Hand-Tuned SGEMM on GT200 GPU

Lung-Sheng Chien
Department of Mathematics, Tsing Hua university, R.O.C. (Taiwan)
d947207@oz.nthu.edu.tw
February 2010

Abstract: we improve SGEMM of Volkov's code [1]. $C = AB$ is tested on square matrices A, B and C with dimension N. On TeslaC1060 with CUDA 2.3, compared with Volkov's code for large N ($N > 1500$), we have
(1) N = multiple of 64: ~ 20% improvement,
(2) N = multiple of 16: > 15% improvement and
(3) N = multiple of 8: > 10% improvement.
Averagely speaking, we have 10% improvement for large N. As far as peak performance is concerned, our method reaches 440Gflop/s on TeslaC1060, which is 70% of peak performance of single precision without dual issue. Volkov's code reaches 346 Gflop/s, which is 55.45% of peak performance.

Figure 1 shows performance (Gflop/s) of method1on TeslaC1060, GTX285 and GTX295. The baseline is Volkov's code on TeslaC1060 (black dash line). Core frequency of GTX285 is 1.135x than that of TeslaC1060, and it is reasonable that performance of GTX285 is 1.165x than that of TeslaC1060. Peak performance of single precision without dual issue on TeslaC1060 is 624 Gflop/s, and maximum performance of method 1 on TeslaC1060 in Figure 1 is 439.467Gflop/s. Hence method 1 achieves 70% of peak performance without dual issue.

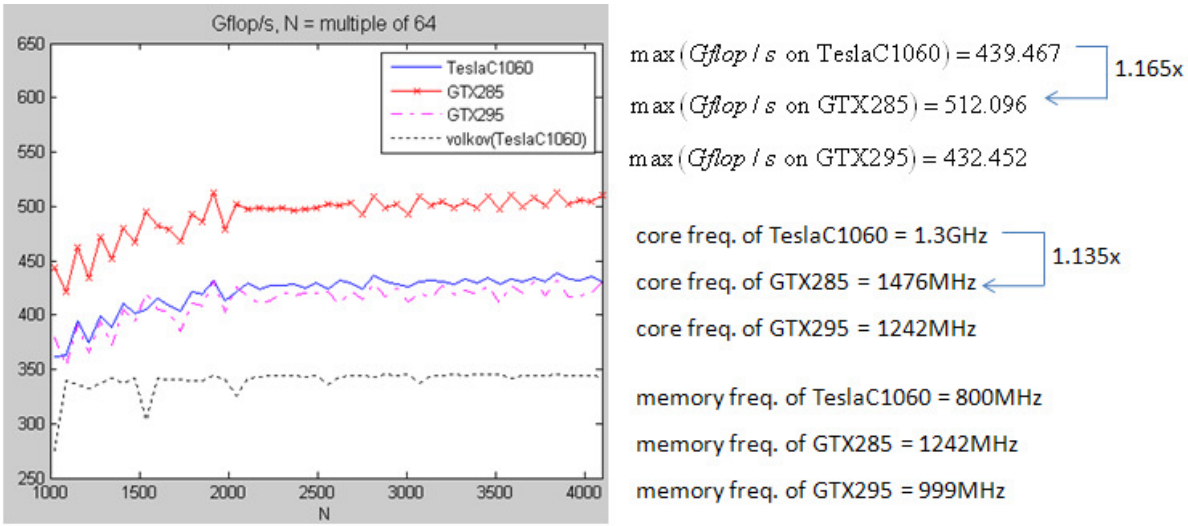

Figure 1: performance of method 1 over N = multiple of 64 on TeslaC1060, GTX285 and GTX295. The baseline is performance of Volkov's code on TeslaC1060.

1. File hierarchy

We propose eight methods and one Volkov's code [1]. Source code can be downloaded from http://oz.nthu.edu.tw/~d947207/NVIDIA/SGEMM/lsc_sgemm.zip. File hierarchy is shown in Figure 2. Source code of method x is put into directory method[x]. The directory "data" contains profiling data of each method on three GPUs, TeslaC1060, GTX285 and GTX295. The directory "matlab" contains .m file which can plot experimental result in directory "data".

Since we only provide binary code, each direction method[x] has a wrapper, which passes parameters of the kernel into shared memory. For example, directory "method1" has wrapper "`method1_DrvWrapper.cpp`".

Main source files and their description are listed in Table 1.

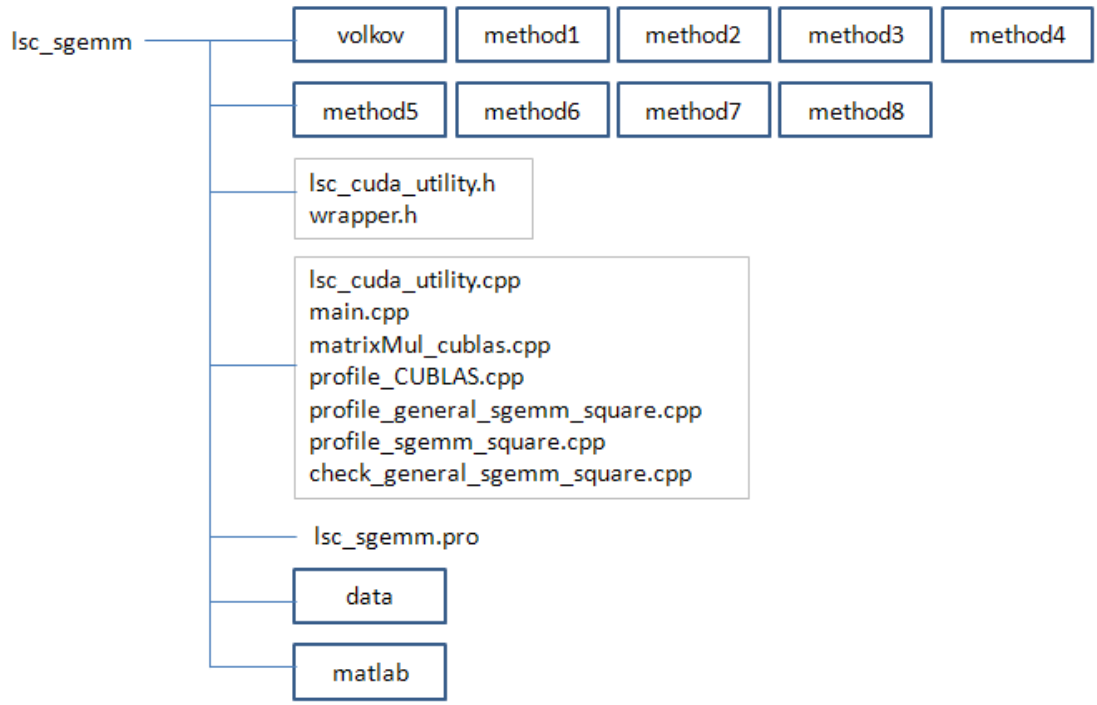

Figure 2: file hierarchy of source code

File name	functionality
matrixMul_cublas.cpp	A wrapper of CUBLAS, matrices A, B and C locate in host memory
profile_CUBLAS.cpp	Measure Gflop/s of CUBLAS 2.3 for $C = AB$ where A, B and C are square matrices
profile_sgemm_square.cpp	Measure Gflop/s of method 1 ~ 4 for $C = AB$ where A, B and C are square matrices. method 1 ~ 4 don't consider out-of-array bound, so allocation sequence in profile_sgemm_square.cpp is A, B, C such that invalid memory access of A and B will fall into memory block of C.
profile_general_sgemm_square.cpp	Measure Gflop/s of method 5 ~ 8 for $C = AB$ where A, B and C are square matrices. method 5 ~ 8 consider out-of-array bound, so allocation sequence is C,

	A, B such that illegal access of B would cause segmentation fault.
Check_general_sgemm_square.cpp	Run $C := \alpha AB + \beta C$ for $m,n = 5:257$ and $k = 5:129$ on method 5 ~ 8. This is consistent check since we only report performance $C = AB$ on square matrices. Method 5 ~ 8 must work for arbitrary dimension.

Table 1: main source files and their functionality.

2. grid information of each method

We show grid information of each method in Figure 3 ~ Figure 9, detailed information of each method is described in document **HandTunedSgemm_2010_v1.1.pdf.** These eight methods can be classified into two groups,

group 1: method 1, 2, 3, 4: without out-of-array bound checker

group 2: method 5, 6, 7, 8: with out-of-array bound checker

moreover as far as we are concerned grid structure, method 5 is the same as Volkov's code, method 6 is the same as method 1.

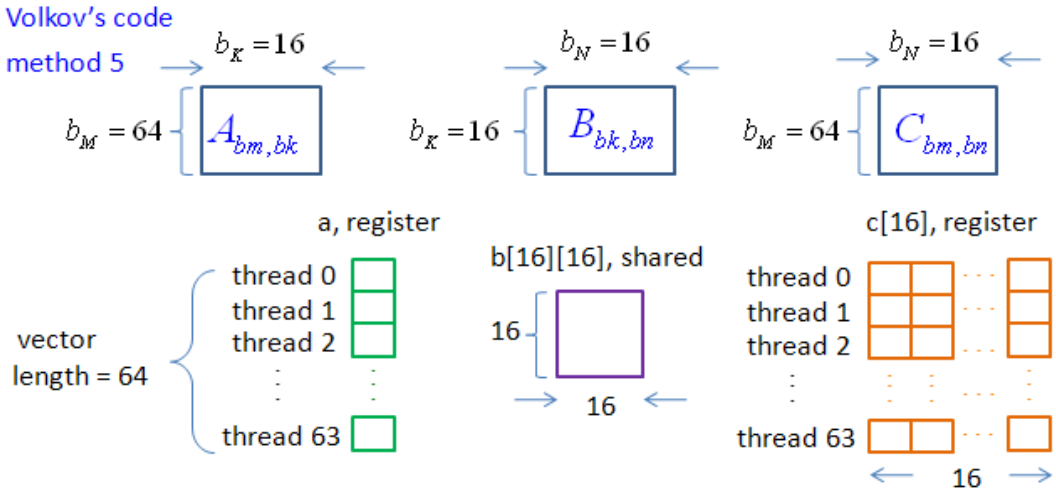

Figure 3: parameters of grid and block of Volkov's code.

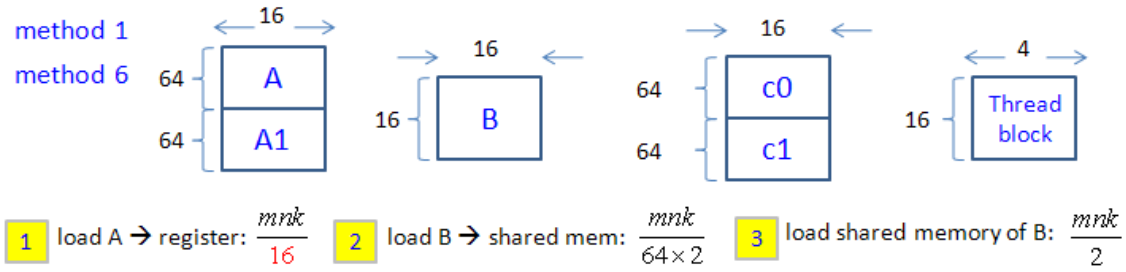

Figure 4: parameters of grid and block of method 1 and method 6.

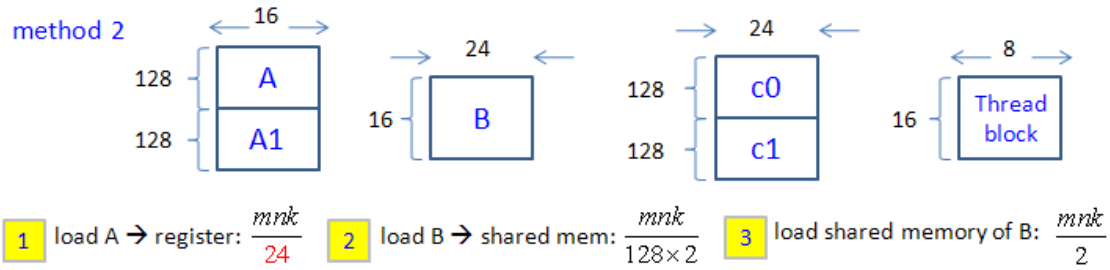

Figure 5: parameters of grid and block of method 2.

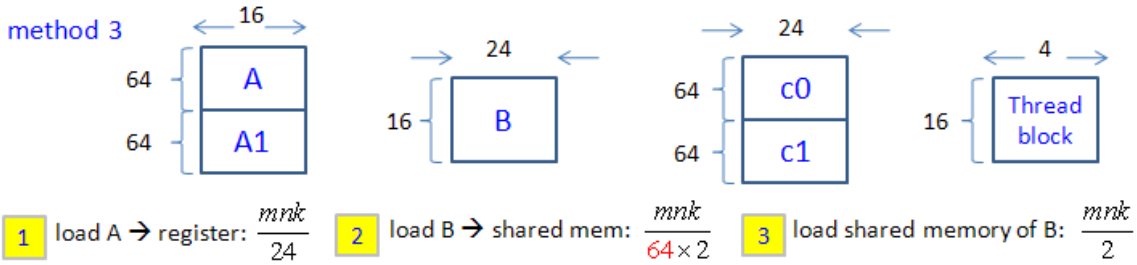

Figure 6: parameters of grid and block of method 3.

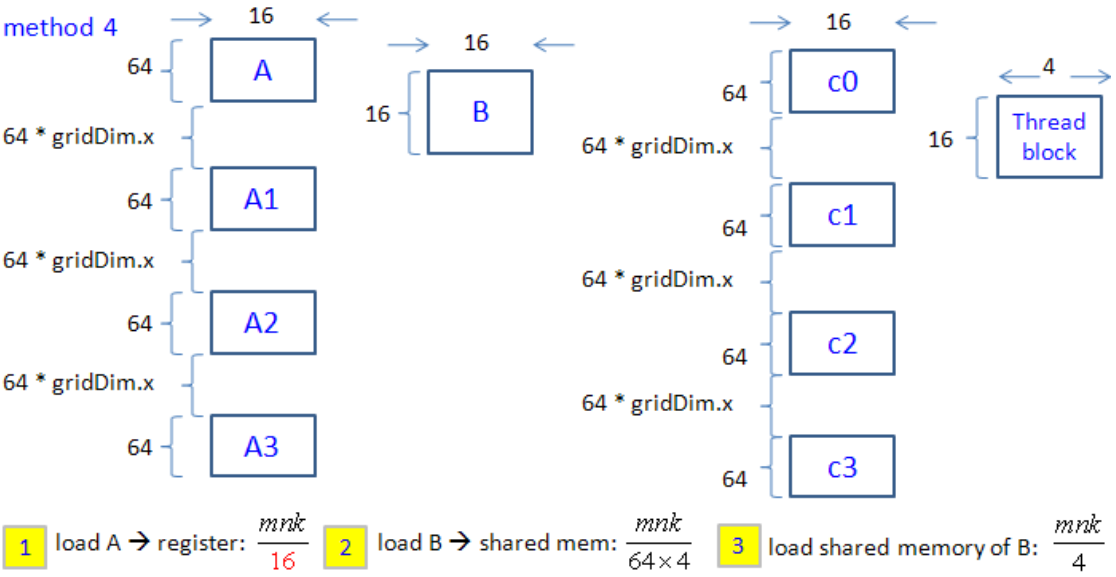

Figure 7: parameters of grid and block of method 4.

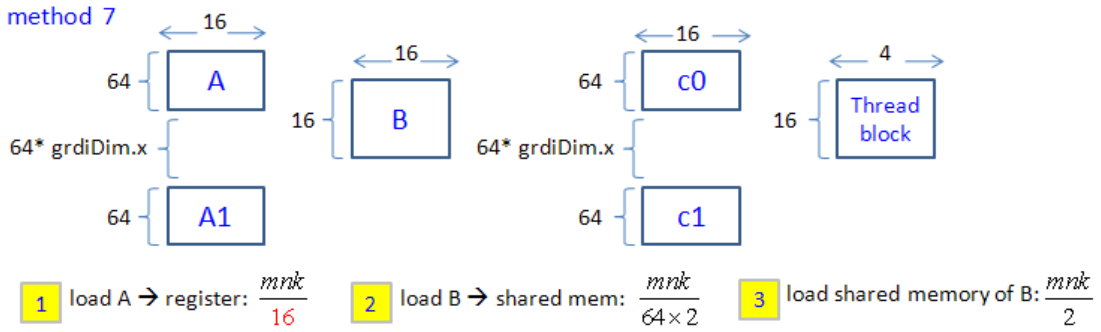

Figure 8: parameters of grid and block of method 7.

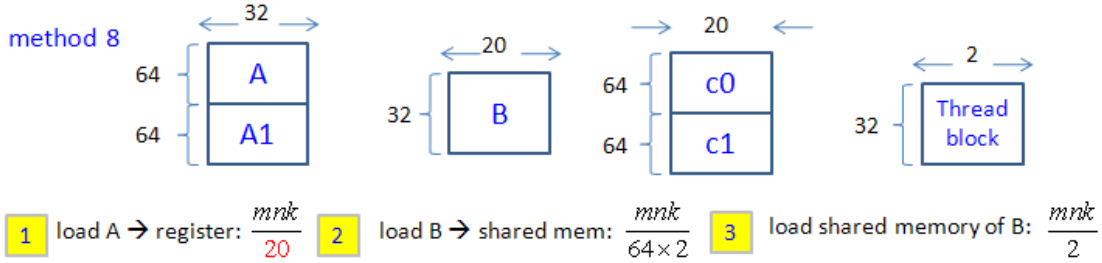

Figure 9: parameters of grid and block of method 8.

3. How to compile source code

We deliver binary codes and use driver API to load .cubin file into application. For example, directory "method1" contains method1.cubin and decuda_ldsb32_cudasm.cubin, which are binary code of method 1 and method1_variant respectively (please be careful that method1_variant.cubin is not correct, it is binary file of the trick "b_reg = b_ptr[j] * 4.0f", please see document for detailed description on method 1). We don't need to use nvcc to compile .cu file, all that you should do is to compile C++ source files by g++ , Intel C++ compiler, ... etc. Here we provide a project file "lsc_sgemm.pro" which can be read by qmake (make file generator of QT). You can use qmake to generate Makefile in unix system or VC2005 project file (remember to correct PATH of library in project file "lsc_sgemm.pro" before using qmake). If you want to write Makefile directly, then please include all source files listed in Figure 10.

```
HEADERS += lsc_cuda_utility.h wrapper.h
SOURCES += check_general_sgemm_suqare.cpp \
           check_sgemm_suqare.cpp \
           lsc_cuda_utility.cpp \
           main.cpp \
           matrixMul_cublas.cpp \
           profile_CUBLAS.cpp \
           profile_general_sgemm_suqare.cpp \
           profile_sgemm_suqare.cpp \
           method1/method1_DrvWrapper.cpp \
           method2/method2_DrvWrapper.cpp \
           method3/method3_DrvWrapper.cpp \
           method4/method4_DrvWrapper.cpp \
           method5/method5_DrvWrapper.cpp \
           method6/method6_DrvWrapper.cpp \
           method7/method7_DrvWrapper.cpp \
           method8/method8_DrvWrapper.cpp \
           volkov/volkov_DrvWrapper.cpp
```

Figure 10: source files and include files

4. How to profile all methods on specific GPU

Figure 11 shows driver in main.cpp. Only two parameters need modifying

(1) choose your target platform

Suppose you have GTX275 on device 0, (you can use function "show_device_info" to obtain this information or /SDK/deviceQuery), then set "device_num = 0"

(2) create directory /data/GTX275 to hold experimental data, and configure

#define OUTPUT_DIR "../data/GTX275/"

Before executing driver, please put executable file in directory "release" or you need to modify path of all methods in function "auto_profile".

```
#define  OUTPUT_DIR  "../data/TeslaC1060/"

int main(int argc, char** argv)
{
  int device_num = 2 ; // TeslaC1060
  initContext_Drv( device_num ) ; // initial context

// show_device_info() ; // show device information

  auto_profile( ) ; // calibrate Volkov's code and 8 methods

    return 0 ;
}
```

Figure 11: driver in main.cpp. Choose target platform via "device_num" and output directory via macro "OUTPUT_DIR"

5. How to plot experimental result as we see in technical report

We provide MATLAB solution here. You only need M-file /matlab/profile_sgemm.m and modify parameters passing into function "profile_sgemm_pair". "profile_sgemm_pair" compares reference model and experimental model given by caller and plot four figures

(1) Gflop/s of both models

(2) performance improvement $R = \frac{\text{time of experimental model}}{\text{time of reference model}} = \frac{\text{Gflop/s of reference model}}{\text{Gflop/s of experimental model}}$

(3) $R(N = \text{multiple of } 8)$

(4) $R(N = \text{multiple of } 64)$

"profile_sgemm_pair" has four input parameters

(1) data file of reference model

(2) model name of reference model, this name would be displayed in figure

(3) data file of experimental model

(4) model name of experimental model

Example: reference model is Volkov's code and experimental model is method1_variant, platform is GTX285, then

```
profile_sgemm_pair( '../data/GTX285/volkov/threads512.txt', 'volkov',
                    '../data/GTX285/method1/variant_threads320.txt' , 'method1(variant)') ;
```

Moreover one can use M-file, compare_GT200.m to plot last figure in document.

References

[1] Vasily Volkov, James W. Demmel, Benchmarking GPUs to Tune Dense Linear Algebra. In SC '08: Preceedings of the 2008 ACM/IEEE conference on Supercomputing. Piscataway, NJ, USA, 2008, IEEE Press. source code can be downloaded from NVIDIA forum, http://forums.nvidia.com/index.php?showtopic=89084