Session for Autotuning :
Case studies and activities from Japan AT community

9th JLESC meeting at ICL, 15-17 April 2019, Knoxville, TN, US

Toshiyuki Imamura (RIKEN CCS)
Incremental approximation property of d-Spline curve

**Idea:**
- Choose initial sample data randomly.
- Calculate d-Spline curve.
- Find the maximum point, then if the parameter is not included in the parameter set sampled already, sample on it. Otherwise, find a parameter to be sampled by the following rule.
  1. neighborhood point which was not sampled
  2. Second-order differential is largest and not yet sampled.
Repeat until no effective change is observed.

*We expect less than 5 percent of the number of sampling points!!*
SA-AMG and ATMathCoreLib

- V-cycle SA-AMG-CG solver
  It needs parameter values at each level. The total parameter combination is 54000.

- ATMathCoreLib
  Provides a sub-optimal sequential experimental design for on-line automatic tuning.
  \[ x_t = E_z(f(t, z)) \]
  \[ t_{opt} = \text{argmin}\{x_t | t \in T\} \]

Bayesian formulation

\[ \pi_{tk+1}(x_t) \propto P(x | x_t)\pi_{tk}(x_t) \]

\[ \xi_{i,k} + 1(x_{t_i}, k_i + 1) \]

be an average of

\[ \pi_{t_i, k_i + 1}(x | x_{t_i}, k_i + 1) \]

\[ i = \text{argmin}_{i | t_i \in T}\{w_i\}, \]

\[ w_i = \int \text{min}\{\xi_0, \xi_i, k_i + 1(x)\} \pi_{t_i, k_i}(x) dx \]
AT case study of CUDA SYMV kernel
Case study for ASPEN.K2 SYMV kernels

- Tesla C2050 (CUDA4.0)
- GeForce GTX580 (CUDA5.0)
- Tesla K20c (CUDA5.0)

- Overhead of Sampling (=cost)
  - For (BLOCK_SIZE, UX, MULTIPLICITY, MX), the force sampling was done. It took 10 to 24 hours. Compiling cost is quite dominant.

<table>
<thead>
<tr>
<th></th>
<th>C2050</th>
<th>GTX580</th>
<th>K20c</th>
</tr>
</thead>
<tbody>
<tr>
<td>N. of Parameters</td>
<td>6646</td>
<td>7276</td>
<td>7200</td>
</tr>
<tr>
<td>Sampling time</td>
<td>23:09 → 1Hr</td>
<td>9:59 → 30m</td>
<td>10:37 → 30m</td>
</tr>
<tr>
<td>Ave. time/sample</td>
<td>6 sec</td>
<td>5 sec</td>
<td>5 sec</td>
</tr>
</tbody>
</table>
Introduction

- Rapid evolutions of HPC market result in a strong demand of software’s catch-up

For Example : GPU

<table>
<thead>
<tr>
<th></th>
<th>Tesla C2075 (Fermi)</th>
<th>GTX580 (Fermi)</th>
<th>Tesla K20c (Kepler, GK110)</th>
<th>Titan-V (Volta, GV100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA cores</td>
<td>448</td>
<td>512</td>
<td>2496</td>
<td>5120</td>
</tr>
<tr>
<td>Memory</td>
<td>6GB GDDR5, 144[GB/s]</td>
<td>1.5GB GDDR5, 192[GB/s]</td>
<td>5GB GDDR5, 208[GB/s]</td>
<td>12GB HBM2, 652.8[GB/s]</td>
</tr>
<tr>
<td>Performance</td>
<td>515GFLOPS(DP)</td>
<td>790 GFLOPS(DP)</td>
<td>1.17TFLOPS(DP)</td>
<td>7.45TFLOPS(DP)</td>
</tr>
<tr>
<td></td>
<td>1.03TFLOPS(SP)</td>
<td>1.58TFLOPS(SP)</td>
<td>3.52TFLOPS(SP)</td>
<td>14.9TFLOPS(SP)</td>
</tr>
</tbody>
</table>

Photos are from http://www.elsa-jp.co.jp/products/.
ASPEN.K2 = Level 2 CUDA BLAS implementation with Automatic performance tuning

Level2 BLAS :: GEMV, SYMV, ...

\[ y = \alpha Ax + \beta y, \quad A^T = A \]

- Limited version of matrix-vector product routine on which A is assumed to a symmetric matrix
- It is categorized in Level2 BLAS, and its performance bounds by memory bandwidth.
- Roughly, upper limit can be estimated by the sustained mem.band*4/8[FLOPS]
- Loss of performance leads low performance on the eigenvalue solver on GPU’s \( \leftrightarrow \) my regular work.

Algorithm of SYMV

\begin{verbatim}
do i=1,n  
do j=1,i-1  
    y(i) = y(i) + a(j,i) * x(j)  
    w(j) = w(j) + a(j,i) * x(i)  
enddo  
y(i) = y(i) + a(i,i) * x(i)  
enddo  
y(1:n) = y(1:n) + w(1:n)
\end{verbatim}
**Parameter in CUDA implementation**

- `<int BLOCK_SIZE>` :: Thread block size
- `<int UX>` :: Unrolling for outer-most loop
- (int MULTIPLICITY) :: Multiplicity for threadBLK.
- `<int MX>` :: Stream order *this may affect consumption of registers*

---

```
// template
template <int BLOCK_SIZE, int UX, int MX>
__global__ void symv_kernel(){
  i0=blockidx.x*UX;
  for(j=0; j<i0-1; j+=BLOCK_SIZE){
    // following loop is unrolled by hand in the
    // real code
    for(i_=0;i_<UX; i_++){
      i=i0+i_;
      y[i] += a[i][i]*x[j];
      w[j] += a[i][i]*x[i];
    }
    y[i] += a[i][i]*x[i];
  }
}
```

```
#define M(x) a##x=a[lda*(x)]
switch (MX){
  case 0: (M(0), M(1), M(2), M(3),...); break;
  case 1: (M(1), M(0), M(3), M(2),...); break;
  case 2: (M(2), M(0), M(6), M(4), ...); break;
  ...
  case 6: (M(4), M(0), M(12), M(8),...); break;
  ...
}
```

---

Parameter in CUDA implementation

- `<int BLOCK_SIZE>` :: Thread block size
- `<int UX>` :: Unrolling for outer-most loop
- `(int MULTIPLICITY)` :: Multiplicity for threadBLK.
- `<int MX>` :: Stream order this may affect consumption of registers

<table>
<thead>
<tr>
<th>Parameter</th>
<th>candidates</th>
<th>Total number</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOCK_SIZE</td>
<td>{32,64,...,256}</td>
<td>8</td>
</tr>
<tr>
<td>UX</td>
<td>{8,9,...,32}</td>
<td>25</td>
</tr>
<tr>
<td>MULTIPLICITY</td>
<td>{1,2,...,8}</td>
<td>8</td>
</tr>
<tr>
<td>MX</td>
<td>{0,1,...,9}</td>
<td>10</td>
</tr>
</tbody>
</table>

Constraint by capacity of the shared mem. :

\[
UX + \max \left\{ \frac{16K}{\text{MULTIPLICITY} \times 8}, (UX)^2 \right\} < 16K
\]

In case of other level2 BLAS kernels, **GEMV(n) :: 37**  **GEMV(t) :: 1400**
Automatic Tuning

- General scheme = two-stage tuning *(Off-line Tuning)*
  1. Sieving the number of parameter set
  2. d-spline interpolation

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

- General scheme = two-stage tuning
  1. **Sieving the number of parameter set**
  2. d-spline interpolation
Automatic Tuning

- General scheme = two-stage tuning
  1. Sieving the number of parameters
  2. d-spline interpolation

\[
\begin{align*}
f &= (f_1, f_2, \ldots, f_N)^T \\
y &= (y_1, y_2, \ldots, y_k)^T \\
s &= (s_1, s_2, \ldots, s_k)^T
\end{align*}
\]

\[
\min(\|y - Ef\|^2 + \alpha^2 \|Df\|^2)
\]

\[
D = (1, -2, 1), E_{s(i), i} = 1
\]

T. Tanaka, et.al, d-Spline Based Incremental Parameter Estimation in Automatic Performance Tuning, LNCS4699.
Multi-dimensional d-spline curve

Definition by Tensor representation

- **1D** \( \leftrightarrow \rightarrow 1 \) parameter
- **2D** \( \leftrightarrow \rightarrow 2 \) parameters

\[
f = (f_1, f_2, \ldots, f_N)^T
\]
\[
y = (y_1, y_2, \ldots, y_k)^T
\]
\[
s = (s_1, s_2, \ldots, s_k)^T
\]

\[
\min \left( \| y - Ef \|^2 + \alpha^2 \| Df \|^2 \right)
\]

\[
D_1^k = (1, -2, 1)^k, \quad E_{s(i),i} = 1
\]
\[
D_2^{k_1 \times k_2} = D_1^{k_1} \otimes I^{k_2} + I^{k_1} \otimes D_1^{k_2}
\]
\[
D_3^{k_1 \times k_2 \times k_3} = D_1^{k_1} \otimes I^{k_2} \otimes I^{k_3}
\]
\[
+ I^{k_1} \otimes D_1^{k_2} \otimes I^{k_3}
\]
\[
+ I^{k_1} \otimes I^{k_2} \otimes D_1^{k_3}
\]
Case study for ASPEN.K2 SYMV kernels

- Tesla C2050 (CUDA4.0)
- GeForce GTX580 (CUDA5.0)
- Tesla K20c (CUDA5.0)

- Overhead of Sampling (=cost)
  - For (BLOCK_SIZE, UX, MULTIPLICITY, MX), the force sampling was done. It took 10 to 24 hours. Compiling cost is quite dominant.

<table>
<thead>
<tr>
<th></th>
<th>C2050</th>
<th>GTX580</th>
<th>K20c</th>
</tr>
</thead>
<tbody>
<tr>
<td>N. of Parameters</td>
<td>6646</td>
<td>7276</td>
<td>7200</td>
</tr>
<tr>
<td>Sampling time</td>
<td>23:09</td>
<td>9:59</td>
<td>10:37</td>
</tr>
<tr>
<td>Ave. time/sample</td>
<td>6 sec</td>
<td>5 sec</td>
<td>5 sec</td>
</tr>
</tbody>
</table>
Case study for SYMV kernels

- Champion kernels by
  \[ N = \{256, 512, 1001, 2001, 4001, 6001, 9001, 12000, 16655, 17000\} \]

<table>
<thead>
<tr>
<th>Rank</th>
<th>C2050</th>
<th>GTX580</th>
<th>K20c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(128,15,5,0)</td>
<td>(128,14,6,9)</td>
<td>(160,14,5,7)</td>
</tr>
<tr>
<td>2</td>
<td>(128,16,5,0)</td>
<td>(128,14,5,9)</td>
<td>(160,13,5,2)</td>
</tr>
<tr>
<td>3</td>
<td>(96,15,6,0)</td>
<td>(128,14,6,1)</td>
<td>(128,14,6,6)</td>
</tr>
<tr>
<td>4</td>
<td>(96,16,5,0)</td>
<td>(160,14,4,9)</td>
<td>(160,13,6,2)</td>
</tr>
<tr>
<td>5</td>
<td>(96,20,3,0)</td>
<td>(128,19,4,5)</td>
<td>(96,16,1,8)</td>
</tr>
<tr>
<td>6</td>
<td>(128,17,4,0)</td>
<td>(96,20,3,4)</td>
<td>(96,14,7,6)</td>
</tr>
</tbody>
</table>

In fact, the best kernel depends on matrix dimension.
Performance summary

CUDA 16~17GFLOPS
MAGMA 35~37GFLOPS

C2050 (Fermi, 448 cores, sustained 99GB/s)

49.5 GFLOPS

N: Matrix dimension, NxN

CUDA 24~25GFLOPS
MAGMA 47~49GFLOPS

GTX580 (Fermi, 512 cores, sustained 166GB/s)

83 GFLOPS

N: Matrix dimension, NxN

Our Eigenvalue solver for GPGPU outperforms MAGMA 1.4.0 (2.3x).
Case study for ASPEN.K2 SYMV kernels

- Tesla C2050 (CUDA4.0)
- GeForce GTX580 (CUDA5.0)
- Tesla K20c (CUDA5.0)

- Overhead of Sampling (=cost)
  - For (BLOCK_SIZE, UX, MULTIPLICITY, MX), the force sampling was done. It took 10 to 24 hours. Compiling cost is quite dominant.

<table>
<thead>
<tr>
<th></th>
<th>C2050</th>
<th>GTX580</th>
<th>K20c</th>
</tr>
</thead>
<tbody>
<tr>
<td>N. of Parameters</td>
<td>6646</td>
<td>7276</td>
<td>7200</td>
</tr>
<tr>
<td>Sampling time</td>
<td>23:09</td>
<td>9:59</td>
<td>10:37</td>
</tr>
<tr>
<td>Ave. time/sample</td>
<td>6 sec</td>
<td>5 sec</td>
<td>5 sec</td>
</tr>
</tbody>
</table>
On going and Future work

- **How to reduce the AT cost from this AT framework?**
  - First stage is too much. (10 to 24 hours)
  - There are four types of precision to be optimized (SP, DP, ZP, CP).
  - We do not want to do Brute Force sampling!!

- We need only a rough value of the performance of each kernel.
- Introduction of interpolation or some approximation technique is enough to predict top-rank kernels.
Conclusion & On-going/Future work

- General tuning framework of two-stage tuning is proposed. ASPEN.K2 is developed by this AT framework.
  - 1st Sieving the number of parameter sets.
  - 2nd performance profile by d-spline with more detailed sampling.

- For the case study, SYMV kernel on two GPU generations, Fermi and Kepler is examined.
  - We recognize that there is a big generation gap between Fermi and Kepler.
  - Auto-Tuning is necessary.
  - However the Brute-Force search is too heavy!

- In future work, reduction in the first sieving stage is an inevitable issue.
  - An incremental D-spline curve interpolation is now developing and will be released in 2013.
  - Online tuning is challenging approach…
  - Avoiding jitter effects from the host system
Bayesian based on line/off-line parameter tuning
Prof. Fujii (Kougakuin U.)
Prof. Suda (U. Tokyo)

** Reiji Suda, and Vivek S.Nittoor, Efficient Monte Carlo Optimization with ATMathCoreLib *IPSJ-SIGHPC, 2012-03-13, 2012-02-16.*
pseudo code for online tuning

ATMathCoreLib is available from
http://olab.is.s.u-tokyo.ac.jp/~reiji/atmathcorelib/

!C exdes is data structure used by ATMathCoreLib
exdes = new_exdesign_nm(parameter_size,1E-8)

!C online tuning with n_max trials at most
nmax= 1000
Do n = 1, nmax
    i = exdes_nm_online(exdes, nmax-n+1)
    Call AMG_parameter_assign(i)
    cost = AMG_solve(A,x,b)
    Call update_exdes_nm(exdes,i,cost)
End Do
bestp = exdes_nm_getbest(exdes)

Solver execution time[s]
Online Tuning
3D Darcy Problem $10^6$ DOF

- ATMathCoreLib selects best parameter setting considering performance fluctuation
- It stops search trials in accordance with future loop length
THANK YOU
Outline

1. Introduction
2. CUDA BLAS kernels
   - ASPEN.K2 = Level2 CUDA BLAS + AT
   - SYMV CUDA implementation
3. Automatic Tuning framework
   - Two-stage scheme
   - D-Spline curve interpolation
   - Tuning Result
     - Fermi Core (Tesla C2050, GeForce GTX580)
     - Kepler Core (Tesla K20c)
4. On-Going work
5. Conclusion & Future work