CASK – Open Source Custom Architectures for Sparse Kernels

Paul Grigoras, Pavel Burovskiy, Wayne Luk

Imperial College London
What is Sparse Algebra?

- **Sparse Algebra** = Solving Sparse Linear & Nonlinear Systems of Equations
- Physical systems are often modelled as large, sparse sets of nonlinear equations
- **Sparsity** can be exploited to improve performance & reduce storage size

---

**Turkish Power Distribution Grid**
Source: [www.geni.org](http://www.geni.org)

**Source:** University of Florida Sparse Matrix Collection

**Corresponding Sparse Representation**
Need for Customisation in Sparse Algebra

For any sparse system – sparsity pattern, value range, order and nature of the algorithm may be used to improve performance

→ Custom Computing for Sparse Algebra
CASK – Custom Architectures for Sparse Kernels

CASK
Custom Architectures for Sparse Kernels

Sparse Matrix Benchmark

Open Source Tool
http://caskorg.github.io/cask/
FPL14, FCCM15, FPGA16

Generic Architecture

FPGA Implementation

Source: University of Florida Sparse Matrix Collection

From FPGA Accelerator DRAM
col ptr values col idx
k inputs

Cache
k inputs

Control Unit
Vector Unit

Parallel Reduction (in row)
Serial Reduction (in row)
Serial Reduction (in partition) C_b

Source: www.nallatech.com
Custom Architectures for Sparse Kernels

Cache Structure (Replicated vs Dynamic) - trade-off storage size/maximum order vs retrieval rate

k-way replication
+ k reads / cycle
- k x Vc storage
- increases traffic
- increases resource usage

p-way replication (p << k)
- conflicts ⇒ <= k reads / cycle
+ p x Vc storage
+ larger Vc ⇒ reduced traffic
+ reduced resource usage

Result Cache (On-Chip vs Off-Chip) - trade-off latency/performance vs problem size/resource usage

+ ~13 cycles latency
+ support sparse vector
- n < 100K
- more BRAMs

+ ~30K cycles latency
+ n > 100K
+ fewer BRAMs
Impact of Automated Customisation in Sparse Algebra

Table 1: Required architectures for each matrix, produced in our approach (for Maxeler Vectis)

<table>
<thead>
<tr>
<th>Name</th>
<th>Order</th>
<th>Nonzeros</th>
<th>Nnz/row</th>
<th>Cx</th>
<th>k</th>
<th>N_p</th>
<th>Cb</th>
<th>Logic/DSP/BRAM %</th>
<th>Peak Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>dense</td>
<td>2048</td>
<td>4194304</td>
<td>2048.00</td>
<td>2048</td>
<td>16</td>
<td>2</td>
<td>2048</td>
<td>42.63 / 23.41 / 43.14</td>
<td>38.4 / 6.30</td>
</tr>
<tr>
<td>psmigr_2</td>
<td>3140</td>
<td>540022</td>
<td>171.98</td>
<td>4096</td>
<td>16</td>
<td>2</td>
<td>3584</td>
<td>42.02 / 23.41 / 54.23</td>
<td>38.4 / 4.76</td>
</tr>
<tr>
<td>rmax2</td>
<td>3242</td>
<td>294276</td>
<td>90.77</td>
<td>4096</td>
<td>16</td>
<td>2</td>
<td>3584</td>
<td>42.02 / 23.41 / 54.23</td>
<td>38.4 / 3.99</td>
</tr>
<tr>
<td>rma10</td>
<td>46835</td>
<td>2374001</td>
<td>50.69</td>
<td>7168</td>
<td>16</td>
<td>2</td>
<td>47104</td>
<td>42.91 / 23.41 / 84.87</td>
<td>38.4 / 1.63</td>
</tr>
<tr>
<td>consph</td>
<td>83334</td>
<td>3046907</td>
<td>36.56</td>
<td>9216</td>
<td>8</td>
<td>2</td>
<td>83456</td>
<td>37.46 / 12.30 / 82.61</td>
<td>19.2 / 1.37</td>
</tr>
<tr>
<td>cant</td>
<td>62451</td>
<td>2034917</td>
<td>32.58</td>
<td>11264</td>
<td>8</td>
<td>2</td>
<td>62464</td>
<td>37.15 / 12.30 / 80.92</td>
<td>19.2 / 1.60</td>
</tr>
<tr>
<td>shipsec</td>
<td>140874</td>
<td>3977139</td>
<td>28.23</td>
<td>14336</td>
<td>16</td>
<td>1</td>
<td>141312</td>
<td>30.24 / 11.71 / 79.65</td>
<td>19.2 / 0.78</td>
</tr>
<tr>
<td>tors2</td>
<td>115967</td>
<td>1033473</td>
<td>8.91</td>
<td>15360</td>
<td>16</td>
<td>1</td>
<td>116224</td>
<td>30.97 / 11.71 / 78.62</td>
<td>19.2 / 0.19</td>
</tr>
<tr>
<td>t2d_q9_A_01</td>
<td>9801</td>
<td>87025</td>
<td>8.88</td>
<td>10240</td>
<td>8</td>
<td>2</td>
<td>10240</td>
<td>36.24 / 12.30 / 60.81</td>
<td>19.2 / 0.87</td>
</tr>
<tr>
<td>epbl</td>
<td>14734</td>
<td>95053</td>
<td>6.45</td>
<td>15360</td>
<td>8</td>
<td>2</td>
<td>14848</td>
<td>37.06 / 12.30 / 75.94</td>
<td>19.2 / 0.69</td>
</tr>
<tr>
<td>mac_econ</td>
<td>206500</td>
<td>1273389</td>
<td>6.17</td>
<td>15360</td>
<td>8</td>
<td>1</td>
<td>206848</td>
<td>27.31 / 6.10 / 73.07</td>
<td>9.6 / 0.08</td>
</tr>
<tr>
<td>scircuit</td>
<td>170998</td>
<td>958936</td>
<td>5.61</td>
<td>14336</td>
<td>16</td>
<td>1</td>
<td>171008</td>
<td>30.39 / 11.71 / 84.54</td>
<td>19.2 / 0.08</td>
</tr>
<tr>
<td>dw8192</td>
<td>8192</td>
<td>41746</td>
<td>5.10</td>
<td>8192</td>
<td>8</td>
<td>3</td>
<td>8192</td>
<td>45.59 / 18.45 / 78.57</td>
<td>28.8 / 0.68</td>
</tr>
</tbody>
</table>

- **Fully Support Wide Range Of Problems**
- **Generate Optimised Architectures**
- **Improve Performance and Resource Utilisation**

Future: application- and system-specific optimisations; run-time reconfiguration; energy reduction
Consider Instance Specific Custom Computing for Sparse Matrix Problems!

http://caskorg.github.io/cask