

KOBE HPCサマースクール (初級)2023

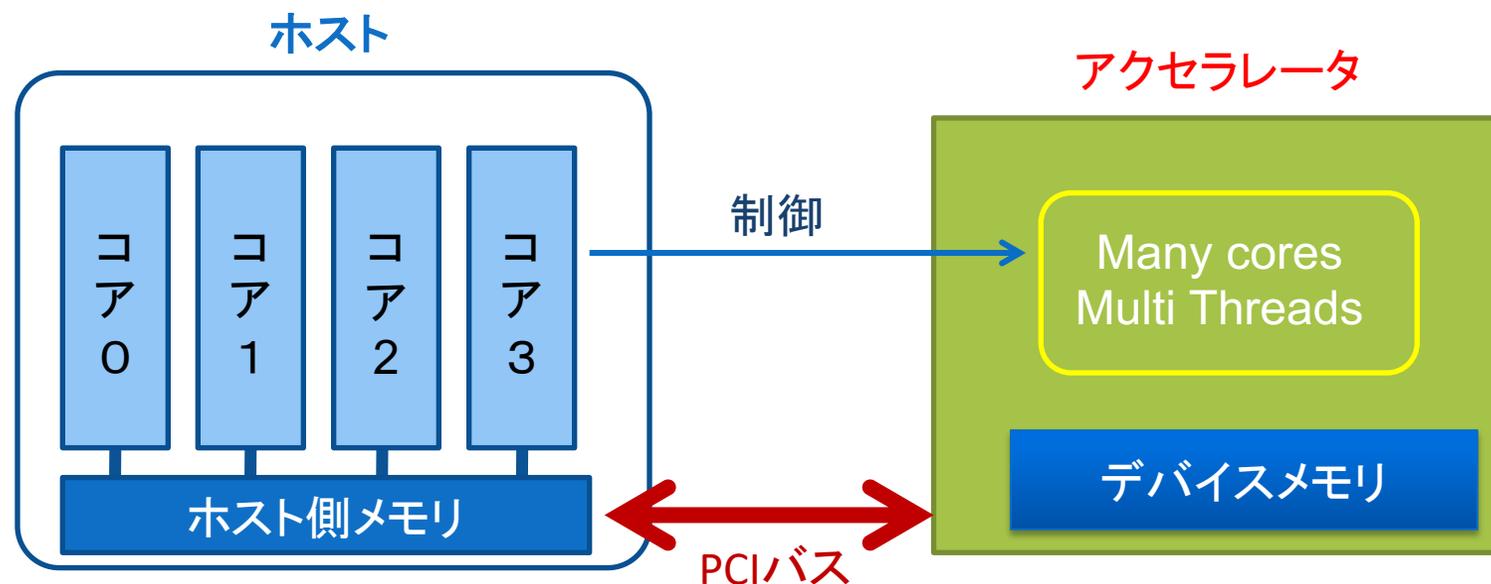
講義資料 2023/09/22

兵庫県立大学 情報科学研究科
安田 修悟

OpenACCによるアクセラレータ計算

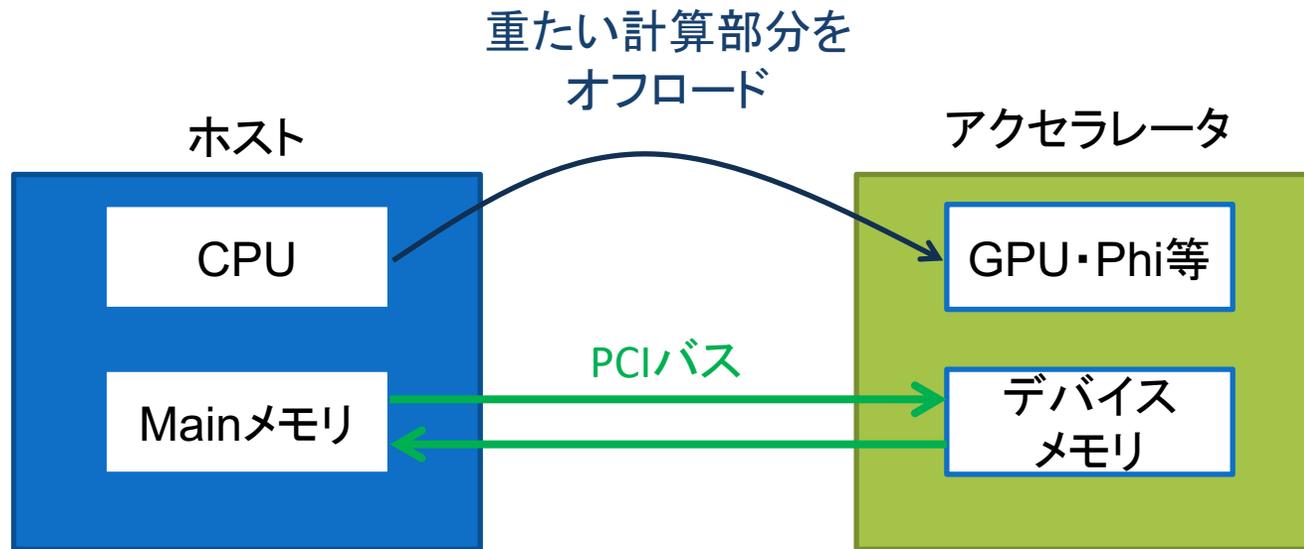
アクセラレータ

- CPUの処理を一部代替して全体の処理の効率を向上させる装置
 - GPGPU (NVIDIA), Xeon Phi (Intel), APU (AMD)など



アクセラレータの実行イメージ

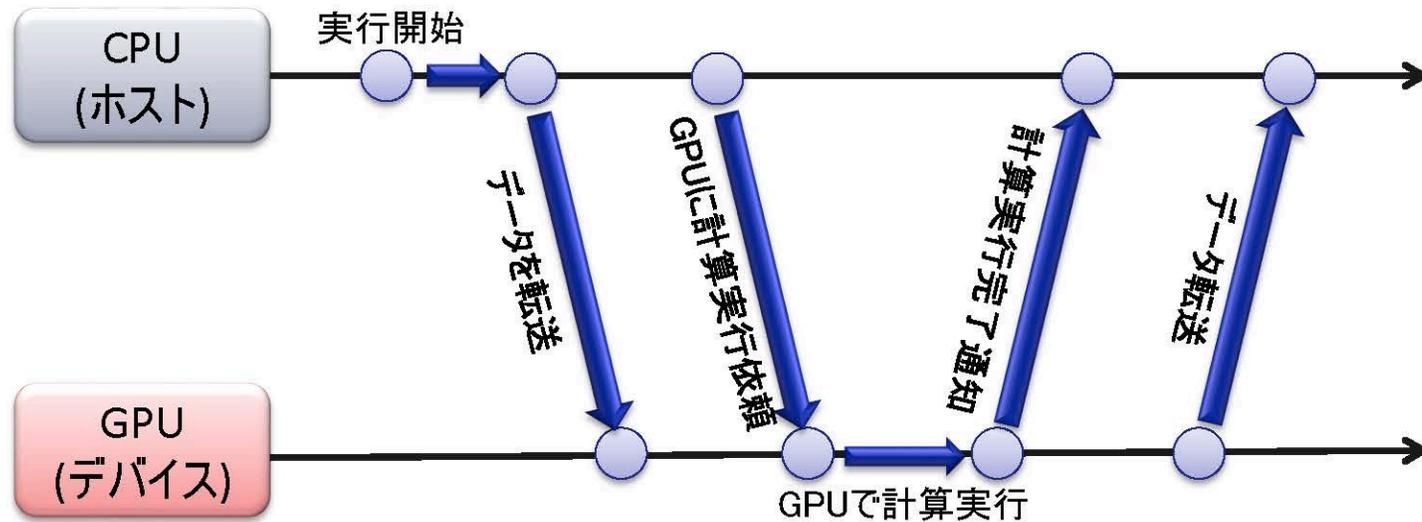
ハイブリット構成（CPU+アクセラレータ）



※データのオフロード通信がボトルネックとなる

- CPUのメモリ空間とデバイスのメモリ空間が異なる
- 一般的なメモリ帯域幅に比べてPCIバスでの低速な転送

GPUでのプログラム実行イメージ



- GPU(デバイス)ではCPU(ホスト)から制御をします。
- GPU(デバイス)の計算に必要なデータはCPU(ホスト)から転送します。
- GPU(デバイス)で計算した結果はCPU(ホスト)に転送します。

OpenACC

- Accelerators用のAPIの一つ。
- CPUからデバイスに処理をオフロードする際に使用する。
- Fortran/C/C++言語で**ディレクティブ**を指定
(**OpenMPと同様**)
- CUDA等の専用プログラムに比べて、導入が容易。

OpenACC

- 基本フォーマット

```
#pragma _acc _ディレクティブ _節1 _節2 _...  
{  
    構造化ブロック(途中で抜け出したり、終了しない)  
}
```

例

```
#pragma _acc _kernels _copy(a)  
for(i=0;i<imax;i++){  
    a[i]=a[i]*a[i];  
    ....  
}
```

OpenACC

- 基本フォーマット (Fortranの場合)

```
!$acc _ディレクティブ_節1_節2_...  
      構造化ブロック  
!$acc end ディレクティブ  
  
!$acc _kernels_ copy(a)  
do i=0, imax  
    a[i]=a[i]*a[i];  
    ....  
end do  
!$acc end kernels
```

基本的なディレクティブ構文

- ① Accelerator compute構文
 - オフロードするループ対象部分を指定する。
 - kernels構文やparallel構文がある
- ② DATA構文
 - ホストとアクセラレータのデータ転送を明示的に指示する。
 - 高速化の鍵を握る
- ③ Loop構文
 - ①のオフロード領域のループのベクトル長や並列分割の詳細を明示的に指示する。

NVIDIA HPC SDK コンパイラー

- Module環境の設定

`module load nvhpc`

- `module_list`でintelが設定されている場合
`module_unload_intel` や `module_purge`
でIntel環境を先にunloadする。

- OpenACCのコンパイル

`nvc -acc prog.c -o acc.out`

- “-acc”はOpenACCディレクティブを有効にするためのオプション。

NVIDIA HPC SDK コンパイラー

- OpenMPのコンパイル

```
nvc -mp prog.c -o mp.out
```

“-mp”はOpenMPディレクティブを有効にするためのオプション。

- 最適化オプション

- “-Mvect=simd” : SSEやAVXのSIMD命令を生成.

- “-fast”:一般的な最適化オプション

- “-O3”, “-O4”: より高度な最適化オプション
(“-fast”より後に入れる

例: nvc -fast -O3 program.c)

NVIDIA HPC SDK コンパイラー

- 環境変数(実行前に設定, [run_acc.sh](#)参照)
 - NVCOMPILER_ACC_TIME
 - 実行後に簡易プロファイル情報を標準出力に出力。
 - 0(ゼロ)を指定すると機能を抑制(デフォルト)、ゼロ以外の整数値で機能する。
 - NVCOMPILER_ACC_NOTIFY
 - デバイスの実行イベントを表示する
 - 整数値1: Kernel launch のイベントを出力、整数値2: データ転送のイベントの出力、など
 - 0(ゼロ)を指定すると機能を抑制(デフォルト)

GPUキューの実行

- インタラクティブ実行

`qsub -I -q G -l select=1:ncpus=N2:ngpus=N4`

- N₂: CPUコア数(1~40), N₄:GPUデバイス数(1~8)
- Ngpus=N₄を指定しない場合, GPUデバイスは利用しない.
- GPUを搭載したノードにてプログラムを実行。

GPUキューの実行

- ホストプログラムの並列化に応じてジョブスクリプトを作成. (以下の例は, ホストが並列化無しの場合)

```
#!/bin/bash
#PBS_-q_ G
#PBS_-l_select=1:ncpus=4:ngpus=1   ・利用するGPUカード数を指定
#PBS_-N_job_acc
#PBS_-j_oe
source_/etc/profile.d/modules.sh
module_purge   ・現在のモジュール環境を全て破棄
module_load_nvhpc   ・NVIDIA HPCコンパイラの環境設定
cd_ ${PBS_O_WORKDIR}
./a.out
```

OpenACCの演習 1

- OpenMPプログラム([pi_omp.c](#))をNVIDIA HPCコンパイラーでコンパイルし、スレッド並列を実行せよ.
- OpenACCプログラム([pi_acc.c](#))をコンパイルし、GPU計算を実行せよ.
・ スレッド並列とGPU計算の計算時間を比較せよ.
 - * 注) [pi_acc.c](#)のnmaxは40000程度以下に設定してください.
- 環境変数(NVCOMPILER_ACC_TIME or NVCOMPILER_ACC_NOTIFY)を設定してデバイスの実行プロファイルを確認しなさい.

OpenACCの演習 2

- `pi2_omp.c` と `pi2_acc.c` をそれぞれ実行し, 先の演習の結果と比較しなさい.
 - `pi2_acc.c`のデバイス実行プロファイルを分析し, 全体の計算時間が短縮されない原因について考えなさい.

pi_acc.cの実行プロファイル

Accelerator Kernel Timing data

main NVIDIA devicenum=0 time(us): 41

16: compute region reached 1 time (compute構文が1回, kernel構文・reduction演算)

18: kernel launched 1 time

grid: [65535] block: [128]

elapsed time(us): total=12,175 max=12,175 min=12,175 avg=12,175

18: reduction kernel launched 1 time

grid: [1] block: [256]

elapsed time(us): total=70 max=70 min=70 avg=70

16: data region reached 2 times (data構文が2回, copyin©out)

16: data copyin transfers: 1

device time(us): total=8 max=8 min=8 avg=8

23: data copyout transfers: 1

device time(us): total=33 max=33 min=33 avg=33

pi2_acc.cの実行プロファイル

main NVIDIA devicenum=0 time(us): 2,026,328

17: compute region reached 1 time

19: kernel launched 1 time

grid: [65535] block: [128]

elapsed time(us): total=19,479 max=19,479 min=19,479 avg=19,479

17: data region reached 2 times

24: data copyout transfers: 763

device time(us): total=987,059 max=1,393 min=1,235 avg=1,293

29: compute region reached 1 time

31: kernel launched 1 time

grid: [65535] block: [128]

elapsed time(us): total=15,533 max=15,533 min=15,533 avg=15,533

31: reduction kernel launched 1 time

grid: [1] block: [256]

elapsed time(us): total=114 max=114 min=114 avg=114

29: data region reached 2 times

29: data copyin transfers: 764

device time(us): total=1,039,257 max=1,372 min=19 avg=1,360

34: data copyout transfers: 1

device time(us): total=12 max=12 min=12 avg=12

**2回のdata構文が
ボトルネック!!**

DATA構文

- #pragma_acc_data_節
- 節には以下のようなものがある。(Kernel構文の節にも使える.)
 - copyin(a,b): 配列a, bをホストからアクセラレータにコピーする.
 - copyout(c): 配列cをアクセラレータからホストにコピーする.
 - copy(A): copyinとcopyoutの双方を行う.
 - create(B): アクセラレータでローカルに使用する配列Bの領域を作成.
 - present(C): 対象となる処理に入ったときにすでにデバイスメモリにある配列Cのデータを使用する.

DATA構文

- `pi2_acc.c`のプログラムに対して,
 1. GPU内だけで必要な変数を予め作成する.
`#pragma acc data create(f,x,y)`
{
.....
}
 2. Kernel構文でデータ属性を指示する.
 - `kernels present(f) copyin(dh)`
 - `kernels present(f) copy(sum)`

修正pi2_acc.cの実行プロファイル

main NVIDIA devicenum=0 time(us): 41

17: data region reached 2 times

20: compute region reached 1 time

22: kernel launched 1 time

grid: [65535] block: [128]

elapsed time(us): total=162,324 max=162,324 min=162,324 avg=162,324

20: data region reached 2 times

20: **data copyin transfers: 1**

device time(us): total=15 max=15 min=15 avg=15

32: compute region reached 1 time

34: kernel launched 1 time

grid: [65535] block: [128]

elapsed time(us): total=15,465 max=15,465 min=15,465 avg=15,465

34: reduction kernel launched 1 time

grid: [1] block: [256]

elapsed time(us): total=107 max=107 min=107 avg=107

32: data region reached 2 times

32: **data copyin transfers: 1**

device time(us): total=7 max=7 min=7 avg=7

37: **data copyout transfers: 1**

device time(us): total=19 max=19 min=19 avg=19

**data構文のボトルネックが解消
スカラ変数のデータコピーのみ**

OpenACCの演習3

1. `laplace_omp.c` をもとに, ラプラス方程式のOpenACCのプログラムを作成せよ.
 - ✓ OpenMPのディレクティブを全てOpenACCのディレクティブに変更.
`#pragma omp parallel for => #pragma acc kernels`
 - ✓ データ構文を使って, データコピーの回数が少なくなるように工夫する.
2. ベンチマークを実行してOpenMPとOpenACCで計算時間を比較せよ.
.

(補足) OpenACC Runtime関数

- `#include <openacc.h>`
- デバイスタイプを取得
`acc_device_t acc_get_device_type(void)`
- 利用するデバイスの総数を取得
`int acc_get_num_devices(acc_device_t t)`
- 利用するデバイスの番号をセット
`void acc_set_device_num(int n, acc_device_t t)`
- などなど

(発展)MPI / OpenACC

- Module環境
module load nvhpc/21.1 ※バージョンまで指定
module load mpt
- コンパイル
nvc -acc -Mmpi=sgimpi prog.c -o prog.out
- 実行 (4プロセス = 4GPU)
#PBS -l select=1:ncpus=4:mpiprocs=4:ngpus=4
...
mpiexec_mpt ./prog.out

(発展)MPI / OpenACC

- `laplace_acc_mpi.c`
ラプラス方程式の右辺の値が異なるケースをMPI並列で同時に計算するコード。

$$-\Delta T = -\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) = 2.5 * \text{myrank}$$

- 実行して結果を可視化で確認せよ。