Architecting the Finite Element Method Pipeline for the GPU

Zhisong Fu, T. James Lewis, Robert M. Kirby, Ross T. Whitaker
The Scientific Computing and Imaging Institute at the University of Utah

Motivation
In this project, we consider the numerical solution of the second order elliptic PDEs defined on a three dimensional tetrahedral domain with Finite Element Method (FEM):

\[-\nabla \cdot (\sigma(x) \nabla u(x)) + \lambda u(x) = f(x)\]

\[\sum_{j=1}^{N} (\nabla \phi_i \cdot \nabla \phi_j) \hat{u}_j + \lambda \sum_{j=1}^{N} (\phi_i \cdot \phi_j) \hat{u}_j = (\phi_i, f)\]

• The second order elliptic PDEs appear in many scientific and engineering problems.
• The FEM is a widely used method for solving the PDEs.

Background

1. GPU architecture

2. Compact assembly step

3. Geometry-informed AMG
• Geometry-informed partitioning for aggregates and patches.
• Smoothed aggregation multigrid.
• Block-Jacobi relaxations.
• New V-cycle.

Algorithm 5.3: V-cycle-new \(A^{(k)}_B, A^{(k)}_R, R^{(k)}, P^{(k)}, b^{(k)}, u^{(k)}\)

if level \(k\) is the coarsest level
then solve \(A^{(k)}u^{(k)} = b^{(k)}\) and return \(u^{(k)}\)

\[r^{(k)} = r^{(k)} - A^{(k)}d^{(k)} \]
\[r^{(k+1)} = R^{(k)}r^{(k)} \]
\[v^{(k+1)} = V^{-1}(A^{(k+1)}u^{(k+1)} + R^{(k+1)}u^{(k+1)}) \]
\[u^{(k+1)} = p^{(k)}v^{(k+1)} + u^{(k)} \]
\[b^{(k)} = b^{(k)} - A_B u^{(k)} \]
\[u^{(k)} = \text{post-relax}(A, u^{(k)}, b^{(k)})\]

Implementation

1. Bottom-up double partitioning with k-MIS.
   • Partition the nodes into aggregates.
   • Build induced graph from aggregates.
   • Partition again into patches.

2. Patch sparse matrix format (patchSPM) data structure.

Result

• CPU: Intel i7 965 Extreme, 3.2GHz, 8MB L3 cache
• GPU: Nvidia GTX 580, 1544MHz, 512 core
For the assembly step, we compare our GPU implementation with our optimized-CPU implementation.

<table>
<thead>
<tr>
<th>meshes</th>
<th>GPU</th>
<th>CPU</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>0.0298</td>
<td>1.080</td>
<td>36</td>
</tr>
<tr>
<td>Irregular</td>
<td>0.0229</td>
<td>1.010</td>
<td>44</td>
</tr>
<tr>
<td>Heart</td>
<td>0.0465</td>
<td>3.114</td>
<td>67</td>
</tr>
<tr>
<td>Brain</td>
<td>0.0355</td>
<td>3.077</td>
<td>87</td>
</tr>
<tr>
<td>Blobs</td>
<td>0.0319</td>
<td>2.525</td>
<td>79</td>
</tr>
</tbody>
</table>

For the iteration step, we compare our method with state of the art CPU and GPU libraries.

<table>
<thead>
<tr>
<th>meshes</th>
<th>patch</th>
<th>Hypre-PCGAMG</th>
<th>S1</th>
<th>CSSP-PCGAMG</th>
<th>S2</th>
<th>CSSP-CG</th>
<th>Hypre-CG</th>
<th>S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>0.139(9)</td>
<td>3.86(25)</td>
<td>28</td>
<td>0.175(36)</td>
<td>1.3</td>
<td>0.68(329)</td>
<td>3.73(329)</td>
<td>5</td>
</tr>
<tr>
<td>Irregular</td>
<td>0.167(31)</td>
<td>3.02(29)</td>
<td>18</td>
<td>0.21(36)</td>
<td>1.3</td>
<td>2.48(1639)</td>
<td>14.8(1639)</td>
<td>6</td>
</tr>
<tr>
<td>Heart</td>
<td>0.218(20)</td>
<td>11.2(31)</td>
<td>51</td>
<td>0.031(46)</td>
<td>2.9</td>
<td>4.6(14)</td>
<td>33.8(137)</td>
<td>7</td>
</tr>
<tr>
<td>Brain</td>
<td>0.162(9)</td>
<td>7.78(27)</td>
<td>47</td>
<td>0.42(45)</td>
<td>2.6</td>
<td>8.1(1389)</td>
<td>60.4(1389)</td>
<td>9</td>
</tr>
<tr>
<td>Blobs</td>
<td>0.172(25)</td>
<td>5.76(28)</td>
<td>33</td>
<td>0.49(50)</td>
<td>2.4</td>
<td>3.34(1048)</td>
<td>16.0(1048)</td>
<td>5</td>
</tr>
</tbody>
</table>