OpenACC

Introduction

全名 : Open Accelerators

- 透過compiler directive讓程式能在GPU上執行
- 和CUDA相比門檻降低許多

OpenACC vs CUDA

- CUDA
  - CudaMalloc(...)：宣告GPU上的記憶體
  - CudaMemcpy(...)：搬移資料
  - functionname<<<thread, blocks>>>(...)：要自己寫Cuda Kernel Function

=== 入門門檻高 ===

- OpenACC
  - 不用宣告device上的記憶體
  - #pragma acc data copy(...)：用簡單的clause就可以搬移資料
  - 直接用parallel region就可以port到GPU上

=== 可以快速上手 ===

OpenACC Directive

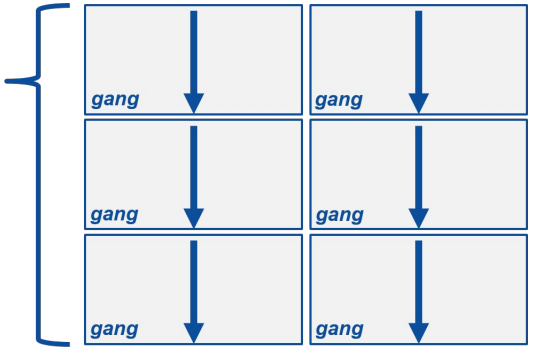
- #pragma acc <directive> <clause>
  - #pragma 是一種 compiler hint
  - acc 告訴compiler這是OpenACC的pragma
  - directive 是OpenACC告訴compiler指示
  - clause 是OpenACC對directive進行補充或優化的指示

OpenACC Directive

- #pragma acc parallel
  - parallel 告訴compiler這段程式碼要redundantly parallelize
  - redundantly parallelize：多餘地平行，並沒有分工

```
#pragma acc parallel
{
}
```

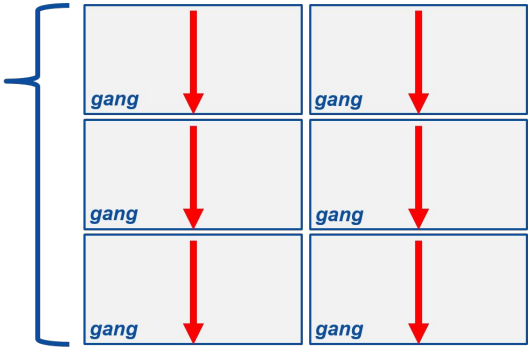
When encountering the ***parallel*** directive, the compiler will generate *1 or more parallel* ***gangs***, which execute redundantly.




```
#pragma acc parallel
{
    for(int i = 0; i < N; i++)
    {
        // Do Something
    }
}
```

This loop will be ***redundantly parallelized*** across the ***gangs***

This means that each ***gang*** will execute the entire loop



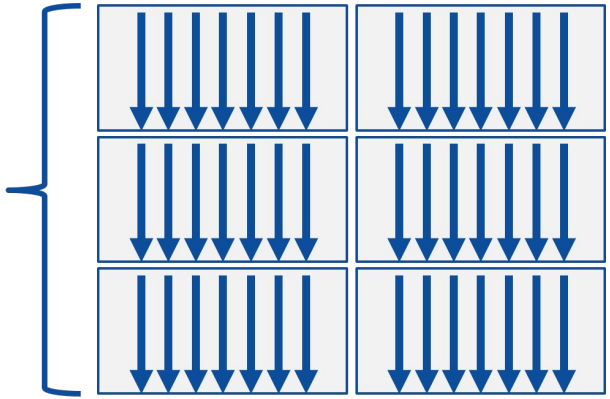
OpenACC Directive

- #pragma acc parallel loop
  - loop 告訴compiler這個loop要parallelize
  - 同時也告訴compiler這個loop是可以被安全地平行

```
#pragma acc parallel
{

    #pragma acc loop
    for(int i = 0; i < N; i++)
    {
        // Do Something
    }

}
```

The ***loop*** directive informs the compiler which loops to parallelize.

The iterations of the loop will be broken up evenly among the parallel ***gangs***.



The ***gangs*** will then execute in parallel with one another.

OpenACC Directive

- #pragma acc parallel loop reduction(<operation>:<target>)
  - reduction 告訴compiler某個目標要被reduce
  - reduce：對選定的目標進行全局的 operation操作

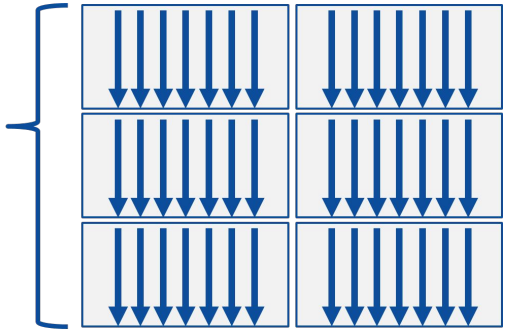
```
int sum = 0;

#pragma acc parallel loop reduction(+:sum)

for(int i = 0, i < N, i++) sum += i;
```

OpenACC Directive

- #pragma acc kernels
  - 全部行為交給compiler決定
  - 可以把sequential code也包進去
- #pragma acc kernels loop independent
  - 告訴compiler這個loop是可以被安全地平行，並強迫平行它

```
#pragma acc kernels
{
    for(int i = 0; i < N; i++)
    {
        // Do Something
    }
    for(int i = 0; i < M; i++)
    {
        // Do Something Else
    }
}
```

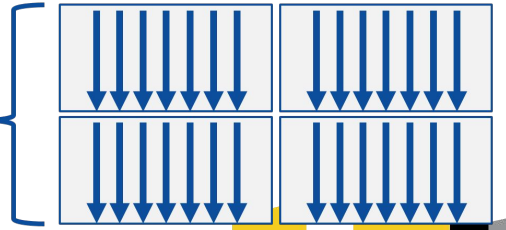
With the ***kernels*** directive, the ***loop*** directive is implied.



```
#pragma acc kernels
{
    for(int i = 0; i < N; i++)
    {
        // Do Something
    }
    for(int i = 0; i < M; i++)
    {
        // Do Something Else
    }
}
```

This process can happen multiple times within the ***kernels*** region.

Each loop can have a different number of gangs, and those gangs can be organized/optimized completely differently.



Inclass Lab

把TODO1和TODO2完成

- module purge
- module load nvhpc-hpcx/23.3
- cp -r /home/hpci24/share/openacc/ ~/
- cd openacc

=== 改code ===

- make
- srun --gres=gpu:1 ./laplace

這裡用的是unified memory，不用複製資料

```
[daniellin@hades01 lab1]$ srun --gres=gpu:1 ./laplace
Jacobi relaxation Calculation: 4096 x 4096 mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
total: 2.831587 s
```

沒加速32s > 加速後3s

Data Management

- #pragma acc data copy(...)
  - 把資料複製進GPU，在parallel region結束後將資料複製回CPU
- #pragma acc data copyin(...)
  - 把資料複製進GPU，在parallel region結束後刪除GPU上的資料
- #pragma acc data copyout(...)
  - 把資料複製回CPU，在parallel region結束後刪除GPU上的資料
- #pragma acc data create(...)
  - 在GPU宣告一個空間，不進行任何複製動作
  - 有暫存用的變數時，使用這個 clause就不需要進行複製進出的動作

- 可以只複製部分資料
  - #pragma acc parallel loop copy(A[1:N-2])

Data Management

```
#pragma acc data copy(A[0:N])
#pragma acc parallel
{
    #pragma acc loop
    for(int i = 0; i < N; i++) A[i] = 0;
}
```


```
#pragma acc kernels copy(a[0:N])
for(int i = 0; i < N; i++){
  a[i] = 0;
}
```

Loop Optimization

- #pragma acc parallel loop collapse(...)
  - 在tightly nested的迴圈中可以使用
  - collapse 可以攤平迴圈，把多個迴圈變成一個大迴圈平行

```
#pragma acc parallel loop collapse( 2 )
for(int j = 0; j < M; j++) {
        for(int k = 0; k < Q; k++) {
                < loop code >
        }
}
```

TIP1：
當外層的迴圈過小時，攤平迴圈可以增加GPU的使用率

TIP2：
Compiler會傾向於vectorize最內層的迴圈，因此如果最內層的迴圈過小時，collapse就可以增加迴圈深度，至此內層迴圈的vectorize更有效

Loop Optimization

- #pragma acc parallel loop tile(x, y)
  - 把迴圈break給多個tiles(blocks)計算

```
#pragma acc parallel loop tile( 32, 32 )
for(int j = 0; j < 128; j++) {
    for(int k = 0; k < 128; k++) {
        < loop code >
    }
}
```

TIP1:
盡量讓tile大小是32的倍數，Nvidia GPU的一個worker和vector中的threads都是以32為單位執行

TIP2:
不要用超過32*32大小的tiles，因為在NVIDIA GPU中，一個gang的threads數最高是1024（32*32）

Extra Study

OpenACC教學

OpenACC官網

Gang/Worker/Vector