ASPEN-K2: Automatic-tuning and Stabilization for the Performance of CUDA BLAS Level 2 Kernels

Toshiyuki IMAMURA

The University of Electro-Communications, JAPAN

CREST JST
CUDA-BLAS
Linear Algebra Software on GPGPU

- Emerging HW GPU accelerates Linear Algebra software.
  - CULA developed by EM Photonics
    - CUDA version LAPACK. Referred from NVIDIA
    - Commercial based, Not free
    - Most of the major functions are implemented
  - MAGMA developed by U. Tennessee
    - Version 1.1 is now available
    - Some of LAPACK routines are implemented, and performs very well
    - Some Reports on SC series or other workshops
    - Coupling with PLASMA by Quark technology

- They run excellently fast, and reduce calculation cost.
- They basically rely on CUDA-BLAS performance
CUDA-BLAS

- Many implementations
  - CUBLAS
    - CUDA 3.1, 3.2, 4.0. Higher version performs more stably.
  - MAGMABLAS
    - MAGMAv1.1, faster than CUBLAS.
  - CULABLAS
    - CULA
  - GLAS
    - Developed by GPUlab at DTU (Tech. Univ. of Denmark). Auto-tuned CUDA-BLAS. Some of Level 1 and Level2 kernels are implemented.
  - ASPEN.K2
    - Developed by T.I and others. Auto-tuned CUDA-BLAS, xGEMV, xSYMV are implemented.
CUBLAS 3.x and 4.0 show higher performance but peaky. They are not optimized enough.

- In particular, Level 2 kernels (xGEMV-T, xSYMV, etc)

- Can we stabilize and improve performance more?

---

**DGEMV-N on a TeslaC2050**

- Magmabl 1.1
- CUBLAS 3.1
- CUBLAS 3.2
- CUBLAS 4.0

**DGEMV-T on a TeslaC2050**

- Magmabl 1.1
- CUBLAS 3.1
- CUBLAS 3.2
- CUBLAS 4.0
Outline

- **CUDA-BLAS**
  - Higher Performance
  - Peaky or Stable?

- **Automatic-tuning for CUDA-BLAS**
  - Analysis of kernel codes
  - What is the performance parameters?

- **ASPEN.K2**
  - Automatic Tuning Framework
  - Performance on a GTX580

- **Conclusion and Future works**
AUTOMATIC-TUNING FOR CUDA-BLAS
Automatic-Tuning for CUDA BLAS

Typical example for Level 2 kernel, DGEMV

\[ y := \alpha A^{\text{opt}} x + \beta y \]

```c
__shared__ double buff[BLOCK_SIZE];
A += threadIdx.x+blockIdx.x*BLOCk_SIZE;
x += threadIdx.x;

for(i=0; i<n_col; i += BLOCK_SIZE){
    __syncthreads();
    buff[threadIdx.x] = x[i];
    __syncthreads();
#pragma unroll
    for(int j=0; j<BLOCK_SIZE ; j++){
        res += A[0]*buff[j];
        A += lda;
    }
}
```

Performance on a GTX280
**Code Analysis**

- **DGEMV-N : MAGMABLAS ver1.1**

  ```c
  __shared__ double buff[BLOCK_SIZE];
  A += threadIdx.x + blockIdx.x * BLOCK_SIZE;
  x += (tid = threadIdx.x);
  
  for (i = 0; i < n_col; i += BLOCK_SIZE) {
    syncthreads(); buff[tid] = x[i]; syncthreads();
    for (int j = 0; j < BLOCK_SIZE; j++) {
      res += A[0] * buff[j]; A += lda;
    }
  }
  ```

- **BLOCK_SIZE is performance parameter.**

  On MAGMA, BLOCK_SIZE is fixed 64.
  → We can generate kernel codes with other value of BLOCK_SIZE.
DGEMV-N : ASPEN.K2 ver 0.1

```c
__shared__ double buff[BLOCK_SIZE];
A += threadIdx.x+blockIdx.x*BLOC K_Size;
A += threadIdx.y*BASE;
x += (tid=threadIdx.x+threadIdx.y*BLOC K_Size);

for(i=0; i<n_co/BASE; i += BLOCK_SIZE*BASE){
    syncthreads(); buff[tid] = x[i]; syncthreads();
    for(int j=0; j<BLOCK_SIZE*BASE; j+=BASE){
        res += A[0]*buff[j]; A += lda*BASE;
    }
} Sumup (res);
```

- **BLOCK_SIZE** and **BASE** are performance parameters.
  Many threads can work, thus performance is expected to improve well in case of smaller dimension matrices.
- xGEMV-T and xSYMV are optimized in a similar way.
Code Modification in ASPEN.K2

- **DGEMV-T : ASPEN.K2 ver 0.1**

```c
#define BASE 4
__shared__ double w[THREAD_SIZE];
col_m = (n_col/BLOCK_SIZE)*BLOCK_SIZE;
row = blockIdx.x* BASE + threadIdx.y*VMAX;
A += threadIdx.x; x += threadIdx.x;
ak = A + row*lda;
#pragma unroll 8
for(int i=0; i<col_m; i+=BLOCK_SIZE){
    s0 += ak[0]*x[i];
    s1 += ak[lda]*x[i];
    s2 += ak[lda*2]*x[i];
    s3 += ak[lda*3]*x[i];
    ak+=BLOCK_SIZE;
}
{ int i=col_m; if(i+threadIdx.x<n2){
    s0 += ak[0]*x[i];
    s1 += ak[lda]*x[i];
    s2 += ak[lda*2]*x[i];
    s3 += ak[lda*3]*x[i];
}}
sumup(&s0,w); sumup(&s1, w);
sumup(&s2,w); sumup(&s3, w);

- **BLOCK_SIZE, BASE and VMAX are performance parameters on xGEMV-T**
On a GTX280, periodic behavior is observed.

$\text{Period} = \text{BLOCK\_SIZE} \times 10$

This comes from increment of the thread-block assignment with a TPC grouping.

Resources are assigned in a round robin fashion.

$$P = \frac{2N^2}{o_0 + \sum (o_i + \alpha_i N)[N/\beta_i]}$$

Primary mode approximation,

$$P = \frac{2N^2}{o_0 + (o_1 + \alpha_1 N)[N/\beta_1]} \sim \frac{2N}{\alpha_1 [N/\beta_1]} \left(1 - \frac{o}{\alpha_1 N [N/\beta_1]}\right)$$

N tasks are divided and assigned into $\beta_i$ resources.
Automatic Stabilization and Performance Tuning for CUDA BLAS, focused on level 2 Kernel, the current version is 0.1

ASPNEN.K2
Automatic Tuning Framework

- The idea of AT Framework on ASPEN.K2

- Parameter script writer, and auto-code generation
- Auto-sampling at 1st stage
- Auto-screening with ranking point scheme
- Auto-sampling at 2nd stage
- Fitting with non-linear function by least square
  \[ f(x) = \frac{aN^2}{o + N^2} \]
- Auto-code generation and restructuring
The idea of AT Framework on ASPEN.K2

- Parameter script writer, and auto-code generation
- Auto-sampling at 1st stage
- Auto-screening with ranking point scheme
- Auto-sampling at 2nd stage
- Auto-code generation and restructuring

Code generation by using template meta programming of CUDA's C++ property

```cpp
template < int BLOCK_SIZE, int BASE>
__global__ void DGEMV_N ( ...) 
{
    ...
}
```

```cpp
switch (kid) {
    case 64: DGEMV_N < 64, 1 > ( ... ); break;
    case 96: DGEMV_N < 96, 3 > ( ... ); break;
    ...
}
```
The idea of AT Framework on ASPEN.K2

- Parameter script writer, and auto-code generation
- Auto-sampling at 1st stage
- Auto-screening with ranking point scheme
- Auto-sampling at 2nd stage
- Auto-code generation and restructuring

Sample the data in **Brute force** way but at small number of points like 1000, 2000, 3000, .... Here, the number of kernels candidate is too large. To pack into a single object is too difficult, so multiple and divided compiling is automatically done. It takes an hour or more.
Automatic Tuning Framework

The idea of AT Framework on ASPEN.K2

- Parameter script writer, and auto-code generation
- Auto-sampling at 1st stage
- Auto-screening with ranking point scheme
- Auto-sampling at 2nd stage
- Fitting with non-linear function by least square
  \[ f(x) = \frac{aN^2}{o + N^2} \]
- Auto-code generation

Kernel candidates are selected by ranking point scheme.
Automatic Tuning Framework

- The idea of AT Framework on ASPEN.K2

Parameter script writer, and auto-code generation

Auto-sampling at 1st stage

Auto-screening with ranking point scheme

Sample the data at many points, 100, 200, 300, ..., 10000 in order to fit cost functions accurately.

Auto-sampling at 2nd stage

Auto-code generation and restructuring

Fitting with non-linear function by least square

\[ f(x) = \frac{aN^2}{o + N^2} \]
Automatic Tuning Framework

The idea of AT Framework on ASPEN.K2

- Parameter script writer, and auto-code generation
- Auto-sampling at 1st stage
- Auto-screening with ranking point scheme
- Auto-sampling at 2nd stage
- Fitting with non-linear function by least square

Define cost functions by using GNU fit functionality plugged in gnuplot.
Cost function model is given not automatically but empirically at the moment.
Automatic Tuning Framework

- The idea of AT Framework on ASPEN.K2

- Parameter script writer, and auto-code generation
- Auto-sampling at 1st stage
- Auto-sampling at 2nd stage
- Fitting with non-linear function by least square
- Auto-code generation and restructuring

Generate an automatic-code-selection function which evaluates the cost functions derived at the previous step.

Usage example,

```python
kid = get_optimul_kernel_id(N);
do_kernel ( kid, N, ... );
```
ASPEN.K2 Performance

- **GPGPU environment**
  - CUDA 4.0
  - NVCC option: `--compiler-options -fno-strict-aliasing -DUNIX -O3 -- ptxas-options=-v -gencode arch=compute_20,code=compute_20 - maxrregcount=64`
  - Compiler: gcc version 4.4.6
  - GPU: GTX580 1.54GHz, 512 cores (GRAM1.5GB, 2GHz)
  ```bash
  % numactl --physcpubind=0 ./exe
  ```

- **Target BLAS kernels and parameters to be tuned up**
  - DGEMV-N < BLOCK_SIZE, BASE >
  - DGEMV-T < BLOCK_SIZE, VMAX, BASE, unroll >
  - DSYMV-U < BLOCK_SIZE, BASE, MULTIPLICITY >

  Non-thunking mode, i.e. without host ↔ device data transfer
DGEMV-N on a GTX580

192.4GB/s = 48.1 GFLOPS, 42/48.1 = 0.87

ASPEN.K2 0.1
CUBLAS 3.2
MAGMA1.1
CUBLAS 4.0
CUBLAS 3.1

192.4 GB/s = 48.1 GFLOPS

192.4 GB/s = 48.1 GFLOPS

[Matrix Dimension]

[GFLOPS]
Example, cost functions obtained by fitting
DGEMV-T on a GTX580

192.4GB/s = 48.1GFLOPS, \( \frac{42}{48.1} = 0.87 \)

[Graph showing performance comparison between ASPEN.K2 0.1, CUBLAS 4.0, CUBLAS 3.2, CUBLAS 3.1, and MAGMA 1.1. The x-axis represents matrix dimension, and the y-axis represents GFLOPS.]
DSYMV-U on a GTX580

![Graph showing performance comparison between DSYMV-U and various libraries on a GTX580.](image)

- ASPEN.K2 0.1
- MAGMA 1.1
- CUBLAS 3.2
- CUBLAS 4.0
- CUBLAS 3.1

[Matrix Dimension]

[GFLOPS]
Summary and discussion

More stable and higher performance obtained

- DGEMV-N : Higher, but still jaggy slightly.
- DGEMV-T : Stable and Better at most dimensions.
  Both performance attain 42 GFLOPS, equivalent to 87% usage of memory bandwidth on a GTX580.

- Additional result : DSYMV-U
  Higher performance than magmablas and cublas.
  - Sorry, more details about algorithm and implementation will be presented in other GPU or HPC workshop.
CONCLUSION AND FUTURE WORK
Conclusion

- **ASPEN.K2**
  - Stable and Higher performance CUDA-BLAS library with automatic-tuning
  - ASPEN Auto-tuning framework is automated except performance modeling and writing candidate kernel templates.
  - The current version performs excellently, faster than major CUDA BLAS’s, magmablas, cublas and culablas. Sustained performance of DGEMV-[NT] is 42 GFLOPS on a GTX580, equivalent to 87% usage of bandwidth.

- **Future works**
  - Complete Level 2 kernels
    - xTRMV, xGER, xSYR, xSYR2, etc.
  - Multiple-GPU version on a GTX590 or SLI
  - Open Source publicity
THANK YOU
Performance on a Tesla C2050

ADDITIONAL SLIDES
DGEMV-N on a C2050

![Graph showing performance of DGEMV-N on a C2050 with different libraries.]

- **ASPN.K2 0.1**
- **CUBLAS 3.2**
- **MAGMA1.1**
- **CUBLAS 4.0**
- **CUBLAS 3.1**
DGEMV-N on a C2050

![Graph showing performance comparison of DGEMV-N on a C2050 with different libraries: ASPEN.K2 0.1, MAGMA 1.1, CUBLAS 4.0, CUBLAS 3.2, and CUBLAS 3.1.](image-url)
DSYMV-U on a C2050

[Graph showing performance comparison between different libraries for DSYMV-U on a C2050.]

- CUBLAS 4.0
- CUBLAS 3.2
- MAGMA 1.1
- ASPEN.K2 0.1

[Legend for different libraries and data points is visible on the graph.]