CUDA 프로그래밍

CUDA Programming

biztripcru@gmail.com

Giga-size Vector Addition

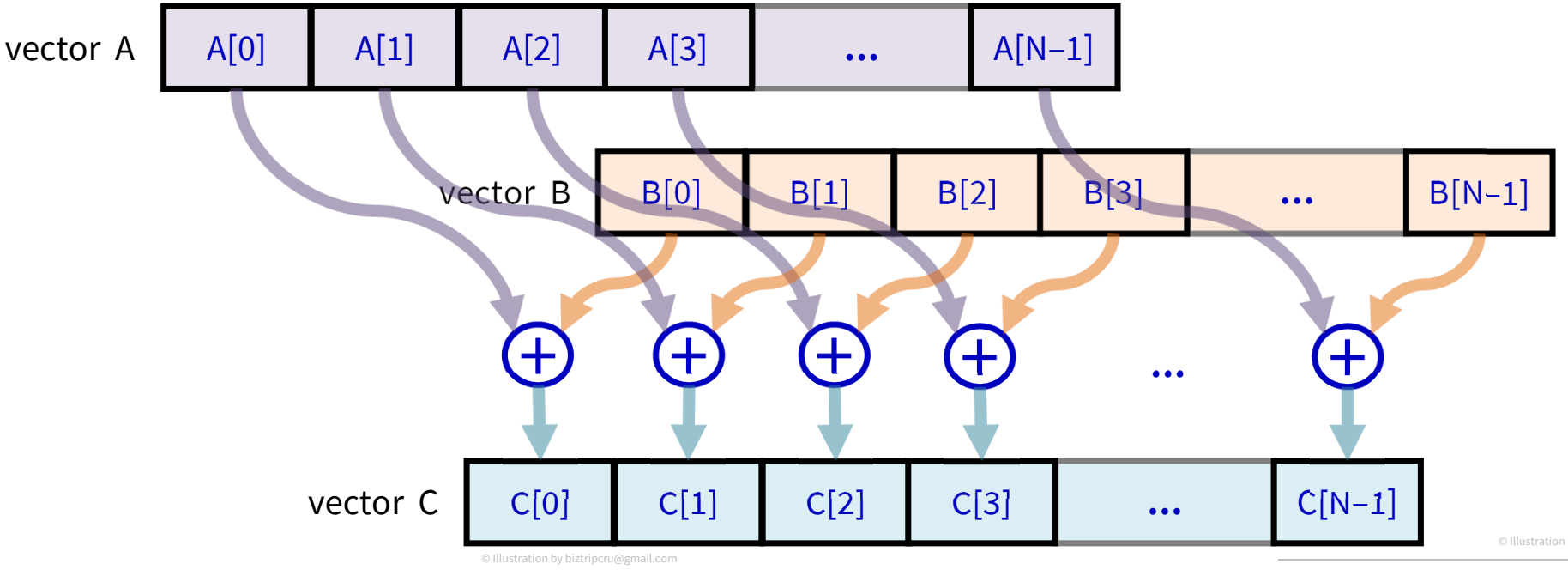
대규모 벡터 더하기

내용 contents

- vector addition with 256M elements
- host version – CPU 사용
- CUDA version – core 1개 사용
- CUDA version – 최대로 가속
- CUDA version – clock() 함수로 시간 측정
- CUDA version – argument 처리 (512M elements)

Vector Addition, Again

- **vector addition, with *n* elements:** **for a large *n***
  - ***n* = 256M to 1G**



giga-add-host.cpp

```
#include "./common.cpp"

const unsigned SIZE = 256 * 1024 * 1024;   // big-size elements

int main(void) {
   …
   // kernel: vector addition
   ELAPSED_TIME_BEGIN(0);
   for (register unsigned i = 0; i < SIZE; ++i) {
      vecC[i] = vecA[i] + vecB[i];
   }
   ELAPSED_TIME_END(0);
   …
}
```

giga-add-host.cpp 실행 결과

- **459,271 usec** for **256M** elements vector addition (Intel Core i5-3570)

```
linux/cuda-work > ./13a-giga-add-host.exe
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 459271 usec
SIZE = 268435456
sumA = 134076296.000000
sumB = 134079704.000000
sumC = 268156016.000000
diff(sumC, sumA+sumB) =  16.000000
diff(sumC, sumA+sumB) / SIZE =  0.000000
vecA=[ 0.38300  0.88600  0.77700  0.91500 ...  0.79900  0.17900  0.51000  0.83300]
vecB=[ 0.06600  0.74400  0.27000  0.44600 ...  0.07400  0.81700  0.77500  0.85100]
vecC=[ 0.44900  1.63000  1.04700  1.36100 ...  0.87300  0.99600  1.28500  1.68400]
linux/cuda-work >
linux/cuda-work >
```

giga-add-single.cu

```
// CUDA kernel function
__global__ void singleKernelVecAdd( float* c, const float* a, const float* b ) {
   for (register unsigned i = 0; i < SIZE; ++i) {
      c[i] = a[i] + b[i];
   }
}

int main(void) {
   …
   // CUDA kernel call: single thread !
   ELAPSED_TIME_BEGIN(0);
   singleKernelVecAdd <<< 1, 1>>>( dev_vecC, dev_vecA, dev_vecB );
   cudaDeviceSynchronize();
   ELAPSED_TIME_END(0);
   CUDA_CHECK_ERROR();
   …
}
```

giga-add-single.cu 실행 결과

- **11,913,504 usec for 256M elements vector addition** (GeForce RTX 2070)
  - 12,651,351 usec, including “memcpy” between host and device

```
linux/cuda-work > ./13b-giga-add-single.exe
elapsed wall-clock time[1] started
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 11913504 usec
elapsed wall-clock time[1] = 12651351 usec
SIZE = 268435456
sumA = 134076296.000000
sumB = 134079704.000000
sumC = 268156016.000000
diff(sumC, sumA+sumB) =  16.000000
diff(sumC, sumA+sumB) / SIZE =  0.000000
vecA=[ 0.38300  0.88600  0.77700  0.91500 ...  0.79900  0.17900  0.51000  0.83300]
vecB=[ 0.06600  0.74400  0.27000  0.44600 ...  0.07400  0.81700  0.77500  0.85100]
vecC=[ 0.44900  1.63000  1.04700  1.36100 ...  0.87300  0.99600  1.28500  1.68400]
linux/cuda-work >
```

giga-add-dev.cu

- **kernel function**

```
// CUDA kernel function
__global__ void kernelVecAdd( float* c, const float* a, const float* b, unsigned n ) {
   unsigned i = blockIdx.x * blockDim.x + threadIdx.x; // CUDA-provided index
   if (i < n) {
      c[i] = a[i] + b[i];
   }
}
```

- **boundary check: if (i < n) { … }**
  - blockDim 과 vector size 가 맞아떨어진다는 보장이 없음
  - 예: vector size = 999, block은 32의 배수로 실행됨

giga-add-dev.cu

- **dimGrid, dimBlock 의 계산**
  - 전체 thread 개수 = dimBlock.x * dimGrid.x **≥ SIZE 를 보장**해야
  - dimGrid.x = ceil(SIZE / dimBlock.x) = **⌈ SIZE / dimBlock.x ⌉** (올림)

```
int main(void) {
  ...
  // CUDA kernel call
  ELAPSED_TIME_BEGIN(0);
  dim3 dimBlock( 1024, 1, 1 );
  dim3 dimGrid( (SIZE + dimBlock.x - 1) / dimBlock.x, 1, 1 );
  kernelVecAdd <<< dimGrid, dimBlock >>> ( dev_vecC, dev_vecA, dev_vecB, SIZE );
  cudaDeviceSynchronize();
  ELAPSED_TIME_END(0);
  CUDA_CHECK_ERROR();
  ...
}
```

giga-add-dev.cu 실행 결과

- **8,136 usec for 256M elements vector addition**
  - 746,448 usec, including “memcpy” between host and device

```
linux/cuda-work > ./13c-giga-add-dev.exe
elapsed wall-clock time[1] started
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 8136 usec
elapsed wall-clock time[1] = 746448 usec
SIZE = 268435456
sumA = 134076296.000000
sumB = 134079704.000000
sumC = 268156016.000000
diff(sumC, sumA+sumB) =  16.000000
diff(sumC, sumA+sumB) / SIZE =  0.000000
vecA=[ 0.38300  0.88600  0.77700  0.91500 ...  0.79900  0.17900  0.51000  0.83300]
vecB=[ 0.06600  0.74400  0.27000  0.44600 ...  0.07400  0.81700  0.77500  0.85100]
vecC=[ 0.44900  1.63000  1.04700  1.36100 ...  0.87300  0.99600  1.28500  1.68400]
linux/cuda-work >
```

clock() in the Kernel Function

- clock_t **clock(** void **);**
- long long int **clock64(** void **);**
  - returns the value of a per-multiprocessor counter (or clock ticks)
  - CAUTION: executed in __device__ and __global__ functions

```
__global__ void kernelVecAdd( float* c, const float* a, const float* b, unsigned n, long long* times ) {
  clock_t start = clock();
  unsigned i = blockIdx.x * blockDim.x + threadIdx.x; // CUDA-provided index
  … (do something) …
  clock_t end = clock();
  times[i] = (long long)(end – start);
}
```

cudaDeviceGetAttribute( )

- cudaError_t **cudaDeviceGetAttribute**(int* *value*, cudaDeviceAttr *attr*, int *device*)
  - returns the device attribute value into "*value*"
  - *device*: device number to query (in most cases, 0)
  - *attr* = cudaDevAttrClockRate to get the clock frequency in kHz
    - elapsed time (usec) = (# clock ticks) * 1000 / (clock freq in kHz)

```
// kernel clock calculation
int clk_freq = 1;
cudaDeviceGetAttribute( &clk_freq, cudaDevAttrClockRate, 0 );
float elapsed_usec = clk_ticks * 1000.0F / clk_freq;
```

elapsed_usec 계산 과정

- **주어진 값들**
  - clk_ticks = num of clock ticks (scalar, 단위 없음)
  - clk_freq = clock frequency (kHz)
- **문제: clock tick 횟수 (clk_ticks)로 부터, 소요 시간 elapsed time 을 계산?**
  - elapsed time (usec) = (num of clock ticks) × (1 clk tick 소요 시간)
  - $$(1 \text{ clk tick time}) = \frac{1}{(\text{clk freq in Hz})} \text{ sec} = \frac{1}{(\text{clk freq in kHz}) \times \frac{1,000 \text{ Hz}}{1 \text{ kHz}}} \text{ sec} \times \frac{1,000,000 \text{ usec}}{1 \text{ sec}}$$
    $$= \frac{1,000}{(\text{clk freq in kHz})} \text{ usec}$$
  - elapsed time (usec) = (num of clock ticks) × 1,000 / (clk freq in kHz)
- **결론:** float elapsed_usec = clk_ticks * 1000.0F / clk_freq;

giga-add-clock.cu

```
// CUDA kernel function
__global__ void kernelVecAdd( float* c, const float* a, const float* b, unsigned n, long long* times ) {
   clock_t start = clock();
   unsigned i = blockIdx.x * blockDim.x + threadIdx.x; // CUDA-provided index
   if (i < n) {
      c[i] = a[i] + b[i];
   }
   clock_t end = clock();
   if (i == 0) {
      times[0] = (long long)(end - start);
   }
}
```

giga-add-clock.cu 계속

```
int main(void) {
  …
  long long* dev_times = nullptr;
  cudaMalloc( (void**)&dev_times, 1 * sizeof(long long) );
  …
  // CUDA kernel call
  ELAPSED_TIME_BEGIN(0);
  dim3 dimBlock( 1024, 1, 1 );
  dim3 dimGrid( (SIZE + dimBlock.x - 1) / dimBlock.x, 1, 1 );
  kernelVecAdd <<< dimGrid, dimBlock >>> ( dev_vecC, dev_vecA, dev_vecB, SIZE, dev_times );
  cudaDeviceSynchronize();
  ELAPSED_TIME_END(0);
  CUDA_CHECK_ERROR();
  …
```

giga-add-clock.cu 계속

```
    // kernel clock calculation
    int peak_clk = 1;
    cudaDeviceGetAttribute(&peak_clk, cudaDevAttrClockRate, 0);
    printf("num clock = %lld, peak clock rate = %dkHz, elapsed time: %f usec\n",
        host_times[0], peak_clk, host_times[0] * 1000.0f / (float)peak_clk);
    …
}
```

giga-add-clock.cu 실행 결과

- **8,047 usec for 256M elements vector addition** (GeForce RTX 2070)
  - 747,203 usec, including "memcpy" between host and device
- **for a single thread, 1.075 usec (or 1743 clock ticks) (1.6GHz GPU)**

```
linux/cuda-work > ./13d-giga-add-clock.exe
elapsed wall-clock time[1] started
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 8047 usec
elapsed wall-clock time[1] = 747203 usec
num clock = 1743, peak clock rate = 1620000kHz, elapsed time: 1.075926 usec
SIZE = 268435456
sumA = 134076296.000000
sumB = 134079704.000000
sumC = 268156016.000000
diff(sumC, sumA+sumB) =  16.000000
diff(sumC, sumA+sumB) / SIZE =  0.000000
vecA=[ 0.38300  0.88600  0.77700  0.91500 ...  0.79900  0.17900  0.51000  0.83300]
vecB=[ 0.06600  0.74400  0.27000  0.44600 ...  0.07400  0.81700  0.77500  0.85100]
vecC=[ 0.44900  1.63000  1.04700  1.36100 ...  0.87300  0.99600  1.28500  1.68400]
linux/cuda-work >
```

giga-add-arg.cu

```
unsigned vecSize = 256 * 1024 * 1024; // 256M elements

int main( const int argc, const char* argv[] ) {
  // argv processing
  char* pEnd = nullptr;
  switch (argc) {
  case 1:
    break;
  case 2:
    vecSize = strtol( argv[1], &pEnd, 10 );
    break;
  default:
    printf("usage: %s [size]\n", argv[0]);
    exit( EXIT_FAILURE );
    break;
  }
  …
```

- **argument 의 적용**
  - vecSize 가 variable (변경 가능)
  - strtol : string to long

- **$ ./giga-add-arg.exe 512000000**
  - 이제, 512,000,000 element 처리
  - 512 * 1024 * 1024 와는 다름

procArg(...) in "./common.cpp"

```
template <typename TYPE>
TYPE procArg( const char* progname, const char* str,
                        TYPE lbound = -1, TYPE ubound = -1) {
  char* pEnd = nullptr;
  TYPE value = 0;
  if (typeid(TYPE) == typeid(float) ||
          typeid(TYPE) == typeid(double)) {
    value = strtof( str, &pEnd );
  } else {
    value = strtol( str, &pEnd, 10 );
  }
  . . .
      case 'M':
        value *= (1024 * 1024);
  . . .
}
```

- **좀더 다양한 처리 추가**
  - int, float 모두 가능
  - bound 체크 추가
    - 범위를 벗어나면 error
- **K, M 의 의미를 추가**
  - 16K = 16 * 1024
  - 32M = 32 * 1024 * 1024

giga-add-arg.cu 실행 결과

- **8,140 usec for 256M elements vector addition**

```
linux/cuda-work > ./13e-giga-add-arg.exe
elapsed wall-clock time[1] started
elapsed wall-clock time[0] started
vecSize = 268435456, dimBlock = 1024, dimGrid = 262144, total # thread = 268435456
elapsed wall-clock time[0] = 8140 usec
elapsed wall-clock time[1] = 744026 usec
num clock = 2087, peak clock rate = 1620000kHz, elapsed time: 1.288272 usec
SIZE = 268435456
sumA = 134076296.000000
sumB = 134079704.000000
sumC = 268156016.000000
diff(sumC, sumA+sumB) =  16.000000
diff(sumC, sumA+sumB) / SIZE =  0.000000
vecA=[ 0.38300  0.88600  0.77700  0.91500 ...  0.79900  0.17900  0.51000  0.83300]
vecB=[ 0.06600  0.74400  0.27000  0.44600 ...  0.07400  0.81700  0.77500  0.85100]
vecC=[ 0.44900  1.63000  1.04700  1.36100 ...  0.87300  0.99600  1.28500  1.68400]
linux/cuda-work >
```

giga-add-arg.cu 실행 결과

- **15,422 usec** for 512,000,000 elements vector addition

```
linux/cuda-work > ./13e-giga-add-arg.exe 512000000
elapsed wall-clock time[1] started
elapsed wall-clock time[0] started
vecSize = 512000000, dimBlock = 1024, dimGrid = 500000, total # thread = 512000000
elapsed wall-clock time[0] = 15422 usec
elapsed wall-clock time[1] = 1427251 usec
num clock = 2718, peak clock rate = 1620000kHz, elapsed time: 1.677778 usec
SIZE = 512000000
sumA = 255732944.000000
sumB = 255756848.000000
sumC = 511490336.000000
diff(sumC, sumA+sumB) =  544.000000
diff(sumC, sumA+sumB) / SIZE =  0.000001
vecA=[ 0.38300  0.88600  0.77700  0.91500 ...  0.17100  0.82300  0.93100  0.79900]
vecB=[ 0.98900  0.44600  0.20800  0.17500 ...  0.59700  0.12100  0.31700  0.92000]
vecC=[ 1.37200  1.33200  0.98500  1.09000 ...  0.76800  0.94400  1.24800  1.71900]
linux/cuda-work >
```

- "out of memory" for 800,000,000 elements

```
linux/cuda-work > ./13e-giga-add-arg.exe 800000000
cuda failure "out of memory" at 13e-giga-add-arg.cu:52
linux/cuda-work >
```

내용 contents

- vector addition with 256M elements (GeForce RTX 2070)
- host version – CPU 사용 **459,271 usec**
- CUDA version – core 1개 사용 **11,913,504 usec**
- CUDA version – 최대로 가속 **8,136 usec**
- CUDA version – clock() 함수로 시간 측정 1743 clock ticks
- CUDA version – argument 처리 (512M elements) **15,422 usec**

Giga-size Vector Addition

대규모 벡터 더하기

폰트 끝단 일치 → 큰 교자 타고 혼례 치른 날
정참판 양반댁 규수 큰 교자 타고 혼례 치른 날
정참판 양반댁 규수 큰 교자 타고 혼례 치른 날
본고딕 Noto Sans KR

The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
Source Sans Pro

Mathematical Notations $O(n \log n)$
Source Serif Pro