CUDA 프로그래밍

CUDA Programming

biztripcru@gmail.com

Vector Addition

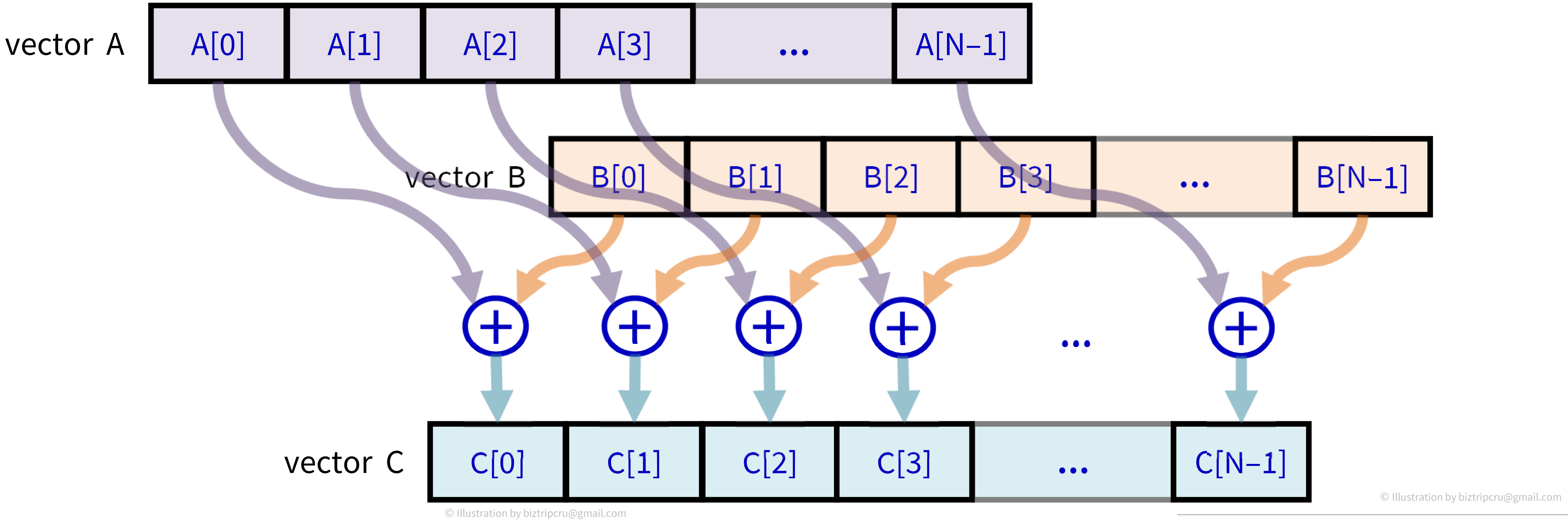
벡터 더하기

내용 contents

- vector addition 설명
  - vector = 1D array
- host version – CPU 사용
- CUDA version – core 1개 사용
- CUDA version – 최대로 가속
- CUDA version – C++ flavor

Vector Addition

- vector : represented as **1D array**, with *n* elements
  - C[i] = A[i] + B[i]



Vector Addition 계속

- **vector : represented as 1D array**
  - const float a[SIZE];
  - const float b[SIZE];
  - float c[SIZE];

- **vector addition: c[...] = a[...] + b[...]**
  - 1D array + 1D array → 1D array

- **for a big size**
  - SIZE = 1024 * 1024 → **1 million** additions

random data 생성

- **big-size data 생성?**
  - 난수 random data 사용 !

```
#include <stdlib.h>
int rand( void );
```

- [0 ~ RAND_MAX] 사이의 난수 random number 생성

```
void srand( unsigned int seed );
```

- seed 에 따라서, 다른 난수가 생성됨

- 응용: [0.000, 1.000) 의 float 형식 난수 생성
  - `num = (rand() % 1000) / 1000.0F;`

vecadd-host.cpp

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

// set random value of [0.000, 1.000) to dst array
void setRandomData( float* dst, int size ) {
   while (size--) {
      *dst++ = (rand() % 1000) / 1000.0F;
   }
}

// get total sum of dst array
float getSum( float* dst, int size ) {
   register float sum = 0.0F;
   while (size--) {
      sum += *dst++;
   }
   return sum;
}
```

vecadd-host.cpp 계속

```
const unsigned SIZE = 1024 * 1024; // 1M elements

int main( void ) {
   // host-side data
   float* vecA = new float[SIZE];
   float* vecB = new float[SIZE];
   float* vecC = new float[SIZE];
   // set random data to A and B
   srand( 0 );
   setRandomData( vecA, SIZE );
   setRandomData( vecB, SIZE );
   // kernel: vector addition
   chrono::system_clock::time_point time_begin = chrono::system_clock::now();
   for (register unsigned i = 0; i < SIZE; ++i) {
      vecC[i] = vecA[i] + vecB[i];
   }
   chrono::system_clock::time_point time_end = chrono::system_clock::now();
```

vecadd-host.cpp 계속

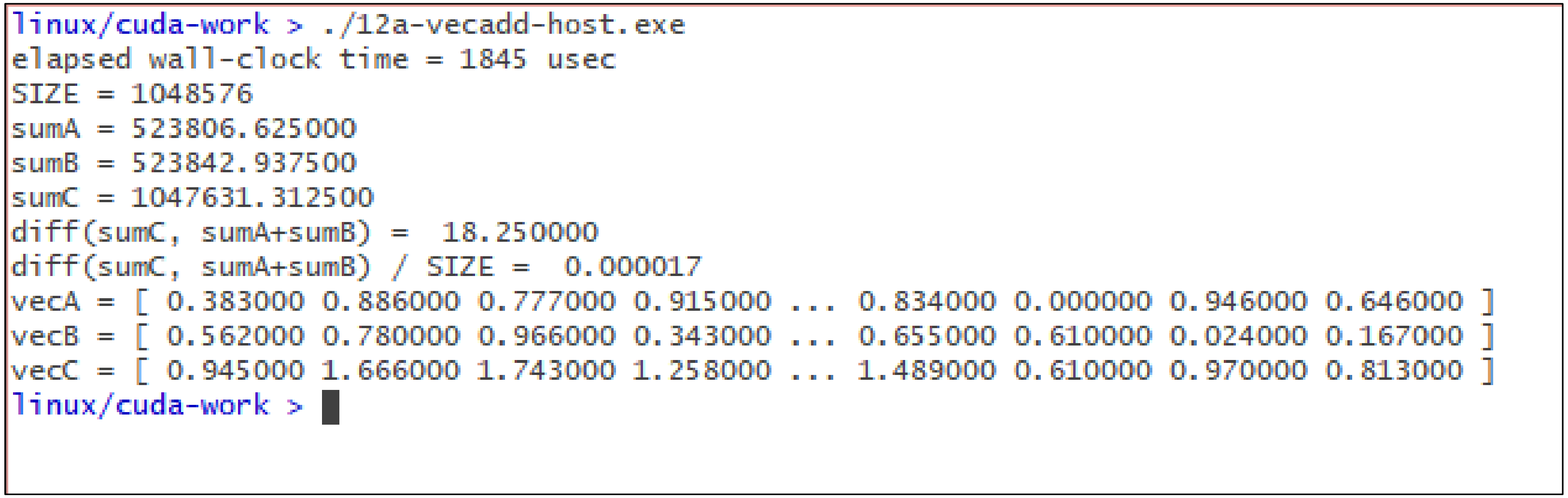
```
chrono::microseconds time_elapsed_msec
        = chrono::duration_cast<chrono::microseconds>(time_end - time_begin);
printf("elapsed wall-clock time = %ld usec\n", (long)time_elapsed_msec.count());
// check the result
float sumA = getSum( vecA, SIZE );
float sumB = getSum( vecB, SIZE );
float sumC = getSum( vecC, SIZE );
float diff = fabsf( sumC - (sumA + sumB) );
printf("SIZE = %d\n", SIZE);
printf("sumA = %f\n", sumA);
printf("sumB = %f\n", sumB);
printf("sumC = %f\n", sumC);
printf("diff(sumC, sumA+sumB) =  %f\n", diff);
printf("diff(sumC, sumA+sumB) / SIZE =  %f\n", diff / SIZE);
```

vecadd-host.cpp 계속

```
    printf("vecA = [ %8f %8f %8f %8f ... %8f %8f %8f %8f ]\n",
        vecA[0], vecA[1], vecA[2], vecA[3],
        vecA[SIZE - 4], vecA[SIZE - 3], vecA[SIZE - 2], vecA[SIZE - 1]);
    printf("vecB = [ %8f %8f %8f %8f ... %8f %8f %8f %8f ]\n",
        vecB[0], vecB[1], vecB[2], vecB[3],
        vecB[SIZE - 4], vecB[SIZE - 3], vecB[SIZE - 2], vecB[SIZE - 1]);
    printf("vecC = [ %8f %8f %8f %8f ... %8f %8f %8f %8f ]\n",
        vecC[0], vecC[1], vecC[2], vecC[3],
        vecC[SIZE - 4], vecC[SIZE - 3], vecC[SIZE - 2], vecC[SIZE - 1]);
    // cleaning
    delete[] vecA;
    delete[] vecB;
    delete[] vecC;
    // done
    return 0;
}
```

vecadd-host.cpp – results

- 실행 결과: 1,845 usec



```
linux/cuda-work > ./12a-vecadd-host.exe
elapsed wall-clock time = 1845 usec
SIZE = 1048576
sumA = 523806.625000
sumB = 523842.937500
sumC = 1047631.312500
diff(sumC, sumA+sumB) =  18.250000
diff(sumC, sumA+sumB) / SIZE =  0.000017
vecA = [ 0.383000 0.886000 0.777000 0.915000 ... 0.834000 0.000000 0.946000 0.646000 ]
vecB = [ 0.562000 0.780000 0.966000 0.343000 ... 0.655000 0.610000 0.024000 0.167000 ]
vecC = [ 0.945000 1.666000 1.743000 1.258000 ... 1.489000 0.610000 0.970000 0.813000 ]
linux/cuda-work > 
```

- 개선책: “common.cpp” 에 추가
  - void **setNormalizedRandomData**(float* dst, int num);
  - float **getSum**(float* dst, int num);
  - void **printVec**(const char* name, float* dst, int num);

setNormalizedRandomData( ) in “common.cpp”

- void **setNormalizedRandomData**(float* dst, int num);

```
template <typename TYPE>
void setNormalizedRandomData( TYPE* pDst, long long num,
                                      TYPE bound=static_cast<TYPE>(1000)) {
  int32_t bnd = static_cast<int32_t>(bound);
  while (num--) {
    *pDst++ = (rand() % bnd) / static_cast<TYPE>(bnd);
  }
}
```

getSum( ) in “common.cpp”

- float **getSum(** float* *dst*, int *num* **);**
  - *dst* 배열의 0 ~ (*num* – 1) 번째 원소의 합을 구함

```
float getSum( float* dst, int size ) {
   register float sum = 0.0F;
   while (size--) {
      sum += *dst++;
   }
   return sum;
}
```

- **실제 구현:**
  - float 타입의 정밀도 precision 문제로,
  - 몇 개씩 묶어서 **partial sum**을 구한 후에, 다시 합쳐서 total sum을 구함

getSum( ) in “common.cpp” 계속

```
template <typename TYPE>
TYPE getSum( const TYPE* pSrc, int num ) {
   register TYPE sum = static_cast<TYPE>(0);
   // add 128K elements in a chunk
   const int chunk = 128 * 1024;
   while (num > chunk) {
      register TYPE partial = static_cast<TYPE>(0);
      register int n = chunk;
      while (n--) {
         partial += *pSrc++;
      }
      sum += partial;
      num -= chunk;
   }
```

```
   // add remaining elements
   register TYPE partial = static_cast<TYPE>(0);
   while (num--) {
      partial += *pSrc++;
   }
   sum += partial;
   return sum;
}
```

printVec( ) in “common.cpp”

```cpp
template <typename TYPE>
void printVec( const char* name, const TYPE* vec, int num ) {
   std::streamsize ss = std::cout.precision();
   std::cout.precision(5);
   std::cout << name << "=[";
   std::cout << fixed << showpoint << std::setw(8) << vec[0] << " ";
   std::cout << fixed << showpoint << std::setw(8) << vec[1] << " ";
   std::cout << fixed << showpoint << std::setw(8) << vec[2] << " ";
   std::cout << fixed << showpoint << std::setw(8) << vec[3] << " ... ";
   std::cout << fixed << showpoint << std::setw(8) << vec[num - 4] << " ";
   std::cout << fixed << showpoint << std::setw(8) << vec[num - 3] << " ";
   std::cout << fixed << showpoint << std::setw(8) << vec[num - 2] << " ";
   std::cout << fixed << showpoint << std::setw(8) << vec[num - 1] << "]" << std::endl;
   std::cout.precision(ss);
}
```

vecadd-host-kernel.cpp

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

const unsigned SIZE = 1024 * 1024; // 1M elements

// kernel function: body of the FOR loop
void kernelVecAdd( unsigned i, float* c, const float* a, const float* b ) {
   c[i] = a[i] + b[i];
}

int main(void) {
   // host-side data
   float* vecA = new float[SIZE];
   float* vecB = new float[SIZE];
   float* vecC = new float[SIZE];
```

vecadd-host-kernel.cpp 계속

```
// set random data to A and B
srand( 0 );
setNormalizedRandomData( vecA, SIZE );
setNormalizedRandomData( vecB, SIZE );
// kernel: vector addition
ELAPSED_TIME_BEGIN(0);
for (register unsigned i = 0; i < SIZE; ++i) {
    kernelVecAdd( i, vecC, vecA, vecB );
}
ELAPSED_TIME_END(0);
// check the result
float sumA = getSum( vecA, SIZE );
float sumB = getSum( vecB, SIZE );
float sumC = getSum( vecC, SIZE );
float diff = fabsf( sumC - (sumA + sumB) );
```

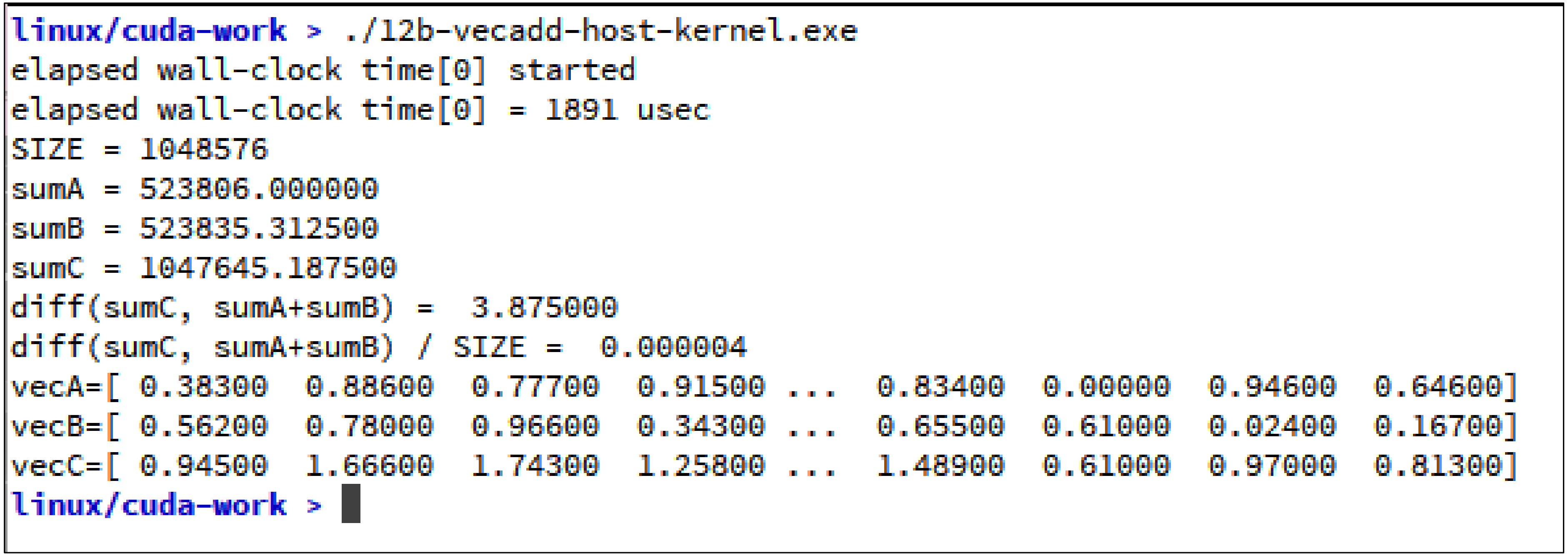
vecadd-host-kernel.cpp 계속

```
    printf("SIZE = %d\n", SIZE);
    printf("sumA = %f\n", sumA);
    printf("sumB = %f\n", sumB);
    printf("sumC = %f\n", sumC);
    printf("diff(sumC, sumA+sumB) =  %f\n", diff);
    printf("diff(sumC, sumA+sumB) / SIZE =  %f\n", diff / SIZE);
    printVec( "vecA", vecA, SIZE );
    printVec( "vecB", vecB, SIZE );
    printVec( "vecC", vecC, SIZE );
    …
    // done
    return 0;
}
```

vecadd-host-kernel.cpp – results

- 실행 결과: **1,891 usec** (Intel Core i5-3570)

CPU	1,891 usec



```
linux/cuda-work > ./12b-vecadd-host-kernel.exe
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 1891 usec
SIZE = 1048576
sumA = 523806.000000
sumB = 523835.312500
sumC = 1047645.187500
diff(sumC, sumA+sumB) =  3.875000
diff(sumC, sumA+sumB) / SIZE =  0.000004
vecA=[ 0.38300  0.88600  0.77700  0.91500 ...  0.83400  0.00000  0.94600  0.64600]
vecB=[ 0.56200  0.78000  0.96600  0.34300 ...  0.65500  0.61000  0.02400  0.16700]
vecC=[ 0.94500  1.66600  1.74300  1.25800 ...  1.48900  0.61000  0.97000  0.81300]
linux/cuda-work > 
```

vecadd-single.cu

- **CUDA kernel function**

```
// 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];
   }
}
```

- **CUDA kernel launce – single core 로 1개의 core만 사용**

```
   // CUDA kernel call
   ELAPSED_TIME_BEGIN(0);
   singleKernelVecAdd <<< 1, 1>>>( dev_vecC, dev_vecA, dev_vecB );
   cudaDeviceSynchronize();
   ELAPSED_TIME_END(0);
```

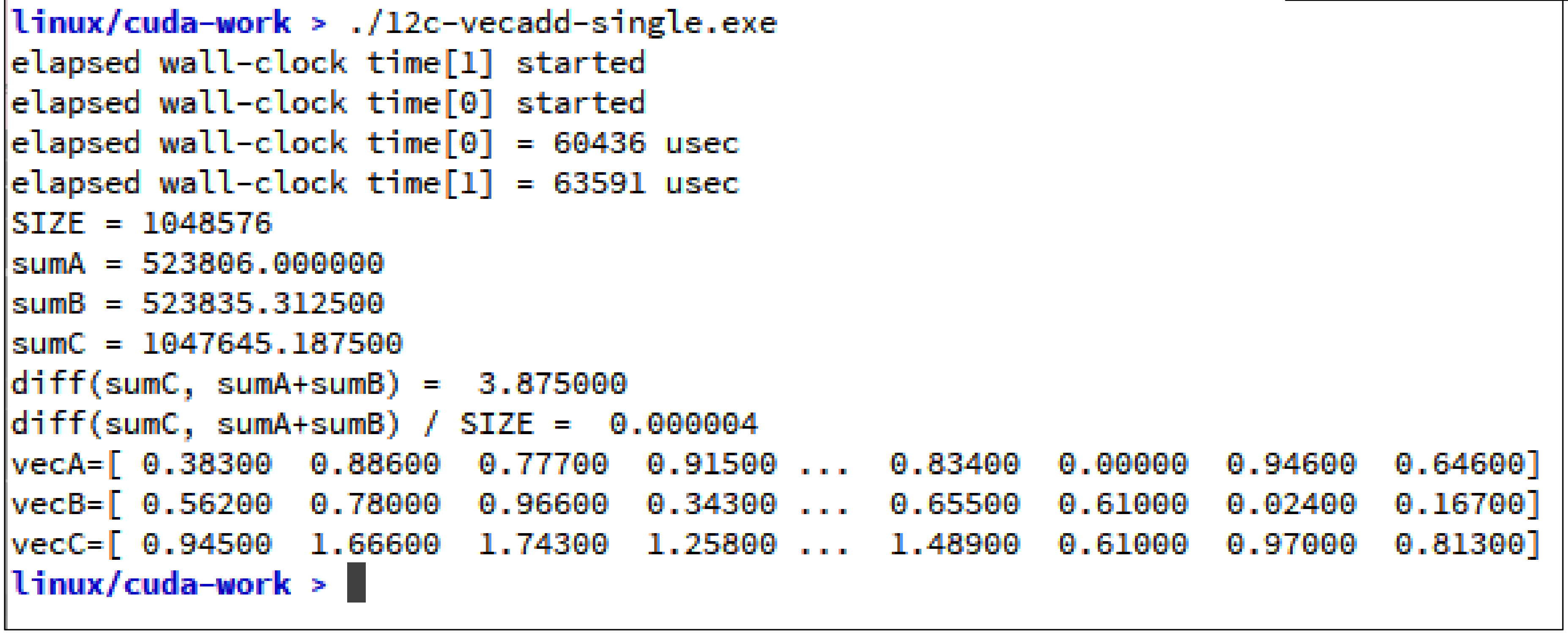
vecadd-single.cu

```
…
int main(void) {
   …
   // copy to device from host
   ELAPSED_TIME_BEGIN(1);
   cudaMemcpy( dev_vecA, vecA, SIZE * sizeof(float), cudaMemcpyHostToDevice );
   …
   // CUDA kernel call
   ELAPSED_TIME_BEGIN(0);
   singleKernelVecAdd <<< 1, 1>>>( dev_vecC, dev_vecA, dev_vecB );
   cudaDeviceSynchronize();
   ELAPSED_TIME_END(0);
   …
   // copy to host from device
   cudaMemcpy( vecC, dev_vecC, SIZE * sizeof(float), cudaMemcpyDeviceToHost );
   ELAPSED_TIME_END(1);
   …
}
```

vecadd-single.cu – results

- 실행 결과: **60,436 usec** (GeForce RTX 2070)

CPU	1,891 usec
CUDA, core 1개	60,436 usec



```
linux/cuda-work > ./12c-vecadd-single.exe
elapsed wall-clock time[1] started
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 60436 usec
elapsed wall-clock time[1] = 63591 usec
SIZE = 1048576
sumA = 523806.000000
sumB = 523835.312500
sumC = 1047645.187500
diff(sumC, sumA+sumB) =  3.875000
diff(sumC, sumA+sumB) / SIZE =  0.000004
vecA=[ 0.38300  0.88600  0.77700  0.91500 ...  0.83400  0.00000  0.94600  0.64600]
vecB=[ 0.56200  0.78000  0.96600  0.34300 ...  0.65500  0.61000  0.02400  0.16700]
vecC=[ 0.94500  1.66600  1.74300  1.25800 ...  1.48900  0.61000  0.97000  0.81300]
linux/cuda-work > 
```

vecadd-error.cu

- **CUDA kernel function**

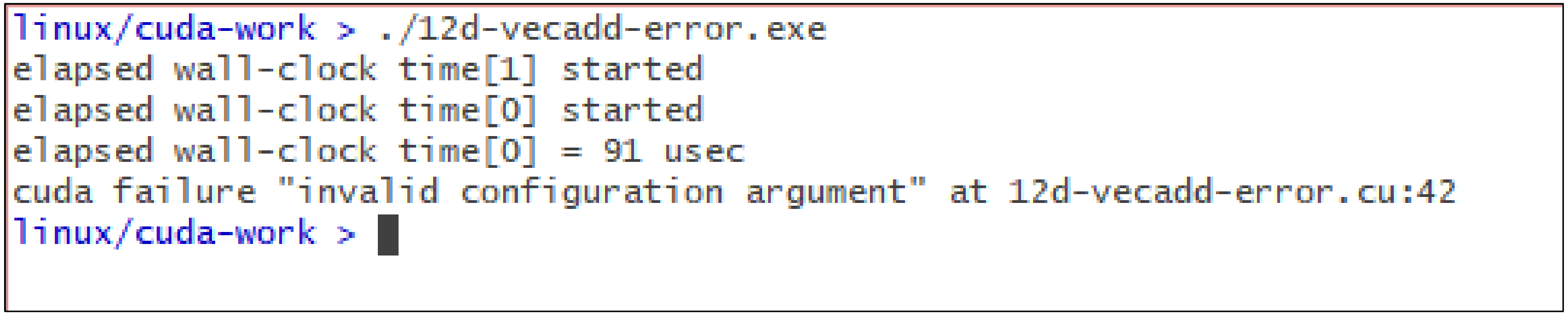
```
// CUDA kernel function
__global__ void kernelVecAdd( float* c, const float* a, const float* b, unsigned n ) {
   unsigned i = threadIdx.x; // CUDA-provided index
   if (i < n) {
      c[i] = a[i] + b[i];
   }
}
```

- **CUDA kernel launce – SIZE = 1 million 개의 core를 사용 요구**

```
   // CUDA kernel call
   ELAPSED_TIME_BEGIN(0);
   kernelVecAdd <<< 1, SIZE>>>( dev_vecC, dev_vecA, dev_vecB );
   cudaDeviceSynchronize();
   ELAPSED_TIME_END(0);
```

vecadd-error.cu – results

- **실행 결과: 실패... invalid configuration argument**



```
linux/cuda-work > ./12d-vecadd-error.exe
elapsed wall-clock time[1] started
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 91 usec
cuda failure "invalid configuration argument" at 12d-vecadd-error.cu:42
linux/cuda-work > 
```

- **실패 원인?**

kernelVecAdd <<< 1, SIZE>>>(dev_vecC, dev_vecA, dev_vecB);

  - SIZE = 1M 개의 thread를 동시 실행 요구
  - SM streaming multi-processor 에서 **1M 개의 thread를 동시 실행 불가능**
  - 실제로는 1024 개가 한계

vecadd-dev.cu

- **CUDA kernel launce – SIZE 개의 core를 사용 요구**

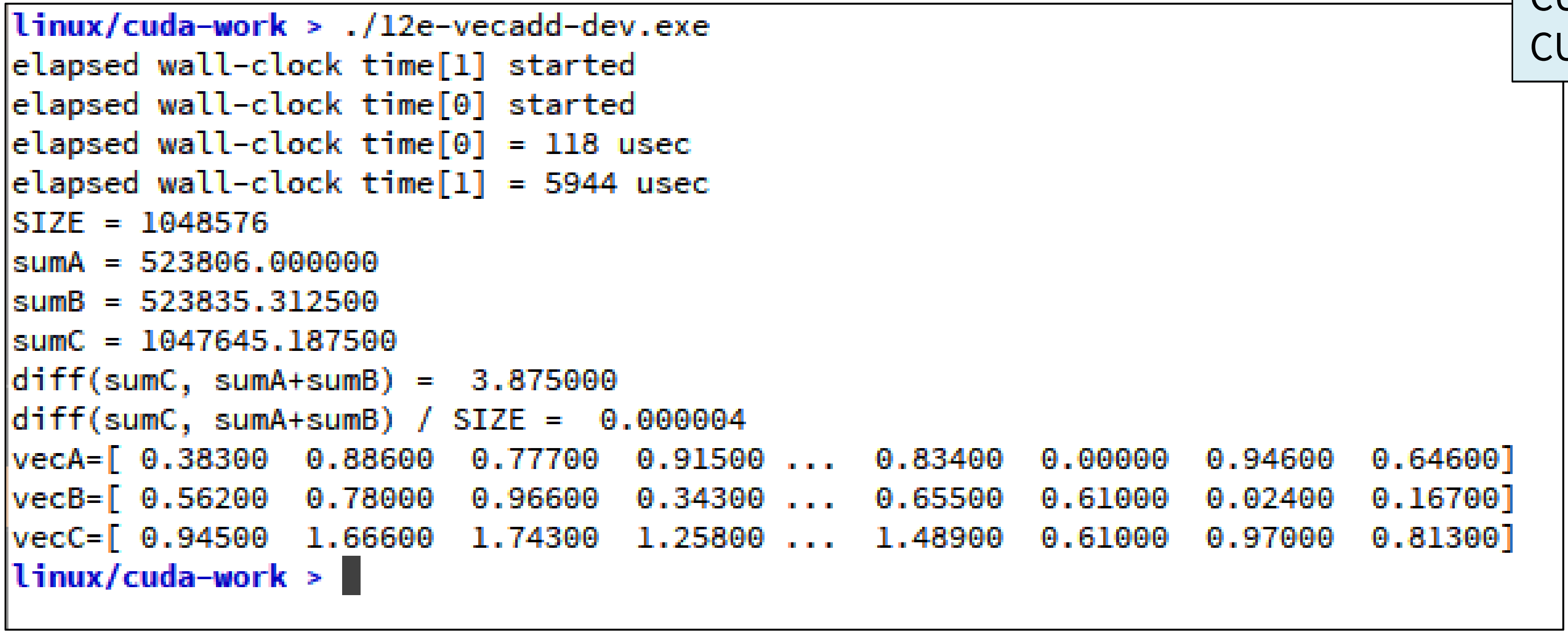
```
// CUDA kernel call
ELAPSED_TIME_BEGIN(0);
kernelVecAdd <<< SIZE/1024, 1024 >>>( dev_vecC, dev_vecA, dev_vecB );
cudaDeviceSynchronize();
ELAPSED_TIME_END(0);
```

- **1D layout**
  - SIZE = 1M
  - gridDim → (SIZE / 1024) blocks → 1024 blocks
  - blockDim → 1024 threads

vecadd-dev.cu – results

- 실행 결과: **118 usec** (GeForce RTX 2070)

CPU	1,891 usec
CUDA, core 1개	60,436 usec
CUDA, 1K blocks	118 usec



```
linux/cuda-work > ./12e-vecadd-dev.exe
elapsed wall-clock time[1] started
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 118 usec
elapsed wall-clock time[1] = 5944 usec
SIZE = 1048576
sumA = 523806.000000
sumB = 523835.312500
sumC = 1047645.187500
diff(sumC, sumA+sumB) =  3.875000
diff(sumC, sumA+sumB) / SIZE =  0.000004
vecA=[ 0.38300  0.88600  0.77700  0.91500 ...  0.83400  0.00000  0.94600  0.64600]
vecB=[ 0.56200  0.78000  0.96600  0.34300 ...  0.65500  0.61000  0.02400  0.16700]
vecC=[ 0.94500  1.66600  1.74300  1.25800 ...  1.48900  0.61000  0.97000  0.81300]
linux/cuda-work > 
```

- 비교: vecadd-host.cpp 는 **1,845 usec**

vecadd-dev.cu – 전체 코드

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

const unsigned SIZE = 1024 * 1024; // 1M elements

// 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];
   }
}

int main(void) {
   // host-side data
   float* vecA = new float[SIZE];
   float* vecB = new float[SIZE];
   float* vecC = new float[SIZE];
```

vecadd-dev.cu – 전체 코드

```
// set random data to A and B
srand( 0 );
setNormalizedRandomData( vecA, SIZE );
setNormalizedRandomData( vecB, SIZE );
// device-side data
float* dev_vecA = nullptr;
float* dev_vecB = nullptr;
float* dev_vecC = nullptr;
// allocate device memory
cudaMalloc( (void**)&dev_vecA, SIZE * sizeof(float) );
cudaMalloc( (void**)&dev_vecB, SIZE * sizeof(float) );
cudaMalloc( (void**)&dev_vecC, SIZE * sizeof(float) );
CUDA_CHECK_ERROR();
// copy to device from host
ELAPSED_TIME_BEGIN(1);
cudaMemcpy( dev_vecA, vecA, SIZE * sizeof(float), cudaMemcpyHostToDevice );
cudaMemcpy( dev_vecB, vecB, SIZE * sizeof(float), cudaMemcpyHostToDevice );
CUDA_CHECK_ERROR();
```

vecadd-dev.cu – 전체 코드

```
// CUDA kernel call
ELAPSED_TIME_BEGIN(0);
kernelVecAdd <<< SIZE / 1024, 1024 >>> ( dev_vecC, dev_vecA, dev_vecB, SIZE );
cudaDeviceSynchronize();
ELAPSED_TIME_END(0);
CUDA_CHECK_ERROR();
// copy to host from device
cudaMemcpy( vecC, dev_vecC, SIZE * sizeof(float), cudaMemcpyDeviceToHost );
ELAPSED_TIME_END(1);
CUDA_CHECK_ERROR();
// free device memory
cudaFree( dev_vecA );
cudaFree( dev_vecB );
cudaFree( dev_vecC );
CUDA_CHECK_ERROR();
```

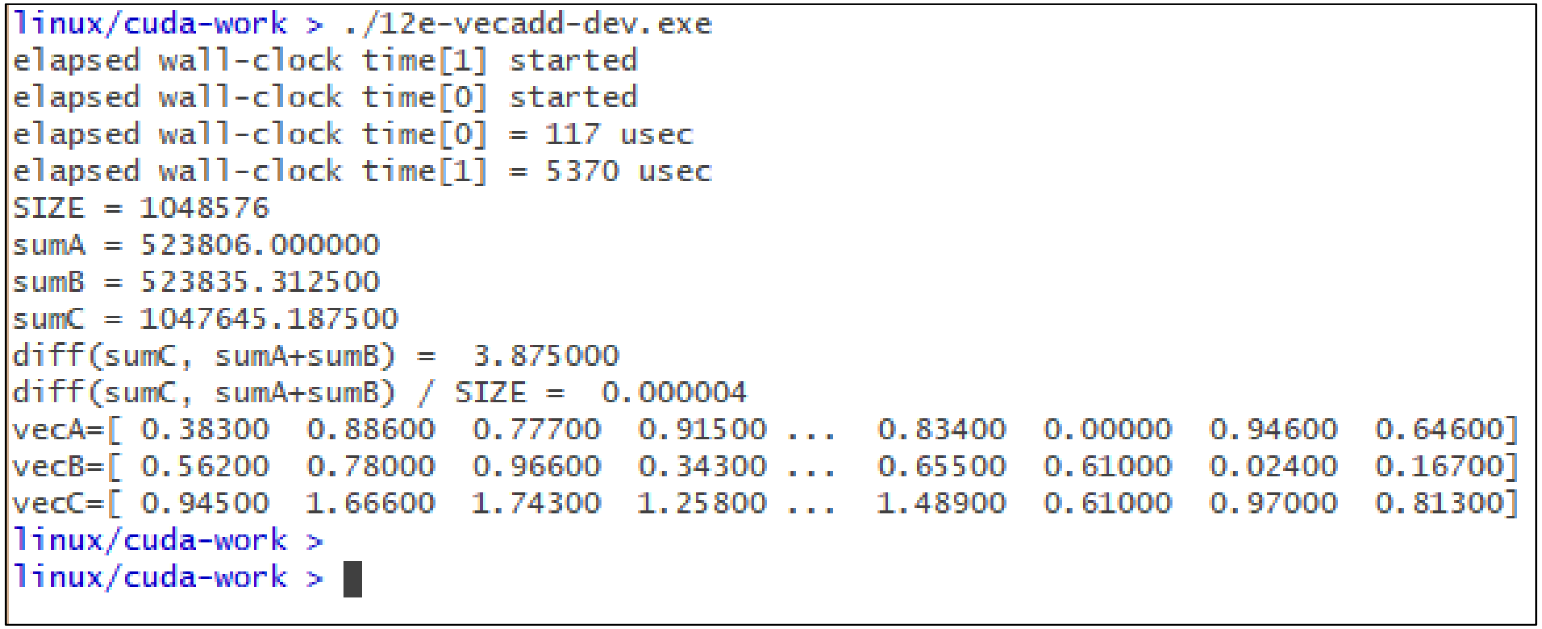
vecadd-dev.cu – 전체 코드

```
// check the result
float sumA = getSum( vecA, SIZE );
float sumB = getSum( vecB, SIZE );
float sumC = getSum( vecC, SIZE );
float diff = fabsf( sumC - (sumA + sumB) );
printf("SIZE = %d\n", SIZE);
printf("sumA = %f\n", sumA);
printf("sumB = %f\n", sumB);
printf("sumC = %f\n", sumC);
printf("diff(sumC, sumA+sumB) =  %f\n", diff);
printf("diff(sumC, sumA+sumB) / SIZE =  %f\n", diff / SIZE);
printVec( "vecA", vecA, SIZE );
printVec( "vecB", vecB, SIZE );
printVec( "vecC", vecC, SIZE );
```

vecadd-dev.cu – 전체 코드

```
    // cleaning
    delete[] vecA;
    delete[] vecB;
    delete[] vecC;
    // done
    return 0;
}
```

CPU	1,891 usec
CUDA, core 1개	60,436 usec
CUDA, 1K blocks	118 usec



```
linux/cuda-work > ./12e-vecadd-dev.exe
elapsed wall-clock time[1] started
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 117 usec
elapsed wall-clock time[1] = 5370 usec
SIZE = 1048576
sumA = 523806.000000
sumB = 523835.312500
sumC = 1047645.187500
diff(sumC, sumA+sumB) =  3.875000
diff(sumC, sumA+sumB) / SIZE =  0.000004
vecA=[ 0.38300  0.88600  0.77700  0.91500 ...  0.83400  0.00000  0.94600  0.64600]
vecB=[ 0.56200  0.78000  0.96600  0.34300 ...  0.65500  0.61000  0.02400  0.16700]
vecC=[ 0.94500  1.66600  1.74300  1.25800 ...  1.48900  0.61000  0.97000  0.81300]
linux/cuda-work >
linux/cuda-work > █
```

C++ 구현

- **kernel function**
  - C++ template 적용은 가능
  - class member 로는 불가능

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

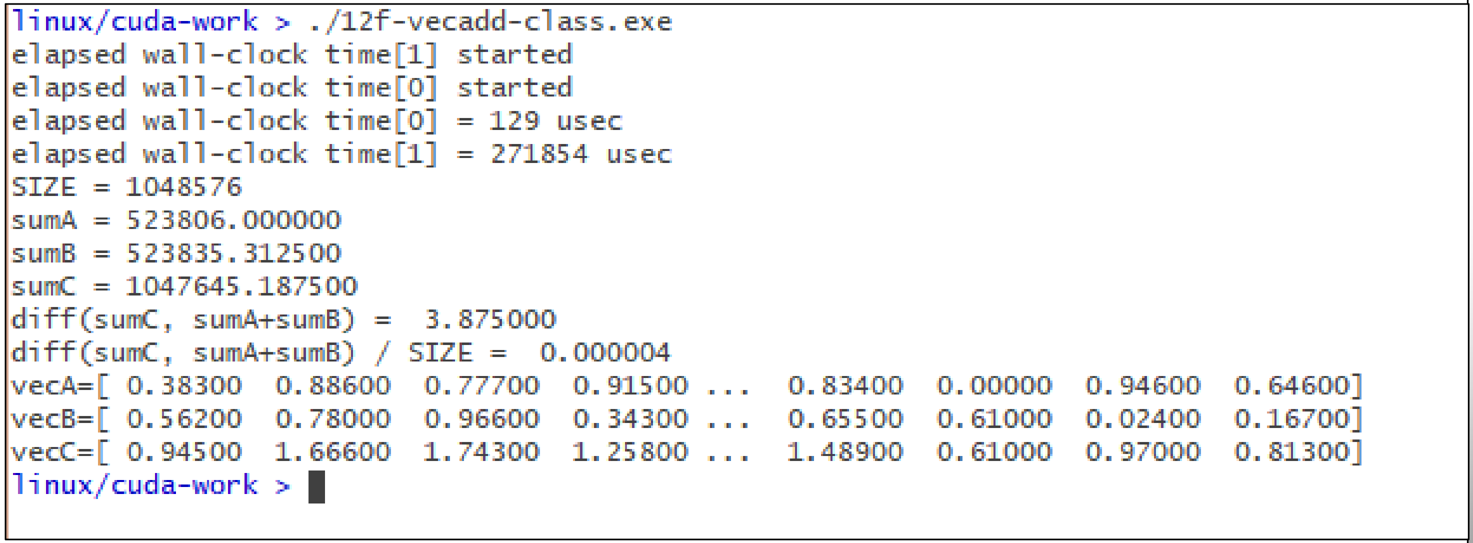
vecadd-class.cu

```
class VecAdd {
protected:
   const unsigned SIZE = 1024 * 1024; // 1M elements
   float* vecA;
   …
public:
   void prepare_host(void) { … }
   void copy_to_device(void) { … }
   void execute_kernel(void) {
      kernelVecAdd<float> <<< SIZE / 1024, 1024 >>> ( dev_vecC, dev_vecA, dev_vecB, SIZE );
      cudaDeviceSynchronize();
      CUDA_CHECK_ERROR();
   }
   …
}
```

vecadd-class.cu 계속

```
int main(void) {
   VecAdd vecadd;
   vecadd.prepare_host();
   ELAPSED_TIME_BEGIN(1);
   vecadd.copy_to_device();
   ELAPSED_TIME_BEGIN(0);
   vecadd.execute_kernel();
   ELAPSED_TIME_END(0);
   vecadd.copy_to_host();
   ELAPSED_TIME_END(1);
   vecadd.check();
   vecadd.clear();
   // done
   return 0;
}
```

CPU	1,891 usec
CUDA, core 1개	60,436 usec
CUDA, 1K blocks	118 usec
CUDA, C++	129 usec



```
linux/cuda-work > ./12f-vecadd-class.exe
elapsed wall-clock time[1] started
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 129 usec
elapsed wall-clock time[1] = 271854 usec
SIZE = 1048576
sumA = 523806.000000
sumB = 523835.312500
sumC = 1047645.187500
diff(sumC, sumA+sumB) =  3.875000
diff(sumC, sumA+sumB) / SIZE =  0.000004
vecA=[ 0.38300  0.88600  0.77700  0.91500 ...  0.83400  0.00000  0.94600  0.64600]
vecB=[ 0.56200  0.78000  0.96600  0.34300 ...  0.65500  0.61000  0.02400  0.16700]
vecC=[ 0.94500  1.66600  1.74300  1.25800 ...  1.48900  0.61000  0.97000  0.81300]
linux/cuda-work > 
```

내용 contents

- **vector addition 설명**
  - vector = 1D array   1 million elements
- **host version – CPU 사용**   **1,891 usec** **(Intel Core i5-3570)**
- **CUDA version – core 1개 사용**   **60,436 usec**
- **CUDA version – 1K blocks * 1K threads**   **118 usec** **(GeForce RTX 2070)**
- **CUDA version – C++ flavor**   **129 usec**

Vector Addition

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