CS601: Software Development for Scientific Computing

Autumn 2024

Week8: Motifs- Sparse Matrix Computation

Matrix Data and Efficiency

- Sparse Matrices
 - E.g. banded matrices
 - Diagonal
 - Tridiagonal etc.
- Symmetric Matrices

Admit optimizations w.r.t.

- Storage
- Computation

Sparse Matrices - Motivation

 Matrix Multiplication with Upper Triangular Matrices (C=C+AB)

$$\begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} + a_{12}b_{22} & a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{13} \\ 0 & a_{22}b_{22} & a_{22}b_{23} + a_{23}b_{33} \\ 0 & 0 & a_{33}b_{33} \end{bmatrix}$$

A*B

The result, A*B, is also upper triangular.

The non-zero elements appear to be like the result of *inner-product*

Sparse Matrices - Motivation

 C=C+AB when A, B, C are upper triangular, pseudocode: for i=

```
for j=
    for k=
        C[i][j] = C[i][j] + A[i][k]*B[k][j]
```

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{bmatrix}^* \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ 0 & b_{22} & b_{23} \\ 0 & 0 & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} + a_{12}b_{22} & a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{13} \\ 0 & a_{22}b_{22} & a_{23}b_{23} + a_{23}b_{33} \\ 0 & 0 & a_{33}b_{33} \end{bmatrix}$$

$$A^*B$$

Nikhil Hegde

Sparse Matrices - Motivation

 C=C+AB when A, B, C are upper triangular, pseudocode: for i=1 to N

- Cost = $\sum_{i=1}^{N} \sum_{j=i}^{N} 2(j-i+1)$ flops (why 2?)
- Using $\Sigma_{i=1}^{N} i \approx \frac{n^2}{2}$ and $\Sigma_{i=1}^{N} i^2 \approx \frac{n^3}{3}$
- $\Sigma_{i=1}^N \Sigma_{j=i}^N 2(j-i+1) \approx \frac{n^3}{3}$, 1/3rd the number of flops required for dense matrix-matrix multiplication

Sparse Matrices

Have lots of zeros (a large fraction)

```
      X
      X
      0
      0
      X
      0
      0
      X

      0
      X
      0
      0
      X
      0
      0
      0
      0

      0
      X
      X
      X
      0
      X
      0
      0
      X

      X
      0
      0
      X
      X
      0
      0
      0
      X

      0
      X
      X
      0
      0
      0
      X
      X
      X
```

- Representation
 - Many formats available
 - Compressed Sparse Row (CSR)

```
Implementation:Three arrays:
double *val;
int *ind;
int *rowstart;
```

Sparse Matrices - Example

Using Arrays

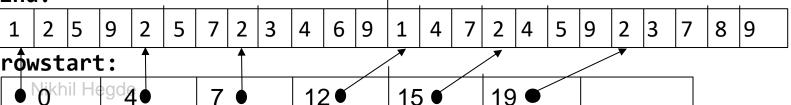
 A_{11} A_{12} A_{12} <t

double *val; //size= NNZ
int *ind; //size=NNZ
int *rowstart; //size=M=Number of rows

val:

	. _ _ _ _	_ _ _	- - - -	_ _
$ a_{11} a_{12} a_{13} a_{14} a_{14} a_{22} a_{23} a_{24} a_{24$	ı ₃₆ a ₃₉ a ₄₁ a ₄₄ a ₄₇	Ja-Ja-Ja-		la colla co
$ d_{11} d_{12} d_{15} d_{19} d_{22} d_{25} d_{27} d_{32} d_{33} d_{34} d_{34}$	'361~391~411~441~4 <i>7</i>	a ₅₂ a ₅₄ a ₅₅	0145914621463146/	d ₆₈ d ₆₉

ind:



Gaxpy with Sparse Matrices: y=y+Ax

Using arrays

```
for i=0 to numRows
  for j=rowstart[i] to rowstart[i+1]-1
  y[i] = y[i] + val[j]*x[ind[j]]
```

- Does the above code reuse y, x, and val ? (we want our code to reuse as much data elements as possible while they are in fast memory):
 - y? Yes. Read and written in close succession.
 - x? Possible. Depends on how data is scattered in val.
 - val? Good spatial locality here. Less likely for a sparse matrix in general.

Nikhil Hegde

Gaxpy with Sparse Matrices: y=y+Ax

Optimization strategies:

```
for i=0 to numRows
  for j=rowstart[i] to rowstart[i+1]-1
  y[i] = y[i] + val[j]*x[ind[j]]
```

- Unroll the j loop // we need to know the number of non-zeros per row
- Eliminate ind[i] and thereby the indirect access to elements of x.
 Indirect access is not good because we cannot predict the pattern of data access in x. //We need to know the column numbers
- Reuse elements of x //The elements of a should be e.g. located closely

These optimizations will not work for y=y+Ax pseudocode in general. When you know the data pattern and metadata info as mentioned above, you can reorder computations (scheduling optimization), reorganize data for better locality.

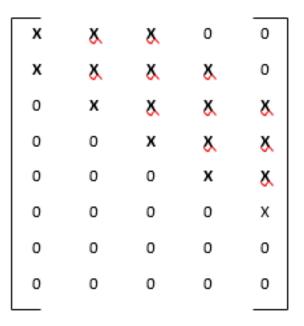
Banded Matrices

- Special case of sparse matrices, characterized by two numbers:
 - Lower bandwidth p, and upper bandwidth q

Exercise: When is $a_{ij} = 0$? (Write the constraints in terms of i, j, p, q)

$$- a_{ij} = 0 \text{ if } i > j+p$$

$$- a_{ij} = 0 \text{ if } j > i+q$$



Banded Matrices - Representation

Optimizing storage (specific to banded matrices)

	-		•	` -				•			
a ₁₁	a ₁₂	a ₁₃	0	0							
a ₂₁	a ₂₂	a ₂₃	a ₂₄	0	*	*	a ₁₃	a ₂₄	a ₃₅		
0	a ₃₂	a ₃₃	a ₃₄	a ₃₅	*	a ₁₂	a ₂₃	a ₃₄	a ₄₅		
0	0	a ₄₃	a ₄₄	a ₄₅	$\begin{vmatrix} a_{11} \end{vmatrix}$	a ₂₂	a ₃₃	a ₄₄	a ₅₅		
0	0	0	a ₅₄	a ₅₅	a ₂₁	a ₃₂	a ₄₃	a ₅₄	a ₆₅		
0	0	0	0	a ₆₅		Aband					
0	0	0	0	0		Abanu					

E.g. $A_{44} = Aband_{34}$

Exercise: A_{ij}=Aband(i-j+q+1, j)

Gaxpy with Banded Matrices: y = y + Aband x

A=Aband: optimizing computation and storage

```
for j=1 to n
   alpha1=max(1, j-q)
   alpha2=min(n, j+p)
   beta1=max(1, q+2-j)
   for i=alpha1 to alpha2
     y[i]=y[i] + Aband(beta1+i-alpha1,j)*x[j]
```

 Cost? 2n(p+q+1) time! Much lesser than 2N² time required for regular y=y+Ax (assuming p and q are much smaller than n)

Nikhil Hegde