Parallel sparse matrix vector multiplications and models for efficient parallelization

Bora Uçar

CNRS & ENS Lyon, France

28–29 November 2011
Murcia, Spain

Outline

Sparse matrices

A sparse matrix is a matrix with a lot of zero entries.

More importantly: all or some zeros are not stored.

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

Sparse matrices are abound in scientific computing: ▶ large scale optimization, ▶ chemical process simulation, ▶ computational fluid dynamics, ▶ numerical solution of partial differential equations, ▶ web information retrieval (e.g., Google's page rank),...

Sparse matrices: Coordinate format

There are many ways to store a sparse matrix.

We will look at three standard representations which store only the nonzero entries.

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

Coordinate (Triplet) format

Two integer arrays (irn, jcn) and a double array `A`:

```
irn = [  1   2   2   3   3   4   5   5   5 ]
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 ]
```

The kth entry a_{ij} is stored as `irn[k]` $= i$, `jcn[k]` $= j$, `a[k]` $= a_{ij}$.

Let τ denote the number of nonzeros, then the storage is 2τ integer and τ double (or single or complex). In general, $\tau = \mathcal{O}(m + n)$.

Sparse matrices: Compressed row storage

There are many ways to store a sparse matrix.

We will look at three 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`:

```
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 ]
```

The nonzeros of the ith 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`.

Sparse matrices: Compressed row storage

There are many ways to store a sparse matrix.

We will look at three 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`:

```
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 ]
```

The nonzeros of the ith row are stored at the `ia[i]... ia[i+1]-1` positions of `jcn and A`.

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

Sparse matrices: Compressed column storage

There are many ways to store a sparse matrix.

We will look at three 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 (`irn, ja`) and a double array `A`:

```
ja  = [  1         3         5    6              9   10 ]
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 ]
```

The nonzeros of the jth column are stored at the `ja[j]... ja[j+1]-1` positions of `irn and A`.

For example the 2nd col: starts at `ja[2] =3` and finishes at `ja[2+1]-1=4`. The row indices are therefore `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.

We will look at three 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 (`irn`, `ja`) and a double array `A`:

```
ja  = [  1         3         5    6              9   10 ]
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 ]
```

The nonzeros of the jth column are stored at the `ja[j]... ja[j+1]-1` positions of `irn and A`.

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

Reminder: Dense matrix vector multiplication

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

Row major order

for $i = 1$ **to** m **do**
  $y[i] \leftarrow 0.0$
  for $j = 1$ **to** n **do**
    $y[i] \leftarrow y[i] + A[i, j] * x[j]$

Column-major order

for $i = 1$ **to** m **do**
  $y[i] \leftarrow 0.0$
for $j = 1$ **to** n **do**
  for $i = 1$ **to** m **do**
    $y[i] \leftarrow y[i] + A[i, j] * x[j]$

Sparse matrices: Sparse matrix vector multiplies

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

Coordinate format with τ nonzeros (`irn, jcn, A`)

for $i = 1$ **to** m **do**
  $y[i] \leftarrow 0.0$
for $k = 1$ **to** τ **do**
  $y[irn[k]] \leftarrow y[irn[k]] + A[k] * x[jcn[k]]$

Sparse matrices: Sparse matrix vector multiplies

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

Compressed row storage (ia, jcn, A)

$$
\begin{array}{l}
\textbf{for } i = 1 \textbf{ to } m \textbf{ do} \\
\quad val \leftarrow 0.0 \\
\quad \textbf{for } k = ia[i] \textbf{ to } ia[i+1]-1 \textbf{ do} \\
\quad\quad val \leftarrow val + A[k] * x[jcn[k]] \\
\quad y[i] \leftarrow val
\end{array}
$$

Compressed column storage (ja, irn, A)

$$
\begin{array}{l}
\textbf{for } i = 1 \textbf{ to } m \textbf{ do} \\
\quad y[i] \leftarrow 0.0 \\
\textbf{for } j = 1 \textbf{ to } n \textbf{ do} \\
\quad xval \leftarrow x[j] \\
\quad \textbf{for } k = ja[j] \textbf{ to } ja[j+1]-1 \textbf{ do} \\
\quad\quad y[irn[k]] \leftarrow y[irn[k]] + A[k] * xval
\end{array}
$$

- 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 $y \leftarrow \mathbf{A}x$ 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 $\mathbf{A}$,
- The matrix $\mathbf{A}$ can be symmetric, unsymmetric, rectangular,
- Sometimes multiplies are of the form $y \leftarrow \mathbf{ADA}^T z$ with a diagonal $\mathbf{D}$ (in interior point methods for linear programs).
    - computation proceeds (why?) as $w \leftarrow \mathbf{A}^T z$, then $x \leftarrow \mathbf{D}w$, then $y \leftarrow \mathbf{A}x$
- Sometimes we have multiplies with $\mathbf{A}$ and $\mathbf{A}^T$ independent; $y \leftarrow \mathbf{A}x$ and $w \leftarrow \mathbf{A}^T z$ (QMR, CGNE, and CGNR methods with square unsymmetric $\mathbf{A}$; rectangular $\mathbf{A}$ in Lanczos method).
- Most of the time the SpMxV's are of the form $y \leftarrow \mathbf{AM}^{-1}x$ (called preconditioning).

Iterative solvers: How do they look?

Basic form

while not converged **do**
  computations
  check convergence

Computations are of the form:

- Linear vector operations
  $x \leftarrow x + \alpha y$ ▶ $x_i = x_i + \alpha y_i$
- Inner products
  $\alpha \leftarrow (x, y)$ ▶ $\alpha = \sum x_i y_i$
- Sparse matrix vector multiplies
  $y \leftarrow \mathbf{A}x$
  $y \leftarrow \mathbf{A}^T x$

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 $\mathbf{M}$ and/or $\mathbf{M}^T$. Replace $\mathbf{A}p_j$ with $\mathbf{AM}p_j$.

Outline

Parallel sparse matrix vector multiplies

We restrict ourselves to the distributed memory setting.

- nonzeros in **A** are distributed,
- the input vector entries, x_js, are distributed,
- the ouput vector entries, y_is, 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 processors;
  - 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.

Parallel sparse matrix vector multiplies

We restrict ourselves to the distributed memory setting.

- nonzeros in **A** are distributed,
- the input vector entries, x_js, are distributed,
- the ouput vector entries, y_is, 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 processors;
  - 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.

Parallel sparse matrix vector multiplies

We restrict ourselves to the distributed memory setting.

- nonzeros in $\mathbf{A}$ are distributed,
- the input vector entries, x_js, are distributed,
- the ouput vector entries, y_is, 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 processors;
  - 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.
$\Rightarrow$ 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 $y \leftarrow \mathbf{A}x$ after the multiply operation.

In half of the cases(!), the input vector x and the output vector y undergo vector operations (such as $x \leftarrow x + \beta y$, 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 $\mathbf{A}$ among the processors of a row of the mesh).

Outline

Parallel SpMxV: Row-parallel

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

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

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

- Block $\mathbf{A}_{k\ell}$ 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 kth row stripe $[\mathbf{A}_{k1} \cdots \mathbf{A}_{kK}]$ 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.

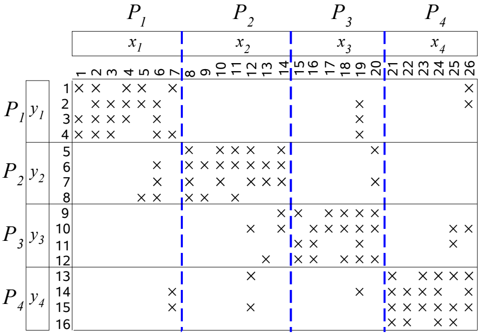
Parallel SpMxV: Row-parallel

In $y \leftarrow \mathbf{A}x$, y and x are column vectors of size m and n; $\mathbf{A}$ is partitioned as shown in the previous slide.

- A rowwise partition of matrix $\mathbf{A}$ defines a partition on the output vector y.
- The input vector x is assumed to be partitioned conformably with the column permutation of matrix $\mathbf{A}$.
- 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.



- 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

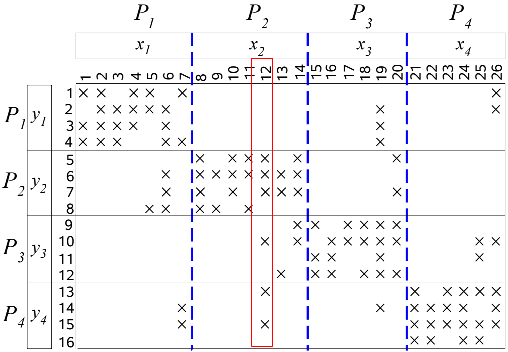
Executes the following steps at each processor P_k:

1. For each nonzero off-diagonal block $\mathbf{A}_{\ell k}$, send sparse vector $\hat{x}_k^\ell$ to processor P_ℓ, where $\hat{x}_k^\ell$ contains only those entries of x_k corresponding to the nonzero columns in $\mathbf{A}_{\ell k}$.
2. Compute the diagonal block product $y_k^k \leftarrow \mathbf{A}_{kk} \times x_k$, and set $y_k = y_k^k$.
3. For each nonzero off-diagonal block $\mathbf{A}_{k\ell}$, receive $\hat{x}_\ell^k$ from processor P_ℓ, then compute $y_k^\ell \leftarrow \mathbf{A}_{k\ell} \times \hat{x}_\ell^k$, and update $y_k \leftarrow y_k + y_k^\ell$.

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 $\mathbf{A}$. This multicast-like operation is called the expand operation.

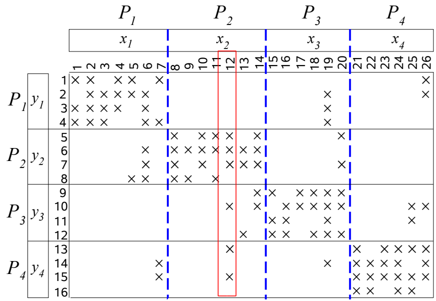
Parallel SpMxV: Row-parallel

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.



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

Row-parallel SpMxV: Communication requirements



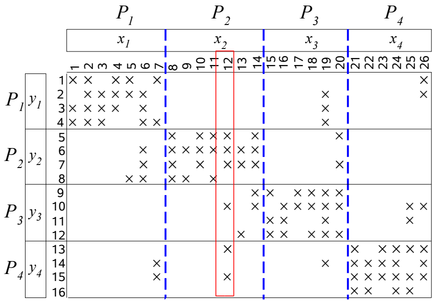
1. Expand x vector (sends/receives)
2. Compute with diagonal blocks
3. Receive x and compute with off-diagonal blocks

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 kth virtual column stripe of $\mathbf{A}$.

- P_2, sends $x[12:14]$ to P_3—nonzero columns 12, 13, and 14 in $\mathbf{A}_{32}$.
- P_2 sends $x[12]$ to P_4—nonzero column 12 in $\mathbf{A}_{42}$.
- The number of messages sent by P_2 is 2.

Row-parallel SpMxV: Communication requirements



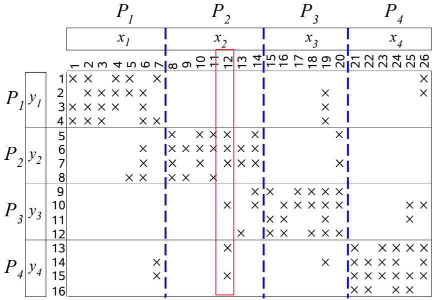
1. Expand x vector (sends/receives)
2. Compute with diagonal blocks
3. Receive x and compute with off-diagonal blocks

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 each off-diagonal block in the kth virtual column stripe of $\mathbf{A}$.

- 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



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

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.

Outline

Parallel SpMxV: Column-parallel

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

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

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

- Block $\mathbf{A}_{k\ell}$ 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 kth column stripe $[\mathbf{A}_{1k}^T \cdots \mathbf{A}_{Kk}]^T$ 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.

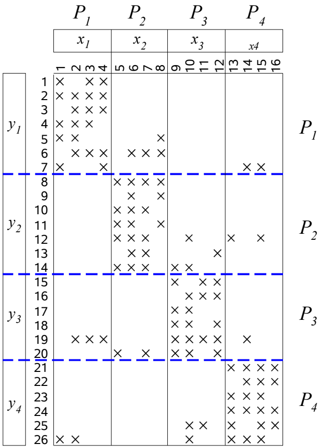
Parallel SpMxV: Column-parallel

In $y \leftarrow \mathbf{A}x$, y and x are column vectors of size m and n; $\mathbf{A}$ is partitioned as shown in the previous slide.

- A columnwise partition of matrix $\mathbf{A}$ defines a partition on the input-vector x.
- The output vector y is assumed to be partitioned conformably with the row permutation of matrix $\mathbf{A}$.
- 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: Column-parallel

Matrix is partitioned columns among 4 processors.



- 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 P_1;
  26 nonzeros assigned to processor P_2;
  25 nonzeros assigned to processor P_3;
  25 nonzeros assigned to processor P_4.

Parallel SpMxV: Column-parallel algorithm

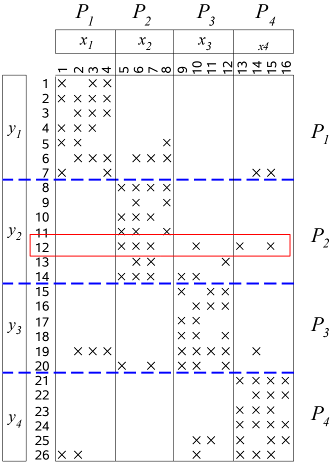
Executes the following steps at each processor P_k:

1. For each nonzero off-diagonal block $\mathbf{A}_{\ell k}$, form sparse vector $\hat{y}_\ell^k$ which contains only those results of $y_\ell^k = \mathbf{A}_{\ell k} \times x_k$ corresponding to the nonzero rows in $\mathbf{A}_{\ell k}$. Send $\hat{y}_\ell^k$ to processor P_ℓ.
2. Compute the diagonal block product $y_k^k \leftarrow \mathbf{A}_{kk} \times x_k$, and set $y_k = y_k^k$.
3. For each nonzero off-diagonal block $\mathbf{A}_{k\ell}$ receive partial-result vector $\hat{y}_k^\ell$ from processor P_ℓ, 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.

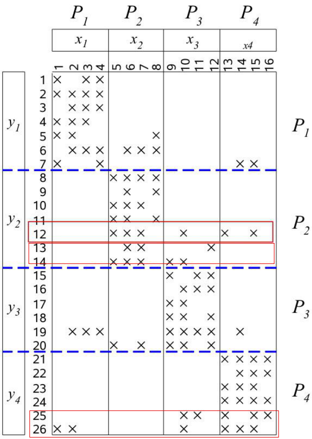
Parallel SpMxV: Column-parallel

Matrix is partitioned columnwise among 4 processors. x vector entries are partitioned according to the columnwise partition of $\mathbf{A}$; assume the y vector entries are partitioned and the rows of $\mathbf{A}$ 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

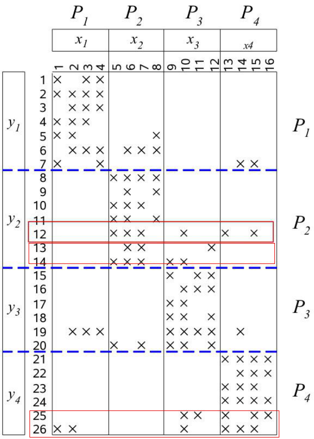


Fact 1: Number of messages sent by P_k

The number of messages sent by processor P_k in column-parallel $y \leftarrow \mathbf{A}x$ is equal to the number of nonzero off-diagonal blocks in the kth column stripe of $\mathbf{A}$.

- 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

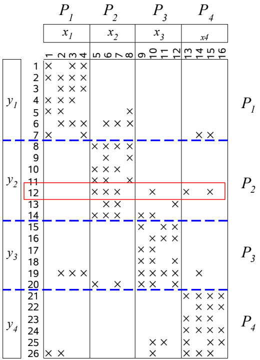


Fact 2: Volume of messages sent by P_k

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

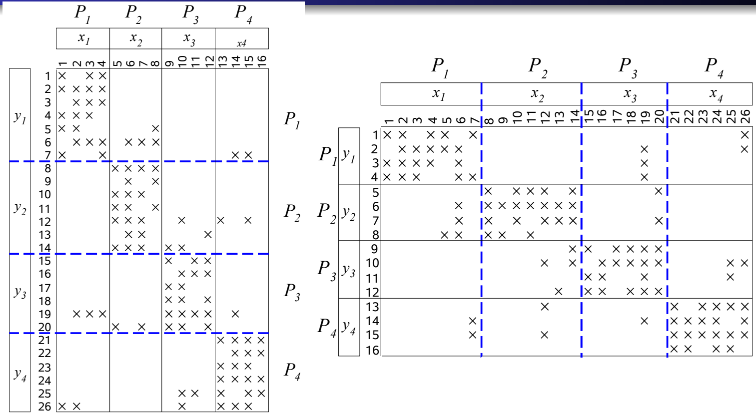


1. Compute with off diagonal blocks; obtain partial y results, issue sends/receives
2. Compute with diagonal block
3. Receive partial results on y for nonzero off-diagonal blocks and add the partial results

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 $y \leftarrow \mathbf{A}^T x$ and row parallel $y \leftarrow \mathbf{A}x$ are duals of each other (the columnwise partition on $\mathbf{A}^T$ is equal to the rowwise partition on $\mathbf{A}$).

Outline

Parallel SpMxV: Row-column parallel algorithm

Consider $y \leftarrow \mathbf{A}x$, 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 $\mathbf{A}^k$

$$\mathbf{A}^k = \begin{bmatrix} \mathbf{A}_{11}^k & \cdots & \mathbf{A}_{1k}^k & \cdots & \mathbf{A}_{1K}^k \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{A}_{k1}^k & \cdots & \mathbf{A}_{kk}^k & \cdots & \mathbf{A}_{kK}^k \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{A}_{K1}^k & \cdots & \mathbf{A}_{Kk}^k & \cdots & \mathbf{A}_{KK}^k \end{bmatrix}$$

of size $m \times n$, where $\mathbf{A} = \sum \mathbf{A}^k$ (here $\mathbf{A}^k$s are disjoint).

Parallel SpMxV: Row-column parallel algorithm

P_k has $\mathbf{A}^k$, holds x_k and is responsible for y_k.

$$\mathbf{A}^k = \begin{bmatrix} A^k_{11} & \cdots & A^k_{1k} & \cdots & A^k_{1K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ A^k_{k1} & \cdots & A^k_{kk} & \cdots & A^k_{kK} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ A^k_{K1} & \cdots & A^k_{Kk} & \cdots & A^k_{KK} \end{bmatrix}$$

- The blocks in row-block stripe $A^k_{k*} = [A^k_{k1}, \cdots, A^k_{kk}, \cdots, A^k_{kK}]$ have row dimension of size m_k.
- The blocks in column-block stripe $A^k_{*k} = [A^k_{1k}, \cdots, A^k_{kk}, \cdots, A^k_{Kk}]$ have column dimension of size n_k.

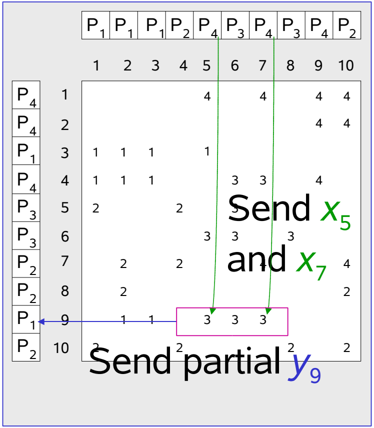
- The x-vector entries that are to be used by processor P_k are represented as $x^k = [x^k_1, \cdots, x^k_k, \cdots, x^k_K]$, where x^k_k corresponds to x_k and other x^k_ℓ are belonging to some other processor P_ℓ.
- The y-vector entries for which processor P_k computes partial results are represented as $y^k = [y^k_1, \cdots, y^k_k, \cdots, y^k_K]$, where y^k_k corresponds to y_k and other y^k_ℓ are to be sent to some other processor P_ℓ.

Parallel SpMxV: Row-column parallel algorithm

Executes the following steps at each processor P_k:

1. For each $\ell \neq k$ having nonzero column-block stripe $\mathbf{A}^{\ell}_{*k}$, send sparse vector $\hat{x}^{\ell}_{k}$ to processor P_ℓ, where $\hat{x}^{\ell}_{k}$ contains only those entries of x_k corresponding to the nonzero columns in $\mathbf{A}^{\ell}_{*k}$.
2. Compute the column-block stripe product $y^k \leftarrow \mathbf{A}^{k}_{*k} \times x^{k}_{k}$.
3. For each nonzero column-block stripe $\mathbf{A}^{k}_{*\ell}$, receive $\hat{x}^{k}_{\ell}$ from processor P_ℓ, then compute $y^k \leftarrow y^k + \mathbf{A}^{k}_{*\ell} \times \hat{x}^{k}_{\ell}$, and set $y_k = y^k_k$.
4. For each nonzero row-block stripe $\mathbf{A}^{k}_{\ell *}$, form sparse partial-result vector $\hat{y}^{k}_{\ell}$ which contains only those results of $y^k_\ell = \mathbf{A}^{k}_{\ell *} \times x^k$ corresponding to the nonzero rows in $\mathbf{A}^{k}_{\ell *}$. Send $\hat{y}^{k}_{\ell}$ to processor P_ℓ.
5. For each $\ell \neq k$ having nonzero row-block stripe $\mathbf{A}^{\ell}_{k *}$ receive partial-result vector $\hat{y}^{\ell}_{k}$ from processor P_ℓ, and update $y_k \leftarrow y_k + \hat{y}^{\ell}_{k}$.

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.*

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

Outline

Parallelization objectives

Achieve load balance

Load of a processor: number of nonzeros.
$\Rightarrow$ 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?).

The common metric in different works: total volume of communication.

Hypergraphs: Definitions

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.

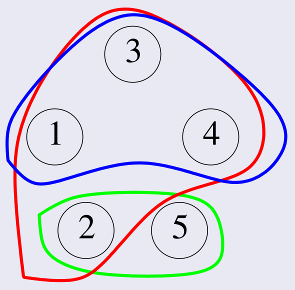
A weight $w(v)$ is associated with each vertex v.

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

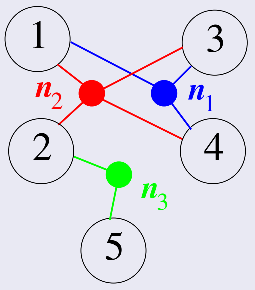
Hypergraphs: Example

$\mathcal{H} = (\mathcal{V}, \mathcal{N})$ with $\mathcal{V} = \{1, 2, 3, 4, 5\}$ $\mathcal{N} = \{n_1, n_2, n_3\}$ where
$n_1 = \{1, 3, 4\}$ $\quad n_2 = \{1, 2, 3, 4\}$ $\quad n_3 = \{2, 5\}$

Venn diagram-like representation



Graph-like representation



Hypergraphs: Partitioning

Partition

$\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is a K-way vertex partition if

- $\mathcal{V}_k \neq \emptyset$,
- parts are mutually exclusive: $\mathcal{V}_k \cap \mathcal{V}_\ell = \emptyset$,
- parts are collectively exhaustive: $\mathcal{V} = \bigcup \mathcal{V}_k$.

In Π, a net connects a part if it has at least one vertex in that part, i.e., h connects $\mathcal{V}_k$ if $h \cap \mathcal{V}_k \neq \emptyset$.

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

Objective: minimize cutsize(Π)

$$\sum_h c(h)(\lambda(h) - 1),$$

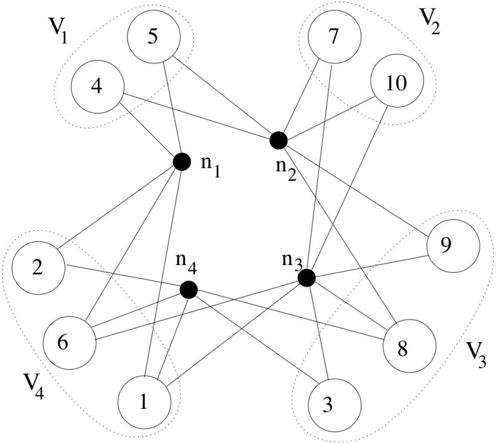
Constraint: balanced part weights

$$\sum_{v \in \mathcal{V}_k} w(v) \leq (1 + \varepsilon) \frac{\sum_{v \in \mathcal{V}} w(v)}{K}.$$

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.
$V_1 = \{4, 5\}$ $V_2 = \{7, 10\}$ $V_3 = \{3, 8, 9\}$ $V_4 = \{1, 2, 6\}$



net	connectivity–1
1	2–1=1
2	3–1=2
3	3–1=2
4	2–1=1

Cutsize: 6

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.

Hypergraph models for matrix partitioning for parallel computations

In all of the cases we will see, we will have unit net-costs, that is $c(h) = 1$ The objective function becomes

$$\sum_{h}(\lambda(h) - 1)$$

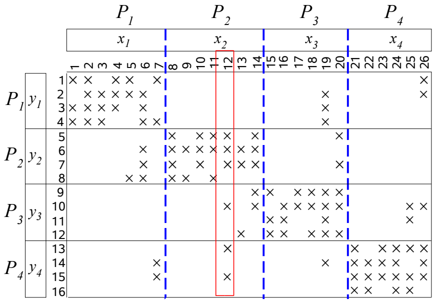
- Make the data to be partitioned as vertices of the hypergraph.
- Assign weights to the vertices.
- Put nets to represent dependencies of computations to the input data; and dependencies of output data into computations.
- Partition into K parts, each V_k holds the data of a processor.
- Load balance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

Outline

Hypergraph models for row-parallel SpMxV

- Three entities to partition y, rows of $\mathbf{A}$, 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_is where $a_{ij} \neq 0$ need x_j
  connect x_j and all such r_i

Row-parallel SpMxV: Communication requirements

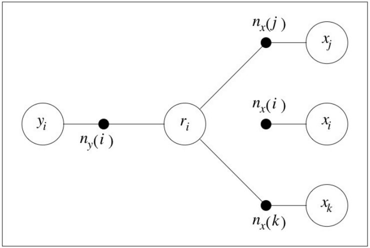


1. Expand x vector (sends/receives)
2. Compute with diagonal blocks
3. Receive x and compute with off-diagonal blocks

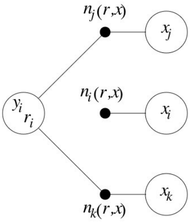
Total volume and number of messages

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

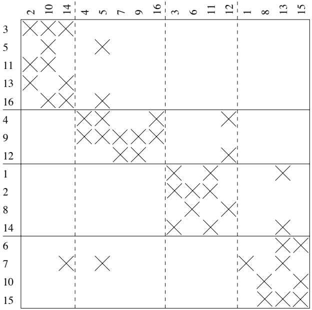


Combine y_i and r_i: owner computes rule

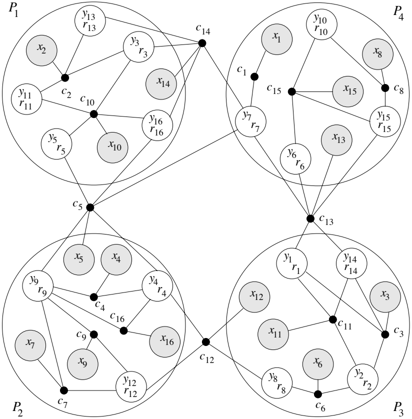
Partition the vertices into K parts
(partition the data among K processors)

Part weights=processor's loads in terms of nnz.

Hypergraph models for row-parallel SpMxV



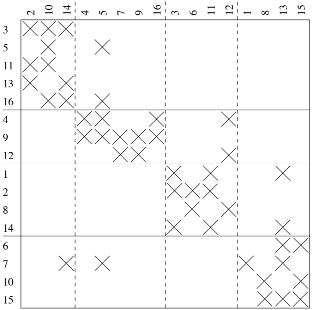
Number of nonzeros columns in off-diagonal blocks is 5. Total volume is 5.



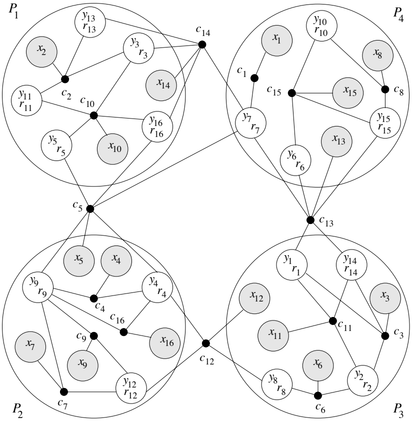
Column-net c_{14} connects 2 parts; c_5 connects 3 parts; c_{12} connects 2 parts; c_{13} connects 2 parts. Cut-size is 5.

Hypergraph models for row-parallel SpMxV

What about load balance?



There are 12 nnz in the first row stripe.



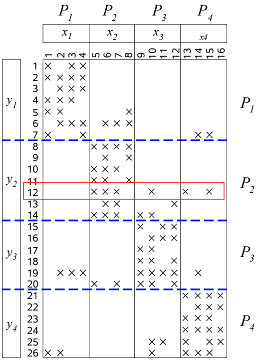
Each row-vertex gets a weight equivalent to the number of nonzeros in the associated row of **A**.

Outline

Hypergraph models for column-parallel SpMxV

- Three entities to partition y, columns of $\mathbf{A}$, 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



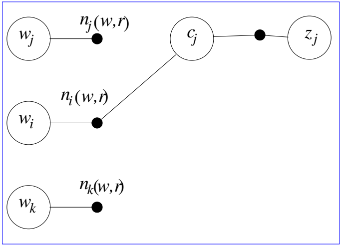
1. Compute with off diagonal blocks; obtain partial y results, issue sends/receives
2. Compute with diagonal block
3. Receive partial results on y for nonzero off-diagonal blocks and add the partial results

Total volume and number of messages

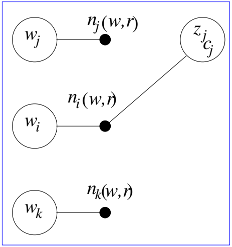
The total volume of messages is equal to the number of nonzero rows in off-diagonal blocks. (13)

Hypergraph models for column-parallel SpMxV

For column-parallel $w \leftarrow \mathbf{A}z$ computations.



Elementary hypergraph model for 1D colwise partitioning.

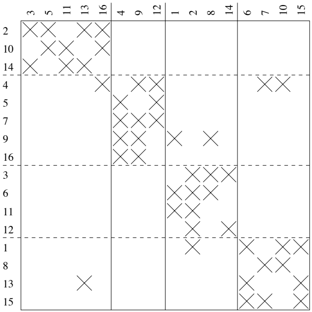


Combine c_j and z_j; one column needs only one z-vector entry.

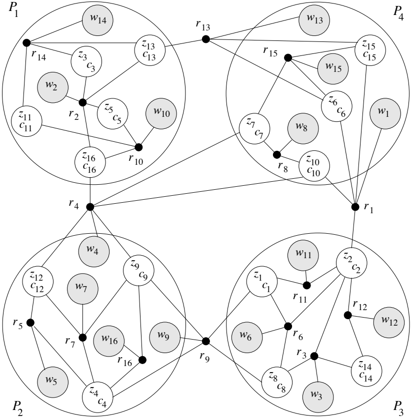
Partition the vertices into K parts (partition the data among K processors). Part weights=processor loads in terms of number of nonzeros.

Hypergraph models for column-parallel SpMxV

The computation is $w \leftarrow \mathbf{A}z$



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

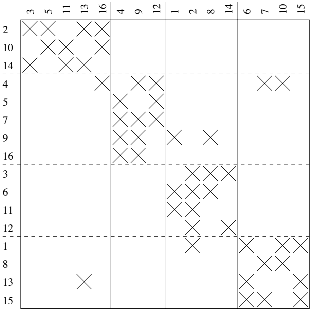


Row-net r_{13} connects 2 parts; r_1 connects 2 parts; r_9 connects 2 parts; r_4 connects 3 parts.
Cut-size is 5.

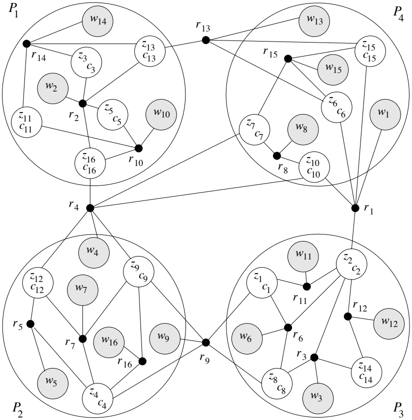
Hypergraph models for column-parallel SpMxV

What about load balance?

The computation is $w \leftarrow \mathbf{A}z$



There are 11 nonzeros in the second column stripe.



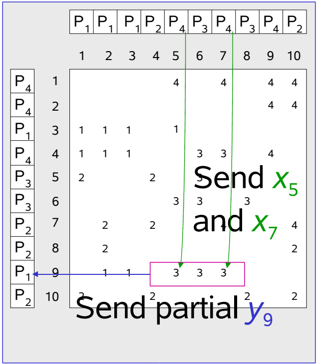
Each column-vertex gets a weight equal to the number of nonzeros in the associated column of $\mathbf{A}$.

Outline

Hypergraph models for row column-parallel SpMxV

- Three entities to partition y, nonzeros of $\mathbf{A}$, 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 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

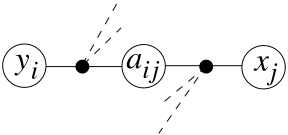


1. Expand x vector
2. Scalar multiply and add $(y_i \leftarrow a_{ij}x_j + a_{ik}x_k)$
3. Fold on y vector (send and receive partial results)

- 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.

Hypergraph models for row column-parallel SpMxV

For row-column-parallel $y \leftarrow \mathbf{A}x$ computations.

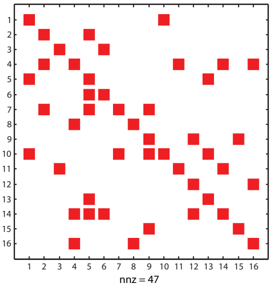


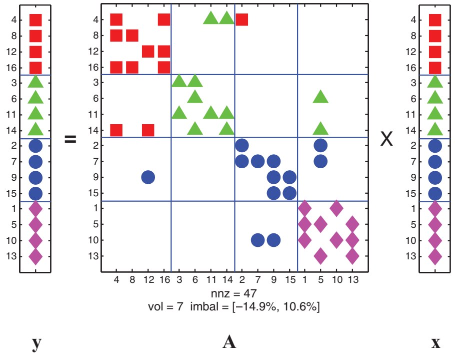
Elementary hypergraph model for row-column-parallel algorithm

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 $y \leftarrow \mathbf{A}x$ computations.





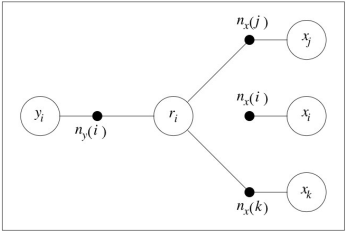
[Part of the fine grain model on the board....]

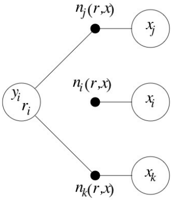
Outline

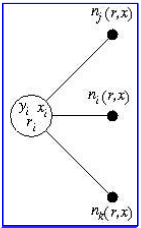
The approach

- 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







Combine x_i: symmetric partitioning

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 $\mathbf{A}$ **rowwise** for the $y \leftarrow \mathbf{A}x$ computations under **given, possibly different, partitions** on the input and output vectors x and y.

A parallel algorithm that carries out the $y \leftarrow \mathbf{A}x$ computations under given partitions of x and y should have a communication phase on x, a computation phase, and a communication phase on y.

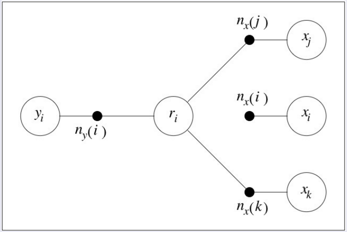
Solution to Problem 1

Problem

Describe a hypergraph model which can be used to partition the matrix **A** **rowwise** for the $y \leftarrow \mathbf{A}x$ 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 **different** than the **partition on the rows** of $\mathbf{A}$ for the $y \leftarrow \mathbf{A}x$ computations.

The previous parallel algorithm will be used.

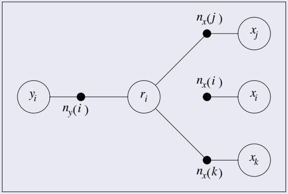
Solution to Problem 2

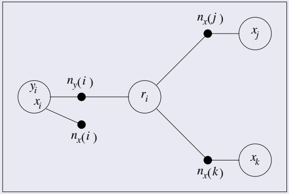
Problem

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 $\mathbf{A}$ for the $y \leftarrow \mathbf{A}x$ computations.

Solution

Take the elementary model and amalgamate the vertices x_i and y_i



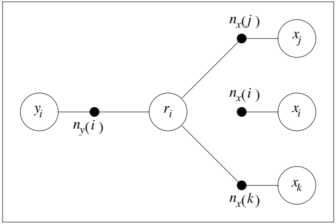


Problem 3

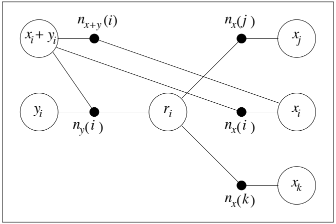
Problem

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

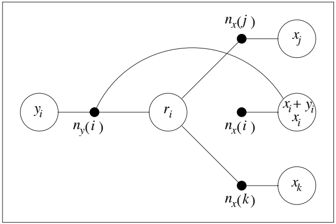
Solution to Problem 3



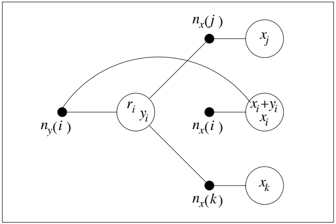
(a) Elementary model for $y \leftarrow \mathbf{A}x$



(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 $\mathbf{A}x = b$ 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 $\mathbf{M}$ such that $\mathbf{AM} \approx I$
- Instead of solving $\mathbf{A}x = b$, use right preconditioning and solve

$$\mathbf{AM}y = b$$

and then set

$$x = \mathbf{M}y$$

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

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

p, r, v should be partitioned conformably

$$\hat{p} = M p^i$$

$$v^i = A\hat{p}$$

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

s should be with r and v

$$\hat{s} = Ms$$

$$t = A\hat{s}$$

$$\omega_i = \langle t, s\rangle / \langle t, t\rangle$$

t should be with s

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

x should be with p and s

$$r^i = s - \omega_i t$$

Problem 4: Preconditioned BiCG-STAB

$\Rightarrow p, r, v, s, t$, and, x should be partitioned conformably

- What remains?

$\hat{p}$

$\hat{s}$

should be conformal

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

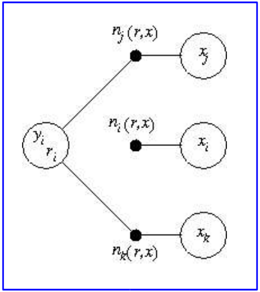
Columns of M and rows of A should be conformal

Rows of M and columns of A should be conformal

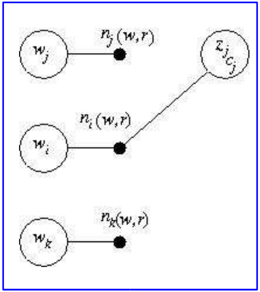
PAQ^T
QMP^T

Problem 4: Preconditioned BiCG-STAB

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



Rowwise model (y=Ax)



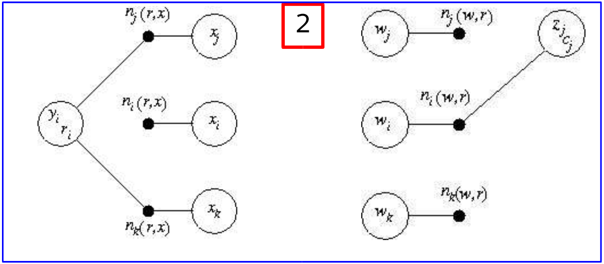
Colwise model (w=Mz)

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 (lets say $\mathbf{A}$ rowwise and $\mathbf{M}$ 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 ($\mathbf{PAQ}^T$ and $\mathbf{QMP}^T$ or $\mathbf{PAMP}^T$).

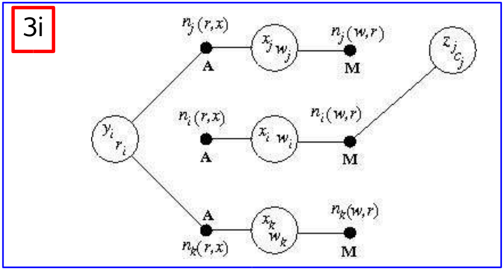
Problem 4: Preconditioned BiCG-STAB

- BiCG-STAB requires $PAMP^T$ → 1
  - Reminder: rows of A and columns of M; columns of A and rows of M
- A rowwise (y=Ax), M columnwise (w=Mz)



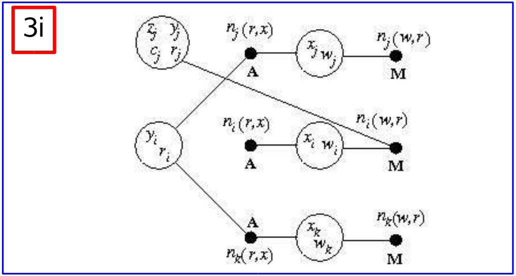
Problem 4: Preconditioned BiCG-STAB

- columns of A and rows of M



Problem 4: Preconditioned BiCG-STAB

- Rows of A and columns of M



Problem 4: Preconditioned BiCG-STAB

Parallel speed-up values

	8-way					16-way				
	Volume		Message		Sp.	Volume		Message		Sp.
Matrix	tot	max	tot	max	up	tot	max	tot	max	up
	CR									
Zhao1-A	4098	746	32.2	5.7	6.2	6444	586	96.7	9.3	8.7
Zhao1-M	3514	694	32.2	5.6		5478	551	97.0	9.3	
big-A	1032	201	31.4	5.7	5.7	1581	156	73.2	7.5	7.3
big-M	989	191	31.9	5.6		1527	150	75.3	7.5	
cage11-A	24424	4144	54.6	7.0	5.5	34835	3314	201.2	14.8	8.1
cage11-M	14663	2439	55.1	7.0		21010	1917	208.9	15.0	
cage12-A	87542	14306	56.0	7.0	5.9	122878	11925	230.3	15.0	9.4
cage12-M	50962	7839	56.0	7.0		71066	6136	233.1	15.0	
epb2-A	2326	429	39.0	6.4	6.4	3357	371	102.8	9.6	8.6
epb2-M	2242	438	35.0	6.5		3335	335	84.7	8.4	
epb3-A	2354	442	23.9	4.3	7.3	3971	393	66.0	6.5	12.4
epb3-M	3003	536	23.9	4.3		5023	496	66.3	6.5	
mark3_060-A	5249	960	35.2	6.3	5.8	9370	786	115.0	11.3	8.7
mark3_060-M	6323	1182	32.2	6.0		10287	964	105.3	11.0	
olafu-A	3908	960	25.8	5.0	6.7	6489	781	66.2	6.8	10.6
olafu-M	6749	1449	28.0	5.4		11258	1285	77.8	7.8	
stomach-A	14614	2815	21.1	4.0	7.1	24436	2351	67.2	7.0	14.1
stomach-M	16193	3206	21.4	4.0		28014	2652	67.8	7.1	
xenon1-A	10848	2037	36.2	6.5	6.7	15998	1496	113.2	11.3	11.2
xenon1-M	14437	2523	37.7	6.7		21459	2032	117.8	11.8	

Outline

Summary

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 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.

Thanks!

Thanks for your attention.

`http://perso.ens-lyon.fr/bora.ucar/`

Some of the material are from papers by Aykanat, Çatalyürek, Bisseling.