



Hands-on (OpenACC)

OpenACC Documentation

<https://www.openacc.org/resources>

Guides

Introduction to OpenACC Quick Guides

- [OpenACC Programming and Best Practices Guide \(PDF\)](#)
- [OpenACC 2.7 API Reference Card](#)

Best practice guide

Tutorials

Video tutorials to help start with OpenACC and advance your skills

- [3 Steps To More Science Tutorial](#)
- [Youtube Tutorials](#)

Courses and Training Events

- [Upcoming Open Hackathons and Bootcamps](#)
- [OpenACC online classes with hands-on labs](#)

Code Samples

Code samples for hands-on experience

- [OpenACC GitHub Repository](#)
- [Open Hackathons GitHub Repository](#)

Books



OpenACC for Programmers: Concepts and Strategies



Parallel Programming with OpenACC



**Programming Massively Parallel Processors, Third Edition:
A Hands-on Approach**

Specification

[OpenACC 3.3 Specification](#)

Specification

Teaching Materials

Slides, Code Samples, Videos, Books and more to teach OpenACC

DAXPY

$$y[i] = a * x[i] + y[i]$$

```
//  
// daxpy: y[i] = a * x[i] + y[i]  
//  
void daxpy(int n, double a, double *x, double *y)  
{  
    int i;  
    for (i = 0; i < n; i++) {  
        y[i] += a*x[i];  
    }  
}
```

3つのタスク

- Task 1: 配列を Unified Memory で確保する
- Task 2: GPU カーネルを実行できるようにする
- Task 3: CPU と GPU 間で明示的に配列を確保、移動する

ディレクトリ構成、進め方

- タスクの進め方
 - ソースファイル (daxpy.c)、Makefile を修正して、正しく動作するようにする

reference	... リファレンス CPU コード
task1	... タスク 1
task1-solution	... タスク 1 の答え
task2	... タスク 2
task2-solution	... タスク 2 の答え
task3	... タスク 3
task3-solution	... タスク 3 の答え

Task 0

教材コピー, ビルド, ジョブ投入

```
$ cp /gs/bs/tgz-training/hands-on.tar.gz .  
$ tar xvzf hands-on.tar.gz
```

```
$ cd reference  
$ module load nvhpc  
$ make  
  
$ qsub -g tgz-training -ar 699 ./batch.sh  
Your job ***** ("batch.sh") has been submitted  
  
$ qstat  
  
$ ls batch.sh.*  
batch.sh.e***** batch.sh.o*****
```

```
#!/bin/sh  
#$ -cwd  
  
#$ -l node_q=1  
#$ -l h_rt=00:05:00  
  
module load nvhpc  
  
./daxpy  
  
batch.sh
```

Task 1

配列を Unified Memory で確保する

Unified Memory で配列を確保する

- コンパイル時のオプションで、`-acc=gpu -gpu=managed -Minfo=accel` を指定
 - `-acc` : OpenACC を有効化
 - `gpu` : NVIDIA GPU 向けにビルド
 - `-Minfo` : コンパイル時のフィードバックを表示
 - `accel` : OpenACC に関連するフィードバックを指定
 - `-gpu` : GPU コード生成に関する詳細を指定
 - `managed` : Unified Memory を有効化

Task 2

GPU カーネルを実行できるようにする

- ループを並列化

```
#pragma acc parallel loop
for(int i = 0; j < N; i++)
  < Your code >
```

- コンパイラフィードバック

```
daxpy:
  24, Generating implicit firstprivate(n,a)
    Generating NVIDIA GPU code
    24, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
  24, Generating implicit copy(y[:n]) [if not already present]
    Generating implicit copyin(x[:n]) [if not already present]
```

GPU は使われている？

プロファイラ (nsys) で確認する

```
#!/bin/sh
#$ -cwd

#$ -l node_q=1
#$ -l h_rt=00:05:00

module load nvhpc/24.1

nsys profile --stats=true -o report ./daxpy
nsys stats --report cuda_gpu_sum report.sqlite
```

batch.sh

```
...

Processing [report.sqlite] with [/apps/t4/rhel9/isv/nvidia/hpc_sdk/Linux_x86_64/24.1/profilers/Nsight_Systems/host-linux-
x64/reports/cuda_gpu_sum.py]...

** CUDA GPU Summary (Kernels/MemOps) (cuda_gpu_sum):
```

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Category	Operation
64.5	5,140,019	1	5,140,019.0	5,140,019.0	5,140,019	5,140,019	0.0	CUDA_KERNEL	_7daxpy_c_daxpy_24_gpu
26.0	2,075,393	673	3,083.8	2,463.0	2,207	35,616	2,909.1	MEMORY_OPER	[CUDA memcpy Unified Host-to-Device]
9.4	752,029	116	6,483.0	2,783.5	2,335	36,064	8,115.8	MEMORY_OPER	[CUDA memcpy Unified Device-to-Host]

batch.sh.o*

Task 3

CPU と GPU 間で明示的に配列を確保、移動する

- CPU と GPU 間で明示的にデータを移動

```
#pragma acc data clauses
{
  < Sequential and/or Parallel code >
}
```

- copyin : データ領域開始時に CPU -> GPU
- copyout : データ領域終了時に CPU <- GPU
- copy : copyin と copyout の両方
- create : GPU 上にメモリを確保

- コンパイラフィードバック

```
daxpy:
    24, Generating present(y[:],x[:])
        Generating implicit firstprivate(n,a)
        Generating NVIDIA GPU code
    24, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
main:
    57, Generating copy(y[:1000000]) [if not already present]
        Generating copyin(x[:1000000]) [if not already present]
```

データはどう転送されている？

プロファイラ (nsys) で確認する

```
#!/bin/sh
#$ -cwd

#$ -l node_q=1
#$ -l h_rt=00:05:00

module load nvhpc/24.1

nsys profile --stats=true -o report ./daxpy
nsys stats --report cuda_gpu_sum report.sqlite
```

batch.sh

```
...

Processing [report.sqlite] with [/apps/t4/rhel9/isv/nvidia/hpc_sdk/Linux_x86_64/24.1/profilers/Nsight_Systems/host-linux-x64/reports/cuda_gpu_sum.py]...

** CUDA GPU Summary (Kernels/MemOps) (cuda_gpu_sum):
```

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Category	Operation
81.2	3,332,483	1	3,332,483.0	3,332,483.0	3,332,483	3,332,483	0.0	MEMORY_OPER	[CUDA memcpy Device-to-Host]
18.6	765,241	2	382,620.5	382,620.5	314,301	450,940	96,618.4	MEMORY_OPER	[CUDA memcpy Host-to-Device]
0.2	6,688	1	6,688.0	6,688.0	6,688	6,688	0.0	CUDA_KERNEL	_7daxpy_c_daxpy_24_gpu

batch.sh.o*

