멀티코어 HW6 보고서

2016-11464 정재훈

1. SIMD 벡터화

벡터화 방법 : __mm256형 벡터를 사용하여 루프 순서를 i>k>j로 바꾼 후 벡터 a를 한 원소((i,k) 좌표의 원소)에 대하여 _m256_set1_ps로 세팅, b와 c를 주어진 배열 포인터에서 load해와서 mul, add한 후 다시 c에 store 했다. 256bit 벡터를 사용하기 위해서 Makefile에 -mavx를 추가했다.

결과 :

명시적 이용 - 17.146초

```
Problem size:   2048 x 2048 x 2048

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   17.146 sec (17146.274 ms)
```

자동 벡터화 (simd_ref를 사용) - 18.022초

```
Problem size:   2048 x 2048 x 2048

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   18.022 sec (18022.384 ms)
```

분석 : 조금이나마 명시적 이용을 하는 쪽이 빠르지만, 전역/지역변수 초기화, 알고리즘 변화, 다수의 thread 사용, load/store 횟수의 증가/감소 등으로 결정되는 시간에 비해서는 작은 차이(0.9초)로 컴파일러가 자동 벡터화를 거의 명시적으로 이용하듯 잘 시킴을 알 수 있었다.

2. CUDA 병렬화

병렬화 방법에 대한 설명 : setup에서 device를 세팅하고 cudaMalloc으로 전역변수 a1, b1, c1에 공간을 할당한다. host 함수에서는 cudaMemcpy를 사용하여 a와 b를 a1과 b1으로 복사하고 device 함수의 실행이 끝난 후 결과가 저장된 c1을 c로 복사해온다. 이 때 block 크기와 grid의 크기를 세팅한다. (block 크기는 16*16으로 세팅했다) device 함수에서는 thread들마다 한 줄(a1 한줄 * b1 한줄)씩 행렬 곱셈을 해서 c1에 채워 넣는다.

결과 :

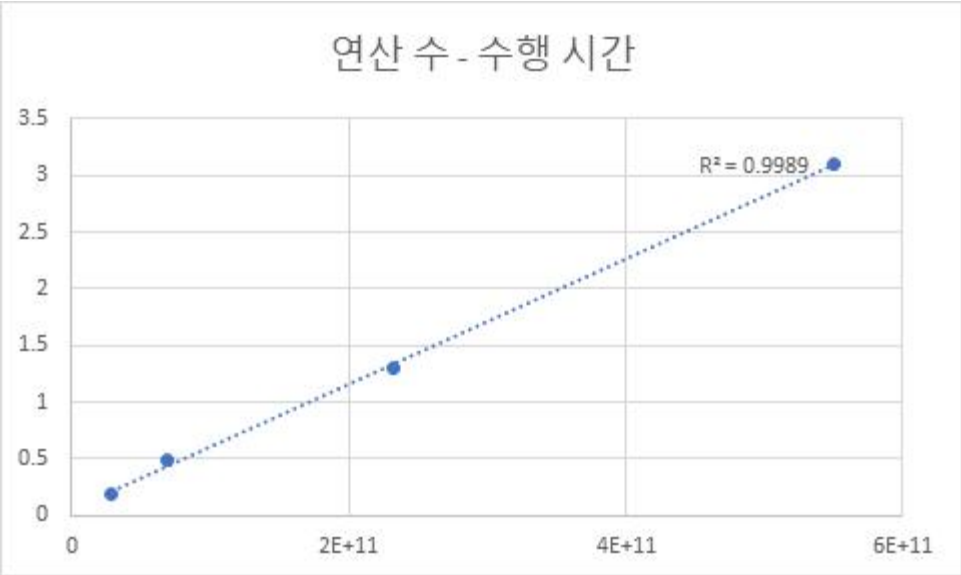
```
Problem size:   3072 x 3072 x 3072

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   0.194 sec (194.085 ms)
```

```
Problem size:   4096 x 4096 x 4096

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   0.495 sec (494.930 ms)
```

```
Problem size:   6144 x 6144 x 6144

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   1.299 sec (1298.514 ms)
```

```
Problem size:   8192 x 8192 x 8192

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   3.106 sec (3106.496 ms)
```

분석 :



연산 수 대비 수행 시간 그래프를 그려본 결과 거의 일직선이므로 CUDA 역시 각 thread에 대하여 균등하게 일을 수행함을 알 수 있었다. 추가로 cuda free/memcpy 연산의 overhead가 작다는 것을 알 수 있었다.

OpenCL과 비교하여 CUDA를 사용한 소감 : kernel.cl로 옮겨 다니면서 디버깅하지 않아도 돼서 편했다. OpenCL보다 간단하게 코드를 짤 수 있으나 다양한 최적화(tiling 등)를 하기 위해서는 OpenCL보다 생각해야 할 점이 많다고 생각했다.