Conjugate Gradient Solver for SU(2) Staggered Fermion on GPU

Kenji Ogawa
Chung Yuan Christian University
Seminar at Kobayashi-Maskawa Institute
4th March. 2013
Outline

Motivation
Staggered Fermion
Conjugate Gradient Solver
GPU Computation
Example
Summary and Comments
Motivation

SU(2) Gauge Theory with many flavor
= Candidate of Beyond Standard Model
(Walking Techni-Color)

Strongly Interacting System
=> Lattice Simulation is useful tool

HMC Simulation & Measurement
=> Need to Solve Linear Equation $s = Dx$ (x is unknown)
Conjugate Gradient or it’s modification is widely used.

GPU is effective for sparse matrix calculation
Staggered Fermion

Way out of Nielsen Ninomiya No-go theorem

\[ S_F = \frac{1}{2} \sum_{n,\mu} \eta_\mu(n) \left[ \bar{\chi}(n)U_\mu(n)\chi(n + \hat{\mu}) - \bar{\chi}(n)U_\mu^\dagger(n - \hat{\mu})\chi(n - \hat{\mu}) \right] + M \sum_n \bar{\chi}(n)\chi(n) \]

\[ \eta_\mu(n) = (-1)^{n_1+n_2+\cdots+n_{\mu-1}} \]

\[ S_F = \frac{1}{2} \sum_{n,\mu} \left[ \bar{\chi}(n)U_\mu'(n)\chi(n + \hat{\mu}) - \bar{\chi}(n)U_\mu'^\dagger(n - \hat{\mu})\chi(n - \hat{\mu}) \right] + M \sum_n \bar{\chi}(n)\chi(n) \]

\[ U_\mu'(n) = \eta_\mu(n)U_\mu(n) \]

Hyper Cube (16 DOF)

=>

Considered as 4 dirac and “4 taste”

\[ q(x) = \sum_{\rho_i} \gamma^{\rho_1}\gamma^{\rho_2}\gamma^{\rho_3}\gamma^{\rho_4}\chi(x + \rho) \]
Even-Odd Preconditioning (Checkerboard)

Even Site:
\[(ix + iy + iz + it) \mod 2 = 0\]

\[
\begin{pmatrix}
m & D_{EO} \\
D_{OE} & m
\end{pmatrix} = \begin{pmatrix}
1 & \frac{1}{m} & 0 \\
0 & D_{OE} \frac{1}{m} & 1
\end{pmatrix} \begin{pmatrix}
m & 0 \\
0 & m - D_{OE} \frac{1}{m} D_{EO}
\end{pmatrix} \begin{pmatrix}
1 & D_{EO} \\
0 & 1
\end{pmatrix}
\]

Inverse
\[
\begin{pmatrix}
m & D_{EO} \\
D_{OE} & m
\end{pmatrix}^{-1} = \begin{pmatrix}
1 & -D_{EO} \\
0 & 1
\end{pmatrix} \begin{pmatrix}
m & 0 \\
0 & m - D_{OE} \frac{1}{m} D_{EO}
\end{pmatrix}^{-1} \begin{pmatrix}
1 & \frac{1}{m} \\
-D_{OE} & 0
\end{pmatrix}
\]

Determinant
\[
\det \begin{pmatrix}
m & D_{EO} \\
D_{OE} & m
\end{pmatrix} = \det \begin{pmatrix}
m & 0 \\
0 & m - D_{OE} \frac{1}{m} D_{EO}
\end{pmatrix}
\]
Conjugate Gradient \( b = Ax \)

Initial Condition
\[
d_0 = r_0 = b - Ax_0
\]

Iteration \( i=0,1,2, \ldots \)
\[
x_{i+1} = x_i + \alpha_i d_i \quad \quad 2 \quad \text{Solution}
\]
\[
d_{i+1} = r_{i+1} + \beta_{i+1} d_i \quad \quad 5 \quad \text{Update Direction}
\]
\[
r_{i+1} = r_i - \alpha_i Ad_i \quad \quad 3 \quad \text{Residual}
\]

Coefficients are
\[
\alpha_i = \frac{(r_i, r_i)}{(d_i, Ad_i)} \quad \quad 1
\]
\[
\beta_{i+1} = \frac{(r_{i+1}, r_{i+1})}{(r_i, r_i)} \quad \quad 4
\]
Conjugate Gradient \( b = Ax \)

Initial Condition

\[ d_0 = r_0 = b - Ax_0 \]

Iteration \( i=0,1,2, \ldots \)

\[ \alpha_i = \frac{(r_i, r_i)}{(d_i, Ad_i)} \]

\[ r_{i+1} = r_i - \alpha_i Ad_i \quad \text{Residual} \]

\[ x_{i+1} = x_i + \alpha_i d_i \quad \text{Solution} \]

\[ \beta_{i+1} = \frac{(r_{i+1}, r_{i+1})}{(r_i, r_i)} \]

\[ d_{i+1} = r_{i+1} + \beta_{i+1} d_i \quad \text{Update Direction} \]
Example  $b = Ax$ with $A = \begin{pmatrix} 1 & 2 \\ 2 & 5 \end{pmatrix}$  $b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

$$d_0 = r_0 = b - Ax_0 \quad \alpha_i = \frac{(r_i, r_i)}{(d_i, Ad_i)} \quad x_{i+1} = x_i + \alpha_i d_i$$

$$r_{i+1} = r_i - \alpha_i Ad_i \quad \beta_{i+1} = \frac{(r_{i+1}, r_{i+1})}{(r_i, r_i)} \quad d_{i+1} = r_{i+1} + \beta_{i+1} d_i$$

$$(d_i, r_j) = 0 \text{ for } i < j \quad (r_i, r_j) = 0 \text{ for } i \neq j$$

This guarantees $r_N = 0$
Comments on $b=Ax$

It needs efforts to find the solution.
On the other hand,
It is very easy to check the solution.

This property is crucial for Mixed Precision CG
**Concept of Mixed Precision Solver**

- Difference of star and square of above
- Difference of star and square of above
- Sum of square
Defect Correction and Reliable Update

Defect Correction

SP: CG Solver, with $d_0 = r_0$

DP: Sum SP solution to DP solution

DP: Calculate residual vector and set it as SP source

$$ (d_i, r_j) = 0 \text{ for } i < j \text{ is reset in defect correction} $$

Reliable Update

SP: CG Solver, without refreshing $d_0$

DP: Sum SP solution to DP solution

DP: Calculate residual vector and set it as SP source

Convergence is faster for Reliable Update
GPU (Graphic Processor Units)

Originally Developed for Video Gamers

many core (~1000) works for SIMD (Single Instruction Multiple Data) simultaneously

Nvidia’s recent product have several level of memory with different size and access time to the computing cores
Coding with CUDA

CPU Code

for( int ii = 0 ; ii < n ; ii ++ )
{
    a[ii] = b[ii] + c * d[ii];
    ...
}

operation 1

operation 2

operation 3

Scheduler in GPU
automatically assigns
threads to cores in an effective way

Thread: smallest unit of operation in CUDA.

computing cores on GPU

parallelize into “thread”s and “block”s

thread ID

ii=0

ii=1

ii=n-1
Tuning of GPU

Principle = Let processors works all time.
=> reduce the time to get / send data

Memory
Turn on/off L1 Cache, Texture, Shared Memory

Coalesce
Reorder the data as GPU friendly

Magic Number (depend on the architecture)
Warp, Block Size, ....

Reordering the Operation
Recycle the data on the cache as much as possible
Coalesce

core (threads)

2 access

global memory

8 access
### Example of Mixed Precision CG

**L24^3T48 Lattice, Beta=2.7, aM = 0.25, Naive Staggered**

“Tesla” is used

<table>
<thead>
<tr>
<th>Precision Level</th>
<th>Equation</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double Precision</td>
<td>$(vR, vR) / (vSrc, vSrc)$</td>
<td>$(d_i, A d_i) \cdot (r_{i-1}, r_{i-1}) \cdot (r_i, r_i)$</td>
</tr>
<tr>
<td>Single Precision</td>
<td>$(S, T, U)$</td>
<td>$9.14439850E+01 / 8.18142605E+00 / 2.7459325E+00$</td>
</tr>
<tr>
<td>Double Precision</td>
<td>$(S, T, U)$</td>
<td>$2.68391886E-04 / 2.98086998E-05 / 7.48642242E-06$</td>
</tr>
<tr>
<td>Single Precision</td>
<td>$(vR, vR) / (vSrc, vSrc)$</td>
<td>$8.44834140E-05 / 7.48642378E-06 / 2.51462211E-06$</td>
</tr>
<tr>
<td>Double Precision</td>
<td>$(S, T, U)$</td>
<td>$3.13231913E-10 / 3.60102989E-11 / 1.46738698E-11$</td>
</tr>
<tr>
<td>Single Precision</td>
<td>$(vR, vR) / (vSrc, vSrc)$</td>
<td>$6.73919755E-20$</td>
</tr>
<tr>
<td>Double Precision</td>
<td>$(S, T, U)$</td>
<td>$1.67335437E-10 / 1.46738732E-11 / 4.63271088E-12$</td>
</tr>
<tr>
<td>Single Precision</td>
<td>$(vR, vR) / (vSrc, vSrc)$</td>
<td>$4.74928462E-16 / 5.84574362E-17 / 3.02788688E-17$</td>
</tr>
</tbody>
</table>

100 SP Iteration

Time-CG[sec] 1.340796
Summary and Comments

SU(2) gauge theory
=> Candidate of Beyond Standard Modeled

Lattice Simulation
=> Large Part is Solving Linear Problem of Dirac Operator

Speed up of Conjugate Gradient
Even Odd Preconditioning
Mixed Precision
Defect Correction, Reliable Update

1 GPU ~ 1 Node of Super Computer

Special Thanks, Aoyama-san (KMI)