HPCG Performance Improvement on the K computer ~short introduction~

Kiyoshi Kumahata\(^1\), Kazuo Minami\(^1\)
Akira Hosoi\(^2\), Ikuo Miyoshi\(^2\)
1) RIKEN AICS
2) FUJITSU LIMITED
Our previous code until SC15

Our previous tuned code marked 0.461 PFLOPS for SC15

<table>
<thead>
<tr>
<th>Rank</th>
<th>Computer</th>
<th>Country</th>
<th>HPL PFLOPS</th>
<th>HPCG PFLOPS</th>
<th>Ratio to HPL %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tianhe-2</td>
<td>China</td>
<td>33.86</td>
<td>0.580</td>
<td>1.7%</td>
</tr>
<tr>
<td>2</td>
<td>K computer</td>
<td>Japan</td>
<td>10.51</td>
<td>0.461</td>
<td>4.4%</td>
</tr>
<tr>
<td>3</td>
<td>Titan</td>
<td>USA</td>
<td>17.59</td>
<td>0.322</td>
<td>1.8%</td>
</tr>
<tr>
<td>4</td>
<td>Trinity</td>
<td>USA</td>
<td>8.10</td>
<td>0.183</td>
<td>2.3%</td>
</tr>
<tr>
<td>5</td>
<td>Mira</td>
<td>USA</td>
<td>8.59</td>
<td>0.167</td>
<td>1.9%</td>
</tr>
</tbody>
</table>


Bandwidth on the K computer@Compute Node

<table>
<thead>
<tr>
<th>Theoretical</th>
<th>STREAM</th>
<th>SPMV</th>
<th>SYMGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GB/s</td>
<td>64</td>
<td>46</td>
<td>48</td>
</tr>
</tbody>
</table>

It is impossible to improve the bandwidth. To get more score, we have to use another way, especially hot kernel SYMGS.
Symmetric Gauss-Seidel

Structure of Original Symmetric Gauss-Seidel

**Forward Loop**

```java
for (int i = 0; i < nrow; i++) {
    double sum = r[i];
    for (int j = 0; j < nzInRow[i]; j++)
        sum -= Val[j]*x[Col[j]];
    sum += x[i]*Diag;
    x[i] = sum/Diag;
}
```

**Backward Loop**

```java
for (int i = nrow-1; i >= 0; i--) {
    double sum = r[i];
    for (int j = 0; j < nzInRow[i]; j++)
        sum -= Val[j]*x[Col[j]];
    sum += x[i]*Diag;
    x[i] = sum/Diag;
}
```
Symmetric Gauss-Seidel

**Forward** loop can be split into two loops: Loop1 & Loop2

```c
#omp parallel for 
for (int i = 0; i < nrow; i++) {
    double sum = r[i];
    for (int j = nzLInRow[i]; j < nzInRow[i]; j++)
        sum -= Val[j]*x[Col[j]];
    y[i] = sum;
}
```

```
for (int i = 0; i < nrow; i++) {
    double sum = y[i];
    for (int j = 0; j < nzLInRow[i]; j++)
        sum += Val[j]*x[Col[j]];
    x[i] = sum/Diag;
}
```

- Loop2

- Loop1

**Backward** loop can be split into two loops: Loop3 & Loop4

```c
#omp parallel for 
for (int i = nrow-1; i >= 0; i--) {
    double sum = r[i];
    for (int j = nzLInRow[i]; j < nzInRow[i]; j++)
        sum -= Val[j]*x[Col[j]];
    y[i] = sum;
}
```

```
for (int i = nrow-1; i >= 0; i--) {
    double sum = y[i];
    for (int j = 0; j < nzLInRow[i]; j++)
        sum += x[i]*Diag;
    x[i] = sum/Diag;
}
```

- Loop4

- Loop3

**Execution Order:** Loop1 → Loop2 → Loop3 → Loop4
Symmetric Gauss-Seidel

- Loop3 can be reversed
- Row $i$ of Loop3 can be calculated only if from row 0 to $i$ of Loop2 are calculated → more chance to use cache effectively

**Loop2**

```java
for (int i = 0; i < nrow; i++) {
    ... double sum = y[i];
    for (int j = 0; j < nzLInRow[i]; j++)
        sum -= Val[j] * x[Col[j]];
    sum += x[i] * Diag;
    x[i] = sum / Diag;
}
```

**Loop3**

```java
for (int i = 0; i < nrow; i++) {
    ... double sum = r[i];
    for (int j = 0; j < nzLInRow[i]; j++)
        sum -= Val[j] * x[Col[j]];
    y[i] = sum;
}
```

\[
x_i^{(k+\frac{1}{2})} = \frac{1}{a_{ii}} \left\{ y_i - \sum_{j<i}^N a_{ij} x_j^{(k+\frac{1}{2})} \right\}
\]

Loop direction reversing does not change arithmetic order!!

updated row 0 to i of Loop2

\[
y_i = r_i - \sum_{j<i}^N a_{ij} x_j^{(k+\frac{1}{2})}
\]
Symmetric Gauss-Seidel

Marching of updating of Loop2 and Loop3 (sample by 4X4 problem)

$$x_i^{(k+\frac{1}{2})} = \frac{1}{a_{ii}} \left\{ y_i - \sum_{j<i}^N a_{ij}x_j^{(k+\frac{1}{2})} \right\}$$

$$y_i = r_i - \sum_{j<i}^N a_{ij}x_j^{(k+\frac{1}{2})}$$
Symmetric Gauss-Seidel

Marching of updating of Loop2 and Loop3 (sample by 4X4 problem)

\[ x^{(k+\frac{1}{2})}_i = \frac{1}{a_{ii}} \left\{ y_i - \sum_{j<i}^N a_{ij}x^{(k+\frac{1}{2})}_j \right\} \]

\[ y_i = r_i - \sum_{j<i}^N a_{ij}x^{(k+\frac{1}{2})}_j \]
Symmetric Gauss-Seidel

Marching of updating of Loop2 and Loop3
(sample by 4×4 problem)

\[ x^{(k+1/2)}_i = \frac{1}{a_{ii}} \left\{ y_i - \sum_{j<i}^N a_{ij} x^{(k+1/2)}_j \right\} \]

\[ y_i = r_i - \sum_{j<i}^N a_{ij} x^{(k+1/2)}_j \]
Symmetric Gauss-Seidel

Marching of updating of Loop2 and Loop3 (sample by 4X4 problem)

\[ x^{(k+\frac{1}{2})}_i = \frac{1}{a_{ii}} \left\{ y_i - \sum_{j<i}^N a_{ij} x^{(k+\frac{1}{2})}_j \right\} \]

\[ y_i = r_i - \sum_{j<i}^N a_{ij} x^{(k+\frac{1}{2})}_j \]
Symmetric Gauss-Seidel

Marching of updating of Loop2 and Loop3
(sample by 4X4 problem)

\[ x^{(k+1/2)}_i = \frac{1}{a_{ii}} \left\{ y_i - \sum_{j<i}^N a_{ij}x^{(k+1/2)}_j \right\} \]

\[ y_i = r_i - \sum_{j<i}^N a_{ij}x^{(k+1/2)}_j \]
## History of score improvement

<table>
<thead>
<tr>
<th>BoF@</th>
<th>Tune</th>
<th>PFLOPS</th>
<th>RANK</th>
</tr>
</thead>
</table>
| SC15  | Previous
DOI: 10.1177/1094342015607950          | 0.461  | 2    |
| ISC2016 | Cache utilization by splitting
SYMGS loops (just implement) | 0.554  | 2    |
| SC16  | Cache utilization by splitting
SYMGS loops (more adjustment) | 0.602  | 1    |