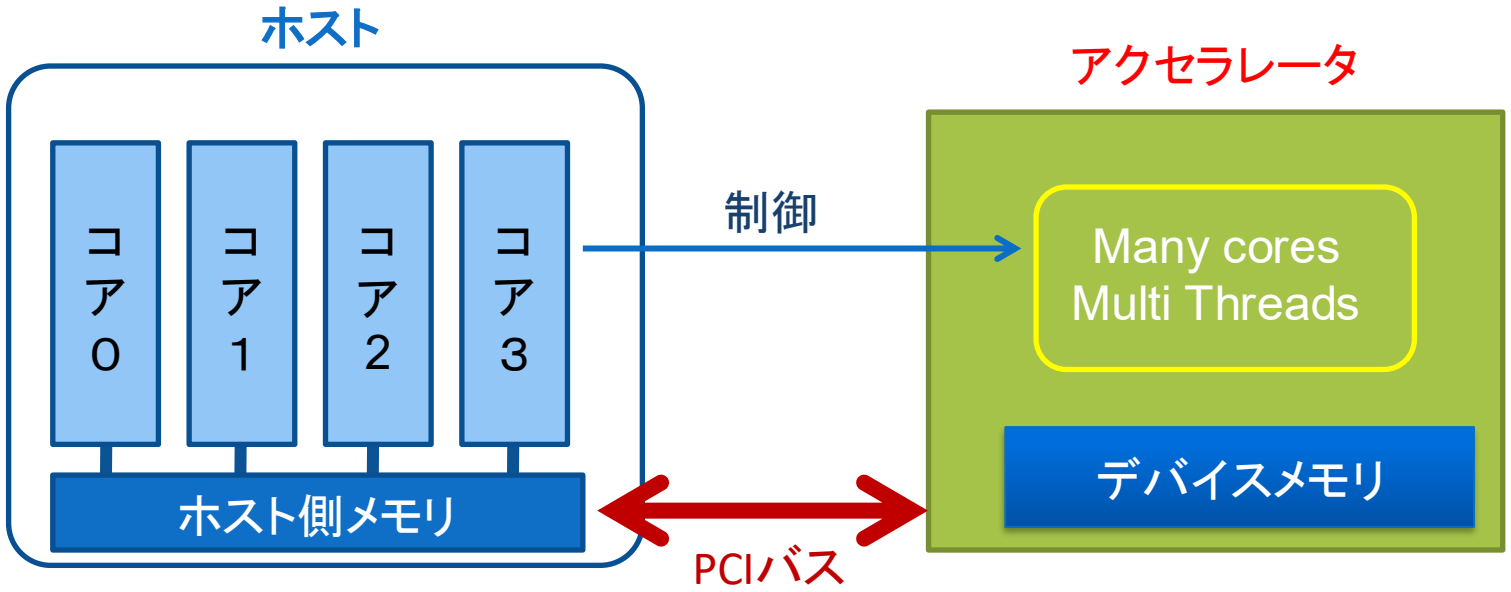
KOBE HPCサマースクール（初級）2026 講義資料 2026/09/18

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

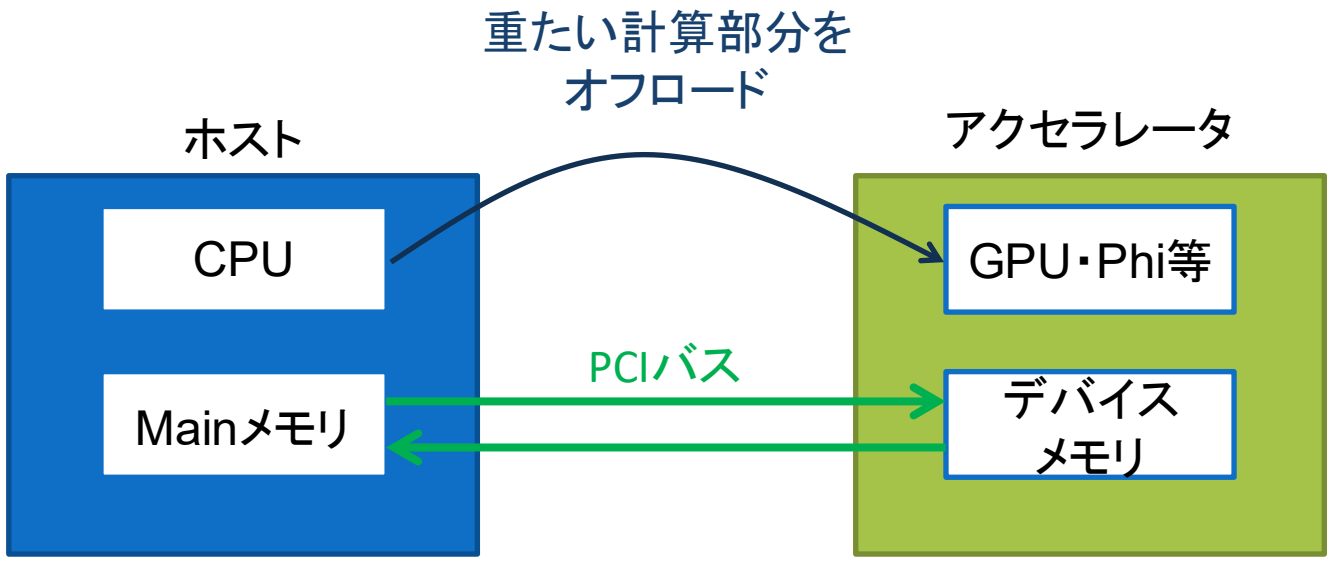
アクセラレータ

- 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

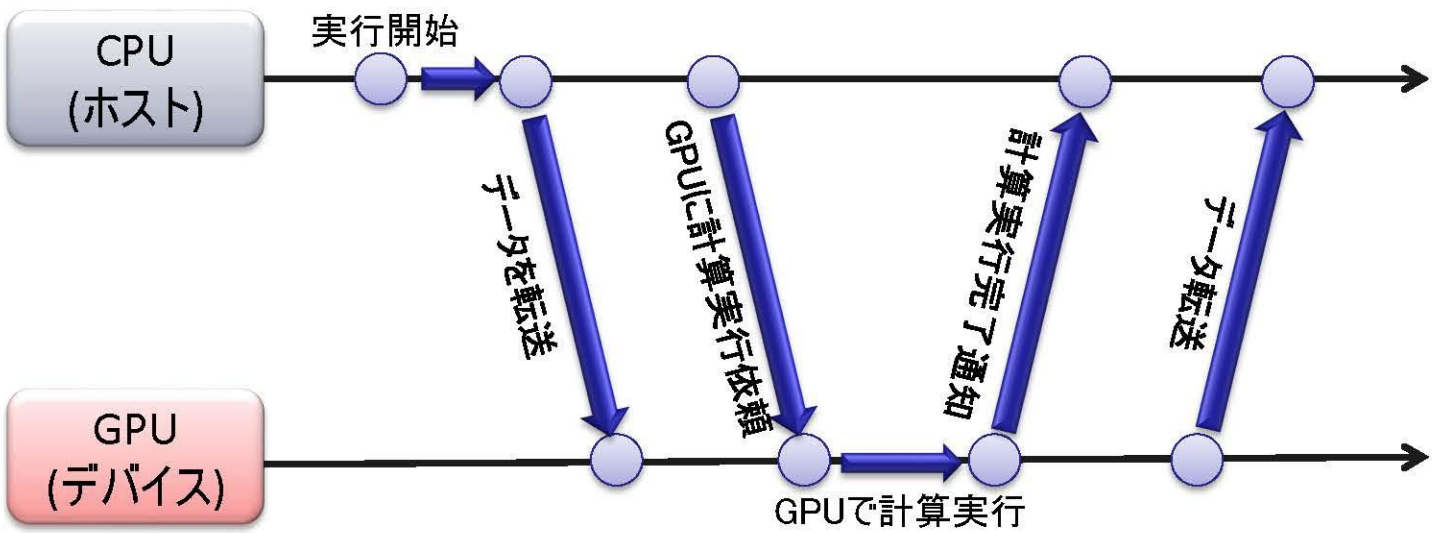
ハイブリッド構成 （CPU＋アクセラレータ）



※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)の計算で必要なデータは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

- アクセラレータ向けのディレクティブベースの並列プログラミングモデル。
- 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

- アクセラレータで実行する計算領域を指定する。
- kernels構文: 並列化可能な部分をコンパイラが解析
- parallel構文：並列実行可能であることをプログラマが明示する

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。
- 不要なデータ転送を減らすことが高速化の鍵となる

③ Loop構文

- ループの並列化方法を指定する。
- collapse：多重ループをまとめて並列化する。
- gang, worker, vector：GPU上での並列処理の割り当てを指定。

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

- コンパイル結果の診断レポート (コンパイルオプション)
  - “-Minfo=accel” : コンパイラがどう並列化したかを表示する

- 実行時の簡易プロファイル・通知（環境変数で設定）
  - 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
    - ディ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=N_2:ngpus=N_4

  – N2: CPUコア数（1～40），N4:GPUディバイス数（1～8）
  – Ngpus=N_4を指定しない場合，GPUディ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
#PBS␣-N␣job_acc
#PBS␣-j␣oe
source␣/etc/profile.d/modules.sh
module␣purge
module␣load␣nvhpc
cd␣${PBS_O_WORKDIR}
./a.out
```

・利用するGPUカード数を指定

・現在のモジュール環境を全て破棄

・NVIDIA HPCコンパイラの環境設定

演習 1 (kernels構文)

演習ファイル: /home/y273s011/share/khpc2026/20260918.tar

- OpenMPプログラム(gi_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(gi_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）gi_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）を設定してディバイスの実行プロファイルを確認しなさい.

parallel 構文とloop構文

並列領域と、並列化するループを明示する。

(gi_acc_para.c)

```
sum = 0.0;
#pragma acc parallel loop reduction(+:sum)
for(i=0;i<nmax;i++){
    double x = dh*i;

    for(j=0;j<nmax;j++){
        double y = dh*j;
        sum += exp(-x*x-y*y);
    }
}
```

並列化ループの指定

reduction変数を明示

並列領域

演習 2 (parallel構文)

- Kernels構文 (gi_acc.c)とParallel構文(gi_acc_para.c）のコンパイラの診断レポート(-Minfo=accel)と実行プロファイルを比較して、その違いを確認しなさい。

- gi_acc_para.cのParallel構文にLoop構文の*collapse(2)*節を追加したときのコンパイル診断レポートと実行プロファイルを確認し、先の2つとの違いを比較しなさい。

GPU計算におけるデータ転送の影響 (gi_acc.c vs. gi2_acc.c)

gi_acc.c
計算 → reduction
大規模データ転送なし

gi2_acc.c
配列 f の計算→ reduction
↑
データ転送が発生

gi_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 （sumのデータ転送(copyin/copyout)）
    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
```

gi2_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
```

配列fのデータ転送（copyout/copyin）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):デバイスメモリ上に配列Bの領域を確保する.
  - 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

- gi2_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)
{
.......
}
```

2. **Kernel構文でデータ属性を指示する.**
   - kernels present(f)
   - kernels present(f) copy(sum)

修正gi3_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
```

配列fの大規模なデータ転送が解消
スカラ変数のデータ転送のみ

演習3（DATA構文）

1. poisson_omp.c をもとに，ポアソン方程式の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” を “#pragma acc kernels”に変更.
   - ✓ 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. OpenMPとOpenACCで計算時間を比較せよ.

（補足1）MPI / OpenACC の演習

- poisson_acc_mpi.c
  ポアソン方程式の右辺の値が下記のようにrankごとに異なるケースをMPI並列で同時に計算するコードを完成させなさい。

$$-\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right)=2.5*(\mathrm{myrank}+1)$$

- 実行して結果を可視化で確認せよ。

（補足1）MPI / OpenACCの実行

- 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

（補足1）OpenACC Runtime関数

- #include <openacc.h>

- ディバイスタイプを取得
  acc_device_t acc_get_device_type(void)

- 利用するディバイスの総数を取得
  int acc_get_num_devices(acc_device_t t)

- 利用するディ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

（補足2）セルオートマトン　ルール184

- セルオートマトンとは、
  格子状のセルと単純な規則による，離散的計算モデル.
  現在の格子セルの状態から次の時刻の格子セルの状態が決まる.
- ルール184
  - 1つのセルに入ることのが出来る車両は1台だけ.
  - 車両は1タイムステップで右に1セル分だけ進める.
  - 1つ先のセルに車両が存在すれば，次のタイムステップでは先のセルに進めず，そのセルに留まる.
  - 1つ先のセルに車両が存在しなければ，次のタイムステップに先のセルに進む.

（補足2）ルール184

- 周期境界条件を考えて手でやってみよう
  - 1が車両有り，0が車両無しを表す．
  - 上の段から下の段に時間が進んでいく．

1	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**
1	0	1	0	1	0	0	1	0	1

（補足2）なぜルール184？

あるセルの次の状態はそのセルと前後のセルの現在の状態で決まる。

現在の状態	1	1	1
中央セルの次の状態		1	

現在の状態	1	1	0
中央セルの次の状態		0	

現在の状態	111	110	101	100	011	010	001	000
中央セルの次の状態	1	0						

（補足2）なぜルール184？

あるセルの次の状態はそのセルと前後のセルの現在の状態で決まる。

現在の状態	1	1	1
中央セルの次の状態		1	

現在の状態	1	1	0
中央セルの次の状態		0	

現在の状態	111	110	101	100	011	010	001	000
中央セルの次の状態	1	0	1	1	1	0	0	0

２進数の10111000を10進数に直すと 184

（補足2）ルール184の演習

- ルール184のOpenACCコード r184_acc.cとMPIコード r184_mpi.cを完成させなさい.
- r184_acc.c : データ構文を完成させなさい.
- r184_mpi.c : MPI_Allreduce( )を完成させなさい.