Parallel Programming with CUDA

Streltsova O., Zuev M.

Laboratory of Information Technologies

Joint Institute for Nuclear Research

cuBLAS library: cublas<t>scal()

The cuBLAS library is an implementation of BLAS (Basic Linear Algebra Subprograms) on top of the NVIDIA®CUDA runtime.

Site:

http://docs.nvidia.com/cuda/cublas/

- cublasStatus_t cublasSscal(cublasHandle_t handle, int n,
  const float *alpha, float *x, int incx)
- cublasStatus_t cublasDscal(cublasHandle_t handle, int n,
  const double *alpha, double *x, int incx)
- cublasStatus_t cublasCscal(cublasHandle_t handle, int n,
  const cuComplex *alpha, cuComplex *x, int incx)
- cublasStatus_t cublasCsscal(cublasHandle_t handle, int n,
  const float *alpha, cuComplex *x, int incx)
- cublasStatus_t cublasZscal(cublasHandle_t handle, int n,
  const cuDoubleComplex *alpha,
  cuDoubleComplex *x, int incx)
- cublasStatus_t cublasZdscal(cublasHandle_t handle, int n,
  const double *alpha, cuDoubleComplex *x, int incx)

Program example

Location

```
cd Tutorial_HSchool2014/CUDA
```

File

cuda_blas.cu

Program example

```
#include <stdio.h>
#include <stdlib.h>
#include <cublas_v2.h>  // API v2 from CUDA 4.0

#include <math.h>
#include <cuda.h>

#define NX 16384 // 64

//--------------------------  Error Handling  -----------------------------//
#define CUDA_CALL(x) do { cudaError_t err = x; if (( err ) != cudaSuccess ){ \
printf ("Error \"%s\" at %s :%d \n" , cudaGetErrorString(err), \
__FILE__ , __LINE__ ) ; return -1;\
}} while (0);

#define CUBLAS_CALL(x) do { if (( x ) != CUBLAS_STATUS_SUCCESS ) {\
printf ("Error at %s :%d \n" , __FILE__ , __LINE__ ) ;\
return -1; }} while (0);
```

Program example

```
int main() {
    cublasHandle_t handleNorm;
    cublasCreate(&handleNorm) ;

    printf(" ==== Element-wise  vector  multiplication  by  a  scalar ==== \n" );
    printf(“ Array  size  NX  =  %d\n", NX);

    //------------------- Input data: allocate memory on CPU -------------------//
    double* A_input = new double[NX];
    double* A_out    = new double[NX];

    for (int ix= 0; ix< NX; ix++){
        A_input[ix]= (double)(ix);
    }

    double normScalar= 3.0*M_PI;
```

Program example

```
35.   //------------------  Allocate  memory on device --------------------//
36.   double* dev_Ainput;
37.   CUDA_CALL( cudaMalloc((void**)&dev_Ainput, NX*sizeof(double)) );
```

```
cudaMalloc ( void ** devPtr,  // Pointer to allocated memory
             size_t  size )   // Requested allocation size in bytes
```

```
...
Error "out of memory" at cuda_blas.cu :37
```

```
38.   //-------------------- Timing a CUDA application using events --------------------//
39.   cudaEvent_t start, stop_ini,  stop_cublas, stop_cpu;
40.   cudaEventCreate(&start);
41.   cudaEventCreate(&stop_ini);
42.   cudaEventCreate(&stop_cublas);
43.   cudaEventCreate(&stop_cpu);
44.
45.   cudaEventRecord(start);
```

Program example

```
//------------------- Copy from Host to Device Input data --------------------//
CUDA_CALL( cudaMemcpy(dev_Ainput , A_input, NX*sizeof(double),
                                              cudaMemcpyHostToDevice) );
```

```
cudaMemcpy ( void *                 dst,    // Destination
             const void *           src,    // Source
             size_t                 count,  // Size
             enum cudaMemcpyKind    kind )  // Type of transfer

cudaMemcpyKind: cudaMemcpyHostToHost, cudaMemcpyHostToDevice,
cudaMemcpyDeviceToHost, cudaMemcpyDeviceToDevice
```

Program example

```
//------------------- Calculate time --------------------//
cudaEventRecord(stop_ini, 0);
float time_ini = 0.0;
cudaEventSynchronize(stop_ini);
cudaEventElapsedTime(&time_ini, start, stop_ini);
printf("GPU compute time_load data(msec): %.5f\n", time_ini);

cudaError_t error = cudaGetLastError();
if(error != cudaSuccess){
    printf("CUDA error, CUBLAS:  %s\n", cudaGetErrorString(error));
    exit(-1);
}

//------------------- Check error --------------------//
cudaError_t error = cudaGetLastError();
if(error != cudaSuccess){
    printf("CUDA error, CUFFT:  %s\n", cudaGetErrorString(error));
    exit(-1);
}
```

Program example

```
//----------------- For multiplication use function of CUBLAS library --------------------//
CUBLAS_CALL( cublasDscal( handleNorm, NX, &normScalar, &dev_Ainput[0], 1) );
```

```
cublasDscal( cublasHandle_t handle,  // Handle to the cuBLAS library context
             int            n,       // Number of elements in the vector x
             const double * alpha,   // Scalar used for multiplication
             double *       x,       // Elements in the vector x
             int            incx )   // Stride between elements of x
```

```
//-------------------- Timing --------------------//
cudaEventRecord(stop_cublas);
float time_cublas = 0.0;
cudaEventSynchronize(stop_cublas);
cudaEventElapsedTime(&time_cublas, stop_ini, stop_cublas);
printf("GPU compute time_cublas data(msec): %.5f\n", time_cublas);
```

Program example

```
//------------------- CPU computation -------------------//
for (int ix= 0; ix< NX; ix++)
    A_out[ix]= A_input[ix]*normScalar;

cudaEventRecord(stop_cpu);
float time_cpu = 0.0;
cudaEventSynchronize(stop_cpu);
cudaEventElapsedTime(&time_cpu, stop_cublas,stop_cpu);
printf("CPU compute time_cpu (msec): %.5f\n", time_cpu);

//------------------- Copy from Device to Host Output data -------------------//
CUDA_CALL( cudaMemcpy(A_out, dev_Ainput, NX*sizeof(double),
                                            cudaMemcpyDeviceToHost) );
cudaDeviceSynchronize();
```

Program example

```
//-------------------- Output result  on screen and file --------------------//
char* file_name = "Rezult_GPU.dat" ;
printf("============  Test result Vector for [0] and [1] elements \n");
printf(  "ix=0, iy =0, A_out = %.16e, A_input = %.16e  \n", A_out[0], A_input[0]);
printf(  "ix=1, iy =0, A_out = %.16e, A_input = %.16e  \n", A_out[1], A_input[1]);

FILE *fp4;
fp4=fopen(file_name, "w");
fprintf(fp4, "ix   ,   A_out[ix]),   A_input[ix]" );
for (int ix = 0; ix < NX; ix++)
    fprintf(fp4, " %d    %.16e   %.16e  \n", ix, A_out[ix], A_input[ix]);

fclose(fp4);
cublasDestroy(handleNorm) ;
cudaFree(dev_Ainput);
delete[] A_input;
delete[] A_out;
return 0;
}
```

Program example. Compilation

Add module

```
> module add cuda-6.0-x86_64
```

Compilation with libraries

```
> nvcc -gencode=arch=compute_35,code=sm_35
-lcublas -O3 cublas_ex.cu -o cublas
```

Program example. Running

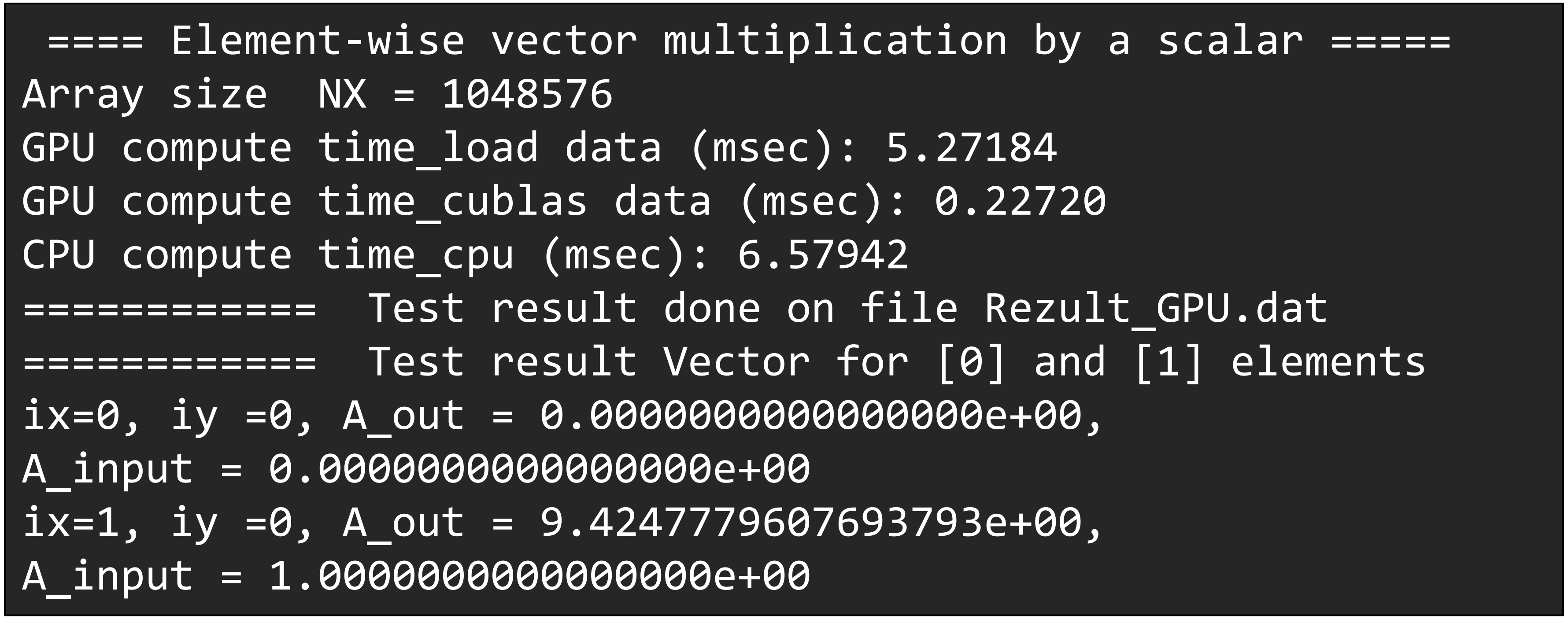
Listing of ***script_cuda***

```
#!/bin/sh
#SBATCH -p gpu
./test1
```

Running in batch

```
> sbatch script_cuda
```

Program example. Output on the screen



```
 ==== Element-wise vector multiplication by a scalar =====
Array size  NX = 1048576
GPU compute time_load data (msec): 5.27184
GPU compute time_cublas data (msec): 0.22720
CPU compute time_cpu (msec): 6.57942
============  Test result done on file Rezult_GPU.dat
============  Test result Vector for [0] and [1] elements
ix=0, iy =0, A_out = 0.0000000000000000e+00,
A_input = 0.0000000000000000e+00
ix=1, iy =0, A_out = 9.4247779607693793e+00,
A_input = 1.0000000000000000e+00
```