# Hand-Tuned CGEMM on GT200 GPU

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

**Abstract:** we extend the idea in SGEMM [1,2] to CGEMM (matrix multiplication on complex type) and reach 445.7Gflop/s whereas CUBALS reaches 277.7Gflop/s, we have 37.69% improvement.

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 1.155x than that of TeslaC1060.
If we compare method 1 with CUBALS, then we have 37.69% improvement because CUBALS only reaches 277.7Gflop/s.

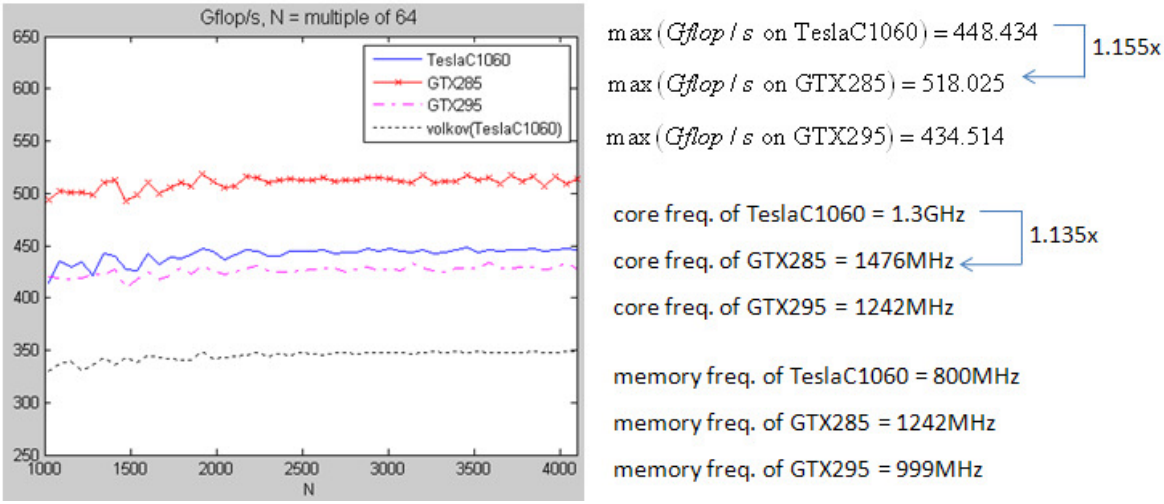


Figure 1: performance of method 1 over $N$ = multiple of 64 on TeslaC1060, GTX285 and GTX295. The baseline is performance of Volkov's code on TeslaC1060.

## 1. File hierarchy

We propose four methods and one Volkov's code [1]. Source code can be downloaded from http://oz.nthu.edu.tw/~d947207/NVIDIA/CGEMM/lsc_cgemm_v2.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.

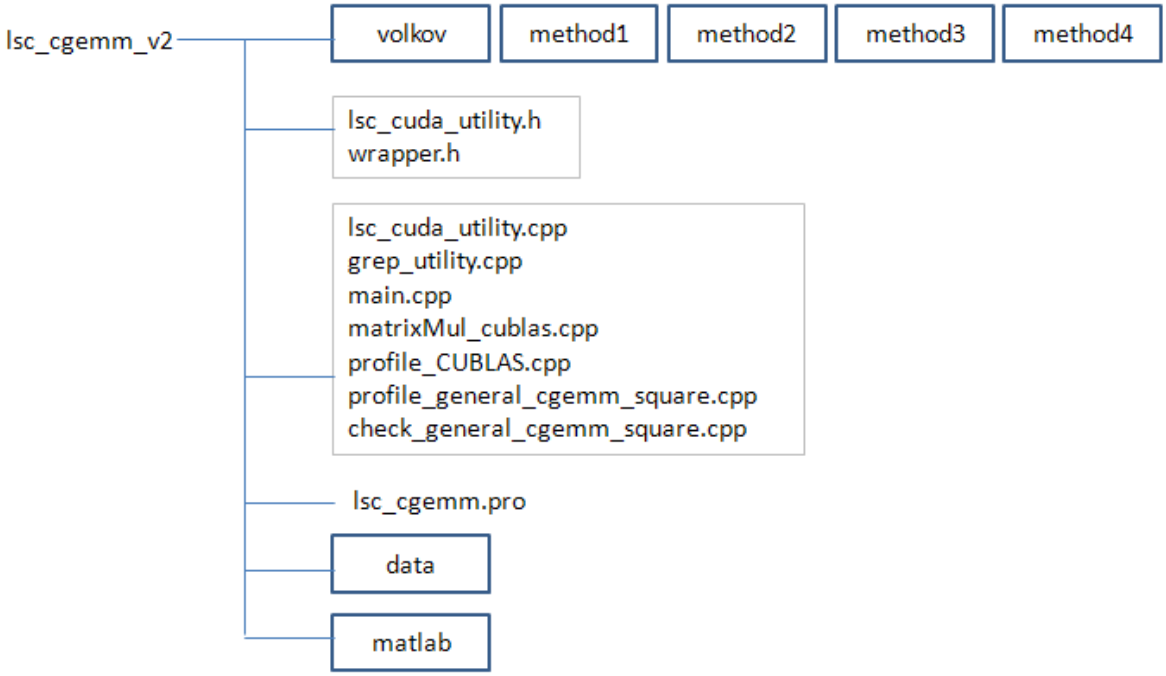


Figure 2: file hierarchy of source code

| 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_general_sgemm_square.cpp | Measure Gflop/s for $C = AB$ where A, B and C are square matrices. All methods consider out-of-array bound. |
| check_general_sgemm_square.cpp | Run $C := \alpha AB + \beta C$ for $m, n = 5:257$ and $k = 5:129$ on all method. |
| /method1/rank1_update_method1.cpp | Generate pattern of rank-1 update,<br>set.le.s32 $p0\|$o127, $r38, c1[0x0008]<br>@$p0.ne bra.label label5<br>rank-1 update<br>bar.sync.u32 0x00000000<br>add.b32 $r38, $r38, 0xfffffff0 |
| grep_utility.cpp | Remove blanks in .cubin file in order to do comparison of two .cubin files via UNIX command "diff" |

Table 1: main source files and their functionality.

## 2. grid information of each method

We show grid information of each method in Figure 3, detailed information of each method is described in document **HandTunedCgemm_2010_v1.pdf.** All (five) methods work for arbitrary dimension (out-of-array bound is considered).

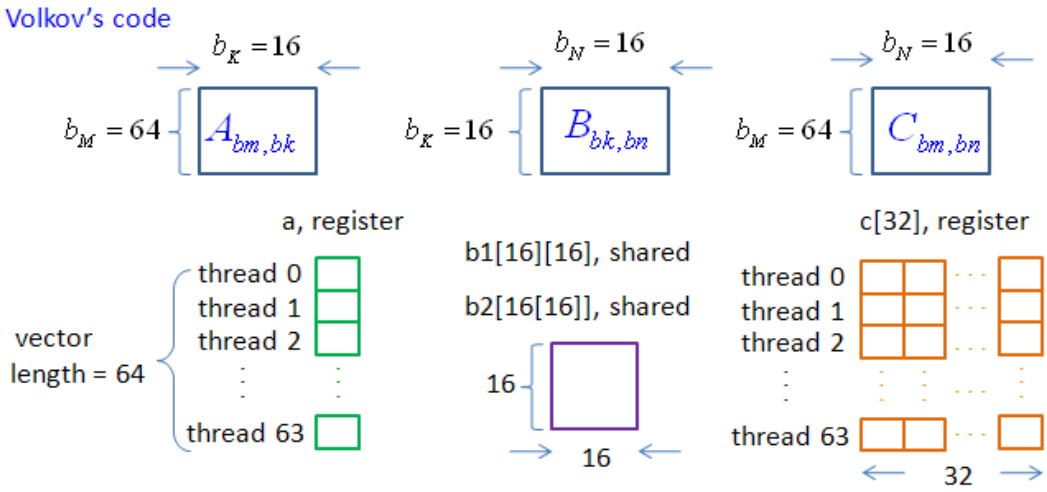


Figure 3: parameters of grid and block of Volkov's code.

## 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. 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_cgemm_v2.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_cgemm_v2.pro" before using qmake). If you want to write Makefile directly, then please include all source files listed in Figure 4.

```
HEADERS += lsc_cuda_utility.h wrapper.h
SOURCES += check_general_cgemm_suqare.cpp \
           grep_utility.cpp \
           lsc_cuda_utility.cpp \
           main.cpp \
           matrixMul_cublas.cpp \
           profile_CUBLAS.cpp \
           profile_general_cgemm_suqare.cpp \
           method1/rank1_update_method1.cpp \
           method1_variant/rank1_update_method1_variant.cpp \
           method2/rank1_update_method2.cpp \
           method2_variant/rank1_update_method2_variant.cpp \
           method3/rank1_update_method3.cpp \
           method3_variant/rank1_update_method3_variant.cpp \
           method4/rank1_update_method4.cpp \
           volkov/volkov_DrvWrapper.cpp \
           volkov_unroll1/rank1_update_volkov_unroll1.cpp \
           volkov_unroll2/rank1_update_volkov_unroll2.cpp \
           volkov_unroll4/rank1_update_volkov_unroll4.cpp \
           volkov_unroll8/rank1_update_volkov_unroll8.cpp
```

Figure 4: source files and include files

## 4. How to profile all methods on specific GPU

Figure 5 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 4 methods

//  check_general_cgemm_square( "../method1/method1.cubin", "method1", &volkov_DrvWrapper )

    return 0 ;
}
```

Figure 5: 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_cgemm.m and modify parameters passing into function "profile_cgemm_pair". "profile_cgemm_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_cgemm_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_cgemm_pair( '../data/GTX285/volkov/threads256.txt', 'volkov',
                    '../data/GTX285/method1/threads320.txt' , 'method 1') ;
```

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

[2]Lung-Sheng Chien, Hand-Tuned SGEMM on GT200 GPU, http://forums.nvidia.com/index.php?showtopic=159033