Motivation

Sparse matrix formats and operations (e.g., SpMV) are hard to implement and verify in low-level languages like C.

RB-CSR vs. SpMV

\[
\begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & b & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & d \\
\end{pmatrix}
\]

for (i = 0; i < M; i++, y += 2) {
  double y0 = y[0], y1 = y[1];
  for (k = Ap[i]; k < Ap[i + 1]; k++, Av += 6) {
    int j = Ai[k];
    double x0 = x[j], x1 = x[j + 1], x2 = x[j + 2];
    y0 += Av[0] * x0; y1 += Av[3] * x0;
    y0 += Av[1] * x1; y1 += Av[4] * x1;
    y0 += Av[2] * x2; y1 += Av[5] * x2;
  }
  y[0] = y0; y[1] = y1;
}

Research question: can sparse formats/operations be...

- Formulated as simple dataflow problems? Implemented in a small, high-level language?
- Formally proved correct?
- Compiled into efficient low-level code? Parallelized easily? Scale well?

Implementing sparse codes in LL

We represent sparse data with nested lists and pairs.

\[
\langle 0, a \rangle \quad \langle 1, b \rangle \quad \langle 2, c \rangle \quad \langle 2, d \rangle \quad \langle 3, e \rangle
\]

Example: CSR SpMV using dataflow.

Code generation

A naïve nested layout is undesirable. Instead, we (a) flatten nested lists and (b) layout lists of pairs as a pair of lists.

\[
\begin{array}{c}
\langle 0, a \rangle \\
\langle 1, b \rangle \\
\langle 2, c \rangle \\
\langle 2, d \rangle \\
\langle 3, e \rangle
\end{array}
\]

Compilation is syntax-directed: pipelines are implemented using temporaries and maps/reduces using loops. Optimization is applied to fuse operations in adjacent maps/reduces. We use rich type information to specialize loop iteration counts and optimize pointer increments, and deploy data dependency analysis to eliminate redundant use of sublist boundaries indirection (below).

Our compiled CSR/RB-CSR SpMV performs close to handwritten (sequential) C code, and scales well to multiple cores.

Benchmarking environment: 2.3 GHz single socket quad-core AMD Opteron processor with 8GB of memory.