]$ gzip autoconf-2.12.tar
[sedu50@login03 Linux]$ ls
autoconf-2.12  autoconf-2.12.tar.gz
[sedu50@login03 Linux]$ rm -rf autoconf-2.12
```

디렉터리 / 파일 관련

- 기본 사용법 (tar) – 두 번째
  - -z(gzip) 옵션을 사용해서 한꺼번에 압축을 풀고, tar로 묶음을 품

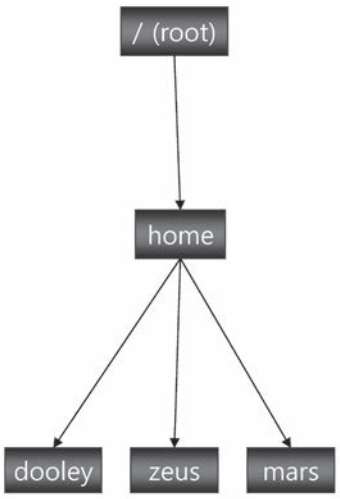
```
[sedu50@login03 Linux]$ ls
autoconf-2.12.tar.gz  my1.txt  my2.txt
[sedu50@login03 Linux]$ tar zxvf autoconf-2.12.tar.gz
[sedu50@login03 Linux]$ ls
autoconf-2.12  autoconf-2.12.tar.gz  dir/
[sedu50@login03 Linux]$ tar -xzvf autoconf-2.12.tar.gz -C dir
[sedu50@login03 Linux]$ ls dir
autoconf-2.12
```

  - -z(gzip) 옵션을 사용해서 한꺼번에 tar로 묶고, gzip으로 압축

```
[sedu50@login03 Linux]$ rm -rf autoconf-2.12.tar.gz
[sedu50@login03 Linux]$ ls
autoconf-2.12 dir/
[sedu50@login03 Linux]$ tar czvf autoconf-2.12.tar.gz autoconf-2.12/
[sedu50@login03 Linux]$ ls
autoconf-2.12  autoconf-2.12.tar.gz dir/
```


절대 경로와 상대 경로

- 절대 경로(Absolute Path)
  - /(root directory) 기준
- 상대 경로(Relative Path)
  - .(current directory) 기준
- (e.g.) zeus -> mars
  - cd /home/mars
  - cd ../mars



파일 속성

➢ ls -al 명령을 사용하여 속성 확인

```
[sedu50@login03 Linux]$ ls -l
total 4
-rw-r----- 1 sedu50 in0163 1475 Jan 24 16:21 my.txt
```

필드 정보	속성/허가권	링크 수	사용자	그룹	크기	생성일자	생성시간	파일명
예제	-rw-r---	1	sedu50	in0163	1516	1월 24일	16시 21분	my.txt

파일 속성

➢ 첫 번째 필드
- 파일 속성 및 허가권을 총 10개의 문자로 표시
- 1+3+3+3 으로 구분하여 읽음
- -rw-r--r-- :
  - - + rw- + r-- + r--
  - 파일 유형 + 소유주(user) 권한 + 그룹(group) 권한 + 나머지(other) 사용자 권한
- 파일 유형

문자	파일의 유형별 의미
-	일반파일
b	파일 입출력과 관련된 블록 디바이스 용도의 장치 파일(/dev/sda, /dev/hda 등)
c	터미널, 네트워크, 프린트, 마우스, 사운드카드와 같은 장치관련 캐릭터 디바이스 용도의 장치파일 (/dev/console 등)
d	디렉터리
l	심볼릭 링크파일
p	파이프
s	소켓

파일 속성

- 파일 허가권 영역
  - 사용자(user) / 그룹(group) / 나머지(other) 권한 설정
  - rwx 로 표기
  - r : 읽기, w : 쓰기, x : 실행, - : 해당 권한 없음
  - 예제 : 654 → 110 101 100 → rw- r-x r--
  - 예제 : 725 → 111 010 101 → rwx -w- r-x

옥텟(숫자)으로 표현한 파일 허가권	권 한
7 (rwx = 4 + 2 + 1)	읽기 / 쓰기 / 실행 가능
6 (rw- = 4 + 2+ 0)	읽기 / 쓰기 가능
5 (r-x = 4 + 0 + 1)	읽기 / 실행 가능
4 (r-- = 4 + 0 + 0)	읽기만 가능
3 (-wx = 0 + 2 + 1)	쓰기 / 실행 가능
2 (-w- = 0 + 2 + 0)	쓰기만 가능
1 (--x = 0 + 0 + 1)	실행만 가능
0 (--- = 0 + 0 + 0)	읽기/쓰기/실행 불가능

파일 속성 변경

- chmod
  - 파일 권한 변경
  - chmod [who] [+/-] [새로운 권한] [파일명]
  - who ?

WHO	설 명
u	파일을 소유한 사용자(user)
g	파일에 대한 소유권을 가진 그룹(group)
o	다른 사용자(other)
a	모든 사용자(all)

  - + 를 사용하여 권한을 더해줄 수 있고, - 를 사용하여 권한을 삭제
  - chmod u+rw my.txt
    - my.txt라는 파일의 사용자 권한 중 r, w를 추가
  - chmod u-w, o+r test.exe
    - test.exe 라는 파일의 사용자 권한 중 user의 w권한 삭제, other의 r권한 추가

파일 속성 변경

➢ chmod 실습


```
[sedu50@login03 Linux]$ cat > date.x
date
^C
[sedu50@login03 Linux]$ ls -l date.x
-rw-r--r-- 1 sedu50 in0163 5 Jan 24 16:34 date.x
[sedu50@login03 Linux]$ ./date.x
-bash: ./date.x: Permission denied
[sedu50@login03 Linux]$ chmod u+x date.x
[sedu50@login03 Linux]$ ls -l
total 4
-rwxr--r-- 1 sedu50 in0163 5 Jan 24 16:34 date.x
[sedu50@login03 Linux]$ ls -lF date.x
-rwxr--r-- 1 sedu50 in0163 5 Jan 24 16:34 date.x*
[sedu50@login03 Linux]$ ./date.x
Thu Jan 24 16:35:47 KST 2019
[sedu50@login03 Linux]$ chmod u-x date.x
[sedu50@login03 Linux]$ ls -lF date.x
-rw-r--r-- 1 sedu50 in0163 5 Jan 24 16:34 date.x
[sedu50@login03 Linux]$ chmod 740 date.x
[sedu50@login03 Linux]$ ls -lF date.x
-rwxr----- 1 sedu50 in0163 5 Jan 24 16:34 date.x*
```


리다이렉팅

➢ 리눅스에서 지원하는 가장 강력한 명령어 방식

- 명령어나 프로그램의 결과를 스크린에 보여주는 것처럼 파일로 방향 전환을 하는 기능
- 파일을 입력 또는 출력으로 사용
- 명령의 결과를 다른 명령의 입력으로 사용할 수 있음
- 실행프로그램의 결과를 파일로 보내기

  ex) $ test.exe > test.out

  $ cat test.out

  ex) $ cal > october.txt

  $ cat october.txt
- 파일의 내용을 명령의 입력으로 사용
  - test.x < input.dat > output.dat
  - mail mark < letter

```
$ cat test.f90
PROGRAM TEST
IMPLICIT NONE
INTEGER::I,J
READ *, I
J=10*I
PRINT*, J
END PROGRAM TEST
$ gfortran test.f90 -o test.x
$ ./test.x
5
          50
$ cat input.dat
5
$ ./test.x <input.dat >output.dat
$ cat output.dat
          50
```

리다이렉팅

- 명령어 간의 파이프 '|'
- 어떤 명령어의 결과가 다른 명령어의 입력이 되도록 해 줌

Ex) $ who | grep "sedu"

- who 명령어는 시스템에 접속한 사용자가 누구인지, 어떤 터미널이 사용되고 있으며, 언제 로그인 했는지를 살펴보는 명령어
- grep 은 주어진 패턴과 일치하는 line 만을 골라내어 보여주는 명령어 (주요옵션 -i : ignore, -v : Not match)
- 'who' 로 나열되는 사용자 중 sedu이 포함된 User가 있는지를 보여줌

```
[sedu50@login03 Linux]$ who | grep "sedu"
sedu50   pts/82       2019-01-24 11:22
sedu50   pts/84       2019-01-24 11:22
```


파일의 검색

- find 명령어 (특정조건에 맞는 파일을 찾음)
  - 명령어 형식
    $ find [찾을 디렉터리 경로] [찾기 옵션] [찾은 후 수행작업]
  - [찾을 디렉터리 경로]
    . : 현재 디렉터리 밑으로 검색
    / : 루트 디렉터리 밑으로 검색, 즉 전체 파일시스템 검색
  - [찾기 옵션]
    -name : 지정한 형식을 갖는 파일 이름
    -user : 특정 파일을 소유하고 있는 소유자의 파일
    -uid n : 특정 uid를 갖는 파일
    -gid n : 특정 gid를 갖는 파일
    -used n : 최근 n일 이후에 변경된 파일
  - [찾은 후 행할 작업]
    -print : 가장 많이 쓰이는 옵션, 찾고자 하는 파일을 보여줌 (-ls 도 가능)
    (-ls 사용시 ls –dlis와 동일한 출력 형태)
    -exec : 찾고자 하는 파일에 대해 특정 명령을 수행

파일의 검색

➢ find 명령어 기본 사용법

```
[sedu50@login03 Linux]$ find / -name passwd -print
/etc/passwd
/usr/bin/passwd
[sedu50@login03 Linux]$ id
uid=100016349(sedu50) gid=1000163(in0163) groups=1000163(in0163)
[sedu50@login03 Linux]$ ls
aaa.x  bbb.x  ccc.txt
[sedu50@login03 Linux]$ cat aaa.x
date
[sedu50@login03 Linux]$ cat bbb.x
cal
[sedu50@login03 Linux]$ cat ccc.txt
[sedu50@login03 Linux]$ find ./ -uid 100016349 -exec sh {} \;
./: ./: is a directory
    January 2019
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

Thu Jan 24 17:41:19 KST 2019
```


시스템 모니터링

➢ top
- 시스템 모니터링

$ top -d 시간(초) : Refresh 초

시스템 총 사용시간 $ uptime 과 동일

CPU, Memory 사용률

```
top - 11:44:05 up 24 days, 18:18, 122 users,  load average: 1.48, 1.57, 1.04
Tasks: 934 total,   2 running, 918 sleeping,  14 stopped,   0 zombie
%Cpu(s):  2.2 us,  4.4 sy,  0.0 ni, 93.3 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem : 39487516+total, 28259836+free, 16995640 used, 95281168 buff/cache
KiB Swap: 16777212 total, 16777212 free,        0 used. 35427776 avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
247894 x1698a02  20   0  147968   4292   1248 S  24.1  0.0  16:47.35 rsync
273168 root      20   0  169124   7688   4828 S  23.4  0.0   1:18.78 sshd
294493 applic    20   0    4340    344    276 R   7.9  0.0   0:00.24 rm
273352 pmg96     20   0   74268   3516   2636 S   5.0  0.0   0:18.00 sftp-server
274996 applic    20   0  137256   3116   1540 S   4.3  0.0   0:06.19 configure
247659 root      20   0  173428  11888   4848 S   2.3  0.0   1:43.12 sshd
436389 root      20   0       0      0      0 S   1.3  0.0   1:17.02 kworker/2:2
   421 root     -51   0       0      0      0 S   1.0  0.0  10:35.21 irq/393-
hfi1_0
287636 sedu50    20   0  175120   3512   1904 R   1.0  0.0   0:02.00 top
 66993 root      20   0 1493084 216724  23684 S   0.7  0.1 165:17.84 klnagent
408474 root      20   0       0      0      0 S   0.7  0.0   0:24.33 kworker/1:2
     1 root      20   0  192528   5692   2520 S   0.3  0.0   9:02.27 systemd
```

시스템 모니터링

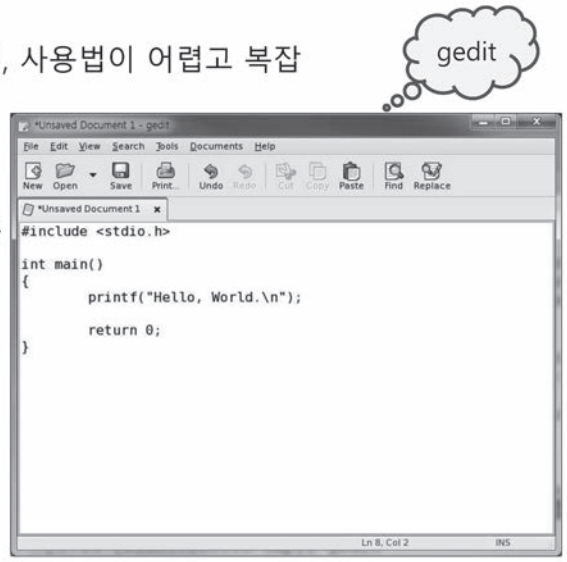
- top
  - CPU 상태 보기
    - '1' : CPU의 상태를 보여줌
    - 'q' : 명령 종료

```
top - 11:46:16 up 24 days, 18:20, 122 users,  load average: 1.68, 1.60, 1.12
Tasks: 934 total,   3 running, 917 sleeping,  14 stopped,   0 zombie
%Cpu0  :  4.0 us,  1.3 sy,  0.0 ni, 94.1 id,  0.0 wa,  0.0 hi,  0.7 si,  0.0 st
%Cpu1  :  1.3 us, 38.2 sy,  0.0 ni, 60.2 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu2  :  1.0 us, 13.2 sy,  0.0 ni, 85.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
…
%Cpu23 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 39487516+total, 28352208+free, 17000044 used, 94353048 buff/cache
KiB Swap: 16777212 total, 16777212 free,        0 used. 35431958+avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
273168 root      20   0  169124   7688   4828 R  28.4  0.0   1:48.66 sshd
247894 x1698a02  20   0  147968   4292   1248 S  23.8  0.0  17:18.32 rsync
295946 applic    20   0    6580    224    172 R   5.9  0.0   0:00.18 sed
273352 pmg96     20   0   74268   3516   2636 S   4.3  0.0   0:22.48 sftp-server
    13 root      rt   0       0      0      0 S   3.0  0.0   1:33.20 migration/1
247659 root      20   0  173428  11888   4848 S   2.6  0.0   1:46.25 sshd
    12 root      rt   0       0      0      0 S   2.0  0.0   0:23.20 watchdog/1
```


Vi Editor

Editor(편집기) 종류

- vim(vi)
  - 가장 기본적인 텍스트 에디터, OS에 기본적으로 포함
- emacs
  - 강력하고 기능이 다양한 에디터, 사용법이 어렵고 복잡
- gedit
  - GUI 기반의 텍스트 에디터
  - 메모장과 비슷해서 사용이 쉬움



Editor(편집기) 소개

- VI
  - BSD의 C Shell을 개발한 빌 조이가 1976년에 개발한 문서 편집기
  - 'Visual edit' 준말로 종이에 글을 쓰듯이 문서를 작성
  - 기존에는 명령어로 한 줄씩 문서를 편집 (라인 에디터를 사용)
- VIM
  - VI Improved의 준말로 기존의 vi 기능에 여러 편리한 기능이 추가
  - 확장된 정규 표현식 문법, highlighting 기능, 다중 되돌리기, 유니코드 등
- EMACS
  - 1976년에 GNU 창시자인 리차드 스톨만이 TMACS를 발전시켜서 만듦
  - 편집기 기능 외에도 파일 관리, 디버거, 웹 브라우저, ftp 등 수많은 기능 제공
  - Lisp 언어로 만들어졌으므로 lisp 언어로 기능을 추가할 수 있음

vi 에디터란?

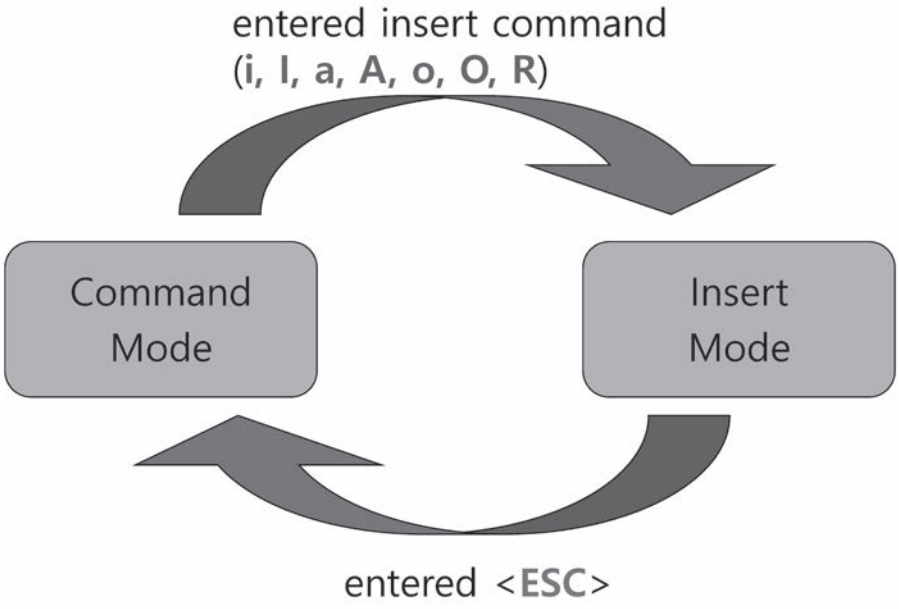
- vi editor
  - VIsual display editor 를 의미
  - 처음 접하는 사람에게 쉽지 않음
  - 대부분의 유닉스 계열 시스템에 설치 되어 있음
  - 워드 프로세서의 기능의 상당 부분을 가지고 있음
  - vim 이라는 vi 의 클론이 포함되어 있음

vi editor

- Open a file
  - $ vi file (편집 모드)
  - $ view file (읽기 모드)

- modes
  - Insert Mode(입력 모드)
    - 입력하는 모든 것이 편집 버퍼에 입력됨
    - 입력모드에서 빠져 나올 때(명령 행 모드로 변경 시) : "ESC" key
  - Command Mode(명령 행 모드)
    - 입력하는 모든 것이 명령어 해석됨

vi editor



vi editor

➢ 기본 단위 커서 이동

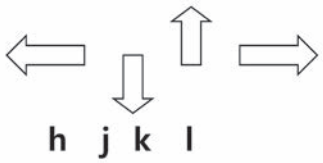
h	커서를 한 문자 왼쪽으로 이동
j	커서를 아래 줄 같은 칼럼으로 이동
k	커서를 위 줄 같은 칼럼으로 이동
l	커서를 한 문자 오른쪽으로 이동
<ENTER>	커서를 다음 줄의 첫 문자로 이동 (10ENTER)
방향키	← ↓ ↑ →

➢ 화면 단위 커서 이동

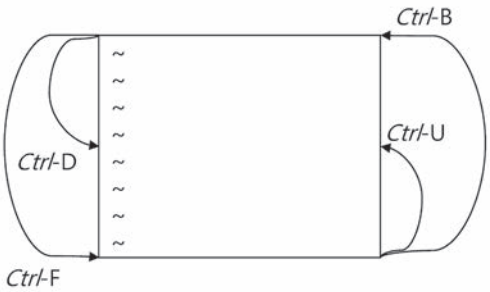
H	화면의 맨 위쪽으로 커서 이동
M	화면의 중간 줄로 커서 이동
L	화면의 맨 아래 줄로 커서 이동

vi editor

➢ 화면 커서(Cursor) 이동
  – nn 번 줄로 이동 : nn<Enter>, nnG



h j k l



vi editor

➢ 단어, 문장, 절 단위 이동

w	다음 단어의 첫 문자로 커서 이동
b	커서가 위치한 앞 방향의 단어로 커서 이동
e	커서가 위치한 단어의 끝으로 커서 이동
)	다음 문장으로 커서 이동
}	다음 절로 커서 이동
(	)와 반대
{	}와 반대
0	현재 줄의 첫 문자로 커서 이동
^	현재 줄의 처음으로 커서 이동 (탭이나 공백이 아닌)
$	현재 줄의 마지막으로 커서 이동

vi editor

➢ 단어, 문장, 절 단위 이동

^F	한 화면 아래로 이동
^B	한 화면 위로 이동
^D	반 화면 아래로 이동
^U	반 화면 위로 이동
:$	마지막 줄로 이동
G	nG : 줄 번호 n으로 이동 1G : 편집 버퍼의 첫 줄로 이동하기 G : 편집 버퍼의 마지막 줄로 이동하기

vi editor

➢ 데이터 첨가, 삽입, 치환, 해제

a	현재 커서 위치 오른쪽부터 데이터 첨가
i	현재 커서 위치의 왼쪽부터 데이터 첨가
o	현재 줄 아래 줄을 공백 줄로 만듦
A	현재 줄의 마지막에 데이터 첨가
I	현재 줄의 첫 문자 앞에 문자 삽입
O	현재 줄의 위 줄을 공백 줄로 만듦
r	현재 커서가 위치한 한 문자 치환
R	현재 커서 위치에서 <esc>키를 칠 때까지 문자 치환
cw	현재 커서 위치의 단어를 다른 단어로 치환
C	현재 커서의 위치부터 그 줄의 마지막까지 치환
s	현재 커서의 문자를 치환
S	현재 커서의 줄을 치환
cc	현재 커서가 위치한 줄을 다른 내용으로 치환
~	현재 커서 위치의 한 문자를 소문자, 대문자로 전환
<ESC>	데이터 입력 모드에서 벗어남

vi editor

KISTI 과학데이터스쿨

➢ 데이터 삭제

x	현재 커서가 위치한 한 문자삭제
X	현재 커서가 위치한 앞 문자를 삭제
dw	현재 커서가 위치한 오른쪽 단어 삭제
db	현재 커서가 위치한 왼쪽 단어 삭제
dd	현재 커서가 위치한 줄 삭제
D	현재 커서가 위치한 곳에서 오른쪽의 내용삭제
:n1, n2 d	n1 ~ n2 라인을 삭제

➢ 명령의 취소 및 반복

u	바로 앞에서 수행한 명령 취소
U	한 줄 내에서 수행한 명령 취소
.	바로 앞에서 수행한 명령 재수행

vi editor

KISTI 과학데이터스쿨

➢ 데이터 이동과 복사

yw	현재 커서가 위치한 단어를 버퍼에 복사
yy 10yy	현재 줄이 버퍼에 복사 10줄을 복사
nY	현재 커서 위치로부터 n 줄 만큼 복사
p	현재 커서 오른쪽 또는 아래 줄에 버퍼 내용 복사
P	현재 커서의 왼쪽 또는 위 줄에 버퍼 내용 복사
: n1, n2 y	n1부터 n2 라인까지 복사

➢ 줄의 결합

J	현재 줄과 다음 줄을 연결

vi editor

➢ vi 종료

:x ZZ	파일을 디스크에 저장한 후 vi를 벗어난다
:wq	파일을 디스크에 저장한 후 vi를 벗어난다
:wq 파일명	기존 파일명을 새로운 파일명에 저장하고 vi 벗어남
:q!	파일을 디스크에 저장하지 않고 vi에서 벗어남

➢ 기타 명령어

:set nu	vi 상태에 있는 파일에 줄 번호를 부여
:set nonu	파일에 있는 줄 번호 취소

vi editor

➢ vi 옵션들

-t tag	tag를 포함하는 파일과 위치에 편집
-r 파일명	지정한 파일을 복구
-Wn	생략시의 윈도우 크기를 n으로 한다
-R	읽기 전용 모드(파일의 overwrite 방지)
+명령어	편집 전에 지정한 ex 명령어 수행
-L	모든 파일의 이름이 출력
-C 명령어	지정된 편집기의 명령어를 실행함으로써 편집 시작
view 파일명	파일을 vi의 읽기 전용모드로 화면에 표시
vi file1 file2 file3	파일 편집을 위해 3개 파일을 vi로 부른다.

vi editor

➢ 검색 관련키

/검색어	커서가 위치한 행포함 아래로 해당 검색어 검색
?검색어	커서가 위치한 행 포함 위로 해당 검색어 검색
n or /	커서의 위치를 검색한 검색어의 다음 위치로 이동
N or ?	커서의 위치를 검색한 검색어의 이전 위치로 이동

➢ 파일 관련 명령어

:w 파일명	변경한 파일 내용 저장(:w 파일명 -> 해당파일명으로 저장) (1,5w 파일명 -> 1부터 5번째 라인까지 해당파일로 저장)
:q	Vi 종료
:e 파일명	Vi 실행 후 특정 파일을 불러들여 편집 :e# 편집했던 이전 파일을 다시 연다
:r 파일명	특정 파일을 불러들임(1r 파일명 -> 1행 뒤에 해당 파일의 내용을 삽입)

vi editor

➢ 치환 관련 명령

:s/pattern/replace	현재 줄에서 치환
:%s/old/new/g	파일 내의 특정 단어를 새로운 단어로 치환(문장 치환 시 해당 문장을 " "으로 묶음)
:%s/^old/new/g	파일 내의 특정 단어로 시작하는 단어만 새로운 단어로 치환(해당 단어가 중간에 포함된 단어 제외)
:%s/old$/new/g	파일 내의 특정 단어로 끝나는 단어만 새로운 단어로 치환(해당 단어가 중간에 포함된 단어 제외)
:%s/aaa//g	파일 내의 aaa라는 단어를 삭제(문장 삭제 시 해당 문장을 ""으로 묶음)
:10,50s/old/new/g	파일 내의 10행에서 50행사이의 특정 단어를 새로운 단어로 치환(문장 치환 시 해당 문장을 " "으로 묶음)

vi editor

➢ vi 환경에서 shell 명령을 수행

– 명령 모드에서 ":!" 을 입력하고 원하는 shell command 입력

:!ls	Shell로 돌아가 'ls' 명령 수행
:!pwd	현재 작업 directory 보여줌
:r !command	command(명령어)의 결과를 커서 위치부터 삽입

– 명령을 수행한 후에 enter를 치면 vi 편집기 명령모드로 복귀

➢ 여러 파일의 열기 / 편집 / 저장

– vi file1 file2
– vi *.txt
– :n 다음 파일로 이동, :N 이전 파일로 이동

vi editor 연습

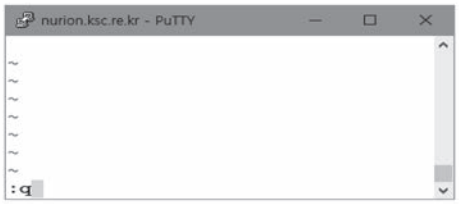
➢ 파일 생성 및 편집

– 처음에는 명령 모드로 실행
– 편집 모드로 전환해야 글자 편집 가능
– ' a ', ' i ' 를 누르면 명령모드에서 편집모드로 전환됨
  • ' a ' : 현재 커서 위치에서 오른쪽부터 데이터 첨가 (**a**ppend)
  • ' i ' : 현재 커서 위치에서 왼쪽부터 데이터 첨가 (**i**nsert)

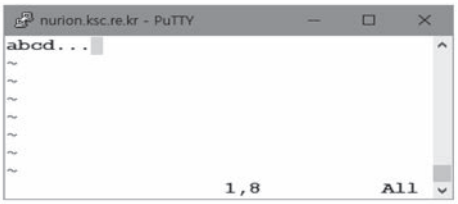
vi 편집기로 파일 생성

```
$ vi filename
```

명령 모드



편집 모드

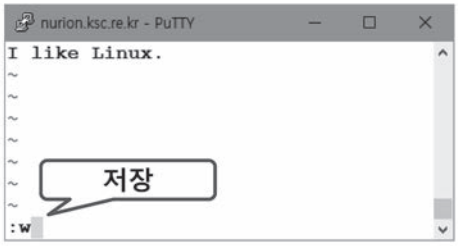


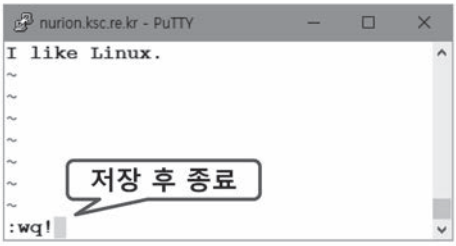
vi editor 연습

➢ 저장 및 종료

- 편집 모드에서 명령 모드로 전환 (' ESC ' 키 누름)
- 저장 : ' **ESC** ' + ' **:w** '
- 저장하지 않고 종료 : ' **ESC** ' + ' **:q** '
- 저장 후 종료 : ' **ESC** ' + ' **:wq!** '

(' wq ' 명령은 저장 후 종료, ' ! ' 는 강제 저장, 종료를 의미)





vi editor 연습

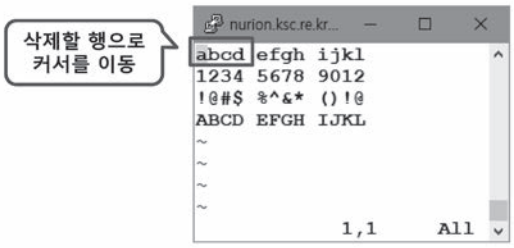
➢ 편집

- 명령 모드에서 실행

1) 삭제

x : 커서 위치의 글자 삭제
dw : 한 단어를 삭제
dd : 커서가 있는 행을 삭제
3dd : 커서로부터 3개의 행이 삭제

삭제 실행 전



한 글자 삭제 (x)

한 단어 삭제 (dw)

한 행 삭제 (dd)

3행 삭제 (3dd)

vi editor 연습

➢ 편집
- 명령 모드에서 실행

2) 복사와 붙이기

yw : 현재 커서 위치의 단 단어를 복사
3yw : 현재 커서 위치에서 3개의 단어를 복사
yy : 커서 위치의 한 행을 복사
3yy : 커서 위치에서 3 행을 복사

p : 복사한 내용을 붙여 넣음
2p : 복사한 내용을 2번 붙여 넣음

1) 2단어 복사(2yw) 커서 이동 — 붙여 넣기 (p)

2) 2 행 복사(2yy) 커서 이동 — 붙여 넣기 (p)

vi editor 연습

➢ 되돌리기와 되살리기
- 명령 모드에서 실행
- 되돌리기 : ' **u** '
- 되살리기 : ' **Ctrl** ' + ' **r** '

1) 수정 전

2) 수정 후

명령모드 전환(ETC)

3) 되돌리기 수행

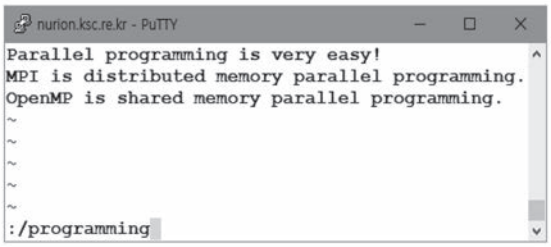
수정하기 전 상태로 돌아감

4) 되살리기 수행

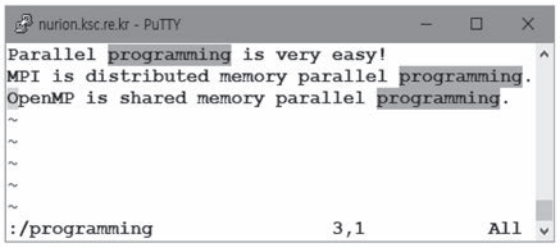
vi editor 연습

- 문자열 탐색
  - 명령 모드에서 실행
  - / [찾고자 하는 문자열] 또는 ? [찾고자 하는 문자열]
  - 실행 후 ' n ' 을 누르면 다음 매칭 문자열로 이동
  - 실행 후 ' N ' 을 누르면 이전 매칭 문자열로 이동

문자열 탐색 실행

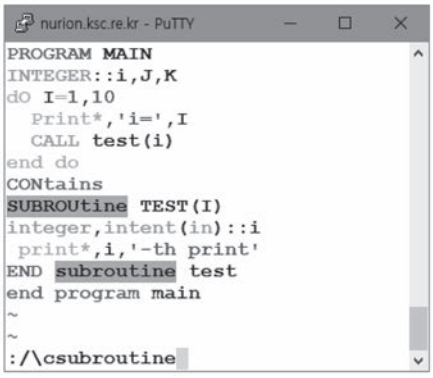


검색된 문자열 표시



vi editor 연습

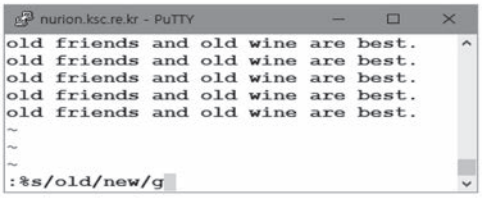
- 문자열 탐색
  - '₩c' 대소문자 구분하지 않음
  - 대소문자를 구분하지 않는 코드 작성할 때 편리



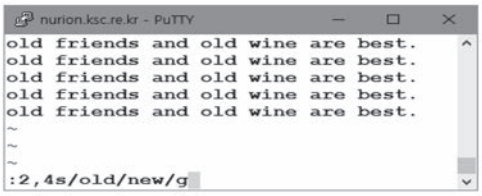
vi editor 연습

- 문자열 치환
  - 명령 모드에서 실행
  - 치환 명령어 : ' **%s/old/new/g** '

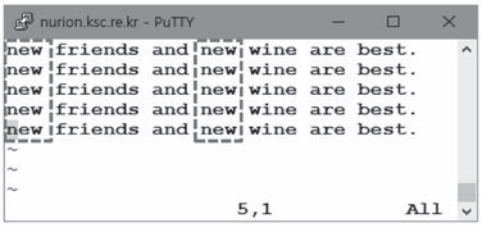
전체 범위의 'old'문자를 'new'로 치환



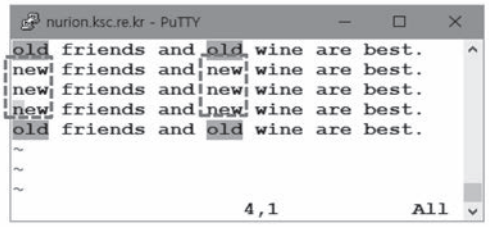
2행부터 4행까지 문자열 치환



전체 범위 치환 결과



2행부터 4행까지 치환 결과

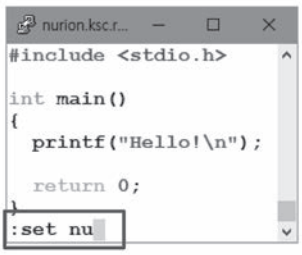


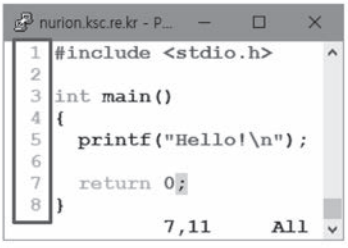
vi editor 연습

- 줄 번호 표시 및 이동
  - 명령 모드에서 실행
  - 줄 번호 표시 : ' **:set nu** '
  - 줄 번호 삭제 : ' **set nonu** '
  - 줄 번호 이동 : ' **: 줄 번호** '

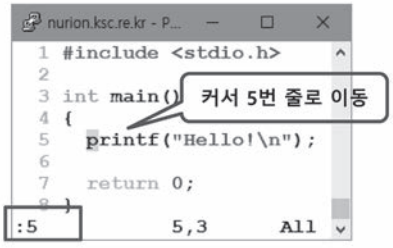
(' wq ' 명령은 저장 후 종료, ' ! ' 는 강제 저장, 종료를 의미)

줄 번호 표시





줄 번호 이동



vi editor 연습

- 치환 관련 명령(^M 문자 삭제)
  - MS 윈도우에서 작성된 파일을 vi로 열면 ^M문자가 나타남
    - MS 윈도우의 개행 문자는 '₩r₩n'인데 반해, 유닉스 계열은 '₩n'이기 때문
  - ':s/^M//g
    - ^M : Ctrl+v+Enter
  - dos2unix 명령 이용

nurion.ksc.re.kr - PuTTY

```
PROGRAM TEST^M
INTEGER::I,j,k^M
dO I=1,10^M
  Print*,'i=',I^M
  CALL test(i)^M
end do^M
CONtains^M
SUBROUtine TEST(i)^M
integer,intent(in)::i^M
  print*,i,'-th print'^M
END subroutine^M
end program main^M
~
~
:%s/^M//g
```


vi editor 연습

- 블록 지정
  - 원하는 위치에서 v를 누르고 블록을 지정
  - 사각 블록 지정 : ctrl + v

nurion.ksc.re.kr - PuTTY

```
PROGRAM TEST
INTEGER::I,j,k
dO I=1,10
  Print*,'i=',I
  CALL test(i)
end do
CONtains
SUBROUtine TEST(i)
integer,intent(in)::i
  print*,i,'-th print'
END subroutine
end program main
                2,15          All
```

nurion.ksc.re.kr - PuTTY

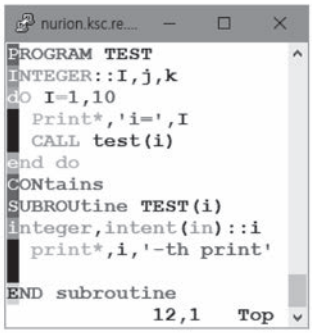
```
PROGRAM TEST
INTEGER::I,j,k
dO I=1,10
  Print*,'i=',I
  CALL test(i)
end do
CONtains
SUBROUtine TEST(i)
integer,intent(in)::i
  print*,i,'-th print'
END subroutine
end program main
                2,7           All
```

  - 블록 지정 후 사용 가능한 키

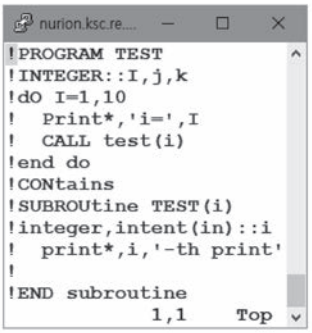
~	대소문자 전환	d	삭제
y	복사	U	대문자로 만듦
>	라인 앞에 탭 삽입	u	소문자로 만듦
<	라인 앞에 탭 제거		

vi editor 연습

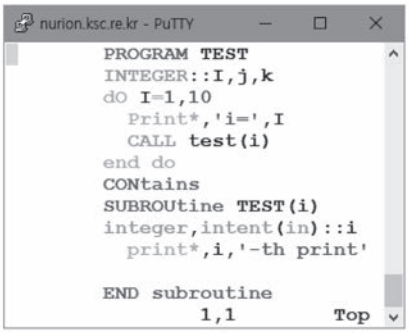
➢ 블록에 일괄 삽입
- 사각 블록 지정 : ctrl + v
- 대문자 I 입력
- 원하는 키 입력
- ESC를 두 번 누름



● 원래 파일(블록 지정)



● '!' 입력



● Tab 키 입력

vi editor 연습

➢ 예제 문장

I hope that the KISTI Supercomputing Center is known as the place where the dreams of Korean scientists and engineers are becoming a reality.

I hope that the Center contributes to furthering national public welfare and also to opening up a bright future for the wold.

It is my pleasure to watch as these dreams are realized through scientists' creativity and engineers' efforts using the tool of supercomputing.

vi editor 연습

➢ 실습 방법

1. 앞 예제 에서 작성한 파일 이름을 contact1.txt 변경
2. contact1.txt 를 contact2.txt 로 복사
3. contact1.txt 와 contact2.txt 를 novel.txt 파일로 합치기
4. wold -> world 로 수정 (검색 후)
5. 4, 5번째 줄 삭제


KISTI
과학데이터스쿨
Thank you!
SUPERCOMPUTING EDUCATION CENTER
86

슈퍼컴퓨터 5호기
NURION 활용

[KISTI-충북대 SW중심대학사업단 HPC 여름학교]

국가와 국민을 위한
데이터 생태계 중심기관 KISTI

슈퍼컴퓨터 5호기 NURION 활용

2020. 8. 24.

슈퍼컴퓨팅인프라센터 윤준원

KISTI과학데이터스쿨
KISTI SCIENCE DATA SCHOOL

Agenda

KISTI 과학데이터스쿨

- 13:00 – 13:50 Introduction & 5th Supercomputer NURION
- 14:00 – 14:50 Basic Environment
- 15:00 – 15:50 Job Scheduler (PBS)
- 16:00 – 16:50 PFS & Burst Buffer
- 16:50 – 17:00 Summary

Refrences

- KISTI 국가슈퍼컴퓨팅센터 (https://www.ksc.re.kr)
  - 슈퍼컴퓨팅 자원 소개, 사용자 사용 신청 및 상담, 지침서 등 제공
- KISTI blog (http://blog.ksc.re.kr/)
- KISTI 국가슈퍼컴퓨팅센터 YouTube 채널
  - https://www.youtube.com/results?search_query=국가슈퍼컴퓨팅센터
- Etc.
  - PBS Works & PBS Professional 매뉴얼 (Altair)
  - Intel Xeon Phi Processor High Performance Programming 2nd Edition (Knights Landing Edition)

Agenda

- Motivation
  - Supercomputer?
  - 성능 결정 요인
  - TOP 500 LIST
  - 병렬 컴퓨터
    - 대표적인 S/W
- 슈퍼컴퓨터 5호기(누리온) 소개
  - KISTI Supercomputer 역사
  - 누리온 시스템 구축/성능
- Basic Environment(1)
  - 실습 시스템 접속
  - Linux 기초
  - Environment Module

Agenda

- Architecture Overview
  - KNL Architecture
  - Vector Registers
  - Performance(Flops)
  - Instruction Set Architecture
  - Memory Modes
  - numactl
- Basic Environment(2)
  - 작업 디렉터리 및 쿼터 정책
  - 프로그래밍 도구 설치 현황
  - 상용 소프트웨어 설치 정보
  - 프로그램 컴파일

Agenda

- Job Scheduler
  - 누리온 큐
  - PBS 명령
  - Job script 작성
  - Job script 예제
  - Interactive 작업 제출
- PFS & Burst Buffer 사용법
  - Rack configuration
  - Nurion storage summary (Burst Buffer, PFS)
  - Nurion storage user quota & policy
  - Burst Buffer IME
  - How to use Burst Buffer on Nurion
- 부록 :
  - 예제 소스 코드
  - 프로파일러 & 디버거 사용법
    - DDT, Intel vtune

Motivation

1. **Supercomputer ?**
2. 성능 결정 요인
3. **TOP 500**
4. 병렬컴퓨터
   - MPI
   - OpenMP
   - 선형대수 라이브러리

Supercomputer

- Supercomputer의 정의
  - 당대의 기술로 가능한 최고 성능의 컴퓨터
    - 대의 범용 컴퓨터에 비해 수백 ~ 수천 혹은 그 이상의 성능
    - 주로 과학기술 연산에 사용됨
    - 컴퓨터의 계산 성능(FLOPS)으로 순위를 따져서 구분
      - TOP 500 List
    - HPC(High-Performance Computer)라고도 함
- Supercomputer의 필요성
  - 빠른 컴퓨터 필요
  - 큰 용량 필요
  - 정확한 해 필요

Supercomputer

- 현재의 모든 Supercomputer는 다수의 프로세서를 사용
- 왜 병렬인가 ?
  - 단일 컴퓨터의 한계
  - 큰 규모의 문제
  - 시간과 비용 절약

프로세서의 구분

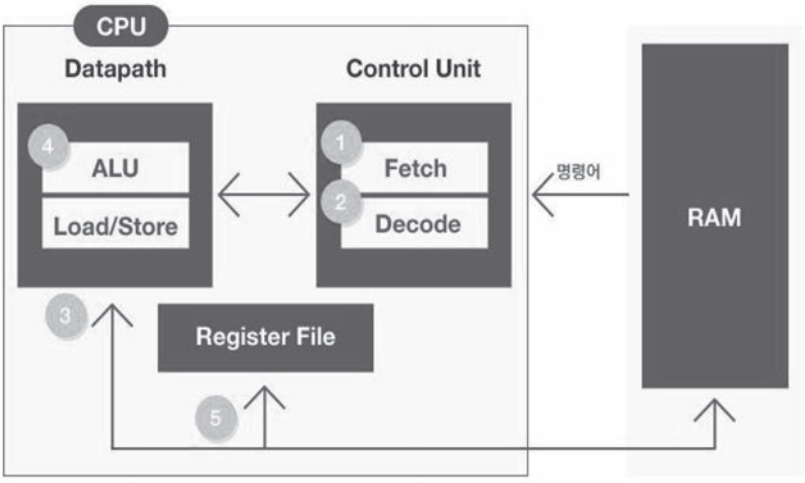
- 스칼라 프로세서
  - 한번에 하나의 데이터(정수 또는 부동소수)를 처리하는 CPU
  - CPU로 가장 단순한 방식의 프로세서
- 벡터 프로세서(Vector processor) / 어레이 프로세서(Array processor)
  - 다수의 데이터를 처리하는 명령어를 가진 CPU
    - 벡터: 1차원 배열의 데이터
  - SIMD(Single Instruction Multiple Data) 명령어 지원

성능 결정 요인

- Supercomputer 성능 결정 요인
  - Hertz(Machine cycle)
    - 프로세서의 동작 주기
    - 반도체 기술에 의해 좌우되며 기술의 한계에 접근
  - CPI(Cycles Per Instruction)
    - 명령어 당 소요 사이클 수
    - 진보된 스칼라 프로세서들의 목표는 CPI 값을 낮추는 것
    - 진보된 스칼라 프로세서들의 CPI 값 감소 배경
      - 파이프라이닝
      - 슈퍼스칼라 등

성능 결정 요인

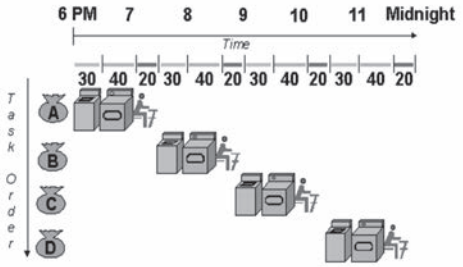
- 파이프라이닝
  - Control Unit 명령어의 실행을 제어
  - Data Path 데이터의 연산 처리를 담당

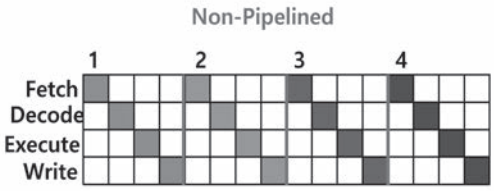


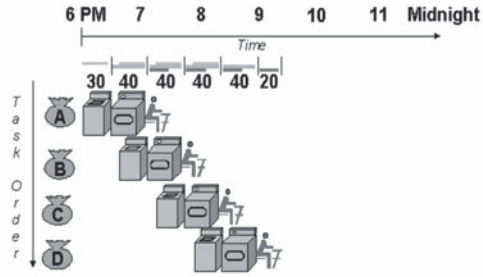
https://www.samsungsds.com/global/ko/news/story/1203239_2919.html

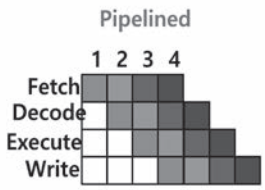
성능 결정 요인

- 파이프라이닝





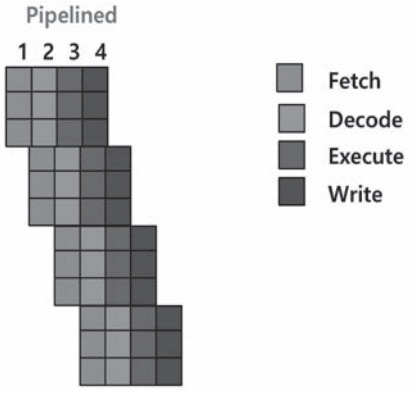


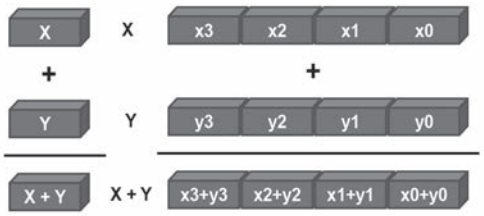


https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/pipelining/index.html

성능 결정 요인

- Superscalar
  - CPU 내에 파이프라인을 여러 개 두어 여러 명령어를 동시에 실행하는 기술
- 파이프 라인 (Pipeline)이 복수의 명령어를 동시에 처리하는 것이라면 벡터 프로세싱은 복수의 데이터를 동시에 처리하는 것

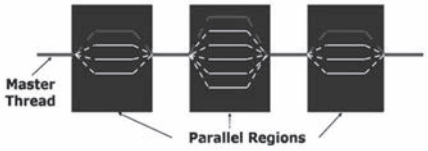


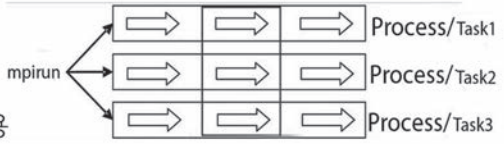


- superscalar
- vectorization

병렬 컴퓨터

- 현재의 슈퍼컴퓨터는 다수의 프로세서를 사용하는 병렬 컴퓨터
- Supercomputer를 분류하는 데는 여러 다른 방법이 존재
  - Supercomputer를 명확하게 분류할 수 있는 다른 구분 방법은 없음
- 메모리 접근에 의한 구분 방법
  - 공유 메모리 시스템
    - OpenMP 병렬 프로그래밍 기법 사용
  - 분산 메모리 시스템
    - MPI 병렬 프로그래밍 기법 사용
  - 공유 분산 메모리 시스템
    - MPI + OpenMP 병렬 프로그래밍 기법 사용

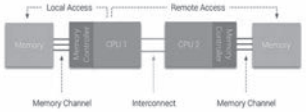


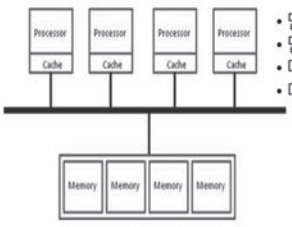


http://www.umsl.edu/~siegelj/CS4740_5740/MPIandOpenMP/MPI_OMP.html

병렬 컴퓨터

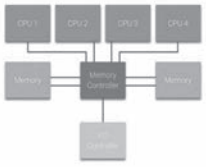
- 공유 메모리 시스템

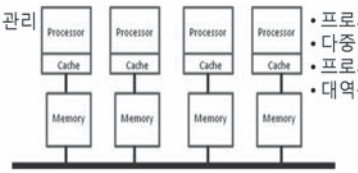




<UMA(SMP)>

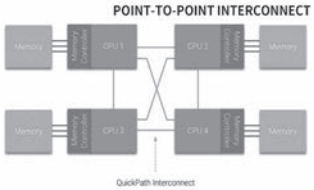
- 단일 운영체제가 모든 프로세서 관리
- 단일 메모리 컨트롤러 사용
- 메모리 액세스 시간 동일
- 대역폭 제한



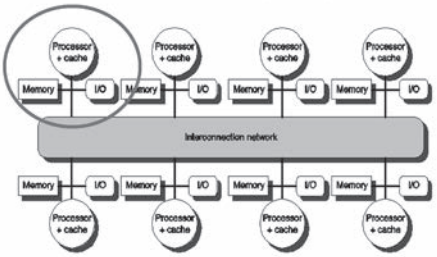


<NUMA>

- 프로세서마다 독립적인 메모리 할당
- 다중 메모리 컨트롤러 사용
- 프로세서 거리에 따라 차이
- 대역폭 향상 가능



- 분산 메모리 시스템 (MPP)



- 시스템의 CPU 간 데이터 링크인 QuickPath point-to-point Interconnect (QPI) 도입

https://frankdenneman.nl/2016/07/07/numa-deep-dive-part-1-uma-numa/

- 독립적인 컴퓨터를 고속의 네트워크로 연결 (클러스터)
- 운영체제를 개별적으로 탑재한 독립적인 노드(물리적 구분)
- 노드들 간에 메모리가 공유되지 않음
- 확장이 용이하여 대규모 대용량의 데이터 처리에 가장 적합한 구조

Nurion 대표 S/W

- Compilers
  - GCC, Intel Compiler, Cray Compiler
- MPI
  - Message Passing Interface
  - 프로세서 사이의 통신과 데이터 교환을 위한 라이브러리
  - 분산 메모리 시스템에서 사용됨
  - MPICH, OpenMPI, IMPI
- OpenMP
  - 대표적인 공유 메모리 구조를 위한 프로그래밍 규약
  - 컴파일러에 의해 지원
- 수치 라이브러리 : BLAS, LAPACK, ScaLAPACK, Intel MKL, etc.

Parallel Computing

- 작업(Task)을 여러 개의 세부 작업(Sub-Task)으로 구분하고, 각각의 작업들을 동시에 수행함으로써 성능 향상
- 병렬 연산의 조건 : 상호간에 종속성이 없어야 함
  - Bernstein's conditions : 두 개의 작업 S1와 S2에 대해서 다음 조건 만족

$I(S_j) \cap O(S_i) = \Phi$	**flow independence**
$O(S_i) \cap O(S_j) = \Phi$	**output independence**
$I(S_i) \cap O(S_j) = \Phi$	**anti independence**

e.g. 1

s_1 : A := x + B;

s_2 : C := A * 3;

s_2 is **flow dependent** on s_1 indicated by an arrow directed from s_1 to s_2.

e.g. 2

s_1 : A := x + B;

s_2 : A := 3;

s_2 is **output dependent** on s_1 indicated by an arrow directed from s_1 to s_2 with a small circle anywhere on the arrow.

e.g. 3

s_1 : B := A + 3;

s_2 : A := 3;

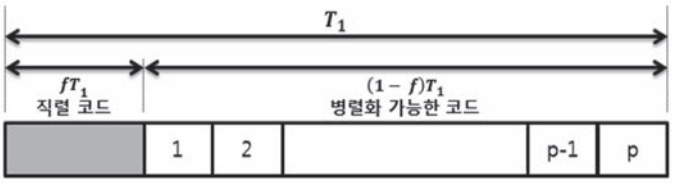
s_2 is **anti dependent** on s_1 indicated by an arrow directed from s_1 to s_2 with a small line across the arrow.

Parallelism in Algorithms – Bernstein's Conditions

Processing speed of Computer

- 프로세스 개수와 성능 향상 관계
  - 프로세서 개수에 비례해 지속적으로 증가 ?

$$speedup = \frac{\text{직렬 프로그램의 실행 시간(1개 프로세서)}}{\text{병렬 프로그램의 실행 시간(}p\text{개 프로세서)}} = \frac{T_1}{T_p}$$

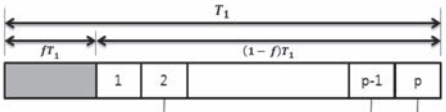


Processing speed of Computer

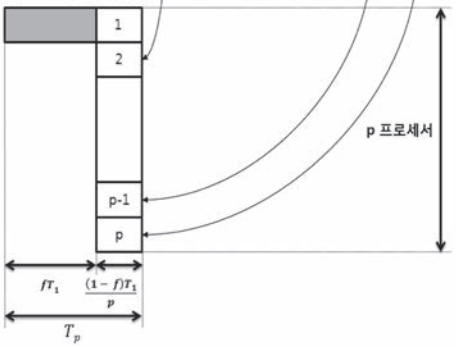
- 병렬 프로그램 실행시간

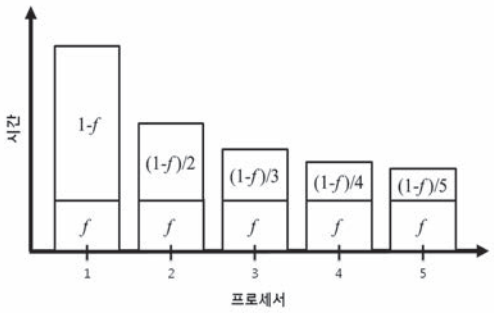
$$speedup = \frac{T_1}{T_p} = \frac{T_1}{fT_1 + \frac{(1-f)T_1}{p}} = \frac{T_1}{T_1\left(f + \frac{(1-f)}{p}\right)} = \frac{1}{f + \frac{(1-f)}{p}}$$

▪ 직렬 프로그램의 실행 시간



▪ 병렬 프로그램의 실행 시간



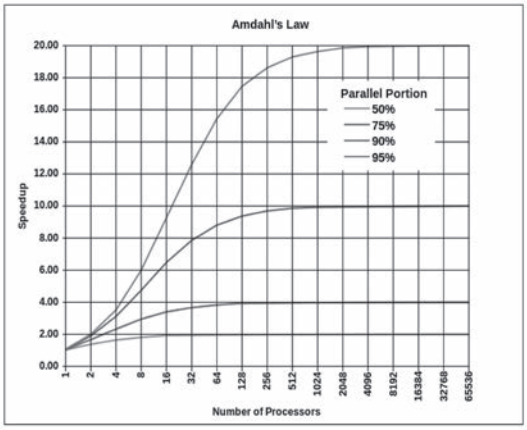


Processing speed of Computer

➢ 예) 직렬 프로그램의 90%가 병렬화 가능한 코드로 구성되어 있다면, 16개의 프로세서에서 병렬화된 프로그램의 성능 향상은?

$$speedup = \frac{1}{0.1 + \frac{(1-0.1)}{16}} = \frac{1}{0.1 + \frac{0.9}{16}} = \frac{1}{0.1 + 0.05625} = \frac{1}{0.15625} = 6.4$$

– 프로세서의 개수에 따른 성능 향상



● *Amdahl's Law* $= \frac{1}{(1-P)+\frac{P}{S}}$

S: 병렬처리 프로세서의 개수
P: 병렬화가 가능한 Instruction의 수

● *Gustafson's law*

✓ 고정된 작업량과 고정된 문제 크기만 고려
→ 작업량(workload) 에 대해서는 고려하지 않음

✓ 코드가 100% 병렬화가 된다고 가정하더라도, 버스 통신 부하, Load Balancing, 발열 등으로 성능이 무한대, 혹은 O(N)배로 늘어나지는 못함

프로그램의 100%를 병렬화 시키는 것은 사실상 불가능 하나, 병렬화 가능 코드 영역을 극대화하는 것이 중요 !

시스템 성능 측정

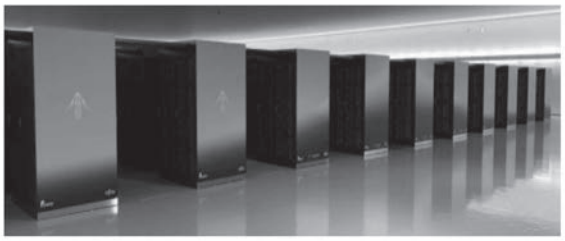
➢ 시스템 성능 (FLOPS: floating-point operations per second)
– 초당 수행할 수 있는 부동 소수점(floating-point) 연산 회수

➢ 이론성능(Rpeak) vs. 실제성능 (Rmax)
– 이론값(theoretical peak performance)은 주로 Rpeak로 표시
– Rpeak = CPU frequency(cycle/s) x Operation per cycle(flops/cycle) X Number of cores
– 실제성능(Rmax)은 여러가지 테스트를 통해 얻어낸 실제로 측정된 성능

➢ 데이터 전송 능력
– 계산에 필요한 데이터를 연산 장치로 공급하는 능력
– CPU↔메모리, 메모리 ↔디스크(파일시스템), 시스템 ↔시스템(인터커넥트)

TOP 500

- 세계에서 가장 빠른 컴퓨터 500
  - 1993년 이후 실시
  - 매년 6월, 11월 두 차례 발표
  - http://www.top500.org



Fugaku in Japan is the first ARM-based supercomputer to be the world's most powerful

- 성능 평가 기준 : LINPACK 벤치마크
  - N by N 행렬로 구성된 연립방정식의 해를 구하는 성능을 Flops 단위로 표시
- Supercomputer 시장의 동향, 아키텍처 및 관련 기술 발정 방향에 대한 중요 지표

이미지 출처 : https://www.eenewspower.com/news/arm-based-exascale-supercomputer-takes-top-spot-tackles-covid-19

TOP 500 (2019.11 기준)

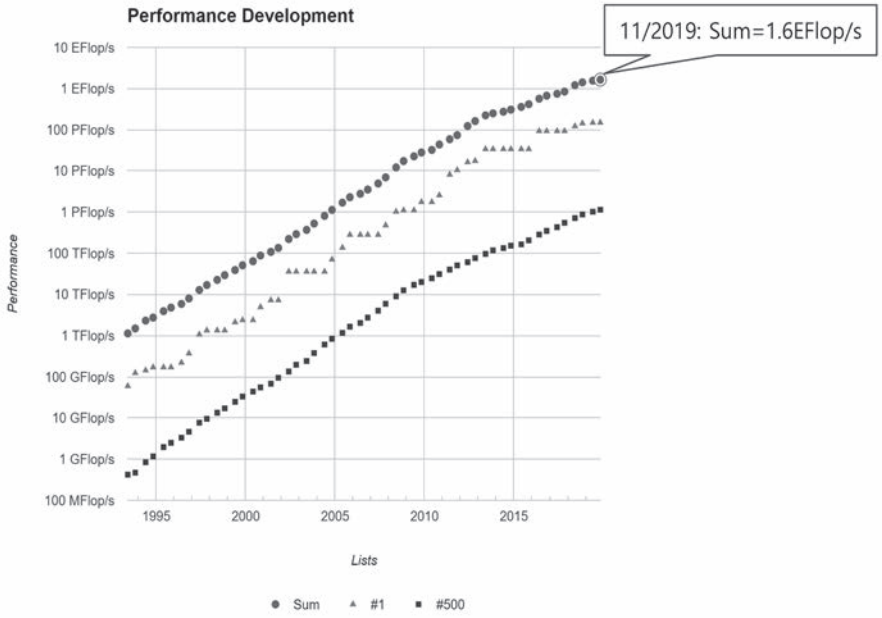
Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	DOE/SC/Oak Ridge National Laboratory United States	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM	2,414,592	148,600.0	200,794.9	10,096
2	DOE/NNSA/LLNL United States	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM / NVIDIA / Mellanox	1,572,480	94,640.0	125,712.0	7,438
3	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
4	National Super Computer Center in Guangzhou China	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 NUDT	4,981,760	61,444.5	100,678.7	18,482
5	Texas Advanced Computing Center/Univ. of Texas United States	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR Dell EMC	448,448	23,516.4	38,745.9	
6	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 Cray/HPE	387,872	21,230.0	27,154.3	2,384
7	DOE/NNSA/LANL/SNL United States	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect Cray/HPE	979,072	20,158.7	41,461.2	7,578
8	National Institute of Advanced Industrial Science and Technology (AIST) Japan	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 M4, Xeon Gold 6148 20C 2.40GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR Fujitsu	391,680	19,880.0	32,576.6	1,649
9	Leibniz Rechenzentrum Germany	**SuperMUC-NG** - ThinkSystem SD650, Xeon Platinum 8174 24C 3.1GHz, Intel Omni-Path Lenovo	305,856	19,476.6	26,873.9	
10	DOE/NNSA/LLNL United States	**Lassen** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, Dual-rail Mellanox EDR Infiniband, NVIDIA Tesla V100 IBM / NVIDIA / Mellanox	288,288	18,200.0	23,047.2	
11	Total Exploration Production France	**PANGEA III** - IBM Power System AC922, IBM POWER9 18C 3.45GHz, Dual-rail Mellanox EDR Infiniband, NVIDIA Volta GV100 IBM	291,024	17,860.0	25,025.8	1,367
12	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
13	DOE/SC/LBNL/NERSC United States	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect Cray/HPE	622,336	14,014.7	27,880.7	3,939
14	Korea Institute of Science and Technology Information Korea, South	**Nurion** - Cray CS500, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path Cray/HPE	570,020	13,929.3	25,705.9	
15	Joint Center for Advanced High Performance Computing Japan	**Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path Fujitsu	556,104	13,554.6	24,913.5	2,719

TOP 500 (2020.06 기준)

1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, **Fujitsu** RIKEN Center for Computational Science **Japan**	7,299,072	415,530.0	513,854.7	28,335
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, **IBM** DOE/SC/Oak Ridge National Laboratory **United States**	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, **IBM / NVIDIA / Mellanox** DOE/NNSA/LLNL **United States**	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, **NRCPC** National Supercomputing Center in Wuxi **China**	10,649,600	93,014.6	125,435.9	15,371
5	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, **NUDT** National Super Computer Center in Guangzhou **China**	4,981,760	61,444.5	100,678.7	18,482
6	**HPC5** - PowerEdge C4140, Xeon Gold 6252 24C 2.1GHz, NVIDIA Tesla V100, Mellanox HDR Infiniband, **Dell EMC** Eni S.p.A. **Italy**	669,760	35,450.0	51,720.8	2,252
7	**Selene** - DGX A100 SuperPOD, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, **Nvidia** NVIDIA Corporation **United States**	272,800	27,580.0	34,568.6	1,344
8	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR, **Dell EMC** Texas Advanced Computing Center/Univ. of Texas **United States**	448,448	23,516.4	38,745.9	
9	**Marconi-100** - IBM Power System AC922, IBM POWER9 16C 3GHz, Nvidia Volta V100, Dual-rail Mellanox EDR Infiniband, **IBM** CINECA **Italy**	347,776	21,640.0	29,354.0	1,476
10	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100, **Cray/HPE** Swiss National Supercomputing Centre (CSCS) **Switzerland**	387,872	21,230.0	27,154.3	2,384
15	**PANGEA III** - IBM Power System AC922, IBM POWER9 18C 3.45GHz, Dual-rail Mellanox EDR Infiniband, NVIDIA Volta GV100, **IBM** Total Exploration Production **France**	291,024	17,860.0	25,025.8	1,367
16	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , **Cray/HPE** DOE/SC/LBNL/NERSC **United States**	622,336	14,014.7	27,880.7	3,939
17	**Nurion** - Cray CS500, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path, **Cray/HPE** Korea Institute of Science and Technology Information **South Korea**	570,020	13,929.3	25,705.9	
18	**Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path, **Fujitsu** Joint Center for Advanced High Performance Computing **Japan**	556,104	13,554.6	24,913.5	2,719
19	**HPC4** - Proliant DL380 Gen10, Xeon Platinum 8160 24C 2.1GHz, Mellanox InfiniBand EDR, NVIDIA Tesla P100, **HPE** Eni S.p.A. **Italy**	253,600	12,210.0	18,621.1	1,320
20	**Tera-1000-2** - Bull Sequana X1000, Intel Xeon Phi 7250 68C 1.4GHz, Bull BXI 1.2, **Atos** Commissariat a l'Energie Atomique (CEA) **France**	561,408	11,965.5	23,396.4	3,178

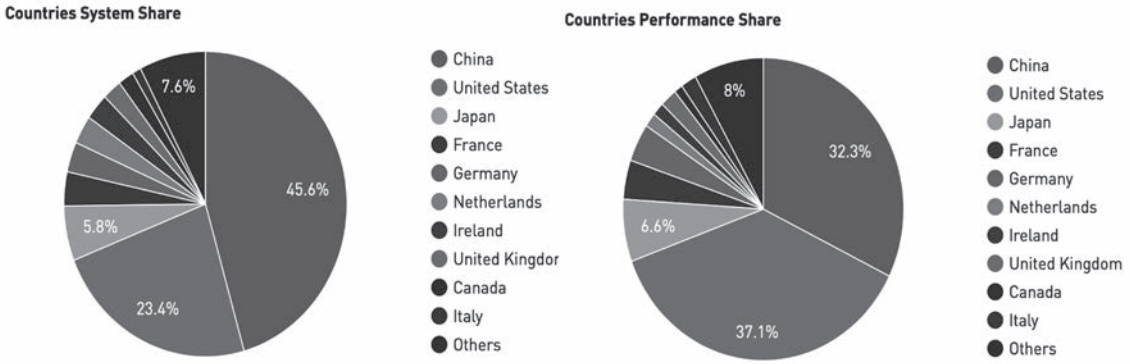
TOP 500

➢ Performance Development (2019년 11월 기준)



TOP500

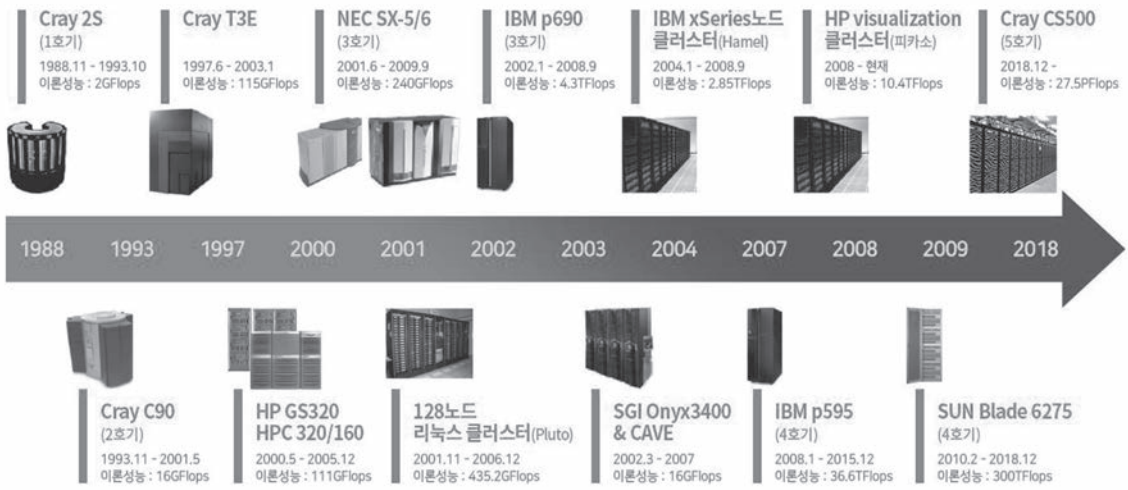
➢ Countries System Share(2019년 11월 기준)



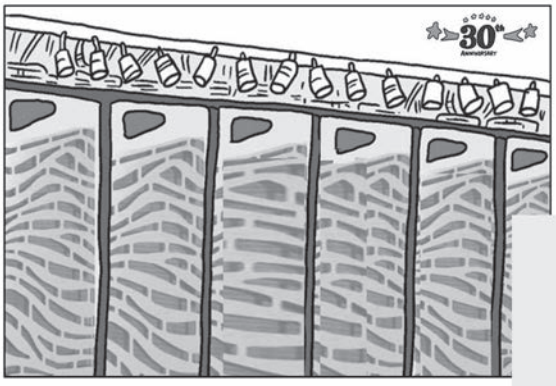
슈퍼컴퓨터 5호기(누리온) 소개

1. **KISTI Supercomputer** 역사
2. **Nurion System** 구축
   - Nurion 시스템 구성
   - Compute Node(Cray 3112-AA000T)
   - CPU-only Node(Cray 3111-BA000T)
   - Performance

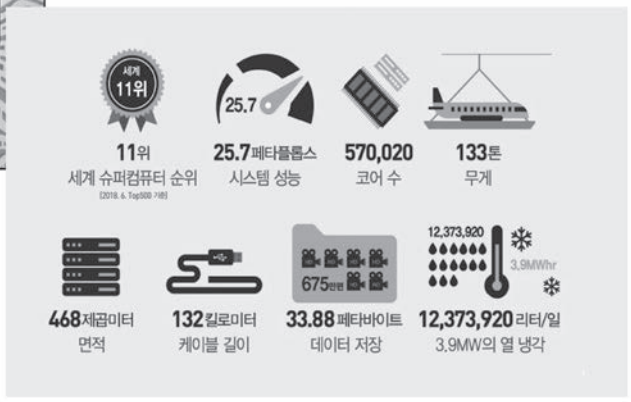
History of KISTI Supercomputer



5th Supercomputer Nurion



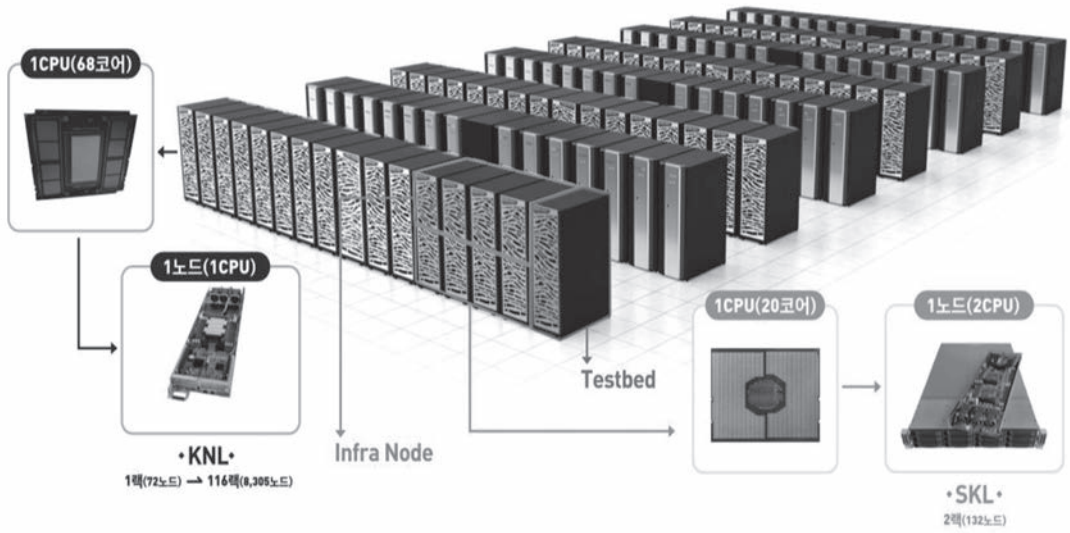
『2018년 12월 3일 공식 서비스 오픈』



Supercomputer Nurion Configuratioin

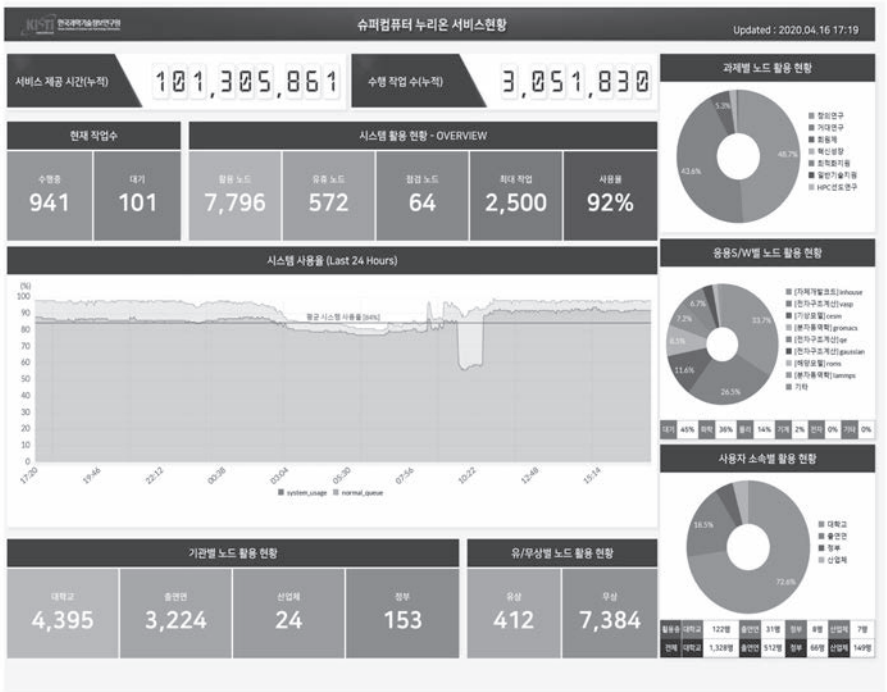
KISTI 과학데이터스쿨

- 2018년 10월(Nurion)
  - 누리(세상,세계, 함께 누리다)+온(전부, 모두의)
  - 온 국민이 다 함께 누리는 국가 슈퍼 컴퓨터



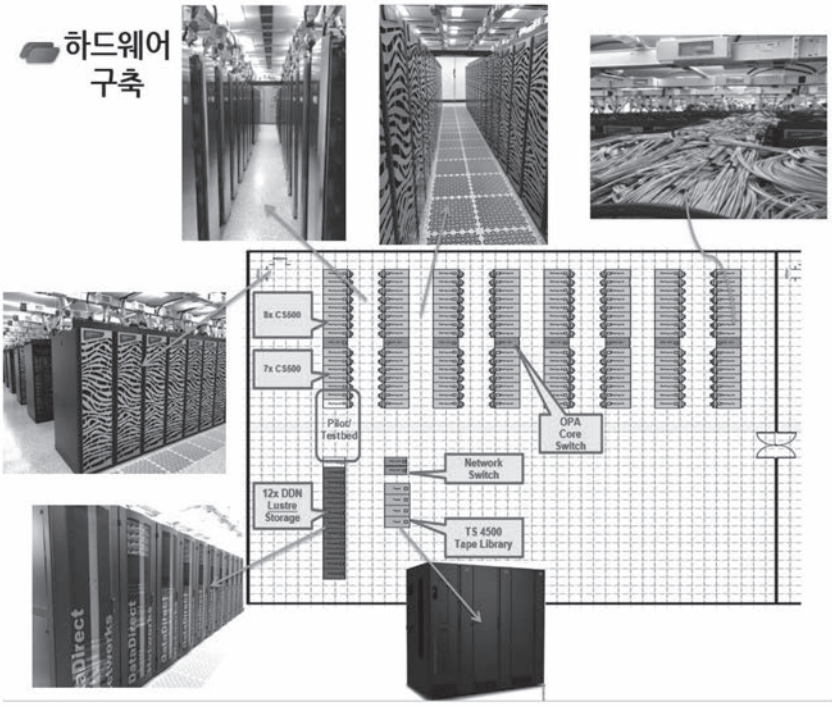
슈퍼컴 누리온 서비스 현황

KISTI 과학데이터스쿨



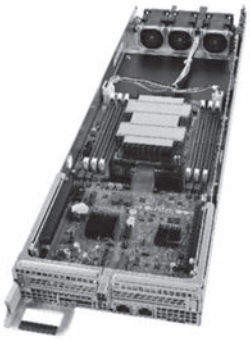
Nurion System 구축

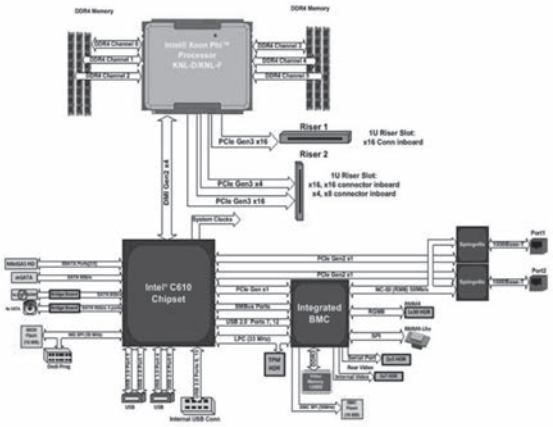
하드웨어 구축

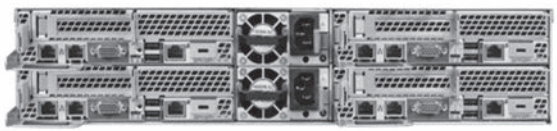


Compute Node(KNL)

- Cray 3112-AA000T
  - 1 x Intel Xeon Phi KNL 7250 processor(68 cores per processor)
  - 96GB(6*16GB) DDR-2400 RAM
  - 1x Single-port 100Gbps OPA HFI card
  - 1x On-board GigE(RJ45) port

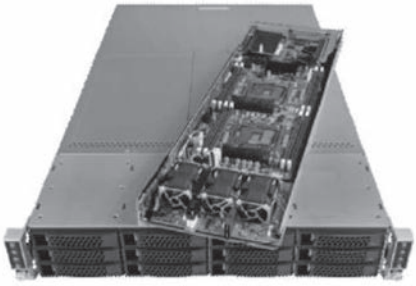


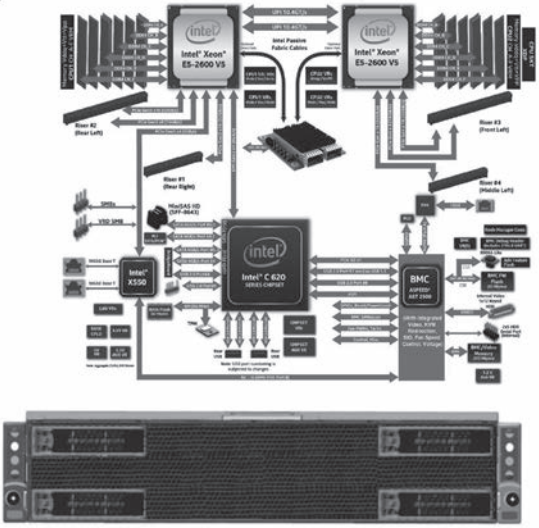




Compute Node(SKL)

- Cray 3111-BA000T
  - 2 x Intel Xeon SKL 6148 processors
  - 192GB (12x 16GB) DDR4-2666 RAM
  - 1x Single-port 100Gbps OPA HFI card
  - 1x On-board GigE(RJ45) port





Performance(Flops)

- 노드 당 성능
  - KNL
    - Core : (8*2)*(2)*1.4G = 44.8Gflops
    - Node : 44.8*68=3046.4Gflops
  - SKL
    - Core : (8*2)*(2)*2.4G=76.8Gflops
    - Node : 40*76.8Gflops/core = 3.072Tflops

- 누리온
  - KNL
    - 8305 nodes : 3.0464Tflps*8305=25.3Pflops
  - SKL
    - 132 nodes : 3.072Tflps*132=405.5Tflops=0.4Pflops
  - KNL+SKL : (25.3+0.4) Pflops = 25.7Pflops

- Tachyon2 : 300.032Tflops

SW 리스트

구분	항목
Cray 의존 라이브러리	cdt/17.10 cray-impi/1.1.4(default) mvapich2_cce/2.2rc1.0.3_noslurm(default) perftools-lite/6.5.2(default) cray-ccdb/3.0.3(default) cray-lgdb/3.0.7(default) mvapich2_gnu/2.2rc1.0.3_noslurm PrgEnv-cray/1.0.2(default) ...
컴파일러	cce/8.6.3(default) gcc/6.1.0 gcc/7.2.0 intel/17.0.5(default) intel/18.0.1 intel/18.0.3
MPI 라이브러리	ime/mvapich-verbs/2.2.ddn1.4 impi/17.0.5(default) impi/18.0.3 openmpi/3.1.0 ime/openmpi/1.10.ddn1.0 impi/18.0.1 mvapich2/2.3
MPI 의존 라이브러리	fftw_mpi/2.1.5 fftw_mpi/3.3.7 hdf5-parallel/1.10.2 netcdf-hdf5-parallel/4.6.1 parallel-netcdf/1.10.0 pio/2.3.1
Libraries	hdf4/4.2.13 hdf5/1.10.2 lapack/3.7.0 ncl/6.5.0 ncview/2.1.7 netcdf/4.6.1
Commercial applications	cfx/v145 cfx/v181 fluent/v145 fluent/v181 gaussian/g16.a03 lsdyna/mpp cfx/v170 cfx/v191 fluent/v170 fluent/v191 gaussian/g16.a03.linda lsdyna/smp
applications	advisor/17.0.5 forge/18.1.2 ImageMagick/7.0.8-20 python/3.7 R/3.5.0 singularity/2.5.1 vtune/17.0.5 advisor/18.0.1 grads/2.2.0 lammps/8Mar18 qe/6.1 siesta/4.0.2 singularity/2.5.2 vtune/18.0.1 advisor/18.0.3 gromacs/2016.4 namd/2.12 qt/4.8.7 siesta/4.1-b3 singularity/3.0.1 vtune/18.0.3 cmake/3.12.3 gromacs/5.0.6 python/2.7.15 qt/5.9.6 singularity/2.4.2 tensorflow/1.12.0

Basic Environment(1)

1. 시스템 접속
2. **Linux** 기초
   - 기본 명령어
   - VI Editor
3. **Environment Module**
   - Module 명령어
     - avail
     - add
     - rm
     - list
     - purge
   - 권장 컴파일러 옵션

시스템 접속

➢ 노드 구성

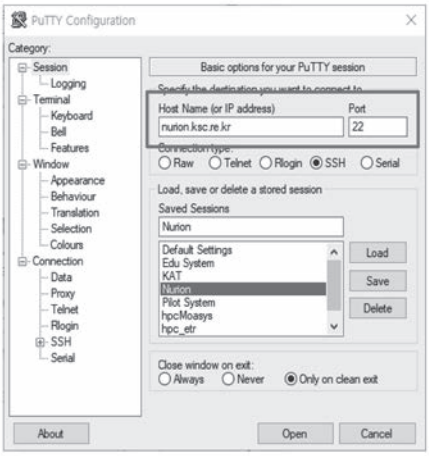
		호스트 명	CPU Limit	비고
로그인 노드		nurion.ksc.re.kr	20분	▪ ssh/scp/sftp 접속 가능 ▪ 컴파일 및 batch 작업제출용 ▪ ftp 접속 불가
Datamover 노드		nurion-dm .ksc.re.kr	-	▪ ssh/scp/sftp 접속 가능 ▪ ftp 접속 가능 ▪ 컴파일 및 작업 제출 불가
계산 노드	KNL	node[0001-8305]	-	▪ PBS 스케줄러를 통해 작업 실행 가능 ▪ 일반사용자 직접 접근 불가
	CPU-Only	cpu[0001-0132]	-	

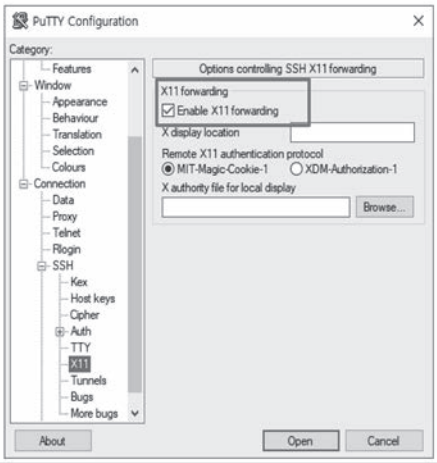
시스템 접속

➢ Xming
  – X 환경 실행을 위해 필요

➢ Putty 사용
  – Host Name : nurion.ksc.re.kr(port : 22)

※ Xming 실행 필요





시스템 접속

- 접속 ID & otp
  - sedu##(01~30)
  - OTP : 0135
  - Passwd : qwe123

```
Last login: Mon Jan  7 10:00:35 2019 from xxx.xxx.xxx.xx
================ KISTI 5th NURION System
====================
  * Compute Nodes(node[0001-8305],cpu[0001-0132)
  - KNL(XeonPhi 7250 1.40GHz 68C) / 16GB(MCDRAM),96GB(DDR4)
  - CPU-only(XeonSKL6148 2.40GHz 20C x2) / 192GB(DDR4)

 * Software
  - OS: CentOS 7.4(3.10.0-693.21.1.el7.x86_64)
  - System S/W: BCM v8.1, PBS v14.2, Lustre v2.10

 * Current Configurations
  - All KNL Cluster modes - Quadrant
  - Memory modes
   : Cache-node[0001-7980,8281-8300]/Flat-node[7981-8280]
   : PBS job sharing mode-Exclusive(running 1 job per node)
     (Except just the commercial queue)
   ...
* Policy on User Job
     ....
   (Use the # showq & # pbs_status commands for more queue info.)
```


시스템 접속

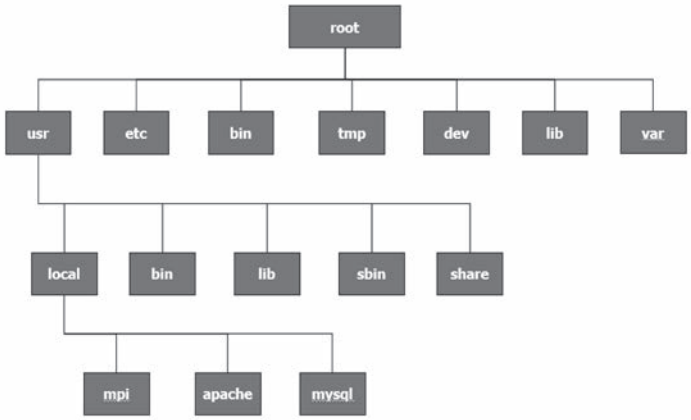
- Policy on User Job

Queue	Wall-Clock Limit	Max Running jobs	Max Active Jobs (running+waiting)
exclusive	unlimited	100	100
normal	48h	30	50
burst_buffer	48h	10	20
long	120h	10	20
flat	48h	10	20
debug	48h	2	2
commercial	48h	3	10
norm_skl	48h	7	14

  - KISTI 큐 정책에 의해 변경될 수 있음

Linux 기초

- File Hierarchy



- 경로
  - 절대 경로 : /home/userid/MPI/examples
  - 상대 경로 : ../../MPI/example

Linux 기초

- 명령어 구조
  - (command) + (options) + (arguments)
  - ls
  - ls -a
  - ls -a /home
- Manual page
  - 시스템에서 제공하는 도움말(man page)
  - 기본적으로 command 마다 해당 man page를 가짐
    - 다음 페이지를 보기 위해 서는 space bar 또는 'f' 입력
    - 이전 페이지를 보기 위해서는 'b' 입력
    - 마치려면 'q' 입력

```
$ man who
WHO(1)                    User Commands
      WHO(1)

NAME
    who - show who is logged on

SYNOPSIS
    who [OPTION]... [ FILE | ARG1 ARG2 ]

DESCRIPTION
    -a, --all
        same as -b -d --login -p -r -t -T -u

    -b, --boot
        time of last system boot

    -d, --dead
        print dead processes
```

기본 명령어

- ls
  - 디렉터리 내의 파일 목록을 위한 명령

- 자주 사용되는 명령어

명령어	내용
cd	디렉터리 이동 명령
pwd	현재 디렉터리 위치를 보여줌
mkdir	새로운 디렉터리를 만들 때 사용
cp	파일 복사 명령, 속성을 유지할 경우 '-a' 옵션 사용
rm	파일이나 디렉터리 삭제
mv	파일과 디렉터리의 이름을 변경하거나 경로를 옮길 때 사용
cat	간단한 텍스트 파일 내용확인
echo	텍스트를 화면 상에 출력
diff	2개의 텍스트 파일 내용을 비교할 때 사용, 바이너리 파일인 경우 같은지 여부만 알려줌
file	파일의 타입(ASCII, Binary)를 알아볼 때 사용

기본 명령어

- tar 명령어
  - 단순하게 파일을 압축하는 용도가 아닌 파일이나 디렉터리를 묶는 용도
  - gzip, unzip과 같이 압축프로그램과 같이 쓰이는 게 일반적
- 기본적인 옵션
  - -z : gzip으로 압축 또는 압축해제 할 때 사용
  - -f : tar 명령어를 이용할 때 반드시 사용(default)
  - x : tar 파일로 묶여있는 것을 해제할 때 사용(extract)
  - c : tar 파일을 생성할 때 사용(create)

- tar –cvf testfile.tar / tar –xvf testfile.tar
- Tar –ccvf testfile.tar.gz [폴더명] / tar –zxvf testfile.tar.gz

VI Editor

- vim(vi)
  - 가장 기본적인 텍스트 에디터, OS에 기본적으로 포함됨
  - VIsual display editor를 의미
- 파일 개방
  - $ vi file(편집 모드)
  - $ view file(읽기 모드)
- modes
  - 입력 모드
    - 입력모드로 전환 : i (,I, a, A, o, O, R)
    - 입력하는 모든 것이 편집 버퍼에 입력됨
    - 입력 모드에서 빠져 나올 때(명령 행 모드로 변경 시) : “ESC” key
  - 명령 행 모드
    - 입력하는 모든 것이 명령어 해석됨
- 파일 저장/종료 명령 : w, q

Environment Module

- 사용자가 쉘 환경(shell environment)을 관리하도록 도와주는 도구
- ‘module’ 명령
  - 부명령 (subcommand)
    - avail(av)
      - 사용 가능한 모듈파일들(modulefiles)을 보여줌
    - add(load)
      - 쉘 환경으로 모듈파일들을 적재함(load)
    - rm(unload)
      - 쉘 환경에서 적재된 모듈파일들을 제거함
    - li(list)
      - 적재된 모듈파일들을 나열함
    - purge
      - 적재된 모든 모듈파일들을 제거함

Environment Module

- Default modulefiles
  - login을 하면, 기본 모듈파일이 적재됨

```
$ module list
Currently Loaded Modulefiles:
  1) craype-network-opa
```

- module 명령
  - 사용가능 모듈 확인 (avail)

```
$ module avail
-------- /opt/cray/craype/default/modulefiles ----------------------
craype-mic-knl     craype-network-opa craype-x86-skylake
---------------- /opt/cray/modulefiles ---------------------------
cdt/17.10                                cray-impi/1.1.4(default)
…
perftools-base/6.5.2(default)

--------- /apps/Modules/modulefiles/compilers ---------------------
cce/8.6.3(default)    gcc/6.1.0             gcc/7.2.0
intel/17.0.5(default) intel/18.0.1          intel/18.0.3
…
```


Environment Module

- 모듈 명령
  - 모듈 정보 출력

```
$ module help impi/17.0.5

----------- Module Specific Help for 'impi/17.0.5' ----------------

        This module is for use of impi/17.0.5
        use example:
            $ module load intel/17.0.5 impi/17.0.5
```

  - 모듈 적재

```
$ module load craype-mic-knl
$ module load intel/17.0.5
(or
$ module add craype-mic-knl intel/17.0.5 )
```

Environment Module

- Default modulefiles in Nurion
  - 적재된 모듈 파일 확인(list subcommand)

```
$ module list
Currently Loaded Modulefiles:
  1) craype-network-opa   2) craype-mic-knl       3) intel/17.0.5
```

  - 적재된 모듈 삭제/ 모듈 추가(rm / add subcommand)

```
$ module rm craype-mic-knl
$ module add craype-x86-skylake
$ module list
Currently Loaded Modulefiles:
  1) craype-network-opa   2) intel/17.0.5     3) craype-x86-skylake
```

  - 적재된 모든 모듈 삭제

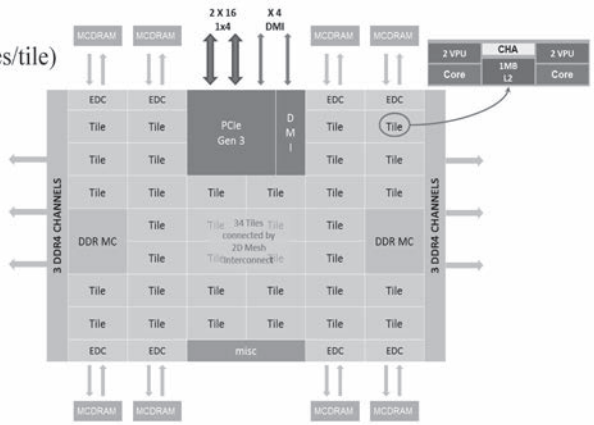
```
$ module list
Currently Loaded Modulefiles:
  1) craype-network-opa   2) intel/17.0.5     3) craype-x86-skylake
$ module purge
$ module li
No Modulefiles Currently Loaded.
```

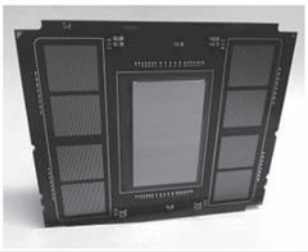

Architecture Overview

1. KNL Architecture
2. Vector Registers
3. Performance(Flops)
4. Instruction Set Architecture
5. Cluster Modes
6. Memory Modes
7. numactl

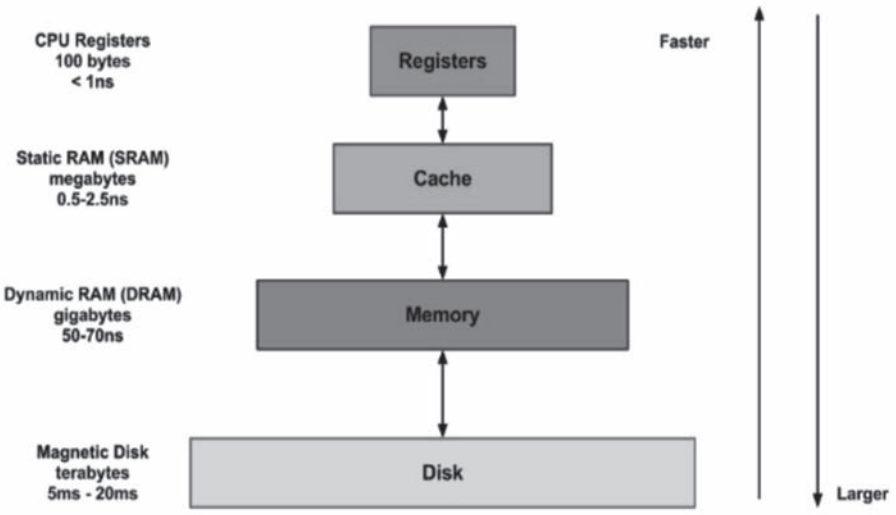
KNL Architecture

- Intel KNL (7250-1.4GHz, 68 cores)
  - Based on Intel Atom architecture (2cores/tile)
  - 1MB shared L2 cache /tile
  - 2D mesh interconnect
  - CHA: DTD to keep L2s coherent. MESIF protocol
- 2 DDR memory controller
- 96GB DDR4 (6 CH)
  - Up to 90 GB/s
- 16 GB MCDRAM (8CH)
  - Up to 450 GB/s



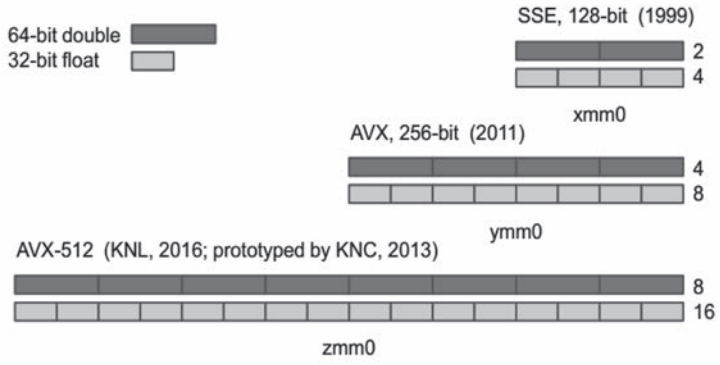


(참조) Register, Cache, Memory, Disk



Vector Registers

- KNL
  - 512-bit register
    - 512bit*(1byte/8bit)=64byte
    - Double Precision : 64byte*(1DP/8byte)=8DP



Performance(Flops)

- KNL
  - Core : (8*2)*(2)*1.4G = 44.8Gflops
  - Node : 44.8*68=3046.4Gflops
- Nurion
  - KNL
    - 8305 nodes : 3.0464Tflps*8305=25.3Pflops
  - CPU-only
    - Node : 2cpu*1.536Tflops/cpu = 3.072flops
    - 132 nodes : 405.5Tflops=0.4Pflops
  - KNL+CPU-only : (25.3+0.4)PFlops = 25.7Pflops
- Tachyon2 : 300.032Tflops

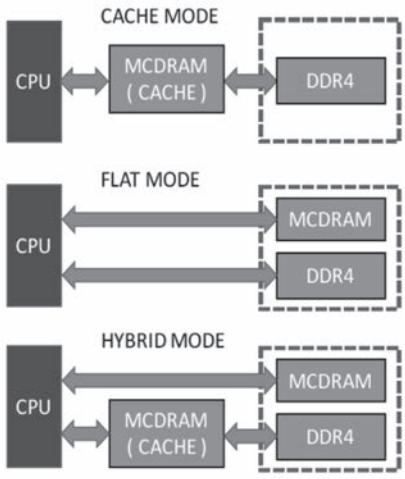
Number of cores	68
SIMD width (doubles)	8 * 2
Multiply/add in 1 cycle	2
Clock speed(Gcycle/s)	1.4
DP Gflop/s/core	44.8
DP Gflops/s/processor	3046

Instruction Set Architecture(ISA)

- 인텔 AVX-512 명령어 세트(Instruction Set Architecture: ISA)는 4 개의 그룹으로 구성됨
  - AVX-512 Foundation 명령어 : AVX-512F
  - AVX-512 Conflict Detection 명령어: AVX-512CD
  - AVX-512 Exponential and Reciprocal 명령어: AVX-512 ER
  - AVX-512 Prefetch 명령어: AVX-512PF
  - AVX-512BW, AVX-512DQ, AVX-512VL(for Xeon processor)
- 인텔 컴파일러 옵션
  - -xCOMMON-AVX512: AVX-512F + AVX-512CD
  - -xMIC-AVX512: AVX-512F + AVX-512CD+AVX-512ER + AVX-512PF
  - -xCORE-AVX512: AVX-512F + AVX-512CD + AVX-512BW + AVX-512DQ + AVX-512VL

Memory Modes

- 캐시 모드(Cache mode)
  - default
  - MCDRAM은 L3 캐시처럼 동작
- 플랫 모드(Flat mode)
  - MCDRAM과 DDR4는 RAM 처럼 동작
    - numactl 명령
    - memkind/autohbw library
  - 다른 NUMA nodes
- 혼합 모드(Hybrid mode)
  - MCDRAM 은 다음과 같이 동작
    - L3 cache
    - DDR



numactl

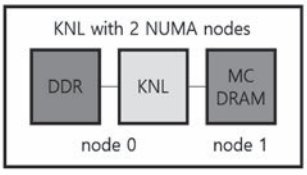
- numactl -H(numactl --hardware)
  - quadrant + cache : 하나의 NUMA (DDR)

```
$ numactl -H (numactl --hardware)
available: 1 nodes (0)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
65 66 67
node 0 size : 98207 MB
node 0 free : 80432 MB
node distances:
node  0
  0: 10
```


numactl

- numactl -H
  - quadrant + flat : 두 개의 NUMA (MCDRAM and DDR)

```
$ numactl -H (numactl --hardware)
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
65 66 67
node 0 size : 98207 MB
node 0 free : 54711 MB
node 1 cpus:
node 1 size : 16384 MB
node 1 free : 15950 MB
node distances:
node  0    1
  0: 10   31
  1: 31   10
```



numactl

- MCDRAM에 메모리 할당
  - "numactl -m (NUMA nodes) + application name"
    - numactl -m 1 a.out(quadrant + flat) : (node 1은 MCDRAM node)
    - 요청된 것 보다 MCDRAM의 메모리가 부족하면 프로그램은 종료됨(Killed)
  - "-p(preferred)" 옵션 사용을 권장
    - numactl -p 1 a.out(quadrant + flat)
    - 요청된 메모리를 MCDRAM에 우선적으로 할당
    - MCDRAM의 용량이 부족하면 DDR 메모리에 할당
    - MCDRAM의 메모리가 부족하여 application이 종료되는 경우는 없음

Basic Environment(2)

1. 작업 디렉터리 및 쿼터 정책
2. 프로그래밍 도구 설치 현황
3. 상용 소프트웨어 설치 정보
4. 프로그램 컴파일
   - 컴파일러 별 주요 옵션
   - 순차 프로그램 컴파일
   - OpenMP 프로그램 컴파일
   - MPI 프로그램 컴파일

Basic Environment

➢ 작업 디렉터리 및 쿼터 정책

구분	디렉터리 경로	용량 제한	파일 수 제한	파일 삭제 정책	파일 시스템	백업 유무
홈 디렉터리	/home01	64GB	100K	N/A	Lustre	O
스크래치 디렉터리	/scratch	100TB	1M	15일 동안 접근하지 않은 파일은 자동 삭제		X

– 현재 사용량 확인

```
$ lfs quota /home01
Disk quotas for usr sedu01 (uid 1000163):
 Filesystem  kbytes   quota   limit      grace   files   quota   limit   grace
    /home01     104  67108864 67108864       -      26  100000  100000       -
$ lfs quota /scratch
Disk quotas for usr sedu01 (uid 1000163):
 Filesystem  kbytes   quota             limit   grace   files    quota   limit   grace
   /scratch       4  107374182400 107374182400       -       1  1000000 1000000       -
Disk quotas for grp in0163 (gid 1000163):
```

– 홈 디렉터리는 용량 및 I/O 성능이 제한되어 있기 때문에, 모든 계산 작업은 스크래치 디렉터리에서 이루어져야 함.

Basic Environment

➢ 프로그래밍 도구 설치 현황

– 컴파일러 및 라이브러리 모듈

구분	항목	
아키텍처 구분 모듈	▪ craype-mic-knl ▪ craype-x86-skylake	▪ craype-network-opa
Cray 모듈	▪ perftools/6.5.2 ▪ perftools-base/6.5.2 …	▪ PrgEnv-cray/1.0.2 …
컴파일러	▪ cce/8.6.3 ▪ gcc/7.2.0 ▪ gcc/6.1.0	▪ intel/17.0.5(default) ▪ intel/18.0.1 ▪ intel/18.0.3
컴파일러 의존 라이브러리	▪ hdf4/4.2.13 ▪ hdf5/1.10.2 ▪ lapack/3.7.0	▪ ncl/6.5.0 ▪ ncview/2.1.7 ▪ netcdf/4.6.1
MPI 라이브러리	▪ impi/17.0.5(default) ▪ impi/18.0.1 ▪ impi/18.0.3	▪ openmpi/3.1.0 ▪ mvapich2/2.3

Basic Environment

➢ 프로그래밍 도구 설치 현황

– 컴파일러 및 라이브러리 모듈

구분	항목	
MPI 의존 라이브러리	▪ fftw_mpi/2.1.5 ▪ fftw_mpi/3.3.7 ▪ hdf5-parallel/1.10.2	▪ netcdf-hdf5-parallel/4.6.1 ▪ parallel-netcdf/1.10.0 ▪ pio/2.3.1
Intel 패키지	▪ advisor/17.0.5 ▪ advisor/18.0.1 ▪ advisor/18.0.3	▪ vtune/17.0.5 ▪ vtune/18.0.1 ▪ vtune/18.0.3
응용 소프트웨어	▪ forge/18.1.2 ▪ ImageMagick/7.0.8-20 ▪ python/2.7.15 ▪ python/3.7 ▪ gromacs/2016.4 ▪ namd/2.12 ▪ qt/4.8.7 ▪ qt/5.9.6	▪ R/3.5.0 ▪ grads/2.2.0 ▪ lammps/8Mar18 ▪ qe/6.1 ▪ siesta/4.0.2 ▪ siesta/4.1-b3 ▪ cmake/3.12.3 ▪ gromacs/5.0.6
가상화 모듈	▪ singularity/2.5.1 ▪ singularity/2.5.2 ▪ singularity/3.0.1	▪ singularity/2.4.2 ▪ tensorflow/1.12.0

Basic Environment

➢ 상용 소프트웨어 설치 정보

분야	소프트웨어	버전	라이선스	디렉터리 위치
구조역학	Abaqus	6.14-6 2016 2017 2018	151 토큰	/apps/commercial/abaqus/
	MSC ONE (Nastran)	20182	60 토큰	/apps/commercial/MSC/Nastran
	LS-DYNA	R10.1.0 R9.2.0	최대 128 코어 사용 가능	/apps/commercial/LSDYNA
열유체 역학	ANSYS CFX	V145 V170 V181 V191	17 Solvers (HPC 640)	/apps/commercial/ANSYS/
	ANSYS Fluent			
화학/생명	Gaussian	G16-a03	작업 수 제한 없음 단일 노드 내 CPU수 제한 없음	/apps/commercial/G16/g16
		G16-a03.linda		

Basic Environment

➢ 프로그램 컴파일

– 누리온 시스템

- Intel 컴파일러, GNU 컴파일러, Cray 컴파일러 제공
- Intel MPI(IMPI), Mvapich2, OpenMPI 제공

– 기본 필요 모듈

- craype-network-opa
- craype-mic-knl(KNL), craype-x86-skylake(SKL)

Basic Environment

➢ 프로그램 컴파일

– 순차 프로그램 컴파일

프로그램	벤더	컴파일러	소스 확장자	사용 모듈
C / C++	Intel	icc / icpc	.C, .cc, .cpp, .cxx, .c++	intel/17.0.5 | intel/18.0.1 | intel/18.0.3
	GNU	gcc / g++		gcc/6.1.0 | gcc/7.2.0
	Cray	cc / CC		PrgEnv-cray/1.0.2 & cce/8.6.3
F77/F90	Intel	ifort	.f, .for, .ftn, .f90, .fpp, .F, .FOR, .FTN, .FPP, .F90	intel/17.0.5 | intel/18.0.1 | intel/18.0.3
	GNU	gfortran		gcc/6.1.0 | gcc/7.2.0
	Cray	ftn		PrgEnv-cray/1.0.2 & cce/8.6.3

Basic Environment

➢ 프로그램 컴파일

– 순차 프로그램 컴파일

• Intel 컴파일러 주요 옵션

컴파일러 옵션	설명
-O[1|2|3]	오브젝트 최적화, 숫자는 최적화 레벨
-qopt-report=[0|1|2|3|4|5]	벡터 진단 정보의 양을 조절
-xCORE-AVX512 -xMIC-AVX512	512bit 레지스터를 가진 CPU 지원 512bit 레지스터를 가진 MIC 지원
-qopenmp	OpenMP 기반의 multi-thread 코드 사용
-fPIC, -fpic	PIC(Position Independent Code)가 생성되도록 컴파일 (공유 라이브러리 내 심볼이 어느 주소에 로드되더라도 동작)

• 권장 옵션

– -O3 -fPIC -xCORE-AVX512 (Skylake)

– -O3 -fPIC -xMIC-AVX512 (KnightsLanding)

– -O3 -fPIC -xCOMMON-AVX512(Skylake & KnightsLanding)

```
$ icc|ifort -o test.exe -O3 -fPIC -xMIC-AVX512 test.[c|cc|f90]
```


Basic Environment

➢ 프로그램 컴파일

– 순차 프로그램 컴파일

• GNU 컴파일러 주요 옵션

컴파일러 옵션	설명
-O[1|2|3]	오브젝트 최적화, 숫자는 최적화 레벨
-march=skylake-avx512 -march=knl	512bits 레지스터를 가진 CPU 지원 512bits 레지스터를 가진 MIC 지원
-Ofast	-O3 -ffast-math 매크로
-fopenmp	OpenMP 기반의 multi-thread 코드 사용
-fPIC	PIC(Position Independent Code)가 생성되도록 컴파일

• 권장 옵션

– -O3 -fPIC -march=skylake-avx512 (Skylake)

– -O3 -fPIC -march=knl (KnightsLanding)

– -O3 -fPIC -mpku (Skylake & KnightsLanding)

```
$ gcc|gfortran -o test.exe -O3 -fPIC -march=knl test.[c|cc|f90]
```

Basic Environment

➢ 프로그램 컴파일

– 순차 프로그램 컴파일

• Cray 컴파일러 주요 옵션

컴파일러 옵션	설명
-O[1|2|3]	오브젝트 최적화, 숫자는 최적화 레벨
-hcpu=mic-knl	512bits 레지스터를 가진 MIC 지원 사용하지 않으면 Skylake 지원(default)
-homp(default)	OpenMP 기반의 multi-thread 코드 사용
-h pic	2GB 이상의 static memory가 필요한 경우 사용(-dynamic과 함께 사용)
-dynamic	공유 라이브러리를 링크

• 권장 옵션

– Default 옵션 사용을 권장

```
$ cc|ftn -o test.exe -hcpu=mic-knl test.[c|cc|f90]
```


Basic Environment

➢ 프로그램 컴파일

– 병렬 프로그램 컴파일

• OpenMP 컴파일

– OpenMP는 컴파일러 지시어만으로 멀티 스레드를 활용할 수 있도록 개발된 기법임

– 컴파일러 옵션을 추가하여 병렬 컴파일을 할 수 있음

» Intel compiler : -qopenmp

» GNU compiler : -fopenmp

» Cray compiler : -homp

```
$ icc|ifort -o test.exe -qopenmp -O3 -fPIC -xMIC-AVX512 test.[c|cc|f90]
$ gcc|gfortran -o test.exe -fopenmp -O3 -fPIC -march=knl test.[c|cc|f90]
$ cc|ftn -o test.exe -homp -hcpu=mic-knl test.[c|cc|f90]
```

Basic Environment

- 프로그램 컴파일
  - 병렬 프로그램 컴파일
    - MPI 컴파일
      - MPI 명령을 이용하여 컴파일
      - MPI 명령은 일종의 wrapper로써 지정된 컴파일러가 소스를 컴파일 함

구분	Intel	GNU	Cray
Fortran	ifort	gfortran	ftn
Fortran + MPI	mpiifort	mpif90	ftn
C	icc	gcc	cc
C + MPI	mpiicc	mpicc	cc
C++	icpc	g++	CC
C++ + MPI	mpiicpc	mpicxx	CC

```
$ mpiicc|mpiifort -o test.exe -O3 -fPIC -xMIC-AVX512 test.[c|90]
$ mpicc|mpif90 -o test.exe -O3 -fPIC -march=knl test.[c|f90]
$ cc|ftn -o test.exe -hcpu=mic-knl test.[c|f90]
```


Job Scheduler

1. **Nurion Queue**
2. **PBS command**
3. **Job script** 작성
4. **Job script examples**
   - Serial code(KNL, SKL)
   - OpenMP code(KNL, SKL)
   - MPI code(KNL, SKL)
   - Hybrid code(KNL, SKL)
5. **Interactive** 작업 제출

Scheduler 명령어 모음

➢ KISTI Job Scheduler 명령

– 누리온은 PBS, 뉴론은 Slurm 작업 스케줄러 사용

User Commands	PBS (Nurion)	SGE (Tachyon2)	Slurm (Neuron)	LoadLeveler (Sinbaram)
작업 제출	qsub [script_file]	qsub [script_file]	sbatch [script_file]	llsubmit [script_file]
작업 삭제	qdel [job_id]	qdel [job_id]	scancle [job_id]	llcancel [job_id]
작업 조회(job_id)	qstat [job_id]	qstat -u * [-j job_id]	squeue [job_id]	llq -l [job_id]
작업 조회(user)	qstat -u [user_name]	qstat [-u user_name]	squeue -u [user_name]	llq -u [user_name]
Queue 목록	qstat -Q	qconf -sql	squeue	llclass
Node 목록	pbsnodes -aS	qhost	sinfo -N or scontrol show nodes	llstatus -L machine
Cluster 상태	pbsnodes -aSj	qhost -q	sinfo	llstatus -L cluster
GUI	xpbsmon	qmon	sview	xload

Nurion Queue

➢ 큐 정책

Queue	Wall-Clock Limit	Max Running jobs	Max Active Jobs (running+waiting)
exclusive	unlimited	100	100
normal	48h	30	50
burst_buffer	48h	10	20
long	120h	10	20
flat	48h	10	20
debug	48h	2	2
commercial	48h	3	10
norm_skl	48h	7	14

– 큐 정책에 의해 변경가능

Nurion Queue

- 큐 정책
  - 누리온 시스템은 배타적 노드 할당 정책을 기본으로 함
    - 한 노드에 한 사용자의 작업만이 실행될 수 있도록 보장
  - normal 큐
    - 일반 사용자를 위한 큐
  - commercial 큐
    - 상용 SW 수행을 위한 큐
    - 공유 노드 정책이 적용됨
      - 노드의 규모가 크지 않아서 효율적으로 자원을 활용하기 위함임
  - debug 큐
    - 공유 노드 정책이 적용됨
      - 사용한 자원만큼만 과금됨
    - Interactive job 제출이 가능

Nurion Queue

- 큐 조회
  - showq, pbs_status

```
[sedu01@login01 ~]$ showq
                              [PBS QUEUE CONFIGURATION]                                    updated on 2019-01-07
=============================================================================================================================
      Queue     Limits            Nodes   Cores           Modes   Property Charge_rate                Period          Comment
=============================================================================================================================
  exclusive  unlimited  node[0001-2600]  176800  Quadrant/Cache  Exclusive  prescribed  2019.01.01-2019.12.31 Prescribed_users
       khoa  unlimited  node[2601-2710]    7480  Quadrant/Cache  Exclusive  prescribed  2019.01.01-2019.12.31 Prescribed_users
  rokaf_knl  unlimited  node[2711-2775]    4420  Quadrant/Cache  Exclusive  prescribed  2019.01.07-2019.02.12 Prescribed_users
     normal      48hrs  node[2776-7800]  341700  Quadrant/Cache     Normal           1  2019.01.01-2019.12.31      Normal_users
burst_buffer      48hrs  node[6801-7800]   68000  Quadrant/Cache     Normal           1  2019.01.01-2019.12.31      Normal_users
       long     120hrs  node[7801-7980]   12240  Quadrant/Cache     Normal           1  2019.01.01-2019.12.31      Normal_users
       flat      48hrs  node[7981-8280]   20400   Quadrant/Flat     Normal           1  2019.01.01-2019.12.31      Normal_users
      debug      48hrs  node[8281-8300]    1360  Quadrant/Cache     Normal           1  2019.01.01-2019.12.31      Normal_users
  rokaf_skl  unlimited      cpu[0001-0005]   200             N/A  Exclusive  prescribed  2019.01.07-2019.02.12 Prescribed_users
 commercial      48hrs      cpu[0006-0132]  5080             N/A     Normal         1.7  2019.01.01-2019.12.31 Prescribed_users
   norm_skl      48hrs      cpu[0006-0132]  5080             N/A     Normal         1.7  2019.01.01-2019.12.31      Normal_users
=============================================================================================================================
[sedu01@login01 ~]$ pbs_status
      Queue     total      free     alloc      down  run_jobs  que_jobs  largest_Job     Usage
-----------------------------------------------------------------------------------------------
  exclusive      2600        99      2500         1         1         1         2500       96%
       khoa       110        90        19         1         3         0           10       17%
  rokaf_knl        65         3        62         0         2         0           31       95%
     normal      5025      2853      2162        10       178        12          256       43%
burst_buffer     1000       720       280         0         0         0            0       28%
       long       180         0       180         0        43        13           32      100%
       flat       300       300         0         0         0         0            0        0%
      debug        20        20         0         0         0         0            0        0%
  rokaf_skl         5         5         0         0         1         0            1        0%
 commercial       127        11       116         0        14         0            2       91%
   norm_skl       127        11       116         0        37         4           20       91%
-----------------------------------------------------------------------------------------------
      total      8432      3381      5039        12       279        30         2500       59%
```

Nurion Queue

KISTI 과학데이터스쿨

➢ 큐 조회

– qstat

- queue 목록 조회 : -Q
- queue 상세 정보 조회 : -f

```
$ qstat -Q
Queue              Max   Tot Ena Str   Que   Run   Hld   Wat   Trn   Ext Type
---------------- ----- ----- --- --- ----- ----- ----- ----- ----- ----- ----
exclusive            0     1 yes yes     0     1     0     0     0     0 Exec
commercial           0     6 yes yes     0     6     0     0     0     0 Exec
norm_skl             0    56 yes yes     9    46     1     0     0     0 Exec
…
```

```
$ qstat -Qf normal
Queue: normal
    queue_type = Execution
    Priority = 100
    total_jobs = 143
    state_count = Transit:0 Queued:0 Held:8 Waiting:0 Running:135 Exiting:0 Beg
        un:0
    max_queued = [u:PBS_GENERIC=40]
    acl_host_enable = False
    acl_user_enable = False
    resources_max.walltime = 48:00:00
    resources_min.walltime = 00:00:00
    …
```


Nurion Queue

KISTI 과학데이터스쿨

➢ 큐 조회

– 현재 계정으로 사용 가능한 큐 리스트 조회

- ‘pbs_queue_check’

```
[sedu01@login01 job_examples]$ pbs_queue_check
[id:sedu01] belongs to the queue below:

Queue              Max   Tot Ena Str   Que   Run   Hld   Wat   Trn   Ext Type
---------------- ----- ----- --- --- ----- ----- ----- ----- ----- ----- ----
normal               0   185 yes yes     0   174    11     0     0     0 Exec
long                 0    56 yes yes    11    43     2     0     0     0 Exec
flat                 0     0 yes yes     0     0     0     0     0     0 Exec
debug                0     0 yes yes     0     0     0     0     0     0 Exec
commercial           0    14 yes yes     0    14     0     0     0     0 Exec
norm_skl             0    40 yes yes     2    37     1     0     0     0 Exec
```

PBS command : node 조회 및 변경

- pbsnodes
  - '-a' : 등록된 계산 노드 목록 조회
  - '-aSj' : 노드 사용 내역 조회

```
$ pbsnodes -aSj
                                                          mem       ncpus   nmics   ngpus
vnode           state           njobs   run   susp      f/t        f/t     f/t     f/t   jobs
--------------- --------------- ------ ----- ------ ------------ ------- ------- ------- -----
--
node0001        free                0     0      0  110gb/110gb   68/68     0/0     0/0 --
…
node0007        free                1     1      0  110gb/110gb    4/68     0/0     0/0 6615
node0008        free                0     0      0  110gb/110gb   68/68     0/0     0/0 --
node0009        free                0     0      0  110gb/110gb   68/68     0/0     0/0 --
node0010        free                0     0      0  110gb/110gb   68/68     0/0     0/0 --
cpu0004         job-busy            1     1      0  188gb/188gb    0/40     0/0     0/0 6643
cpu0003         job-busy            1     1      0  188gb/188gb    0/40     0/0     0/0 6644
cpu0002         job-busy            1     1      0  188gb/188gb    0/40     0/0     0/0 6628
cpu0001         job-busy            1     1      0  188gb/188gb    0/40     0/0     0/0 6627
(↑ : pilot system에서 출력 예임)
```

Column	Description
mem	기가바이트(GB) 단위의 메모리 양
ncpus	이용 가능한 총 CPU 개수
nmics	이용 가능한 많은 통합 코어들(MIC)의 총 개수 - Intel
ngpus	이용 가능한 총 GPU의 개수
f/t	f=free, t=total

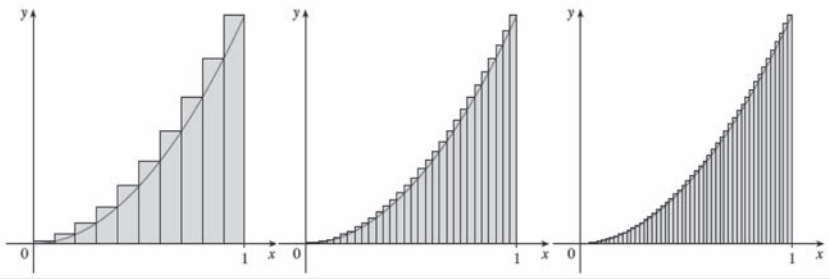
PBS Job Script 사용 예제: (PI 코드)

- 정적분을 이용한 pi 계산

$$\int_0^1 \frac{1}{1+x^2}\,dx = \tan^{-1}(x)\Big|_0^1 = \tan^{-1}(1) - \tan^{-1}(0) = \frac{\pi}{4}$$

- 구분 구적법

$$\int_a^b f(x)dx = \lim_{n\to\infty}\sum_{k=1}^{n} f(x_k)\Delta x, \qquad \left(\Delta x = \frac{b-a}{n}\right)$$



순차 코드 예제(Pi 계산)

➢ 순차 코드(pi.c)

```
#include <stdio.h>
#include <math.h>
#include <sys/time.h>

inline double cpuTimer(){
    struct timeval tp;
    gettimeofday(&tp,NULL);
    return ((double)tp.tv_sec + (double)tp.tv_usec*1e-6);
}
int main(){
    double iStart, ElapsedTime;
    const long num_step = 5000000000;
    long i;
    double sum, step, pi, x;
    step = (1.0/(double)num_step);
    sum = 0.0;
    iStart=cpuTimer();
    printf("--------------------------------------\n");
    for(i=1;i<=num_step;i++){
        x = ((double)i-0.5)*step;
        sum += 4.0/(1.0+x*x);
    }
    pi = step*sum;
    ElapsedTime=cpuTimer() - iStart;
    printf("PI= %.15f (Error = %e)\n",pi, fabs(acos(-1)-pi));
    printf("Elapsed Time = %f, [sec]\n", ElapsedTime);
    printf("--------------------------------------\n");
    return 0;
}
```

● 코드 컴파일

```
$ module add intel/18.0.3
$ icc pi.c -o pi_serial
```

● Job script 작성

```
$ cat serial.sh
#!/bin/bash
#PBS -V
#PBS -N Serial_Job
#PBS -q normal
#PBS -l walltime=00:10:00
### Select 1 nodes with 1 CPU
#PBS -l select=1:ncpus=1
#PBS -A etc

cd $PBS_O_WORKDIR
./pi_serial
```


PBS command : 작업 제출

➢ 작업 제출

– 사용자 작업은 반드시 /scratch 에서만 제출이 가능함

• /home 디렉터리에서 제출 불가능

```
qsub {job_script_name}
```

– ‘depend’ 옵션을 사용하여 의존성 있는 작업 제출 가능

```
qsub -W depend={option}:{JOBID} {job_scropt_name}
```

• afterok : 의존 작업이 성공 시 다음 작업 수행
• afternotok : 의존 작업이 실패 시 다음 작업 수행
• afterany : 의존 작업의 성공 여부에 관계없이 다음 작업 수행

```
$ qsub serial.sh
1820015.pbs
$ qsub -W depend=afterok:1820015.pbs serial.sh
1820017.pbs
$ qstat -u “sedu01"
pbs:
                                                          Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1820015.pbs     sedu01   normal   Serial_Job  47089   1   1    --  00:10 R 00:00
1820017.pbs     sedu01   normal   Serial_Job    --    1   1    --  00:10 H   --
```

PBS command : 작업 제출 및 삭제

- qdel
  - 제출된 작업 삭제

```
qdel {JOBID}
```

```
$ qstat -u "sedu01"

pbs:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1822816.pbs     sedu01   normal   Serial_Job  63673   1   1    --  00:10 R 00:00
1822817.pbs     sedu01   normal   Serial_Job    --    1   1    --  00:10 H   --

$ qdel 1822817.pbs

$ qstat -u "sedu01"

pbs:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1822816.pbs     sedu01  normal   Serial_Job  63673   1   1    --  00:10 R 00:00
```


PBS command : 수행 중 작업 조회

- qstat
  - 실행 및 대기 중인 작업 조회
    - 기본 값은 모든 사용자의 작업 목록 출력
    - 지정 계정 작업 목록 출력 : -u
    - 작업 수행 계산 노드 정보 출력: -n

```
$ qstat
Job id            Name             User             Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
1819461.pbs       G16-Si-b-TD      x1679a02          3756:42: R long
1819463.pbs       G16-Si-c-TD      x1679a02          3715:10: R long
…
1822818.pbs       Serial_Job       sedu01            00:00:00 R normal
$ qstat -u sedu01
pbcm:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1822818.pbs     sedu01  normal   Serial_Job  63895   1   1    --  00:10 R 00:00

$ qstat -n -u sedu01
pbcm:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1822818.pbs     sedu01  normal   Serial_Job  63895   1   1    --  00:10 R 00:00
   node2780/0
```

PBS command : 종료된 작업 조회

- qstat -x
  - 기본 값은 모든 사용자의 작업 출력
    - '-u' : 지정 계정의 종료 작업 목록 출력
    - '-f {JOBID}' : 종료 작업 상세 정보 출력

```
$ qstat -xu sedu01

pbcm:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1822810.pbs     sedu01 norm_skl Serial_Job 425430   1   1    --  00:10 F 00:00
…
1822818.pbs     sedu01   normal   Serial_Job  63895   1   1    --  00:10 F 00:01

$ qstat -xf 1822818.pbs
Job Id: 1822818.pbs
    Job_Name = Serial_Job
    Job_Owner = sedu01@login01
    resources_used.cpupercent = 99
    resources_used.cput = 00:00:57
    resources_used.mem = 3636kb
    resources_used.ncpus = 1
    resources_used.vmem = 250260kb
    resources_used.walltime = 00:01:11
    …
```


Job Script 작성

- "#PBS" 지시자를 사용하여 옵션 지정
- chunk 단위로 host/vnode에 자원 할당
  - '-l select'로 chunk 자원 할당
  - '-l select=<numerical>:<res1>=<value>:<res2>=<value>…'
    - 각 리소스는 colon(:)으로 구분
  - 기본은 '1 chunk == 1 task'
    - '#PBS -l select=128' : 128개의 chunks
    - '#PBS -l select=1:mem=16gb+15:mem=1gb' : 16GB를 사용하는 1개의 chunk와 1GB를 사용하는 15개의 chunk로 작업 수행

```
#!/bin/sh
#PBS -V                         # 작업 제출 노드의 쉘 환경변수를 컴퓨팅 노드에도 적용
#PBS -N hybrid_node             # 작업 이름 지정
#PBS -q workq                   # 작업 queue 지정
#PBS -l walltime=01:00:00       # 작업 walltime 지정
#PBS -M abc@abc.com             # 작업 관련 메일을 수신 할 주소
#PBS -m abe                     # a(작업 실패)/b(작업 시작)/e(작업 종료) 시 메일 발송, n : 메일 보내지 않음
#PBS -l select=2:ncpus=2        # 2 chunk 로 작업 자원 할당 지정
#PBS -A etc                     # application 종류에 따라 값을 다르게 주어야 함

cd $PBS_O_WORKDIR               # PBS는 작업 제출 경로가 WORKDIR로 설정 되지만 기본값으로 $HOME 에서
                                # 작업이 실행됨. 상대 경로 파일을 사용한 경우 PBS_O_WORKDIR 로 변경 필요.
mpirun -machinefile $PBS_NODEFILE ./hostname.x
```

Job Script 작성

➢ 작업 스크립트 주요 키워드

옵션	형식	설명
-V		환경 변수 내보내기
-N	<alphanumeric>	Job 이름 지정
-q	<queue_name>	서버나 큐의 이름 지정
-l	<resource_list>	Job 리소스 요청
-M	<id@domain.xxx>	이 메일 받는 사람 리스트 설정
-m	<string>	이 메일 알람 지정
-W sandbox=	[HOME | PRIVATE]	스테이징 디렉터리와 실행 디렉터리
-X		Interactive job으로부터의 X output

– PBS 배치 작업 수행하는 경우

- STDOUT과 STDERR을 시스템 디렉터리의 output에 저장하였다가 작업 완료 후 사용자 작업 제출 디렉터리로 복사 함
- 사용자는 작업 완료 시까지 작업 진척 내용을 알 수 없음
- '#PBS –W sandbox=PRIVATE'을 추가하여 스크립트를 작성하는 경우, STDOUT과 STDERR을 작업 실행 중 확인 가능

Job Script 작성

➢ 사용 가능한 환경 변수

환경 변수	설명
PBS_JOBID	Job에 할당되는 식별자
PBS_JOBNAME	사용자에 의해 제공되는 Job 이름
PBS_NODEFILE	작업에 할당된 계산 노드들의 리스트를 포함하고 있는 파일 이름
PBS_O_PATH	제출 환경의 경로 값
PBS_O_WORK_DIR	qsub이 실행된 절대 경로 위치
TMPDIR	Job을 위해 지정된 임시 디렉터리

Job Script 작성

➢ 사용하는 어플리케이션에 맞게 PBS -A 옵션을 지정해야 함(2019년 4월 기준)

Application 종류	PBS -A {이름}	Application 종류	PBS -A {이름}
MOM	mom	VASP	vasp
WRF	wrf	ANSYS	ansys
ROMs	roms	Abaqus	abaqus
MPAS	mpas	Gaussian	gaussian
CESM	cesm	Nastran	nastran
LAMMPS	lammps	LS-DYNA	lsdyna
NAMD	namd	Charmm	charm
Amber	amber	Gromacs	gromacs
OpenFoam	openfoam	Quantum Espresso	qe
QMCpack	qmc	그 외 applications	etc
BWA	bwa		

– 예: 그 외 applications : #PBS -A etc

PBS Job Script 사용 예제: (PI 코드)

➢ 코드 컴파일

– Intel Compiler/MPI 사용

```
$ module add intel/18.0.3 impi/18.0.3
```

– KNL(Knights Landing) 노드 사용시

• craype-mic-knl 모듈 사용

```
$ module add craype-mic-knl
$ icc -xMIC-AVX512 source.c -o executable.x
```

– SKL(Skylake) 노드 사용시

• craype-x86-skylake 모듈 사용

```
$ module add craype-x86-skylake
$ icc -xCORE-AVX512 source.c -o executable.x
```

– craype-mic-knl 모듈과 craype-x86-skylake 모듈을 동시에 사용할 수 없음

• 모듈을 변경할 때 충돌되는 모듈을 unload하고, 사용하고자 하는 모듈을 load 해야 함

PBS Job Script 사용 예제: (PI 코드)

- 컴파일(pi.c)
  - KNL

```
$ module add craype-mic-knl
$ icc pi.c -o pi_serial_no_vec
$ icc -xMIC-AVX512 pi.c -o pi_serial_vec
```

  - SKL

```
$ module rm craype-mic-knl
$ module add craype-x86-skylake
$ icc pi.c -o pi_serial_no_vec
$ icc -xCORE-AVX512 pi.c -o pi_serial_vec
```


PBS Job Script 사용 예제: (PI 코드)

- serial.sh(KNL)

```
$ cat serial.sh
#!/bin/bash
#PBS -V
#PBS -N Serial_job
#PBS -q normal
#PBS -l walltime=00:05:00
#PBS -l select=1:ncpus=1
#PBS -A etc
cd $PBS_O_WORKDIR
./pi_serial_no_vec
./pi_serial_vec
```

serial.sh(SKL)

```
$ cat serial.sh
#!/bin/bash
#PBS -V
#PBS -N Serial_job
#PBS -q norm_skl
#PBS -l walltime=00:05:00
#PBS -l select=1:ncpus=1
#PBS -A etc
cd $PBS_O_WORKDIR
./pi_serial_no_vec
./pi_serial_vec
```

KNL

```
w/o AVX512
PI= 3.141592653589798 (Error = 4.440892e-15)
Elapsed Time = 57.227066, [sec]

w/ AVX512
PI= 3.141592653589845 (Error = 5.151435e-14)
Elapsed Time = 22.057640, [sec]
```

SKL

```
w/o AVX512
PI= 3.141592653589798 (Error = 4.440892e-15)
Elapsed Time = 6.036585, [sec]

w/ AVX512
PI= 3.141592653589783 (Error = 9.769963e-15)
Elapsed Time = 3.929958, [sec]
```

PBS Job Script 사용 예제: (PI 코드)

- 컴파일(piOpenMP.c)
  - KNL

```
$ module add craype-mic-knl
$ icc -qopenmp piOpenMP.c -o piOpenMP_no_vec
$ icc -qopenmp  -xMIC-AVX512 piOpenMP.c -o piOpenMP_vec
```

  - SKL

```
$ module rm craype-mic-knl
$ module add craype-x86-skylake
$ icc -qopenmp piOpenMP.c -o piOpenMP_no_vec
$ icc -qopenmp  -xCORE-AVX512 piOpenMP.c -o piOpenMP_vec
```


PBS Job Script 사용 예제: (PI 코드)

- openmp.sh(KNL)

```
$ cat openmp.sh
#!/bin/bash
#PBS -V
#PBS -N OMP_job
#PBS -q normal
#PBS -l walltime=00:02:00
#PBS -l select=1:ncpus=34:ompthreads=34
(#PBS -l select=1:ncpus=68:ompthreads=68)
#PBS -A etc
cd $PBS_O_WORKDIR
./piOpenMP_no_vec
./piOpenMP_vec
```

openmp.sh(SKL)

```
$ cat openmp.sh
#!/bin/bash
#PBS -V
#PBS -N OMP_job
#PBS -q norm_skl
#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=20:ompthreads=20
(#PBS -l select=1:ncpus=40:ompthreads=40)
#PBS -A etc
cd $PBS_O_WORKDIR
./piOpenMP_no_vec
./piOpenMP_vec
```

KNL	SKL
# of threads : 34 w/o AVX512 Elapsed Time = 1.647456, [sec] w/ AVX512 Elapsed Time = 0.626319, [sec]	# of threads : 20 w/o AVX512 Elapsed Time = 0.316807, [sec] w/ AVX512 Elapsed Time = 0.199470, [sec]
# of threads : 68 w/o AVX512 Elapsed Time = 0.868751, [sec] w/ AVX512 Elapsed Time = 0.350071, [sec]	# of threads : 40 w/o AVX512 Elapsed Time = 0.259656, [sec] w/ AVX512 Elapsed Time = 0.162671, [sec]

PBS Job Script 사용 예제: (PI 코드)

- 컴파일(piMPI.c)
  - KNL

```
$ module add craype-mic-knl
$ mpiicc piMPI.c -o piMPI_no_vec
$ mpiicc -xMIC-AVX512 piMPI.c -o piMPI_vec
```

  - SKL

```
$ module rm craype-mic-knl
$ module add craype-x86-skylake
$ mpiicc piMPI.c -o piMPI_no_vec
$ mpiicc -xCORE-AVX512 piMPI.c -o piMPI_vec
```


PBS Job Script 사용 예제: (PI 코드)

- mpi.sh(KNL)

```
$ cat mpi.sh
#!/bin/bash
#PBS -V
#PBS -N MPI_job
#PBS -q normal
#PBS -l walltime=00:02:00
#PBS -l select=1:ncpus=68:mpiprocs=68:ompthreads=1
(#PBS -l select=2:ncpus=68:mpiprocs=68:ompthreads=1)
#PBS -A etc
cd $PBS_O_WORKDIR
mpirun -machinefile $PBS_NODEFILE ./piMPI_no_vec
mpirun -machinefile $PBS_NODEFILE ./piMPI_vec
```

KNL

```
# of processes : 68
w/o AVX512
Elapsed Time = 1.587632, [sec]
w/ AVX512
Elapsed Time = 0.900600, [sec]
```

```
# of processes : 136
w/o AVX512
Elapsed Time = 0.792766, [sec]
w/ AVX512
Elapsed Time = 0.489747, [sec]
```

PBS Job Script 사용 예제: (PI 코드)

➢ mpi.sh(SKL)

```
$ cat mpi.sh
#!/bin/bash
#PBS -V
#PBS -N MPI_job
#PBS -q norm_skl
#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=40:mpiprocs=40:ompthreads=1
(#PBS -l select=2:ncpus=40:mpiprocs=40:ompthreads=1)
#PBS -A etc
cd $PBS_O_WORKDIR
mpirun -machinefile $PBS_NODEFILE ./piMPI_no_vec
mpirun -machinefile $PBS_NODEFILE ./piMPI_vec
```

SKL

```
# of processes : 40
w/o AVX512
Elapsed Time = 0.176598, [sec]
w/ AVX512
Elapsed Time = 0.162338, [sec]
```

```
# of processes : 80
w/o AVX512
Elapsed Time = 0.094650, [sec]
w/ AVX512
Elapsed Time = 0.327157, [sec]
```


PBS Job Script 사용 예제: (PI 코드)

➢ 컴파일(piHybrid.c)

– KNL

```
$ module add craype-mic-knl
$ mpiicc -qopenmp piHybrid.c -o piHybrid_no_vec
$ mpiicc -qopenmp -xMIC-AVX512 piHybrid.c -o piHybrid_vec
```

– SKL

```
$ module rm craype-mic-knl
$ module add craype-x86-skylake
$ mpiicc -qopenmp piHybrid.c -o piHybrid_no_vec
$ mpiicc -qopenmp -xCORE-AVX512 piHybrid.c -o piHybrid_vec
```

PBS Job Script 사용 예제: (PI 코드)

- hybrid.sh(KNL)

```
$ cat hybrid.sh
#!/bin/bash
#PBS -V
#PBS -N Hybrid_job
#PBS -q normal
#PBS -l walltime=00:02:00
#PBS -l select=2:ncpus=68:mpiprocs=2:ompthreads=34
#PBS -A etc
cd $PBS_O_WORKDIR
mpirun -machinefile $PBS_NODEFILE ./piHybrid_no_vec
mpirun -machinefile $PBS_NODEFILE ./piHybrid_vec
```

KNL

```
# of processes : 4
w/o AVX512
Elapsed Time = 0.940793, [sec]
---------------------------------------
# of processes : 4
w/ AVX512
Elapsed Time = 0.562912, [sec]
```


PBS Job Script 사용 예제: (PI 코드)

- hybrid.sh(SKL)

```
$ cat hybrid.sh
#!/bin/bash
#PBS -V
#PBS -N Hybrid_job
#PBS -q norm_skl
#PBS -l walltime=00:02:00
#PBS -l select=2:ncpus=40:mpiprocs=2:ompthreads=20
#PBS -A etc
cd $PBS_O_WORKDIR
mpirun -machinefile $PBS_NODEFILE ./piHybrid_no_vec
mpirun -machinefile $PBS_NODEFILE ./piHybrid_vec
```

SKL

```
# of processes : 4
# of threads : 20
w/o AVX512
Elapsed Time = 0.117037, [sec]
---------------------------------------
# of processes : 4
w/ AVX512
Elapsed Time = 0.091773, [sec]
```

PBS Interactive 작업 제출

- 누리온 시스템은 debug 노드 대신 debug 큐를 제공
- debug 큐를 이용하여 작업을 제출함으로써 디버깅 수행이 가능
- qsub –I (대문자 i 임)
  - qsub를 이용한 Interactive 작업 사용 예 (MPI)

```
[sedu01@pbcm Pi_Calc]$ qsub -I -V -A etc -l select=1:ncpus=68:mpiprocs=68 -l
walltime=00:10:00 -q debug
qsub: waiting for job 6719.pbcm to start
qsub: job 6719.pbcm ready

Intel(R) Parallel Studio XE 2017 Update 2 for Linux*
Copyright (C) 2009-2017 Intel Corporation. All rights reserved.

[sedu01@node8281 ~]$ cd $PBS_O_WORKDIR
[sedu01@node8281 ~]$ mpirun -n 68 ./piMPI_vec
[sedu01@node8281 Pi_Calc]$ mpirun -np 68 ./piMPI_vec
# of processes : 68
PI= 3.141592653989790 (Error = 3.999969e-10)
Elapsed Time = 3.176321, [sec]
---------------------------------------
[sedu01@node8281 ~]$ exit
[sedu01@login04 ~] $
```


PBS Interactive 작업

- Interactive 작업 조회: qstat, pbsnodes

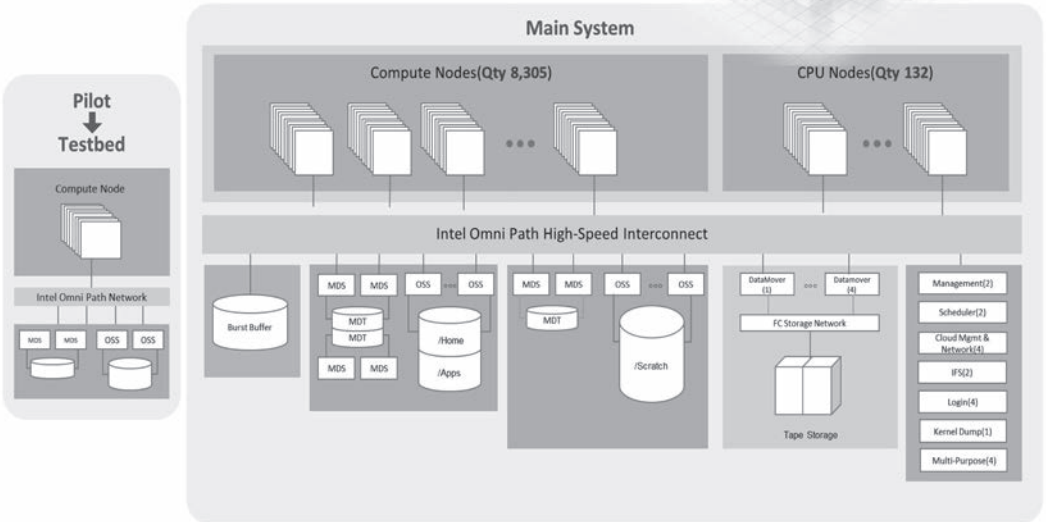
```
$ qstat
Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
6538.pbcm         vasp_07          hskim0            11830:20 R knl
6615.pbcm         vasp_13          hskim0            4664:02: R knl
6628.pbcm         ESM_pos2_0.0139  hskim0            2387:09: R cpu
6638.pbcm         vasp_16          hskim0            2536:51: R knl
6641.pbcm         vasp_18          hskim0            2533:49: R knl
6643.pbcm         ESM_pos1_0.0139  hskim0            1177:07: R cpu
6644.pbcm         ESM_pos1_0.0559  hskim0            1176:39: R cpu
6719.pbcm         STDIN            sedu01             00:05:30 R knl

$ pbsnodes -aSj
                                                      mem       ncpus   nmics   ngpus
vnode           state           njobs   run   susp      f/t        f/t     f/t     f/t   jobs
--------------- --------------- ------ ----- ------ ------------ -------- ------- ------- -----
--
node8281        job-busy            1     1      0  110gb/110gb    0/68     0/0     0/0 6719
node8282        free                1     1      0  110gb/110gb    4/68     0/0     0/0 6638
…
node0010        free                0     0      0  110gb/110gb   68/68     0/0     0/0 --
cpu0004         job-busy            1     1      0  188gb/188gb    0/40     0/0     0/0 6643
cpu0003         job-busy            1     1      0  188gb/188gb    0/40     0/0     0/0 6644
cpu0002         job-busy            1     1      0  188gb/188gb    0/40     0/0     0/0 6628
cpu0001         free                0     0      0  188gb/188gb   40/40     0/0     0/0 --
```

PFS & Burst Buffer 사용법

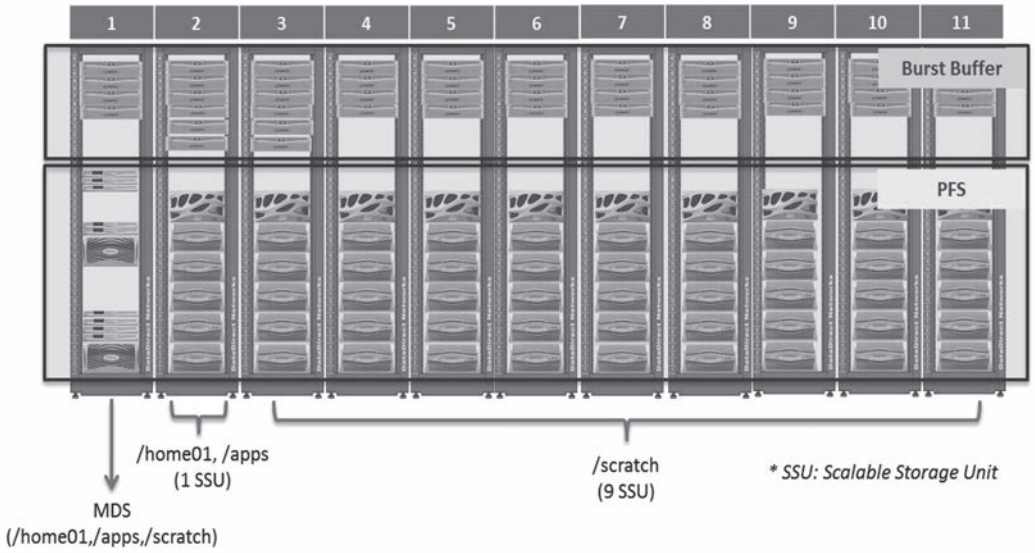
1. Burst Buffer, PFS Rack configuration (DDN)
2. Nurion storage summary (Burst Buffer, PFS)
3. Nurion storage user quota & policy
4. Burst Buffer IME
5. How to use Burst Buffer on Nurion

KISTI 5th Supercomputer

- 128 Racks of Cluster Components
  - 8 rows, with 16 Cabinets in each Row
  - 8,305 Compute Nodes / 132 CPU Nodes



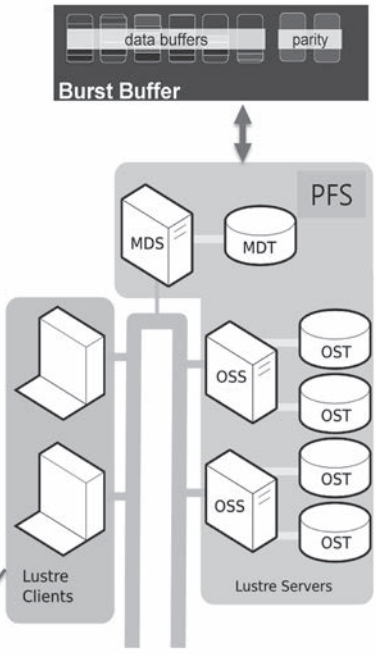
Burst Buffer, PFS Rack configuration (DDN)



한국과학기술정보연구원

Nurion Storage Summary (Burst Buffer, PFS)

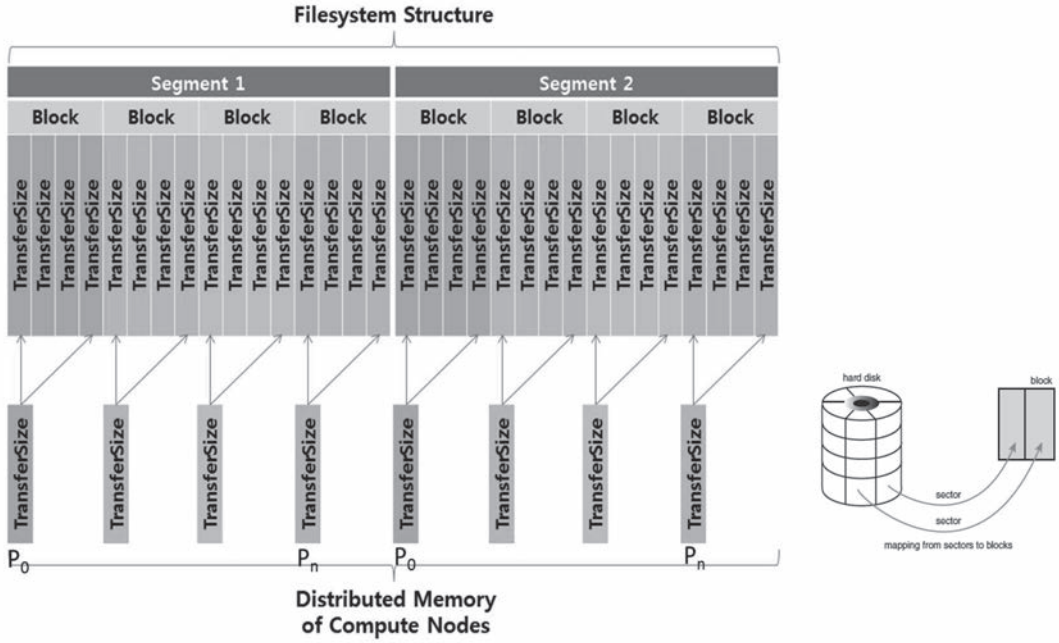
- Burst Buffer
  - 48x IME240
  - Performance : 0.8TB/s, Capacity : 960TB
- Scratch
  - 9x ES14KX (Lustre Embedded Storage System)
  - Performance : 300GB/s, Capacity : 21PB
- Home/Apps
  - 1x ES14KX
  - Performance : 30GB/s, Capacity : 1.26PB



Computing Nodes, Login Nodes, Data Mover, etc.

한국과학기술정보연구원

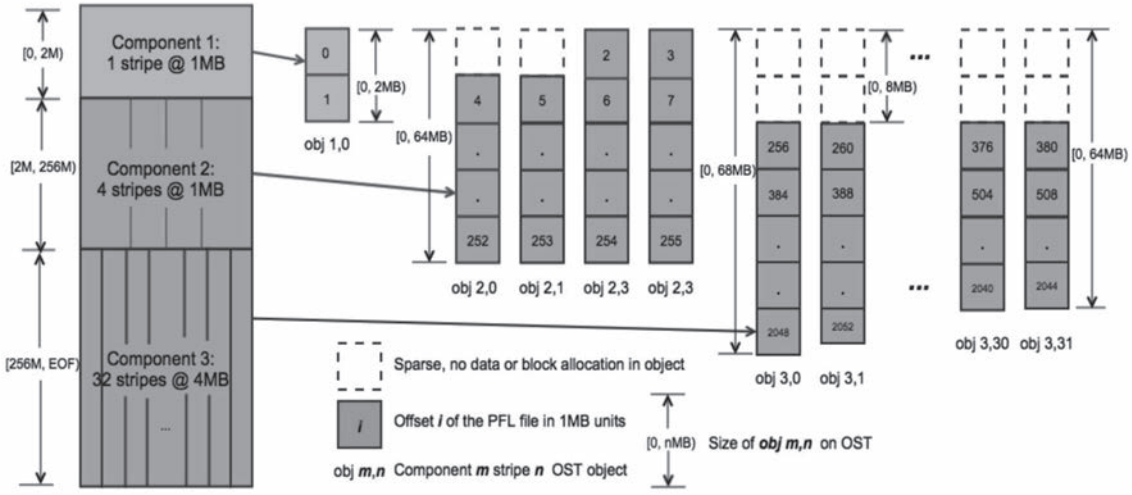
Parallel File System Overview



✓ *FileSize = BlockSize × Number of Tasks (P_n) × SegmentCount*

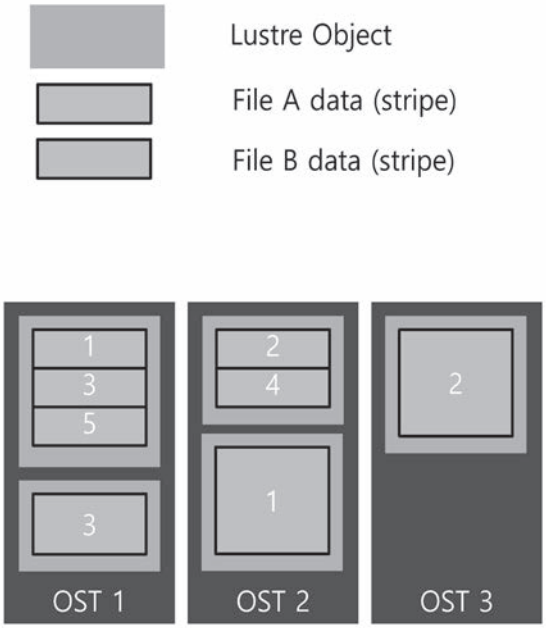
Progressive File Layouts - PFL

- 사용자의 스트라이핑 패턴 문제
  - 큰 파일, 작은 스트라이프 카운트 -> OST 사용량의 불균형
  - 작은 파일, 큰 스트라이프 카운트 -> 성능의 하락
- PFL(Progressive File Layout)은 여러 레이어의 조합을 사용



Lustre Striping

- 파일데이터는 OST에 저장
- OST
  - OSS를 통해 서비스
  - 스트라이프 데이트 저장
  - 로컬파일시스템으로 포맷 (ext4, ldiskfs, ZFS)
- 파일 데이터는 다수의 OSTs를 통해 스트라이프 가능



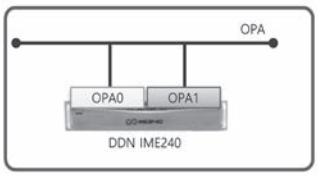
Nurion storage user quota & policy

- Job submission only from the */scratch* FS

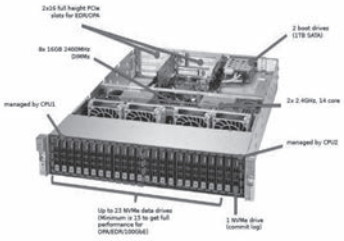
디렉터리	용량제한	파일 수 제한	파일 삭제 정책	파일시스템	백업유무
/home01	64GB	200K	N/A	Lustre	O
/scratch	100TB	1M	15일 동안 접근하지 않은 파일은 자동 삭제		X

- How to check your FS usage?
  - $ lfs quota –h /home01
  - $ lfs quota –h /scratch

Burst Buffer (IME240)



주요 구성 요소	내역
시스템	DDN IME240, Infinite Memory Engine Appliance
소프트웨어 버전	CentOS 7.3, IME 1.2
최대 IO 성능	20 GB/s
네트워크 인터페이스	2x OPA
SSD 타입	1.2TB, NVMe 드라이브
SSD 수량	16ea(1.2TB NVMe)+1ea(450GB SSD)
용량(RAW)	20.4TB
전력량(W,Nominal)	1,028
발열량(BTU/hr,Nominal)	3,506

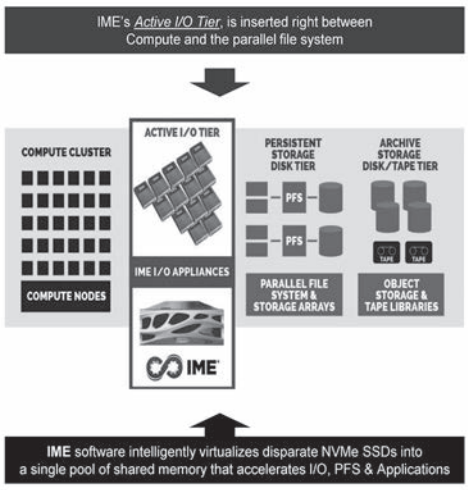


8(0~7) 8(12~19)+1(23) → 17EA NVMe SSD

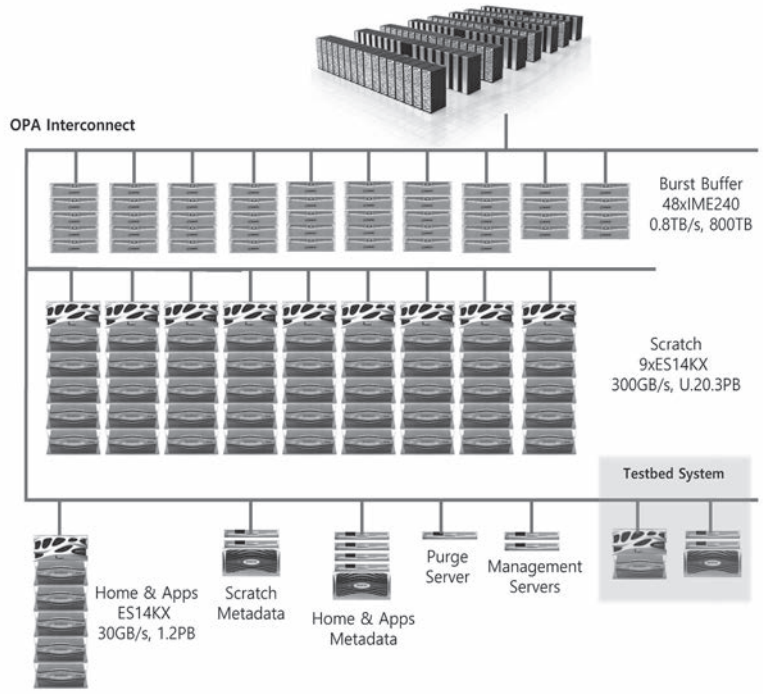
주요 구성 요소	내역
총 시스템 수	48 x IME240
총 용량	979.2TB
총 최대 IO 성능	약 800GB/s
총 전력량(W,Nominal)	41,120
총 발열량(BTU/hr,Nominal)	140,307

What is IME ?

- New Cache Layer using NVMe SSDs inserted between compute cluster and PFS
  - IME is configured as CLUSTER with multiple SSD servers
  - All compute nodes can access cache data on IME
- Accelerates "bad" IO on PFS
- Accelerates small IO or random IO by high IOPS due to SSD and IO management
  - PFS is pretty good at large sequential IO
- Can be configured as cache layer having huge IO bandwidth



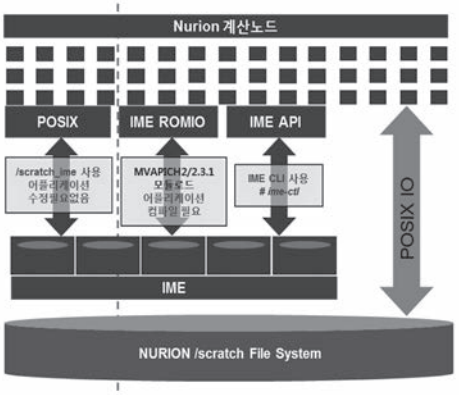
Storage Diagram



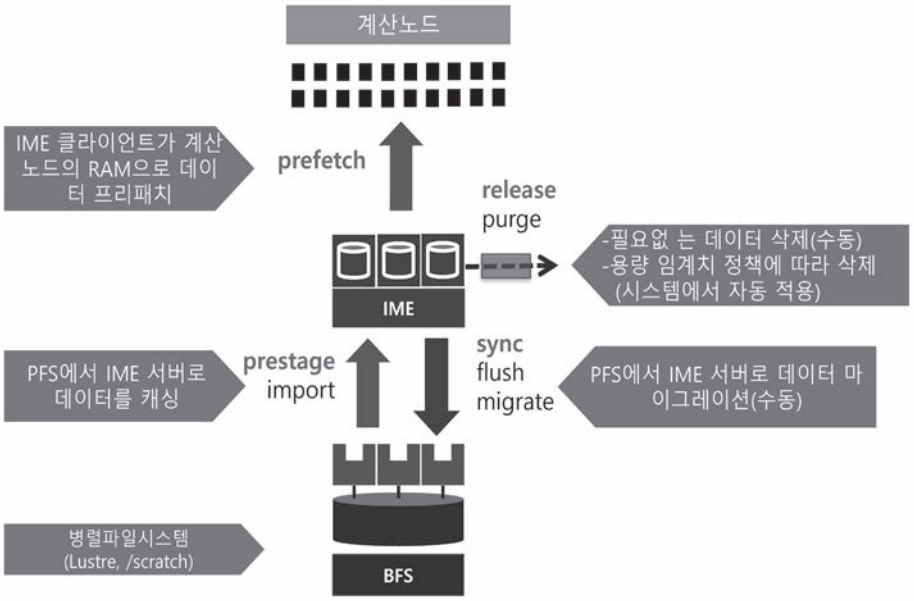
How to use burst buffer on Nurion

- Burst buffer is available on all compute nodes (/scratch_ime)
  - Mandatory option (#PBS -P burst_buffer)

```
#!/bin/bash
#PBS -N bb_test
#PBS -V
#PBS -q normal
#PBS -A etc
#PBS -P burst_buffer
#PBS -l select=4:ncpus=1:mpiprocs=1
#PBS -l walltime=48:00:00
cd $PBS_O_WORKDIR

# 아래 ① 또는 ② 선택
OUTFILE=/scratch_ime/$USER/output.dat
# ① IME마운트 포인트 사용(POSIX) 또는

OUTFILE=ime:///scratch/$USER/output.dat
# ② mvapich2/2.3.1모듈 사용(MPI-IO)

mpirun ./a.out > $OUTFILE
```



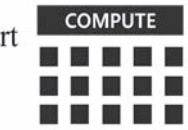
Data Movement

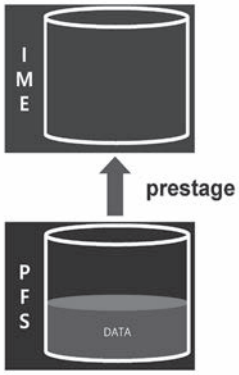


Data Residency Control

➢ Initial Step is to prestage data from the Parallel Filesystem into IME prior to job start

```
ime-ctl -i $INPUT_FILE
```





Data Residency Control

- Initial Step is to prestage data from the Parallel Filesystem into IME prior to job start

```
ime-ctl -i $INPUT_FILE
```

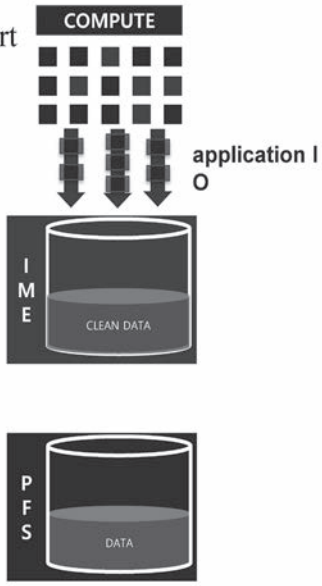
- Now data is in IME in a CLEAN state (i.e. corresponding data also exists in the PFS)

Data Residency Control

- Initial Step is to prestage data from the Parallel Filesystem into IME prior to job start

```
ime-ctl -i $INPUT_FILE
```

- Now data is in IME in a CLEAN state (i.e. corresponding data also exists in the PFS)
- IO commences from application creating new (DIRTY) data in IME cache



Data Residency Control

- Initial Step is to prestage data from the Parallel Filesystem into IME prior to job start

```
ime-ctl -i $INPUT_FILE
```

- Now data is in IME in a CLEAN state (i.e. corresponding data also exists in the PFS)
- IO commences from application creating new (DIRTY) data in IME cache

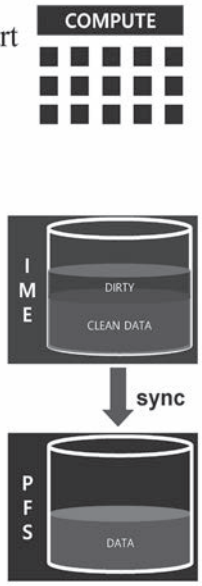
Data Residency Control

- Initial Step is to prestage data from the Parallel Filesystem into IME prior to job start

```
ime-ctl -i $INPUT_FILE
```

- Now data is in IME in a CLEAN state (i.e. corresponding data also exists in the PFS)
- IO commences from application creating new (DIRTY) data in IME cache
- DIRTY data is synced to the PFS

```
ime-ctl -r $OUTPUT_FILE
```



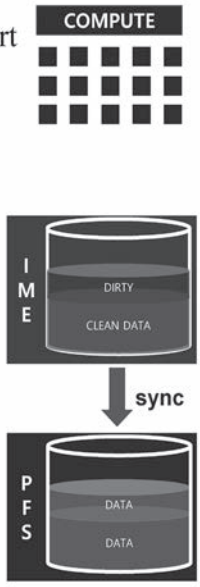
Data Residency Control

- Initial Step is to prestage data from the Parallel Filesystem into IME prior to job start

```
ime-ctl -i $INPUT_FILE
```

- Now data is in IME in a CLEAN state (i.e. corresponding data also exists in the PFS)
- IO commences from application creating new (DIRTY) data in IME cache
- DIRTY data is synced to the PFS

```
ime-ctl -r $OUTPUT_FILE
```



Data Residency Control

- Initial Step is to prestage data from the Parallel Filesystem into IME prior to job start

```
ime-ctl -i $INPUT_FILE
```

- Now data is in IME in a CLEAN state (i.e. corresponding data also exists in the PFS)
- IO commences from application creating new (DIRTY) data in IME cache
- DIRTY data is synced to the PFS

```
ime-ctl -r $OUTPUT_FILE
```

- and is marked clean in IME

Data Residency Control

- Initial Step is to prestage data from the Parallel Filesystem into IME prior to job start

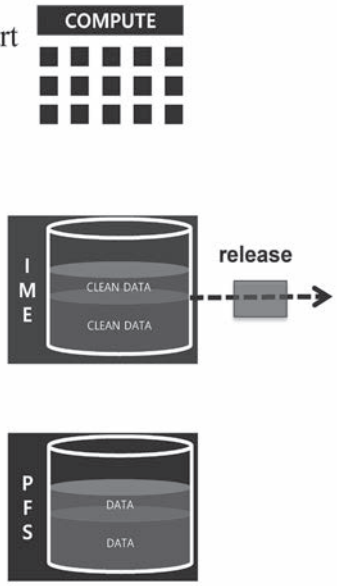
```
ime-ctl -i $INPUT_FILE
```

- Now data is in IME in a CLEAN state (i.e. corresponding data also exists in the PFS)
- IO commences from application creating new (DIRTY) data in IME cache
- DIRTY data is synced to the PFS

```
ime-ctl -r $OUTPUT_FILE
```

- and is marked clean in IME
- Finally clean data in IME can be purged ready for a new job

```
ime-ctl -p $TMP_FILE
```



Data Residency Control

- Initial Step is to prestage data from the Parallel Filesystem into IME prior to job start

```
ime-ctl -i $INPUT_FILE
```

- Now data is in IME in a CLEAN state (i.e. corresponding data also exists in the PFS)
- IO commences from application creating new (DIRTY) data in IME cache
- DIRTY data is synced to the PFS

```
ime-ctl -r $OUTPUT_FILE
```

- and is marked clean in IME
- Finally clean data in IME can be purged ready for a new job

```
ime-ctl -p $TMP_FILE
```

한국과학기술정보연구원

Data Movement

- Scheduler executes Stage-In for job data

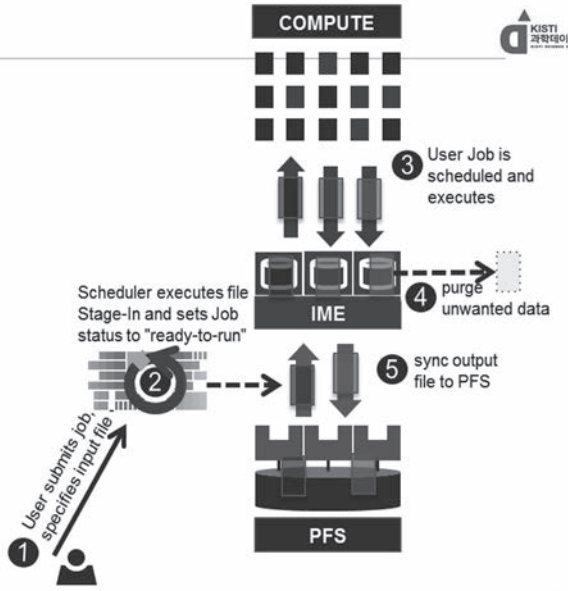
```
# ime-ctl -i $INPUT_FILE
```

- and then sets the job as "ready to run"
- Job Executes: All reads/writes to IME
- Temporary data that is not required can be purged from IME

```
# ime-ctl -p $TMP_FILE
```

- Completion of the User Job initiates sync of output data to PFS

```
# ime-ctl -r $OUTPUT_FILE
```



✓	ime-ctl -i $INPUT_FILE	작업 데이터에 대해 Stage-In 실행 (/scratch에서 /scratch_ime로 데이터 캐싱)
✓	ime-ctl -r $OUTPUT_FILE	BB 캐시 데이터를 PFS와 동기화 (/scratch_ime 캐싱 데이터를 /scratch로 전송)
✓	ime-ctl -p $TMP_FILE	BB 캐시의 데이터 삭제 (/scratch_ime 퍼지)
✓	ime-ctl -s $FILE	BB에 캐시된 데이터의 상태정보 제공

한국과학기술정보연구원

Data Movement

PFS (/scratch)

```
[jwyoon@login01 ime_test]$ pwd
/scratch/jwyoon/ime_test
[jwyoon@login01 ime_test]$ dd if=/dev/zero
of=/scratch/jwyoon/ime_test/test1 bs=10M count=10
10+0 records in
10+0 records out
104857600 bytes (105 MB) copied, 0.218153 s, 481 MB/s
[jwyoon@login01 ime_test]$ls
test1  test2
```

```
[jwyoon@login01 ime_test]$ ime-ctl -p test2
[jwyoon@login01 ime_test]$ ime-ctl -s test2
  File: `test2'
        Number of bytes:
  Dirty: 0
  Clean: 0
Syncing: 0
```

Burst Buffer (/scratch_ime)

```
[jwyoon@login01 ime_test]$ pwd
/scratch_ime/jwyoon/ime_test
[jwyoon@login01 ime_test]$ ls
test1
[jwyoon@login01 ime_test]$ dd if=/dev/zero
of=/scratch_ime/jwyoon/ime_test/test2 bs=10M
count=10
10+0 records in
10+0 records out
104857600 bytes (105 MB) copied, 0.188909 s, 555
MB/s
[jwyoon@login01 ime_test]$ ime-ctl -s test1
  File: `test1'
        Number of bytes:
  Dirty: 0
  Clean: 0
Syncing: 0
[jwyoon@login01 ime_test]$ ime-ctl -s test2
  File: `test2'
        Number of bytes:
  Dirty: 104857600
  Clean: 0
Syncing: 0
[jwyoon@login01 ime_test]$ ime-ctl -r test2
[jwyoon@login01 ime_test]$ ime-ctl -s test2
  File: `test2'
        Number of bytes:
  Dirty: 0
  Clean: 104857600
Syncing: 0
```

한국과학기술정보연구원

Burst Buffer usage check

```
# imequota
[ ime01 ] Usage: 1.164TB(used) 19.236TB(free) 20.400(total) - Avail: 94.289%
[ ime02 ] Usage: 1.161TB(used) 19.239TB(free) 20.400(total) - Avail: 94.299%
[ ime03 ] Usage: 1.161TB(used) 19.239TB(free) 20.400(total) - Avail: 94.302%
[ ime04 ] Usage: 1.161TB(used) 19.239TB(free) 20.400(total) - Avail: 94.299%
[ ime05 ] Usage: 1.161TB(used) 19.239TB(free) 20.400(total) - Avail: 94.295%
[ ime06 ] Usage: 1.163TB(used) 19.237TB(free) 20.400(total) - Avail: 94.296%
[ ime07 ] Usage: 1.165TB(used) 19.235TB(free) 20.400(total) - Avail: 94.277%
[ ime08 ] Usage: 1.161TB(used) 19.239TB(free) 20.400(total) - Avail: 94.297%
[ ime09 ] Usage: 1.161TB(used) 19.239TB(free) 20.400(total) - Avail: 94.298%
[ ime10 ] Usage: 1.162TB(used) 19.238TB(free) 20.400(total) - Avail: 94.298%
                                •••••
[ ime41 ] Usage: 1.160TB(used) 19.240TB(free) 20.400(total) - Avail: 94.304%
[ ime42 ] Usage: 1.160TB(used) 19.240TB(free) 20.400(total) - Avail: 94.302%
[ ime43 ] Usage: 1.160TB(used) 19.240TB(free) 20.400(total) - Avail: 94.296%
[ ime44 ] Usage: 1.160TB(used) 19.240TB(free) 20.400(total) - Avail: 94.294%
[ ime45 ] Usage: 1.165TB(used) 19.235TB(free) 20.400(total) - Avail: 94.291%
[ ime46 ] Usage: 1.166TB(used) 19.234TB(free) 20.400(total) - Avail: 94.282%
[ ime47 ] Usage: 1.164TB(used) 19.236TB(free) 20.400(total) - Avail: 94.298%
[ ime48 ] Usage: 1.163TB(used) 19.237TB(free) 20.400(total) - Avail: 94.296%
--------------------------------------------------------------------------------
Total Usage: 55.814TB(used) 923.386TB(free) 979.200TB(total) - Avail: 94.300%
--------------------------------------------------------------------------------
```

※ 1시간 단위로 갱신되어 반영

한국과학기술정보연구원

Data Movement

PFS (/scratch)

```
[jwyoon@login01 ime_test]$ pwd
/scratch/jwyoon/ime_test
[jwyoon@login01 ime_test]$ dd if=/dev/zero
of=/scratch/jwyoon/ime_test/test1 bs=10M count=10
10+0 records in
10+0 records out
104857600 bytes (105 MB) copied, 0.218153 s, 481 MB/s
[jwyoon@login01 ime_test]$ls
test1  test2
```

```
[jwyoon@login01 ime_test]$ ime-ctl -p test2
[jwyoon@login01 ime_test]$ ime-ctl -s test2
  File: `test2'
       Number of bytes:
  Dirty: 0
  Clean: 0
Syncing: 0
```

Burst Buffer (/scratch_ime)

```
[jwyoon@login01 ime_test]$ pwd
/scratch_ime/jwyoon/ime_test
[jwyoon@login01 ime_test]$ ls
test1
[jwyoon@login01 ime_test]$ time dd if=/dev/zero
of=/scratch_ime/jwyoon/ime_test/test2 bs=10M
count=10
10+0 records in
10+0 records out
104857600 bytes (105 MB) copied, 0.188909 s, 555
MB/s
[jwyoon@login01 ime_test]$ ime-ctl -s test1
  File: `test1'
       Number of bytes:
  Dirty: 0
  Clean: 0
Syncing: 0
[jwyoon@login01 ime_test]$ ime-ctl -s test2
  File: `test2'
       Number of bytes:
  Dirty: 104857600
  Clean: 0
Syncing: 0
[jwyoon@login01 ime_test]$ ime-ctl -r test2
[jwyoon@login01 ime_test]$ ime-ctl -s test2
  File: `test2'
       Number of bytes:
  Dirty: 0
  Clean: 104857600
Syncing: 0
```

한국과학기술정보연구원

Job Submit Using Burst Buffer

```
#!/bin/bash -l
#PBS -l select=1
#PBS -q bb_cache (or bb_
INPUT=/scratch/$USER/input.dat
FILE=/scratch/$USER/output.dat
echo "[ `date` ] PFS: start"
dd if=$INPUT of=$FILE bs=1M count=$(( 50 * 1024 ))
echo "[ `date` ] PFS: end"
INPUT=/scratch_ime/$USER/input.dat
FILE=/scratch_ime/$USER/output.dat
# stage in
/opt/ddn/ime/bin/ime-ctl -i $INPUT
/opt/ddn/ime/bin/ime-ctl -s $INPUT ①
echo "[ `date` ] IME: start"
dd if=$INPUT of=$FILE bs=1M count=$(( 50 * 1024 )) ②
echo "[ `date` ] IME: end"
/opt/ddn/ime/bin/ime-ctl -s $FILE ③
# release
/opt/ddn/ime/bin/ime-ctl -r $FILE
/opt/ddn/ime/bin/ime-ctl -s $FILE ④
# purge (주의)
/opt/ddn/ime/bin/ime-ctl -p $FILE ⑤
```

① ※ 출력값
File: /scratch_ime/$USER/input.dat
Number of bytes:
Dirty: 0
Clean: 51200 (BB로 파일 stage-in 됨)
Syncing: 0

② ※ BB의 output.dat 파일이 변경

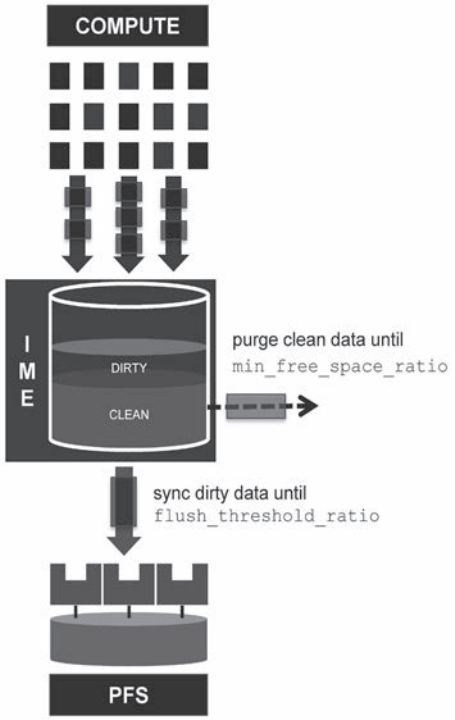
③ ※ 출력값
File: `output.dat'
Number of bytes:
Dirty: 53687091200 (PFS-BB의 데이터가 다름)
Clean: 0
Syncing: 0

④ ※ 출력값
File: `output.dat'
Number of bytes:
Dirty: 0
Clean: 4085252096 (BB 데이터 PFS로 release)
Syncing: 49601839104

⑤ ※ BB 데이터 캐쉬를 삭제 (purge)

Data Residency Control

- maximum percentage of dirty data resident in IME before the data is automatically synchronized to the PFS:
  `flush_threshold_ratio 45% [0% ~100%]`
- [AUTO SYNC]
- Once Synchronised, the data is marked clean
- The clean data is kept in IME until the min_free_space_ratio is reached. `min_free_space_ratio 15% [0% ~ 100%]`
- [AUTO RELEASE]



KISTI 한국과학기술정보연구원

THANK
YOU

부록 : 사용된 예제 소스 코드

1. **pi.c** : 순차 코드
2. **piOpenMP.c** : **OpenMP** 병렬코드
3. **piMPI.c** : **MPI** 병렬코드
4. **piHybrid.c** : **Hybrid(MPI + OpenMP)** 병렬코드

(pi.c : 순차 코드)

```
#include <stdio.h>
#include <math.h>
#include <sys/time.h>
inline double cpuTimer(){
    struct timeval tp;
    gettimeofday(&tp,NULL);
    return ((double)tp.tv_sec + (double)tp.tv_usec*1e-6);
}
int main(){
    double iStart, ElapsedTime;
    const long num_step = 5000000000;
    long i;
    double sum, step, pi, x;
    step = (1.0/(double)num_step);                    // Del_x
    sum = 0.0;
    iStart=cpuTimer();
    printf("-------------------------------------\n");
    for(i=1;i<=num_step;i++){
        x = ((double)i-0.5)*step;                              // x_k
        sum += 4.0/(1.0+x*x);                         // f(x_k)
    }
    pi = step*sum;                                    // sum{f(x_k)}*Del_x
    ElapsedTime=cpuTimer() - iStart;
    printf("PI= %.15f (Error = %e)\n",pi, fabs(acos(-1)-pi));
    printf("Elapsed Time = %f, [sec]\n", ElapsedTime);
    printf("---------------------------------------\n");
    return 0;
}
```


(piOpenMP.c : OpenMP 병렬코드)

```
#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include <omp.h>
inline double cpuTimer()
{
    struct timeval tp;
    gettimeofday(&tp,NULL);
    return ((double)tp.tv_sec + (double)tp.tv_usec*1e-6);
}
int main()
{
    double iStart, ElapsedTime;
    const long num_step = 5000000000;
    long i;
    double sum, step, pi, x;
    int num_threads;
    step = (1.0/(double)num_step);
    sum = 0.0;
    iStart=cpuTimer();
    printf("-------------------------------------\n");
```

(piOpenMP.c : OpenMP 병렬코드)

```
#pragma omp parallel
{
#pragma omp master
{
    num_threads=omp_get_num_threads();
    printf("# of threads : %d\n",num_threads);
}
#pragma omp for reduction(+:sum), private(x)
    for(i=1;i<=num_step;i++){
        x = ((double)i-0.5)*step;
        sum += 4.0/(1.0+x*x);
    }
}
    pi = step*sum;
    ElapsedTime= cpuTimer() - iStart;
    printf("PI= %.15f (Error = %e)\n",pi, fabs(acos(-1)-pi));
    printf("Elapsed Time = %f, [sec]\n", ElapsedTime);
    printf("----------------------------------------\n");
    return 0;
}
```


(piMPI.c : MPI 병렬코드)

```
#include <stdio.h>
#include <math.h>
#include "mpi.h"
int main(int argc, char *argv[]){
    long i;     int myrank, nprocs;
    const long num_step = 5000000000;
    double mypi, x, pi, h, sum;
    double st, et;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    if(myrank==0) printf("# of processes : %d\n",nprocs);
    h=1.0/(double)num_step;
    sum = 0.0;
    st = MPI_Wtime();
    for(i=myrank;i<num_step;i+=nprocs){
        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);
    et=MPI_Wtime();
    if(myrank==0){
        printf("PI= %.15f (Error = %e)\n",pi, fabs(acos(-1)-pi));
        printf("Elapsed Time = %f, [sec]\n", et-st);
        printf("----------------------------------------\n");
    }
    MPI_Finalize();
    return 0;
}
```

(piHybrid.c : Hybrid(MPI + OpenMP)병렬코드)

```
#include <stdio.h>
#include <math.h>
#include "mpi.h"
#include "omp.h"

int main(int argc, char *argv[])
{
    long i;
    int myrank, nprocs,provide;
    const long num_step = 5000000000;
    double mypi, x, pi, h, sum;
    double st, et;
    int num_threads;
    MPI_Init_thread(&argc, &argv,MPI_THREAD_FUNNELED,&provide);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    if(myrank==0)printf("# of processes : %d\n",nprocs);


    h=1.0/(double)num_step;
    sum = 0.0;
    st = MPI_Wtime();
```


(piHybrid.c : Hybrid(MPI + OpenMP)병렬코드)

```
#pragma omp parallel
{
#pragma omp master
{
    num_threads=omp_get_num_threads();
    printf("# of threads : %d\n",num_threads);
}
#pragma omp  for reduction(+:sum), private(x)
    for(i=1;i<=num_step;i+=nprocs)
    {
        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);
    et=MPI_Wtime();
    if(myrank==0){
        printf("PI= %.15f (Error = %e)\n",pi, fabs(acos(-1)-pi));
        printf("Elapsed Time = %f, [sec]\n", et-st);
        printf("----------------------------------------\n");
    }
    MPI_Finalize();
    return 0;
}
```

OpenMP Programming

KISTI SUPERCOMPUTING EDUCATION CENTER

OpenMP Programming

August 2020

KISTI과학데이터스쿨
KISTI SCIENCE DATA SCHOOL
http://kacademy.kisti.re.kr

KISTI 과학데이터스쿨

Day 2: OpenMP

- Introduction
  - Parallel Computing, threads, and OpenMP
- Basic
  - Creating Threads
  - Data Environment
  - Parallel Loops
  - Synchronization
  - Scheduling
- Intermediate
  - Tasks

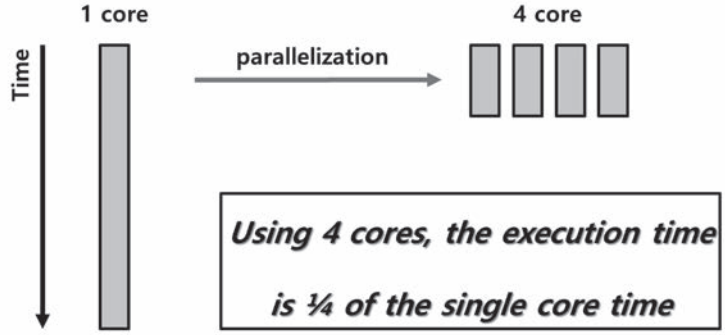
Introduction

- **Parallel Computing, threads, and OpenMP**

Parallel Computing

Parallelization is another optimization technique
The goal is to reduce the execution time

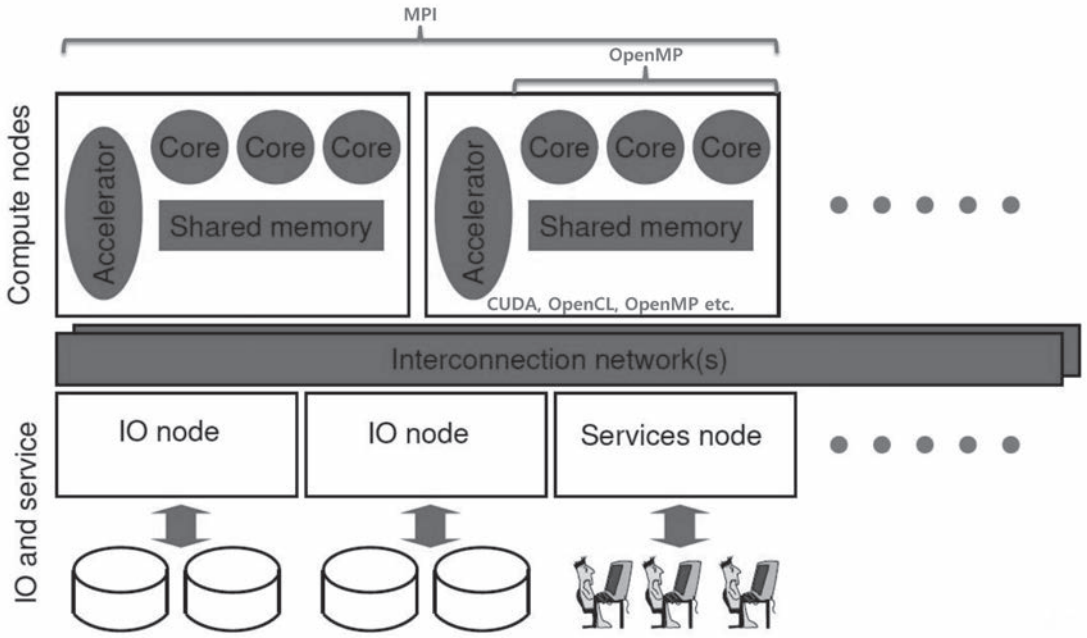
To this end, multiple processors, or cores, are used



Using 4 cores, the execution time is ¼ of the single core time

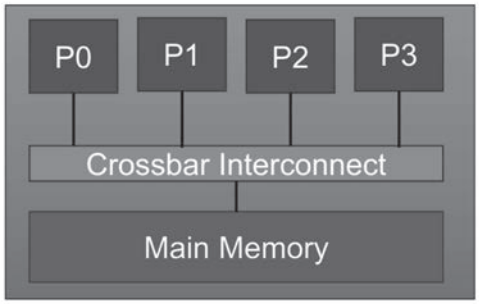
Modern HPC Architecture

Thread[One Node(OS)] ⊂ Process



Shared Memory System Architecture

➢UMA



- 프로그램 용이
- 메인 메모리 대역폭 병목
- 시스템 크기 제한

➢NUMA

- 상대적 높은 메모리 대역폭
- 프로세서 추가로 대역폭 증가
- 프로그램 최적화 난이도 증가

What is OpenMP?

- Open specifications for Multi Processing
- (공유메모리 환경에서) 다중 스레드 병렬 프로그램 작성을 위한 응용프로그램 인터페이스(API)
  - OpenMP Architecture Review Board (ARB)에서 주도
  - 공유 메모리 프로그래밍 모델의 (사실상) 표준

www.openmp.org

History

- 1997.10: OpenMP for Fortran 1.0
- 1998.10: OpenMP for C/C++ 1.0
- 2000.11: OpenMP for Fortran 2.0
- 2002.3: OpenMP C/C++ 2.0
- 2005.5: OpenMP 2.5 (combined C/C++ & Fortran)
- 2008.5: OpenMP 3.0 (Task)
- 2011.7: OpenMP 3.1
- 2013.7: OpenMP 4.0 (Accelerator, SIMD)
- 2015.11: OpenMP 4.5
- 2018.11: OpenMP 5.0

OpenMP Compilers

- **Intel**
  - OpenMP 4.5 C/C++/Fortran supported in version 17.0, 18.0, and 19.0 compilers
  - OpenMP 4.5 and subset of OpenMP 5.0 C/C++/Fortran supported in 19.1 compilers under -qnextgen -fiopenmp.
- **GCC**
  - From GCC 4.9.1, OpenMP 4.0 is fully supported for C/C++/Fortran.
  - From GCC 6.1, OpenMP 4.5 is fully supported for C and C++.
  - From GCC 7.1, OpenMP 4.5 is partially supported for Fortran.
  - From GCC 9.1, OpenMP 5.0 is partially supported for C and C++
- **https://www.openmp.org/resources/openmp-compilers-tools/**

OpenMP API

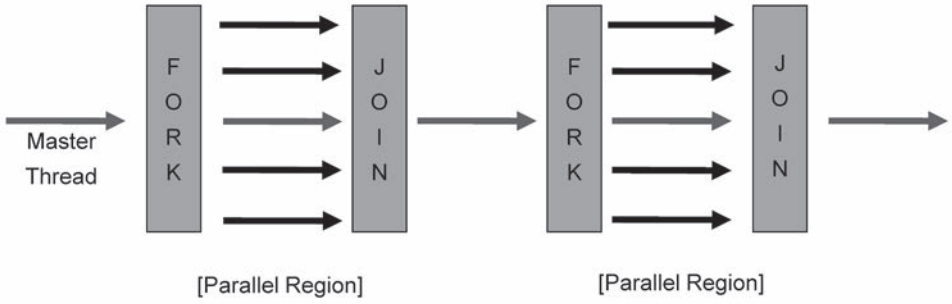
- **Directives + Runtime Library Routines + Environment Variables**
- 스레드 병렬화를 위한 기능 제공
  - 스레드 생성과 소멸
  - 각 스레드에 작업 할당/분배
  - 스레드 공유 데이터, 사유 데이터 지정
  - 공유 데이터에 대한 스레드 접근 조정

OpenMP API

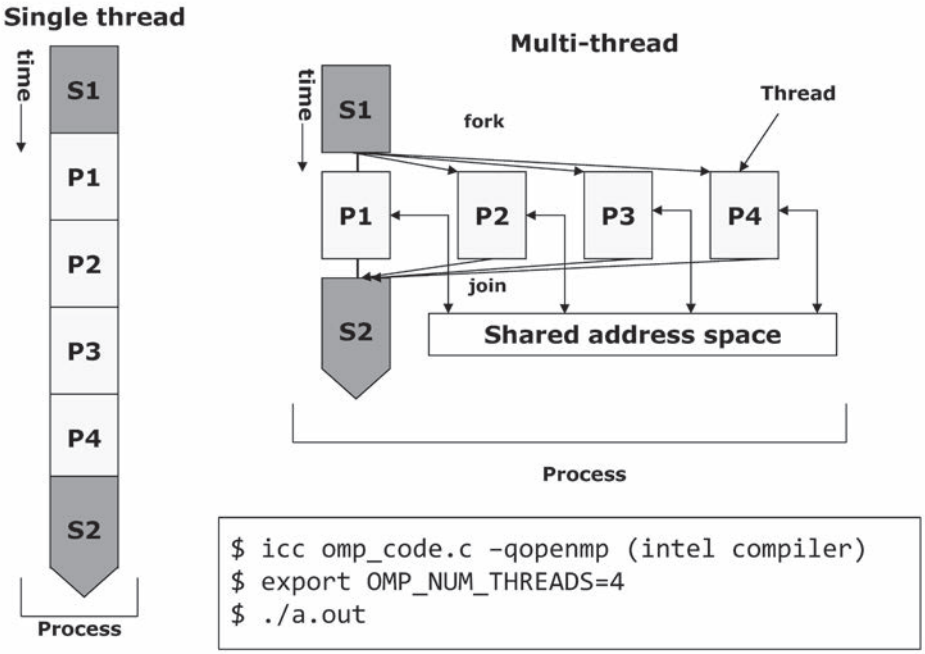
- Compiler Directives
  - 스레드 사이의 작업분담, 통신, 동기화를 담당
  - 좁은 의미의 OpenMP
    예) C$OMP PARALLEL DO
- Runtime Library
  - 병렬 매개변수(참여 스레드의 개수, 번호 등)의 설정과 조회
    예) CALL omp_set_num_threads(128)
- Environmental Variables
  - 실행 시스템의 병렬 매개변수(스레드 개수 등)를 정의
    예) export OMP_NUM_THREADS=8

OpenMP Programming Model

- 명시적 병렬화
  - 병렬 영역 지정
- 공유 메모리 프로그래밍 모델(스레드 기반)
  - 스레드-스레드 통신은 공유 변수 이용
- Fork-Join 모델

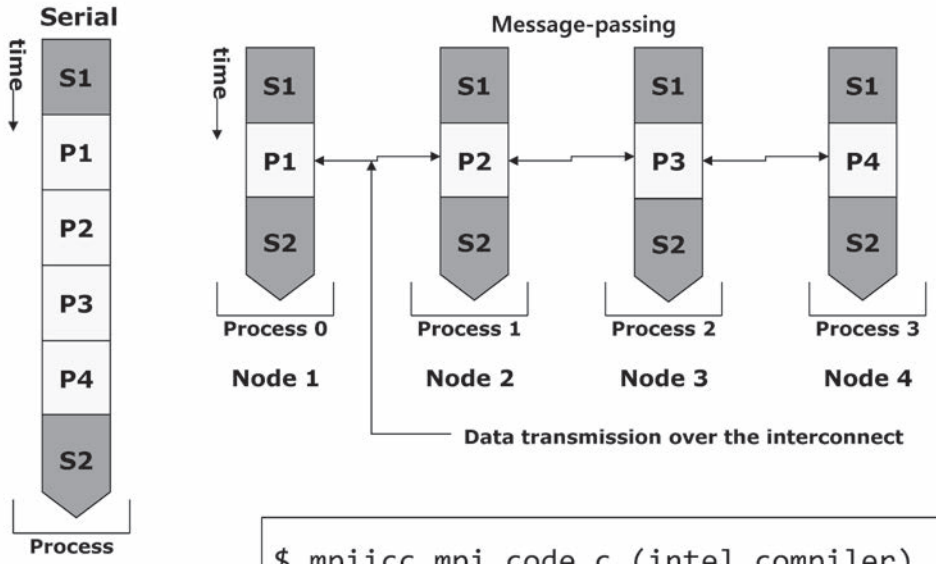


Shared Memory Programming Model




```
$ icc omp_code.c -qopenmp (intel compiler)
$ export OMP_NUM_THREADS=4
$ ./a.out
```

Message Passing Programming Model




```
$ mpiicc mpi_code.c (intel compiler)
$ mpirun -np 4 -hostfile mf ./a.out
```


OpenMP vs. MPI

	MPI	OpenMP*
Portable	Yes	Yes
Scalable	Yes	Partially
Performance Oriented	Yes	Yes
Supports Data Parallel	No	Yes
Incremental Parallelism	No	Yes
High Level	No	Yes
Serial Code Intact	No	Yes
★ Distributed Memory	Yes	No

Basic

- **Creating Threads**
- Data Environment
- Parallel Loops
- Synchronization
- Scheduling

Compile and Execution

- Header file include(C/C++ only)
  - #include <omp.h>
- Compile(+Link) Option
  - INTEL: -qopenmp, -openmp(old)
  - PGI: -mp
  - GCC: -fopenmp

gfortran –fopenmp ompsrc.f90 –o omp.exe
(or gcc –fopenmp ompsrc.c –o omp.exe)
[export OMP_NUM_THREADS=8]
./omp.exe

OpenMP Syntax

	Fortran (고정형식:f77)	Fortran (자유형식:f90)	C
지시어 시작 (감시문자)	!$OMP <Directive> C$OMP <Directive> *$OMP <Directive>	!$OMP <Directive>	#pragma omp <Directive>
줄 바꿈	!$OMP <Directive> !$OMP&...	!$OMP <Directive> & ...	#pragma omp ... ₩ ...
선택적 컴파일	!$ ... C$ ... *$ ...	!$ ...	#ifdef _OPENMP ... #endif
시작위치	첫번째 열	무관	무관

Creating Threads: Parallel Regions

➢ 병렬 영역 지정

Fortran	C

!$OMP PARALLEL
!$OMP END PARALLEL
	#pragma omp parallel
	{
	}

➢ 스레드 개수 설정
- 환경 변수 : **export OMP_NUM_THREADS = xxx**
- 실행시간 라이브러리 : **omp_set_num_threads(xxx)**
- 지시어 : #pragma omp parallel **num_threads(xxx)**

➢ 스레드와 관련된 Runtime libraries
- **omp_set_num_threads**(integer): 스레드 개수 설정
- **omp_get_num_threads**(): 현재 스레드 팀에서 스레드 개수 반환
- **omp_get_thread_num**(): 스레드 ID 반환

PBS Interactive 작업 제출

- 교육 중 실습은 **debug** 큐 이용
  - debug 큐에 작업 제출 또는 인터랙티브 작업 이용
- 인터랙티브 노드 작업: **qsub –I (in the "scratch" directory)**

```
$ cd /scratch/sedu##
$ qsub -I -V -A etc -l select=1:ncpus=16 -l walltime=04:00:00 -q debug
qsub: waiting for job 4997862.pbs to start
qsub: job 4997862.pbs ready

[sedu50@node8281 C]$ export OMP_NUM_THREADS=4
[sedu50@node8281 C]$ ./hello.x
Hello World
Hello World
Hello World
Hello World
----------------------------------------
[sedu01@node8281 C]$ exit
```


Creating Threads: Parallel Regions

Fortran

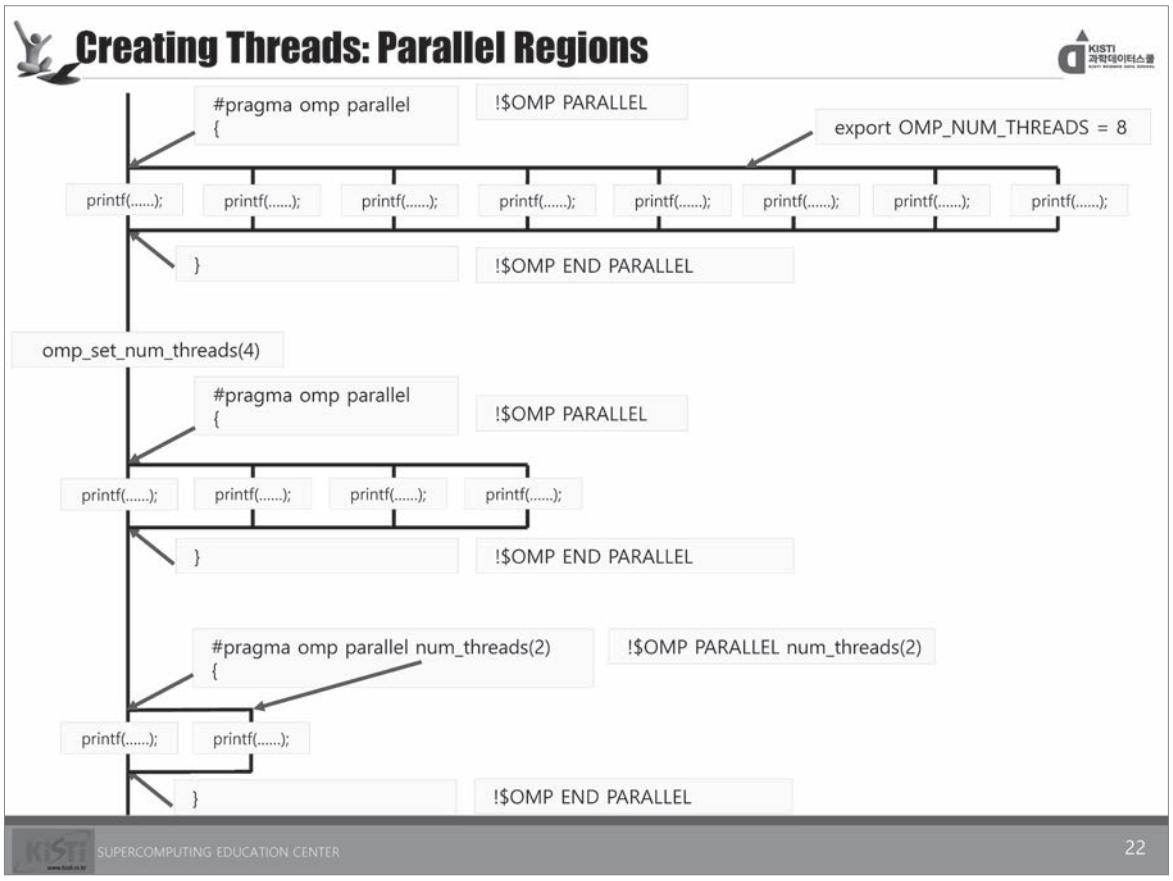
```
PROGRAM hello_world

  INTEGER omp_get_thread_num
!$OMP PARALLEL
  PRINT *, 'Hello World', omp_get_thread_num()
!$OMP END PARALLEL
  print *, ''
  call omp_set_num_threads(4)
!$OMP PARALLEL
  PRINT *, 'Hello World', omp_get_thread_num()
!$OMP END PARALLEL
  print*, ''
!$OMP PARALLEL num_threads(2)
  PRINT *, 'Hello World', omp_get_thread_num()
!$OMP END PARALLEL

END
```

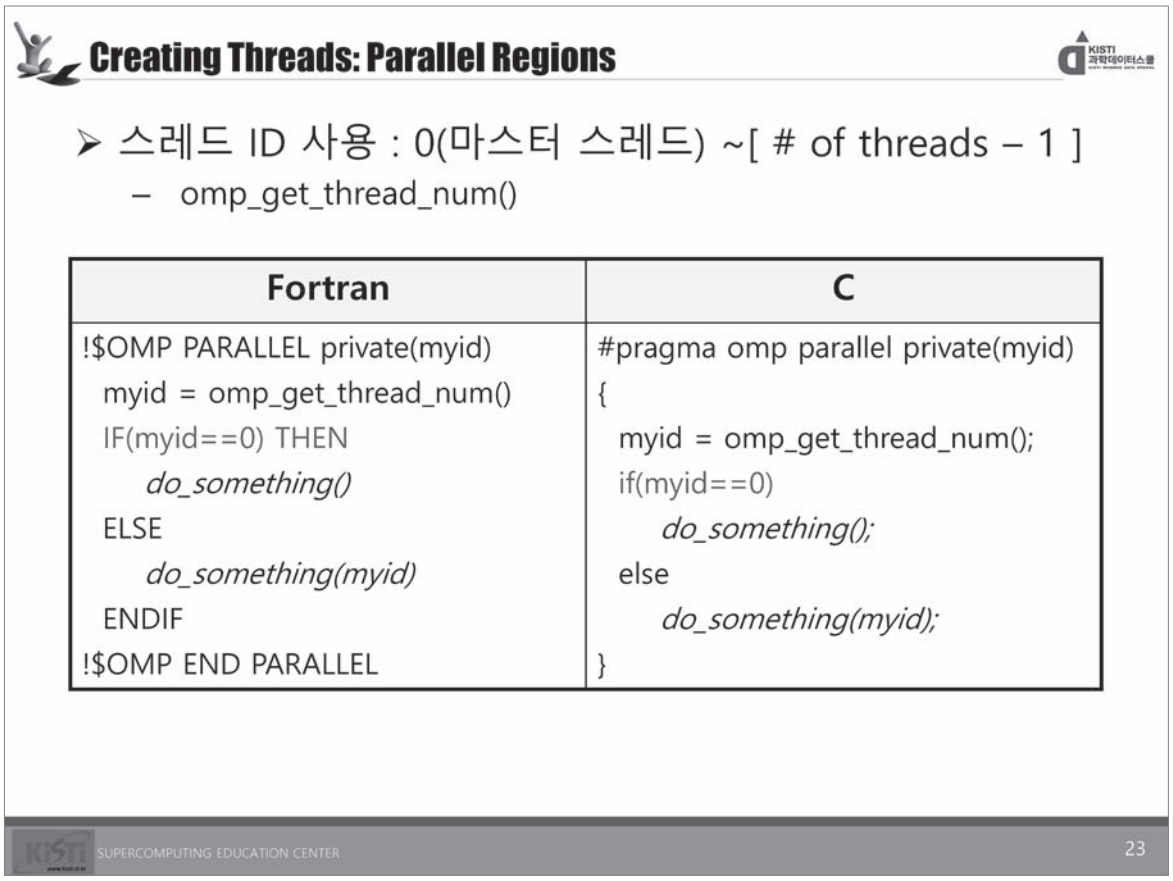
```
$ gfortran -fopenmp –o create_thread.x create_thread.f90
$ export OMP_NUM_THREADS = 8
$ ./create_thread.x
```

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
#pragma omp parallel
{
   printf ("Hello World %d₩n", omp_get_thread_num());
}
   printf("₩n");
   omp_set_num_threads(4);
#pragma omp parallel
{
   printf ("Hello World %d₩n", omp_get_thread_num());
}
   printf("₩n");
#pragma omp parallel num_threads(2)
{
   printf ("Hello World %d₩n", omp_get_thread_num());
}
}
```

```
$ gcc –fopenmp –o create_thread.x create_thread.c
$ export OMP_NUM_THREADS = 8
$ ./create_thread.x
```


Creating Threads: Parallel Regions
#pragma omp parallel
{
!$OMP PARALLEL
export OMP_NUM_THREADS = 8
printf(.....);
printf(.....);
printf(.....);
printf(.....);
printf(.....);
printf(.....);
printf(.....);
printf(.....);
}
!$OMP END PARALLEL
omp_set_num_threads(4)
#pragma omp parallel
{
!$OMP PARALLEL
printf(.....);
printf(.....);
printf(.....);
printf(.....);
}
!$OMP END PARALLEL
#pragma omp parallel num_threads(2)
{
!$OMP PARALLEL num_threads(2)
printf(.....);
printf(.....);
}
!$OMP END PARALLEL
SUPERCOMPUTING EDUCATION CENTER
22


Creating Threads: Parallel Regions
➢ 스레드 ID 사용 : 0(마스터 스레드) ~[# of threads – 1]
– omp_get_thread_num()
Fortran
C
!$OMP PARALLEL private(myid)
myid = omp_get_thread_num()
IF(myid==0) THEN
do_something()
ELSE
do_something(myid)
ENDIF
!$OMP END PARALLEL
#pragma omp parallel private(myid)
{
myid = omp_get_thread_num();
if(myid==0)
do_something();
else
do_something(myid);
}
SUPERCOMPUTING EDUCATION CENTER
23

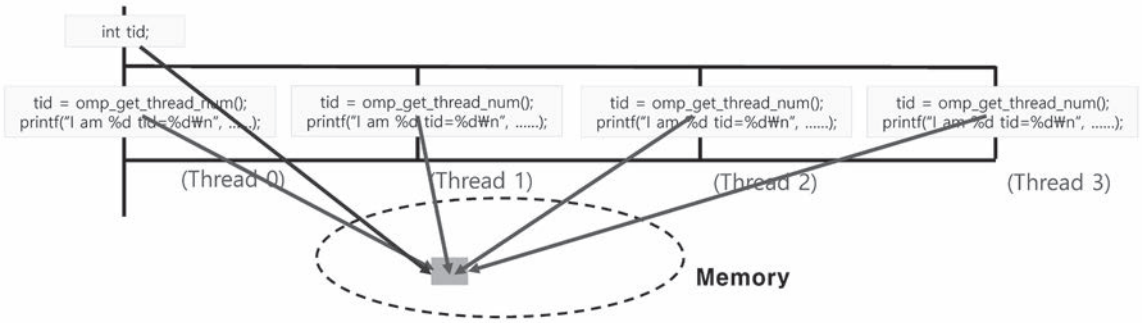
Basic

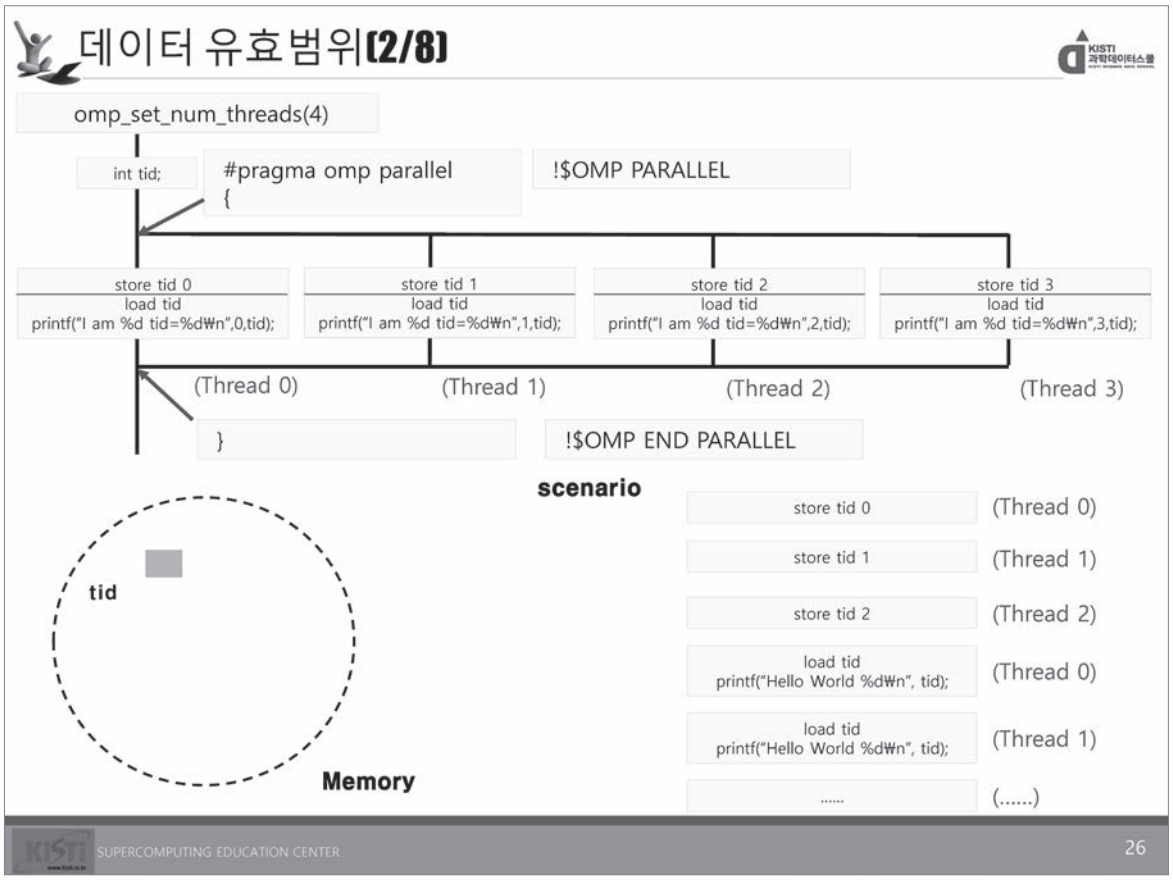
- Creating Threads
- **Data Environment**
- Parallel Loops
- Synchronization
- Scheduling

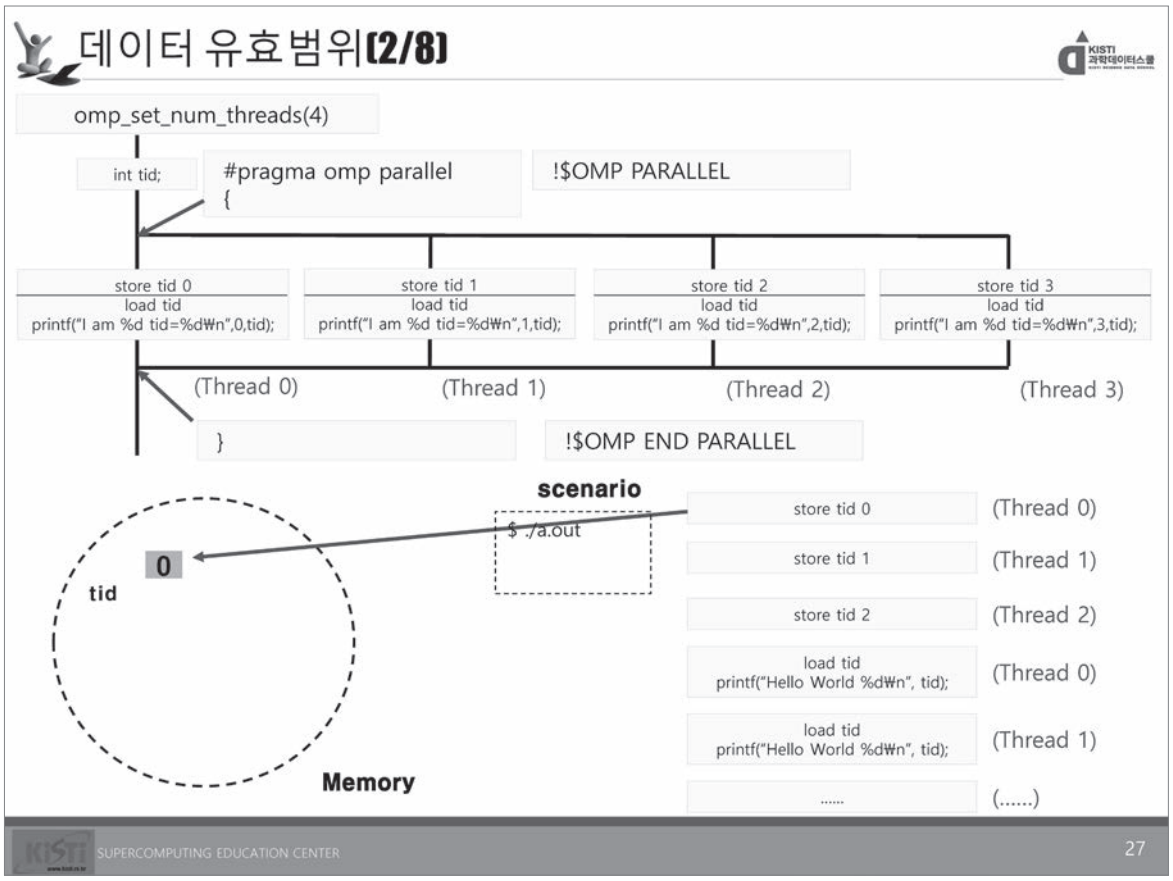
데이터 유효범위(1/8)

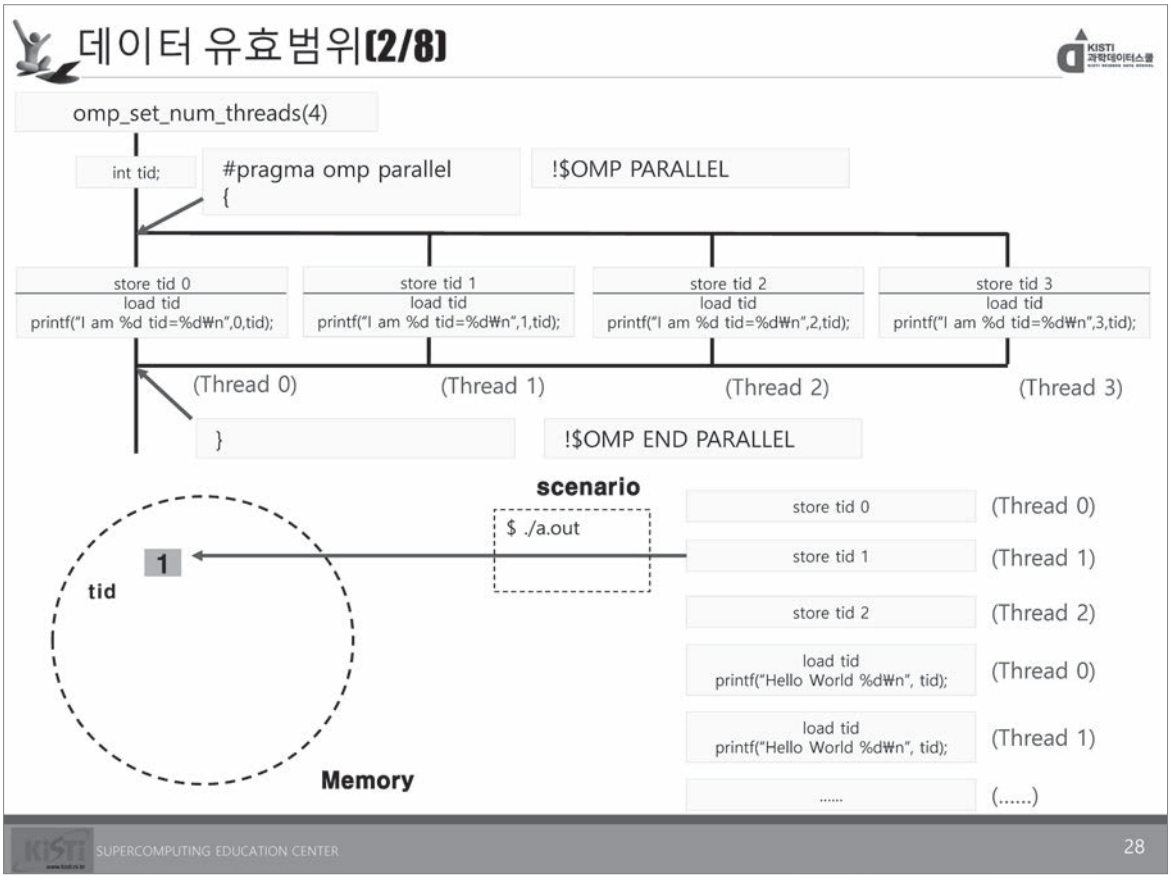
Fortran(data_scope_wrong)	C(data_scope_wrong)

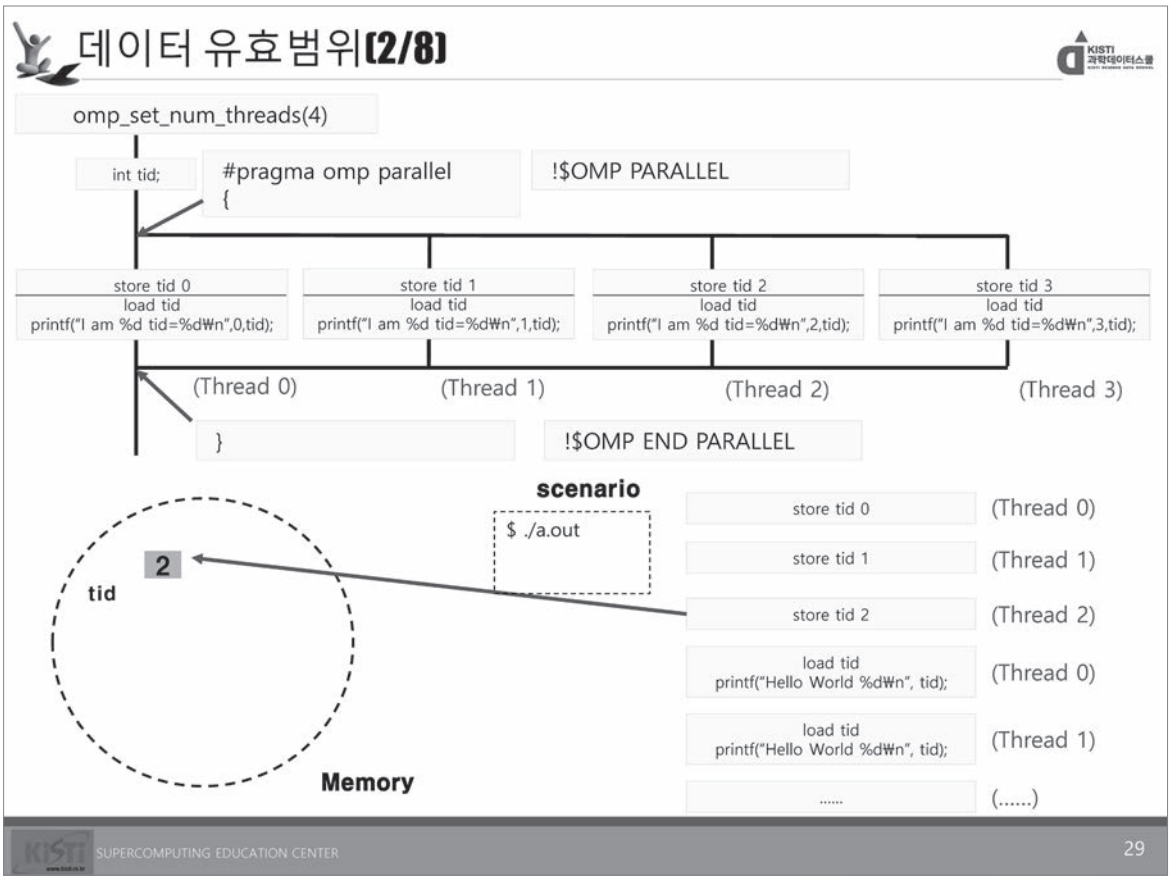
PROGRAM hello_wrong
 	
INTEGER a, tid, omp_get_thread_num
call **omp_set_num_threads(4)**
!$OMP PARALLEL
tid = omp_get_thread_num()
PRINT *, ' I am ', omp_get_thread_num(), ' tid = ', tid
!$OMP END PARALLEL
 	
END
	#include <stdio.h>
	#include <omp.h>
	int main()
	{
	int tid;
	omp_set_num_threads(4);
	#pragma omp parallel
	{
	tid = omp_get_thread_num();
	printf ("I am %d tid = %d₩n", omp_get_thread_num(), tid);
	}
	}

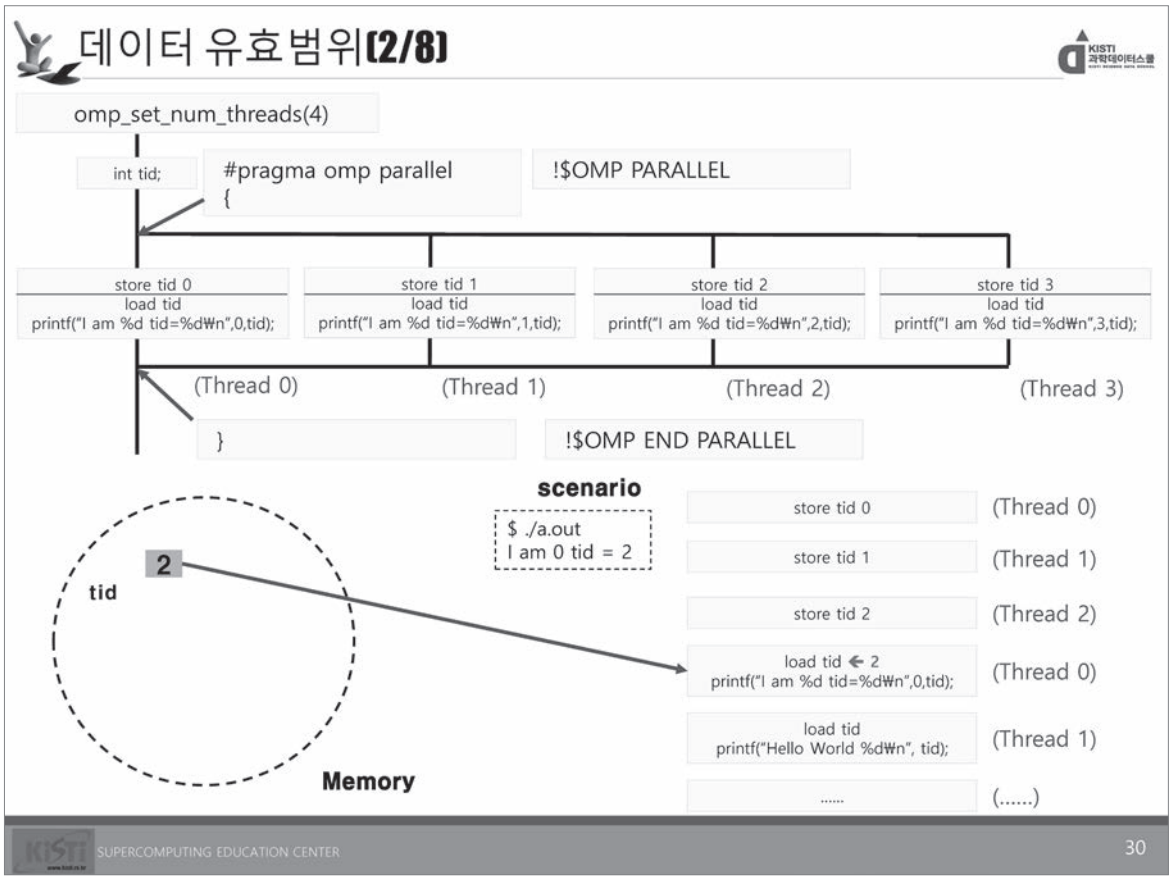


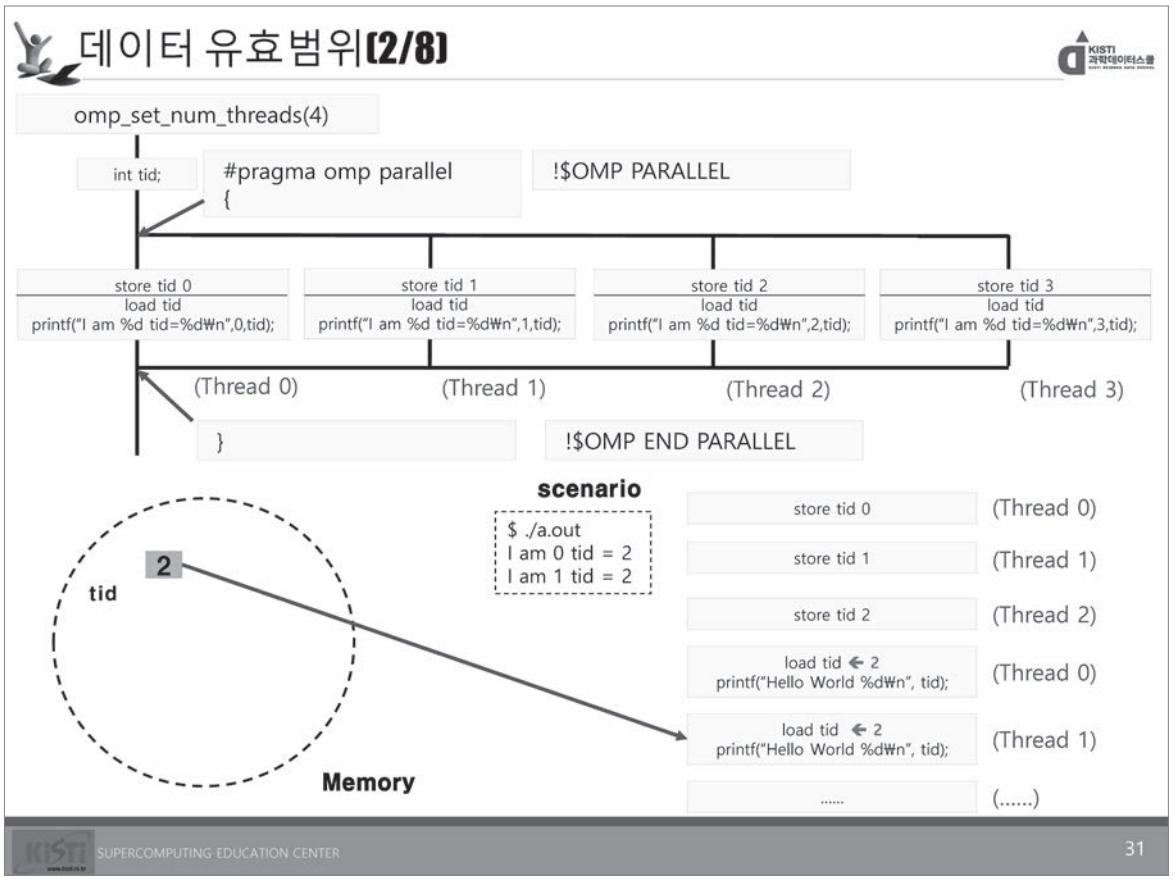

데이터 유효범위(2/8)
omp_set_num_threads(4)
int tid;
#pragma omp parallel
{
!$OMP PARALLEL
store tid 0
load tid
printf("I am %d tid=%d₩n",0,tid);
store tid 1
load tid
printf("I am %d tid=%d₩n",1,tid);
store tid 2
load tid
printf("I am %d tid=%d₩n",2,tid);
store tid 3
load tid
printf("I am %d tid=%d₩n",3,tid);
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
}
!$OMP END PARALLEL
scenario
store tid 0
(Thread 0)
store tid 1
(Thread 1)
store tid 2
(Thread 2)
load tid
printf("Hello World %d₩n", tid);
(Thread 0)
load tid
printf("Hello World %d₩n", tid);
(Thread 1)
.....
(.....)
tid
Memory
SUPERCOMPUTING EDUCATION CENTER
26


데이터 유효범위(2/8)
omp_set_num_threads(4)
int tid;
#pragma omp parallel
{
!$OMP PARALLEL
store tid 0
load tid
printf("I am %d tid=%d₩n",0,tid);
store tid 1
load tid
printf("I am %d tid=%d₩n",1,tid);
store tid 2
load tid
printf("I am %d tid=%d₩n",2,tid);
store tid 3
load tid
printf("I am %d tid=%d₩n",3,tid);
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
}
!$OMP END PARALLEL
scenario
$ ./a.out
store tid 0
(Thread 0)
store tid 1
(Thread 1)
store tid 2
(Thread 2)
load tid
printf("Hello World %d₩n", tid);
(Thread 0)
load tid
printf("Hello World %d₩n", tid);
(Thread 1)
.....
(.....)
0
tid
Memory
SUPERCOMPUTING EDUCATION CENTER
27


데이터 유효범위(2/8)
omp_set_num_threads(4)
int tid;
#pragma omp parallel
{
!$OMP PARALLEL
store tid 0
load tid
printf("I am %d tid=%d₩n",0,tid);
store tid 1
load tid
printf("I am %d tid=%d₩n",1,tid);
store tid 2
load tid
printf("I am %d tid=%d₩n",2,tid);
store tid 3
load tid
printf("I am %d tid=%d₩n",3,tid);
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
}
!$OMP END PARALLEL
scenario
$ ./a.out
1
tid
Memory
store tid 0
(Thread 0)
store tid 1
(Thread 1)
store tid 2
(Thread 2)
load tid
printf("Hello World %d₩n", tid);
(Thread 0)
load tid
printf("Hello World %d₩n", tid);
(Thread 1)
.....
(.....)
SUPERCOMPUTING EDUCATION CENTER
28


데이터 유효범위(2/8)
omp_set_num_threads(4)
int tid;
#pragma omp parallel
{
!$OMP PARALLEL
store tid 0
load tid
printf("I am %d tid=%d₩n",0,tid);
store tid 1
load tid
printf("I am %d tid=%d₩n",1,tid);
store tid 2
load tid
printf("I am %d tid=%d₩n",2,tid);
store tid 3
load tid
printf("I am %d tid=%d₩n",3,tid);
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
}
!$OMP END PARALLEL
scenario
$ ./a.out
2
tid
Memory
store tid 0
(Thread 0)
store tid 1
(Thread 1)
store tid 2
(Thread 2)
load tid
printf("Hello World %d₩n", tid);
(Thread 0)
load tid
printf("Hello World %d₩n", tid);
(Thread 1)
.....
(.....)
SUPERCOMPUTING EDUCATION CENTER
29


데이터 유효범위(2/8)
omp_set_num_threads(4)
int tid;
#pragma omp parallel
{
!$OMP PARALLEL
store tid 0
load tid
printf("I am %d tid=%d₩n",0,tid);
store tid 1
load tid
printf("I am %d tid=%d₩n",1,tid);
store tid 2
load tid
printf("I am %d tid=%d₩n",2,tid);
store tid 3
load tid
printf("I am %d tid=%d₩n",3,tid);
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
}
!$OMP END PARALLEL
scenario
$ ./a.out
I am 0 tid = 2
tid
2
Memory
store tid 0
(Thread 0)
store tid 1
(Thread 1)
store tid 2
(Thread 2)
load tid ← 2
printf("I am %d tid=%d₩n",0,tid);
(Thread 0)
load tid
printf("Hello World %d₩n", tid);
(Thread 1)
.....
(.....)
SUPERCOMPUTING EDUCATION CENTER
30


데이터 유효범위(2/8)
omp_set_num_threads(4)
int tid;
#pragma omp parallel
{
!$OMP PARALLEL
store tid 0
load tid
printf("I am %d tid=%d₩n",0,tid);
store tid 1
load tid
printf("I am %d tid=%d₩n",1,tid);
store tid 2
load tid
printf("I am %d tid=%d₩n",2,tid);
store tid 3
load tid
printf("I am %d tid=%d₩n",3,tid);
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
}
!$OMP END PARALLEL
scenario
$ ./a.out
I am 0 tid = 2
I am 1 tid = 2
tid
2
Memory
store tid 0
(Thread 0)
store tid 1
(Thread 1)
store tid 2
(Thread 2)
load tid ← 2
printf("Hello World %d₩n", tid);
(Thread 0)
load tid ← 2
printf("Hello World %d₩n", tid);
(Thread 1)
.....
(.....)
SUPERCOMPUTING EDUCATION CENTER
31

데이터 유효범위(3/8)

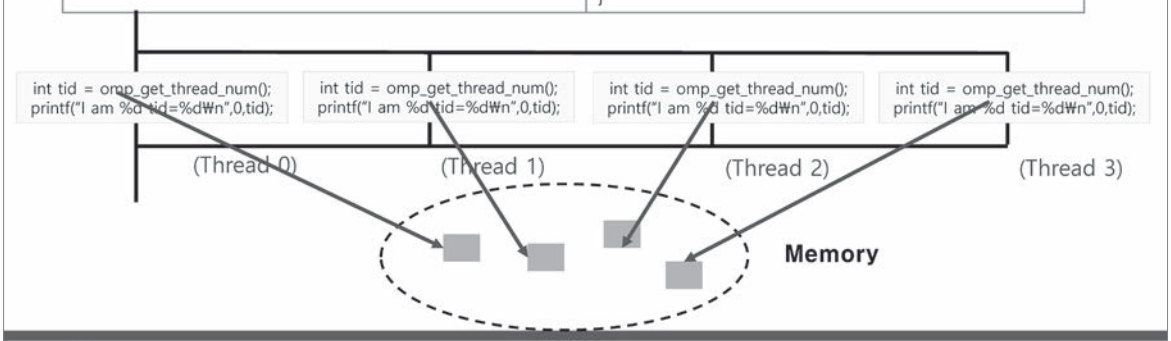
Fortran

```
PROGRAM hello_wrong
  INTEGER omp_get_thread_num
  call omp_set_num_threads(4)
!$OMP PARALLEL
    INTEGER tid
    tid = OMP_GET_THREAD_NUM()
    PRINT *, 'I am',    &
    OMP_GET_THREAD_NUM(), &
          'TID =' tid
!$OMP END PARALLEL
END
```

Not allowed

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
   omp_set_num_threads(4);
#pragma omp parallel
{
   int tid = omp_get_thread_num();
   printf ("I am %d tid = %d₩n",
            omp_get_thread_num(), tid);
}
}
```



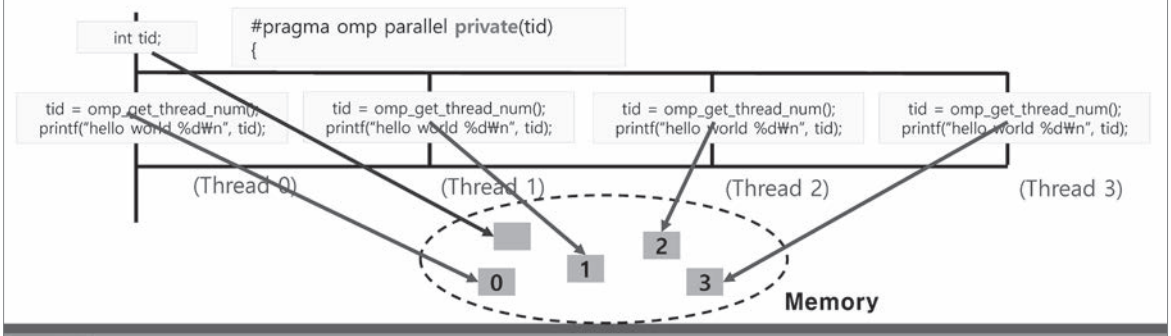
데이터 유효범위(4/8)

Fortran(data_scope_private)

```
PROGRAM hello_right

INTEGER tid, omp_get_thread_num
call omp_set_num_threads(4)
!$OMP PARALLEL PRIVATE(tid)
    tid = OMP_GET_THREAD_NUM()
    PRINT *, 'I am ', OMP_GET_THREAD_NUM(), ' tid = ' tid
!$OMP END PARALLEL

END
```

C(data_scope_private)

```
#include <stdio.h>
#include <omp.h>
int main()
{
  int tid;
  omp_set_num_threads(4);
#pragma omp parallel private(tid)
{
   tid = omp_get_thread_num();
   printf ("I am %d tid = %d₩n", omp_get_thread_num(), tid);
}
}
```

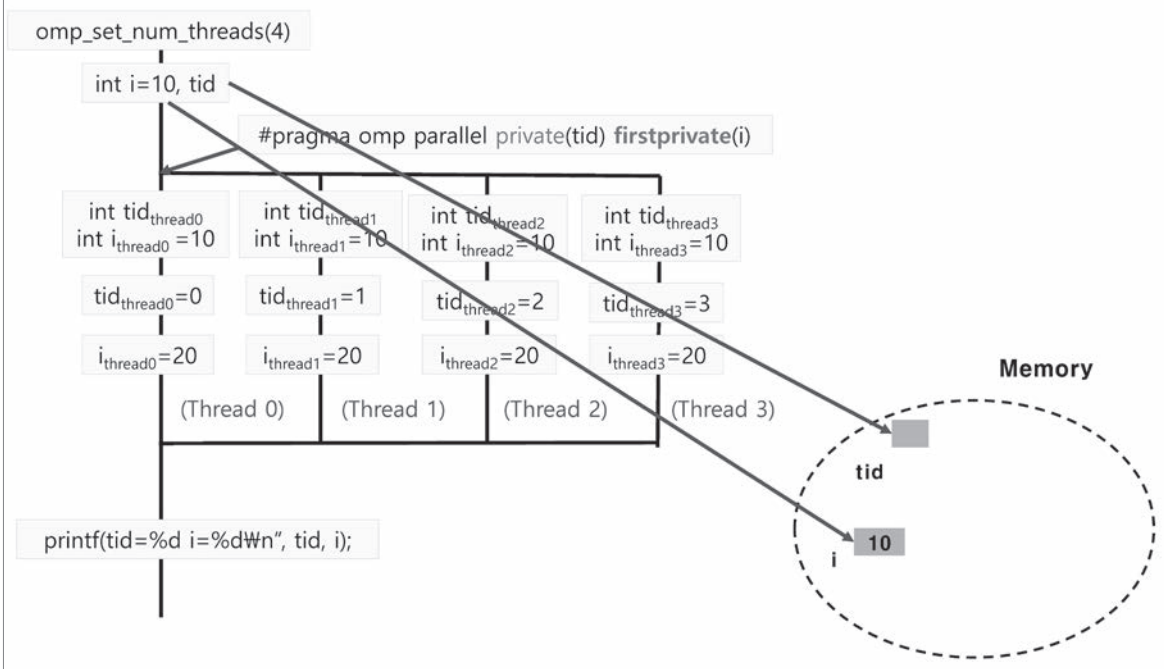


데이터 유효범위(5/8)

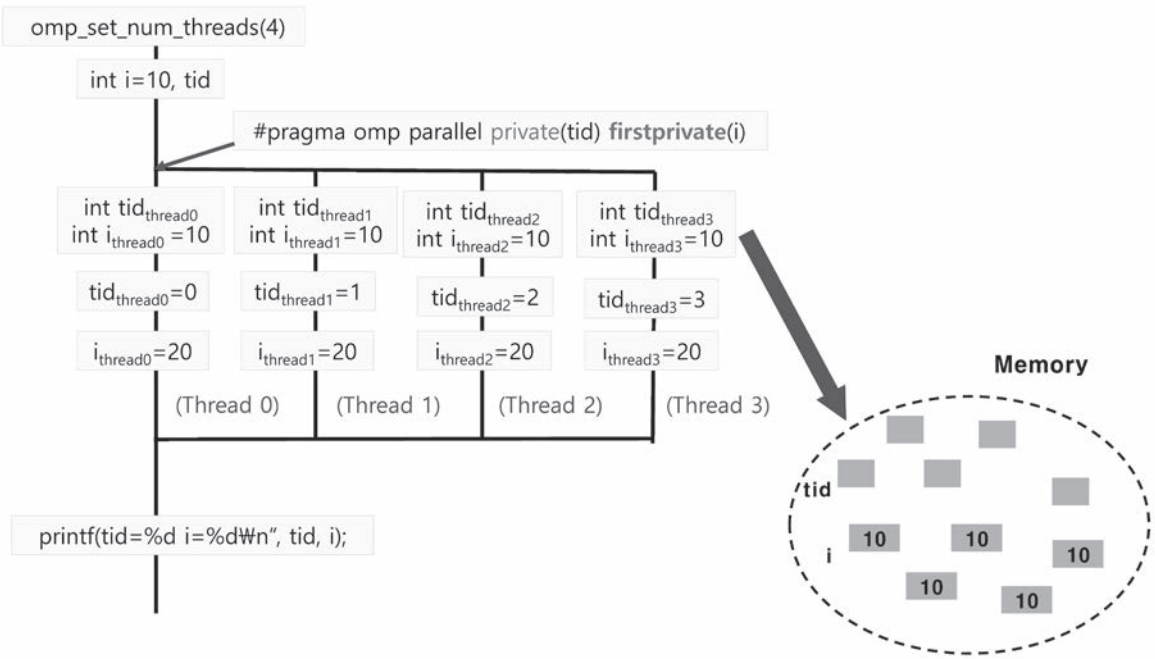
Fortran	C

PROGRAM data_scope_firstprivate
 	
INTEGER :: i=10, tid, omp_get_thread_num
 	
call omp_set_num_threads(4)
!$OMP PARALLEL PRIVATE(tid) **FIRSTPRIVATE(i)**
tid = OMP_GET_THREAD_NUM()
PRINT *, 'tid =' , tid, 'i =', i
i = 20
!$OMP END PARALLEL
print *, 'tid =', tid, 'i =', i
 	
END
	#include <stdio.h>
	#include <omp.h>
	
	int main()
	{
	int i = 10, tid;
	omp_set_num_threads(4);
	#pragma omp parallel private(tid) **firstprivate(i)**
	{
	tid = omp_get_thread_num();
	printf ("tid = %d i = %d₩n", tid, i);
	i = 20;
	}
	printf("tid = %d i=%d₩n", tid, i);
	}

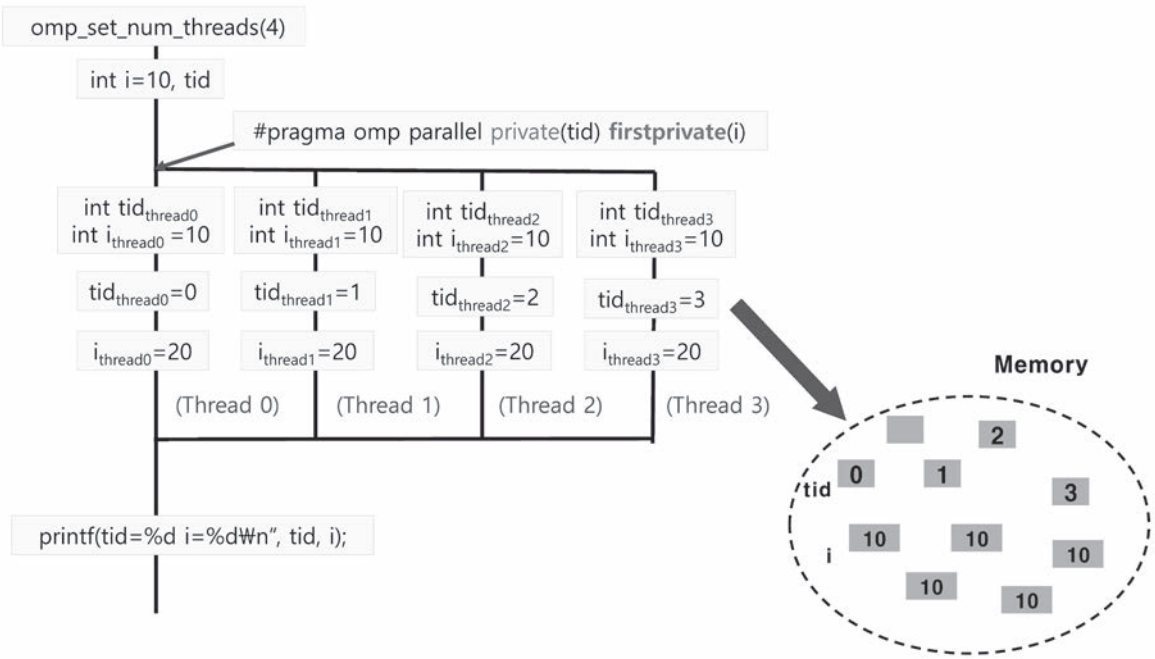
데이터 유효범위(6/8)



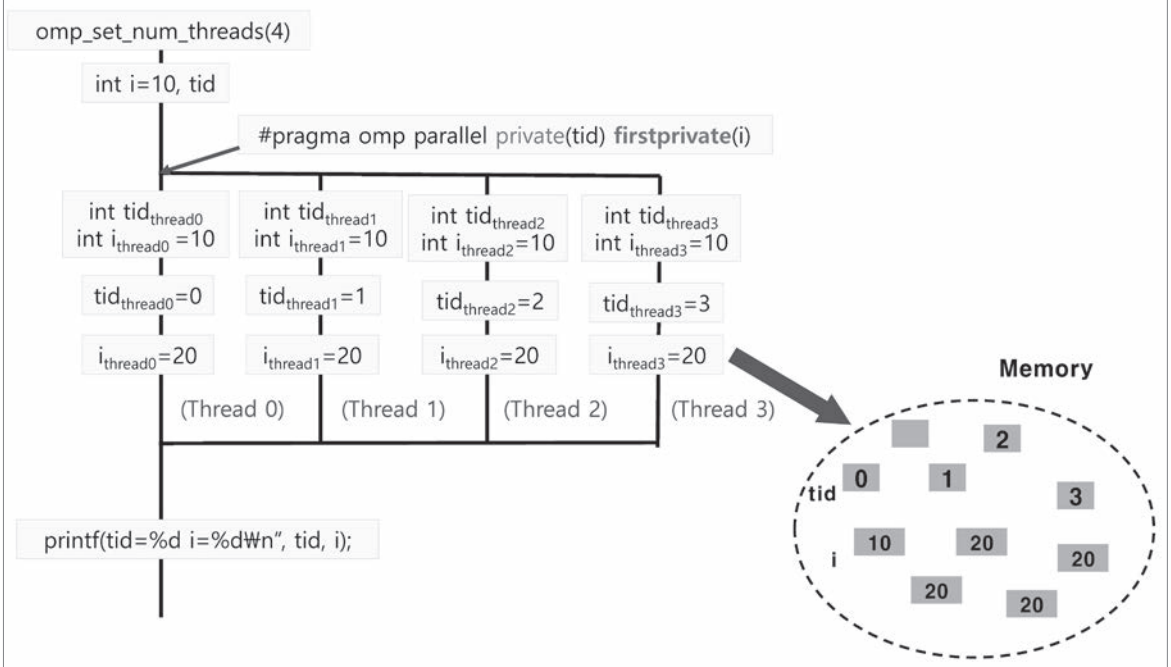
데이터 유효범위(6/8)



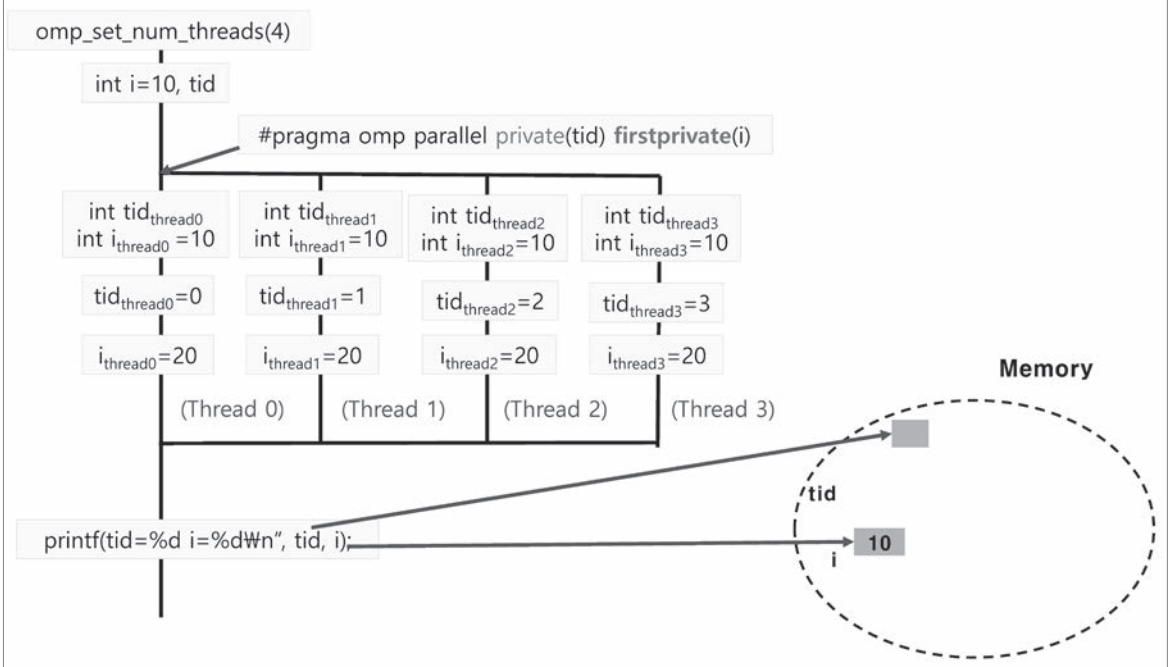
데이터 유효범위(6/8)



데이터 유효범위(6/8)



데이터 유효범위(6/8)



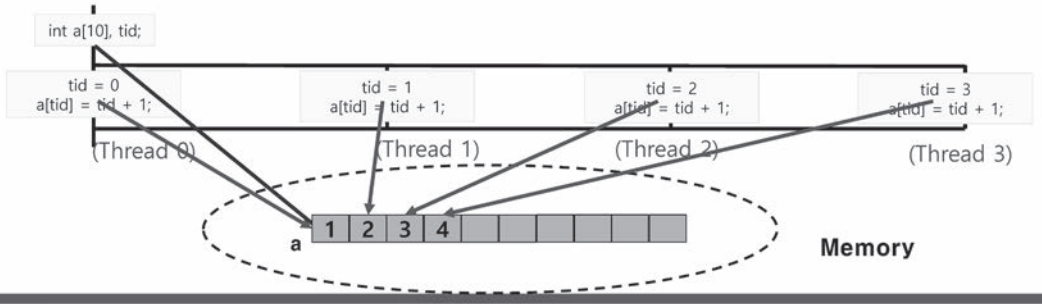
데이터 유효범위(7/8)

Fortran

```
PROGRAM data_scope_shared
    INTEGER a(0:9), tid, i, omp_get_thread_num

    call omp_set_num_threads(4)
!$OMP PARALLEL shared(a) PRIVATE(tid)
    tid = OMP_GET_THREAD_NUM()
    a(tid) = tid + 1
!$OMP END PARALLEL

    do i=0, 3
       print *, 'a(', i, ') =', a(i)
    end do
END
```

C

```
#include <stdio.h>
#include <omp.h>
int main()
{
  int a[10], tid, i;
  omp_set_num_threads(4);
#pragma omp parallel shared(a) private(tid)
{
   tid = omp_get_thread_num();
   a[tid] = tid + 1;
}
   for(i=0; i<4; i++)
      printf("a[%d] = %d₩n", i, a[i]);
}
```



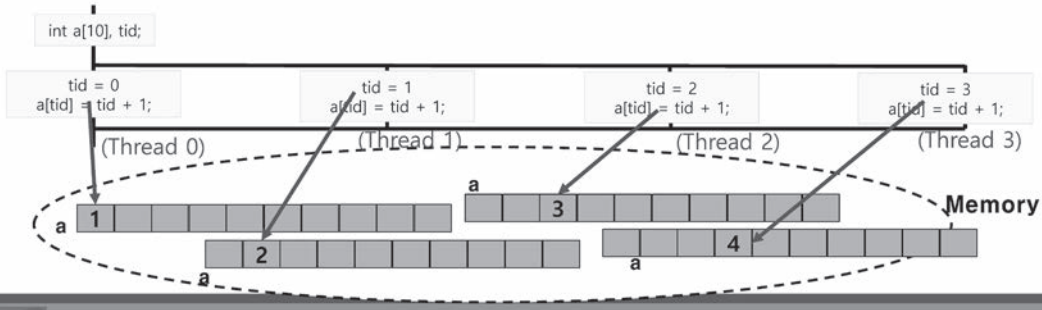
데이터 유효범위(8/8)

Fortran

```
PROGRAM data_scope_private_array
    INTEGER a(0:9), tid, i, omp_get_thread_num

    call omp_set_num_threads(4)
!$OMP PARALLEL private(a) PRIVATE(tid)
    tid = OMP_GET_THREAD_NUM()
    a(tid) = tid + 1
!$OMP END PARALLEL

    do i=0, 3
       print *, 'a(', i, ') =', a(i)
    end do
END
```

C

```
#include <stdio.h>
#include <omp.h>
int main()
{
  int a[10], tid, i;
  omp_set_num_threads(4);
#pragma omp parallel private(a) private(tid)
{
   tid = omp_get_thread_num();
   a[tid] = tid + 1;
}
   for(i=0; i<4; i++)
      printf("a[%d] = %d₩n", i, a[i]);
}
```



Data Environment: Default

- 유효범위가 지정되지 않은 변수는 기본적으로 **shared**
  - 전역변수: SAVE(STATIC in C), COMMON Block
    - MODULE 변수, 선언하면서 초기화 되는 변수(integer n=10)
  - 동적 할당 변수는 shared

- **Default Private**
  - 루프 인덱스
    - Fortran: 병렬영역의 정적 범위에서 순차 루프 인덱스도 private
    - C/C++: 정적 범위에서 순차루프 인덱스는 기본 shared!
  - 병렬영역에서 호출되는 서브루틴의 지역 변수 (but, save (static) 변수는 shared)
  - C/C++:
    - value parameter
    - 병렬영역의 정적 범위에서 선언된 자동 변수

Data Environment: Default

```
SUBROUTINE CALLER(A,N)

INTEGER N, A(N), I, J, M

M=3
!$OMP PARALLEL

DO I=1, N
   DO J=1,5
     CALL CALLEE(A(I),M,J)
   ENDDO
ENDDO

END
```

```
SUBROUTINE CALLEE(X,Y,Z)

COMMON /COM/C
INTEGER X,Y,Z,C,II,CNT
SAVE CNT

CNT=CNT+1
DO II = 1, Z
    X=Y+C
ENDDO

END
```

Data Environment: Default

변수	유효범위	설 명
A	shared	병렬영역 밖에서 선언
N	shared	병렬영역 밖에서 선언
I	private	병렬루프 인덱스
J	private	순차루프 인덱스 (Fortran)
M	shared	병렬영역 밖에서 선언
X	shared	실제 인수 A가 shared
Y	shared	실제 인수 M이 shared
Z	private	실제 인수 J가 private
C	shared	Common block으로 선언
II	private	호출된 서브루틴의 지역 변수
CNT	shared	Save 속성을 가지는 지역 변수

clause : private, shared, default

```
!$OMP PARALLEL shared(a) private(myid, x)
    myid = OMP_GET_THREAD_NUM()
    x = work(myid)
    IF (x<1.0) THEN
      a(myid) = x
    ENDIF
!$OMP END PARALLEL
```

```
!$OMP PARALLEL default(private) shared(a)
    myid = OMP_GET_THREAD_NUM()
    x = work[myid]
    IF (x<1.0) THEN
      a[myid] = x
    ENDIF
!$OMP END PARALLEL
```

Basic

- Creating Threads
- Data Environment
- **Parallel Loops**
- Synchronization
- Scheduling

Example: PI

$$\int_0^1 \frac{1}{1+x^2}\,dx = \arctan(x)\,\Big|_0^1 = \arctan(1) - \arctan(0) = \arctan(1) = \frac{\pi}{4}$$

Fortran	C

PARAMETER (NUM_STEPS=1000000)
SUM = 0.0
STEP = 1.0/REAL(NUM_STEPS)
DO I = 0, NUM_STEPS-1
  X=(I+0.5)*STEP
  SUM = SUM + 4.0/(1.0+X*X)
ENDDO
PI=STEP*SUM
	static long num_steps = 1000000;
	double step;
	void main ()
	{
	  int i;
	  double x, pi, sum = 0.0;
	  step = 1.0/(double) num_steps;
	  for (i=1;i<= num_steps; i++){
	  x = (i+0.5)*step;
	  sum = sum + 4.0/(1.0+x*x);
	  }
	  pi = step * sum;
	}

Parallel: Fortran

```
   call omp_set_num_threads(4)
!$OMP PARALLEL PRIVATE(TID,istat,iend,X)
   TID = OMP_GET_THREAD_NUM()
   nthreads = omp_get_num_threads()

   istat = tid * num_steps / nthreads
   iend = (tid+1) * num_steps / nthreads – 1

   SUM(TID)=0.0
   DO I = istat, iend
      X = (I+0.5)*STEP
      SUM(TID) = SUM(TID)+ 4.0/(1.0+X*X)
   ENDDO
!$OMP END PARALLEL
   PI=0.0
   DO I=0, NTHREADS-1
      PI = PI + SUM(I)*STEP
   ENDDO
```

```
gfortran pi_serial.f90 –o spi.x
time ./spi.x
gfortran -fopenmp pi_omp_exp.f90 -o ppi1.x
time ./ppi1.x
```


Parallel: C

```
omp_set_num_threads(4);
#pragma omp parallel private(tid,istat,iend,x)
{
     tid = omp_get_thread_num();
     nthreads = omp_get_num_threads();

     istat= tid * num_steps/ nthreads;
     iend= (tid+1) * num_steps / nthreads-1;

     sum[tid]=0.0;
     for (i=istat; i<iend; i++){
       x = (i+0.5)*step;
       sum[tid] += 4.0/(1.0+x*x);
     }
}
for(i=0, pi=0.0;i<NTHREADS;i++)
   pi += sum[i] * step;
```

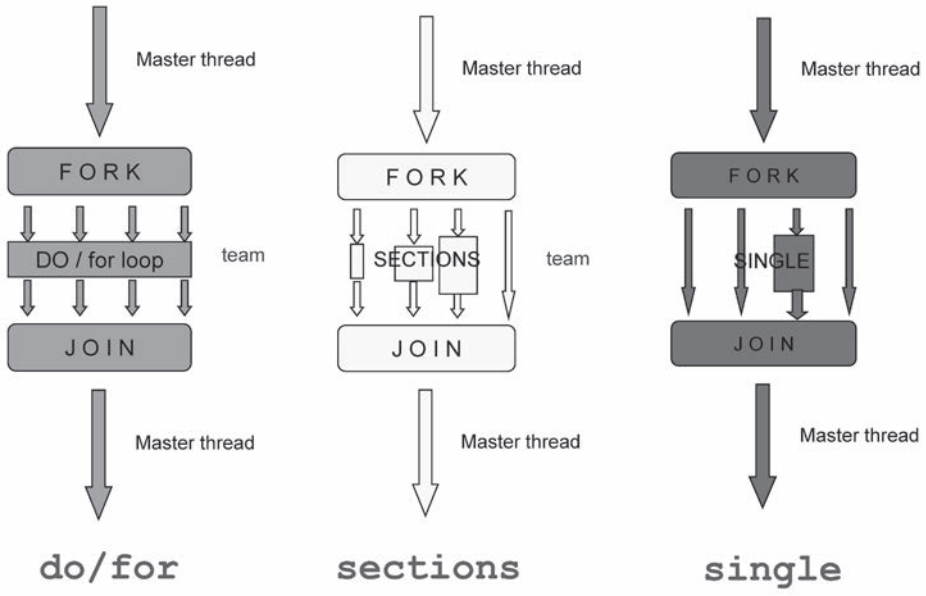
```
gcc pi_serial.c –o spi.x
time ./spi.x
gcc -fopenmp pi_omp_exp.c -o ppi1.x
time ./ppi1.x
```

Worksharing Construct

- 병렬영역 내부에 삽입하여 작업할당에 이용
- 새로운 스레드 생성 없이 기존 스레드에 관련작업 실행 분배
- 구문의 시작점 : 스레드 사이의 동기화 없음
  - 먼저 접근하는 스레드에 우선적 작업 할당
- 구문의 끝점 : 동기화(암시적 장벽)
  - 작업이 먼저 끝나도 다른 작업의 완료까지 대기
  - 대기 없이 다른 작업을 실행하려면 nowait clause 사용

- **do/for**
- **sections**
- **single**
- **task (OpenMP 3.0)**

Worksharing Construct



Worksharing : DO/for

- 바로 뒤에 오는 루프의 반복실행을 스레드에 분배
- 동기화 : 루프 끝에 암시적 장벽
  - 기다리지 않으려면 nowait clause 사용

Fortran	C

!$OMP PARALLEL shared(a,b) &
private(j)
!$OMP DO
DO j = 1, N
a(j) = a(j) + b(j)
ENDDO
[!$OMP END DO [nowait]]
!$OMP END PARALLEL
	#pragma omp parallel \
	shared(a,b) private(j)
	{
	#pragma omp for
	for (j=0;j<N; j++)
	a[j] = a[j] + b[j];
	}

DO/for: Fortran

```
!$OMP PARALLEL PRIVATE(TID)
      TID = OMP_GET_THREAD_NUM()
      NTHREADS = OMP_GET_NUM_THREADS()
      SUM(TID)=0.0
      !$OMP DO PRIVATE(X)
      DO I = 0, NUM_STEPS-1
         X = (I+0.5)*STEP
         SUM(TID) = SUM(TID) + 4.0/(1.0+X*X)
      ENDDO
      !$OMP END DO
!$OMP END PARALLEL
      PI=0.0
      DO I=0, NTHREADS-1
         PI = PI + SUM(I)*STEP
      ENDDO
```

```
gfortran -fopenmp pi_omp_do.f90 -o ppi2.x
time ./ppi2.x
```

DO/for: C

```
#pragma omp parallel private(tid)
{
     tid = omp_get_thread_num();
     nthreads = omp_get_num_threads();

     sum[tid] = 0.0;
    #pragma omp for private(x)
     for (i=0; i<num_steps; i++){
          x = (i+0.5)*step;
          sum[tid] += 4.0/(1.0+x*x);
     }
}
For(i=0, pi=0.0; i<nthreads; i++) pi += sum[i] * step;
```

```
gcc -fopenmp pi_omp_for.c -o ppi2.x
time ./ppi2.x
```


Worksharing : sections

- 독립된 여러 개 작업을 각 스레드에 분산 실행
- Functional decomposition
- 동기화 : sections 구문 끝에 암시적 장벽
  - 기다리지 않으려면 nowait clause 사용

Fortran	C
!$OMP PARALLEL !$OMP SECTIONS !$OMP SECTION *CALL computeXpart()* !$OMP SECTION *CALL computeYpart()* !$OMP END SECTIONS [nowait] !$OMP END PARALLEL	#pragma omp parallel { #pragma omp sections [nowait] { #pragma omp section *computeXpart();* #pragma omp section *computeYpart();* }}

sections : Fortran

```
!$OMP PARALLEL PRIVATE(ID)
    ID = OMP_GET_THREAD_NUM()
    SUM(ID)=0.0
!$OMP SECTIONS PRIVATE(X)
!$OMP SECTION
    DO I = 0, (NUM_STEPS-1)/2
       X = (I+.5)*STEP
       SUM(ID)= SUM(ID) + 4.0/(1.0+X*X)
    ENDDO
!$OMP SECTION
    DO I = (NUM_STEPS-1)/2+1, NUM_STEPS-1
       X = (I+.5)*STEP
       SUM(ID)= SUM(ID) + 4.0/(1.0+X*X)
    ENDDO
!$OMP END SECTIONS
```

```
gfortran -fopenmp pi_omp_sec.c -o ppi3.x
time ./ppi3.x
```


sections: C

```
#pragma omp parallel private(id)
{
    id = omp_get_thread_num();
    sum[id]=0.0;
#pragma omp sections private(x)
{
#pragma omp section
    for (i=0; i< num_steps/2; i++){
        x = (i+0.5)*step;
        sum[id] += 4.0/(1.0+x*x);
    }
#pragma omp section
    for (i= num_steps/2+1; i< num_steps; i++){
        x = (i+0.5)*step;
        sum[id] += 4.0/(1.0+x*x);
    }
}
}
```

```
gcc -fopenmp pi_omp_sec.c -o ppi3.x
time ./ppi3.x
```

Combined Worksharing

- 병렬영역 안에 **DO/for** 또는 **sections** 구문을 하나만 가지는 경우
- **nowait**을 제외한 모든 **clause** 동일하게 사용 가능
- 동기화 : 병렬구문 마지막에 암시적 장벽 있음

Fortran	
!$OMP PARALLEL !$OMP DO	!$OMP PARALLEL DO
!$OMP PARALLEL !$OMP SECTIONS	!$OMP PARALLEL SECTIONS

C	
#pragma omp parallel #pragma omp for	#pragma omp parallel for
#pragma omp parallel #pragma omp sections	#pragma omp parallel sections

Basic

- Creating Threads
- Data Environment
- Parallel Loops
- **Synchronization**
- Scheduling

Synchronization

- High Level
  - **nowait**
  - **critical**
  - **atomic**
  - barrier
  - ordered
- Low Level
  - flush
  - locks (simple and nested)

동기화 : nowait

Fortran

```
PROGRAM nowait

  INTEGER :: x=1

  call omp_set_num_threads(16)
!$OMP PARALLEL
  !$OMP SINGLE
  x = x+1
  PRINT *, 'x =', x
  !$OMP END SINGLE

  !$OMP SINGLE
  x = x+1
  PRINT *, 'x =', x
  !$OMP END SINGLE
!$OMP END PARALLEL

END
```

```
$ ./a.out
x=2
x=3
```

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
  int x = 1;
  omp_set_num_threads(16);
#pragma omp parallel
{
  #pragma omp single
  {
    x++;
    printf("x=%d₩n", x);
  }
  #pragma omp single
  {
    x++;
    printf("x=%d₩n", x);
  }
}
  return 0;
}
```

```
$ ./a.out
x=2
x=3
```

동기화 : nowait

Fortran

```
PROGRAM nowait

   INTEGER :: x=1

   call omp_set_num_threads(16)
!$OMP PARALLEL
   !$OMP SINGLE
   x=x+1
   print*, 'x =', x
   !$OMP END SINGLE nowait

   !$OMP SINGLE
   x=x+1
   print*, 'x =', x
   !$OMP END SINGLE
!$OMP END PARALLEL

END
```

What happen?

```
$ ./a.out
x=2
x=3
$ ./a.out
x=3
x=2
$ ./a.out
x=3
x=3
```

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
   int x = 1;
   omp_set_num_threads(16);
#pragma omp parallel
{
   #pragma omp single nowait
   {
      x++;
      printf("x=%d₩n", x);
   }
   #pragma omp single
   {
      x++;
      printf("x=%d₩n", x);
   }
}
   return 0;
}
```

What happen?

```
$ ./a.out
x=2
x=3
$ ./a.out
x=3
x=2
$ ./a.out
x=3
x=3
```


동기화 : nowait

Fortran

```
program nowait_for
    integer, parameter :: N = 9999
    integer i, j, a(0:N, 0:N)

    call omp_set_num_threads(4)
!$omp parallel private(i,j)
    !$omp do
    do j=0, N
        do i=0, j
            a(i,j) = i+j
        end do
    end do
    !$omp do
    do j=0, N
        do i=j+1, N
            a(i,j) = i-j
        end do
    end do
!$omp end parallel
End
```

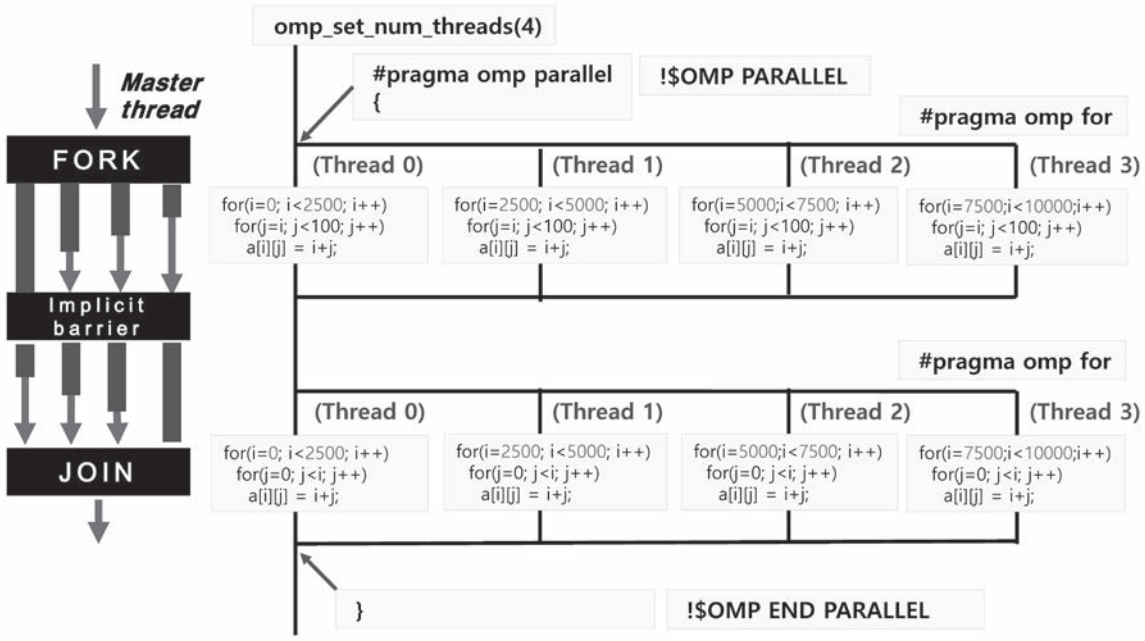
```
$ gfortran –fopenmp –o for_nowait1.x nowait.f90
$ time ./for_nowait1.x
```

C

```
#include <stdio.h>
#include <omp.h>
#define N 10000
int main(){
   int i, j, a[N][N];
   omp_set_num_threads(4);
#pragma omp parallel private(i, j)
{
   #pragma omp for
   for(i=0; i<N; i++)
      for(j=i; j<N; j++)
         a[i][j] = i+j;

   #pragma omp for
   for(i=0; i<N; i++)
      for(j=0; j<i; j++)
         a[i][j] = i-j;
}
}
```

```
$ gcc –fopenmp –o for_nowait1.x nowait.c
$ time ./for_nowait1.x
```

동기화 : nowait



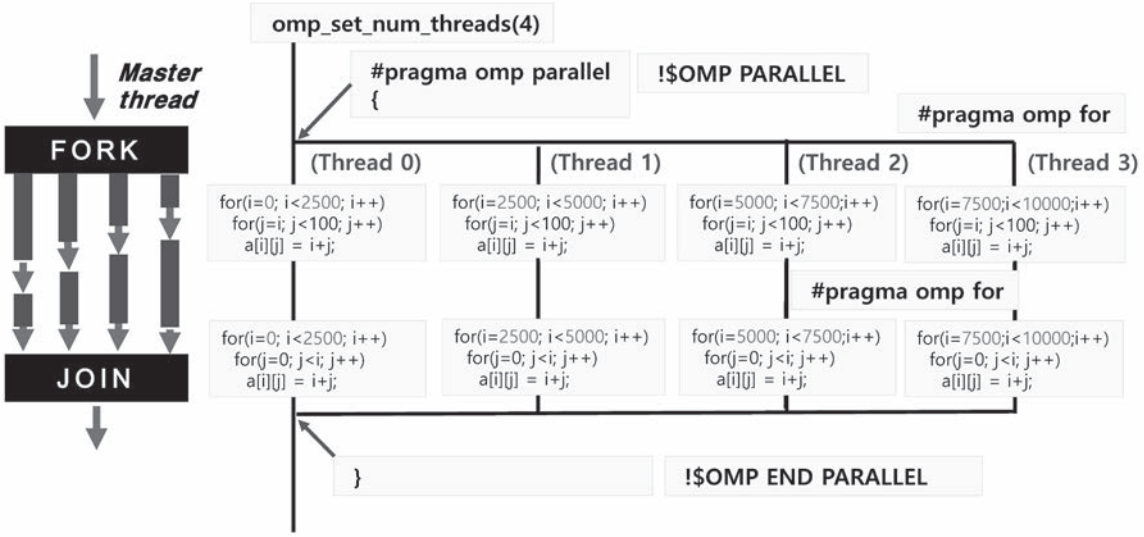
동기화 : nowait

Fortran	C

program nowait_for
integer, parameter :: N = 9999
integer i, j, a(0:N, 0:N)
call omp_set_num_threads(4)
!$omp parallel private(i,j)
!$omp do
do j=0, N
do i=0, j
a(i,j) = i+j
end do
end do
!$omp end do **nowait**
!$omp do
do j=0, N
do i=j+1, N
a(i,j) = i-j
end do
end do
!$omp end parallel
End
	#include <stdio.h>
	#include <omp.h>
	#define N 10000
	
	int main(){
	int i, j, a[N][N];
	omp_set_num_threads(4);
	#pragma omp parallel private(i, j)
	{
	#pragma omp for **nowait**
	for(i=0; i<N; i++)
	for(j=i; j<N; j++)
	a[i][j] = i+j;
	
	#pragma omp for
	for(i=0; i<N; i++)
	for(j=0; j<i; j++)
	a[i][j] = i-j;
	}
	}

$ gfortran –fopenmp –o for_nowait2.x nowait.f90 $ time ./for_nowait2.x	$ gcc –fopenmp –o for_nowait2.x nowait.c $ $ time ./for_nowait2.x

동기화 : nowait



critical

- **Fortran**

```
!$OMP CRITICAL [(name)]
      structured block
!$OMP END CRITICAL [(name)]
```

- **C**

```
#pragma omp critical [(name)]
      structured block
```

critical

- 병렬영역 안에서 **critical section**을 지정
- 매 순간 오직 하나의 **critical section**이 하나의 스레드에서 실행
  - 각 section들의 실행순서는 정해져 있지 않음
  - 하나의 section이 실행되면 다른 모든 section들은 대기
- 같은 **name**을 가지면 같은 **critical section**
  - name이 없는 section들은 동일 section 간주

critical

```
!$OMP PARALLEL PRIVATE(i) SHARED(cnt1, cnt2)
!$OMP DO
   DO i = 1, n
     ... do_work ...
     IF (condition1) THEN
       !$OMP CRITICAL (name1)
         cnt1 = cnt1 + 1
       !$OMP END CRITICAL (name1)
     ELSE
       !$OMP CRITICAL (name1)
         cnt1 = cnt1 - 1
       !$OMP END CRITICAL (name1)
     ENDIF
     IF (condition2) THEN
       !$OMP CRITICAL(name2)
         cnt2 = cnt2 + 1
       !$OMP END CRITICAL (name2)
     ENDIF
   ENDDO
!$OMP END PARALLEL
```

critical: Fortran

```
      SUM=0.0
!$OMP PARALLEL DO PRIVATE(X)
      DO I = 0, NUM_STEPS-1
        X = (I+0.5)*STEP
!$OMP CRITICAL
        SUM= SUM + 4.0/(1.0+X*X)
 !$OMP END CRITICAL
      ENDDO
[!$OMP END PARALLEL DO]
      PI = PI + SUM*STEP
```


critical: C

```
  sum=0.0;
#pragma omp parallel for private (x)
  for (i=0; i< num_steps; i++)
  {
      x = (i+0.5)*step;
  #pragma omp critical
  {
      sum += 4.0/(1.0+x*x);
  }
  }
  pi += sum*step;
```

atomic

- **Fortran**

```
!$OMP ATOMIC
    x = x operator expr
...
!$OMP ATOMIC
    x = intrinsic(x, expr)
```

- **C**

```
#pragma omp atomic
    statement
```


atomic

- 하나의 스칼라 변수를 경신하는 지정된 단일 문장(하나의 메모리 위치)에 대해 여러 스레드가 접근하는 것을 방지
- **mini-critical section**의 역할
- 사용 가능한 연산자

Fortran	C
+, -, *, /, .AND., .OR., .EQV., .NEQV, MAX, MIN, IAND, IOR, IEOR	+, -, *, /, &, ^, |, <<, >>

atomic: Fortran

```
      SUM=0.0
!$OMP PARALLEL DO PRIVATE(X)
      DO I = 0, NUM_STEPS-1
        X = (I+0.5)*STEP
!$OMP ATOMIC
        SUM= SUM + 4.0/(1.0+X*X)
      ENDDO
[!$OMP END PARALLEL DO]
      PI = PI + SUM*STEP
```

```
gfortran -fopenmp pi_omp_atom.f90 -o ppi4.x
time ./ppi4.x
```


atomic: C

```
  sum=0.0;
#pragma omp parallel for private (x)
  for (i=0; i< num_steps; i++)
  {
      x = (i+0.5)*step;
  #pragma omp atomic
      sum += 4.0/(1.0+x*x);
  }
  pi += sum*step;
```

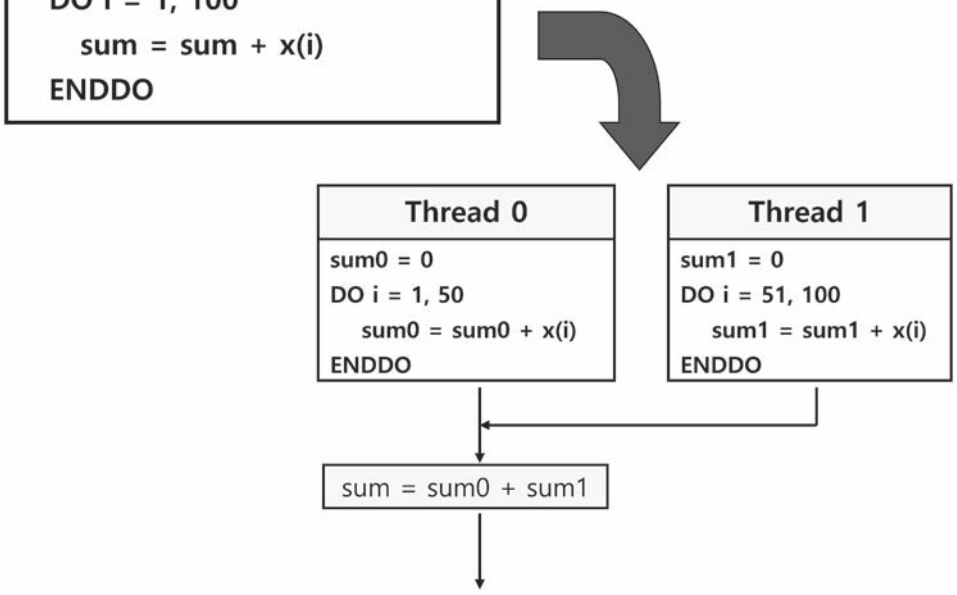
```
gcc -fopenmp pi_omp_atom.c -o ppi4.x
time ./ppi4.x
```

clause : reduction

- reduction(operator|intrinsic:var1, var2,...)
  - reduction 변수는 shared
    - 배열 가능(Fortran only): deferred shape, assumed shape array 사용 불가
    - C는 scalar 변수만 가능
  - 각 스레드에 복제돼 연산에 따라 다른 값으로 초기화되고(표 참조) 병렬 연산 수행
  - 다중 스레드에서 병렬로 수행된 계산결과를 환산해 최종 결과를 마스터 스레드로 내 놓음

clause : reduction

```
!$OMP DO reduction(+:sum)
  DO i = 1, 100
    sum = sum + x(i)
  ENDDO
```



clause : reduction

➢ Reduction Operators : Fortran

Operator	Data Types	초기값
+	integer, floating point (complex or real)	0
*	integer, floating point (complex or real)	1
-	integer, floating point (complex or real)	0
.AND.	logical	.TRUE.
.OR.	logical	.FALSE.
.EQV.	logical	.TRUE.
.NEQV.	logical	.FALSE.
MAX	integer, floating point (real only)	가능한 최소값
MIN	integer, floating point (real only)	가능한 최대값
IAND	integer	all bits on
IOR	integer	0
IEOR	integer	0

reduction: Fortran

```
!$OMP PARALLEL DO REDUCTION(+:SUM) PRIVATE(X)
      DO I = 0, NUM_STEPS-1
        X = (I+0.5)*STEP
        SUM = SUM + 4.0/(1.0+X*X)
      ENDDO
!$OMP END PARALLEL DO
      PI = SUM*STEP
```

```
gfortran -fopenmp pi_omp_rd.f90 -o ppi5.x
time ./ppi5.x
```

reduction: C

```
#pragma omp parallel for reduction(+:sum) private(x)
      for (i=0;i< num_steps; i++){
            x = (i+0.5)*step;
            sum = sum + 4.0/(1.0+x*x);
      }
      pi = step * sum;
```

```
gcc -fopenmp pi_omp_rd.c -o ppi5.x
time ./ppi5.x
```


Basic

- Creating Threads
- Data Environment
- Parallel Loops
- Synchronization
- **Scheduling**

clause: schedule

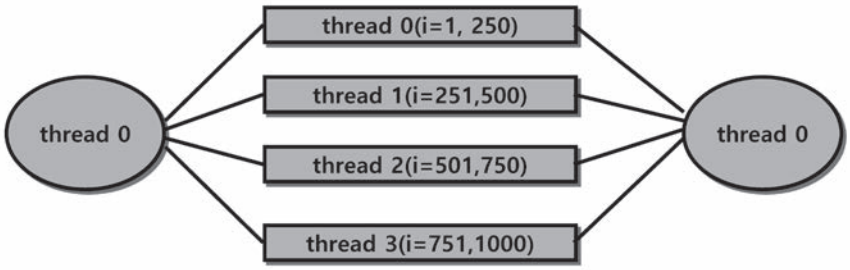
- schedule(type [,chunk_size])
  - 루프의 schedule : 루프실행의 분배문제
  - 기본적인 schedule 정책 : 실행 회수 균등 분배
  - 작업의 균등 분배를 위해 schedule clause 사용
  - type (default는 implementation depend)
    - static
    - dynamic
    - guided
    - runtime

```
!$OMP PARALLEL DO private(xkind)
   DO i = 1, N
     xkind = f(i)
     IF (xkind < 10) THEN
        CALL fast(x(i))
     ELSE
        CALL slow(x(i))
     ENDIF
   ENDDO
```


schedule: static

- schedule (static)
  - 반복실행이 각 스레드에 균일하게 할당

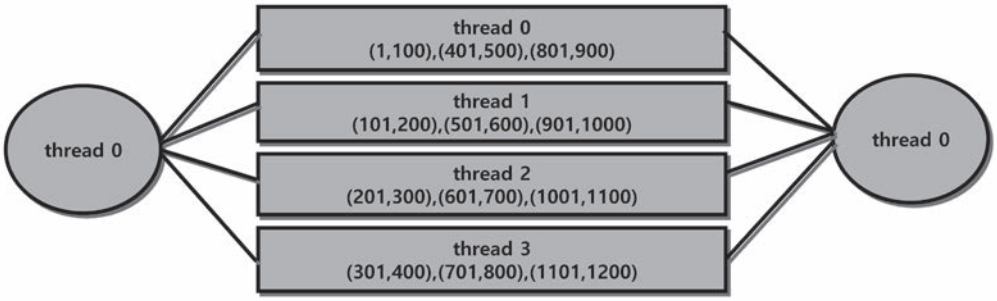
```
!$OMP DO shared(x) private(i) &
!$ schedule(static)
   DO i = 1, 1000
     x(i) = a
   ENDDO
```



schedule: static

- schedule(static, chunk_size)
  - 총 반복실행 회수를 chunk_size로 나누어 chunk 생성
  - chunk들을 스레드에 라운드-로빈 방식으로 정적 할당

```
!$OMP DO shared(x) private(i) &
!$ schedule(static, 100)
   DO i = 1, 1200
      ... work ...
   ENDDO
```



schedule: dynamic

- schedule (dynamic, chunk_size)
  - chunk들을 스레드에 동적할당
    - 작업이 먼저 끝나는 스레드에 다음 chunk 할당
  - chunk_size가 없으면 디폴트 chunk_size = 1

```
!$OMP DO schedule(dynamic, 1000)
   DO i = 1, 10000
      ... work ...
   ENDDO
```

schedule: guided

- schedule (guided, chunk_size)
  - dynamic scheduling
  - chunk 크기가 변한다.(N_0:반복회수, P:스레드 개수)
    - N_n = MAX(N_{n-1}-size($chunk_{n-1}$),P*chunk_size) (n≥1)
    - size($chunk_n$) = CEILING(N_n/P)
  - 각 스레드가 DO구문에 도착하는 순서가 다를 때 유용

```
!$OMP DO schedule(guided, 55)
  DO i = 1, 12000
    ... work ...
  ENDDO
```


schedule: runtime

- schedule(runtime)
  - 프로그램 실행 중에 환경변수 OMP_SCHEDULE 값을 참조
  - 재 컴파일 없이 다양한 스케줄링 방식 시도 가능

```
export OMP_SCHEDULE="static,1000"
export OMP_SCHEDULE="dynamic"
```

SCHEDULE

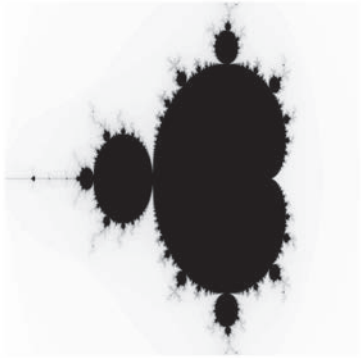
- **Mandelbrot**
  - Mandelbrot is an image computing and display computation
  - Pixels of an image are stored in a 2D array
  - Each pixel is computed by iterating the complex function

$$Z_{k+1} = {Z_k}^2 + C, \qquad Z_k = a_k + b_k i, \qquad C = C_{real} + C_{imag} i$$

$$Z_{k+1} = (a_k + b_k i)^2 + (C_{real} + C_{imag} i)$$

$$= \left({a_k}^2 - {b_k}^2 + C_{real} \right) + \left(2 a_k b_k + C_{imag} \right) i$$

  - The magnitude of z is given by $Z_{length} = \sqrt{a^2 + b^2}$



SCHEDULE (Exercise)

C (Serial)

```
#include <stdio.h>

#define   X_RESN  4000      /* x resolution */
#define   Y_RESN  4000      /* y resolution */
#define   X_MIN   -2.0
#define   X_MAX    2.0
#define   Y_MIN   -2.0
#define   Y_MAX    2.0

typedef struct complextype {
   float real, imag;
} Compl;

int main ( int argc, char* argv[])
{
   int i, j, k, maxIterations = 1000;
   Compl   z, c;
   float   lengthsq, temp;
   int res[X_RESN][Y_RESN];
```

```
   for(i=0; i < X_RESN; i++) {
      for(j=0; j < Y_RESN; j++) {
         z.real = z.imag = 0.0;
         c.real = X_MIN + j * (X_MAX - X_MIN)/X_RESN;
         c.imag = Y_MAX - i * (Y_MAX - Y_MIN)/Y_RESN;
         k = 0;

         do  {
            temp = z.real*z.real - z.imag*z.imag + c.real;
            z.imag = 2.0*z.real*z.imag + c.imag;
            z.real = temp;
            lengthsq = z.real*z.real+z.imag*z.imag;
            k++;
         } while (lengthsq < 4.0 && k < maxIterations);

         if (k >= maxIterations) res[i][j] = 0;
         else res[i][j] = 1;
      }
   }
}
```

```
gcc -fopenmp mandelbrot_serial.c -o mans.x
time ./mans.x
```

SCHEDULE (Exercise)

Fortran (Serial)

```
program ex

  integer, parameter :: X_RESN=4000, Y_RESN=4000
  real, parameter  :: X_MIN=-2.0, X_MAX=2.0, Y_MIN=-2.0,
   Y_MAX=2.0

  complex :: z, c
  integer  :: i, j, k, maxIterations=1000
  real  :: lengthsq, temp
  integer, dimension(0:X_RESN-1, 0:Y_RESN-1)  :: res

  DO j=0, Y_RESN-1
     DO i=0, X_RESN-1
        z = (0.0, 0.0)
        c = cmplx( X_MIN+real(j)*(X_MAX-X_MIN)/real(X_RESN),
   Y_MAX - real(i) * (Y_MAX - Y_MIN)/real(Y_RESN) )
        k=0
```

```
loop:  DO
            temp = REAL(z)*REAL(z)-AIMAG(z)*AIMAG(z) + REAL(c)
            z=cmplx( temp, 2.0*REAL(z)*AIMAG(z)+AIMAG(c) )
            lengthsq = REAL(z)*REAL(z)+AIMAG(z)*AIMAG(z)
            k=k+1
            if (lengthsq >= 4.0 .or. k >= maxIterations) exit loop
         END DO loop

         if ( k >= maxIterations ) then
            res(i, j) = 0
         else
            res(i, j) = 1
         end if

      END DO
    ENDDO

end
```

```
gfortran -fopenmp mandelbrot_serial.f90 -o mans.x
time ./mans.x
```


SCHEDULE (Solution)

C (parallel)

```
#include <stdio.h>
#include <omp.h>

#define        X_RESN  4000       /* x resolution */
#define        Y_RESN  4000       /* y resolution */
#define        X_MIN   -2.0
#define        X_MAX    2.0
#define        Y_MIN   -2.0
#define        Y_MAX    2.0

typedef struct complextype {
   float real, imag;
} Compl;

int main ( int argc, char* argv[])
{
   int i, j, k, maxIterations = 1000;
   Compl   z, c;
   float   lengthsq, temp;
   int res[X_RESN][Y_RESN];
```

```
gcc –fopenmp mandelbrot_omp.c  -o mandp.x
export OMP_NUM_THREADS=8
export OMP_SCHEDULE=static [or "dynamic,5"]
time ./manp.x
```

```
#pragma omp parallel for shared(res, maxIterations)
    private(j,z,c,k,temp,lengthsq) schedule(runtime)
   for(i=0; i < X_RESN; i++) {
      for(j=0; j < Y_RESN; j++) {
         z.real = z.imag = 0.0;
         c.real = X_MIN + j * (X_MAX - X_MIN)/X_RESN;
         c.imag = Y_MAX - i * (Y_MAX - Y_MIN)/Y_RESN;
         k = 0;

         do  {
            temp = z.real*z.real - z.imag*z.imag + c.real;
            z.imag = 2.0*z.real*z.imag + c.imag;
            z.real = temp;
            lengthsq = z.real*z.real+z.imag*z.imag;
            k++;
         } while (lengthsq < 4.0 && k < maxIterations);

         if (k >= maxIterations) res[i][j] = 0;
         else res[i][j] = 1;
      }
   }
}
```

SCHEDULE (Solution)

Fortran (Parallel)

```
program solution

  integer, parameter :: X_RESN=4000, Y_RESN=4000
  real, parameter  :: X_MIN=-2.0, X_MAX=2.0, Y_MIN=-2.0,
   Y_MAX=2.0

  complex :: z, c
  integer  :: i, j, k, maxIterations=1000
  real  :: lengthsq, temp
  integer, demension(0:X_RESN-1, 0:Y_RESN-1)  :: res

!$OMP PARALLEL DO shared(res, maxIterations) private(z, c, k,
    temp, lengthsq) schedule(runtime)
  DO j=0, Y_RESN-1
     DO i=0, X_RESN-1
        z = (0.0, 0.0)
        c = cmplx( X_MIN+real(j)*(X_MAX-Y_MIN)/real(X_RESN),
 Y_MAX - real(i) * (Y_MAX - Y_MIN)/real(Y_RESN) )
        k=0
```

```
loop:    DO
           temp = REAL(z)*REAL(z)-AIMAG(z)*AIMAG(z) + REAL(c)
           z=cmplx( temp, 2.0*REAL(z)*AIMAG(z)+AIMAG(c) )
           lengthsq = REAL(z)*REAL(z)+AIMAG(z)*AIMAG(z)
           k=k+1
           if (lengthsq >= 4.0 .or. k >= maxIterations) exit loop
         END DO loop

       if ( k >= maxIterations ) then
         res(i, j) = 0
       else
         res(i, j) = 1
       end if

     END DO
  ENDDO

end
```

```
gfortran –fopenmp mandelbrot_omp.f90  -o mandel_omp
export OMP_NUM_THREADS=8
export OMP_SCHEDULE=static [or "dynamic,5"]
./mandel_omp
```


Directives and clauses

Clauses	Directives					
	parallel	do/for	sections	single	parallel do/for	parallel sections
if	●				●	●
private	●	●	●	●	●	●
shared	●	●			●	●
default	●				●	●
firstprivate	●	●	●	●	●	●
lastprivate		●	●		●	●
reduction	●	●	●		●	●
copyin	●				●	●
schedule		●			●	
ordered		●			●	
nowait		●	●	●		

Intermediate

- **Task**

Task

- **Thread encountering parallel construct packages up a set of implicit tasks, one per thread.**
- **Team of threads is created.**
- **Each thread in team is assigned to one of the tasks(and tied to it).**
- **Barrier holds original master thread until all implicit tasks are finished.**

➔ **We have simply added a way to create a task explicitly for the team to execute !!**

Task

- **The most innovative feature**
- **OpenMP 3.0 simply adds the ability to create explicit tasks.**
- **Allows to parallelize irregular problems**
  - While loops
  - Recursive structures
  - …

Task Construct

!$omp task [clause[[,] clause] ...]

structured-block

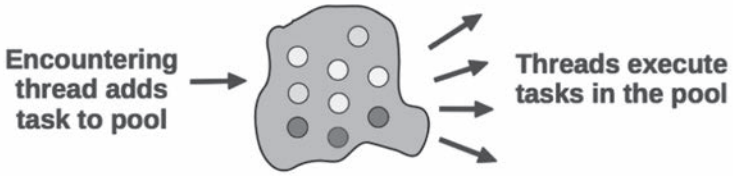
!$omp end task

clause:

if(*scalar-expression*)
untied
default(shared | none)
private(*list*)
firstprivate(*list)*
shared(*list*)

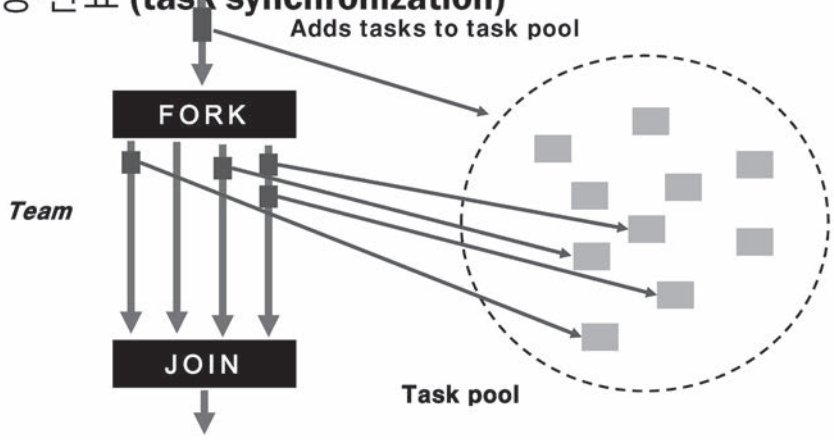
Task Concepts

- **task scheduling =** 은행 번호표 시스템
- **task =** 은행에서 처리할 일
- **thread =** 은행원



The life cycle of a task

1. **task** 생성 **(task** 구문**)**
2. **task** 실행 **(**즉시 또는 대기**)**
3. 경우에 따라 일시 정지 후 재 실행 가능, 이때 다른 스레드에 의해 재 실행 가능**(task switching)**
4. 실행 완료 **(task synchronization)**



Task (1/3)

Fortran	C

Program task1
 	
!$OMP PARALLEL num_threads(32)
write(*, '(A)', advance="no") 'A '
write(*, '(A)', advance="no") 'B '
write(*, '(A)', advance="no") 'C '
write(*, '(A)', advance="no") 'D '
write(*, *) ' '
!$OMP END PARALLEL
 	
END
	#include <stdio.h>
	#include <omp.h>
	
	int main()
	{
	#pragma omp parallel num_threads(32)
	{
	printf("A ");
	printf("B ");
	printf("C ");
	printf("D ");
	printf("₩n");
	}
	}

Task (2/3)

Fortran	C

Program task2
integer omp_get_thread_num
 	
!$OMP PARALLEL num_threads(32)
!$OMP SINGLE
print *, 'A tid =', omp_get_thread_num()
print *, 'B tid =', omp_get_thread_num()
print *, 'C tid =', omp_get_thread_num()
print *, 'D tid =', omp_get_thread_num()
!$OMP END SINGLE
!$OMP END PARALLEL
 	
END
	#include <stdio.h>
	#include <omp.h>
	
	int main()
	{
	#pragma omp parallel num_threads(32)
	{
	#pragma omp single
	{
	printf("A tid=%d₩n", omp_get_thread_num());
	printf("B tid=%d₩n", omp_get_thread_num());
	printf("C tid=%d₩n", omp_get_thread_num());
	printf("D tid=%d₩n", omp_get_thread_num());
	}
	}
	}

Task (3/3)

Fortran	C

Program task3
  integer omp_get_thread_num
 	
!$OMP PARALLEL num_threads(32)
  !$OMP SINGLE
  print *, 'A tid =', omp_get_thread_num()
  !$OMP TASK
  print *, 'B tid =', omp_get_thread_num()
  !$OMP END TASK
  !$OMP TASK
  print *, 'C tid =', omp_get_thread_num()
  !$OMP END TASK
  print *, 'D tid =', omp_get_thread_num()
  !$OMP END SINGLE
!$OMP END PARALLEL
 	
END
	#include <stdio.h>
	#include <omp.h>
	
	int main()
	{
	#pragma omp parallel num_threads(32)
	{
	  #pragma omp single
	  {
	  printf("A tid=%d₩n", omp_get_thread_num());
	  #pragma omp task
	  {
	  printf("B tid=%d₩n", omp_get_thread_num());
	  }
	  #pragma omp task
	  {
	  printf("C tid=%d₩n", omp_get_thread_num());
	  }
	  printf("D tid=%d₩n", omp_get_thread_num());
	  }
	}
	}

Task : taskwait

Fortran	C

Program task4
  integer omp_get_thread_num
 	
!$OMP PARALLEL num_threads(32)
!$OMP SINGLE
  print *, 'A tid =', omp_get_thread_num()
!$OMP TASK
  print *, 'B tid =', omp_get_thread_num()
!$OMP END TASK
!$OMP TASK
  print *, 'C tid =', omp_get_thread_num()
!$OMP END TASK
!$OMP TASKWAIT
  print *, 'D tid =', omp_get_thread_num()
!$OMP END SINGLE
!$OMP END PARALLEL
 	
END
	#include <stdio.h>
	#include <omp.h>
	
	int main()
	{
	#pragma omp parallel num_threads(32)
	{
	  #pragma omp single
	  {
	  printf("A tid=%d₩n", omp_get_thread_num());
	  #pragma omp task
	  {
	  printf("B tid=%d₩n", omp_get_thread_num());
	  }
	  #pragma omp task
	  {
	  printf("C tid=%d₩n", omp_get_thread_num());
	  }
	  #pragma omp taskwait
	  printf("D tid=%d₩n", omp_get_thread_num());
	  }
	}
	}

Task : 데이터 유효범위 (1/2)

Fortran

```
Program task_data_scope
  integer :: a, b, c, d, e

!$OMP PARALLEL private(b,d,e)
  !$OMP TASK private(e)
    a, b, c, d, e : private? shared?
  !$OMP END TASK
!$OMP END PARALLEL
END
```

```
Program task
USE OMP_LIB or include "omp_lib.h"

  integer :: a, b, c, d, e
!$OMP PARALLEL private(b,d,e)
  !$OMP TASK private(e)
    a : shared
    b : firstprivate
    c : shared
    d : firstprivate
    e : private
  !$OMP END TASK
!$OMP END PARALLEL

END
```

C

```
int foo()
{
  int a, b, c,d,e;
#pragma omp parallel private(b,d,e)
{
  #pragma omp task private(e)
  {     a, b, c, d, e : private? shared?   }
}
}
```

```
int foo()
{
  int a, b, c,d,e;
#pragma omp parallel private(b,d,e)
{
  #pragma omp task private(e)
  {
    a : shared
    b : firstprivate
    c : shared
    d : firstprivate
    e : private
  }
}
}
```


Task : 데이터 유효범위 (2/2)

Fortran

```
program task_data_scope
    integer :: a=1, b=2, c=3, d=4, e=5
    integer tid, omp_get_thread_num

    call omp_set_num_threads(4)
!$omp parallel private(b,d,e,tid)
    tid = omp_get_thread_num()
    print 10, 'tid=',tid, ' a=',a, ' b=',b, ' c=',c, ' d=',d, ' e=',e
    !$omp single
    print*, "in single:", tid
    a=2; b=3; c=4; d=5;e=6
    !$omp end single
    !$omp task private(e)
    print 10, 'task tid=',omp_get_thread_num(), &
              & ' a=',a, ' b=',b, ' c=',c, ' d=',d, ' e=',e
    !$omp end task
!$omp end parallel

10 format(A,I4,A,I4,A,I4,A,I4,A,I4,A,I4)
End
```

C

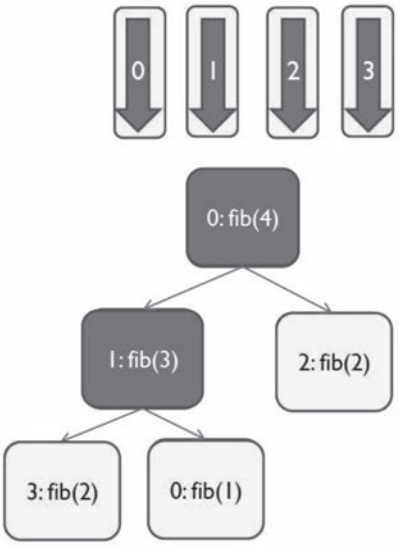
```
#include <stdio.h>
#include <omp.h>

int main()
{
    int a=1, b=2, c=3, d=4, e=5, tid;

    omp_set_num_threads(4);
#pragma omp parallel private(b,d,e, tid)
{
    tid = omp_get_thread_num();
    printf("tid=%d a=%d b=%d c=%d d=%d e=%d₩n",
              tid, a, b, c, d, e);
    #pragma omp single
    { printf("in single: %d ₩n", tid);
      a=2, b=3, c=4, d=5, e=6; }
    #pragma omp task private(e)
    {
      printf("task tid=%d a=%d b=%d c=%d d=%d e=%d₩n",
                omp_get_thread_num(), a, b, c, d, e);
    }
}
}
```

Task : Fibonacci

```
program fibonacci
implicit none
   integer :: result
   result = fibon(MAX)
   print *, "Fibonacci (", MAX, ") =", result
   contains
recursive integer function fibon(n) RESULT(fib)
implicit none
   integer :: n, x, y
   if(n<2) then
      fib=n
   else
      x=fibon(n-1);
      y=fibon(n-2);
      fib = x + y
   endif
end function
end

$ gfortran –DMAX=30 –cpp  –o fibon.x fibonacci.f90
$ ./fibon.x
```

```
#include <stdio.h>

int fibon(int n) // f(n) = f(n-1) + f(n-2)
{
   int x, y;
   if(n<2) return n;
   x=fibon(n-1);
   y=fibon(n-2);
   return (x+y);
}

int main()
{
   int result = fibon(MAX);
   printf("Fibonacci (%d) = %d₩n", MAX, result);
   return 0;
}


$ gcc –DMAX=30 –o fibon.x fibonacci.c
$ ./fibon.x
```


Task : Fibonacci



Task : Fibonacci

```
recursive integer function fibon(n) RESULT(fib)
   integer :: n, x, y
   if(n<2) then
      fib = n
   else
!$OMP TASK shared(x)
      x=fibon(n-1)
!$OMP END TASK
!$OMP TASK shared(y)
      y=fibon(n-2)
!$OMP END TASK
!$OMP TASKWAIT
      fib = x + y
   end if
end function
```

```
int fibon(int n) // f(n) = f(n-1) + f(n-2)
{
   int x, y;
   if(n<2) return n;
   #pragma omp task shared(x)
   { x=fibon(n-1); }
   #pragma omp task shared(y)
   { y=fibon(n-2); }
   #pragma omp taskwait
   return (x+y);
}
```


Task : Fibonacci

```
recursive integer function fibon(n) RESULT(fib)
   integer :: n, x, y
   if(n<2) then
      fib = n
   else if(n<30) then
      fib = fibon(n-1) + fibon(n-2)
   else
!$OMP TASK shared(x)
      x=fibon(n-1)
!$OMP END TASK
!$OMP TASK shared(y)
      y=fibon(n-2)
!$OMP END TASK
!$OMP TASKWAIT
      fib = x + y
   end if
end function
```

```
int fibon(int n) // f(n) = f(n-1) + f(n-2)
{
   int x, y;
   if(n<2) return n;
   if(n<30) return fibon(n-1)+fibon(n-2);
   #pragma omp task shared(x)
   { x=fibon(n-1); }
   #pragma omp task shared(y)
   { y=fibon(n-2); }
   #pragma omp taskwait
   return (x+y);
}
```

Fibonacci Performance

➢ **gfortran –DMAX=50 –fopenmp –cpp –O3 ...**

Threads	Time (sec)
1 (serial)	8'27.37"
2	4'59.85"
4	3'48.93"
8	1'44.20"
16	1'0.50"
32	34.88"
64	18.23"

Q & A

MPI Programming

KISTI SUPERCOMPUTING EDUCATION CENTER

MPI Programming

August 2020

KISTI과학데이터스쿨
KISTI SCIENCE DATA SCHOOL
http://kacademy.kisti.re.kr

KISTI 과학데이터스쿨

Day 3: MPI

- Introduction
  - Parallel Computing, processes, and MPI
  - MPI Implementations, Compiling, and Execution
- MPI Functions
  - P2P Communications
  - Collective Communications
  - Derived Data Types
- How to Parallelize
  - Parallelizing DO Loops
  - Parallelization and Message Passing

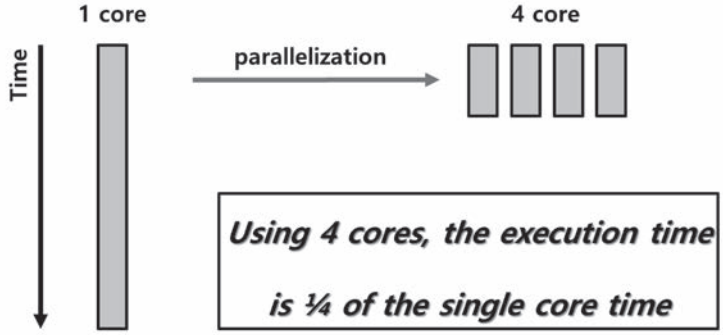
Introduction

- **Parallel Computing, processes, and MPI**
- MPI Implementations, Compiling, and Execution

Parallel Computing

Parallelization is another optimization technique
The goal is to reduce the execution time

To this end, multiple processors, or cores, are used

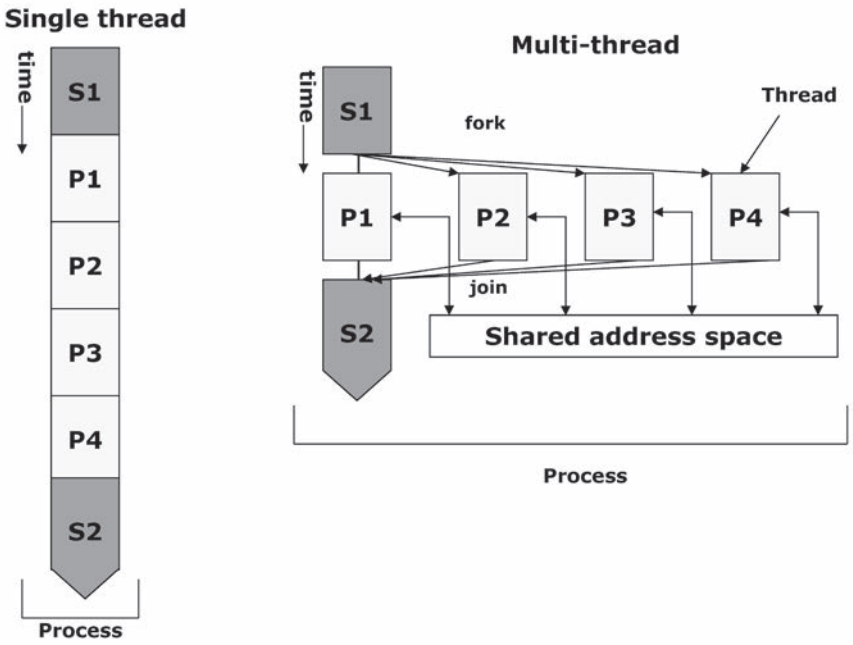


Using 4 cores, the execution time is ¼ of the single core time

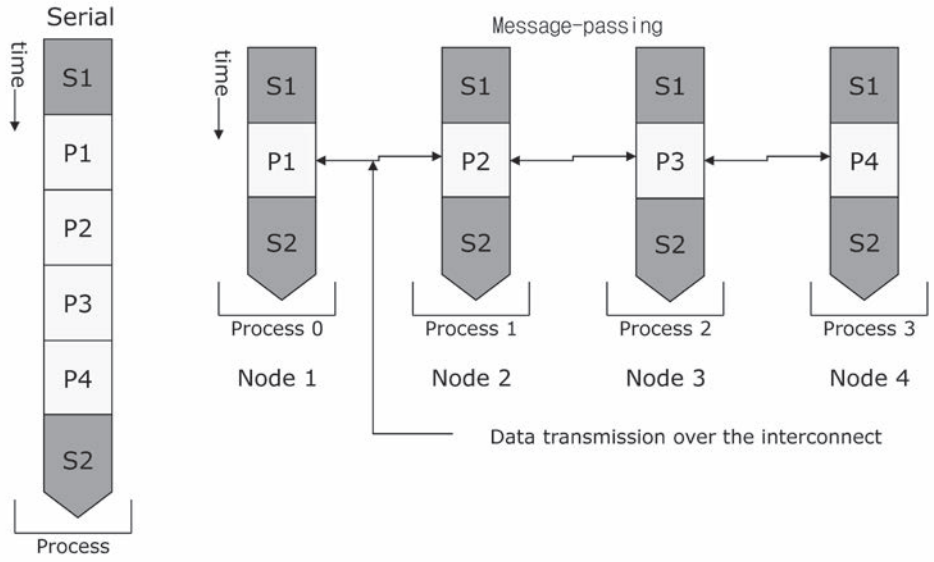
Message Passing

- 지역적으로 메모리를 따로 가지는 프로세스들이 데이터를 공유하기 위해 메시지(데이터)를 송신, 수신하여 통신하는 방식
  - 병렬화를 위한 작업할당, 데이터분배, 통신의 운용 등 모든 것을 프로그래머가 담당 : 어렵지만 유용성 좋음(Very Flexible)
  - 다양한 하드웨어 플랫폼에서 구현 가능
    - 분산 메모리 다중 프로세서 시스템
    - 공유 메모리 다중 프로세서 시스템
    - 단일 프로세서 시스템
- 메시지 패싱 라이브러리
  - MPI, PVM, SHMEM

Shared Memory Programming Model



Message Passing



MPI란 무엇인가?

- **Message Passing Interface**
- 메시지 패싱 병렬 프로그래밍을 위해 표준화된 데이터 통신 라이브러리
- **www.mpi-forum.org**

History of MPI

- MPI-1: 1994년
- MPI-1.1: 1995년
- MPI-2 (MPI-1.2): 1997년
- MPI-1.3: 2008년
- MPI-2.1(MPI1 + MPI2): 2008년
- MPI-2.2: 2009년
- MPI-3: 2012년
- MPI-3.1: 2015년(6월)

www.mpi-forum.org

Introduction

- Parallel Computing, processes, and MPI
- **MPI Implementations, Compiling, and Execution**

Common MPI Implementations

- **MPICH(Argonne National Laboratory)**
  - Most common MPI implementation
  - Derivatives
    - Mvapich, Cray, Intel, IBM, MS, Tianhe
- **Open MPI(Indiana University/LANL)**
  - Derivatives
    - SUN
- **Scali MPI Connect**
- **MPI/Pro (MPI Software Technology)**
- **...**

Implementations: MVAPICH

- **MVAPICH, also known as MVAPICH2, is a BSD-licensed implementation of the MPI standard developed by Ohio State University.**
- **MVAPICH comes in a number of flavors:**
  - MVAPICH2, with support for InfiniBand, iWARP, RoCE, and Intel Omni-Path
  - MVAPICH2-X, with support for PGAS and OpenSHMEM
  - MVAPICH2-GDR, with support for InfiniBand and NVIDIA CUDA GPUs
  - MVAPICH2-MIC, with support for InfiniBand and Intel MIC
  - MVAPICH2-Virt, with support for InfiniBand and SR-IOV
  - MVAPICH2-EA, which is energy-aware and supports InfiniBand, iWARP, and RoCE

Compiling an MPI Program

- **Most MPI implementations supply compilation scripts, eg.**

Language	Command Used to Compile
Fortran 77	mpif77 mpi_prog.f
Fortran 90	mpif90 mpi_prog.f90
C	mpicc mpi_prog.c
C++	mpiCC or mpicxx mpi_prog.C

- **Manual compilation/linking also possible**
  - *Extremely complex*

```
$ gcc –o hello.x -L/applic/compilers/gcc/4.1.2/mpi/openmpi/1.4.2/lib64
  -I/applic/compilers/gcc/4.1.2/mpi/openmpi/1.4.2/include hello.c -lmpi
```

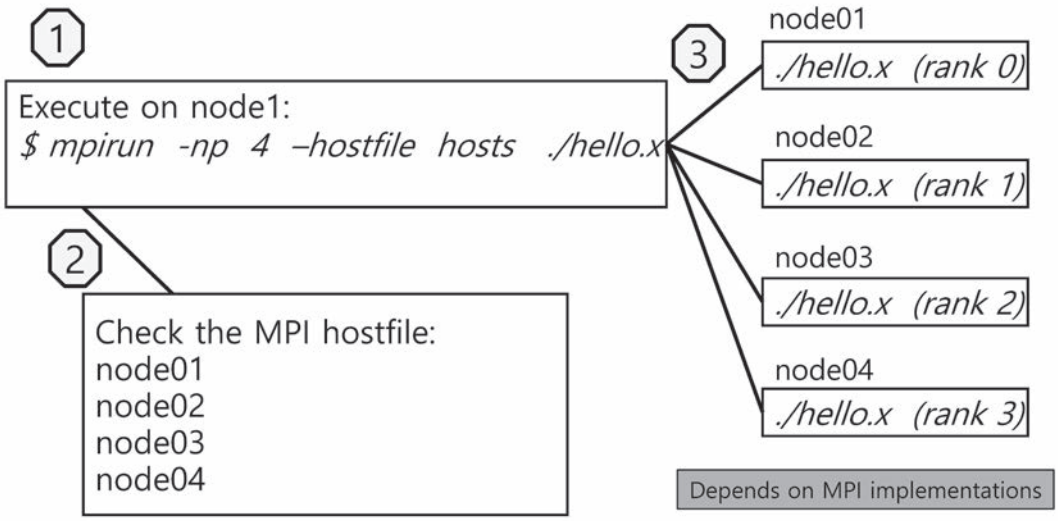
MPI Machine File

- A text file telling MPI where to launch processes
- Put separate host name on each line
- Example

```
$ cat hostfile
node01
node02
node03
node04
```

- Check implementation for multi-processor node formats
- Default file found in MPI installation

Starting an MPI Program



PBS Interactive 작업 제출

- 교육 중 실습은 **debug** 큐 이용
  - debug 큐에 작업 제출 또는 인터랙티브 작업 이용
- 인터랙티브 노드 작업: **qsub –I**

```
$ module add intel/18.0.3 impi/18.0.3
$ cd /scratch/sedu##
$ qsub -I -V -A etc -l select=2:ncpus=8 -l walltime=04:00:00 -q debug
qsub: waiting for job 4997862.pbs to start
qsub: job 4997862.pbs ready

$ qstat -nu sedu50

pbs:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
4997862.pbs     sedu50   debug    STDIN       55500   2  20   25gb 02:00 R 00:00
   node8107/0*10+node8109/0*10

$ cat > mf
node8107
node8109
```


"Hello, World" with hostname : C

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

int main(int argc, char *argv[])
{
   int ver,subver;
   int nRank, nProcs;
   char procName[MPI_MAX_PROCESSOR_NAME];
   int nNameLen;

   MPI_Init(NULL,NULL);                        // MPI Start
   MPI_Comm_rank(MPI_COMM_WORLD,&nRank);       // Get Current process rank id
   MPI_Comm_size(MPI_COMM_WORLD,&nProcs);      // Get number of processes

   MPI_Get_version(&ver,&subver);              // MPI Version Information
   if(nRank==0)printf("MPI Version %d.%d\n",ver,subver);
   MPI_Get_processor_name(procName, &nNameLen);

   printf("Hello World.(Process name=%s, nRank=%d, nProcs=%d)\n",procName, nRank,
          nProcs);
   MPI_Finalize();                            // MPI End
   return 0;
}
```

```
$ mpiicc hello_host.c -o hello_host.x
$ mpirun -np 4 -hostfile mf -ppn 2 ./hello_host.x
MPI Version 3.1
Hello World.(Process name=node8281, nRank=1, nProcs=4)
Hello World.(Process name=node8282, nRank=2, nProcs=4)
Hello World.(Process name=node8281, nRank=0, nProcs=4)
Hello World.(Process name=node8282, nRank=3, nProcs=4)
```

"Hello, World" with hostname : Fortran

```
PROGRAM hello
USE mpi_f08
IMPLICIT NONE
INTEGER::nRank, nProcs, nNameLen
CHARACTER(LEN=MPI_MAX_PROCESSOR_NAME)::procName
INTEGER::ver, subver

CALL MPI_Init
CALL MPI_Comm_size(MPI_COMM_WORLD, nProcs)
CALL MPI_Comm_rank(MPI_COMM_WORLD, nRank)

CALL MPI_Get_version(ver,subver)
IF(nRank==0) PRINT*,'MPI Version',ver,'.',subver
CALL MPI_Get_processor_name(procName,nNameLen)
PRINT*,'Hello World.(Process name=',TRIM(procName),',nRank=',nRank,',nProcs=',nProcs
CALL MPI_Finalize
END PROGRAM hello
```

```
$ mpiifort hello_host.f90 -o hello_host.x
$ mpirun -np 4 -hostfile mf -ppn 2 ./hello_host.x
 MPI Version           3 .           1
 Hello World.(Process name=node8281, nRank=           1 ,nProcs=           4
 Hello World.(Process name=node8281, nRank=           0 ,nProcs=           4
 Hello World.(Process name=node8282, nRank=           2 ,nProcs=           4
 Hello World.(Process name=node8282, nRank=           3 ,nProcs=           4
```


MPI Functions

- **P2P Communications**
- Collective Communications
- Derived Data Types

MPI 프로그램의 기본 구조

include MPI header file
variable declarations
initialize the MPI environment

... do computation and MPI communication calls ...

close MPI environment

MPI 헤더파일

Fortran	C
INCLUDE 'mpif.h'	#include "mpi.h"

MPI 초기화

Fortran	C
CALL MPI_INIT(ierr)	int MPI_Init(&argc, &argv)

- **MPI** 환경 초기화
- **MPI** 루틴 중 가장 먼저 오직 한 번 반드시 호출되어야 함

Communicator

- 서로 통신할 수 있는 프로세스들의 집합을 나타내는 핸들
- 모든 **MPI** 통신 루틴에는 커뮤니케이터 인수가 포함 됨
- 커뮤니케이터를 공유하는 프로세스들끼리 통신 가능
- **MPI_COMM_WORLD**
  - 프로그램 실행시 정해진, 사용 가능한 모든 프로세스를 포함하는 커뮤니케이터
  - MPI_Init이 호출될 때 정의 됨

Communicator

- 프로세스 랭크
  - 같은 커뮤니케이터에 속한 프로세스의 식별 번호
    - 프로세스가 n개 있으면 0부터 n-1까지 번호 할당
  - 메시지의 송신자와 수신자를 나타내기 위해 사용
  - 프로세스 랭크 가져오기

Fortran	`CALL MPI_COMM_RANK(comm, rank, ierr)`
C	`int MPI_Comm_rank(MPI_Comm comm, int *rank)`

Communicator

- 커뮤니케이터 사이즈
  - 커뮤니케이터에 포함된 프로세스들의 총 개수
  - 커뮤니케이터 사이즈 가져오기

Fortran	`CALL MPI_COMM_SIZE(comm, size, ierr)`
C	`int MPI_Comm_size(MPI_Comm comm, int *size)`

MPI 프로그램 종료

Fortran	C
`CALL MPI_FINALIZE(ierr)`	`int MPI_Finalize();`

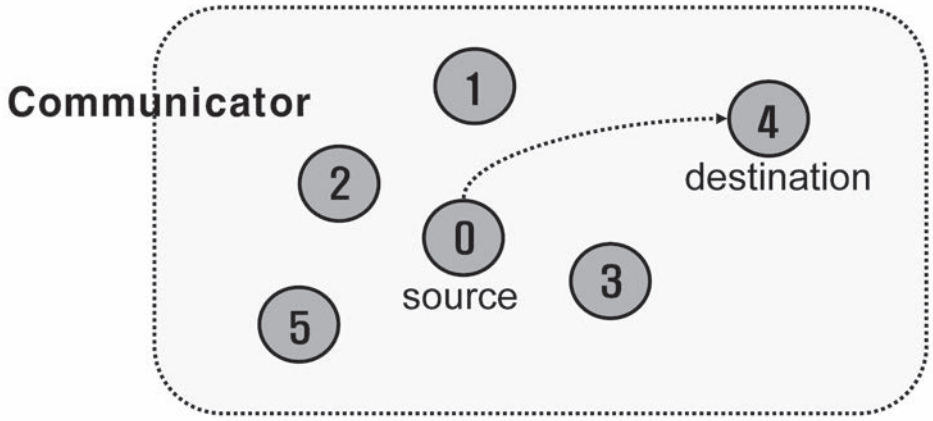
- 모든 **MPI** 자료구조 정리
- 모든 프로세스들에서 마지막으로 한 번 호출되어야 함

bones.f

```
PROGRAM skeleton
INCLUDE 'mpif.h'
INTEGER ierr, rank, size
CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)

! ... your code here ...

CALL MPI_FINALIZE(ierr)
END
```

점대점 통신

- 두 개의 프로세스만 참여하는 통신
- 송신/수신 프로세스의 확인을 위해 커뮤니케이터와 랭크 사용



점대점 통신

- 통신의 완료
  - 메시지 전송에 이용된 메모리 위치에 안전하게 접근할 수 있음을 의미
    - 송신 : 송신 변수는 통신이 완료되면 다시 사용될 수 있음
    - 수신 : 수신 변수는 통신이 완료된 후부터 사용될 수 있음
- 블록킹 통신과 논블록킹 통신
  - 블록킹
    - 통신이 완료된 후 루틴으로부터 리턴 됨
  - 논블록킹
    - 통신이 시작되면 완료와 상관없이 리턴, 이후 완료 여부를 검사
- 통신 완료에 요구되는 조건에 따라 통신 모드 분류

블록킹 송신:표준

C	`int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)`
Fortran	`MPI_SEND(buf, count, datatype, dest, tag, comm, ierr)`

(CHOICE) buf : 송신 버퍼의 시작 주소 (IN)

INTEGER count : 송신될 원소 개수 (IN)

INTEGER datatype : 각 원소의 MPI 데이터 타입(핸들) (IN)

INTEGER dest : 수신 프로세스의 랭크 (IN)

통신이 불필요하면 MPI_PROC_NULL

INTEGER tag : 메시지 꼬리표 (IN)

INTEGER comm : MPI 커뮤니케이터(핸들) (IN)

```
If(myrank==0)
  MPI_SEND(a, 50, MPI_REAL, 5, 1, MPI_COMM_WORLD, ierr)
```


블록킹 수신

C	`int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)`
Fortran	`MPI_RECV(buf, count, datatype, source, tag, comm, status, ierr)`

(CHOICE) buf : 수신 버퍼의 시작 주소 (OUT)

INTEGER count : 수신될 원소 개수 (IN)

INTEGER datatype : 각 원소의 MPI 데이터 타입(핸들) (IN)

INTEGER source : 송신 프로세스의 랭크 (IN)

통신이 불필요하면 MPI_PROC_NULL

INTEGER tag : 메시지 꼬리표 (IN)

INTEGER comm : MPI 커뮤니케이터(핸들) (IN)

INTEGER status(MPI_STATUS_SIZE) : 수신된 메시지의 정보 저장 (OUT)

```
If(myrank==5)
  MPI_RECV(a,50,MPI_REAL,0,1,MPI_COMM_WORLD,status,ierr)
```

블록킹 수신

- 수신자는 와일드 카드를 사용할 수 있음
- 모든 프로세스로부터 메시지 수신
  MPI_ANY_SOURCE
- 어떤 꼬리표를 단 메시지든 모두 수신
  MPI_ANY_TAG

블록킹 수신

- 수신자의 **status** 인수에 저장되는 정보
  - 송신 프로세스
  - 꼬리표
  - 데이터 크기 : MPI_GET_COUNT 사용

Information	Fortran	C
source	status(MPI_SOURCE)	status.MPI_SOURCE
tag	status(MPI_TAG)	status.MPI_TAG
count	MPI_GET_COUNT	MPI_Get_count

블록킹 수신

➢ MPI_GET_COUNT

– 수신된 메시지의 원소 개수 리턴

C	`int MPI_Get_count(MPI_Status *status, MPI_Datatype datatype, int *count)`
Fortran	`MPI_GET_COUNT(status, datatype, count, ierr)`

INTEGER status(MPI_STATUS_SIZE) : 수신된 메시지의 상태 (IN)
INTEGER datatype : 각 원소의 데이터 타입 (IN)
INTEGER count : 원소의 개수 (OUT)

SEND RECV Example : C

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

int main(void)
{
   int rank, i, count;
   float data[100], value[200];
   MPI_Status status;
   MPI_Init(NULL,NULL);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   if(rank==0) {
      for(i=0;i<100;++i) data[i]=i;
      MPI_Send(data,100,MPI_FLOAT, 1, 55, MPI_COMM_WORLD);
   }
   else if(rank==1){
      MPI_Recv(value,200,MPI_FLOAT,MPI_ANY_SOURCE, 55, MPI_COMM_WORLD, &status);
      printf("P:%d Got data from processor %d \n",rank, status.MPI_SOURCE);
      MPI_Get_count(&status,MPI_FLOAT,&count);
      printf("P:%d Got %d elements \n",rank,count);
      printf("P:%d value[5]=%f \n",rank,value[5]);
   }
   MPI_Finalize();
   return 0;
}
```

```
$ mpiicc send_recv.c -o send_recv.x
$ mpirun -np 2 ./send_recv.x
P:0 Got data from processor 1
P:0 Got 100 elements
P:0 value[5]=5.000000
```

SEND RECV Example : Fortran

```
PROGRAM send_recv
USE mpi_f08
IMPLICIT NONE
INTEGER::rank,size,count,k
REAL:: data(100), value(200)
TYPE(MPI_STATUS)::status
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,rank)
IF(rank==0) THEN
   data = (/(k, k=1,100)/)
   CALL MPI_SEND(data,100,MPI_REAL,1,55,MPI_COMM_WORLD)
ELSEIF(rank==1)THEN
   CALL MPI_RECV(value,200,MPI_REAL,MPI_ANY_SOURCE,55,MPI_COMM_WORLD,status)
   PRINT*,"p:",rank," got data from processor ",status%MPI_SOURCE
   CALL MPI_GET_COUNT(status,MPI_REAL,count)
   PRINT*,"P:",rank," got ",count," elements"
   PRINT*,"P:",rank," value(5)=",value(5)
ENDIF
CALL MPI_Finalize
END PROGRAM send_recv
```

```
$ mpiifort send_recv.f90 -o send_recv.x
$ mpirun -np 2 ./send_recv.x
 p:           1  got data from processor          0
 P:           1  got          100  elements
 P:           1  value(5)=   5.00000000
```


논블록킹 통신

- 통신을 세 가지 상태로 분류
  1. **논블록킹 통신의 초기화 : 송신 또는 수신의 포스팅**
  2. **전송 데이터를 사용하지 않는 다른 작업 수행**
     - 통신과 계산 작업을 동시 수행
  3. **통신 완료 : 대기 또는 검사**
- 교착 가능성 제거, 통신 부하 감소

논블록킹 통신 초기화

C	`int MPI_ISend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)`
Fortran	`MPI_ISEND(buf, count, datatype, dest, tag, comm, request, ierr)`

C	`int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)`
Fortran	`MPI_IRECV(buf, count, datatype, source, tag, comm, request, ierr)`

INTEGER request : 초기화된 통신의 식별에 이용 (핸들) (OUT)

논블록킹 수신에는 status 인수가 없음

논블록킹 통신의 완료

- 대기**(waiting)** 또는 검사**(testing)**
  - 대기
    - 루틴이 호출되면 통신이 완료될 때까지 프로세스를 블록킹
    - 논블록킹 통신 + 대기 = 블록킹 통신
  - 검사
    - 루틴은 통신의 완료 여부에 따라 참 또는 거짓 리턴

대기

C	`int MPI_Wait(MPI_Request *request, MPI_Status *status)`
Fortran	`MPI_WAIT(request, status, ierr)`

INTEGER request : 포스팅된 통신의 식별에 이용 (핸들) (IN)

INTEGER status(MPI_STATUS_SIZE) : 수신 메시지에 대한 정보 또는 송신 루틴에 대한 에러코드 (OUT)

검사

C	`int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)`
Fortran	`MPI_TEST(request, flag, status, ierr)`

INTEGER request : 포스팅된 통신의 식별에 이용 (핸들) (IN)

LOGICAL flag : 통신이 완료되면 참, 아니면 거짓을 리턴 (OUT)

INTEGER status(MPI_STATUS_SIZE) : 수신 메시지에 대한 정보 또는 송신 루틴에 대한 에러코드 (OUT)

Non-blocking SEND RECV Example : C

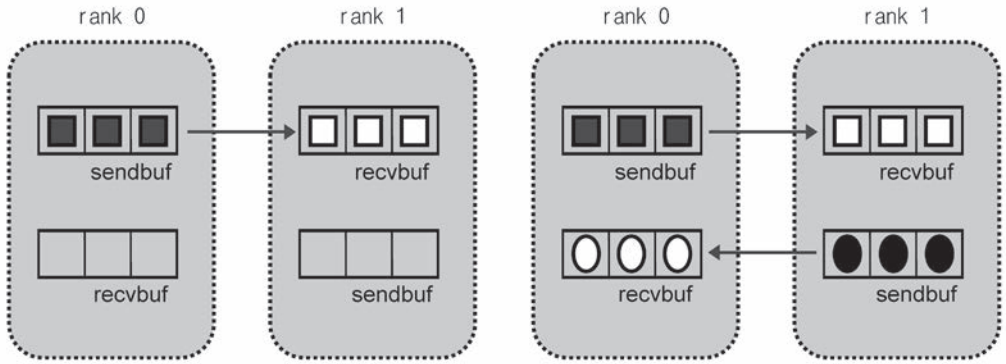
```c
#include <stdio.h>
#include "mpi.h"

int main(void)
{
   int rank, i, count;
   float data[100], value[200];
   MPI_Status status; MPI_Request req;
   MPI_Init(NULL,NULL);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   if(rank==0) {
      for(i=0;i<100;++i) data[i]=i;
      MPI_Isend(data,100,MPI_FLOAT, 1, 55, MPI_COMM_WORLD,&req);
   }
   else if(rank==1)
      MPI_Irecv(value,200,MPI_FLOAT,MPI_ANY_SOURCE, 55, MPI_COMM_WORLD, &req);
   MPI_Wait(&req,&status)
   if(rank==1){
      printf("P:%d Got data from processor %d \n",rank, status.MPI_SOURCE);
      MPI_Get_count(&status,MPI_FLOAT,&count);
      printf("P:%d Got %d elements \n",rank,count);
      printf("P:%d value[5]=%f \n",rank,value[5]);
   }
   MPI_Finalize();   return 0;
}
```


Non-blocking SEND RECV Example : Fortran

```fortran
PROGRAM send_recv
USE mpi_f08
IMPLICIT NONE
INTEGER::rank,size,count,k
REAL:: data(100), value(200)
TYPE(MPI_STATUS)::status
TYPE(MPI_REQUEST):: req
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,rank)
IF(rank==0) THEN
   data = (/(k, k=1,100)/)
   CALL MPI_ISEND(data,100,MPI_REAL,1,55,MPI_COMM_WORLD,req)
ELSEIF(rank==1)THEN
   CALL MPI_IRECV(value,200,MPI_REAL,MPI_ANY_SOURCE,55,MPI_COMM_WORLD,req)
ENDIF
CALL MPI_WAIT(req,status)
if(rank==1) then
   PRINT*,"p:",rank," got data from processor ",status%MPI_SOURCE
   CALL MPI_GET_COUNT(status,MPI_REAL,count)
   PRINT*,"P:",rank," got ",count," elements"
   PRINT*,"P:",rank," value(5)=",value(5)
endif
CALL MPI_Finalize
END PROGRAM send_recv
```

점대점 통신의 사용

- 단방향 통신과 양방향 통신
- 양방향 통신은 교착**(deadlock)**에 주의



양방향 통신

- 선 송신, 후 수신 : 교착 가능

```
IF (myrank==0) THEN
   CALL MPI_SEND(sendbuf, ...)
   CALL MPI_RECV(recvbuf, ...)
ELSEIF (myrank==1) THEN
   CALL MPI_SEND(sendbuf, ...)
   CALL MPI_RECV(recvbuf, ...)
ENDIF
```

Deadlock_blocking : C

```c
#include <stdio.h>
#include "mpi.h"
#define BUF_SIZE   (1024*MAX)

int main()
{
  int nrank,i;
  double a[BUF_SIZE],b[BUF_SIZE];
  MPI_Init(NULL,NULL);
  MPI_Comm_rank(MPI_COMM_WORLD,&nrank);
  if(nrank==0) {
    MPI_Send(a,BUF_SIZE,MPI_DOUBLE, 1,17,MPI_COMM_WORLD);
    MPI_Recv(b,BUF_SIZE,MPI_DOUBLE, 1,19,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
  }
  else if(nrank==1){
    MPI_Send(a,BUF_SIZE,MPI_DOUBLE, 0,19,MPI_COMM_WORLD);
    MPI_Recv(b,BUF_SIZE,MPI_DOUBLE, 0,17,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
  }
  MPI_Finalize();
  return 0;
}
```

```
$ mpiicc -DMAX=24 (or 25) deadlock.c -o deadlock.x   (on the KNL intra node)
$ mpirun -np 2 ./deadlock.x
```


Deadlock_blocking : Fortran

```
PROGRAM deadlock
USE mpi_f08
INTEGER::nrank,nprocs,i
INTEGER,PARAMETER::buf_size=1024*MAX

REAL(KIND=8),DIMENSION(buf_size)::a,b
CALL MPI_Init
CALL MPI_Comm_rank(MPI_COMM_WORLD,nrank)
IF(nrank==0)THEN
  CALL MPI_Send(a,buf_size,MPI_REAL8,1,17,MPI_COMM_WORLD)
  CALL MPI_Recv(b,buf_size,MPI_REAL8,1,19,MPI_COMM_WORLD,MPI_STATUS_IGNORE)
ELSEIF(nrank==1)THEN
  CALL MPI_send(a,buf_size,MPI_REAL8,0,19,MPI_COMM_WORLD)
  CALL MPI_Recv(b,buf_size,MPI_REAL8,0,17,MPI_COMM_WORLD,MPI_STATUS_IGNORE)
ENDIF
CALL MPI_Finalize
END PROGRAM deadlock
```

```
$ mpiifort -fpp -DMAX=24 (or 25) deadlock.f90 -o deadlock.x (on the KNL intra
node)
$ mpirun -np 2 ./deadlock.x
```

양방향 통신

- 한쪽은 송신부터, 다른 한쪽은 수신부터
  : 블록킹, 논블록킹 루틴의 사용과 무관하게 교착 없음

```
IF (myrank==0) THEN
   CALL MPI_SEND(sendbuf, ...)
   CALL MPI_RECV(recvbuf, …)
ELSEIF (myrank==1) THEN
   CALL MPI_RECV(recvbuf, …)
   CALL MPI_SEND(sendbuf, ...)
ENDIF
```


양방향 통신

- 권장 코드

```
IF (myrank==0) THEN
   CALL MPI_ISEND(sendbuf, …, ireq1, …)
   CALL MPI_IRECV(recvbuf, …, ireq2, …)
ELSEIF (myrank==1) THEN
   CALL MPI_ISEND(sendbuf, …, ireq1, …)
   CALL MPI_IRECV(recvbuf, …, ireq2, …)
ENDIF
CALL MPI_WAIT(ireq1, …)
CALL MPI_WAIT(ireq2, …)
```

MPI Functions

- P2P Communications
- **Collective Communications**
- Derived Data Types

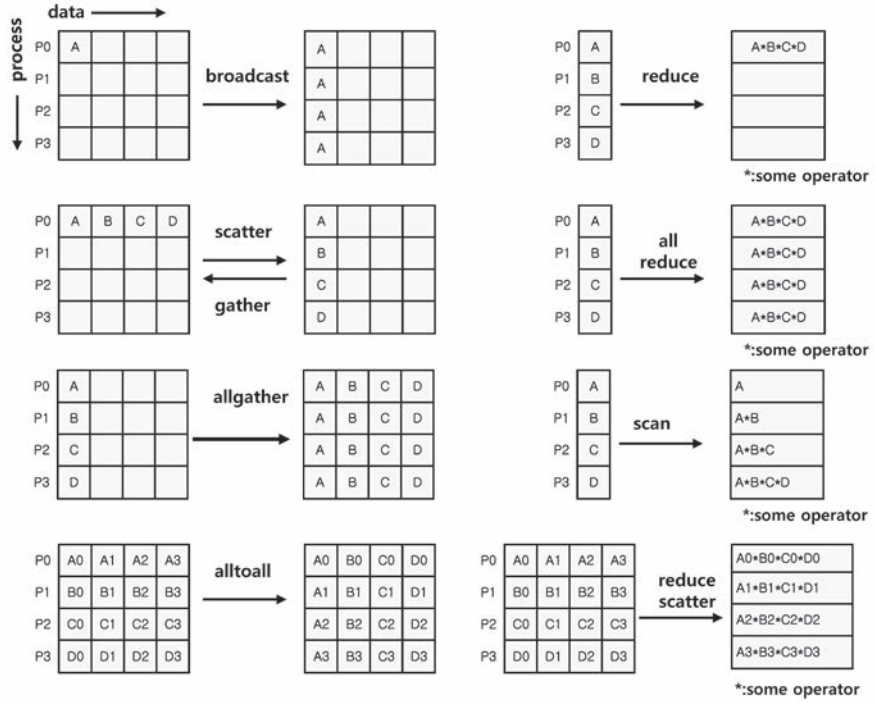
집합 통신

- 한 그룹의 프로세스가 참여하는 통신
- 점대점 통신 기반
- 점대점 통신을 이용한 구현보다 편리하고 성능면에서 유리
- 집합 통신 루틴
  - 커뮤니케이터 내의 모든 프로세스에서 호출
  - 동기화가 보장되지 않음 (MPI_Barrier 제외)
  - 꼬리표 없음

집합 통신

Category	Subroutines
One buffer	**MPI_BCAST**
One send buffer and one receive buffer	**MPI_GATHER**, MPI_SCATTER, **MPI_ALLGATHER**, MPI_ALLTOALL, **MPI_GATHERV**, MPI_SCATTERV, **MPI_ALLGATHERV**, MPI_ALLTOALLV
Reduction	**MPI_REDUCE, MPI_ALLREDUCE**, MPI_SCAN, MPI_REDUCE_SCATTER
Others	MPI_BARRIER, MPI_OP_CREATE, MPI_OP_FREE

집합 통신



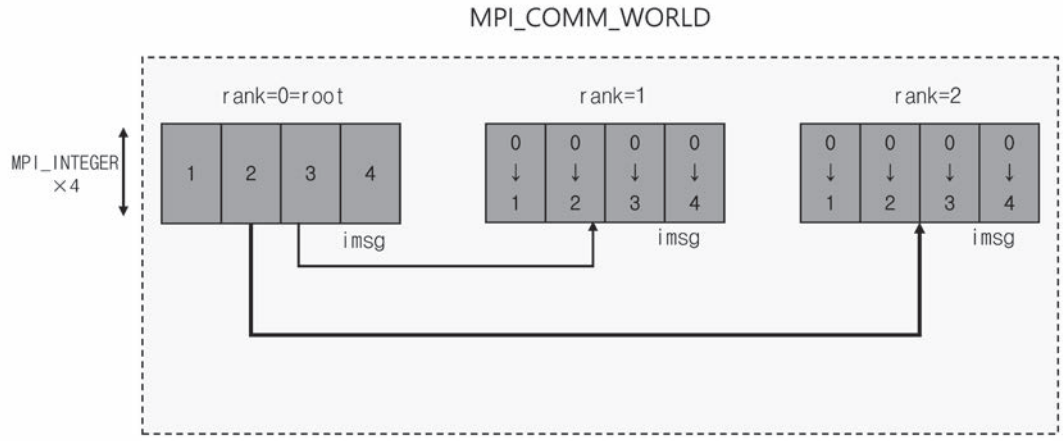
방송 : MPI_BCAST

C	`int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)`
Fortran	`MPI_BCAST(buffer, count, datatype, root, comm, ierr)`

(CHOICE) buffer : 버퍼의 시작 주소 (INOUT)
INTEGER count : 버퍼 원소의 개수 (IN)
INTEGER datatype : 버퍼 원소의 MPI 데이터 타입 (IN)
INTEGER root : 루트 프로세스의 랭크 (IN)
INTEGER comm : 커뮤니케이터 (IN)

➢ 루트 프로세스로부터 커뮤니케이터 내의 다른 프로세스로 동일한 데이터를 전송 : 일대다 통신

MPI_BCAST 예제



MPI_BCAST 예제 : C

```c
/*broadcast*/
#include <mpi.h>
#include <stdio.h>
void main (int argc, char *argv[]){
  int i, myrank ;
  int imsg[4];
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  if (myrank==0) for(i=0; i<4; i++) imsg[i] = i+1;
  else for (i=0; i<4; i++) imsg[i] = 0;
  printf("%d: BEFORE:", myrank);
  for(i=0; i<4; i++) printf(" %d", imsg[i]);
  printf("\n");
  MPI_Bcast(imsg, 4, MPI_INT, 0, MPI_COMM_WORLD);
  printf("%d: AFTER:", myrank);
  for(i=0; i<4; i++) printf(" %d", imsg[i]); printf("\n");
  MPI_Finalize();
}
```


MPI_BCAST 예제 : Fortran

```fortran
PROGRAM bcast
INCLUDE 'mpif.h'
INTEGER imsg(4)
CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
If (myrank ==0) then
   imsg = (/(i,i=1,4)/)
else
   imsg = 0
Endif
PRINT*, myrank,':Before:',imsg
CALL MPI_BCAST(imsg, 4, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
PRINT*,myrank,':After :',imsg
CALL MPI_FINALIZE(ierr)
END
```

취합 : MPI_GATHER

C	`int MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`
Fortran	`MPI_GATHER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, ierr)`

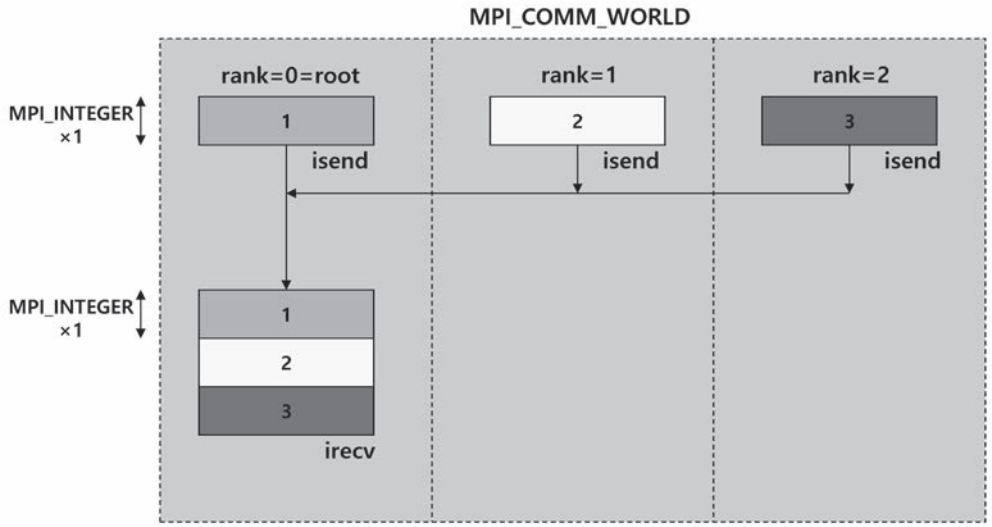
(CHOICE) sendbuf : 송신 버퍼의 시작 주소 (IN)
INTEGER sendcount : 송신 버퍼의 원소 개수 (IN)
INTEGER sendtype : 송신 버퍼 원소의 MPI 데이터 타입 (IN)
(CHOICE) recvbuf : 수신 버퍼의 주소 (OUT)
INTEGER recvcount : 수신할 원소의 개수 (IN)
INTEGER recvtype : 수신 버퍼 원소의 MPI 데이터 타입 (IN)
INTEGER root : 수신 프로세스(루트 프로세스)의 랭크 (IN)
INTEGER comm : 커뮤니케이터 (IN)

- 모든 프로세스(루트 포함)가 송신한 데이터를 취합하여 랭크 순서대로 저장 : 다대일 통신

MPI_GATHER : 주의 사항

- 송신 버퍼(**sendbuf**)와 수신 버퍼(**recvbuf**)의 메모리 위치가 겹쳐지지 않도록 주의할 것. 즉, 같은 이름을 쓰면 안됨
  ➔ 송신 버퍼와 수신 버퍼를 이용하는 모든 집합 통신에 해당
- 전송되는 데이터의 크기는 모두 동일할 것
- 크기가 서로 다른 데이터의 취합 ➔ **MPI_GATHERV**

MPI_GATHER 예제



MPI_GATHER 예제 : C

```
/*gather*/
#include <mpi.h>
#include <stdio.h>
void main (int argc, char *argv[]){
  int i, nprocs, myrank ;
  int isend, irecv[3];
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  isend = myrank + 1;
  MPI_Gather(&isend,1,MPI_INT,irecv,1,MPI_INT,0,
             MPI_COMM_WORLD);
  if(myrank == 0) {
      printf(" irecv = ");
      for(i=0; i<3; i++)
         printf(" %d", irecv[i]); printf("\n");
  }
  MPI_Finalize();
}
```

MPI_GATHER 예제 : Fortran

```
PROGRAM gather
INCLUDE 'mpif.h'
INTEGER irecv(3)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
isend = myrank + 1
CALL MPI_GATHER(isend,1,MPI_INTEGER,irecv,1,MPI_INTEGER,&
     0, MPI_COMM_WORLD, ierr)
IF (myrank==0) THEN
   PRINT *, 'irecv =',irecv
ENDIF
CALL MPI_FINALIZE(ierr)
END
```


취합 : MPI_GATHERV

C	`int MPI_Gatherv(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcounts, int displs, MPI_Datatype recvtype, int root, MPI_Comm comm)`
Fortran	`MPI_GATHERV(sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, root, comm, ierr)`

…

(CHOICE) recvbuf : 수신버퍼의 주소(OUT)

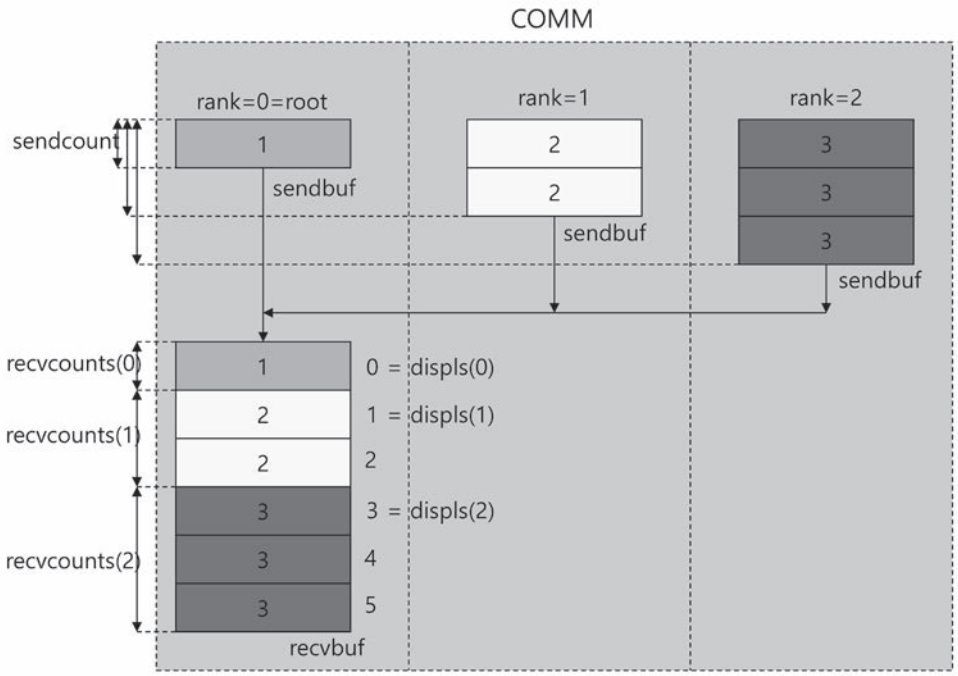
INTEGER recvcounts(*) : 수신된 원소의 개수를 저장하는 정수 배열(IN)

INTEGER displs(*) : 정수 배열, i번째 자리에는 프로세스 i에서 들어오는 데이터가 저장될 수신 버퍼 상의 위치를 나타냄(IN)

…

- 각 프로세스로부터 전송되는 데이터의 크기가 다를 경우 사용
- 서로 다른 메시지 크기는 배열 **recvcounts**에 지정, 배열 **displs**에 루트 프로세스의 어디에 데이터가 위치하게 되는가를 저장

MPI_GATHERV 예제



MPI_GATHERV 예제 : C

```
/*gatherv*/
#include <mpi.h>
#include <stdio.h>
void main (int argc, char *argv[]){
  int i, myrank ;
  int isend[3], irecv[6];
  int iscnt, ircnt[3]={1,2,3}, idisp[3]={0,1,3};
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  for(i=0; i<myrank+1; i++) isend[i] = myrank + 1;
  iscnt = myrank +1;
  MPI_Gatherv(isend, iscnt, MPI_INT, irecv, ircnt, idisp,
              MPI_INT, 0, MPI_COMM_WORLD);
  if(myrank == 0) {
      printf(" irecv = "); for(i=0; i<6; i++) printf(" %d", irecv[i]);
      printf("\n");
  }
  MPI_Finalize();
}
```

MPI_GATHERV 예제 : Fortran

```
PROGRAM gatherv
INCLUDE 'mpif.h'
INTEGER isend(3), irecv(6)
INTEGER ircnt(0:2), idisp(0:2)
DATA ircnt/1,2,3/ idisp/0,1,3/
CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
DO i=1,myrank+1
   isend(i) = myrank + 1
ENDDO
iscnt = myrank + 1
CALL MPI_GATHERV(isend,iscnt,MPI_INTEGER,irecv,ircnt,idisp,&
   MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
IF (myrank==0) THEN
   PRINT *,'irecv =',irecv
ENDIF
CALL MPI_FINALIZE(ierr)
END
```

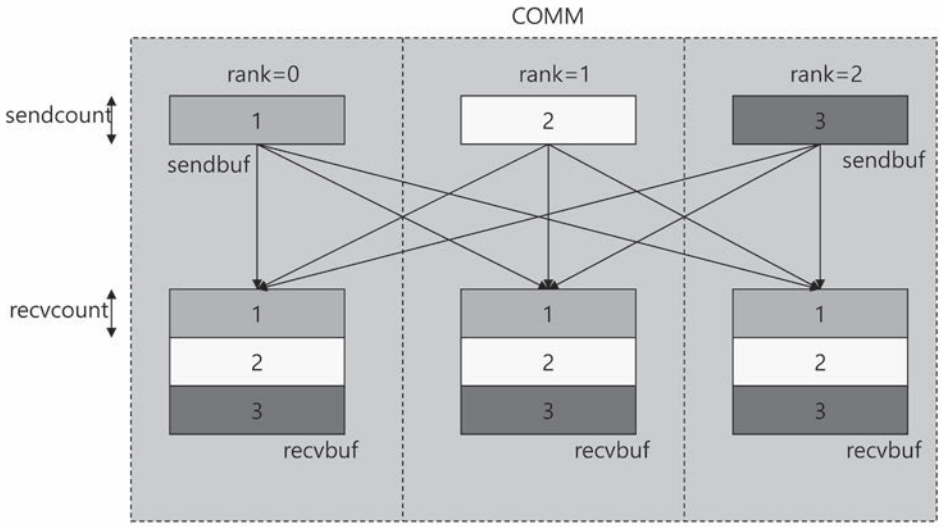

취합 : MPI_ALLGATHER

C	`int MPI_Allgather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)`
Fortran	`MPI_ALLGATHER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, ierr)`

(CHOICE) sendbuf : 송신 버퍼의 시작 주소(IN)
INTEGER sendcount : 송신 버퍼의 원소 개수(IN)
INTEGER sendtype : 송신 버퍼 원소의 MPI 데이터 타입(IN)
(CHOICE) recvbuf : 수신 버퍼의 주소(OUT)
INTEGER recvcount : 각 프로세스로부터 수신된 데이터 개수(IN)
INTEGER recvtype : 수신버퍼 데이터 타입(IN)
INTEGER comm : 커뮤니케이터 (IN)

- **MPI_GATHER + MPI_BCAST**
- 프로세스 **j** 의 데이터 ➔ 모든 수신 버퍼 **j** 번째 블록에 저장

MPI_ALLGATHER 예제



MPI_ALLGATHER 예제 : C

```
/*allgather*/
#include <mpi.h>
#include <stdio.h>
void main (int argc, char *argv[]){
  int i, myrank ;
  int isend, irecv[3];
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  isend = myrank + 1;
  MPI_Allgather(&isend, 1, MPI_INT, irecv, 1,
                MPI_INT, MPI_COMM_WORLD);
  printf("%d irecv = ");
  for(i=0; i<3; i++) printf(" %d", irecv[i]);
  printf("\n");
  MPI_Finalize();
}
```

MPI_ALLGATHER 예제 : Fortran

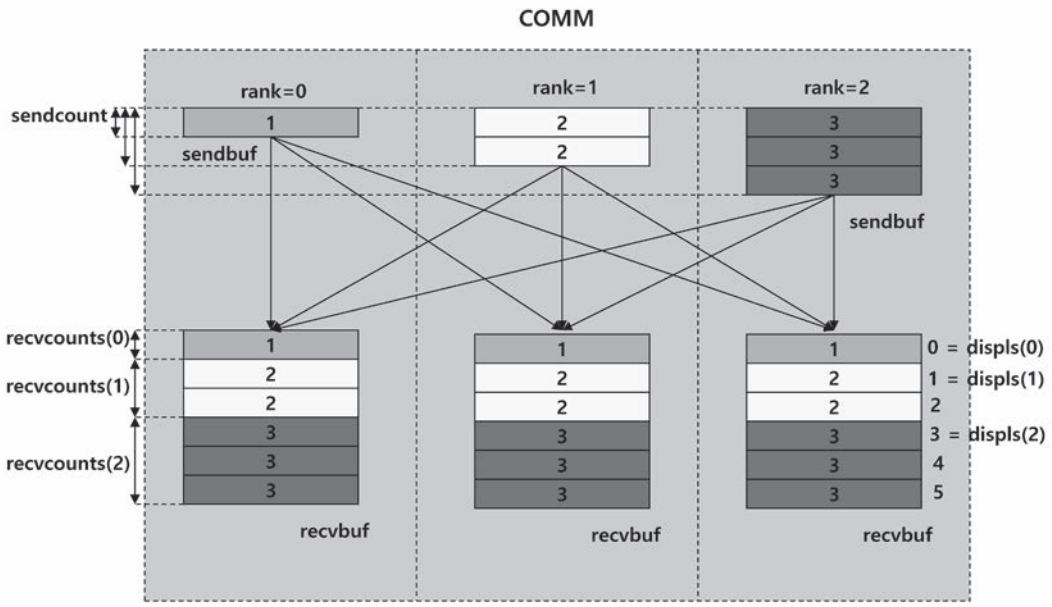
```
PROGRAM allgather
INCLUDE 'mpif.h'
INTEGER irecv(3)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
isend = myrank + 1
CALL MPI_ALLGATHER(isend, 1, MPI_INTEGER, &
 irecv, 1, MPI_INTEGER, MPI_COMM_WORLD, ierr)
PRINT *,'irecv =',irecv
CALL MPI_FINALIZE(ierr)
END
```


취합 : MPI_ALLGATHERV

C	`int MPI_Allgatherv(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcounts, int displs, MPI_Datatype recvtype, MPI_Comm comm)`
Fortran	`MPI_ALLGATHERV(sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, comm, ierr)`

- **MPI_ALLGATHER**와 같은 기능을 하며 서로 다른 크기의 데이터를 취합할 때 사용

MPI_ALLGATHERV 예제



환산 : MPI_REDUCE

C	`int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)`
Fortran	`MPI_REDUCE(sendbuf, recvbuf, count, datatype, op, root, comm, ierr)`

(CHOICE) sendbuf : 송신 버퍼의 시작 주소 (IN)
(CHOICE) recvbuf : 수신 버퍼의 주소 (OUT)
INTEGER count : 송신 버퍼의 원소 개수 (IN)
INTEGER datatype : 송신 버퍼 원소의 MPI 데이터 타입 (IN)
INTEGER op : 환산 연산자 (IN)
INTEGER root : 루트 프로세스의 랭크 (IN)
INTEGER comm : 커뮤니케이터 (IN)

- 각 프로세스로부터 데이터를 모아 하나의 값으로 환산,그 결과를 루트 프로세스에 저장

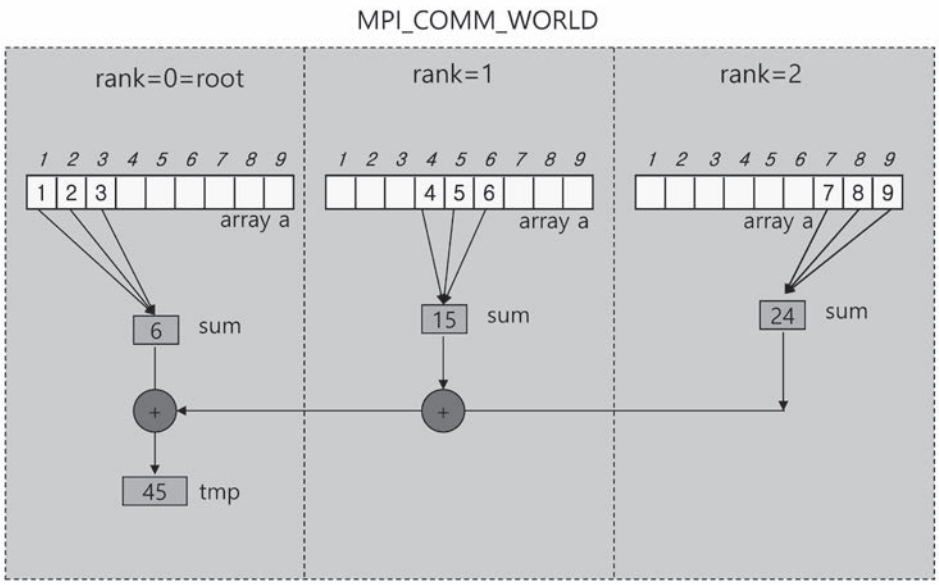
MPI_REDUCE : 연산과 데이터 타입

Operation	Data Type (Fortran)
MPI_SUM(sum), MPI_PROD(product)	MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION, MPI_COMPLEX
MPI_MAX(maximum), MPI_MIN(minimum)	MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION
MPI_MAXLOC(max value and location), MPI_MINLOC(min value and location)	MPI_2INTEGER, MPI_2REAL, MPI_2DOUBLE_PRECISION
MPI_LAND(logical AND), MPI_LOR(logical OR), MPI_LXOR(logical XOR)	MPI_LOGICAL
MPI_BAND(bitwise AND), MPI_BOR(bitwise OR), MPI_BXOR(bitwise XOR)	MPI_INTEGER, MPI_BYTE

MPI_REDUCE : 연산과 데이터 타입

Operation	Data Type (C)
MPI_SUM(sum), MPI_PROD(product) MPI_MAX(maximum), MPI_MIN(minimum)	MPI_INT, MPI_LONG, MPI_SHORT, MPI_UNSIGNED_SHORT, MPI_UNSIGNED MPI_UNSIGNED_LONG, MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE
MPI_MAXLOC(max value and location), MPI_MINLOC(min value and location)	MPI_FLOAT_INT, MPI_DOUBLE_INT, MPI_LONG_INT, MPI_2INT, MPI_SHORT_INT, MPI_LONG_DOUBLE_INT
MPI_LAND(logical AND), MPI_LOR(logical OR), MPI_LXOR(logical XOR)	MPI_INT, MPI_LONG, MPI_SHORT, MPI_UNSIGNED_SHORT, MPI_UNSIGNED MPI_UNSIGNED_LONG
MPI_BAND(bitwise AND), MPI_BOR(bitwise OR), MPI_BXOR(bitwise XOR)	MPI_INT, MPI_LONG, MPI_SHORT, MPI_UNSIGNED_SHORT, MPI_UNSIGNED MPI_UNSIGNED_LONG, MPI_BYTE

MPI_REDUCE 예제



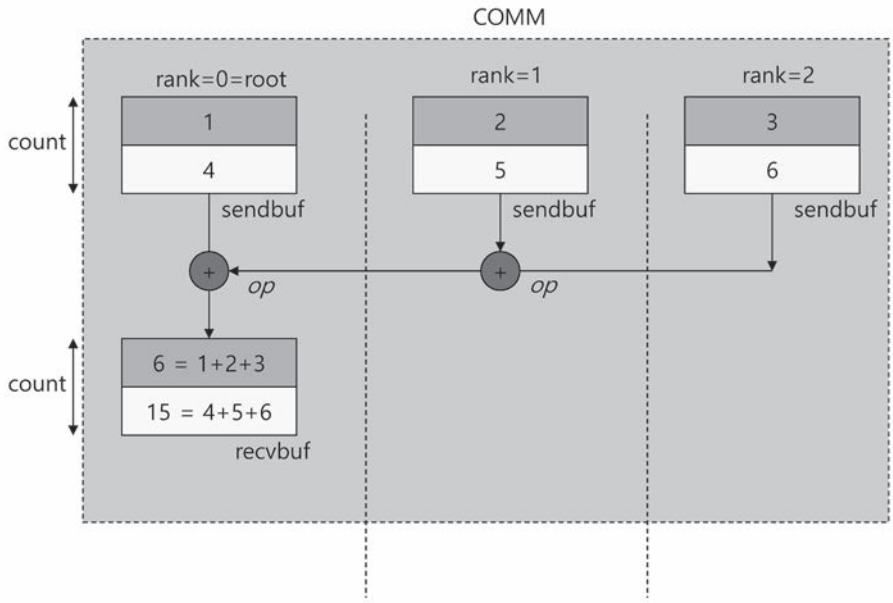
MPI_REDUCE 예제 : C

```
/*reduce*/
#include <mpi.h>
#include <stdio.h>
void main (int argc, char *argv[]){
  int i, myrank, ista, iend;
  double a[9], sum, tsum;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  ista = myrank*3 ;
  iend = ista + 2;
  for(i = ista; i<iend+1; i++)  a[i] = i+1;
  sum = 0.0;
  for(i = ista; i<iend+1; i++)
     sum = sum + a[i];
  MPI_Reduce(&sum,&tsum,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
  if(myrank == 0) printf(" sum = %f \n", tsum);
  MPI_Finalize();
}
```

MPI_REDUCE 예제 : Fortran

```
PROGRAM reduce
INCLUDE 'mpif.h'
REAL a(9),sum, tsum
CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
Ista = myrank * 3 + 1
Iend = ista + 2
DO i=ista,iend
   a(i) = I
ENDDO
Sum = 0.0
DO i=ista,iend
   sum = sum + a(i)
ENDDO
CALL MPI_REDUCE(sum,tsum,1,MPI_REAL,MPI_SUM,0,MPI_COMM_WORLD,ierr)
IF (myrank==0) THEN
   PRINT *,'sum =',tsum
ENDIF
CALL MPI_FINALIZE(ierr)
END
```


MPI_REDUCE 예제 : 배열

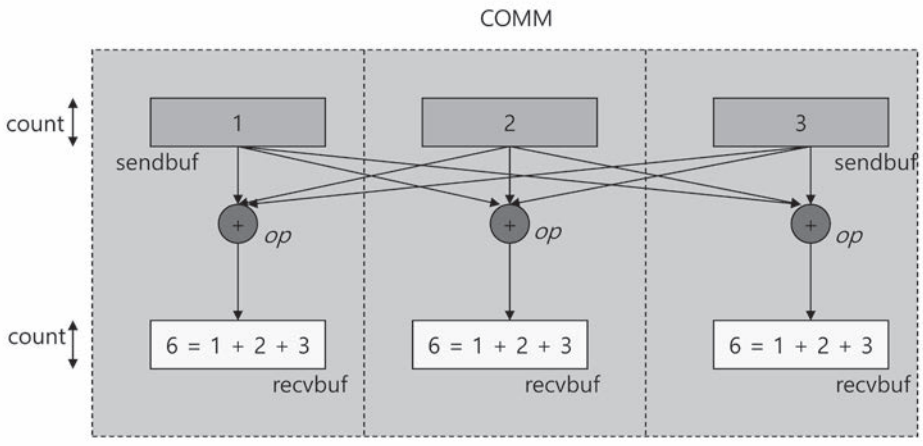


환산 : MPI_ALLREDUCE

C	`int MPI_Allreduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)`
Fortran	`MPI_ALLREDUCE(sendbuf, recvbuf, count, datatype, op, comm, ierr)`

- 각 프로세스로부터 데이터를 모아 하나의 값으로 환산,그 결과를 모든 프로세스에 저장

MPI_ALLREDUCE 예제



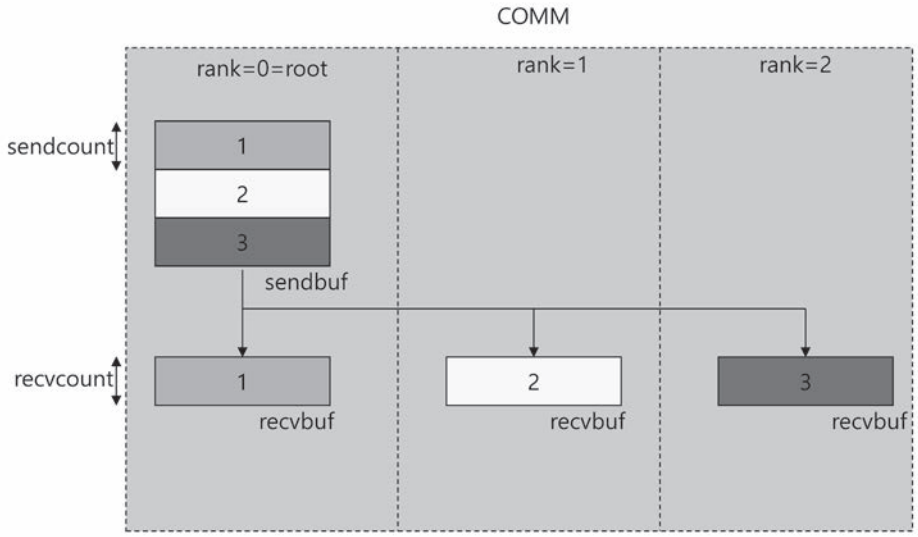
확산 : MPI_SCATTER

C	`int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`
Fortran	`MPI_SCATTER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, ierr)`

(CHOICE) sendbuf : 송신 버퍼의 주소 (IN)
INTEGER sendcount : 각 프로세스로 보내지는 원소 개수 (IN)
INTEGER sendtype : 송신 버퍼 원소의 MPI 데이터 타입 (IN)
(CHOICE) recvbuf : 수신 버퍼의 주소 (OUT)
INTEGER recvcount : 수신 버퍼의 원소 개수 (IN)
INTEGER recvtype : 수신 버퍼의 MPI 데이터 타입 (IN)
INTEGER root : 송신 프로세스의 랭크 (IN)
INTEGER comm : 커뮤니케이터 (IN)

- 루트 프로세스는 데이터를 같은 크기로 나누어 각 프로세스에 랭크 순서대로 하나씩 전송

MPI_SCATTER 예제



MPI_SCATTER 예제 : Fortran

```
PROGRAM scatter
INCLUDE 'mpif.h'
INTEGER isend(3)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
IF (myrank==0) THEN
   DO i=1,nprocs
      isend(i)=i
   ENDDO
ENDIF
CALL MPI_SCATTER(isend, 1, MPI_INTEGER, irecv, 1, MPI_INTEGER, &
     0, MPI_COMM_WORLD, ierr)
PRINT *,'irecv =',irecv
CALL MPI_FINALIZE(ierr)
END
```


MPI_SCATTER 예제 : C

```
/*scatter*/
#include <mpi.h>
#include <stdio.h>
void main (int argc, char *argv[]){
  int i, myrank, nprocs;
  int isend[3], irecv;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  for(i=0; i<nprocs; i++) isend[i]=i+1;
  MPI_Scatter(isend, 1, MPI_INTEGER, &irecv, 1,
              MPI_INT, 0, MPI_COMM_WORLD);
  printf(" %d: irecv = %d\n", myrank, irecv);
  MPI_Finalize();
}
```

확산 : MPI_SCATTERV

C	`int MPI_Scatterv(void *sendbuf, int sendcounts, int displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`
Fortran	`MPI_SCATTERV(sendbuf, sendcounts, displs, sendtype, recvbuf, recvcount, recvtype, root, comm, ierr)`

(CHOICE) sendbuf : 송신 버퍼의 주소 (IN)

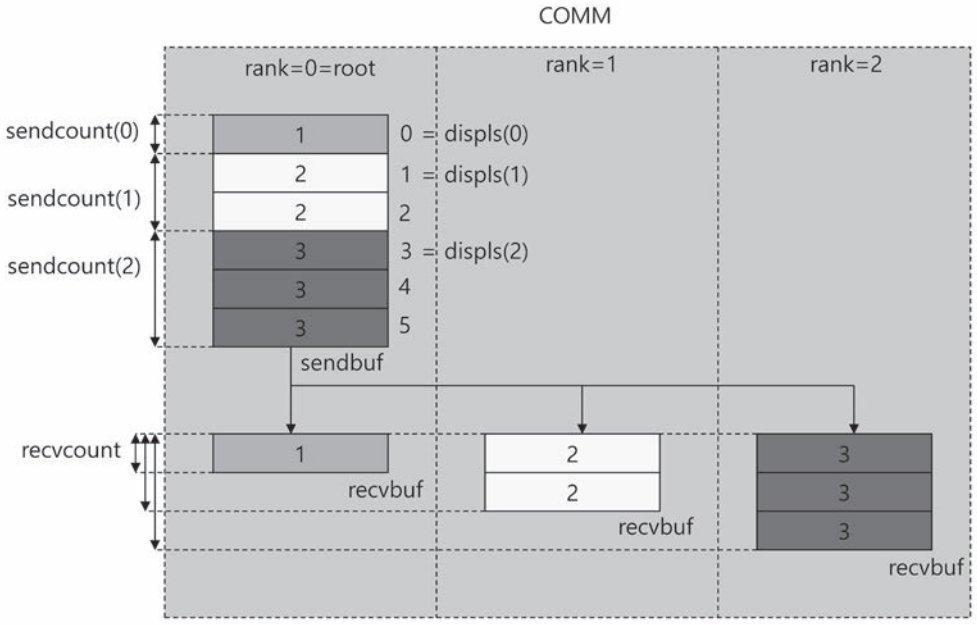
INTEGER sendcounts(*) : 정수 배열, i번째 자리에 프로세스 i 로 전송될 데이터 개수 저장(IN)

INTEGER displs(*) : 정수 배열, i번째 자리에 프로세스 i로 전송될 데이터의 송신 버퍼 상의 상대적 위치가 저장됨 (IN)

…

➢ 루트 프로세스는 데이터를 서로 다른 크기로 나누어 각 프로세스에 랭크 순서대로 하나씩 전송

MPI_SCATTERV

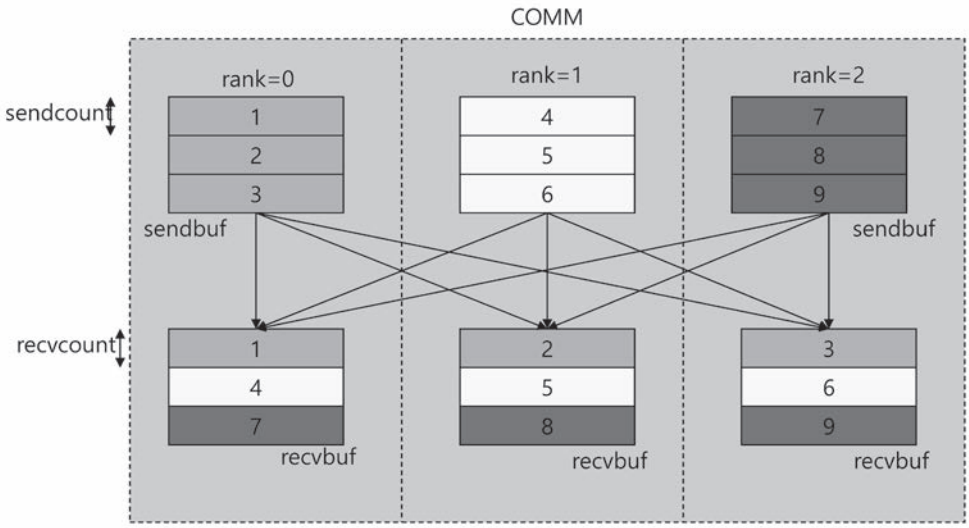


기타 : MPI_ALLTOALL

C	`int MPI_Alltoall(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)`
Fortran	`MPI_ALLTOALL(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, ierr)`

- 각 프로세스로부터 모든 프로세스에 동일한 크기의 개별적인 메시지를 전달
- 프로세스 **i** 로부터 송신되는 **j** 번째 데이터 블록은 프로세스 **j**가 받아 수신버퍼의 **i**번째 블록에 저장

MPI_ALLTOALL

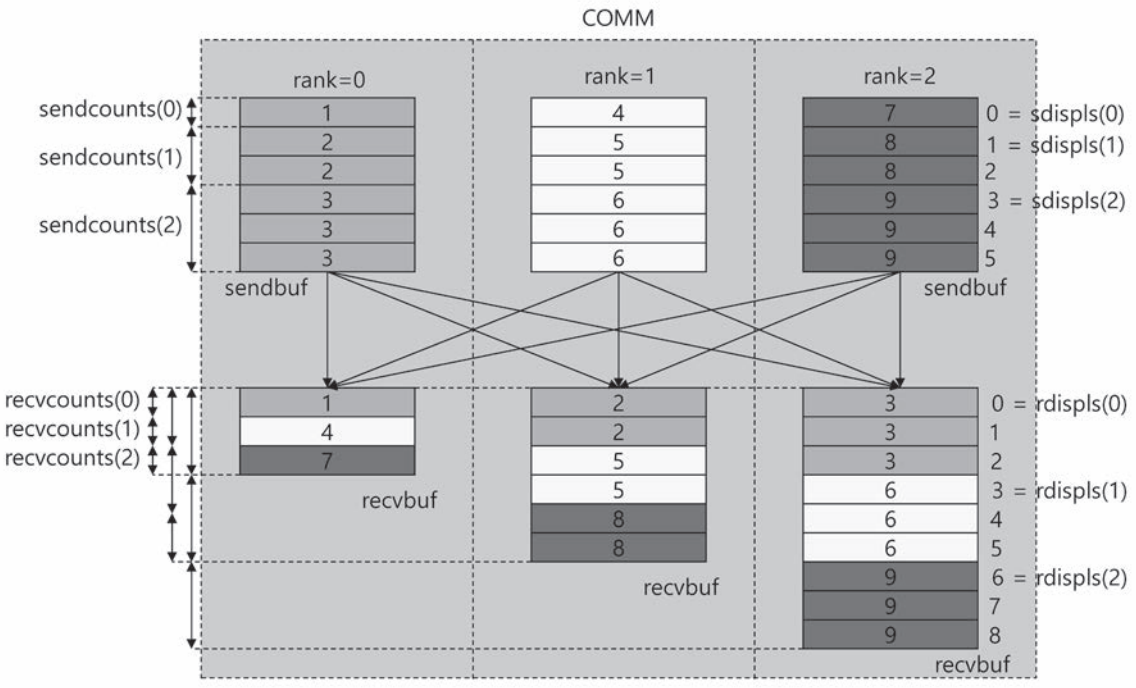


기타 : MPI_ALLTOALLV

C	`int MPI_Alltoallv(void *sendbuf, int sendcounts, int sdispls, MPI_Datatype sendtype, void *recvbuf, int recvcounts, int rdispls, MPI_Datatype recvtype, MPI_Comm comm)`
Fortran	`MPI_ALLTOALLV(sendbuf, sendcounts, sdispls, sendtype, recvbuf, recvcounts, rdispls, recvtype, comm, ierr)`

- 각 프로세스로부터 모든 프로세스에 서로 다른 크기의 개별적인 메시지를 전달
- 프로세스 **i** 로부터 송신되는 **sendcounts(j)**개의 데이터는 프로세스 **j**가 받아 수신버퍼의 **rdispls(i)** 번째 위치부터 저장

MPI_ALLTOALLV



장벽 : MPI_BARRIER

C	`int MPI_Barrier(MPI_Comm comm)`
Fortran	`MPI_BARRIER(comm, ierr)`

- 커뮤니케이터 내의 모든 프로세스가 **MPI_BARRIER**를 호출할 때까지 더 이상의 프로세스 진행을 막음

MPI Functions

- P2P Communications
- Collective Communications
- **Derived Data Types**

유도 데이터 타입

- 데이터 타입이 다르거나 불연속적인 데이터의 전송
  - 동일한 데이터 타입을 가지는 불연속 데이터
  - 다른 데이터 타입을 가지는 연속 데이터
  - 다른 데이터 타입을 가지는 불연속 데이터

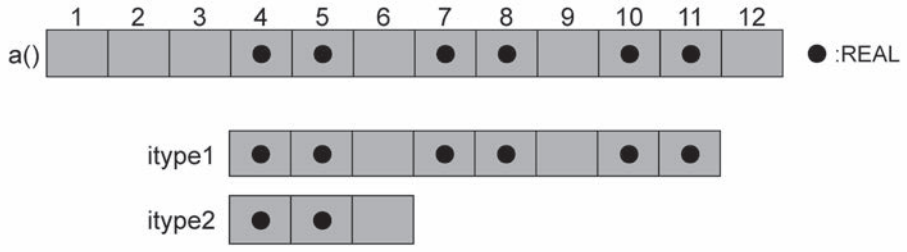
1. 각각을 따로 따로 전송

2. 새로운 버퍼로 묶어서 전송 후 묶음을 풀어 원위치로 저장 : **MPI_PACK/MPI_UNPACK, MPI_PACKED(**데이터 타입**)**

➔ 느린 속도, 불편, 오류의 위험

유도 데이터 타입

- **a(4), a(5), a(7), a(8), a(10), a(11)**의 전송



- **유도 데이터 타입 itype1, 한 개 전송**

```
CALL MPI_SEND(a(4), 1, itype1, idst, itag,
              MPI_COMM_WORLD, ierr)
```

- **유도 데이터 타입 itype2, 세 개 전송**

```
CALL MPI_SEND(a(4), 3, itype2, idst, itag,
              MPI_COMM_WORLD, ierr)
```

유도 데이터 타입의 사용

- **CONSTRUCT**
  - MPI 루틴 이용해 새로운 데이터 타입 작성
    - MPI_Type_contiguous
    - MPI_Type_(h)vector
    - MPI_Type_struct
- **COMMIT**
  - 작성된 데이터 타입 등록
    - MPI_Type_Commit
- **USE**
  - 송신, 수신 등에 새로운 데이터 타입 사용

MPI_TYPE_COMMIT

C	`int MPI_Type_commit (MPI_Datatype *datatype)`
Fortran	`MPI_TYPE_COMMIT (datatype, ierr)`

INTEGER datatype : 등록 데이터 타입(핸들) (INOUT)

- 새롭게 정의된 데이터 타입을 통신상에서 사용 가능하게 함
- 등록된 데이터 타입은 **MPI_TYPE_FREE**을 이용해 해제하기 전까지 계속 사용 가능

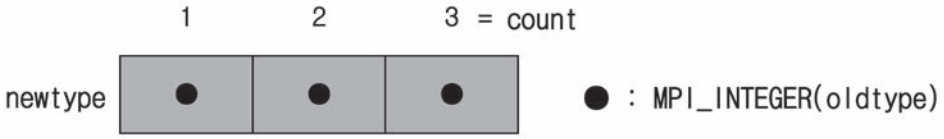
MPI_TYPE_CONTIGUOUS

C	`int MPI_Type_contiguous (int count, MPI_Datatype oldtype, MPI_Datatype *newtype)`
Fortran	`MPI_TYPE_CONTIGUOUS (count, oldtype, newtype, ierr)`

INTEGER count : 하나로 묶을 데이터 개수 (IN)
INTEGER oldtype : 이전 데이터 타입 (핸들) (IN)
INTEGER newtype : 새로운 데이터 타입 (핸들) (OUT)

- 같은 데이터 타입**(oldtype)**을 가지는 연속적인 데이터를 **count** 개 묶어 새로운 데이터 타입**(newtype)** 정의

MPI_TYPE_CONTIGUOUS 예제



MPI_TYPE_CONTIGUOUS 예제 : C

```
/*type_contiguous*/
#include <mpi.h>
#include <stdio.h>
void main (int argc, char *argv[]){
  int i, myrank, ibuf[20];
  MPI_Datatype inewtype ;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  if(myrank==0)  for(i=0; i<20; i++) ibuf[i]=i+1;
  else for(i=0; i<20; i++) ibuf[i]=0;
  MPI_Type_contiguous(3, MPI_INT, &inewtype);
  MPI_Type_commit(&inewtype);
  MPI_Bcast(ibuf, 3, inewtype, 0, MPI_COMM_WORLD);
  printf("%d : ibuf =", myrank);
  for(i=0; i<20; i++) printf(" %d", ibuf[i]);
  printf("\n");
  MPI_Finalize();
}
```


MPI_TYPE_CONTIGUOUS 예제 : Fortran

```
PROGRAM type_contiguous
INCLUDE 'mpif.h'
INTEGER ibuf(20)
INTEGER inewtype
CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
IF (myrank==0) THEN
   DO i=1,20
      ibuf(i) = i
   ENDDO
ENDIF
CALL MPI_TYPE_CONTIGUOUS(3, MPI_INTEGER, inewtype, ierr)
CALL MPI_TYPE_COMMIT(inewtype, ierr)
CALL MPI_BCAST(ibuf, 3, inewtype, 0, MPI_COMM_WORLD, ierr)
PRINT *,'ibuf =',ibuf
CALL MPI_FINALIZE(ierr)
END
```

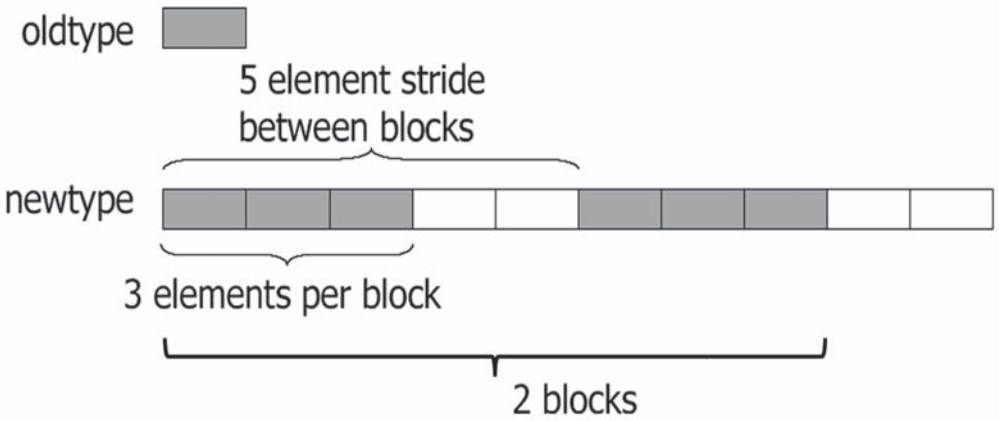
MPI_TYPE_VECTOR

C	`int MPI_Type_vector (int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype)`
Fortran	`MPI_TYPE_VECTOR (count, blocklength, stride, oldtype, newtype, ierr)`

INTEGER count : 블록의 개수(IN)
INTEGER blocklength : 각 블록의 oldtype 데이터의 개수(IN)
INTEGER stride : 인접한 두 블록의 시작점 사이의 폭(IN)
INTEGER oldtype : 이전 데이터 타입(핸들) (IN)
INTEGER newtype : 새로운 데이터 타입(핸들) (OUT)

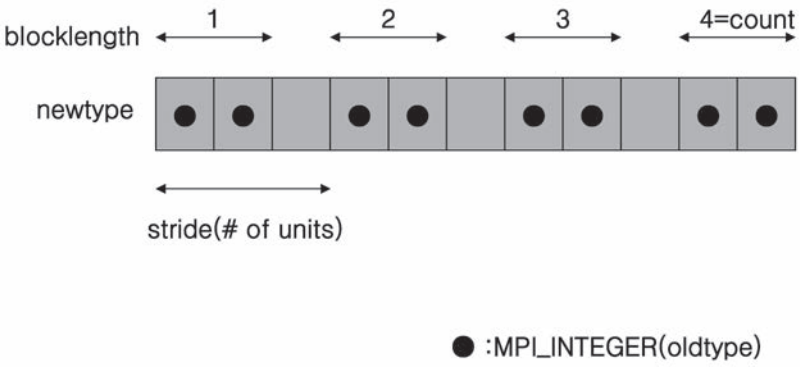
- 똑같은 간격만큼 떨어져 있는 **count**개 블록들로 구성되는 새로운 데이터 타입 정의
- 각 블록에는 **blocklength**개의 이전 타입 데이터 있음

MPI_TYPE_VECTOR



- **count = 2**
- **stride = 5**
- **blocklength = 3**

MPI_TYPE_VECTOR 예제



MPI_TYPE_VECTOR 예제 : C

```
/*type_vector*/
#include <mpi.h>
#include <stdio.h>
void main (int argc, char *argv[]){
  int i, myrank, ibuf[20];
  MPI_Datatype inewtype ;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  if(myrank==0) for(i=0; i<20; i++) ibuf[i]=i+1;
  else for(i=0; i<20; i++) ibuf[i]=0;
  MPI_Type_vector(4, 2, 3, MPI_INT, &inewtype);
  MPI_Type_commit(&inewtype);
  MPI_Bcast(ibuf, 1, inewtype, 0, MPI_COMM_WORLD);
  printf("%d : ibuf =", myrank);
  for(i=0; i<20; i++) printf(" %d", ibuf[i]);
  printf("\n");
  MPI_Finalize();
}
```

MPI_TYPE_VECTOR 예제 : Fortran

```
PROGRAM type_vector
INCLUDE 'mpif.h'
INTEGER ibuf(20), inewtype
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
IF (myrank==0) THEN
DO i=1,20
  ibuf(i) = i
ENDDO
ENDIF
CALL MPI_TYPE_VECTOR(4, 2, 3, MPI_INTEGER, inewtype, ierr)
CALL MPI_TYPE_COMMIT(inewtype, ierr)
CALL MPI_BCAST(ibuf, 1, inewtype, 0, MPI_COMM_WORLD, ierr)
PRINT *, 'ibuf =', ibuf
CALL MPI_FINALIZE(ierr)
END
```


MPI_TYPE_CREATE_STRUCT

C	`int MPI_Type_create_struct (int count, int *array_of_blocklengths, MPI_Aint *array_of_displacements, MPI_Datatype *array_of_type, MPI_Datatype *newtype)`
Fortran	`MPI_TYPE_CREATE_STRUCT (count, array_of_blocklengths, array_of_displacements, array_of_types, newtype, ierr)`

INTEGER count : 블록의 개수, 동시에 배열 array_of_blocklengths, array_of_displacements, array_of_types의 원소의 개수를 나타냄 (IN)

INTEGER array_of_blocklengths(*) : 각 블록 당 데이터의 개수, array_of_blocklengths(i)는 데이터 타입이 array_of_types(i)인 i번째 블록의 데이터 개수 (IN)

INTEGER array_of_displacements(*) : 바이트로 나타낸 각 블록의 위치(IN)

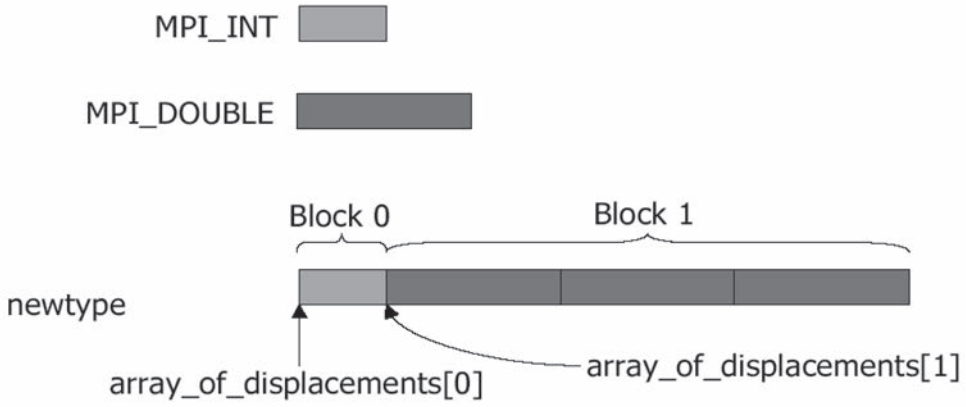
INTEGER array_of_types(*) : 각 블록을 구성하는 데이터 타입, i번째 블록은 데이터 타입이 array_of_types(i)인 데이터로 구성 (IN)

INTEGER newtype : 새로운 데이터 타입 (OUT)

MPI_TYPE_CREATE_STRUCT

- 가장 일반적인 유도 데이터 타입 작성 루틴
- 서로 다른 데이터 타입들로 구성된 변수 정의 가능
  - C 구조체
  - Fortran 커먼 블록
- **count**개 블록으로 구성된 새로운 데이터 타입 정의, **i**번째 블록은 데이터 타입이 **array_of_types(i)**인 **array_of_blocklengths(i)**개의 데이터로 구성되며 그 위치는 **array_of_displacements(i)**가 됨

MPI_TYPE_CREATE_STRUCT



- **count = 2**
- **array_of_blocklengths = { 1, 3}**
- **array_of_types = {MPI_INT, MPI_DOUBLE}**
- **array_of_displacements = {0, extent(MPI_INT)}**

MPI_TYPE_CREATE_STRUCT : C

```
/*type_struct*/
#include <stdio.h>
#include<mpi.h>
void main(int argc, char *argv[]) {
   int rank,i;
   MPI_Status status;
   struct {
      int num; float x; double data[4];
   } a;
   int blocklengths[3]={1,1,4};
   MPI_Datatype types[3]={MPI_INT,MPI_FLOAT,MPI_DOUBLE};
   MPI_Aint disps[3];
   MPI_Datatype restype;

   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD,&rank);
```


MPI_TYPE_CREATE_STRUCT : C

```
  MPI_Get_address(&a.num,&disps[0]);
  MPI_Get_address(&a.x,&disps[1]);
  MPI_Get_address(&a.data,&disps[2]);
  disps[2]-=disps[0];   disps[1]-=disps[0];   disps[0] =
   (MPI_Aint)MPI_BOTTOM;

   MPI_Type_create_struct(3,blocklengths,disps,types,&restype);
   MPI_Type_commit(&restype);

   if (rank==0){
       a.num=6; a.x=3.14;
       for(i=0;i<4;++i) a.data[i]=(double) i;
       MPI_Send(&a,1,restype,1,52,MPI_COMM_WORLD);
   } else if(rank==1) {
       MPI_Recv(&a,1,restype,0,52,MPI_COMM_WORLD,&status);
       printf("P:%d my a is %d %f %lf %lf %lf %lf\n",
   rank,a.num,a.x,a.data[0],a.data[1],a.data[2],a.data[3]);
   }
   MPI_Finalize();
}
```

MPI_TYPE_CREATE_STRUCT : Fortran

```
PROGRAM type_struct
USE mpi_f08
IMPLICIT NONE
INTEGER::rank
TYPE ddt
  INTEGER::num
  real::x
  COMPLEX::data(4)
END TYPE ddt
INTEGER::blocklengths(3)
INTEGER(KIND=MPI_ADDRESS_KIND)::disps(3)
TYPE(MPI_Datatype)::arr_types(3),restype
TYPE(ddt)::a
INTEGER::i,size
blocklengths=(/1,1,4/)
arr_types=(/MPI_INTEGER,MPI_REAL,MPI_COMPLEX/)
CALL MPI_INIT(err)
CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,err)
CALL MPI_Get_address(a%num,disps(1))
CALL MPI_Get_address(a%x,disps(2))
CALL MPI_Get_address(a%data,disps(3))
disps(3)=disps(3)-disps(1);   disps(2)=disps(2)-disps(1);
   disps(1)=MPI_BOTTOM
```


MPI_TYPE_CREATE_STRUCT : Fortran

```
CALL MPI_TYPE_CREATE_STRUCT(3,blocklengths,disps, &
          arr_types,restype,err)
CALL MPI_TYPE_COMMIT(restype,err)
IF(rank.eq.0) THEN
   a%num=6; a%x=3.14
   DO i=1,4
      a%data(i)=cmplx(i,i)
   ENDDO
   CALL MPI_SEND(num,1,restype,1,30,MPI_COMM_WORLD,err)
ELSE IF(rank.eq.1) THEN
   CALL MPI_RECV(num,1,restype,0,30,MPI_COMM_WORLD,status,err)
   PRINT *,'P:',rank,' I got'
   PRINT *, a%num
   PRINT *, a%x
   PRINT *, a%data
END IF
CALL MPI_FINALIZE(err)
END
```

How to parallelize

- **Parallelizing DO Loops**
- Parallelization and Message Passing

Block Distribution

iteration	1	2	3	4	5	6	7	8	9	10	11	12
rank	0	0	0	1	1	1	2	2	2	3	3	3

Block Distribution

- Suppose when you divide n by p, the quotient is q and the remainder is r.
  - n = p X q + r
- Processes 0..r-1 are assigned *q* + 1 iterations each. The other processes are assigned *q* iterations.
  - n = r(q+1) + (p-r)q

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	0	0	0	1	1	1	1	2	2	2	3	3	3

Block Distribution: Fortran

```
SUBROUTINE para_range(n1, n2, nprocs, irank, ista,
  iend)
  iwork1 = (n2 - n1 + 1) / nprocs
  iwork2 = MOD(n2 - n1 + 1, nprocs)
  ista = irank * iwork1 + n1 + MIN(irank, iwork2)
  iend = ista + iwork1 - 1
  IF (iwork2 > irank) iend = iend + 1
END
```

Block Distribution: C

```
void  para_range(int  n1,int  n2,  int  nprocs,  int
  myrank, int *ista, int *iend){
   int iwork1, iwork2;
   iwork1 = (n2-n1+1)/nprocs;
   iwork2 = (n2-n1+1) % nprocs;
   *ista= myrank*iwork1 + n1 + min(myrank, iwork2);
   *iend = *ista + iwork1 - 1;
   if(iwork2 > myrank) *iend = *iend + 1;
}

int min(int x, int y){
 int v;
 if (x>=y) v = y;
 else v = x;
 return v;
}
```


Parallel Sum: C

```
#include <mpi.h>
#include <stdio.h>
#define n 1000
void para_range(int, int, int, int, int*, int*);
int min(int, int);
void main (int argc, char *argv[]){
 int i, nprocs, myrank;
 int ista, iend, diff;
 double *a;  double sum, tsum;
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 para_range(1, n, nprocs, myrank, &ista, &iend);
 diff = iend - ista +1;
 a = (double *)calloc(diff, sizeof(double));
 for(i = ista-1; i<iend; i++)  a[i] = i+1;
 sum = 0.0;
 for(i = ista-1; i<iend; i++) sum = sum + a[i];
 MPI_Reduce(&sum, &tsum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
 if(myrank == 0) printf(" sum = %f \n", tsum);
 MPI_Finalize();
}
```

```
$ mpiicc parasum.c para_range.c -o psum.x
$ mpirun -np 4 ./psum.x
```

Parallel Sum: Fortran

```
PROGRAM main
INCLUDE 'mpif.h'
PARAMETER (n=1000)
REAL, ALLOCATABLE :: a(:)
REAL :: sum, tsum
Call MPI_INIT(ierr)
Call MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
Call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
call para_range(1, n, nprocs, myrank, ista, iend)
ALLOCATE (a(ista:iend))
a = (/(i, i=ista,iend)/)
sum = 0.0
do i = ista, iend
   sum = sum + a(i)
enddo
DEALLOCATE(a)
Call MPI_REDUCE(sum, tsum, 1, MPI_REAL, MPI_SUM, 0, MPI_COMM_WORLD, ierr)
if (myrank==0) then
print *, 'sum =', tsum
endif
Call MPI_FINALIZE(ierr)
End
```

```
$ mpiifort parasum.f90 para_range.f90 -o psum.x
$ mpirun -np 4 ./psum.x
```


Cyclic Distribution

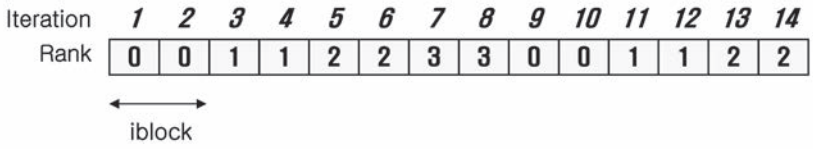
```
DO i = n1, n2                    DO i = n1+myrank, n2, nprocs
   computation          ⇒           computation
ENDDO                            ENDDO
```

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	1	2	3	0	1	2	3	0	1	2	3	0	1

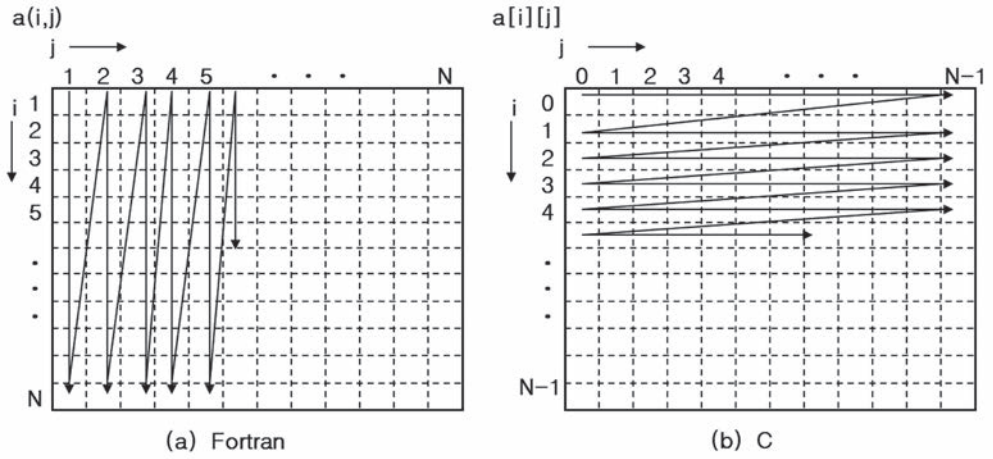
- **More balanced workload for processes than the block distribution**
- **More cache misses than the block distribution**

Block-Cyclic Distribution

```
DO ii = n1+myrank*iblock, n2, nprocs*iblock
   DO i = ii, MIN(ii+iblock-1, n2)
      computation
   ENDDO
ENDDO
```



Nested Loop

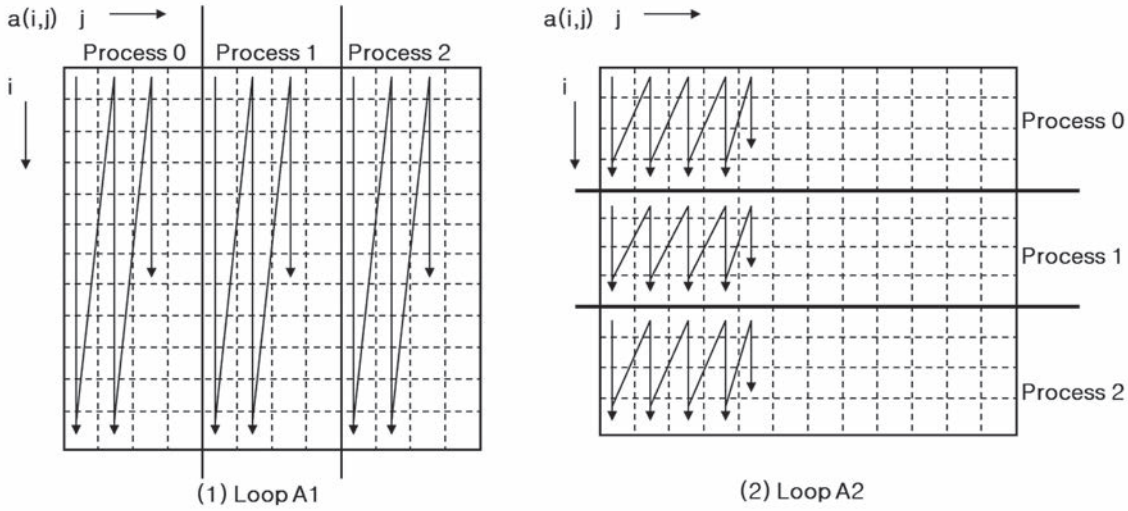


Nested Loop Parallelization

Loop A1	Loop A2

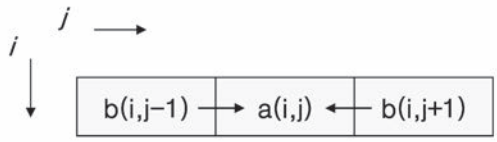
DO j = jsta, jend
  DO i = 1, n
  a(i,j) = b(i,j) + c(i,j)
  ENDDO
ENDDO
	DO j = 1, n
	  DO i = ista, iend
	  a(i,j) = b(i,j) + c(i,j)
	  ENDDO
	ENDDO

Nested Loop Parallelization



통신 고려

➢ 한 방향으로 필요한 데이터를 통신해야 하는 경우

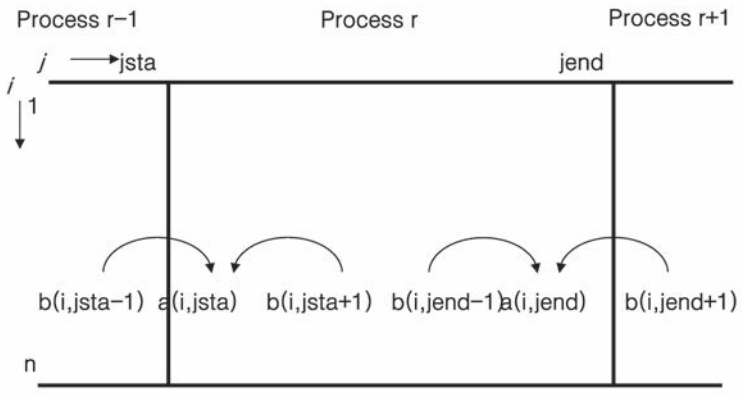


Loop C

```
DO j = 1, n
    DO i = 1, n
        a(i,j) = b(i, j-1) + b(i, j+1)
    ENDDO
ENDDO
```


통신량 고려

➢ 바깥쪽 루프(열 방향) 병렬화에 의해 요구되는 통신



➢ 이 경우 안쪽 루프(행 방향)의 병렬화는 통신 필요 없음

How to parallelize

- Parallelizing DO Loops
- **Parallelization and Message Passing**

1차원 유한 차분법

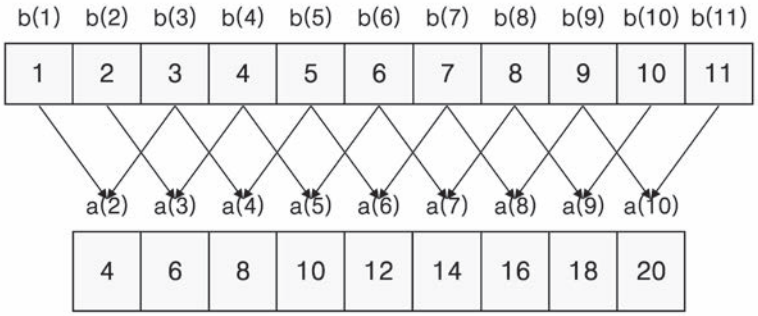
➢ **1**차원 유한 차분법**(FDM)**의 핵심부 **:** 순차 프로그램

Fortran	C

PROGRAM 1D_fdm_serial
IMPLICIT REAL*8 (a-h, o-z)
PARAMETER (n=11)
DIMENSION a(n), b(n)
DO i = 1, n
  b(i) = i
ENDDO
DO i = 2, n-1
  a(i) = b(i-1) + b(i+1)
ENDDO
END
	/*1D_fdm_serial*/
	#define n 11
	main(){
	  double a[n], b[n];
	  int i;
	  for(i=0; i<n; i++)
	  b[i] = i+1;
	  for(i=1; i<n-1; i++)
	  a[i] = b[i-1] + b[i+1];
	}

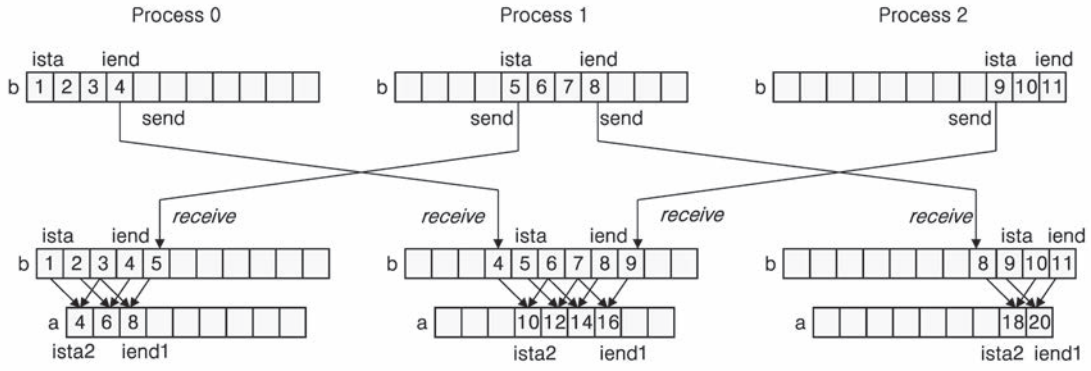
1차원 유한 차분법

➢ **1**차원 **FDM**의 데이터 의존성



1차원 유한 차분법

➢ 병렬화된 1차원 FDM의 데이터 전송



※ 2차원 FDM은 2차원 배열의 경계선, 3차원 FDM은 3차원 배열의 경계면 전송

1D FDM: C

```
/*parallel_1D_fdm*/
#include <mpi.h>
#define n 11
void para_range(int, int, int, int, int*, int*);
int min(int, int);
main(int argc, char *argv[]){
  int i, nprocs, myrank ;
  double a[n], b[n];
  int ista, iend, ista2, iend1, inext, iprev;
  MPI_Request isend1, isend2, irecv1, irecv2;
  MPI_Status istatus;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  para_range(0, n-1, nprocs, myrank, &ista, &iend);
  ista2 = ista; iend1 = iend;
  if(myrank==0) ista2=1;
  if(myrank==nprocs-1)iend1=n-2;
```


1D FDM: C

```
  inext=myrank+1; iprev=myrank-1;
  if(myrank==nprocs-1) inext=MPI_PROC_NULL;
  if(myrank==0) iprev=MPI_PROC_NULL;
  for(i=ista; i<=iend; i++) b[i]=i+1;
  MPI_Isend(&b[iend], 1, MPI_DOUBLE, inext ,1, MPI_COMM_WORLD,
   &isend1);
  MPI_Isend(&b[ista], 1, MPI_DOUBLE, iprev, 1, MPI_COMM_WORLD,
   &isend2);
  MPI_Irecv(&b[ista-1], 1, MPI_DOUBLE, iprev, 1, MPI_COMM_WORLD,
   &irecv1);
  MPI_Irecv(&b[iend+1], 1, MPI_DOUBLE, inext, 1, MPI_COMM_WORLD,
   &irecv2);
  MPI_Wait(&isend1, &istatus);
  MPI_Wait(&isend2, &istatus);
  MPI_Wait(&irecv1, &istatus);
  MPI_Wait(&irecv2, &istatus);
  for(i=ista2; i<=iend1; i++) a[i] = b[i-1] + b[i+1];
  MPI_Finalize();
}
```

1D FDM: Fortran

```
PROGRAM parallel_1D_fdm
INCLUDE 'mpif.h'
PARAMETER (n=11)
DIMENSION a(n), b(n)
INTEGER istatus(MPI_STATUS_SIZE)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
CALL para_range(1, n, nprocs, myrank, ista, iend)
ista2 = ista; iend1 = iend
IF (myrank == 0) ista2 = 2
IF (myrank == nprocs-1) iend1 = n-1
inext = myrank + 1; iprev = myrank - 1
IF (myrank == nprocs-1) inext = MPI_PROC_NULL
IF (myrank == 0)        iprev = MPI_PROC_NULL

DO i = ista, iend
   b(i) = i
ENDDO
```


1D FDM: Fortran

```
CALL MPI_ISEND(b(iend), 1, MPI_REAL, inext, 1, &
     MPI_COMM_WORLD,  isend1, ierr)
CALL MPI_ISEND(b(ista), 1, MPI_REAL, iprev, 1, &
     MPI_COMM_WORLD, isend2, ierr)
CALL MPI_IRECV(b(ista-1), 1, MPI_REAL, iprev, 1, &
     MPI_COMM_WORLD, irecv1, ierr)
CALL MPI_IRECV(b(iend+1), 1, MPI_REAL, inext, 1, &
     MPI_COMM_WORLD, irecv2, ierr)
CALL MPI_WAIT(isend1, istatus, ierr)
CALL MPI_WAIT(isend2, istatus, ierr)
CALL MPI_WAIT(irecv1, istatus, ierr)
CALL MPI_WAIT(irecv2, istatus, ierr)

DO i = ista2, iend1
   a(i) = b(i-1) + b(i+1)
ENDDO
CALL MPI_FINALIZE(ierr)
END
```

Q & A

KISTI과학데이터스쿨은 >Identity

한국과학기술정보연구원 KISTI 정관, 국가초고성능컴퓨터 활용 및 육성에 관한 법률에 따라 **KISTI**가 보유한 데이터 및 슈퍼컴퓨팅 인프라와 전문인력을 활용하여 산·학·연·관 종사자 대상으로 데이터SW & 분석도구 포함와 **HPC** High Performance Computing 이용에 관한 **전문교육**과 과학기술 대중화를 위한 **국민참여 프로그램**, 신산업 분야 일자리 창출을 위한 **4차 산업 인재양성** 과학기술 빅데이터 분석, **내부 인력 역량강화** 교육을 개발하여 운영하고 있습니다.

KISTI과학데이터스쿨 목표 >Mission

데이터·컴퓨팅 관련 활동의 **통합·연계 전문교육을 통한 산업과 과학기술 분야에서 경쟁력을 가진 전문인력 양성**과 4차 산업혁명 시대를 맞아 **빅데이터, 인공지능 등 신산업 창출에 기여 가능한 미래 인재**를 양성합니다.

KISTI과학데이터스쿨 연혁 >History

HPC 활용교육
데이터이용교육

1966
제1회 과학기술 정보관리강좌 개설

1984
초기 교육반 **정보보급실** 운영

1988
슈퍼컴퓨터 1호기 **Cray2S**도입 및 국내 최초 **HPC 활용** 교육 시작

1991
「KISTI 전신(산업기술정보원) 정관 4조(사업) 7항」 의거 정보이용교육 운영

'기관공식 교육부서 **정보교육센터**' 신설

1993
슈퍼컴퓨터 2호기 **Cray YMP**도입 및 **2호기 활용 교육**

2001
슈퍼컴퓨터 3호기**노벨** 도입 및 **3호기** 활용 교육

「KISTI 정관 4조 (사업) 11항」 의거 전문교육 운영

2008
슈퍼컴퓨터 4호기 **타키온** 도입 및 4호기 활용 교육

2011
「국가초고성능컴퓨터 활용 및 육성에 관한 법률」 제정 및 슈퍼컴퓨팅 전문인력 양성 교육 운영

2018
교육과정 개편 및 확대 / 데이터&HPC 교육 통합

- 데이터 이용 및 HPC 활용 교육 통합 운영
- 슈퍼컴퓨터 5호기(누리온) 도입 및 활용 교육
- 교육 통합 홈페이지 구축 및 온라인 콘텐츠 구축
- 국민참여 프로그램(데이터·슈퍼컴퓨팅·인공지능 캠프) 개설
- 4차산업 인재양성(과기정통부 빅데이터분석가 과정)
- 제주대, UNIST, ADD 등 외부 협력 교육 운영

KISTI 과학데이터스쿨
KISTI SCIENCE DATA SCHOOL

교육분야 >The field of education

KISTI과학데이터스쿨은 과학기술분야에서의 연구데이터·컴퓨팅 활용을 위한 전문교육 프로그램을 제공합니다. 또한 연구데이터, 컴퓨팅 활용 온라인 교육환경 및 콘텐츠를 구축하고 과학기술 대국민 이해 확산을 위한 국민참여 프로그램을 운영하고 있습니다.

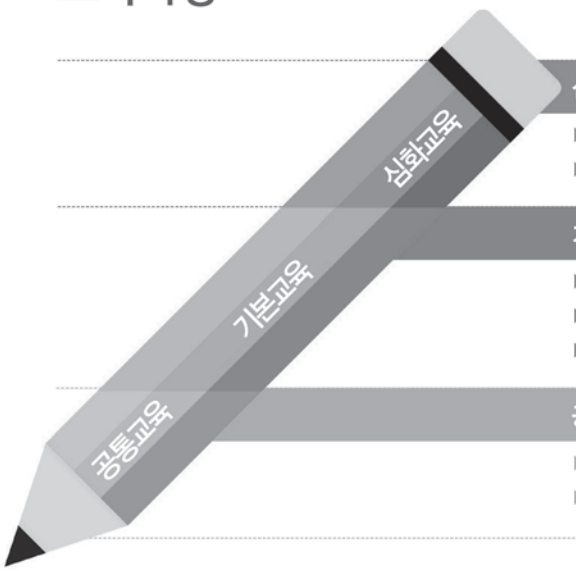
전문교육 프로그램
데이터 이용 전문가 교육
HPC 활용 전문가 교육
AI 응용 전문가 교육

4차산업 인재양성
과학기술 빅데이터 분석가 과정
외부 연계 교육
(제주대, UNIST, ADD 등)

국민참여 프로그램
슈퍼컴퓨팅 청소년캠프
KISTI DATA·AI Camp
KISTI Hackathon

※ KISTI과학데이터스쿨은 「KISTI 정관 제 1장 4조(사업) 11항」와 「국가초고성능컴퓨터 활용 및 육성에 관한 법률」에 따라 전문 교육 사업을 추진하고 있습니다.

교육과정 >Training Course



심화교육 : 과학기술 분야별 데이터 / HPC / AI 활용
- ▶ 시스템 운영, 고급 프로그래밍, 미래기술 예측 등 과학기술 분야별 데이터, HPC, AI 활용 교육
- ▶ 바이오, 천문, 기상 등 과학기술 분야별 데이터와 슈퍼컴퓨터를 활용하여 그룹 프로젝트 진행

기본교육 : 데이터 / HPC / AI 각 영역에 대한 이론과 실습
- ▶ 데이터 : Data Management Plan, 오픈 액세스와 학술연구, 데이터 분석 등
- ▶ HPC : 슈퍼컴퓨터 5호기, 병렬 프로그래밍, 과학데이터 가시화 등
- ▶ AI : 텐서플로우(TensorFlow), 기계학습(Machine Learning) 등

공통교육 : 데이터 / HPC / AI 활용을 위해 공통으로 요구되는 지식과 기술
- ▶ 프로그래밍 언어 : 파이썬(Python), R, 포트란(Fortran) 등
- ▶ 4차산업혁명, 과학기술 관련 이해, 데이터과학 이론 등

교육 신청방법 ▶ **https://kacademy.kisti.re.kr** 회원 가입 후 교육 신청

교육 문의
▶ **e-Mail** : kacademy@kisti.re.kr ▶ https://kacademy.kisti.re.kr
▶ **데이터 이용 교육 :** 김재성 02-3299-6119 ▶ **HPC 활용 교육/국민참여 프로그램 :** 한성근 042-869-0650
▶ **4차 인재양성사업 – 과학기술 빅데이터 분석가 과정 :** 김재성 02-3299-6119

Training Course

교육과정 소개

데이터 이용 전문가 교육

내용 : 데이터 이용 전주기(데이터 수집, 저장, 통합, 분석, 활용)의 각 단계에 대한 이해와 수행능력 향상
분야 : 데이터 검색 및 분석, 연구개발 (R&D), 기술이전, Data Management Plan 등
대상 : 산업체·학교·연구소·정부기관 연구원, 실무담당자

HPC 활용 전문가 교육

내용 : 슈퍼컴퓨터 운영과 코드 성능향상 기법 및 프로그래밍 기술 등의 슈퍼컴퓨터 활용을 학습하여 복잡하고 데이터 집약적인 문제, 거대 계산문제 등을 해결할 수 있는 능력 함양
분야 : 시스템운영, 프로그래밍(언어 포함), 코드 성능향상(최적화/병렬화/벡터화), 계산과학(데이터 전처리 – 계산(시뮬레이션) – 후처리(가시화)) 등
대상 : 산업체·학교·연구소·정부기관 연구원

AI 응용 전문가 교육

내용 : 머신러닝과 딥러닝의 이론적인 이해와 최첨단 응용 프로그램 개발을 위한 AI 기술 적용 및 AI 시스템 응용 능력 함양
분야 : 딥러닝, 통계, 파이썬, R, 하둡, 엑셀, 데이터공학, 분야별 빅데이터 분석 등
대상 : 산업체·학교·연구소·정부기관 연구원

4차산업 인재양성 - 과학기술 빅데이터 분석가 과정

내용 : (비)이공계 미취업 청년을 대상으로 4차 산업혁명 시대에 필요한 과학기술 분야별 빅데이터 관련 교육 운영 및 취업 연계프로그램 (6개월 과정)
분야 : 생명, 기상기후, 천문우주, 생태환경 등
대상 : (비)이공계 미취업 청년

국민참여 프로그램(I)- 슈퍼컴퓨팅 청소년캠프, 데이타 AI 캠프

내용 : 슈퍼컴퓨터의 원리 이해, 딥러닝의 전반적인 절차 이해 및 음성인식 등 인공지능 활용 프로그램 운영 실습/이론 교육 운영
대상 : 청소년, 대학(원)생, 과학교사, 일반인 등

국민참여 프로그램(II)- KISTI Hackathon

내용 : 본인이 보유하고 있는 모듈이나 알고리즘 코드를 GPU 및 openACC 전문가의 도움으로 문제해결(최적화, 병렬화)하는 과정을 통해 거대과학 도전 기회 제공
대상 : 산업체·학교·연구소·정부기관 연구원 (코드 보유자)

※ 상세한 내용 확인 및 교육 신청은 KISTI과학데이터스쿨 홈페이지 **https://kacademy.kisti.re.kr** 에서 확인바랍니다.

KiSTi 한국과학기술정보연구원 Korea Institute of Science and Technology Information

2020년 KISTI 과학데이터스쿨 교육 일정

KISTI과학데이터스쿨 KISTI SCIENCE DATA SCHOOL

■ 서울(분원) 교육

교 육 과 목 명	기간		수강료	교육일정											
	일수	시간	만원	1월	2월	3월	4월	5월	6월	7월	8월	9월	10월	11월	12월
DATA 이용 교육															
KnowledgeMatrix Plus 활용 네트워크 분석 및 시각화	2	12	30						29-30			21-22			
Python 활용 공공데이터 분석 및 시각화	3	18	45						1-3			7-9			
R 활용 데이터분석	3	18	45		12-14			13-15			19-21			18-20	
R 활용 서지데이터 네트워크 분석 및 시각화	2	12	30					26-27						2-3	
R&D 기획 및 사업화를 위한 시장 조사	1	6	15				21						13		
Tableau 활용 공공데이터 분석 및 시각화	1	6	15				29					23			
기술가치 평가	3	18	45				1-3						28-30		
기술로드맵 작성	3	18	45			11-13				13-15			14-16		
기술사업화를 위한 데이터기반 사업타당성 분석 및 평가	2	12	30			5-6				23-24			22-23		
빅데이터 이용 신규사업 아이디어 개발 및 프로세스 이해	1	6	15			3				21			20		
빅데이터 활용 미래산업시장과 기업 분석	1	6	15			4				22			21		
연구개발 기획 ·관리 ·평가	3	18	45		26-28				10-12					4-6	
연구자를 위한 데이터과학	1	6	15				17						27		
특허기술 동향분석	3	18	45				8-10					16-18			
특허데이터 활용 실무	2	12	30			19-20			23-24			3-4			
HPC 활용 교육															
CUDA	2	12	무료						4-5						
Fortran	2	12	무료							16-17					
MPI 초급	1	6	무료					28-29			5-6				
MPI 고급	2	12	무료								7				
OpenACC	2	12	무료						18-19						
OpenMP 초급	1	6	무료							29-30					
OpenMP 고급	2	12	무료							31					
리눅스(Linux)	1	6	무료							3					
성능 최적화	2	12	무료						25-26			24-25			
슈퍼컴퓨터 5호기(누리온) 활용	1	6	무료							2					
AI 응용 교육															
딥러닝(Deep Learning) 데이터 분석	3	18	45		5-7		22-24			8-10				11-13	
인공지능 기술 활용	2	12	30			26-27			16-17			10-11		9-10	
파이썬(Python) 데이터 분석	3	18	45		19-21			20-22			26-28			25-27	

>> 상기의 일정은 기관사정에 따라 변동 가능합니다.
>> 교육시간: 9:30~16:30 (6시간/1일)
>> 교육장소: 02456 서울특별시 동대문구 회기로 66, 한국과학기술정보연구원(KISTI) 1층, 제1교육장
>> 자세한 내용은 홈페이지(https://kacademy.kisti.re.kr)에서 확인 바랍니다.
>> 교육문의: **T** 02. 3299. 6119 **e-Mail** kacademy@kisti.re.kr

■ 대전(본원) 교육

교 육 과 목 명	기간		수강료	교육일정											
	일수	시간	만원	1월	2월	3월	4월	5월	6월	7월	8월	9월	10월	11월	12월
HPC 활용 교육															
CUDA	2	12	무료								20-21	10-11		5-6	
Fortran	2	12	무료		20-21				1-2			17-18			
MPI(고급)	1	6	무료				27-28						29-30		
MPI(초급)	2	12	무료				29								
OpenACC	2	12	무료				2-3					3-4		19-20	
OpenMP(고급)	1	6	무료					21-22			12-13		22-23		
OpenMP(초급)	2	12	무료								14				
리눅스(Linux)	1	6	무료		7			8					8		
성능 최적화	2	12	무료				23-24							26-27	
슈퍼컴퓨터 5호기(누리온) 활용	1	6	무료		6			7					7		

>> 상기의 일정은 기관사정에 따라 변동 가능합니다.
>> 교육시간: 9:30~16:30 (6시간/1일)
>> 교육장소: 34141 대전광역시 유성구 대학로 245, 한국과학기술정보연구원 키움관(기숙사동) 1층, 과학데이터스쿨 강의실2
>> 자세한 내용은 홈페이지(https://kacademy.kisti.re.kr)에서 확인 바랍니다.
>> 교육문의: **T** 042. 869. 0650 **e-Mail** kacademy@kisti.re.kr

■ 국민참여 프로그램

프로그램 명	기간		수강료	교육일정											
	일수	시간	만원	1월	2월	3월	4월	5월	6월	7월	8월	9월	10월	11월	12월
KISTI-INTEL 데이터 캠프	3	-	무료						30	1-2					
KISTI-KT 인공지능 캠프	1	-	무료				21	15							
UST-KISTI-INTEL 인공지능 스쿨	3	-	무료									23-25			
제1회 EDISON 청소년 캠프	5	-	무료							13-17					
제2회 KISTI-NVIDIA OpenACC Hackathon	5	-	무료							6-10					
제6회 슈퍼컴퓨팅 청소년 캠프	5	-	무료							13-17					

>> 상기의 일정은 기관사정에 따라 변동 가능합니다.
>> 자세한 내용은 홈페이지(https://kacademy.kisti.re.kr)에서 확인 바랍니다.
>> 교육문의: **T** 042. 869. 0650 **e-Mail** kacademy@kisti.re.kr