CUDA 프로그래밍

CUDA Programming

biztripcru@gmail.com

AXPY and FMA

AXPY 문제와 FMA 연산

내용 contents

- **AXPY 루틴**
  - SAXPY – CPU 구현
  - SAXPY – CUDA 구현
- **FMA instruction**
  - SAXPY – FMA 적용
- **LERP : linear interpolation**
  - LERP – CUDA 구현
  - LERP – FMA 적용

AXPY 루틴 routine

- BLAS (Basic Linear Algebra Subprograms)
  - 선형 대수 linear algebra, 행렬/벡터 계산에서 매우 유명한 표준 라이브러리
- BLAS 에서 제공하는 AXPY 루틴 routine
  - A X plus Y (a X + Y)
  - **Z ← a X + Y**
    - X, Y, Z 는 vector, a 는 scalar 값
  - SAXPY : single precision (float)
  - DAXPY : double precision (double)
  - CAXPY : complex numbers

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_n \end{bmatrix} = a \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

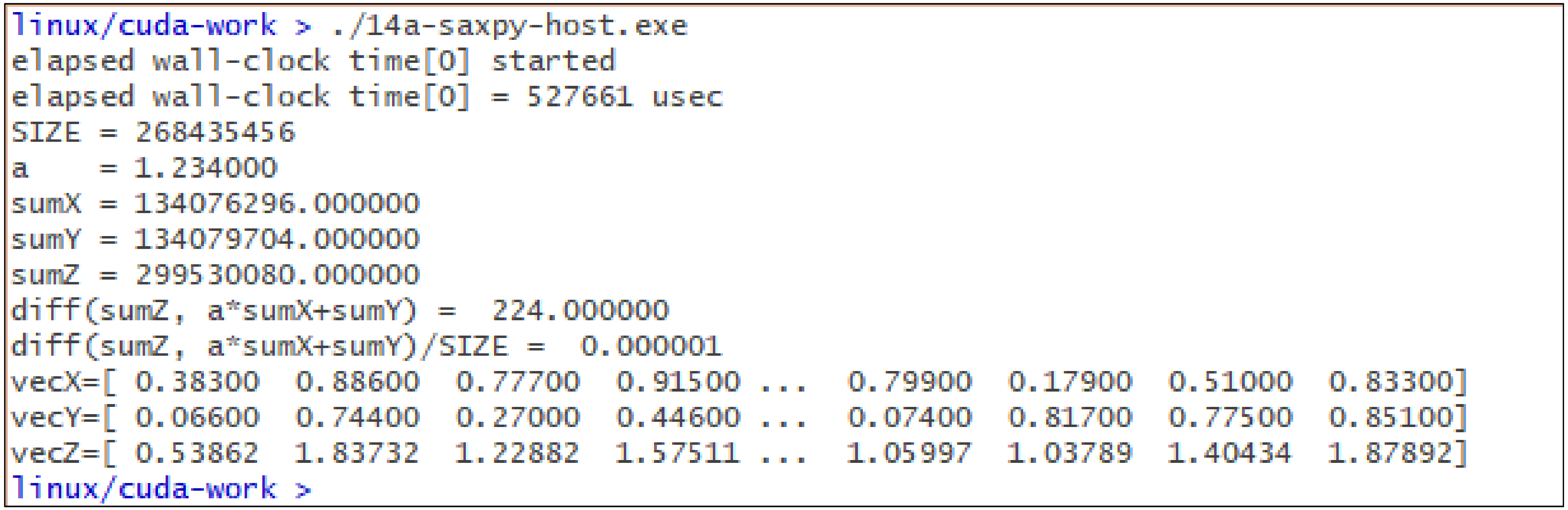
saxpy-host.cpp

```
// input parameters
unsigned vecSize = 256 * 1024 * 1024; // big-size elements
float saxpy_a = 1.234f;

int main( const int argc, const char* argv[] ) {
   …
   vecZ = new float[vecSize];
   …
   // kernel: vector addition
   ELAPSED_TIME_BEGIN(0);
   for (register unsigned i = 0; i < vecSize; ++i) {
      vecZ[i] = saxpy_a * vecX[i] + vecY[i];
   }
   ELAPSED_TIME_END(0);
   …
}
```

saxpy-host.cpp 실행 결과

- 527,661 usec for 256M element vectors (Intel Core i5-3570)

SAXPY – CPU 527,661 usec



```
linux/cuda-work > ./14a-saxpy-host.exe
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 527661 usec
SIZE = 268435456
a    = 1.234000
sumX = 134076296.000000
sumY = 134079704.000000
sumZ = 299530080.000000
diff(sumZ, a*sumX+sumY) =  224.000000
diff(sumZ, a*sumX+sumY)/SIZE =  0.000001
vecX=[ 0.38300  0.88600  0.77700  0.91500 ...  0.79900  0.17900  0.51000  0.83300]
vecY=[ 0.06600  0.74400  0.27000  0.44600 ...  0.07400  0.81700  0.77500  0.85100]
vecZ=[ 0.53862  1.83732  1.22882  1.57511 ...  1.05997  1.03789  1.40434  1.87892]
linux/cuda-work >
```

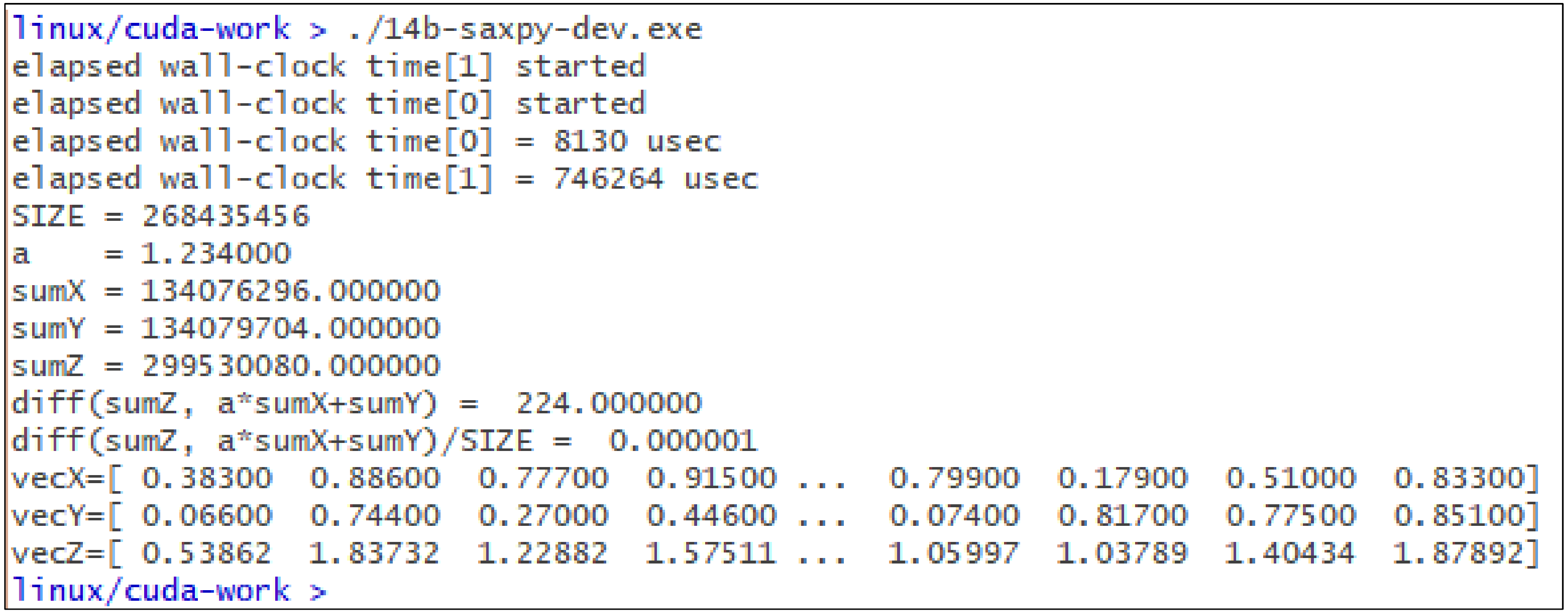
saxpy-dev.cu

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

int main( const int argc, const char* argv[] ) {
   …
   // CUDA kernel call
   dim3 dimBlock( 1024, 1, 1 );
   dim3 dimGrid( (vecSize + dimBlock.x - 1) / dimBlock.x, 1, 1 );
   kernelSAXPY <<< dimGrid, dimBlock >>> ( dev_vecZ, saxpy_a, dev_vecX, dev_vecY, vecSize );
   cudaDeviceSynchronize();
   …
}
```

saxpy-dev.cu 실행 결과

- 8,130 usec for 256M element vectors (GeForce RTX 2070)

SAXPY – CPU	527,661 usec
SAXPY – CUDA	8,130 usec



```
linux/cuda-work > ./14b-saxpy-dev.exe
elapsed wall-clock time[1] started
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 8130 usec
elapsed wall-clock time[1] = 746264 usec
SIZE = 268435456
a    = 1.234000
sumX = 134076296.000000
sumY = 134079704.000000
sumZ = 299530080.000000
diff(sumZ, a*sumX+sumY) =  224.000000
diff(sumZ, a*sumX+sumY)/SIZE =  0.000001
vecX=[ 0.38300  0.88600  0.77700  0.91500 ...  0.79900  0.17900  0.51000  0.83300]
vecY=[ 0.06600  0.74400  0.27000  0.44600 ...  0.07400  0.81700  0.77500  0.85100]
vecZ=[ 0.53862  1.83732  1.22882  1.57511 ...  1.05997  1.03789  1.40434  1.87892]
linux/cuda-work >
```

MAC/MAD and FMA/FMAC

- **goal: multiply-add**
  - $z \leftarrow a * x + y$
- **MAC/MAD : multiply-accumulate / multiply-add instruction (old style)**
  - implementation: round(round(a * x) + y)
- **FMA/FMAC : fused multiply-add / fused multiply-accumulate instruction**
  - fused = blended, integrated
  - new implementation: round(a * x + y)
  - 약간 빠름
  - **정밀도**가 약간 올라감 ! → deep learning 등에서 이쪽을 선호

FMA

- **fused multiply-add instruction**
  - also known as **FMA** or **FMAC** or **FMADD**
- **performs floating-point multiplication and addition as:**
  - $z \leftarrow a\,x + y$
  - **in one step (!)**
  - CPU / GPU 에서 machine instruction 으로 구현해서 제공
- **in CUDA math library,**
  - float fmaf(float *a*, float *x*, float *y*);
  - double fma(double *a*, double *x*, double *y*);
  - returns (a * x + y) with **FMA instruction**

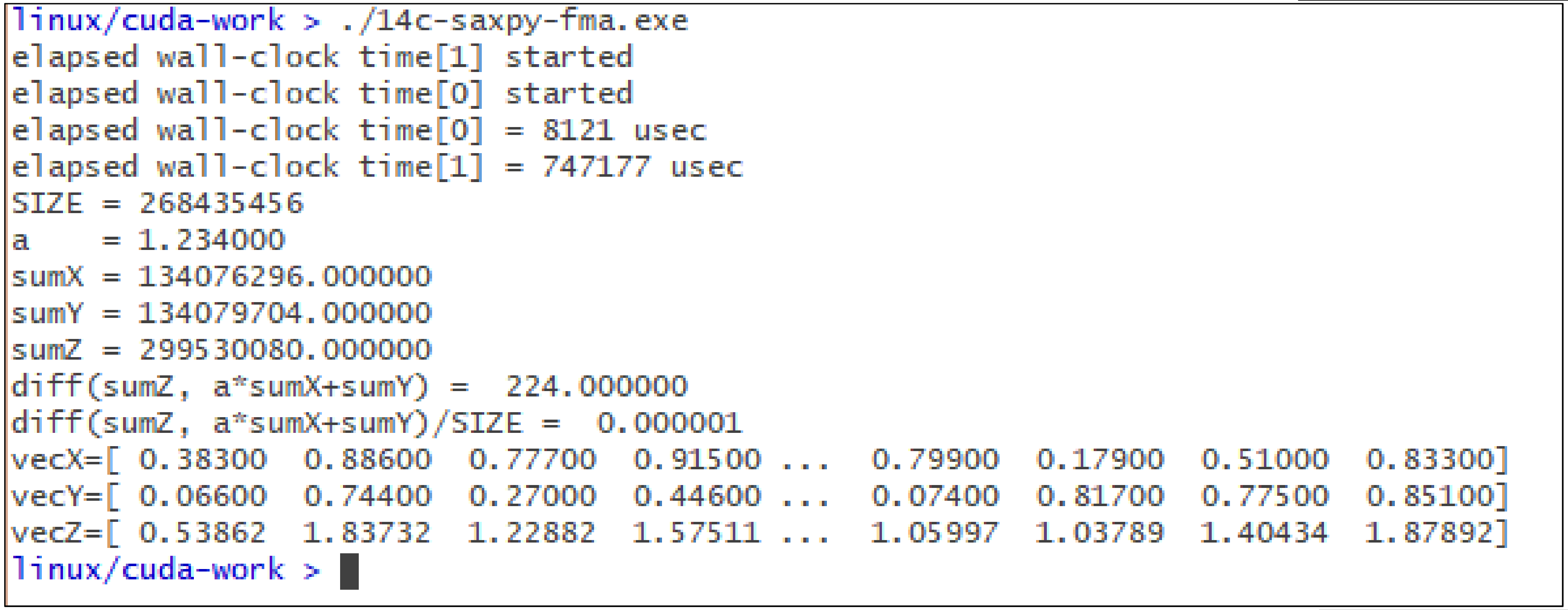
saxpy-fma.cu

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

int main( const int argc, const char* argv[] ) {
   …
   // CUDA kernel call
   dim3 dimBlock( 1024, 1, 1 );
   dim3 dimGrid( (vecSize + dimBlock.x - 1) / dimBlock.x, 1, 1 );
   kernelSAXPY <<< dimGrid, dimBlock >>> ( dev_vecZ, saxpy_a, dev_vecX, dev_vecY, vecSize );
   cudaDeviceSynchronize();
   …
}
```

saxpy-fma.cu

- 8,121 usec for 256M element vectors (GeForce RTX 2070)
  - 아직 의미 있는 결과 meaningful result 는 아님

SAXPY – CPU	527,661 usec
SAXPY – CUDA	8,130 usec
SAXPY – FMA	8,121 usec

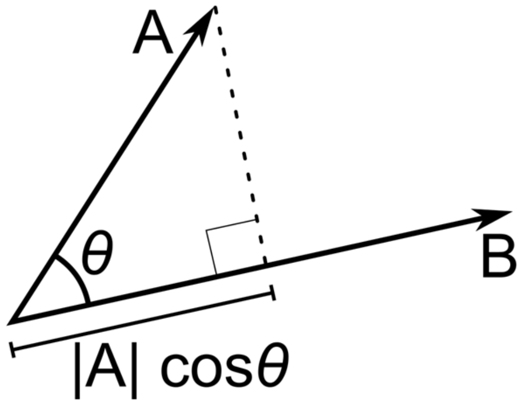


```
linux/cuda-work > ./14c-saxpy-fma.exe
elapsed wall-clock time[1] started
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 8121 usec
elapsed wall-clock time[1] = 747177 usec
SIZE = 268435456
a    = 1.234000
sumX = 134076296.000000
sumY = 134079704.000000
sumZ = 299530080.000000
diff(sumZ, a*sumX+sumY) =  224.000000
diff(sumZ, a*sumX+sumY)/SIZE =  0.000001
vecX=[ 0.38300  0.88600  0.77700  0.91500 ...  0.79900  0.17900  0.51000  0.83300]
vecY=[ 0.06600  0.74400  0.27000  0.44600 ...  0.07400  0.81700  0.77500  0.85100]
vecZ=[ 0.53862  1.83732  1.22882  1.57511 ...  1.05997  1.03789  1.40434  1.87892]
linux/cuda-work > 
```

FMA applications

- 벡터의 내적 dot product
  - for two vectors (a_x, a_y, a_z) and (b_x, b_y, b_z)
  - calculate $a_x * b_x + a_y * b_y + a_z * b_z$

- FMA implementation:
  - answer ← 0
  - answer ← fma(ax, bx , answer)
  - answer ← fma(ay, by , answer)
  - answer ← fma(az, bz , answer)

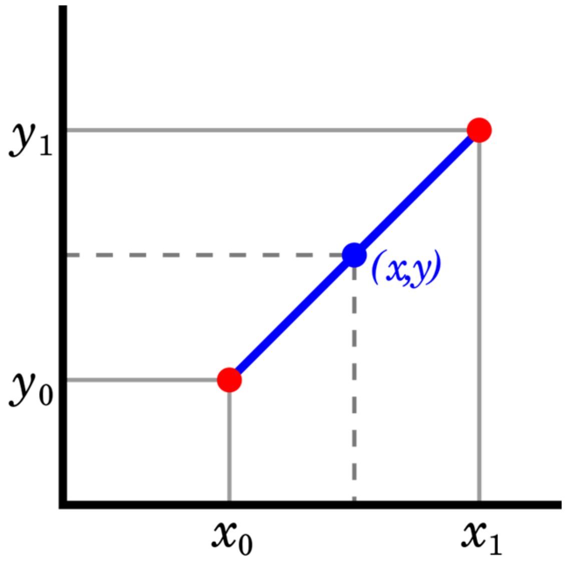


|A·B| = |A| |B| cos θ

$$\vec{a} \cdot \vec{b} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix} \cdot \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

FMA applications 계속

- **lerp 러프 : linear interpolation 선형 보간법**
  - $f(t) = (1 - t)\, v_0 + t\, v_1$
  - $v_0 = (x_0, y_0)$
  - $v_1 = (x_1, y_1)$



public domain
https://commons.wikimedia.org/wiki/File:Linear_interpolation.png#/media/File:LinearInterpolation.svg

- **FMA implementation**
  - $f(t) = (1 - t)\, v_0 + t\, v_1 = (v_0 - t\, v_0) + t\, v_1$
  - fma(t, v_1, fma(–t, v_0, v_0))

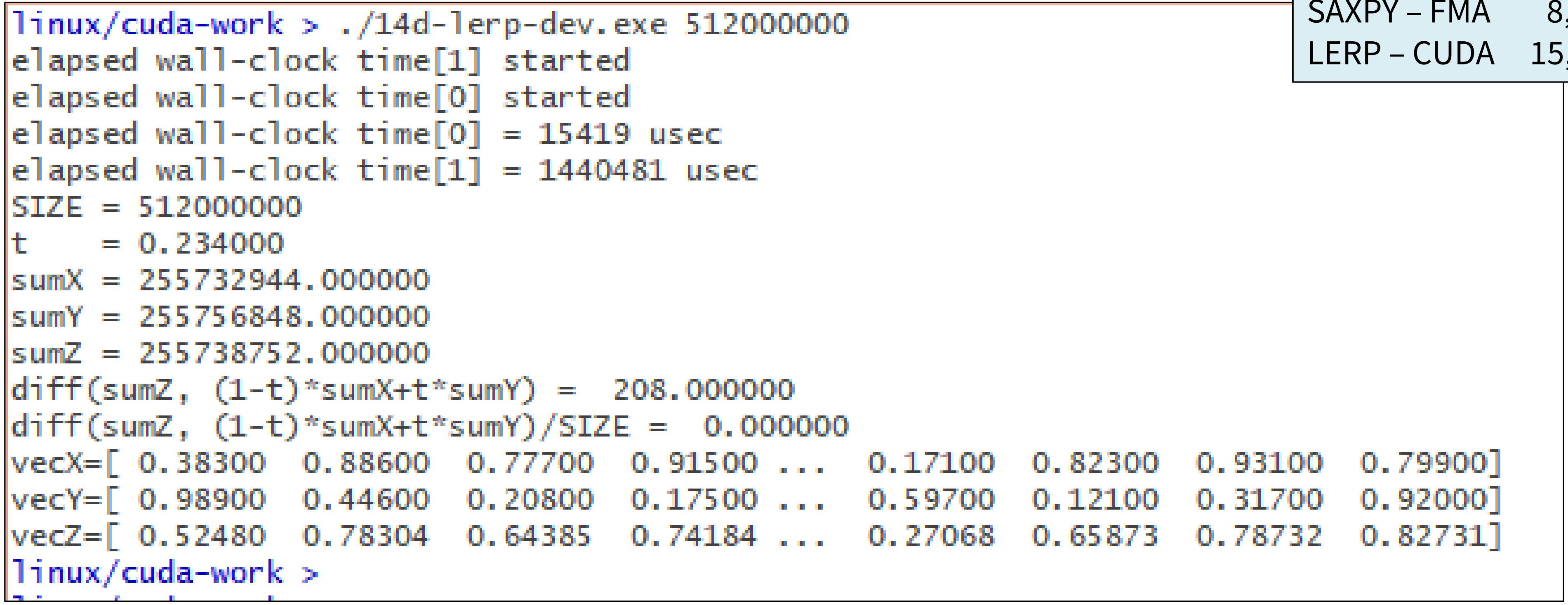
lerp-dev.cu

```
// input parameters
unsigned vecSize = 256 * 1024 * 1024; // big-size elements
float lerp_t = 0.234F;

// CUDA kernel function
__global__ void kernel_lerp( float* z, const float t, const float* x, const float* y, unsigned n ) {
   unsigned i = blockIdx.x * blockDim.x + threadIdx.x; // CUDA-provided index
   if (i < n) {
      z[i] = (1.0F - t) * x[i] + t * y[i];
   }
}

int main( const int argc, const char* argv[] ) {
   …
}
```

lerp-dev.cu 실행 결과

- **15,419 usec for 512,000,000 cases** **(GeForce RTX 2070)**

SAXPY – CPU	527,661 usec
SAXPY – CUDA	8,130 usec
SAXPY – FMA	8,121 usec
LERP – CUDA	15,419 usec



```
linux/cuda-work > ./14d-lerp-dev.exe 512000000
elapsed wall-clock time[1] started
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 15419 usec
elapsed wall-clock time[1] = 1440481 usec
SIZE = 512000000
t    = 0.234000
sumX = 255732944.000000
sumY = 255756848.000000
sumZ = 255738752.000000
diff(sumZ, (1-t)*sumX+t*sumY) =  208.000000
diff(sumZ, (1-t)*sumX+t*sumY)/SIZE =  0.000000
vecX=[ 0.38300  0.88600  0.77700  0.91500 ...  0.17100  0.82300  0.93100  0.79900]
vecY=[ 0.98900  0.44600  0.20800  0.17500 ...  0.59700  0.12100  0.31700  0.92000]
vecZ=[ 0.52480  0.78304  0.64385  0.74184 ...  0.27068  0.65873  0.78732  0.82731]
linux/cuda-work >
```

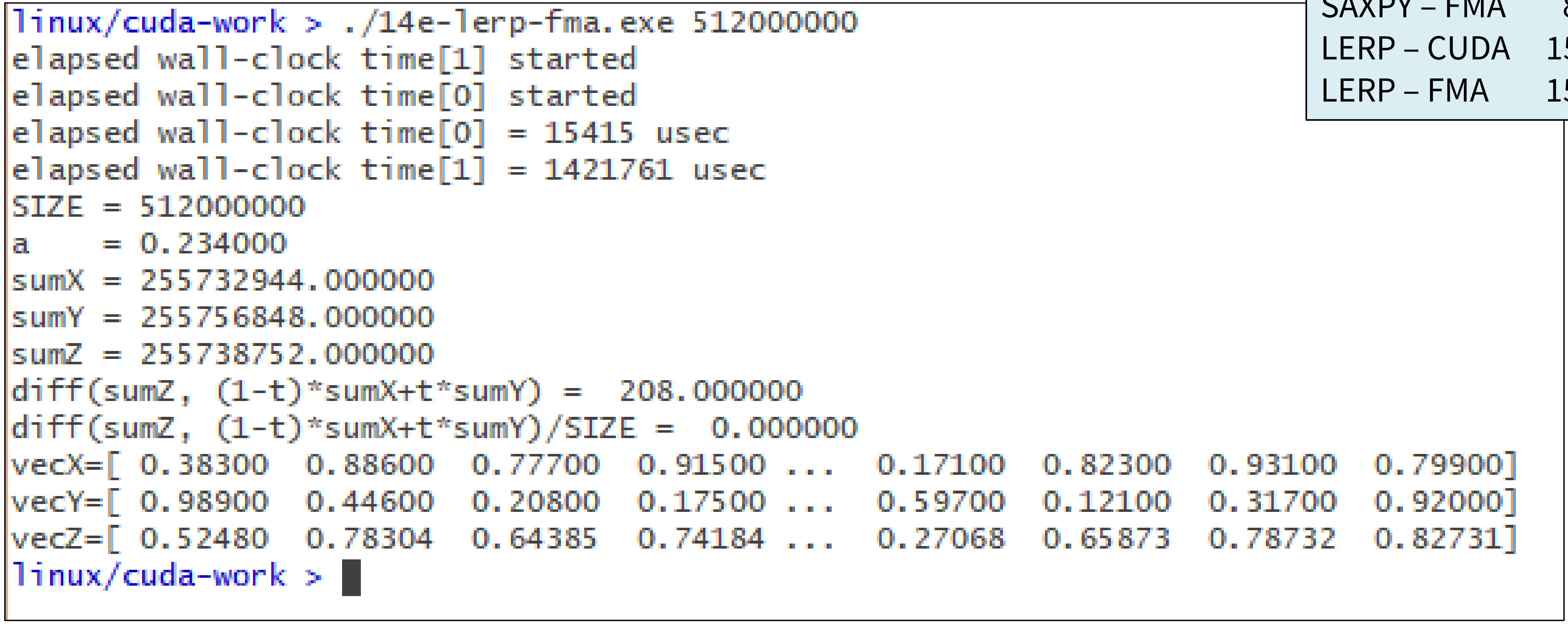
lerp-fma.cu

```
// input parameters
unsigned vecSize = 256 * 1024 * 1024; // big-size elements
float lerp_t = 0.234f;

// CUDA kernel function
__global__ void kernel_lerp( float* z, const float t, const float* x, const float* y, unsigned n ) {
   unsigned i = blockIdx.x * blockDim.x + threadIdx.x; // CUDA-provided index
   if (i < n) {
      z[i] = fmaf( t, y[i], fmaf( -t, x[i], x[i] ) );
   }
}

int main( const int argc, const char* argv[] ) {
   …
}
```

lerp-fma.cu

- 15,415 usec for 512,000,000 cases (GeForce RTX 2070)

SAXPY – CPU	527,661 usec
SAXPY – CUDA	8,130 usec
SAXPY – FMA	8,121 usec
LERP – CUDA	15,419 usec
LERP – FMA	15,415 usec


```
linux/cuda-work > ./14e-lerp-fma.exe 512000000
elapsed wall-clock time[1] started
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 15415 usec
elapsed wall-clock time[1] = 1421761 usec
SIZE = 512000000
a    = 0.234000
sumX = 255732944.000000
sumY = 255756848.000000
sumZ = 255738752.000000
diff(sumZ, (1-t)*sumX+t*sumY) =  208.000000
diff(sumZ, (1-t)*sumX+t*sumY)/SIZE =  0.000000
vecX=[ 0.38300  0.88600  0.77700  0.91500 ...  0.17100  0.82300  0.93100  0.79900]
vecY=[ 0.98900  0.44600  0.20800  0.17500 ...  0.59700  0.12100  0.31700  0.92000]
vecZ=[ 0.52480  0.78304  0.64385  0.74184 ...  0.27068  0.65873  0.78732  0.82731]
linux/cuda-work >
```

argument processing

- **in "./common.cpp",**

```
template <typename TYPE>
TYPE procArg( const char* progname, const char* str,
                                        TYPE lbound = –1, TYPE ubound =  –1);
```

  - *progname* : program name (or argv[0])
  - *str* : argument string to be processed
  - *lbound*, *ubound* : lower and upper bound for valid values

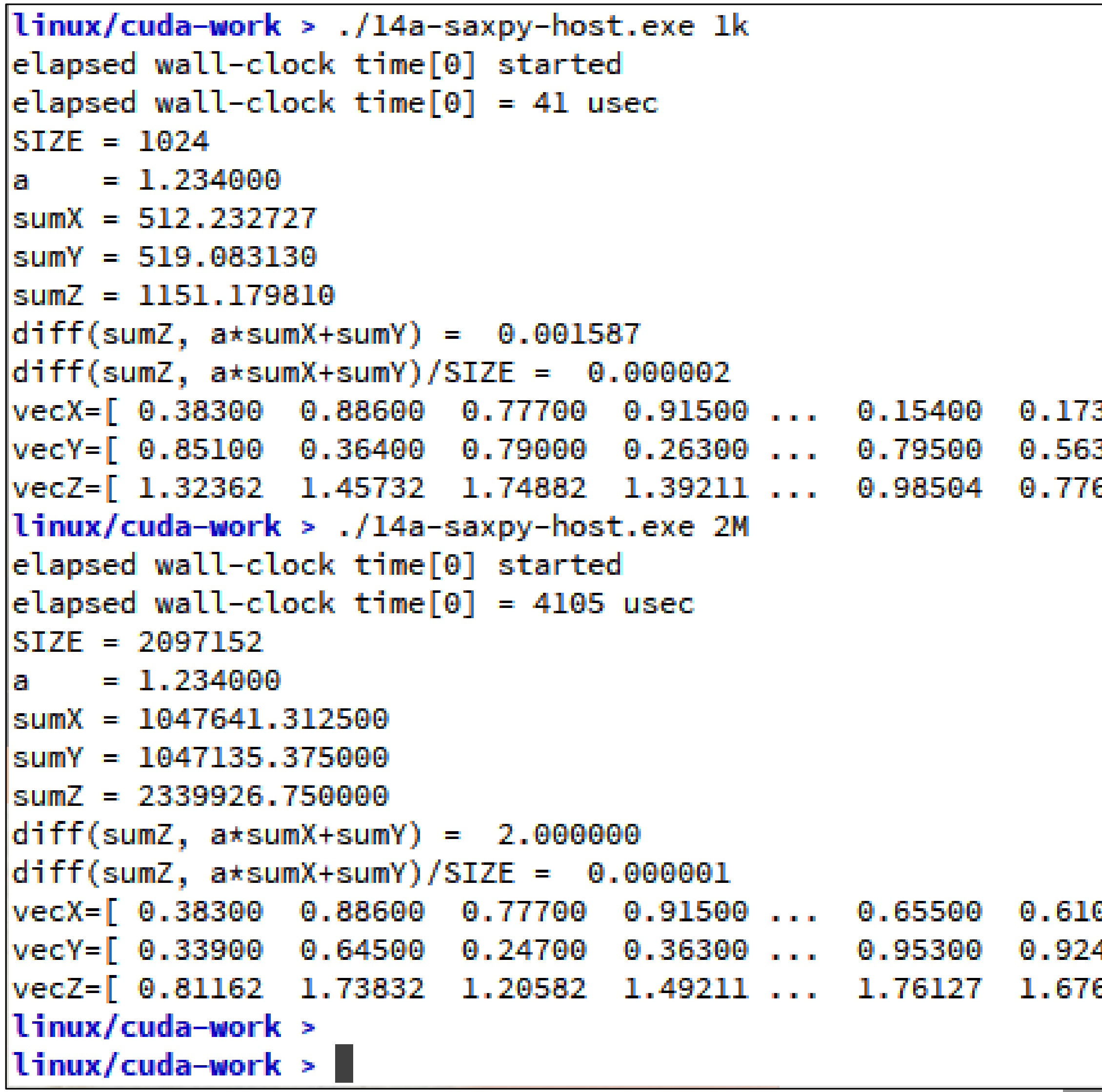
- **example:**
  - `unsigned  vecSize = procArg( argv[0], argv[1], 1, 1000 );`
  - `float  saxpy_a = procArg<float>( argv[0], argv[2] );  // without bound checks`

saxpy-host.cpp again

```
// input parameters
unsigned vecSize = 256 * 1024 * 1024; // big-size elements
float saxpy_a = 1.234f;

int main( const int argc, const char* argv[] ) {
   // argv processing
   switch (argc) {
   case 1:
      break;
   case 2:
      vecSize = procArg( argv[0], argv[1], 1 );
      break;
   case 3:
      vecSize = procArg( argv[0], argv[1], 1 );
      saxpy_a = procArg<float>( argv[0], argv[2] );
      break;
   default:
      printf("usage: %s [num] [a]\n", argv[0]);
      exit( EXIT_FAILURE );
      break;
   }
```

saxpy-host.cpp again

- **argument processing 의 추가 기능**
  - k, K : kilo (= 1,024) 로 해석
  - m, M : million 으로 해석



```
linux/cuda-work > ./14a-saxpy-host.exe 1k
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 41 usec
SIZE = 1024
a    = 1.234000
sumX = 512.232727
sumY = 519.083130
sumZ = 1151.179810
diff(sumZ, a*sumX+sumY) =  0.001587
diff(sumZ, a*sumX+sumY)/SIZE =  0.000002
vecX=[ 0.38300  0.88600  0.77700  0.91500 ...  0.15400  0.173
vecY=[ 0.85100  0.36400  0.79000  0.26300 ...  0.79500  0.563
vecZ=[ 1.32362  1.45732  1.74882  1.39211 ...  0.98504  0.776
linux/cuda-work > ./14a-saxpy-host.exe 2M
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 4105 usec
SIZE = 2097152
a    = 1.234000
sumX = 1047641.312500
sumY = 1047135.375000
sumZ = 2339926.750000
diff(sumZ, a*sumX+sumY) =  2.000000
diff(sumZ, a*sumX+sumY)/SIZE =  0.000001
vecX=[ 0.38300  0.88600  0.77700  0.91500 ...  0.65500  0.610
vecY=[ 0.33900  0.64500  0.24700  0.36300 ...  0.95300  0.924
vecZ=[ 0.81162  1.73832  1.20582  1.49211 ...  1.76127  1.676
linux/cuda-work >
linux/cuda-work > 
```

내용 contents

- **AXPY 루틴**
  - SAXPY – CPU 구현 527,661 usec
  - SAXPY – CUDA 구현 8,130 usec
- **FMA instruction**
  - SAXPY – FMA 적용 8,121 usec
- **LERP : linear interpolation**
  - LERP – CUDA 구현 15,419 usec
  - LERP – FMA 적용 15,415 usec

AXPY and FMA

AXPY 문제와 FMA 연산

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