

Advanced Computing Jobs:

Develop Multicore and GPU HPC
Programming on SaaS Environment
using Jupyter Lab

Chan-Hin IONG
2023/08/01

1. Introduce NVIDIA SDK
2. Jupyter Lab
3. Simple way to monitor cpu cores and GPU
4. Running C++ 17 example on Jupyter Lab
5. Running Larger C++ example
6. OpenMP for multicore and GPU
7. Running OpenMP simple examples

Compilers in NVIDIA HPC SDK

nvc

nvc is a C11 compiler for NVIDIA GPUs and AMD, Intel, OpenPOWER, and Arm CPUs. It invokes the C compiler, assembler, and linker for the target processors with options derived from its command line arguments. nvc supports ISO C11, supports GPU programming with OpenACC, and supports multicore CPU programming with OpenACC and OpenMP.

nvc++

nvc++ is a C++17 compiler for NVIDIA GPUs and AMD, Intel, OpenPOWER, and Arm CPUs. It invokes the C++ compiler, assembler, and linker for the target processors with options derived from its command line arguments. nvc++ supports ISO C++17, supports GPU programming with C++17 parallel algorithms (pSTL) and OpenACC, and supports multicore CPU programming with OpenACC and OpenMP.

nvfortran

nvfortran is a Fortran compiler for NVIDIA GPUs and AMD, Intel, OpenPOWER, and Arm CPUs. It invokes the Fortran compiler, assembler, and linker for the target processors with options derived from its command line arguments. nvfortran supports ISO Fortran 2003 and many features of ISO Fortran 2008, supports GPU programming with CUDA Fortran and OpenACC, and supports multicore CPU programming with OpenACC and OpenMP.








nvcc

nvcc is the CUDA C and CUDA C++ compiler driver for NVIDIA GPUs. nvcc accepts a range of conventional compiler options, such as for defining macros and include/library paths, and for steering the compilation process. nvcc produces optimized code for NVIDIA GPUs and drives a supported host compiler for AMD, Intel, OpenPOWER, and Arm CPUs.

<https://docs.nvidia.com/hpc-sdk/archive/20.11/index.html>

<https://dicos.grid.sinica.edu.tw/dockerapps/>

Jupyter

 Jupyter Lab Version: GPU with Tensorflow v1 Resources: 90% <div><div></div></div> <button>Launch ▾</button>	 Jupyter Lab gpu 3090 Version: GPU with Tensorflow 3090 Resources: 16% <div><div></div></div> <button>Open</button> <button>Delete</button>	 Jupyter Lab GPU 1080ti Version: GPU with Tensorflow v2 Resources: 6% <div><div></div></div> <button>Launch ▾</button>
 Jupyter Lab GPU V100 Version: GPU with Tensorflow V100 Resources: 15% <div><div></div></div> <button>Launch ▾</button>	 Jupyter Lab GPU A100 Version: GPU with Tensorflow A100 Resources: 25% <div><div></div></div> <button>Launch ▾</button>	 Jupyter Lab Cryocare GPU Version: GPU with 1080ti Resources: 6% <div><div></div></div> <button>Launch ▾</button>
 Jupyter Lab GPU A100 Version: GPU with Tensorflow v2.6 Resources: 25% <div><div></div></div> <button>Launch ▾</button>		

CPU monitor

```
top - 04:10:12 up 2 days, 1:41, 0 users, load average: 0.60, 0.37, 0.28
Tasks: 12 total, 3 running, 9 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.4 us, 1.2 sy, 0.0 ni, 97.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 52811904+total, 39151625+free, 17393880 used, 11920891+buff/cache
KiB Swap: 0 total, 0 free, 0 used. 50929052+avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
36457	chiong	20	0	22.4g	16.8g	2076	R	100.0	3.3	0:03.26	a.out
36503	chiong	20	0	30796	14028	1476	R	2.0	0.0	0:00.02	nvidia-smi
200	chiong	20	0	53648	2036	1472	S	1.0	0.0	0:16.24	watch
1	root	20	0	13956	3828	1312	S	0.0	0.0	0:00.43	start_jupyterla
35	root	20	0	82228	2336	1524	S	0.0	0.0	0:00.00	su
36	chiong	20	0	11692	1508	1296	S	0.0	0.0	0:00.00	bash
49	chiong	20	0	398000	70012	7532	S	0.0	0.0	0:21.23	jupyter-lab
83	chiong	20	0	11964	2132	1560	S	0.0	0.0	0:00.27	bash
91	chiong	20	0	11964	2068	1520	S	0.0	0.0	0:00.32	bash
107	chiong	20	0	11964	2100	1544	S	0.0	0.0	0:00.17	bash
33578	chiong	20	0	56200	2012	1464	R	0.0	0.0	0:00.26	top
36502	chiong	20	0	53648	564	0	S	0.0	0.0	0:00.00	watch

single core job

```
top - 04:08:10 up 2 days, 1:39, 0 users, load average: 0.29, 0.23, 0.23
Tasks: 10 total, 2 running, 8 sleeping, 0 stopped, 0 zombie
%Cpu(s): 12.1 us, 2.8 sy, 0.0 ni, 85.1 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 52811904+total, 37466662+free, 34243472 used, 11920894+buff/cache
KiB Swap: 0 total, 0 free, 0 used. 49245321+avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
34663	chiong	20	0	22.8g	22.4g	2128	R	704.0	4.4	0:19.59	a.out
33578	chiong	20	0	56200	2012	1464	R	1.0	0.0	0:00.12	top
1	root	20	0	13956	3828	1312	S	0.0	0.0	0:00.43	start_jupyterla
35	root	20	0	82228	2336	1524	S	0.0	0.0	0:00.00	su
36	chiong	20	0	11692	1508	1296	S	0.0	0.0	0:00.00	bash
49	chiong	20	0	398000	70012	7532	S	0.0	0.0	0:20.80	jupyter-lab
83	chiong	20	0	11964	2132	1560	S	0.0	0.0	0:00.26	bash
91	chiong	20	0	11964	2068	1520	S	0.0	0.0	0:00.32	bash
107	chiong	20	0	11964	2100	1544	S	0.0	0.0	0:00.17	bash
200	chiong	20	0	53648	2036	1472	S	0.0	0.0	0:15.49	watch

multicore job
cpu usage > 100%

GPU monitor

```
Every 0.1s: nvidia-smi                                     Fri Jul 28 04:15:49 2023
23
Fri Jul 28 04:15:49 2023
```

NVIDIA-SMI 535.54.03			Driver Version: 535.54.03		CUDA Version: 12.2	
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr. ECC
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M. MIG M.
0	NVIDIA GeForce RTX 3090	On	00000000:A1:00.0	Off		N/A
0%	41C	P2	108W / 350W	262MiB / 24576MiB	0%	Default N/A

Processes:						
GPU	GI	CI	PID	Type	Process name	GPU Memory Usage
	ID	ID				

Modern C++ new features for Parallel computing

Algorithms and execution policies

Basic for-loop

```
for (Index_t i = 0; i < data.size(r); ++i)
{
}
```

Parallel algorithm for-loop running on Multicore or GPU

```
std::for_each_n(std::execution::par, counting_iterator(0), data.size(r),
[=, &domain](Index_t i)
{
}
);
```


To download examples source code:

```
git clone https://github.com/ASGCOPS/Advanced\_Computing\_Job\_2023
```

```
unzip HPC_src.zip
```

Simple code for multicore and GPU

```
1 #include <vector>
2 #include <iostream>
3 #include <fstream>
4 #include <random>
5 #include <string>
6
7 #include <algorithm>
8 #include <execution>
9
10 int main()
11 {
12     int N=4000;
13
14     std::vector<int> random_number;
15
16     // Create random value vector by CPU on system memory.
17     for (int x=0;x<N;x++)
18     {
19         int random_int;
20         random_int=rand() % 90000;
21         random_number.push_back(random_int);
22     }
23
24     // Sort random number with MCORE or GPU.
25     std::sort(std::execution::par, random_number.begin(),
26             random_number.end());
27
28     // Print the sort result on screen.
29     for (int i=0;i<20;i++)
30     {
31         std::cout<<random_number[i]<<std::endl;
32     }
33 }
```

Running on single core CPU
cpu usage $\leq 100\%$

Running on multicore CPU
cpu usage $> 100\%$

To compile the code and run :

(1) Setup the NVIDIA HPC SDK environment:

```
source /cvmfs/cvmfs.grid.sinica.edu.tw/hpc/nvhpc_sdk/2021_217/setup.sh
```

(2) Change your working directory:

```
cd 01_c_plus_plus_simple_example_mcore_gpu
```

(3) Compile source code for multicore:

```
nvc++ -gpu=cc80 -stdpar=multicore example.cc
```

(4) Or compile source code for GPU:

```
nvc++ -gpu=cc80 -stdpar=gpu example.cc
```

(5) run

```
./a.out
```

For GPU A100 and 3090, the GPU capability is cc80

Larger example: LULESH

<https://github.com/LLNL/LULESH/tree/2.0.2-dev/stdpar>

C++ Algorithms / Policies in LULESH

```
std::for_each_n(std::execution::par, counting_iterator(0), numElem,  
    [=, &domain](Index_t i) {  
        sigxx[i] = sigyy[i] = sigzz[i] = -domain.p(i) - domain.q(i);  
    });
```

```
std::for_each(std::execution::par, domain.symmX_begin(),  
    domain.symmX_begin() + numNodeBC, [&domain](Index_t symmX) {  
        domain.xdd(symmX) = Real_t(0.0);  
    });
```

```
std::transform(std::execution::par, compression, compression + length, bvc,  
    [=](Real_t compression_i) {  
        return cls * (compression_i + Real_t(1.0));  
    });
```


To compile the code and run :

(1) Setup the NVIDIA HPC SDK environment:

```
source /cvmfs/cvmfs.grid.sinica.edu.tw/hpc/nvhpc_sdk/2021_217/setup.sh
```

(2) Change your working directory:

```
cd 02_c_plus_plus_LULUSH_mcore_gpu/build
```

(3) Edit the Makefile:

For multicore:

```
CXXFLAGS = -w -fast -Mnuniform -Mfprelaxed -stdpar=multicore -std=c++11 -DUSE_MPI=0
```

For GPU:

```
CXXFLAGS = -w -fast -Mnuniform -Mfprelaxed -stdpar=gpu -std=c++11 -DUSE_MPI=0
```

(4) Compile

```
make clean
```

```
make all
```

(5) Run

```
./lulesh2.0
```

Makefile

```
1 # Build with nvc++, with parallel algorithm support turned on.
2
3 SHELL = /bin/sh
4 .SUFFIXES: .cc .o
5
6 LULESH_EXEC = lulesh2.0
7
8 CXX = nvc++
9
10 SOURCES2.0 = \
11     lulesh.cc \
12     lulesh-comm.cc \
13     lulesh-viz.cc \
14     lulesh-util.cc \
15     lulesh-init.cc
16 OBJECTS2.0 = $(SOURCES2.0:.cc=.o)
17
18
19 CXXFLAGS = -w -fast -Mnuniform -Mfprelaxed -stdpar=multicore -std=c++11 -DUSE_MPI=0
20
```


OpenMP for MCORE / GPU

Basic For-loop

```
for (Index_t i = 0; i < data.size(r); ++i)
{
}
```

OpenMP Directive

```
#pragma omp target teams distribute parallel for
for (Index_t i = 0; i < data.size(r); ++i)
{
}
```


Simple OpenMP example

```
1 #include<stdio.h>
2 #include <stdlib.h>
3
4 using namespace std;
5
6 #define N 2000000000
7
8 int main()
9 {
10     //Single core or CPU session :
11     int* a = (int*)malloc(sizeof(int) * N);
12     int* b = (int*)malloc(sizeof(int) * N);
13     int* c = (int*)malloc(sizeof(int) * N);
14
15     for (int i = 0; i < N; i++)
16     {
17         a[i] = 9;
18         b[i] = 1;
19         c[i] = 0;
20     }
21
22
23     // Multi-core or GPU session :
24     #pragma omp target device(0) map(to: a[0:N-1]) map(to: b[0:N-1])
25     map(from: c[0:N-1])
26     {
27         #pragma omp teams distribute parallel for
28         for (int i = 0; i < N; i++)
29             c[i] += a[i] + b[i];
30
31     }
32 }
33
```

Initialize arrays by single core cpu
cpu usage $\leq 100\%$

Calculate by multicore cpu
cpu usage $> 100\%$

Transfer data to GPU memory
map(to: a[0:N-1])

Calculate $c[i] += a[i] + b[i]$ on GPU

Get result from GPU memory
map(from: c[0:N-1])

To compile the code and run :

(1) Setup the NVIDIA HPC SDK environment:

```
source /cvmfs/cvmfs.grid.sinica.edu.tw/hpc/nvhpc_sdk/2021_217/setup.sh
```

(2) Change your working directory:

```
cd 03_omp_simple_example_mcore_gpu
```

(3) Compile source code for multicore:

```
nvc++ -gpu=cc80 -stdpar=multicore 01_omp_target_study.cc
```

(4) Or compile source code for GPU:

```
nvc++ -gpu=cc80 -stdpar=gpu 01_omp_target_study.cc
```

(5) run

```
./a.out
```


Simple OpenMP example

```
1 // openmp reduction test
2
3 #include <stdlib.h>
4 #include <stdio.h>
5 #include <math.h>
6
7 #define COUNT 20000000
8
9 int main()
10 {
11     int sum = 0; // Assign an initial value on system memory.
12
13     #pragma omp teams distribute parallel for reduction(+:sum)
14     for(int i = 0; i < COUNT; i++)
15     {
16         // Edit your own formula here:
17         sum += rand() % 3;
18     }
19
20     printf("\n Sum: %d\n\n", sum);
21
22     return 0;
23 }
```

“+” directive in OpenMP

To compile the code and run :

(1) Setup the NVIDIA HPC SDK environment:

```
source /cvmfs/cvmfs.grid.sinica.edu.tw/hpc/nvhpc_sdk/2021_217/setup.sh
```

(2) Change your working directory:

```
cd 03_omp_simple_example_mcore_gpu
```

(3) Compile source code for multicore:

```
nvc++ -gpu=cc80 -stdpar=multicore 02_omp_reduction.cc
```

(4) run

```
./a.out
```