IBS 슈퍼컴퓨터 활용

2024. 11

Agenda

- Motivation
  - Supercomputer?
  - 성능 결정 요인
  - TOP 500 LIST
  - 병렬 컴퓨터
- Basic Environment
  - Linux 기초
  - Environment Module
  - 프로그램 컴파일
  - 컴파일 및 Job 스크립트 사용 예제

Motivation

1. **Super Computer ?**
2. 성능 결정 요인
3. **TOP 500**
4. 병렬컴퓨터

Super Computer

- Supercomputer의 정의
  - 당대의 기술로 가능한 최고 성능의 컴퓨터
    - 당대의 범용 컴퓨터에 비해 수백~수천 혹은 그 이상의 성능
    - 현재 기술의 발달로 Supercomputer와 범용 컴퓨터의 구분이 모호
    - 주로 과학기술 연산에 사용됨
    - 컴퓨터의 계산 성능으로 순위를 따져서 구분
      - TOP 500 List
    - 단순히 HPC(High-Performance Computer)라고도 함
- Super Computer의 필요성
  - 빠른 컴퓨터 필요
  - 큰 용량 필요
  - 정확한 해 필요

Super Computer

- 현재의 모든 Super Computer는 다수의 프로세서를 사용
- 왜 병렬인가 ?
  - 단일 컴퓨터의 한계
  - 큰 규모의 문제
  - 동시성
  - 시간과 비용 절약
  - 타 지역 자원 활용

성능 결정 요인

- Super Computer 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
      - 슈퍼 스칼라 등

TOP 500

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

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

이미지 출처 : https://www.tomshardware.com/news/us-supercomputer-china-top500-summit,37367.html

TOP 500

- The Green 500(http://www.green500.org/)

- The Graph 500 (http://www.graph500.org/)
- The IO 500 (http://www.io500.org/)

병렬 컴퓨터

- 현재의 슈퍼컴퓨터는 다수의 프로세서를 사용하는 병렬 컴퓨터
- Super Computer를 분류하는 데는 여러 다른 방법이 존재
  - Supper Computer를 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, MPI + OpenMP 병렬 프로그래밍 기법 사용

병렬 컴퓨터에의 대표적인 S/W

- MPI
  - Message Passing Interface
  - 프로세서 사이의 통신과 데이터 교환을 위한 라이브러리
  - 분산 메모리 시스템에서 사용됨
  - OpenMPI, IntelMPI, MPICH 등

- OpenMP
  - 대표적인 공유 메모리 구조를 위한 프로그래밍 규약
  - 컴파일러에 의해 지원

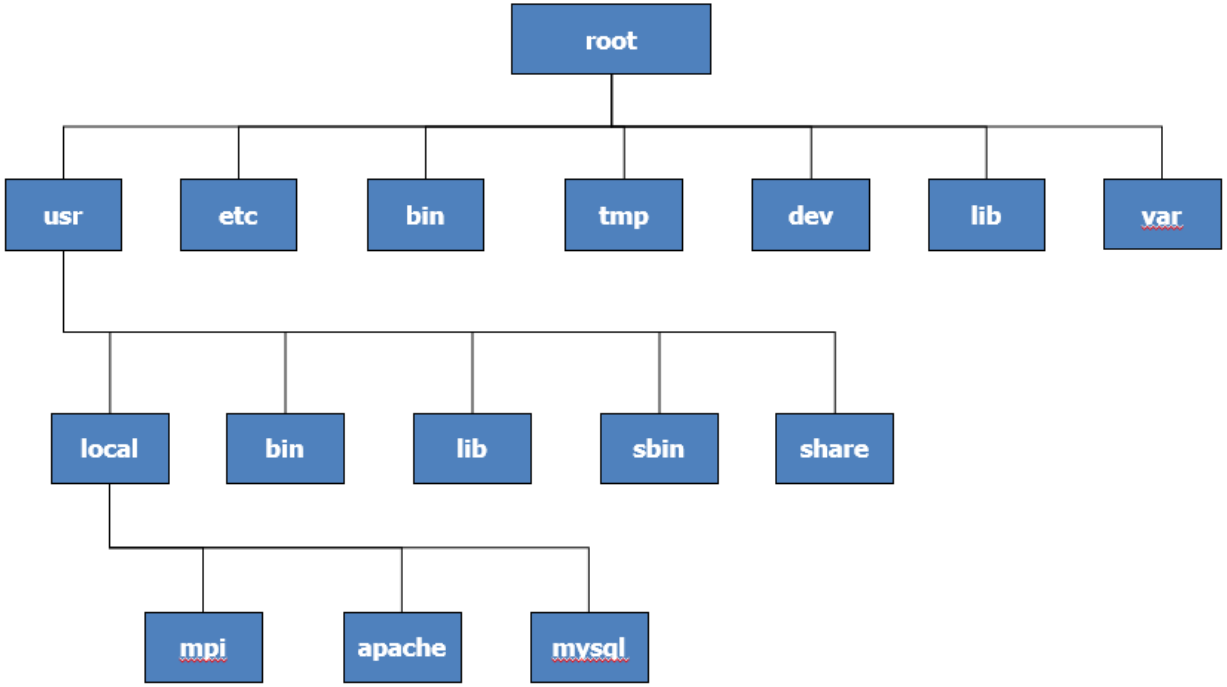
병렬 컴퓨터에의 대표적인 S/W

- 선형대수 라이브러리
  - BLAS
    - 기본적인 벡터와 행렬 연산을 수행
    - LAPACK, ScaLAPACK과 여러 라이브러리에 사용됨
  - LAPACK
    - 선형 대수학의 수치적 계산을 수행하는 Library
      - 일차 방정식, 행렬 분해, 행렬 고유값 등을 계산
  - SCALAPACK(SCAlable LAPACK)
    - 분산 메모리 시스템에서 사용 가능한 선형대수 라이브러리
  - Intel® Math Kernel Library(Intel® MKL)
    - Linear Algebra, Fast Fourier Transforms(FFT), Vector Math and Statistics 함수 등을 포함

Basic Environment

1. **Linux** 기초
   - 기본 명령어
   - VI Editor
2. **Environment Module**
   - Module 명령어
   - 권장 컴파일러 옵션
3. 프로그램 컴파일

- 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 man page를 가짐
    - 다음 페이지 보기는 space bar 또는 'f' 입력
    - 이전 페이지 보기는 '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

- which
  - 명령어의 경로를 보여줌

```
$ which ls
alias ls='ls --color=auto'
        /usr/bin/ls
```

- whatis
  - 명령어가 무슨 일을 하는 지 설명

```
$ whatis ls
ls (1)               - list directory contents
ls (1p)              - list directory contents
```

- top
  - 시스템 모니터링

```
$ top -d 1
…
   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
186662 root      20   0 14.694g 3.861g 364948 S   3.9  1.0 729:01.39 cmd
451401 root      20   0  128012  10688   1872 S   2.9  0.0 275:04.65 bash
331920 root      20   0   26332   2472   1848 S   1.9  0.0   0:00.02 sample_ipm
```

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 내용 비교 시 사용, 바이너리 파일인 경우 동일 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)

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(명령행 모드로 변경 시) : “ESC” key
  - 명령행 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)
      - 셀 환경에서 모듈파일들을 제거함
    - li(list)
      - 적재된 모듈파일들을 나열함
    - purge
      - 적재된 모든 모듈파일들을 나열함

Environment Module

- module 명령
  - 사용가능 모듈 확인 (av)

```
[lenovo@olaf1 ~]$ module av
------------------------------------------------------------ /opt/ibs_lib/modulefiles/compilers ------------------------------------------------------------
gcc/8.5.0  gcc/9.3.0  gcc/11.2.0  gcc/12.2.0  intel/2021.2.0  intel/2021.3.0(default)  intel/2022.0.2  intel/2022.2.1  pgi/23.5

--------------------------------------------------------------- /opt/ibs_lib/modulefiles/mpi ---------------------------------------------------------------
impi/2021.1.1  impi/2021.2.0  impi/2021.3.0  impi/2021.5.1  impi/2021.7.1(default)  openmpi/4.1.1  openmpi/4.1.1-Rocky  openmpi/4.1.4

------------------------------------------------------------ /opt/ibs_lib/modulefiles/libraries ------------------------------------------------------------
anaconda/23.09.0  cudatoolkit/11.7       fftw/3.3.10          hdf4/4.2.14     miniconda/23.01.0     petsc/3.18.2    readline/8.2
bagel/1.2.2       cudatoolkit/11.8       flex/2.6.4           hdf4/4.2.15     mpc/1.3.1             pnetcdf/1.12.3  root/6.26.10
bison/3.8.2       cudatoolkit/12.2       gaussian/g16.c01     hdf5/1.12.1     mpfr/4.1.1            proj/8.2.1      singularity/4.1.1
blas/3.11.0       cudnn/8.9.7.29-cuda12  gdal/3.5.0           isl/0.24        ncurses/6.4           python/3.6.10   sqlite/3.43.0
boost/1.65.1      curl/7.61.1(default)   ghostscript/10.02.1  jasper/3.0.6    ncview/2.1.10         python/3.7.2    trilinos/13.4.1
boost/1.81.0      curl/7.88.1            git-lfs/3.5.1        jpeg/9e         netcdf-fortran/4.5.4  python/3.8.16   ucx/1.13.1
bzip2/1.0.8       difx/2.6.3             gmp/6.2.1            lapack/3.11.0   netcdf/4.8.1          python/3.9.16   wgrib/1.8.2
charm/7.0.0       difx/2.8.1             go/1.22.0            libtirpc/1.3.3  openfoam/v2006        qchem/6.0.1     wgrib2/3.1.1
cmake/3.18.4      ECW/5.4.0              gromacs/2024.3       libxml2/2.11.4  openssl/1.1.1g        qmcpack/3.16.0  xz/5.4.2
cmake/3.28.1      fftw/3.3.8             gsl/2.7.1            lustre/2.15.1   pcre2/10.42           R/4.0.5         zlib/1.2.11(default)

------------------------------------------------------------ /opt/ibs_lib/modulefiles/cryoem ------------------------------------------------------------
relion-gpu/4.0.0  relion/4.0.0

---------------------------------------------------------- /opt/ibs_lib/modulefiles/protdesign ----------------------------------------------------------
DL-Binder-Design/1.0.0  protein-miniconda/23.1.0

------------------------------------------------------------ /usr/share/Modules/modulefiles ------------------------------------------------------------
dot  module-git  module-info  modules  null  use.own
[lenovo@olaf1 ~]$
```

Environment Module

- 모듈 명령
  - 모듈 정보 출력

```
[lenovo@olaf1 ~]$ module help intel/2021.2.0
----------------------------------------------------------------------
Module Specific Help for /opt/ibs_lib/modulefiles/compilers/intel/2021.2.0:

        This module is for use of Intel Compiler 2018.
        It needs module(s):
                        None
        Use example:
                $ module load intel/2021.2.0

        Additional info:
            1. We can use Intel Advisor, Vtune, Inspector, TBB and MKL.
            2. Major environment variables is set up like these:
                  CC=icc    CXX=icpc    FC=ifort    F77=ifort    F90=ifort
----------------------------------------------------------------------
[lenovo@olaf1 ~]$
```

  - 모듈 적재

```
[lenovo@olaf1 ~]$ module list
No Modulefiles Currently Loaded.
[lenovo@olaf1 ~]$ module load intel/2021.2.0
[lenovo@olaf1 ~]$ module load impi/2021.2.0
```

```
[lenovo@olaf1 ~]$ module list
Currently Loaded Modulefiles:
 1) intel/2021.2.0   2) impi/2021.2.0
[lenovo@olaf1 ~]$
```

Environment Module

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

```
[lenovo@olaf1 ~]$ module list
Currently Loaded Modulefiles:
 1) intel/2021.2.0   2) impi/2021.2.0
[lenovo@olaf1 ~]$
```

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

```
[lenovo@olaf1 ~]$ module rm impi/2021.2.0
[lenovo@olaf1 ~]$ module list
Currently Loaded Modulefiles:
 1) intel/2021.2.0
```

```
[lenovo@olaf1 ~]$ module add openmpi/4.1.1
[lenovo@olaf1 ~]$ module list
Currently Loaded Modulefiles:
 1) intel/2021.2.0   2) openmpi/4.1.1
```

  - 적재된 모든 모듈 삭제

```
[lenovo@olaf1 ~]$ module add openmpi/4.1.1
[lenovo@olaf1 ~]$ module list
Currently Loaded Modulefiles:
 1) intel/2021.2.0   2) openmpi/4.1.1
```

프로그램 컴파일

➢ 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
F77/F90	Intel	ifort	.f, .for, .ftn, .f90, .fpp, .F, .FOR, .FPP, .F90	intel/17.0.5 | intel/18.0.1 | intel/18.0.3
	GNU	gfortran		gcc/6.1.0 | gcc/7.2.0

– 컴파일러별 권장 옵션

```
Intel Compiler -O3 -fPIC -xICELAKE-SERVER
```

```
GNU Complier-O3 -fPIC -march=icelake-server
```

프로그램 컴파일

➢ 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
-xICELAKE-SERVER	512bit 레지스터를 가진 CPU 지원
-qopenmp	OpenMP 기반의 multi-thread 코드 사용
-fPIC, -fpic	PIC(Position Independent Code)가 생성되도록 컴파일

- 권장 옵션

  -O3 -fPIC -XICELAKE-SERVER (Skylake)

```
$ icc -o test.exe -O3 -fPIC -xICELAKE-SERVER test.c
$ ifort -o test.exe -O3 -fPIC -xICELAKE-SERVER test.f90
```

프로그램 컴파일

➢ 프로그램 컴파일

– 순차 프로그램 컴파일

- GNU 컴파일러 주요 옵션

컴파일러 옵션	설명
-O[1|2|3]	오브젝트 최적화, 숫자는 최적화 레벨
-march=skylake-avx512	512bits 레지스터를 가진 CPU 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=icelake-server

```
$ gcc –o test.exe –O3 –fPIC –march=icelake-server test.c
$ gfotran –o tetst.exe –O3 –fPIC –march=icelake-server test.f90
```

프로그램 컴파일

- 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

```
$ icc –o test.exe –qopenmp –O3 –fPIC –xICELAKE-SERVER test.c
$ gcc –o test.exe –fopenmp –O3 –fPIC –march=icelake-server test.c
```

프로그램 컴파일

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

구분	Intel	GNU
Fortran	ifort	gfortran
Fortran + MPI	mpiifort	mpif90
C	icc	gcc
C + MPI	mpiicc	Mpicc
C++	icpc	g++
C++ + MPI	mpiicpc	mpicxx

```
$ mpiicc -o test.exe -qopenmp -O3 -fPIC -xICELAKE-SERVER test.c
$ mpicc -o test.exe -fopenmp -O3 -fPIC -march=icelake-server test.c
```

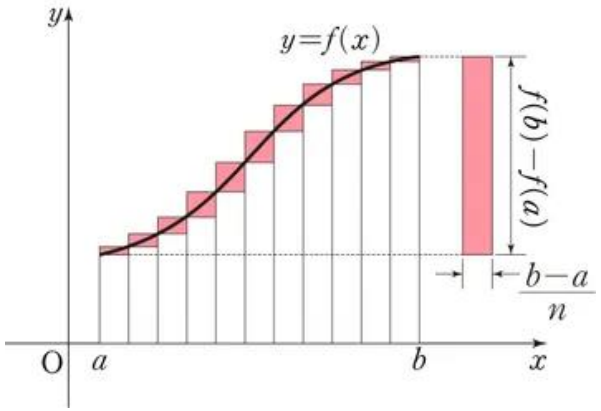
Job Script 사용 예제: (PI 코드)

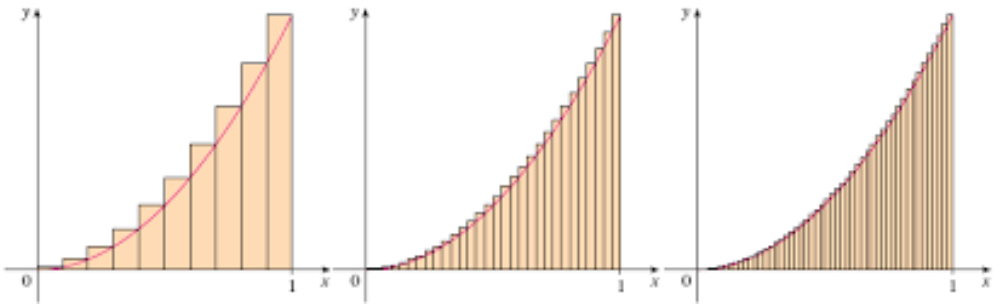
- 정적분을 이용한 pi 계산

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

- 구분 구적법

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





Compile & Job Script 사용 예제: (PI 코드)

- 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);
    }
```

Compile & Job Script 사용 예제: (PI 코드)

- 순차 코드(pi.c)

```
    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;
}
```

- 컴파일

```
$ module purge
$ module load intel
$ icc pi.c -o pi_serial.x -xICELAKE-SERVER
```

Compile & Job Script 사용 예제: (PI 코드)

- PiSrial.sh

```
#!/bin/bash
#SBATCH –j Serial_job
#SBATCH –p normal_cpu
#SBATCH –N 1
#SBATCH –n 1
#SBATCH –C 1
#SBATCH –o %x.o%j
#SBATCH –e %x.e%j
#SBATCH –t 00:10:00

Export OMP_NUM_THREADS=1

module purge
Module load intel

./pi_serial.x
```

```
$ sbatch PiSerial.sh
```

Compile & Job Script 사용 예제: (PI 코드)

- OpenMP(piOpenMP.c)

```
#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");
```

Compile & Job Script 사용 예제: (PI 코드)

- OpenMP(piOpenMP.c)

```
#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;
}
```

Compile & Job Script 사용 예제: (PI 코드)

- 컴파일

```
$ module purge
$ module load intel
$ icc -qopenmp piOpenMP.c -o piOpenMP.x -xICELAKE-SERVER
```

- PiOMP.sh

```
#!/bin/bash
#SBATCH -j OMP_job
#SBATCH -p normal_cpu
#SBATCH --nodes=1
#SBATCH --ntask-per-node=10
#SBATCH -o %x.o%j
#SBATCH -e %x.e%j
#SBATCH -t 00:10:00

Export OMP_NUM_THREADS=10

module purge
Module load intel

./piOpenMP.x
```

```
$ sbatch PiOMP.sh
```

Compile & Job Script 사용 예제: (PI 코드)

- MPI(piMPI.c)

```
#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);
         st = MPI_Wtime();
         }
    h=1.0/(double)num_step;
    sum = 0.0;
    for(i=myrank;i<num_step;i+=nprocs){
        x = h*((double)i-0.5);
        sum += 4.0/(1.0+x*x);
    }
```

Compile & Job Script 사용 예제: (PI 코드)

- MPI(piMPI.c)

```
    mypi= h*sum;
    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    if(myrank==0){
        et=MPI_Wtime();
        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;
}
```

- 컴파일

```
$ module pure
$ module add intel impi
$ mpiicc piMPI.c -o piMPI.x -xICELAKE-SERVER
```

Compile & Job Script 사용 예제: (PI 코드)

- PiMPI.sh

```
#!/bin/bash
#SBATCH –j MPI_jop
#SBATCH -p normal_cpu
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=5
#SBATCH –o %x.o%j
#SBATCH –e %x.e%j
#SBATCH –t 00:10:00

Export OMP_NUM_THREADS=1

mpirun ./piMPI.x
```

```
$ sbatch PiMPI.sh
```

Compile & Job Script 사용 예제: (PI 코드)

- Cuda(piCuda.cu)

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

#define TRIALS_PER_THREAD 4096
#define NUM_BLOCK  256  // Number of thread blocks
#define NUM_THREAD  256  // Number of threads per block
#define NBIN 268435456  // Number of bins 4096*256*256
#ifdef DP
typedef double Real;
#define PI  3.14159265358979323846  // known value of pi
#else
typedef float Real;
#define PI 3.1415926535  // known value of pi
#endif

int tid;
Real pi = 0;
__global__ void cal_pi(Real *sum, int nbin, Real step, int nthreads, int nblocks) {
          int i;
          Real x;
          int idx = blockIdx.x*blockDim.x+threadIdx.x;
          for (i=idx; i< nbin; i+=nthreads*nblocks) {
                    x = (i+0.5)*step;
                    sum[idx] += 4.0/(1.0+x*x);
          }
}
```

Compile & Job Script 사용 예제: (PI 코드)

- Cuda(piCuda.cu)

```
int main(void) {
          clock_t start,end;
          dim3 dimGrid(NUM_BLOCK,1,1);  // Grid dimensions
          dim3 dimBlock(NUM_THREAD,1,1);  // Block dimensions
          Real *sumHost, *sumDev;  // Pointer to host & device arrays
          printf("# of trials per thread = %d, # of blocks = %d, # of threads/block
= %d\n",TRIALS_PER_THREAD,NUM_BLOCK,NUM_THREAD);
          Real step = 1.0/NBIN;  // Step size
          size_t size = NUM_BLOCK*NUM_THREAD*sizeof(Real);  //Array memory size
          sumHost = (Real *)malloc(size);  //  Allocate array on host
          start=clock();
          cudaMalloc((void **) &sumDev, size);  // Allocate array on device
          cudaMemset(sumDev, 0, size);
          cal_pi <<<dimGrid, dimBlock>>> (sumDev, NBIN, step, NUM_THREAD, NUM_BLOCK);
          cudaMemcpy(sumHost, sumDev, size, cudaMemcpyDeviceToHost);
          for(tid=0; tid<NUM_THREAD*NUM_BLOCK; tid++) pi += sumHost[tid];
          pi *= step;
          end=clock();
          printf("GPU PI calculated in : %f s.\n",(end-start)/(float)CLOCKS_PER_SEC);
          #ifdef DP
          printf("GPU estimated PI = %20.18f [error of %20.18f]\n",pi,pi-PI);
          #else
          printf("GPU estimated PI = %f [error of %f]\n",pi,pi-PI);
          #endif
          free(sumHost);
          cudaFree(sumDev);
          return 0;
}
```

Compile & Job Script 사용 예제: (PI 코드)

- 컴파일

```
$ module purge
$ module load pgi/23.5
$ nvcc piCuda.c -o piCuda.x -fast
```

- PiCuda.sh

```
#!/bin/bash
#SBATCH -j Cuda_job
#SBATCH -p normal
#SBATCH --nodes=1
#SBATCH –o %x.o%j
#SBATCH –e %x.e%j
#SBATCH –t 00:10:00
#SBATCH --gres=gpu:1
export OMP_NUM_THREADS=1
./piCuda.x
```

```
$ sbatch PiCuda.sh
```

Compile & Job Script 사용 예제: (PI 코드)

- OpenACC(piACC.c)

```
#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");
```

Compile & Job Script 사용 예제: (PI 코드)

- OpenACC(piACC.c)

```
    num_threads=omp_get_num_threads();
    printf("# of threads : %d\n",num_threads);
#pragma acc parallel loop 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;
}
```

Compile & Job Script 사용 예제: (PI 코드)

- 컴파일

```
$ module purge
$ module load pgi/23.5
$ pgcc –acc piACC.c -o piACC.x –fast -Minfo
```

- PiACC.sh

```
#!/bin/bash
#SBATCH –J ACC_Jop
#SBATCH -p normal
#SBATCH --nodes=1
#SBATCH –o %x.o%j
#SBATCH –e %x.e%j
#SBATCH --gres=gpu:1
#SBATCH --time=00:10:00
Export OMP_NUM_THREADS=10
./piACC.x
```

```
$ qsub PiACC.sh
```

Q&A


Thank
you!