StreamSPAS

Tim Barth
NASA Ames Research Center
Moffett Field, California 94035-1000 USA
StreamSPAS: Stream SParse Algebra Suite

Pronounced: \textit{strēm-spās}

On CVS Repository:

chet.stanford.edu:/u/ianbuck/repository/brook/progs/streamSPAS
Objective

Explore the streaming language implementation of two sparse linear algebra algorithms pervasive in computational science:

- Sparse matrix-vector products
- Sparse complete/incomplete matrix factorization and inverse

*Remark:* I believe these are the two primary sparse matrix operations that people would like to see addressed in evaluating the merits of stream computing for sparse linear algebra.
StreamSPAS

- Provides a test suite of matrices corresponding to $p$-order finite element discretization using $C^0$ continuous Lagrange elements

Figure 1: Simplicial elements with Lagrange function interpolation showing visible placement of nodal degrees of freedom (dofs) (a) linear element (4 dofs), (b) quadratic element (10 dofs) and (c) cubic element with (20 dofs).
Sample tetrahedral meshes/matrices produced by StreamSPAS

<table>
<thead>
<tr>
<th># tetrahedra</th>
<th>$p$</th>
<th>$nrows$</th>
<th>$nnz$</th>
<th>average nnz/row</th>
</tr>
</thead>
<tbody>
<tr>
<td>1916</td>
<td>1</td>
<td>471</td>
<td>5846</td>
<td>12.41</td>
</tr>
<tr>
<td>1916</td>
<td>2</td>
<td>3158</td>
<td>80372</td>
<td>25.45</td>
</tr>
<tr>
<td>1916</td>
<td>3</td>
<td>9978</td>
<td>441676</td>
<td>44.26</td>
</tr>
<tr>
<td>7728</td>
<td>1</td>
<td>1594</td>
<td>21720</td>
<td>13.62</td>
</tr>
<tr>
<td>7728</td>
<td>2</td>
<td>11657</td>
<td>312779</td>
<td>26.83</td>
</tr>
<tr>
<td>7728</td>
<td>3</td>
<td>37918</td>
<td>1744342</td>
<td>46.00</td>
</tr>
<tr>
<td>93678</td>
<td>1</td>
<td>16414</td>
<td>244216</td>
<td>14.87</td>
</tr>
<tr>
<td>93678</td>
<td>2</td>
<td>130315</td>
<td>3669718</td>
<td>28.16</td>
</tr>
<tr>
<td>93678</td>
<td>3</td>
<td>435382</td>
<td>20743534</td>
<td>47.64</td>
</tr>
</tbody>
</table>
Figure 2: Mesh containing 10386 tetrahedra.
StreamSPAS contains several standard sparse matrix storage schemes

- Compressed sparse row (CSR)
- Compressed sparse column (CSC)
- Hypergraph edge storage (HES) for pattern symmetric matrices
- Element-by-element storage (EBES) for FEM matrices
CSR storage scheme:

\[ A = \begin{bmatrix}
  a_0 & 0 & a_1 & 0 & a_2 \\
  a_3 & a_4 & 0 & 0 & a_5 \\
  a_6 & 0 & 0 & a_7 & a_8 \\
  a_9 & a_{10} & a_{11} & a_{12} & a_{13} \\
  a_{14} & a_{15} & 0 & 0 & a_{16}
\end{bmatrix} \]

\[ isp[] = \{0, 3, 6, 9, 14, 17\} \]
\[ jsp[] = \{0, 2, 4, 0, 1, 4, 0, 3, 4, 0, 1, 2, 3, 4, 0, 1, 4\} \]
\[ Asp[] = \{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, a_{16}\} \]
StreamSPAS: HES

HES for pattern symmetric matrices:

\[ \text{#nonzeros} = \text{#rows} + 2 \times \text{#hyperedges} \]

\[ A = \begin{bmatrix}
  a_0 & 0 & a_1 & 0 & a_2 \\
  0 & a_3 & 0 & a_4 & a_5 \\
  a_6 & 0 & a_7 & 0 & a_8 \\
  0 & a_9 & 0 & a_{10} & 0 \\
  a_{11} & a_{12} & a_{13} & 0 & a_{14}
\end{bmatrix} \]

\[ \text{Imat[]} = \{(0, 2), (0, 4), (1, 3), (1, 4), (2, 4)\} \]

\[ \text{Amat[]} = \{(a_1, a_6), (a_2, a_{11}), (a_4, a_9), (a_5, a_{12}), (a_8, a_{13})\} \]

\[ \text{AmatDiag[]} = \{a_0, a_3, a_7, a_{10}, a_{14}\} \]
StreamSPAS: EBES

Find $u_h \in V_h$ such that

$$B(u_h, v_h) = F(v_h), \quad \forall v_h \in V_h$$

where $B(u_h, v_h)$ has an intrinsic elementwise representation, e.g.

$$B(u_h, v_h) = \int_{\text{Domain}} v_h \cdot L(u_h) \, dx = \sum_K \int_K v_h \cdot L(u_h) \, dx = \sum_K B_K(u_h, v_h)$$

so that the global matrix-vector product takes the form

$$Au = \sum_K A_K u_K$$

where $A_K$ and $u_K$ denote the algebraic system resulting from $B_k(u_h, v_h)$. 
StreamSPAS: SpMatVecXXX

Task is to solve

\[ y = Ax \]

given \( A \) sparse and \( x \) dense.
Algorithm **SpMatVecCSR**. Let $\vec{r}_i$ denote the $i$-th row of the matrix $A$,

$$A = \begin{bmatrix}
\vec{r}_0 \\
\vdots \\
\vec{r}_{m-1}
\end{bmatrix} .$$

The matrix-vector product is computed as $m$ independent dot products,

$$y_i = \vec{r}_i \cdot \vec{x} , \quad i = 0, \ldots, m - 1 .$$

Implementation of this algorithm requires memory gather operations with variable record sizes.
Algorithm **SpMatVecCSC**. Let \( \vec{c}_i \) denote the \( i \)-th column of the matrix \( A \),

\[
A = [\vec{c}_0, \ldots, \vec{c}_{m-1}] .
\]

The matrix-vector product is computed as a linear combination of matrix columns,

\[
\vec{y} = \sum_{i=0}^{m-1} x_i \vec{c}_i .
\]

Implementation of this algorithm requires scatter-op operations to memory using variable record sizes.
StreamSPAS: SpMatVecHES

Algorithm SpMatVecHES. Let $e$ denote a hyperedge in the graph of the sparse matrix. The matrix-vector product is computed as

$$A\vec{x} = A_{\text{diag}}\vec{x} + \sum_{e \in \text{hyperedges}} A_e \vec{x}_e$$

where $A_e$ and $\vec{x}_e$ are restrictions of $A$ and $\vec{x}$ to an edge of the hypergraph.

- The amount of matrix storage and arithmetic computation in SpMatVecHES is essentially the same as the SpMatVecCSR and SpMatVecCSC algorithms.
- Mixture of gather and scatter-ops
- The algorithm has a natural Brook implementation using fixed size records.
StreamSPAS: SpMatVecEBES

Algorithm **SpMatVecEBES**. Let \( k \) denote an element in a FEM mesh. The matrix-vector product is computed as

\[
A\vec{x} = \sum_{k \in \text{mesh}} A_k \vec{x}_k
\]

where \( A_k \) and \( \vec{x}_k \) are the element contributions to the global matrix and vector.

- The amount of storage and computation is larger than the **SpMatVecCSR**, **SpMatVecCSC**, and **SpMatVecHES** algorithms (see complexity analysis below).
- Mixture of gather and scatter-ops
- The algorithm has a natural Brook implementation using fixed size records.
Matrix-Vector products are just one small piece of StreamSPAS!

Current version under development includes

- Dense block matrix storage
- On-the-fly element matrix reconstruction to mitigate the effects of memory bandwidth limits
- Approximate sparse matrix factorization/inverse that avoids recursion

**Benchmark Goal:** Brook benchmark for preconditioned conjugate solution of Poisson equation using $p$-order finite elements on tetrahedral meshes