Randomization Algorithm to Compute Low-Rank Approximation

HAN Ru

The Chinese University of Hong Kong
Outline

- Background
- Algorithm and Math Model
- Project Scheme
  - Done
  - To do
Background

● General SVD

\[ A = U \Sigma V^t \]

matrices \( U = [u_1u_2...u_m] \in \mathbb{R}^{m \times m} \); \( V = [v_1v_2...v_n] \in \mathbb{R}^{n \times n} \)

\( \Sigma = \text{diag}(\sigma_1,...,\sigma_n) \), where \( \Sigma \in \mathbb{R}^{m \times n}, n = \min\{m,n\} \) and \( \sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0 \).

● Low-Rank SVD Approximation

\[ A = U_k \Sigma_k V_k^t \]

● LAPACK/MAGMA software framework
Algorithm--Power iteration:

Matlab Code “svd_rand./” SVD approximation

function [u,s,v] = svd_rand(A, k, l, max_iters)
q = randn(n,k+l);
[q,r] = qr(q,0);
for iter=1:(max_iters-1)
   p = A*q;
   q = A'*p;
   [q,r] = qr(q,0);
end
p = A*q;
[p,b] = qr(p,0);
end
[x,s,y] = svd(b);

u_k = p*x(:,1:k);
s = s(1:k,1:k);
v_k = q*y(:,1:k);
## Algorithm and Math Model

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>M-by-N</td>
</tr>
<tr>
<td>Q</td>
<td>N-by-(K+L)</td>
</tr>
<tr>
<td>P</td>
<td>M-by-(K+L)</td>
</tr>
<tr>
<td>B</td>
<td>(K+L)-by-(K+L)</td>
</tr>
<tr>
<td>X</td>
<td>(K+L)-by-(K+L)</td>
</tr>
<tr>
<td>Yᵀ</td>
<td>(K+L)-by-(K+L)</td>
</tr>
<tr>
<td>SỊ</td>
<td>(K+L)-by-1</td>
</tr>
<tr>
<td>S</td>
<td>K-by-1</td>
</tr>
<tr>
<td>uₖ</td>
<td>M-by-K</td>
</tr>
<tr>
<td>vₖ</td>
<td>N-by-K</td>
</tr>
</tbody>
</table>

Error: \[ \| A - Uₖ Sₖ Vₖ^T \|_2 \]

\[ = (k+1)_{th \text{ largest singular value of A}} \]
Project Scheme

1. Implementing the randomized algorithm using LAPACK on CPU

2. Implementing the randomized algorithm using MAGMA on GPU

3. Implementing the out-of-memory randomized algorithm on GPU
   - single queue
   - UMA
   - manual pipelining.
   - Multiple GPUs using CUBLAS-XT

4. Set up tester to compare performance

5. Application
Done

1. Implementing the randomized algorithm using LAPACK on CPU

2. Implementing the randomized algorithm using MAGMA on GPU

3. Implementing the out-of-memory randomized algorithm on GPU with single queue

4. Set up tester to compare the performance

LAPACK &CPU&GPU&OUT-OF-MEMORY

Error is \[ \| A - U_k S_k V_k^T \|_2 \], L=K, performs 10 iterations

<table>
<thead>
<tr>
<th>M</th>
<th>N</th>
<th>K</th>
<th>LAPACK time (s)</th>
<th>Randomized time (s)</th>
<th>LAPACK error</th>
<th>Randomized error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00, 0.00, 0.00, 0.00, 0.00 , 0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NB=100, m=1022, n=1022</td>
<td>1022</td>
<td>1022</td>
<td>2</td>
<td>0.91</td>
<td>0.01, 0.03, 0.09</td>
<td>1.83e+01</td>
</tr>
</tbody>
</table>
Out-of-Memory GPU Implementation

Device: Tesla K80, 823.5 MHz clock, 11439.9 MiB memory, capability 3.7

1 MiB = $2^{20}$ bytes = 1024 kibibytes = 1048576 bytes

$11439.9 \text{MiB} \times 1048576 = 1.1996e+10$ bytes

$\sqrt{(12e9/8)} = 3.8730e+04$
Out-of-Memory GPU Implementation

\[ P = A^*Q \]

\[ P = 0; \]

\[ \text{For } k = 1, 2, 3 \ldots \]

\[ \text{set } (A_k \text{ to } dA); \]

\[ P = P + A_k Q_k; \]

\[ \text{end} \]
Out-of-Memory GPU Implementation

\[ Q = A^t \times P \]

For \( k = 1, 2, 3 \ldots \)
set \( A_k \) to \( dA \);
\[ Q_k = A_k^t \times P; \]
end

\[ A \]

\[ A_1^t \]
\[ A_2^t \]
\[ A_3^t \]

\[ P \]

\[ Q \]

\[ Q1 \]

\[ Q1 \]

\[ Q1 \]
Out-of-Memory GPU Implementation using single queue
To do

- Implementation on single GPU using UMA (Unified Memory Access)
- Implementation on single GPU using manually pipelining
- Implementation on multiple GPUs using CUBLAS-XT
- Application—image processing
To do-Implementation on GPU using UMA
To do-Implementation on GPU using manually pipelining

Time Line

<table>
<thead>
<tr>
<th>queue1</th>
<th>queue2</th>
</tr>
</thead>
<tbody>
<tr>
<td>gemm</td>
<td>set</td>
</tr>
<tr>
<td>gemm</td>
<td>set</td>
</tr>
<tr>
<td>gemm</td>
<td>set</td>
</tr>
</tbody>
</table>
To Do-Application

- Latent Semantic Indexing (LSI)
- Genetic clustering
- subspace tracking
- image processing
Reference


