Sparse LU Factorization for Parallel Circuit Simulation on GPUs

Ling Ren, Xiaoming Chen, Yu Wang, Chenxi Zhang, Huazhong Yang

Nano-scale Integrated Circuit and System Lab.,
Department of Electronic Engineering,
Tsinghua University
Motivation

Parallel SPICE simulator

Fig. 1. The work flow of SPICE simulation
Related works

- **Dense LU** [Volkov2008, Tomov2010]
  - Very efficient on GPUs (850 Gflop/s)

- **Sparse LU**
  - SuperLU and Pardiso: Supernode (dense blocks)
    - [Christen2007] dense blocks on GPU
  - UMFPACK, MUMPS, WSMP: multifrontal

No dense blocks in extremely sparse matrices

- KLU, for circuit matrices, without Supernode
  - G/P left-looking algorithm [G/P 1988]

Sequential version only
Algorithm – left-looking

- Each column (k) is sequentially updated (vector multiply-and-add, MAD) by all the columns on its left.

\[ \begin{bmatrix} b \\ \tilde{b} \end{bmatrix} \rightarrow \begin{bmatrix} a \\ \tilde{a} \end{bmatrix} \]

\[ \tilde{a} = \tilde{a} - c \times \tilde{b} \]

\[ U_{jk} \quad \vec{l}_j \]

- Egraph [chen2011]
  - nodes: columns
  - Edges: vector MAD

Nonzero structure of U determines the dependency and the EGraph.

(a) Upper triangular matrix U
(b) EGraph
Algorithm analysis – parallelism

- Divide EGraph into levels
  - Columns in the same level are independent
- **Cluster mode & pipeline mode**

A sample EGraph

Timing order in pipeline mode
GPU implementation - avoid deadlock

- Traditionally, some warps
  - Inactive at the beginning
  - Activated when other active warps finish

- But in sparse LU, all warps must be active from the beginning

- An upper bound for concurrent columns
GPU implementation – memory access pattern

(a) nonzeros unsorted, access highly random

(b) nonzeros sorted, access more coalesced
GPU implementation - workflow

Preprocessing on CPU:
- Sorting the non-zeros in L and U;

// transfer data to GPU
- Write locations of non-zeros in A, L and U, and Escheduler to GPU;
- Write values of non-zeros in A to GPU;

// numeric factorization
- For each level in cluster mode do
  - Cluster mode on GPU;
- End for
  - Pipeline mode on GPU;

Read from GPU

A: M wavefronts
B: Dense arrays for intermediate results

Global memory

CPU

GPU
Performance analysis

- More concurrent columns, higher performance?
  - No, inexecutable operations.
Experiments

- CPU: 2 Xeon X5680
- GPU: NVIDIA GTX580
- Testing matrices
  - University of Florida Sparse Matrix Collection (not only circuit matrices)
- Hybrid solver
  - 1-core / multi-core / many-core (GPU)

<table>
<thead>
<tr>
<th>Group</th>
<th>Bandwidth (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GPU</td>
</tr>
<tr>
<td>A (flop &lt; 200M)</td>
<td>9.68</td>
</tr>
<tr>
<td>B (flop &gt; 200M)</td>
<td><strong>91.17</strong></td>
</tr>
<tr>
<td>Matrix</td>
<td>( N ) (K)</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>B1</td>
<td>TSOPF_RS_b300_c3</td>
</tr>
<tr>
<td>B2</td>
<td>sbp3</td>
</tr>
<tr>
<td>B3</td>
<td>raj1</td>
</tr>
<tr>
<td>B4</td>
<td>ASIC_680ks</td>
</tr>
<tr>
<td>B5</td>
<td>thermomch_TC</td>
</tr>
<tr>
<td>B6</td>
<td>ASIC_680k</td>
</tr>
<tr>
<td>B7</td>
<td>ASIC_100k</td>
</tr>
<tr>
<td>B8</td>
<td>ASIC_100ks</td>
</tr>
<tr>
<td>B9</td>
<td>rma10</td>
</tr>
<tr>
<td>B10</td>
<td>onetone1</td>
</tr>
<tr>
<td>B11</td>
<td>thermomch_dM</td>
</tr>
<tr>
<td>B12</td>
<td>venkat20</td>
</tr>
<tr>
<td>B13</td>
<td>Zhao</td>
</tr>
<tr>
<td>B14</td>
<td>thermomch_dK</td>
</tr>
<tr>
<td>B15</td>
<td>crashbasis</td>
</tr>
<tr>
<td>B16</td>
<td>G2_circuit</td>
</tr>
<tr>
<td>B17</td>
<td>twotone</td>
</tr>
<tr>
<td>B18</td>
<td>sme3Dc</td>
</tr>
<tr>
<td>B19</td>
<td>xenon1</td>
</tr>
<tr>
<td>B20</td>
<td>helm2d03</td>
</tr>
<tr>
<td>B21</td>
<td>denormal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Average</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Average</td>
<td><strong>6.00</strong></td>
<td><strong>9.68</strong></td>
<td><strong>1.61</strong></td>
<td><strong>0.77</strong></td>
<td><strong>0.54</strong></td>
<td><strong>1.88</strong></td>
<td><strong>1.86</strong></td>
<td><strong>1.69</strong></td>
<td><strong>1.13</strong></td>
<td><strong>0.60</strong></td>
<td><strong>4.23</strong></td>
</tr>
</tbody>
</table>

1. Matrix dimension
2. Number of nonzeros in Matrix A
3. Number of Mega floating point operations
4. All the average values are geometric average
Table 1: Related specifications of different devices

<table>
<thead>
<tr>
<th>Devices</th>
<th>Xeon E5405</th>
<th>Xeon X5680</th>
<th>Radeon 5870</th>
<th>GTX580</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak Bandwidth</td>
<td>--</td>
<td>--</td>
<td>153.6 GB/s</td>
<td>192.4 GB/s</td>
</tr>
<tr>
<td>Number of Cores</td>
<td>2×4</td>
<td>2×6</td>
<td>20 CUs</td>
<td>16 SMs²</td>
</tr>
<tr>
<td></td>
<td>= 8 cores</td>
<td>= 12 cores</td>
<td>320 cores¹</td>
<td>512 cores</td>
</tr>
<tr>
<td>Active groups</td>
<td>--</td>
<td>--</td>
<td>160</td>
<td>512</td>
</tr>
<tr>
<td>Active threads</td>
<td>8</td>
<td>8</td>
<td>10240</td>
<td>16384</td>
</tr>
<tr>
<td>L1 cache</td>
<td>32KB/core</td>
<td>32KB/core</td>
<td>8KB/CU</td>
<td>16KB/SIM</td>
</tr>
<tr>
<td>L2 cache</td>
<td>12MB/4 cores</td>
<td>256KB/core</td>
<td>512KB/all</td>
<td>768KB/all</td>
</tr>
<tr>
<td>L3 cache</td>
<td>--</td>
<td>12MB/6 cores</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Clock rate</td>
<td>2.0 GHz</td>
<td>3.2 GHz</td>
<td>850 MHz</td>
<td>772 MHz</td>
</tr>
</tbody>
</table>

¹ CU = Compute Unit. In Radeon 5870, each core contains 5 processing elements (PEs). But PEs are combined for double precision floating point operations [27], so they can be regarded as a single core in our problem.

² SM = Stream Multiprocessor

---

**Figure 8:** Performance of different devices on the second group of matrices
Hybrid Solver

- Based on the number of flops in the factorization
  - Sequential or parallel? [Chen 2011]
  - Single-core, multi-core or many-core (GPU)

- Accuracy: Pivoting once + several numerical factorization
  - Since nonzero values do not change rapidly
  - When nonzeros do vary greatly, pivot (preprocess) again
Summary

- Sparse LU solver on GPU
  - Timing order and work partitioning on GPU
  - The optimal number of concurrent columns
  - Memory access pattern

- Hybrid Solver
  - As FLOPS increase, left-looking algorithm should be done on 1-core, multi-core or many-core (GPU).
Limitation & Future work

- On distributed-memory machines (e.g. multiple GPU)?
- Limited memory on GPU

- Blocked Algorithm?
  Circuit partition + blocked factorization

- Boarded-Blocked-Diagonal (BBD) matrices

Thank you!
Reference


