Parallel sparse matrix vector multiplication and models for efficient parallelization

Bora Uçar

RO:MA, LIP, ENS Lyon, France

CR-08: Combinatorial scientific computing, December 2014
http://perso.ens-lyon.fr/bora.ucar/CR08/
Outline

1. Introduction to SpMxV

2. Parallel SpMxV
   - Row parallel
   - Column parallel
   - Row-column parallel

3. Hypergraphs and hypergraph partitioning
   - Hypergraph models for row-parallel SpMxV
   - Hypergraph models for column-parallel SpMxV
   - Hypergraph models for row-column parallel SpMxV
   - Some other partitioning problems
Introduction to SpMxV

Parallel SpMxV
- Row parallel
- Column parallel
- Row-column parallel

Hypergraphs and hypergraph partitioning
- Hypergraph models for row-parallel SpMxV
- Hypergraph models for column-parallel SpMxV
- Hypergraph models for row-column parallel SpMxV
- Some other partitioning problems
Sparse matrices: Compressed row storage

There are many ways to store a sparse matrix.

Recall the two standard representations which store only the nonzero entries.

\[
\begin{bmatrix}
1.1 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 2.2 & 0.0 & 2.4 & 0.0 \\
3.1 & 0.0 & 3.3 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 4.4 & 0.0 \\
0.0 & 5.2 & 0.0 & 5.4 & 5.5
\end{bmatrix}
\]

Compressed row storage

Two integer arrays \((ia, jcn)\) and a double array \(A\):

\[
\begin{align*}
ia & = [ 1 \ 2 \ 4 \ 6 \ 7 \ 10 ] \\
jcn & = [ 1 \ 2 \ 4 \ 1 \ 3 \ 4 \ 2 \ 4 \ 5 ] \\
A & = [1.1 \ 2.2 \ 2.4 \ 3.1 \ 3.3 \ 4.4 \ 5.2 \ 5.4 \ 5.5 ]
\end{align*}
\]

The nonzeros of the \(i\)th row are stored at the \(ia[i]...ia[i+1]-1\) positions of \(jcn\) and \(A\).

For example the 3rd row: starts at \(ia[3] = 4\) and finishes at \(ia[3+1]-1=5\). The column indices are therefore \(jcn[4,5]= 1 \ 3\) and values are \(A[4,5]=3.1 \ 3.3\).
There are many ways to store a sparse matrix.

Recall the two standard representations which store only the nonzero entries.

\[
\begin{bmatrix}
1.1 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 2.2 & 0.0 & 2.4 & 0.0 \\
3.1 & 0.0 & 3.3 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 4.4 & 0.0 \\
0.0 & 5.2 & 0.0 & 5.4 & 5.5
\end{bmatrix}
\]

**Compressed row format**

Two integer arrays (ia, jcn) and a double array \( A \):

\[
\begin{align*}
ia &= [1, 2, 4, 6, 7, 10] \\
jcn &= [1, 2, 4, 1, 3, 4, 2, 4, 5] \\
A &= [1.1, 2.2, 2.4, 3.1, 3.3, 4.4, 5.2, 5.4, 5.5]
\end{align*}
\]

The nonzeros of the \( i \)th row are stored at the \( ia[i] \ldots ia[i+1]-1 \) positions of \( jcn \) and \( A \).

Let matrix be of size \( m \times n \), and \( \tau \) be the number of nonzeros, then the storage is \( m + 1 + \tau \) integer and \( \tau \) double (or single or complex).
There are many ways to store a sparse matrix.

Recall the two standard representations which store only the nonzero entries.

\[
\begin{bmatrix}
1.1 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 2.2 & 0.0 & 2.4 & 0.0 \\
3.1 & 0.0 & 3.3 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 4.4 & 0.0 \\
0.0 & 5.2 & 0.0 & 5.4 & 5.5
\end{bmatrix}
\]

**Compressed column storage**

Two integer arrays \((\text{irn}, \text{ja})\) and a double array \(A\):

\[
\begin{align*}
\text{ja} &= [1, 3, 5, 6, 9, 10] \\
\text{irn} &= [1, 3, 2, 5, 3, 2, 4, 5, 5] \\
A &= [1.1, 3.1, 2.2, 5.2, 3.3, 2.4, 4.4, 5.4, 5.5]
\end{align*}
\]

The nonzeros of the \(j\)th column are stored at the \(\text{ja}[j]...\text{ja}[j+1]-1\) positions of \(\text{irn}\) and \(A\).

For example the 2nd col: starts at \(\text{ja}[2]=3\) and finishes at \(\text{ja}[2+1]-1=4\). The row indices are therefore \(\text{irn}[3,4]=2\ 5\) and values are \(A[3,4]=2.2\ 5.2\).
Sparse matrices: Compressed column storage

There are many ways to store a sparse matrix.

Recall the two standard representations which store only the nonzero entries.

$$\begin{bmatrix}
1.1 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 2.2 & 0.0 & 2.4 & 0.0 \\
3.1 & 0.0 & 3.3 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 4.4 & 0.0 \\
0.0 & 5.2 & 0.0 & 5.4 & 5.5
\end{bmatrix}$$

Compressed column format

Two integer arrays \((\text{irn}, \text{ja})\) and a double array \(A\):

\[
\begin{align*}
\text{ja} &= [1 \ 3 \ 5 \ 6 \ 9 \ 10] \\
\text{irn} &= [1 \ 3 \ 2 \ 5 \ 3 \ 2 \ 4 \ 5 \ 5] \\
A &= [1.1 \ 3.1 \ 2.2 \ 5.2 \ 3.3 \ 2.4 \ 4.4 \ 5.4 \ 5.5]
\end{align*}
\]

The nonzeros of the \(j\)th column are stored at the positions of \(\text{ja}[j] \ldots \text{ja}[j+1]-1\) \(\text{irn}\) and \(A\).

Let matrix be of size \(m \times n\), and \(\tau\) be the number of nonzeros, then the storage is \(n + 1 + \tau\) integer and \(\tau\) double (or single or complex).
Reminder: Dense matrix vector multiplication

Need to compute $y \leftarrow A x$ for an $m \times n$ dense matrix $A$ and suitable dense vectors $y$ and $x$.

**Row major order**

```
for i = 1 to m do
    y[i] ← 0.0
    for j = 1 to n do
        y[i] ← y[i] + A[i, j] * x[j]
```

**Column-major order**

```
for i = 1 to m do
    y[i] ← 0.0
    for j = 1 to n do
        for i = 1 to m do
            y[i] ← y[i] + A[i, j] * x[j]
```
Sparse matrices: Sparse matrix vector multiplies

Need to compute $y \leftarrow A x$ for an $m \times n$ sparse matrix $A$ and suitable dense vectors $y$ and $x$.

Compressed row storage

(\textit{ia}, \textit{jcn}, \textit{A})

\begin{verbatim}
for i = 1 to m do
    val $\leftarrow$ 0.0
    for k = \textit{ia}[i] to \textit{ia}[i + 1] − 1 do
        val $\leftarrow$ val + $A[k]$ * $x[jcn[k]]$
    y[i] $\leftarrow$ val
\end{verbatim}

Compressed column storage

(\textit{ja}, \textit{irn}, \textit{A})

\begin{verbatim}
for i = 1 to m do
    y[i] $\leftarrow$ 0.0
for j = 1 to n do
    xval $\leftarrow$ $x[j]$
    for k = \textit{ja}[j] to \textit{ja}[j + 1] − 1 do
        y[\textit{irn[k]]} $\leftarrow$ y[\textit{irn[k]}] + $A[k]$ * xval
\end{verbatim}

- Characterizes a wide range of applications with irregular computational dependency.
- Reduction operation from inputs (here entries of $x$) to outputs (here entries of $y$)
- A fine grain computation: each nnz is read/operated on once. Guaranteeing efficiency will guarantee efficiency in applications with a coarser grain computation.
Sparse matrices: Sparse matrix vector multiplies

SpMxV’s of the form $y \leftarrow Ax$ are the computational kernel of many scientific computations

- Solvers for linear systems, linear programs, eigensystems, least squares problems,
- Repeated SpMxV with the same large sparse matrix $A$,
- The matrix $A$ can be symmetric, unsymmetric, rectangular,
- Sometimes multiplies are of the form $y \leftarrow ADA^Tz$ with a diagonal $D$ (in interior point methods for linear programs).
  - computation proceeds (why?) as $w \leftarrow A^Tz$, then $x \leftarrow Dw$, then $y \leftarrow Ax$
- Sometimes we have multiplies with $A$ and $A^T$ independent; $y \leftarrow Ax$ and $w \leftarrow A^Tz$ (QMR, CGNE, and CGNR methods with square unsymmetric $A$; rectangular $A$ in Lanczos method).
- Most of the time the SpMxV’s are of the form $y \leftarrow AM^{-1}x$ (called preconditioning).
Iterative solvers: How do they look?

Basic form

while not converged do
  computations
  check convergence
EndDo

Computations are of the form:

- Linear vector operations
  \[ y \leftarrow \alpha x + y \quad \Rightarrow \quad y_i = \alpha x_i + y_i \]
- Inner products
  \[ \alpha \leftarrow (x, y) \quad \Rightarrow \quad \alpha = \sum x_i y_i \]
- Sparse matrix vector multiplies
  \[ y \leftarrow Ax \]
  \[ y \leftarrow A^T x \]

The Conjugate Gradient method

Compute \( r_0 := b - Ax_0, p_0 := r_0 \).

For \( j = 0, 1, \ldots, \) until convergence:

\[ \alpha_j := (r_j, r_j) / (Ap_j, p_j) \]
\[ x_{j+1} := x_j + \alpha_j p_j \]
\[ r_{j+1} := r_j - \alpha_j Ap_j \]
\[ \beta_j := (r_{j+1}, r_{j+1}) / (r_j, r_j) \]
\[ p_{j+1} := r_{j+1} + \beta_j p_j \]
EndDo

With certain types of preconditioners, we have SpMxV with another matrix \( M \) and/or \( M^T \). Replace \( Ap_j \) with \( AMp_j \).
Some iterative methods

- Stationary iterative methods: \( x^{(k)} \leftarrow Bx^{(k-1)} + c \)
  
  Jacobi, Gauss-Seidel, Successive overrelaxation (SOR).

- Non-stationary iterative methods (Krylov subspace methods): Computations involve information that changes at each iteration.

  Build and use Krylov subspace
  \( K_r(A, b) = \text{span } \{b, Ab, A^2b, \ldots, A^{r-1}b\} \).

  CG, MINRES, SYMMLQ, CGNE, GMRES, BiCG, QMR, CGS, BiCG-Stab, Chebyshev iterations, ...

Stationary ones are easier to use/implement, use less memory, have small cost per iteration. Slow convergence, if possible at all.

Nonstationary ones easier to implement, use less memory than the direct methods. Requires preconditioning.

These methods are usually less robust (than the direct methods).
Some iterative methods


<table>
<thead>
<tr>
<th>Method</th>
<th>Inner Product</th>
<th>$\text{SAXPY}$</th>
<th>Matrix-Vector Product</th>
<th>Precond Solve</th>
</tr>
</thead>
<tbody>
<tr>
<td>JACOBI</td>
<td></td>
<td>$1^a$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GS</td>
<td></td>
<td>$1^a$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOR</td>
<td></td>
<td>$1^a$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CG</td>
<td>$2$</td>
<td>$3$</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>GMRES</td>
<td>$i + 1$</td>
<td>$i + 1$</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>BiCG</td>
<td>$2$</td>
<td>$5$</td>
<td>$1/1$</td>
<td>$1/1$</td>
</tr>
<tr>
<td>QMR</td>
<td>$2$</td>
<td>$8 + 4^{bc}$</td>
<td>$1/1$</td>
<td>$1/1$</td>
</tr>
<tr>
<td>CGS</td>
<td>$2$</td>
<td>$6$</td>
<td>$2$</td>
<td>$2$</td>
</tr>
<tr>
<td>Bi-CGSTAB</td>
<td>$4$</td>
<td>$6$</td>
<td>$2$</td>
<td>$2$</td>
</tr>
<tr>
<td>CHEBYSHEV</td>
<td>$2$</td>
<td>$1$</td>
<td>$1$</td>
<td></td>
</tr>
</tbody>
</table>

$^a$This method performs no real matrix vector product or preconditioner solve, but the number of operations is equivalent to a matrix-vector multiply.

$^b$True $\text{SAXPY}$ operations + vector scalings.

$^c$Less for implementations that do not recursively update the residual.
Outline

1. Introduction to SpMxV

2. Parallel SpMxV
   - Row parallel
   - Column parallel
   - Row-column parallel

3. Hypergraphs and hypergraph partitioning
   - Hypergraph models for row-parallel SpMxV
   - Hypergraph models for column-parallel SpMxV
   - Hypergraph models for row-column parallel SpMxV
   - Some other partitioning problems
Parallel sparse matrix vector multiplies $y \leftarrow Ax$

We look at the distributed memory setting.

- nonzeros in $A$ are distributed,
- the input vector entries, $x_j$s, are distributed,
- the output vector entries, $y_i$s, are distributed (that is, the responsibility of storing them is decided).

What are the aims of a distribution?

- load balance among processors: equal number of $a_{ij}$ per processor,
- reduced communication requirement:
  - $a_{ij}$ is to be multiplied by $x_j$; these two should meet at a processor;
  - the scalar product $a_{ij}x_j$ is a contribution to $y_i$; the result of the product $a_{ij}x_j$ and the vector entry $y_i$ should meet at a processor.
Parallelization objectives

**Achieve load balance**

Load of a processor: number of nonzeros.  
⇒ assign almost equal number of nonzeros per processor.

**Minimize communication cost**

Communication cost is a complex function (depends on the machine architecture and problem size):

- total volume of messages,
- total number of messages,
- max. volume of messages per processor (sends or receives, both?),
- max. number of messages per processor (sends or receives, both?).
Parallel sparse matrix vector multiplies

We restrict ourselves to the distributed memory setting.

What are the aims of a distribution?

- load balance among processors: equal number of $a_{ij}$ per processor,
- reduced communication requirement: $a_{ij}$ is to be multiplied by $x_j$; these two should meet at a processors; the scalar product $a_{ij}x_j$ is a contribution to $y_i$; the result $a_{ij}x_j$ and $y_i$ should meet at a processor.

Assume there are no operations between $x$ and $y$ of the SpMxV $y \leftarrow Ax$ after the multiply operation.

In half of the cases(!), the input vector $x$ and the output vector $y$ undergo vector operations (such as $y \leftarrow \alpha x + y$, or $\gamma \leftarrow x^T y$), in such cases it is better to have the same partition on $x$ and $y$—we will come to this later.
Parallel sparse matrix vector multiplies: Variants

We classify parallel SpMxV algorithms into three groups (according to the distribution on the matrix)

- **Row-parallel** algorithm: all nonzeros in a row of the matrix is assigned to the same processor ($a_{ij}$ and $a_{ik}$ are in the same processor),

- **Column-parallel** algorithm: all nonzeros in a column of the matrix is assigned to the same processor ($a_{ij}$ and $a_{kj}$ are in the same processor).

- **Row-column parallel** algorithm: many possibilities
  - each nonzero is assigned to a processor on its own ($a_{ij}$ and $a_{ik}$ can be in different processors; $a_{ij}$ and $a_{kj}$ can be in different processors),
  - the nonzeros in a row and/or column are assigned to a small set of processors (e.g., assume a 2D mesh of processors and distribute the nonzeros in a row of $A$ among the processors of a row of the mesh).
Outline

1. Introduction to SpMxV

2. Parallel SpMxV
   - Row parallel
   - Column parallel
   - Row-column parallel

3. Hypergraphs and hypergraph partitioning
   - Hypergraph models for row-parallel SpMxV
   - Hypergraph models for column-parallel SpMxV
   - Hypergraph models for row-column parallel SpMxV
   - Some other partitioning problems
Parallel SpMxV: Row-parallel

The rows and columns of an $m \times n$ matrix $A$ are permuted into a $K \times K$ block structure

\[
A_{BL} = \begin{bmatrix}
A_{11} & A_{12} & \cdots & A_{1K} \\
A_{21} & A_{22} & \cdots & A_{2K} \\
\vdots & \vdots & \ddots & \vdots \\
A_{K1} & A_{K2} & \cdots & A_{KK}
\end{bmatrix}
\]

for rowwise partitioning, where $K$ is the number of processors.

- Block $A_{k\ell}$ is of size $m_k \times n_\ell$, where $\sum_k m_k = m$ and $\sum_\ell n_\ell = n$.
- Processor $P_k$ holds the $k$th row stripe $[A_{k1} \cdots A_{kK}]$ of size $m_k \times n$.
- Load balance: The row stripes should have nearly equal number of nonzeros for having the computational load balance among processors.
In $y \leftarrow Ax$, $y$ and $x$ are column vectors of size $m$ and $n$; $A$ is partitioned as shown in the previous slide.

- A rowwise partition of matrix $A$ defines a partition on the output vector $y$.

- The input vector $x$ is assumed to be partitioned conformably with the column permutation of matrix $A$.

- $y$ and $x$ vectors are partitioned as $y = [y_1^T \cdots y_K^T]^T$ and $x = [x_1^T \cdots x_K^T]^T$, where $y_k$ and $x_k$ are column vectors of size $m_k$ and $n_k$, respectively.

- Processor $P_k$ holds $x_k$ and is responsible for computing $y_k$. 
Parallel SpMxV: Row-parallel

Matrix is partitioned rowwise among 4 processors.

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1$</td>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_3$</td>
<td>$x_4$</td>
</tr>
<tr>
<td>$y_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- row stripes are assigned to processors.
- virtual column stripes shows the assignment of $x$ vector entries.
- The columns of the matrix are permuted according to the partition on $x$.

25 nonzeros in the 1st row stripe (assigned to processor $P_1$)
26 nonzeros in the 2nd row stripe (assigned to processor $P_2$)
25 nonzeros in the 3rd row stripe (assigned to processor $P_3$)
25 nonzeros in the 4th row stripe (assigned to processor $P_4$)
Parallel SpMxV: Row-parallel algorithm

Executes the following steps at each processor $P_k$:

1. For each nonzero off-diagonal block $A_{\ell k}$, send sparse vector $\hat{x}_{\ell k}$ to processor $P_\ell$, where $\hat{x}_{\ell k}$ contains only those entries of $x_k$ corresponding to the nonzero columns in $A_{\ell k}$.

2. Compute the diagonal block product $y_{k k}^k \leftarrow A_{kk} \times x_k$, and set $y_k = y_{k k}^k$.

3. For each nonzero off-diagonal block $A_{k \ell}$, receive $\hat{x}_{k \ell}$ from processor $P_\ell$, then compute $y_{\ell k}^\ell \leftarrow A_{k \ell} \times \hat{x}_{k \ell}^k$, and update $y_k \leftarrow y_k + y_{\ell k}^\ell$.

In Step 1, $P_k$ might be sending the same $x_k$-vector entry to different processors according to the sparsity pattern of the respective column of $A$. This multicast-like operation is called the expand operation.
Parallel SpMxV: Row-parallel

Matrix is partitioned rowwise among 4 processors. $y$ vector entries are partitioned according to the rowwise partition of $A$; assume the $x$ vector entries are partitioned and the columns of $A$ are permuted.

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>1 2 3 4 5 6 7</td>
<td>8 9 10 11 12</td>
<td>13 14 15 16</td>
<td></td>
</tr>
<tr>
<td>$x_2$</td>
<td>2 3 4 5 6 7 8</td>
<td>9 10 11 12 13</td>
<td>14 15 16 17</td>
<td></td>
</tr>
<tr>
<td>$x_3$</td>
<td>3 4 5 6 7 8 9</td>
<td>10 11 12 13 14</td>
<td>15 16 17 18</td>
<td></td>
</tr>
<tr>
<td>$x_4$</td>
<td>4 5 6 7 8 9 10</td>
<td>11 12 13 14 15</td>
<td>16 17 18 19</td>
<td></td>
</tr>
</tbody>
</table>

1. Expand $x$ vector (sends/receives)
2. Compute with diagonal blocks
3. Receive $x$ and compute with off-diagonal blocks
Fact 1: Number of messages sent by $P_k$

The number of messages sent by processor $P_k$ is equal to the number of nonzero off-diagonal blocks in the $k$th virtual column stripe of $A$.

- $P_2$, sends $x[12:14]$ to $P_3$—nonzero columns 12, 13, and 14 in $A_{32}$.
- $P_2$ sends $x[12]$ to $P_4$—nonzero column 12 in $A_{42}$.
- The number of messages sent by $P_2$ is 2.
### Row-parallel SpMxV: Communication requirements

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1$</td>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_3$</td>
<td>$x_4$</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Expand $x$ vector (sends/receives)
2. Compute with diagonal blocks
3. Receive $x$ and compute with off-diagonal blocks

**Fact 2: Volume of messages sent by $P_k$**

The volume of messages sent by $P_k$ is equal to the sum of the number of nonzero columns in the off-diagonal blocks in the $k$th virtual column stripe.

- $P_2$, sends $x[12:14]$ to $P_3$—(size 3).
- $P_2$ sends $x[12]$ to $P_4$—(size 1).
- The volume of messages sent by $P_2$ is 4.
Row-parallel SpMxV: Communication requirements

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>1</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>$x_2$</td>
<td>✗</td>
<td>2</td>
<td>3</td>
<td>✗</td>
</tr>
<tr>
<td>$x_3$</td>
<td>✗</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>$x_4$</td>
<td>✗</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Fact 3: Total volume and number of messages

- The total volume of messages is equal to the number of nonzero columns in off-diagonal blocks.
- The total number of messages is equal to the number of nonzero off-diagonal blocks.

- Total volume of messages is 13.
- Total number of messages is 9.

1. Expand $x$ vector (sends/receives)
2. Compute with diagonal blocks
3. Receive $x$ and compute with off-diagonal blocks
Outline

1 Introduction to SpMxV

2 Parallel SpMxV
   - Row parallel
   - Column parallel
   - Row-column parallel

3 Hypergraphs and hypergraph partitioning
   - Hypergraph models for row-parallel SpMxV
   - Hypergraph models for column-parallel SpMxV
   - Hypergraph models for row-column parallel SpMxV
   - Some other partitioning problems
Parallel SpMxV: Column-parallel

The rows and columns of an $m \times n$ matrix $A$ are permuted into a $K \times K$ block structure

$$
\begin{bmatrix}
A_{11} & A_{12} & \cdots & A_{1K} \\
A_{21} & A_{22} & \cdots & A_{2K} \\
\vdots & \vdots & \ddots & \vdots \\
A_{K1} & A_{K2} & \cdots & A_{KK}
\end{bmatrix}
$$

for columnwise partitioning, where $K$ is the number of processors.

- Block $A_{k\ell}$ is of size $m_k \times n_\ell$, where $\sum_k m_k = m$ and $\sum_\ell n_\ell = n$.
- Processor $P_k$ holds the $k$th column stripe $[A_{1k}^T \cdots A_{Kk}^T]^T$ of size $m \times n_k$.
- **Load balance**: The column stripes should have nearly equal number of nonzeros for having the computational load balance among processors.
In $y \leftarrow A x$, $y$ and $x$ are column vectors of size $m$ and $n$; $A$ is partitioned as shown in the previous slide.

- A columnwise partition of matrix $A$ defines a partition on the input-vector $x$.
- The output vector $y$ is assumed to be partitioned conformably with the row permutation of matrix $A$.
- $y$ and $x$ vectors are partitioned as $y = [y_1^T \cdots y_K^T]^T$ and $x = [x_1^T \cdots x_K^T]^T$, where $y_k$ and $x_k$ are column vectors of size $m_k$ and $n_k$, respectively.
- Processor $P_k$ holds $x_k$ and is responsible for computing $y_k$. 
Matrix is partitioned columns among 4 processors.

<table>
<thead>
<tr>
<th></th>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
<th>P₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>y₁</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y₂</td>
<td>8</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y₃</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y₄</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Column stripes are assigned to processors.
- Virtual row stripes shows the assignment of $y$ vector entries.
- The rows of the matrix are permuted according to the partition on $y$.
- Load balance achieved:
  - 25 nonzeros assigned to processor $P₁$;
  - 26 nonzeros assigned to processor $P₂$;
  - 25 nonzeros assigned to processor $P₃$;
  - 25 nonzeros assigned to processor $P₄$.
Parallel SpMxV: Column-parallel algorithm

Executes the following steps at each processor $P_k$:

1. For each nonzero off-diagonal block $A_{\ell k}$, form sparse vector $\hat{y}_{\ell}^k$ which contains only those results of $y_{\ell}^k = A_{\ell k} \times x_k$ corresponding to the nonzero rows in $A_{\ell k}$. Send $\hat{y}_{\ell}^k$ to processor $P_\ell$.

2. Compute the diagonal block product $y_k^k \leftarrow A_{kk} \times x_k$, and set $y_k = y_k^k$.

3. For each nonzero off-diagonal block $A_{k \ell}$ receive partial-result vector $\hat{y}_{k}^\ell$ from processor $P_\ell$, and update $y_k \leftarrow y_k + \hat{y}_{k}^\ell$.

In Step 3, the multinode accumulation on the $y_k$-vector entries is called the fold operation.
Matrix is partitioned columnwise among 4 processors. $x$ vector entries are partitioned according to the columnwise partition of $A$; assume the $y$ vector entries are partitioned and the rows of $A$ are permuted.

1. Compute with off diagonal blocks; obtain partial $y$ results, issue sends/receives
2. Compute with diagonal block
3. Receive partial results on $y$ for nonzero off-diagonal blocks and add the partial results
Column-parallel SpMxV: Communication requirements

Fact 1: Number of messages sent by $P_k$

The number of messages sent by processor $P_k$ in column-parallel $y \leftarrow Ax$ is equal to the number of nonzero off-diagonal blocks in the $k$th column stripe of $A$.

- $P_3$ sends a message to $P_2$ for $y$ vector entries $y[12, 13, 14]$ and to $P_4$ for $y[25, 26]$.
- $P_4$ sends messages to $P_1$, $P_2$, and $P_3$. 
Column-parallel SpMxV: Communication requirements

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fact 2: Volume of messages sent by $P_k$

The volume of messages sent by $P_k$ is equal to the sum of the number of nonzero rows in each off-diagonal block in the kth column stripe of $A$.

- $P_3$ sends a message to $P_2$ for $y$ vector entries $y[12, 13, 14]$ and another one to $P_4$ for $y[25, 26]$.
- $P_3$ sends 5 units of messages.
Column-parallel SpMxV: Communication requirements

1. Compute with off diagonal blocks; obtain partial $y$ results, issue sends/receives
2. Compute with diagonal block
3. Receive partial results on $y$ for nonzero off-diagonal blocks and add the partial results

Fact 3: Total volume and number of messages

- The total volume of messages is equal to the number of nonzero rows in off-diagonal blocks. (13)
- The total number of messages is equal to the number of nonzero off-diagonal blocks. (9)
Parallel SpMxV: Row parallel and column parallel algorithms

The communication patterns of column parallel $y \leftarrow A^T x$ and row parallel $y \leftarrow Ax$ are duals of each other (the columnwise partition on $A^T$ is equal to the rowwise partition on $A$).
Outline

1. Introduction to SpMxV

2. Parallel SpMxV
   - Row parallel
   - Column parallel
   - Row-column parallel

3. Hypergraphs and hypergraph partitioning
   - Hypergraph models for row-parallel SpMxV
   - Hypergraph models for column-parallel SpMxV
   - Hypergraph models for row-column parallel SpMxV
   - Some other partitioning problems
Parallel SpMxV: Row-column parallel algorithm

Consider \( y \leftarrow A x \), where \( y \) and \( x \) are column vectors of size \( m \) and \( n \), respectively, and the matrix is partitioned in two dimensions among \( K \) processors.

- The vectors \( y \) and \( x \) are partitioned as \( y = [y_1^T \cdots y_K^T]^T \) and \( x = [x_1^T \cdots x_K^T]^T \), where \( y_k \) and \( x_k \) are column vectors of size \( m_k \) and \( n_k \), respectively. As before we have \( \sum_k m_k = m \) and \( \sum_\ell n_\ell = n \).
- Processor \( P_k \) holds \( x_k \) and is responsible for computing \( y_k \).
- Nonzeros of a processor \( P_k \) can be visualized as a sparse matrix \( A^k \)

\[
A^k = \begin{bmatrix}
A_{11}^k & \cdots & A_{1k}^k & \cdots & A_{1K}^k \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
A_{k1}^k & \cdots & A_{kk}^k & \cdots & A_{kK}^k \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
A_{K1}^k & \cdots & A_{Kk}^k & \cdots & A_{KK}^k
\end{bmatrix}
\]

of size \( m \times n \), where \( A = \sum A^k \) (here \( A^k \)'s are disjoint).
Parallel SpMxV: Row-column parallel algorithm

1. Expand $x$ vector
2. Scalar multiply and add
   \[ y_i \leftarrow a_{ij} x_j + a_{ik} x_k \]
3. Fold on $y$ vector
   (send and receive partial results)

Load balance is achieved by assigning almost equal number of nonzeros to the processors.
Row-column-parallel SpMxV: Communication requirements

- Communication on $x$ (expand operations)
  
  Same as that in the row-parallel algorithm

- Communication on $y$ (fold operations)
  
  Same as that in the column-parallel algorithm
Running time comparisons from Vastenhouw and Bisseling’05

Table 5.8 Communication volume (in data words) and time (in ms) of parallel sparse matrix-vector multiplication on an SGI Origin 3800. The lowest volume and time are marked in boldface.

<table>
<thead>
<tr>
<th>Name</th>
<th>p</th>
<th>Volume</th>
<th></th>
<th></th>
<th></th>
<th>Time</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1D row</td>
<td>1D col</td>
<td>2D</td>
<td>1D row</td>
<td>1D col</td>
<td>2D</td>
</tr>
<tr>
<td>tbdmatlab</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5.74</td>
<td>5.71</td>
<td>5.77</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5056</td>
<td>6438</td>
<td>5056</td>
<td>3.28</td>
<td>3.31</td>
<td>3.20</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>14650</td>
<td>14949</td>
<td>11005</td>
<td>2.08</td>
<td>2.06</td>
<td>1.95</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>30982</td>
<td>26804</td>
<td>17792</td>
<td>1.62</td>
<td>1.40</td>
<td>1.34</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>56923</td>
<td>42291</td>
<td>27735</td>
<td>1.34</td>
<td>1.19</td>
<td>1.17</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>98791</td>
<td>62410</td>
<td>40497</td>
<td>1.77</td>
<td>1.58</td>
<td>1.70</td>
</tr>
<tr>
<td>tbdlinux</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>67.55</td>
<td>67.61</td>
<td>74.15</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>15764</td>
<td>24463</td>
<td>15764</td>
<td>36.65</td>
<td>32.26</td>
<td>32.16</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>42652</td>
<td>54262</td>
<td>30444</td>
<td>14.06</td>
<td>12.22</td>
<td>12.14</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>90919</td>
<td>96038</td>
<td>49120</td>
<td>6.49</td>
<td>6.35</td>
<td>6.62</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>177347</td>
<td>155604</td>
<td>75884</td>
<td>5.22</td>
<td>4.22</td>
<td>4.20</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>297658</td>
<td>227368</td>
<td>106563</td>
<td>4.32</td>
<td>4.08</td>
<td>3.23</td>
</tr>
<tr>
<td>bcsstk30</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>50.99</td>
<td>50.96</td>
<td>56.18</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>948</td>
<td>948</td>
<td>940</td>
<td>28.37</td>
<td>28.24</td>
<td>26.04</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2099</td>
<td>2099</td>
<td>2124</td>
<td>6.00</td>
<td>6.03</td>
<td>5.83</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>5019</td>
<td>5019</td>
<td>4120</td>
<td>2.87</td>
<td>2.90</td>
<td>2.88</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>9344</td>
<td>9344</td>
<td>8491</td>
<td>1.53</td>
<td>1.56</td>
<td>1.64</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>15593</td>
<td>15593</td>
<td>14771</td>
<td>1.08</td>
<td>1.12</td>
<td>1.17</td>
</tr>
</tbody>
</table>
Outline

1. Introduction to SpMxV

2. Parallel SpMxV
   - Row parallel
   - Column parallel
   - Row-column parallel

3. Hypergraphs and hypergraph partitioning
   - Hypergraph models for row-parallel SpMxV
   - Hypergraph models for column-parallel SpMxV
   - Hypergraph models for row-column parallel SpMxV
   - Some other partitioning problems
Parallelization objectives

Achieve load balance

Load of a processor: number of nonzeros.
⇒ assign almost equal number of nonzeros per processor.

Minimize communication cost

Communication cost is a complex function (depends on the machine architecture and problem size):
- total volume of messages,
- total number of messages,
- max. volume of messages per processor (sends or receives, both?),
- max. number of messages per processor (sends or receives, both?).

A common metric in different studies: total volume of communication.
A hypergraph is two-tuple $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ where $\mathcal{V}$ is a set of vertices and $\mathcal{N}$ is a set of hyperedges.

A hyperedge $h \in \mathcal{N}$ is a subset of vertices. We call them nets for short.

A cost $c(h)$ is associated with each net $h$.

A weight $w(v)$ is associated with each vertex $v$.

An undirected graph can be seen as a hypergraph where each net contains exactly two vertices.
Hypergraphs: Example

\[ \mathcal{H} = (\mathcal{V}, \mathcal{N}) \text{ with } \mathcal{V} = \{1, 2, 3, 4, 5\} \mathcal{N} = \{n_1, n_2, n_3\}\text{ where } \]
\[ n_1 = \{1, 3, 4\} \quad n_2 = \{1, 2, 3, 4\} \quad n_3 = \{2, 5\} \]
Hypergraphs: Partitioning

Partition

\[ \Pi = \{ V_1, V_2, \ldots, V_K \} \] is a \( K \)-way vertex partition if

- \( V_k \neq \emptyset \),
- parts are mutually exclusive: \( V_k \cap V_\ell = \emptyset \),
- parts are collectively exhaustive: \( V = \bigcup V_k \).

In \( \Pi \), a net connects a part if it has at least one vertex in that part, i.e., \( h \) connects \( V_k \) if \( h \cap V_k \neq \emptyset \).

The connectivity \( \lambda(h) \) of a net is equal to the number of parts connected by \( h \).

Objective: minimize cutsizes(\( \Pi \))

\[ \sum_h c(h)(\lambda(h) - 1), \]

Constraint: balanced part weights

\[ \sum_{v \in V_k} w(v) \leq (1 + \varepsilon) \frac{\sum_{v \in V} w(v)}{K}. \]

Hypergraph partitioning problem is NP-complete.
Hypergraphs partitioning: Example

\( \mathcal{H} = (\mathcal{V}, \mathcal{N}) \) with 10 vertices and 4 nets, partitioned into four parts.

- \( V_1 = \{4, 5\} \)
- \( V_2 = \{7, 10\} \)
- \( V_3 = \{3, 8, 9\} \)
- \( V_4 = \{1, 2, 6\} \)

<table>
<thead>
<tr>
<th>net</th>
<th>connectivity–1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2–1=1</td>
</tr>
<tr>
<td>2</td>
<td>3–1=2</td>
</tr>
<tr>
<td>3</td>
<td>3–1=2</td>
</tr>
<tr>
<td>4</td>
<td>2–1=1</td>
</tr>
</tbody>
</table>

Cutsize: 6
**Introduction to SpMxV**

- Parallel SpMxV

**Hypergraphs and hypergraph partitioning**

**Hypergraphs: Partitioning tools and applications**

**Tools**

- **hMETIS** (Karypis and Kumar, Univ. Minnesota),
- **MLPart** (Caldwell, Kahng, and Markov, UCLA/UMich),
- **Mondrian** (Bisseling and Meesen, Utrecht Univ.),
- **Parkway** (Trifunovic and Knottenbelt, Imperial Coll. London),
- **PaToH** (Çatalyürek and Aykanat, Bilkent Univ.),
- **Zoltan-PHG** (Devine, Boman, Heaphy, Bisseling, and Çatalyürek, Sandia National Labs.).

**Applications**

- VLSI: circuit partitioning,
- Scientific computing: matrix partitioning, ordering, cryptology, etc.,
- Parallel/distributed computing: volume rendering, data aggregation, scheduling, declustering/clustering,
- Software engineering, information retrieval, processing spatial join queries, etc.
In all of the cases we will see, we will have unit net-costs, that is \( c(h) = 1 \) The objective function becomes

\[
\sum_{h} (\lambda(h) - 1)
\]

- Make the data to be partitioned as vertices of the hypergraph.
- Assign weights to the vertices.
- Put nets to represent dependencies of computations to the input data; and dependencies of output data into computations.
- Partition into \( K \) parts, each \( V_k \) holds the data of a processor.
- Load balance would be achieved if part weights are balanced.
- Total volume of communication would be equivalent to the cut-size.
Outline

1. Introduction to SpMxV

2. Parallel SpMxV
   - Row parallel
   - Column parallel
   - Row-column parallel

3. Hypergraphs and hypergraph partitioning
   - Hypergraph models for row-parallel SpMxV
   - Hypergraph models for column-parallel SpMxV
   - Hypergraph models for row-column parallel SpMxV
   - Some other partitioning problems
Hypergraph models for row-parallel SpMxV

- Three entities to partition $y$, rows of $A$, and $x$
  - three types of vertices $y_i$, $r_i$, and $x_j$

- Assign vertex weights
  - weight of $r_i$ is equal to the number of nnz in row $i$
  - weight of $y_i$ and $x_j$ can be set to zero.

- $y_i$ is computed by a single row, that is $r_i$
  - represent the dependency of $y_i$ on $r_i$

- $x_j$ is a data source; $r_i$s where $a_{ij} \neq 0$ need $x_j$
  - connect $x_j$ and all such $r_i$
Row-parallel SpMxV: Communication requirements

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_3$</td>
<td>$x_4$</td>
<td></td>
</tr>
<tr>
<td>$P_1 y_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_2 y_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_3 y_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>$P_4 y_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

1. Expand $x$ vector (sends/receives)
2. Compute with diagonal blocks
3. Receive $x$ and compute with off-diagonal blocks

Total volume and number of messages

The total volume of messages is equal to the number of nonzero columns in off-diagonal blocks. Here, the total volume of messages is 13.
Hypergraph models for row-parallel SpMxV

Elementary hypergraph model for 1D rowwise partitioning

Combine $y_i$ and $r_i$: owner computes rule

Partition the vertices into $K$ parts
(partition the data among $K$ processors)

Part weights = processor’s loads in terms of nnz.
**Hypergraph models for row-parallel SpMxV**

Number of nonzeros columns in off-diagonal blocks is 5. Total volume is 5.

Column-net $c_{14}$ connects 2 parts; $c_5$ connects 3 parts; $c_{12}$ connects 2 parts; $c_{13}$ connects 2 parts. Cut-size is 5.
Hypergraph models for row-parallel SpMxV

What about load balance?

There are 12 nnz in the first row stripe.

Each row-vertex gets a weight equivalent to the number of nonzeros in the associated row of $A$. 
Outline

1. Introduction to SpMxV

2. Parallel SpMxV
   - Row parallel
   - Column parallel
   - Row-column parallel

3. Hypergraphs and hypergraph partitioning
   - Hypergraph models for row-parallel SpMxV
   - Hypergraph models for column-parallel SpMxV
   - Hypergraph models for row-column parallel SpMxV
   - Some other partitioning problems
Hypergraph models for column-parallel SpMxV

- Three entities to partition $y$, columns of $A$, and $x$
  - three types of vertices $y_i$, $c_j$, and $x_j$

- Assign vertex weights
  - weight of $c_j$ is equal to the number of nnz in column $j$
  - weight of $y_i$ and $x_j$ can be set to zero.

- $x_j$ is needed by a single column, that is $c_j$
  - represent the need of $c_j$ on $x_j$

- $y_i$ is computed by contributions from different columns; each column $c_j$ with $a_{ij} \neq 0$ contributes to $y_i$
  - connect $y_i$ and all such $c_j$
Column-parallel SpMxV: Communication requirements

1. Compute with off diagonal blocks; obtain partial $y$ results, issue sends/receives
2. Compute with diagonal block
3. Receive partial results on $y$ for nonzero off-diagonal blocks and add the partial results

Total volume and number of messages

The total volume of messages is equal to the number of nonzero rows in off-diagonal blocks. (13)

\[
\begin{array}{|c|c|c|c|c|}
\hline
 & P_1 & P_2 & P_3 & P_4 \\
\hline
x_1 & \times & \times & \times & \times \\
\hline
x_2 & \times & \times & \times & \times \\
\hline
x_3 & \times & \times & \times & \times \\
\hline
x_4 & \times & \times & \times & \times \\
\hline
\end{array}
\]
For column-parallel $w \leftarrow Az$ computations.

Elementary hypergraph model for 1D colwise partitioning.

Combine $c_j$ and $z_j$; one column needs only one $z$-vector entry.

Partition the vertices into $K$ parts (partition the data among $K$ processors). Part weights = processor loads in terms of number of nonzeros.
The computation is $w \leftarrow Az$

Number of nonzeros rows in off-diagonal blocks is 5. Total volume is 5.

Row-net $r_{13}$ connects 2 parts; $r_1$ connects 2 parts; $r_9$ connects 2 parts; $r_4$ connects 3 parts. Cut-size is 5.
Hypergraph models for column-parallel SpMxV

What about load balance?

The computation is $w \leftarrow A z$

There are 11 nonzeros in the second column stripe.

Each column-vertex gets a weight equal to the number of nonzeros in the associated column of $A$. 
Outline

1 Introduction to SpMxV

2 Parallel SpMxV
   • Row parallel
   • Column parallel
   • Row-column parallel

3 Hypergraphs and hypergraph partitioning
   • Hypergraph models for row-parallel SpMxV
   • Hypergraph models for column-parallel SpMxV
   • Hypergraph models for row-column parallel SpMxV
   • Some other partitioning problems
Hypergraph models for row-column parallel SpMxV

- Three entities to partition $y$, nonzeros of $A$, and $x$
  - three types of vertices $y_i$, $c_j$, and $a_{ij}$

- Assign vertex weights
  - weight of $a_{ij}$-vertex is equal to 1.
  - weight of $y_i$ and $x_j$ can be set to zero.

- $x_j$ is needed by all $a_{ij} \neq 0$
  - connect $x_j$ and all such $a_{ij}$

- $y_i$ is computed by contributions from different nonzeros;
  - each $a_{ij} \neq 0$ contributes to $y_i$
  - connect $y_i$ and all such $a_{ij}$
Parallel SpMxV: Row-column parallel algorithm

1. Expand \( x \) vector
2. Scalar multiply and add
   \[ y_i \leftarrow a_{ij}x_j + a_{ik}x_k \]
3. Fold on \( y \) vector
   (send and receive partial results)

Communication on \( x \) (expand operations): Same as that in the row-parallel algorithm.

Communication on \( y \) (fold operations): Same as that in the column-parallel algorithm.
For row-column-parallel $y \leftarrow A x$ computations.

Elementary hypergraph model for row-column-parallel algorithm

Partition the vertices into $K$ parts (partition the data among $K$ processors).
Part weights = processor loads in terms of nonzeros.
Hypergraph models for row-column parallel SpMxV

For row-column-parallel $y \leftarrow Ax$ computations.

[Part of the fine grain model on the board....]
Outline

1. Introduction to SpMxV
2. Parallel SpMxV
   - Row parallel
   - Column parallel
   - Row-column parallel
3. Hypergraphs and hypergraph partitioning
   - Hypergraph models for row-parallel SpMxV
   - Hypergraph models for column-parallel SpMxV
   - Hypergraph models for row-column parallel SpMxV
   - Some other partitioning problems
The approach

- Put vertices to represent the data items to partition
- Put nets to represent dependencies and needs
- Assign weights to vertices to have load balance
- Try to simplify (not lose the flexibility) by
  - if two data items want to be in the same processors, amalgamate the vertices
  - if there are nets of size 1, remove them.
- We can specify for a set of vertices to which part it should be assigned; if this is imposed by the problem that we want to parallelize.
The approach on row parallel algorithm: Symmetric partitioning wanted

\( y_i \) and \( r_i \) wants to be in the same part processor (owner computes rule—avoids communication).

net \( n_y(i) \) has size 1 after amalgamation; remove it from the model. Some \( n_x(i) \) may have single vertex (in which case?)—they can be removed too.
Problem 1

Problem

Describe a hypergraph model which can be used to partition the matrix $A$ rowwise for the $y \leftarrow Ax$ computations under given, possibly different, partitions on the input and output vectors $x$ and $y$.

A parallel algorithm that carries out the $y \leftarrow Ax$ computations under given partitions of $x$ and $y$ should have a communication phase on $x$, a computation phase, and a communication phase on $y$. 
Solution to Problem 1

Problem

Describe a hypergraph model which can be used to partition the matrix $A$ rowwise for the $y \leftarrow Ax$ computations under given, possibly different, partitions on the input and output vectors $x$ and $y$.

Solution

Take the elementary model and fix the vertices $x_j$ and $y_i$ to the parts as specified by the given partitions.
Problem 2

Problem

Describe a hypergraph model to obtain the same partition on the input and output vectors $x$ and $y$ which is (can be) different than the partition on the rows of $A$ for the $y ← Ax$ computations.

The previous parallel algorithm will be used.
Solution to Problem 2

Problem

Describe a hypergraph model to obtain the same partition on the input and output vectors $x$ and $y$ which is different than the partition on the rows of $A$ for the $y \leftarrow Ax$ computations.

Solution

Take the elementary model and amalgamate the vertices $x_i$ and $y_i$.
Problem 3

Describe a hypergraph model to obtain different partitions on $x$ and on the rows of $A$, where $y$ is partitioned conformably with the rows of $A$ under the owner-computes rule for computations of the form $y \leftarrow Ax$ followed by $x \leftarrow x + y$. 
Solution to Problem 3

(a) Elementary model for $y \leftarrow A x$

(b) New vertices for $x_i \leftarrow x_i + y_i$ and the dependencies for them.

(c) Owner computes rule for $x_i \leftarrow x_i + y_i$

(d) Owner computes rule for $y_i$.
Problem 4: Preconditioned iterative methods

- Iterative methods may converge slowly, or diverge
- transform $A\mathbf{x} = b$ to another system that is easier to solve
- Preconditioner is a matrix that helps in obtaining desired transformation
Problem 4: Preconditioned iterative methods

- We consider parallelization of iterative methods that use approximate inverse preconditioners.
- Approximate inverse is a matrix $M$ such that $AM \approx I$.
- Instead of solving $Ax = b$, use right preconditioning and solve $AMy = b$.

and then set

$$x = My$$
Problem 4: Preconditioned iterative methods

- Additional SpMxV operations with $M$
  never form the matrix $AM$; perform successive SpMxVs
- Parallelizing a full step in these methods requires efficient SpMxV operations with $A$ and $M$
  partition $A$ and $M$
- A blend of dependencies and interactions among matrices and vectors
  partition $A$ and $M$ simultaneously
Problem 4: Preconditioned iterative methods

- Partition $A$ and $M$ simultaneously
- Figure out partitioning requirements through analyzing linear vector operations and inner products
  
  Reminder: never communicate vector entries for these operations
- Different methods have different partitioning requirements
Problem 4: Preconditioned iterative methods

Preconditioned BiCG-STAB

\[ p^i = r^{i-1} + \beta_{i-1} \left( p^{i-1} - \omega_{i-1} v^{i-1} \right) \]

\[ \hat{p} = Mp^i \]

\[ v^i = A\hat{p} \]

\[ s = r^{i-1} - \alpha_i v^i \]

\[ \hat{s} = Ms \]

\[ t = A\hat{s} \]

\[ \omega_i = \langle t, s \rangle / \langle t, t \rangle \]

\[ x^i = x^{i-1} + \alpha_i p^i + \omega_i s \]

\[ r^i = s - \omega_i t \]

- \( p, r, v \) should be partitioned conformably
- \( s \) should be with \( r \) and \( v \)
- \( t \) should be with \( s \)
- \( x \) should be with \( p \) and \( s \)
Problem 4: Preconditioned BiCG-STAB

\[ p, r, v, s, t \] and, \( x \) should be partitioned conformably

- What remains?

\[ \hat{p} = Mp^i \]
\[ v^i = A\hat{p} \]
\[ \hat{s} = Ms \]
\[ t = A\hat{s} \]

Columns of \( M \) and rows of \( A \) should be conformal

\[ PAQ^T \]
\[ QMP^T \]
Problem 4: Preconditioned BiCG-STAB

- We use the previously proposed models
  - define operators to build composite models

Rowwise model ($y=Ax$)  
Colwise model ($w=Mz$)
Problem 4: Preconditioned BiCG-STAB

- Nevel amalgamate/unify nets of individual hypergraphs.
- Combine vertices of individual hypergraphs, and connect the composite vertex to the nets of the individual vertices.
- Define multiple weights for vertices, if the multiply operations are separated by global synchronization type of operations; individual vertex weights are not added up.
- Need to decide how to partition matrices (let's say $A$ rowwise and $M$ columnwise):
  - Generate column-net model for the matrices to be partitioned rowwise.
  - Generate row-net model for the matrices to be partitioned columnwise.
  - Apply vertex amalgamation to respect the partitioning requirement ($PAQ^T$ and $QMP^T$ or $PAMP^T$).
Problem 4: Preconditioned BiCG-STAB

- BiCG-STAB requires \( PAMP^T \)
  - Reminder: rows of \( A \) and columns of \( M \);
    columns of \( A \) and rows of \( M \)
- A rowwise (\( y = Ax \)), \( M \) columnwise (\( w = Mz \))
Problem 4: Preconditioned BiCG-STAB

- columns of $A$ and rows of $M$
Problem 4: Preconditioned BiCG-STAB

- Rows of \( A \) and columns of \( M \)
Coarsening: Vertices that share a large number of hyperedges are coalesced (various metrics).

Initial partitioning: Select a starting vertex and add the vertices to the cluster according to their gains (see refinement).

Refinement: iteratively improve the partitioning using vertex moves.

Let $F$ ("From") be the current block of cell(i) and $T$ ("To") be its complimentary block; so that $F=A$ and $T=B$ or vice-versa. The gain of cell(i) is then given by

$$g(i) = FS(i) - TE(i),$$

where $FS(i)$ is the number on nets which have cell(i) as their only $F$ cell, and $TE(i)$ is the number of nets which contain cell(i) and have an empty $T$ side. Thus a critical net on cell(i) contributes +1 or -1 to $g(i)$. The following algorithm computes the initial gains of all free cells.

```c
/* compute cell gains */
FOR each free cell i DO
  g(i) ← 0
  F ← the "from block" of cell(i)
  T ← the "to block" of cell(i)
  FOR each net n on cell i DO
    IF F(n) = 1 THEN increment g(i)
    IF T(n) = 0 THEN decrement g(i)
  END FOR
END FOR
```
Hypergraph partitioning

**Refinement:** iteratively improve the partitioning using vertex moves.

**A critical issue** is to update the gains associated with vertex moves efficiently. This has to be performed only locally for a move-based iterative improvement algorithm.

```plaintext
/* move base cell and update neighbors' gains */
F ← the "from block" of base cell
T ← the "to block" of base cell
Lock the base cell and Compliment its block
FOR each net n on the base cell DO
   /* check critical nets before the move */
   IF T(n) = 0 THEN increment gains of all free cells on net(n)
   ELSE IF T(n) = 1 THEN decrement gain of the only T cell on net(n), if it is free
   /* change the net distribution to reflect the move */
   decrement F(n)
   increment T(n)
   /* check critical nets after the move */
   IF F(n) = 0 THEN decrement gains of all free cells on net(n)
   ELSE IF F(n) = 1 THEN increment gain of the only F cell on net(n), if it is free
END FOR
```
A sparse matrix is a matrix with a lot of zero entries.

More importantly: all or some zeros are not stored.

Parallel SpMxV is an important computational kernel in many problems; furthermore it characterizes a wide range of applications with irregular computational dependency.

Row-parallel, column-parallel and row-column-parallel algorithms.

Hypergraph models can be quite handy in modeling different kind of problems.

Vertex weights are used to have load balance; nets are used to encode data dependencies. Cut size corresponds to the total communication volume.