OpenCL Sparse Linear Solver for Circuit Simulation
Jason Mak, Matthew Guthaus
University of California, Santa Cruz

Motivation
• Sparse linear systems are common in scientific and engineering problems.
• Computing solutions for large systems may be extremely time consuming.
• Consumer graphics hardware, with its innate parallel architecture and numerous processing cores, has shown success in speeding up such computations.
• Nodal analysis of circuits, which may contain millions of nodes, generates very large, sparse systems.
• Multiple libraries have been written to use a GPU as a parallel linear solver. This project tests the effectiveness of one such library in circuit analysis.

The Conjugate Gradient Algorithm
For j=1 to (max iterations), do
(a) \( \alpha_j := \frac{(r_j, r_j)}{(Ap_j, p_j)} \)
(b) \( x_{j+1} := x_j + \alpha_j p_j \)
(c) \( r_{j+1} := r_j - \alpha_j Ap_j \)
(d) \( \beta_j := \frac{(r_{j+1}, r_{j+1})}{(r_j, r_j)} \)
(e) \( p_{j+1} := r_{j+1} + \beta_j p_j \)

• CG is an iterative method that takes multiple, repeated steps to allow the solution vector \( x \) to converge to the correct value.
• Most of computation is spent on two operations: matrix-vector multiplications (a) and vector inner products (c and d).
• Both operations can naturally be parallelized.
• In transient circuit analysis, the entire algorithm may be run repeatedly to simulate a circuit's behavior over time.

Matrix Vector Multiplication
\[
\begin{bmatrix}
 b_{1,1} & b_{1,2} & \ldots & b_{1,n} \\
 b_{2,1} & b_{2,2} & \ldots & b_{2,n} \\
 \vdots & \vdots & \ddots & \vdots \\
 b_{m,1} & b_{m,2} & \ldots & b_{m,n}
\end{bmatrix}
\begin{bmatrix}
 a_{1,1} \\
 a_{1,2} \\
 \vdots \\
 a_{m,1}
\end{bmatrix}
= \begin{bmatrix}
 b_{1,1}a_{1,1} + b_{1,2}a_{2,1} + \ldots + b_{1,n}a_{n,1} \\
 b_{2,1}a_{1,1} + b_{2,2}a_{2,1} + \ldots + b_{2,n}a_{n,1} \\
 \vdots \\
 b_{m,1}a_{1,1} + b_{m,2}a_{2,1} + \ldots + b_{m,n}a_{n,1}
\end{bmatrix}
\]

Vector Inner Product
\[
\begin{bmatrix}
 b_{1,1} \\
 b_{2,1} \\
 \vdots \\
 b_{m,1}
\end{bmatrix}
\begin{bmatrix}
 a_{1,1} \\
 a_{1,2} \\
 \vdots \\
 a_{1,m}
\end{bmatrix}
= \begin{bmatrix}
 b_{1,1}a_{1,1} + b_{1,2}a_{2,1} + \ldots + b_{1,m}a_{m,1}
\end{bmatrix}
\]

GPUs are known to possess many processing cores (shown in orange). The GTX 285 has 240 cores.

Conclusion
• Both a single threaded implementation and ViennaCL’s implementation of the conjugate gradient method were tested on IBM power grid circuits.
• ViennaCL’s implementation yielded a speedup on both CPUs (Intel Core 2 Duo and Athlon II) and a GPU (Nvidia GTS 250).
• In some cases, the GPU yielded a speedup by a factor of 4.
• The black-box linear solver was successful in speeding up circuit analysis and could be an easy replacement for linear solvers in other scientific applications as well.

OpenCL Sparse Linear Solver for Circuit Simulation
OpenCL and ViennaCL
• OpenCL is both an API and programming language (based on C) designed for making parallel applications.
• OpenCL supports both CPUs and GPUs.
• ViennaCL supports both CPUs and GPUs.
• ViennaCL employs black-box solvers by wrapping its OpenCL implementations in high-level interfaces to hide the tedious intricacies of GPU parallel programming from the user.

Power Grid Circuit Benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>ibmpg1</th>
<th>ibmpg2</th>
<th>ibmpg3</th>
<th>ibmpg4</th>
<th>ibmpg5</th>
<th>ibmpg6</th>
</tr>
</thead>
<tbody>
<tr>
<td>total current (I)</td>
<td>10774</td>
<td>37926</td>
<td>201054</td>
<td>276976</td>
<td>540800</td>
<td>761484</td>
</tr>
<tr>
<td>nodes</td>
<td>30638</td>
<td>127238</td>
<td>851584</td>
<td>953583</td>
<td>1079310</td>
<td>1670494</td>
</tr>
<tr>
<td>resistors</td>
<td>30027</td>
<td>208325</td>
<td>1401572</td>
<td>1560645</td>
<td>1076848</td>
<td>1649002</td>
</tr>
<tr>
<td>shorts (zeroes)</td>
<td>14208</td>
<td>1298</td>
<td>461</td>
<td>11682</td>
<td>606587</td>
<td>836107</td>
</tr>
<tr>
<td>voltage sources</td>
<td>14308</td>
<td>330</td>
<td>955</td>
<td>962</td>
<td>539087</td>
<td>836239</td>
</tr>
<tr>
<td>metal levels</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

IBM Power Grid Benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>ibmpg1</th>
<th>ibmpg2</th>
<th>ibmpg3</th>
<th>ibmpg4</th>
<th>ibmpg5</th>
<th>ibmpg6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (ms)</td>
<td>Single threaded Core 2 Duo</td>
<td>ViennaCL Core 2 Duo</td>
<td>ViennaCL Athlon II Quad</td>
<td>ViennaCL Nvidia GTS 250</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>