RT-CUDA User Guide with Multiple Kernels Support

AYAZ H. KHAN*
ayazhk@gmail.com

February 4, 2015

1 RT-CUDA PACKAGE README

1.1 RT-CUDA Installation and Setup

1.1.1 *Pre-Requisites*

- Java JDK/JRE 1.7 or later, it can be downloaded from the following link: (www.oracle.com/technetwork/java/javase/downloads/index.html)
- NVIDIA CUDA Toolkit, it can be downloaded from the following link: (https://developer.nvidia.com/cuda-toolkit)

1.1.2 *Package Extraction*

To extract RT-CUDA package, run the following command:
unzip RTCUDACompiler.zip

1.2 RT-CUDA Usage

1.2.1 *Input*

RT-CUDA compiler requires following files to be created in src folder:

- **<kernelname>kernel.c:** This file contains the C function that needs to be converted as CUDA kernel to run on GPU device. <kernelname> is the name of the C function defined in the file to be converted to CUDA kernel. Separate file should be created to each function. The function should follow the ANSI C standard with required parameters and no return type. This function implements the C-Loop structure to be partitioned among multiple CUDA blocks and threads to compute the required results. Following is the syntax of the function definition:

  void func_name(type param, ...){ < functionbody > }

- **main.c:** This file contains the C main function that implements the user input, data allocation, initialization, function call (defined in kernel.c), and output of the program following the ANSI C standard. All the arrays that are going to be used by the kernel function should be allocated dynamically using C malloc() function. To run the kernel

on a particular GPU device, user should use the function cudaSetDevice(GPU_ID) before calling the kernel function where GPU_ID is the id of the GPU device available in the system.

- **<kernelfilename>config.txt:** This file contains the configurations for different parameters used by the compiler for optimizations and final code generation. <kernelfilename> is the name of the related <kernelname>kernel.c file. Following are the list of parameters that need to be defined in configuration:
  - **LOOP_COLLAPSING:** Enabled(1)/Disabled(0) loop collapsing optimization. It is only applicable to the kernel with 2D resultant matrix and having two nested loops in the computations.
  - **BLOCK_SKEW:** Enabled(1)/Disabled(0) block skewing optimization. It is only applicable to the kernel with 2D resultant matrix. It increases the thread access locality by merging multiple resultant blocks horizontally to one thread block.
  - **PREFETCHING:** Enabled(1)/Disabled(0) prefetching optimization. It is only applicable to the kernel having 2D matrix in the computation that need to be tiled to store in shared memory.
  - **PREFETCHED_ARRAYS:** List of array variables that need to be tiled. This is only applied if PREFETCHING is enabled.
  - **NON_PREFETCHED_ARRAYS:** List of array variables that should be ignored for tiling. This is only applied if PREFTECHING is enabled.
  - **DATA_TYPE:** It defines the data type of the arrays in the computation and resultant that need to be stored in GPU memory.
  - **KERNEL_NAMES:** name of the function to be converted to CUDA kernel.
  - **2DMATRIX:** It is set to 1 for 2D resultant matrix and 0 for 1D.
  - **ROW_DIM:** This is the leading dimension of the matrices used in computation.
  - **MAX_BLOCKSIZE:** Upper bound of BLOCKSIZE to be analyzed by RT-CUDA Parameter Tunner. This should be less than or equal to the maxmimum possible thread block size of the underlying GPU compute capability. To check all possible block size based on the underlying GPU architecture automatically, set this value to 0.
  - **MAX_MERGE_LEVEL:** Upper bound of MERGE_LEVEL to be analyzed by RT-CUDA Parameter Tunner. This should be less than or equal to the MAX_BLOCKSIZE.
  - **MAX_SKEW_LEVEL:** Upper bound of SKEW_LEVEL to be analyzed by RT-CUDA Parameter Tunner. This should be less than or equal to the MAX_BLOCKSIZE.
  - **MIN_BLOCKSIZE:** Lower bound of BLOCKSIZE to be analyzed by RT-CUDA Parameter Tunner. This should be greater than or equal to 1 and less than or equal to the MAX_BLOCKSIZE.
  - **MIN_MERGE_LEVEL:** Lower bound of MERGE_LEVEL to be analyzed by RT-CUDA Parameter Tunner. This should be greater than or equal to 1 and less than or equal to the MAX_MERGE_LEVEL.

  - **MIN_SKEW_LEVEL:** Lower bound of SKEW_LEVEL to be analyzed by RT-CUDA Parameter Tunner. This should be greater than or equal to 1 and less than or equal to the MAX_SKEW_LEVEL.

Following is an example of a configuration file:

```
LOOP_COLLAPSING=1
BLOCK_SKEW=1
PREFETCHING=0
PREFETCHED_ARRAYS=A
NON_PREFETCHED_ARRAYS=B
DATA_TYPE=float
KERNEL_NAMES=matrix_scale
2DMATRIX=1
ROW_DIM=N
MAX_BLOCKSIZE=0
MAX_MERGE_LEVEL=8
MAX_SKEW_LEVEL=2
MIN_BLOCKSIZE=32
MIN_MERGE_LEVEL=1
MIN_SKEW_LEVEL=1
```

1.2.2 *Execution*

To run the compiler from the command line, go to the dist folder and type the following:
java -jar RTCUDATranslator.jar

1.2.3 *Output*

RT-CUDA generates following files in the output folder:

- **<kernelname>kernel.cu:** This file contains the converted CUDA kernels. Separate files for each kernel will be created. <kernelname> is the name of the converted CUDA kernel.
- **<kernelname>params.h:** This file contains the macro definition of kernel parameters. Separate files for each kernel will be created. <kernelname> is the name of the converted CUDA kernel.
- **main.cu:** This file contains the main program that calls CUDA kernels.
- **Header Files:** The compiler generates three header files params.h, and rcuda.h that are included in the main.cu.
- **Makefile:** For compilation with make program, it generates Makefile and the dependent findcudalib.mk files.

2 RT-CUDA EXAMPLES

2.1 Matrix-Matrix Multiplication

2.1.1 *Inputs*

Listing 1: kernel.c

```
void matrix_mul(float *C, float * restrict A, float *
    restrict B, int N)
{
    float sum=0.0;
    for(int i=0; i < N; i++)
        for(int j=0; j < N; j++){
                for(int k=0; k < N; k++){
                float b = B[k][j];
                        sum += A[i][k] * b;
            }
                C[i][j] = sum;
        }
}
```

Listing 2: main.c

```
int main(int argc, char *argv[])
{
    int N = 1024;
    int GPU = 0;
    if(argc > 1)
        N = atoi(argv[1]);
    if(argc > 2)
        GPU = atoi(argv[2]);

    cudaSetDevice(GPU);

    float *A, *B, *C;

    int memsize = N * N * sizeof(float);
    A = (float *)malloc(memsize);
    B = (float *)malloc(memsize);
    C = (float *)malloc(memsize);

    matrix_mul(C, A, B, N);

    free(A);
    free(B);
    free(C);

    exit(0);
}
```

Listing 3: config.txt

```
LOOP_COLLAPSING=1
BLOCK_SKEW=0
PREFETCHING=1
PREFETCHED_ARRAYS=A
NON_PREFETCHED_ARRAYS=B
DATA_TYPE=float
KERNEL_NAMES=matrix_mul
2DMATRIX=1
```

```
ROW_DIM=N
MAX_BLOCKSIZE=0
MAX_MERGE_LEVEL=0
MAX_SKEW_LEVEL=0
MIN_BLOCKSIZE=32
MIN_MERGE_LEVEL=1
MIN_SKEW_LEVEL=1
```

2.1.2 *Outputs*

Listing 4: kernel.cu

```
__global__ void matrix_mul(float *C, float *
    __restrict__ A, float * __restrict__ B, int N)
{
        float sum[MERGE_LEVEL];

        for(int i=0; i < MERGE_LEVEL; i++)
                sum[i] = 0.0;

        int tid = threadIdx.x;
        int bid = blockIdx.x;
        int ij = bid * BLOCKSIZE + tid;
        int i = (ij / N) * MERGE_LEVEL;
        int j = ij % N;

        __shared__ float As[MERGE_LEVEL][BLOCKSIZE];

        int k=0;
        for(int m=0; m < MERGE_LEVEL; m++)
                As[m][tid] = A[(i+m) * N + k + tid];
        __syncthreads();
        for(k = 0; k < N-BLOCKSIZE; k+=BLOCKSIZE){
                for(int t=0; t < BLOCKSIZE; t++){
                        float b = B[(k+t) * N + j];
                        for(int m = 0; m < MERGE_LEVEL
                            ; m++)
                                sum[m] += As[m][t] * b
                                    ;
                }
                __syncthreads();
                for(int m = 0; m < MERGE_LEVEL; m++)
                        As[m][tid] = A[(i+m) * N + (k+
                            BLOCKSIZE) + tid];
                __syncthreads();
        }
        for(int t=0; t < BLOCKSIZE; t++){
                float b = B[(k+t) * N + j];
                for(int m = 0; m < MERGE_LEVEL; m++)
                        sum[m] += As[m][t] * b;
        }
        for(int m = 0; m < MERGE_LEVEL; m++)
                C[(i + m) * N + j] = sum[m];
```

```
}
```

Listing 5: main.cu

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <cuda.h>
void checkCudaError(const char *msg)
{
        cudaError_t err = cudaGetLastError();
        if(cudaSuccess != err){
                printf("%s(%i) : CUDA error : %s : (%d
                    ) %s\n", __FILE__, __LINE__, msg,
                    (int)err, cudaGetErrorString(err))
                    ;
                exit (-1);
        }
}
#include "params.h"
#include "rcuda.h"
#include "kernel.cu"
int main(int argc,char *argv[]){
        int N=1024;
        int GPU=0;
        if(argc>1)N=atoi(argv[1]);

        if(argc>2)GPU=atoi(argv[2]);

        cudaSetDevice (GPU);

        float *A,*B,*C;
        int memsize=N*N*sizeof(float );
        cudaMallocManaged(&A,memsize);
        cudaMallocManaged(&B,memsize);
        cudaMallocManaged(&C,memsize);

        dim3 threads(BLOCKSIZE,1);
        dim3 grid(N*N/BLOCKSIZE/MERGE_LEVEL/SKEW_LEVEL
            ,1);
        matrix_mul<<<grid,threads>>>(C,A,B,N);
        cudaDeviceSynchronize();

        cudaFree (A);
        cudaFree (B);
        cudaFree (C);
        cudaThreadExit();
}
```

Listing 6: params.h

```
#define BLOCKSIZE 64
```

```
#define MERGE_LEVEL 16
#define SKEW_LEVEL 1
```

2.2 Dense Matrix Operators using RT-CUDA API

2.2.1 *Inputs*

Listing 7: main.c

```
int main(int argc, char *argv[])
{
    int N = 1024;
    int GPU = 0;
    if(argc > 1)
        N = atoi(argv[1]);
    if(argc > 2)
        GPU = atoi(argv[2]);

    cudaSetDevice(GPU);

    float *C, *A, *B, *X, *Y;

    int memsize = N * N * sizeof(float);
    int memsizevec = N * sizeof(float);
    C = (float *)malloc(memsize);
    A = (float *)malloc(memsize);
    B = (float *)malloc(memsize);
    X = (float *)malloc(memsizevec);
    Y = (float *)malloc(memsizevec);

    RTAPIInit();
    RTdSMM(C, A, B, N, N, N);
    RTdSMV(Y, A, X, N, N);
    RTdSMT(C, A, N, N);
    RTdSVV(C, X, Y, N);
    float result;
    RTdSDOT(X, Y, N, &result);
    RTAPIFinalize();

    free(C);
    free(A);
    free(B);
    free(X);
    free(Y);

    exit(0);
}
```

2.2.2 *Outputs*

Listing 8: main.cu

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <cuda.h>
void checkCudaError(const char *msg)
{
        cudaError_t err = cudaGetLastError();
        if(cudaSuccess != err){
                printf("%s(%i) : CUDA error : %s : (%d
                    ) %s\n", __FILE__, __LINE__, msg,
                    (int)err, cudaGetErrorString(err))
                    ;
                exit (-1);
        }
}
#include "params.h"
#include "rcuda.h"
#include "kernel.cu"
int main(int argc, char *argv[])
{
    int N = 1024;
    int GPU = 0;
    if(argc > 1)
        N = atoi(argv[1]);
    if(argc > 2)
        GPU = atoi(argv[2]);

    cudaSetDevice(GPU);

    float *C, *A, *B, *X, *Y;

    int memsize = N * N * sizeof(float);
    int memsizevec = N * sizeof(float);
    cudaMallocManaged(&C,memsize);
    cudaMallocManaged(&A,memsize);
    cudaMallocManaged(&B,memsize);
    cudaMallocManaged(&X,memsizevec);
    cudaMallocManaged(&Y,memsizevec);

    RTAPIInit();
    RTdSMM(C, A, B, N, N, N);
    RTdSMV(Y, A, X, N, N);
    RTdSMT(C, A, N, N);
    RTdSVV(C, X, Y, N);
    float result;
    RTdSDOT(X, Y, N, &result);
    RTAPIFinalize();

    cudaFree(C);
    cudaFree(A);
    cudaFree(B);
```

```
    cudaFree(X);
    cudaFree(Y);

    cudaThreadExit();
}
```

2.3 Sparse Matrix Operators using RT-CUDA API

2.3.1 *Inputs*

Listing 9: main.c

```
int main(int argc, char *argv[])
{
    int N = 1024;
    int GPU = 0;
    if(argc > 1)
        N = atoi(argv[1]);
    if(argc > 2)
        GPU = atoi(argv[2]);

    cudaSetDevice(GPU);

    float *DC, *X, *Y;

    int memsize = N * N * sizeof(float);
    int memsizevec = N * sizeof(float);
    X = (float *)malloc(memsizevec);
    Y = (float *)malloc(memsizevec);
    DC = (float *)malloc(memsize);

    RTAPIInit();
    RTspSArray *A, *B, *C;
    RTspSArrayLoadFromFile(argv[1], A);
    RTspSArrayLoadFromFile(argv[1], B);
    RTspSArrayCreate(DC, C, N, N);

    RTspSMM(C, A, B, N, N, N);
    RTspdSMM(C, A, B, N, N, N);
    RTspdSMM(C, A, B, N, N, N, RTBSR);
    RTspSMV(Y, A, X, N, N);
    RTspSMV(Y, A, X, N, N, RTBSR);
    RTspSMV(Y, A, X, N, N, RTCSR);
    RTspSArrayDestroy(A);
    RTspSArrayDestroy(B);
    RTspSArrayDestroy(C);
    RTAPIFinalize();

    free(X);
    free(Y);
    free(DC);

    exit(0);
```

```
}
```

2.3.2 *Outputs*

Listing 10: main.cu

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <cuda.h>
void checkCudaError(const char *msg)
{
        cudaError_t err = cudaGetLastError();
        if(cudaSuccess != err){
                printf("%s(%i) : CUDA error : %s : (%d
                    ) %s\n", __FILE__, __LINE__, msg,
                    (int)err, cudaGetErrorString(err))
                    ;
                exit (-1);
        }
}
#include "params.h"
#include "rcuda.h"
#include "kernel.cu"
int main(int argc, char *argv[])
{
    int N = 1024;
    int GPU = 0;
    if(argc > 1)
        N = atoi(argv[1]);
    if(argc > 2)
        GPU = atoi(argv[2]);

    cudaSetDevice(GPU);

    float *DC, *X, *Y;

    int memsize = N * N * sizeof(float);
    int memsizevec = N * sizeof(float);
    cudaMallocManaged(&X,memsizevec);
    cudaMallocManaged(&Y,memsizevec);
    cudaMallocManaged(&DC,memsize);

    RTAPIInit();
    RTspSArray *A, *B, *C;
    RTspSArrayLoadFromFile(argv[1], A);
    RTspSArrayLoadFromFile(argv[1], B);
    RTspSArrayCreate(DC, C, N, N);

    RTspSMM(C, A, B, N, N, N);
    RTspdSMM(C, A, B, N, N, N);
```

```
    RTspdSMM(C, A, B, N, N, N, RTBSR);
    RTspSMV(Y, A, X, N, N);
    RTspSMV(Y, A, X, N, N, RTBSR);
    RTspSMV(Y, A, X, N, N, RTCSR);
    RTspSArrayDestroy(A);
    RTspSArrayDestroy(B);
    RTspSArrayDestroy(C);
    RTAPIFinalize();

    cudaFree(X);
    cudaFree(Y);
    cudaFree(DC);

    cudaThreadExit();
}
```

2.4 Conjugate Gradient using RT-CUDA API and Custom Merge Operations

2.4.1 *Inputs*

Listing 11: subkernel.c

```
void sub(float *C, float * restrict A, float *
    restrict B, int N)
{
    for(int i=0; i<N; i++)
        C[i] = A[i] - B[i];
}
```

Listing 12: copykernel.c

```
void copy(float *C, float * restrict A, int N)
{
    for(int i=0; i<N; i++)
        C[i] = A[i];
}
```

Listing 13: scaleaddkernel.c

```
void scaleadd(float *C, float * restrict A, int N,
    float alpha)
{
    for(int i=0; i<N; i++)
        C[i] += alpha * A[i];
}
```

Listing 14: scalesubkernel.c

```
void scalesub(float *C, float * restrict A, int N,
    float alpha)
{
    for(int i=0; i<N; i++)
```

```
        C[i] -= alpha * A[i];
}
```

Listing 15: scaleaddstorekernel.c

```
void scaleaddstore(float *C, float * restrict A, float
    * restrict B, int N, float alpha)
{
    for(int i=0; i<N; i++)
        C[i] = alpha * A[i] + B[i];
}
```

Listing 16: main.c

```
int main(int argc, char *argv[])
{
    int N = 1024;
    int GPU = 0;
    if(argc > 1)
        N = atoi(argv[1]);
    if(argc > 2)
        GPU = atoi(argv[2]);

    cudaSetDevice(GPU);

    float *B, *X, *P, *R, *AP;

    int memsize = N * N * sizeof(float);
    int memsizevec = N * sizeof(float);
    X = (float *)malloc(memsizevec);
    B = (float *)malloc(memsizevec);
    P = (float *)malloc(memsizevec);
    R = (float *)malloc(memsizevec);
    AP = (float *)malloc(memsizevec);

    RTspSArray *A;
    RTspSArrayLoadFromFile(argv[1], A);

    RTspSMV(R, A, X, N, N);
    sub(R, B, R, N);
    copy(P, R, N);
    for(int k=0; k < 100; k++){
        float rr;
        float pp;
        RTdSDOT(R, R, N, &rr);
        RTspSMV(AP, A, P, N, N);
        RTdSDOT(P, AP, N, &pp);
        float alpha = rr / pp;
        scaleadd(X, P, N, alpha);
        scalesub(R, AP, N, alpha);
        float rrn;
        RTdSDOT(R, R, N, &rrn);
        if(rrn < 1e-10) break;
        float beta = rrn / rr;
```

```
        scaleaddstore(P, P, R, N, beta);
    }

    free(A);
    free(B);
    free(X);
    free(P);
    free(R);
    free(AP);

    exit(0);
}
```

Listing 17: subconfig.txt

```
LOOP_COLLAPSING=0
BLOCK_SKEW=0
PREFETCHING=0
PREFETCHED_ARRAYS=A
NON_PREFETCHED_ARRAYS=B,X,NX
DATA_TYPE=float
KERNEL_NAMES=sub
2DMATRIX=0
ROW_DIM=N
MAX_BLOCKSIZE=0
MAX_MERGE_LEVEL=0
MAX_SKEW_LEVEL=1
MIN_BLOCKSIZE=32
MIN_MERGE_LEVEL=1
MIN_SKEW_LEVEL=1
```

Listing 18: copyconfig.txt

```
LOOP_COLLAPSING=0
BLOCK_SKEW=0
PREFETCHING=0
PREFETCHED_ARRAYS=A
NON_PREFETCHED_ARRAYS=B,X,NX
DATA_TYPE=float
KERNEL_NAMES=copy
2DMATRIX=0
ROW_DIM=N
MAX_BLOCKSIZE=0
MAX_MERGE_LEVEL=0
MAX_SKEW_LEVEL=1
MIN_BLOCKSIZE=32
MIN_MERGE_LEVEL=1
MIN_SKEW_LEVEL=1
```

Listing 19: scaleaddconfig.txt

```
LOOP_COLLAPSING=0
BLOCK_SKEW=0
PREFETCHING=0
```

```
PREFETCHED_ARRAYS=A
NON_PREFETCHED_ARRAYS=B,X,NX
DATA_TYPE=float
KERNEL_NAMES=scaleadd
2DMATRIX=0
ROW_DIM=N
MAX_BLOCKSIZE=0
MAX_MERGE_LEVEL=0
MAX_SKEW_LEVEL=1
MIN_BLOCKSIZE=32
MIN_MERGE_LEVEL=1
MIN_SKEW_LEVEL=1
```

Listing 20: scalesubconfig.txt

```
LOOP_COLLAPSING=0
BLOCK_SKEW=0
PREFETCHING=0
PREFETCHED_ARRAYS=A
NON_PREFETCHED_ARRAYS=B,X,NX
DATA_TYPE=float
KERNEL_NAMES=scalesub
2DMATRIX=0
ROW_DIM=N
MAX_BLOCKSIZE=0
MAX_MERGE_LEVEL=0
MAX_SKEW_LEVEL=1
MIN_BLOCKSIZE=32
MIN_MERGE_LEVEL=1
MIN_SKEW_LEVEL=1
```

Listing 21: scaleaddstoreconfig.txt

```
LOOP_COLLAPSING=0
BLOCK_SKEW=0
PREFETCHING=0
PREFETCHED_ARRAYS=A
NON_PREFETCHED_ARRAYS=B,X,NX
DATA_TYPE=float
KERNEL_NAMES=scaleaddstore
2DMATRIX=0
ROW_DIM=N
MAX_BLOCKSIZE=0
MAX_MERGE_LEVEL=0
MAX_SKEW_LEVEL=1
MIN_BLOCKSIZE=32
MIN_MERGE_LEVEL=1
MIN_SKEW_LEVEL=1
```

Listing 22: global.txt

```
KERNEL_FILES=subkernel.c,copykernel.c,scaleaddkernel.c
    ,scalesubkernel.c,scaleaddstorekernel.c
```

```
CONFIG_FILES=subconfig.txt,copyconfig.txt,
    scaleaddconfig.txt,scalesubconfig.txt,
    scaleaddstoreconfig.txt
```

2.4.2 *Outputs*

Listing 23: subkernel.cu

```
__global__ void sub(float *C,float const *__restrict__
    A,float const *__restrict__ B,int N){
        int tid=threadIdx.x;
        int bid=blockIdx.x;
        int i=(bid*blockDim.x+tid)*subMERGE_LEVEL;
        for(int m=0;m<subMERGE_LEVEL;m++)
                C[(i+m)]=A[(i+m)]-B[(i+m)];

}
```

Listing 24: copykernel.cu

```
__global__ void copy(float *C,float const *
    __restrict__ A,int N){
        int tid=threadIdx.x;
        int bid=blockIdx.x;
        int i=(bid*blockDim.x+tid)*copyMERGE_LEVEL;
        for(int m=0;m<copyMERGE_LEVEL;m++)
                C[(i+m)]=A[(i+m)];

}
```

Listing 25: scaleaddkernel.cu

```
__global__ void scaleadd(float *C,float const *
    __restrict__ A,int N,float alpha){
  int tid=threadIdx.x;
  int bid=blockIdx.x;
  int i=(bid*blockDim.x+tid)*scaleaddMERGE_LEVEL;
  for(int m=0;m<scaleaddMERGE_LEVEL;m++)
     C[(i+m)]+=alpha*A[(i+m)];

}
```

Listing 26: scalesubkernel.cu

```
__global__ void scalesub(float *C,float const *
    __restrict__ A,int N,float alpha){
  int tid=threadIdx.x;
  int bid=blockIdx.x;
  int i=(bid*blockDim.x+tid)*scalesubMERGE_LEVEL;
  for(int m=0;m<scalesubMERGE_LEVEL;m++)
     C[(i+m)]-=alpha*A[(i+m)];

}
```

Listing 27: scaleaddstorekernel.cu

```
__global__ void scaleaddstore(float *C,float const *
    __restrict__ A,float const *__restrict__ B,int N,
    float alpha){
  int tid=threadIdx.x;
  int bid=blockIdx.x;
  int i=(bid*blockDim.x+tid)*scaleaddstoreMERGE_LEVEL;
  for(int m=0;m<scaleaddstoreMERGE_LEVEL;m++)
     C[(i+m)]=alpha*A[(i+m)]+B[(i+m)];

}
```

Listing 28: main.cu

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <cuda.h>
void checkCudaError(const char *msg)
{
        cudaError_t err = cudaGetLastError();
        if(cudaSuccess != err){
                printf("%s(%i) : CUDA error : %s : (%d
                    ) %s\n", __FILE__, __LINE__, msg,
                    (int)err, cudaGetErrorString(err))
                    ;
                exit (-1);
        }
}
#include "params.h"
#include "rcuda.h"
#include "subkernelparams.h"
#include "subkernel.cu"
#include "copykernelparams.h"
#include "copykernel.cu"
#include "scaleaddkernelparams.h"
#include "scaleaddkernel.cu"
#include "scalesubkernelparams.h"
#include "scalesubkernel.cu"
#include "scaleaddstorekernelparams.h"
#include "scaleaddstorekernel.cu"
int main(int argc, char *argv[])
{
    int N = 1024;
    int GPU = 0;
    if(argc > 1)
        N = atoi(argv[1]);
    if(argc > 2)
        GPU = atoi(argv[2]);

    cudaSetDevice(GPU);
```

```
    float *B, *X, *P, *R, *AP;

    int memsize = N * N * sizeof(float);
    int memsizevec = N * sizeof(float);
    cudaMallocManaged(&X,memsizevec);
    cudaMallocManaged(&B,memsizevec);
    cudaMallocManaged(&P,memsizevec);
    cudaMallocManaged(&R,memsizevec);
    cudaMallocManaged(&AP,memsizevec);

    RTspSArray *A;
    RTspSArrayLoadFromFile(argv[1], A);

    dim3 subthreads(subBLOCKSIZE, 1)
    dim3 subgrid(N/subBLOCKSIZE/subMERGE_LEVEL/
        subSKEW_LEVEL, 1);
    dim3 copythreads(copyBLOCKSIZE, 1)
    dim3 copygrid(N/copyBLOCKSIZE/copyMERGE_LEVEL/
        copySKEW_LEVEL, 1);
    dim3 scaleaddthreads(scaleaddBLOCKSIZE, 1)
    dim3 scaleaddgrid(N/scaleaddBLOCKSIZE/
        scaleaddMERGE_LEVEL/scaleaddSKEW_LEVEL, 1);
    dim3 scalesubthreads(scalesubBLOCKSIZE, 1)
    dim3 scalesubgrid(N/scalesubBLOCKSIZE/
        scalesubMERGE_LEVEL/scalesubSKEW_LEVEL, 1);
    dim3 scaleaddstorethreads(scaleaddstoreBLOCKSIZE,
        1)
    dim3 subgrid(N/scaleaddstoreBLOCKSIZE/
        scaleaddstoreMERGE_LEVEL/
        scaleaddstoreSKEW_LEVEL, 1);

    RTspSMV(R, A, X, N, N);
    sub<<<subgrid,subthreads>>>(R, B, R, N);
         cudaDeviceSynchronize();
    copy<<<copygrid,copythreads>>>(P, R, N);
         cudaDeviceSynchronize();
    for(int k=0; k < 100; k++){
         float rr;
         float pp;
         RTdSDOT(R, R, N, &rr);
         RTspSMV(AP, A, P, N, N);
         RTdSDOT(P, AP, N, &pp);
         float alpha = rr / pp;
         scaleadd<<<scaleaddgrid,scaleaddthreads>>>(X,
             P, N, alpha);
         cudaDeviceSynchronize();
         scalesub<<<scalesubgrid,scalesubthreads>>>(R,
             AP, N, alpha);
         cudaDeviceSynchronize();
         float rrn;
         RTdSDOT(R, R, N, &rrn);
         if(rrn < 1e-10) break;
         float beta = rrn / rr;
```

```
        scaleaddstore<<<scaleaddstoregrid,
            scaleaddstorethreads>>>(P, P, R, N, beta);
        cudaDeviceSynchronize();
    }

    cudaFree(A);
    cudaFree(B);
    cudaFree(X);
    cudaFree(P);
    cudaFree(R);
    cudaFree(AP);

    cudaThreadExit();
}
```

Listing 29: subkernelparams.h

```
#define subBLOCKSIZE 128
#define subMERGE_LEVEL 2
#define subSKEW_LEVEL 1
```

Listing 30: copykernelparams.h

```
#define copyBLOCKSIZE 512
#define copyMERGE_LEVEL 1
#define copySKEW_LEVEL 1
```

Listing 31: scaleaddkernelparams.h

```
#define scaleaddBLOCKSIZE 128
#define scaleaddMERGE_LEVEL 1
#define scaleaddSKEW_LEVEL 1
```

Listing 32: scalesubkernelparams.h

```
#define scalesubBLOCKSIZE 128
#define scalesubMERGE_LEVEL 1
#define scalesubSKEW_LEVEL 1
```

Listing 33: scaleaddstorekernelparams.h

```
#define scaleaddstoreBLOCKSIZE 512
#define scaleaddstoreMERGE_LEVEL 2
#define scaleaddstoreSKEW_LEVEL 1
```

* *Computer Engineering Department, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia*