

Parallelizing Sparse-BLAS using OpenMP and MPI

Santhosh Sharma A

Saurabh B Joshi

Computer Science and Engineering

IIT Kanpur

Outline

- Problem Statement
- OpenMP
 - Issues and Methodology
 - Experiments and Observations
- MPI
 - Issues and Methodology
 - Experiments and Observations
- References

Problem Statement

- Parallelize BLAS Level-2 routines in a way to get as much speed up as possible
- Level 1 : Scalar-Vector Operations
 - $\mathbf{b} := \text{Trans}(\mathbf{x}) * \mathbf{y}$ (Dot-product)
 - $\mathbf{y} := (\mathbf{a} * \mathbf{x}) + \mathbf{y}$ (Vector-update)
- Level 2 : Matrix-Vector Operations
 - $\mathbf{y} := (\mathbf{a} * \mathbf{A} * \mathbf{x}) + \mathbf{y}$ (Matrix-vector multiply)
 - $\mathbf{x} := (\mathbf{a} * \text{Inv}(\mathbf{T})) * \mathbf{x}$ (Triangular Solve)

Sparse Matrix – Dense Vector Multiplication

- A sparse matrix could be of the following three types

Unstructured Sparse Matrix

- The non-zero elements are stored in the form
 <matrix <row<pair<value, column> > >

Sparse Symmetric Matrix

- Only the non-zero elements of the lower triangle are stored
- Elements of the upper triangle can be obtained by taking transpose of the lower triangle!

Sparse Triangular Matrix

- Elements of the lower triangle and the elements of diagonal are stored separately

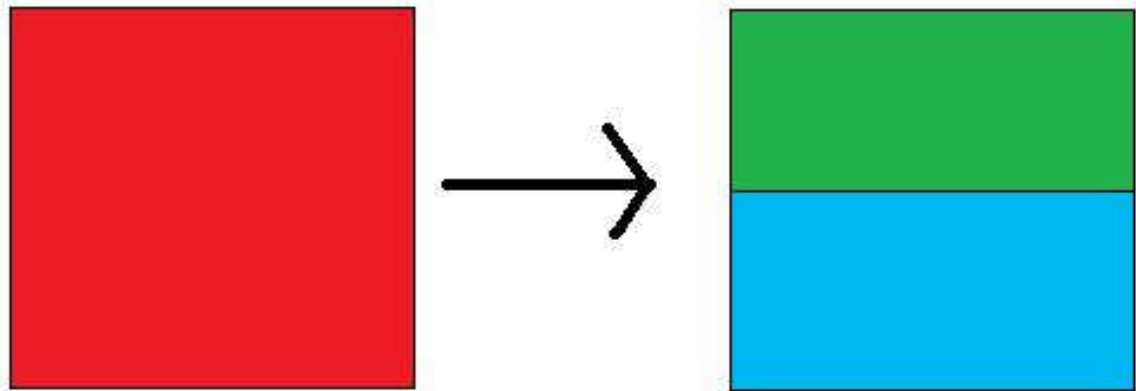
OpenMP Issues

- `for(iterator it=v.start; it!=v.end;it++)`
- `{ do_something(it); }`

Workaround :

- `#pragma omp parallel for ...`
- `for(int i=0;i<v.size;i++)`
- `{ do_something(v[i]); }`

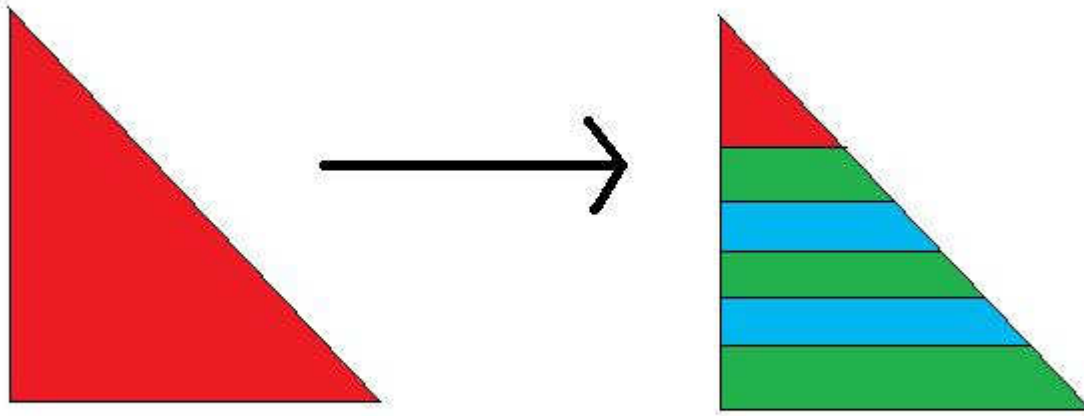
Matrix Vector Parallelization



Triangular Solve Parallelization

- $x_i = (b_i - \text{SUM} (a_{ij} * x_j)) / a_{ii}$
- Each iteration depends on earlier iterations
- Take benefit from sparsity
- Two orthogonal vectors are completely independent

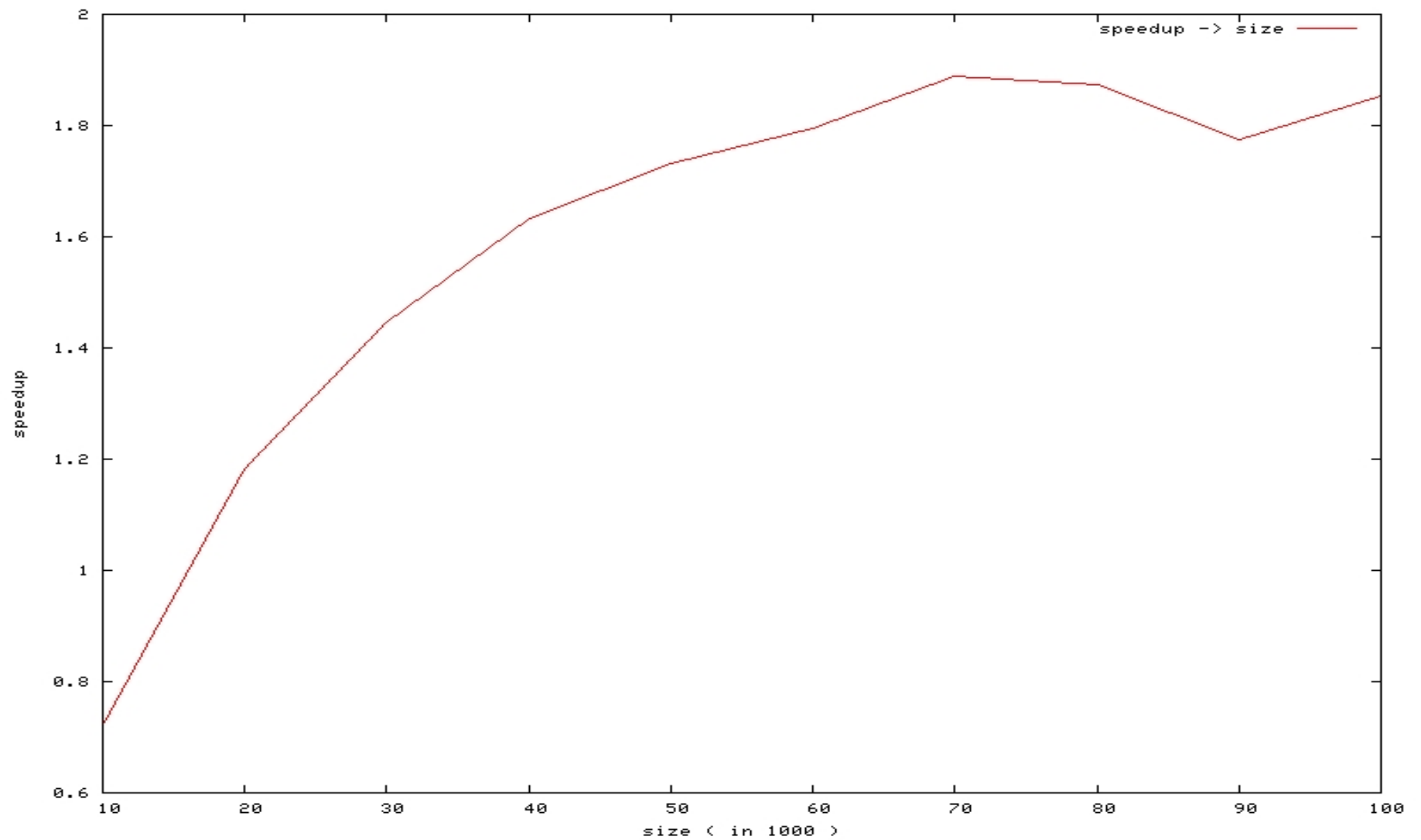
Triangular Solve Parallelization



Experiments and Observation

- For small inputs sequential version is way faster than parallel version
- Speed up increases as the input size increases
- Better speedup when optimizations are suppressed

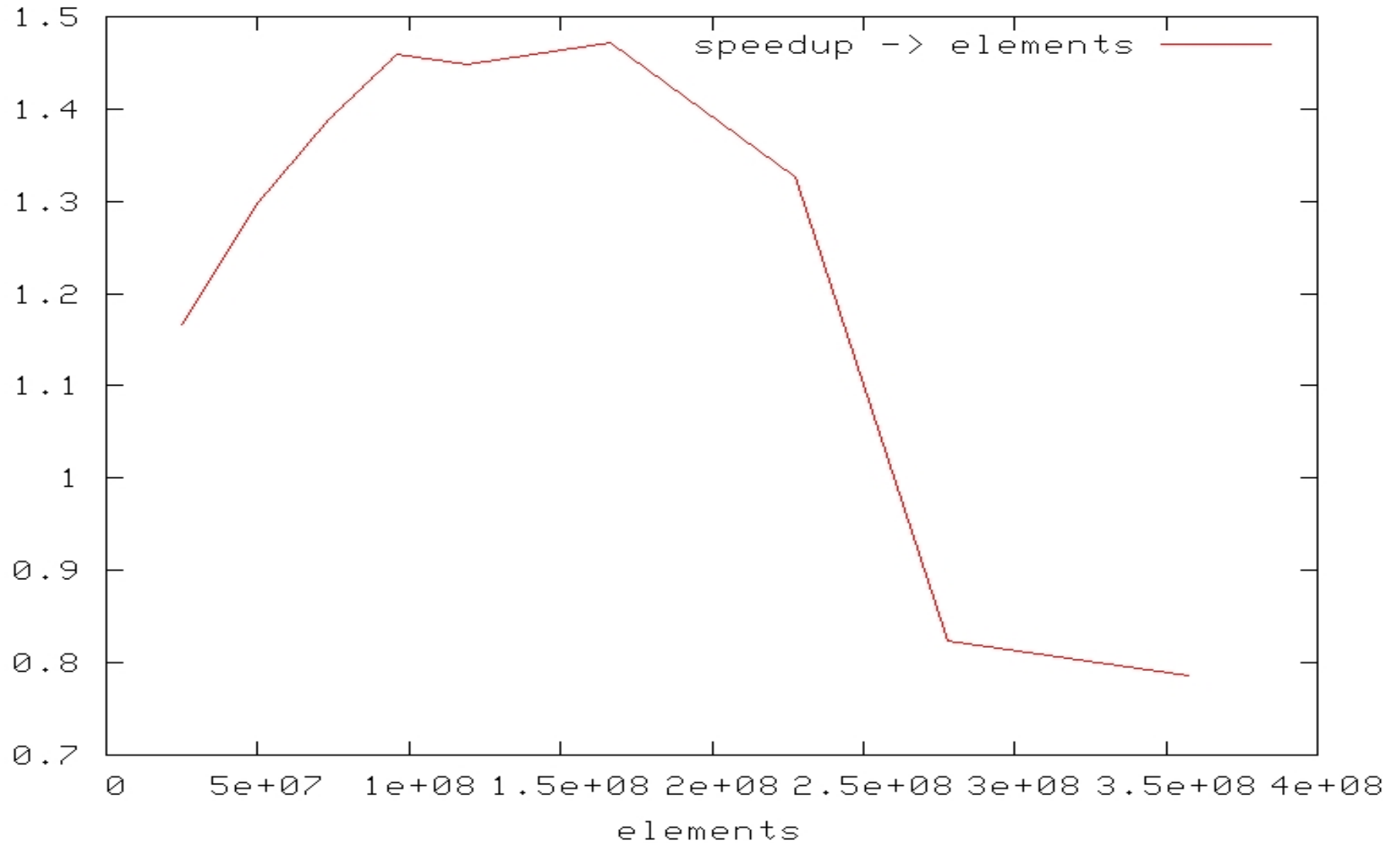
Experiments and Observation



Experiments and Observation

- Data dependency pulls down the speedup
- As the density increases speed up goes down

Experiments and Observation



MPI- MV Basic algorithm

Input 1:- Sparse Matrix M

Input 2:- Vector V

Output :- Vector R

Assumption :- All elements of R are initialized to 0.0

Operation :- $R = M \times V$ (Matrix-vector multiplication)

for every row I in the matrix M

for every pair(column,value) in the row I

$R[I] += \text{value} * V[\text{column}]$ //dot product

Simple Example (MV Mult.)

Matrix :-

a11	0	a13
0	a22	0
0	a32	0

Vector :-

b1
b2
b3

Result

$a_{11} * b_1 + a_{13} * b_3$
$a_{22} * b_2$
$a_{32} * b_2$

One important observation :-

- Total number of non-zero elements = 4
- Total number of additions = 4
- Total number of multiplications = 4
- All are equal !!!!

Execution time of the algorithm

Execution time = (Total number of non-zero elements)
* (Time taken for one addition + Time taken for one
multiplication + Time taken for one load + time taken for
one store)

i.e , Execution time = $NZ * K$ (say)

But K is a constant

Hence , Execution time is proportional to NZ

Use of the above result for parallelization

- Given two processors , we can distribute the load by partitioning the matrix such that first x rows contain nearly $NZ / 2$ elements and
- The next $(M - x)$ rows contain $NZ/2$ elements

Algorithm (Parallelized!)

Processor 1 :-

```
for every row I in the matrix M //from 1 to x
  for every pair(column,value) in the row I
    R[I] += value * V[column] //dot product
Recv(an array of M-x elements)
```

Processor 2 :-

```
for every row I in the matrix M //from x+1 to M-x
  for every pair(column,value) in the row I
    R[I] += value * V[column] //dot product
Send(an array of M-x elements)
```

Triangular Solve Algorithm

Input 1:- Sparse Triangular Matrix M

Input 2:- Vector v

Output :- Vector v

Operation :- Triangular Solve

for I = 1 TO NUM_ROWS

$v[I] = (v[I] - \text{dotproduct}(M[i],v)) / \text{diag}[I]$

Parallelization of Triangular solve Algorithm

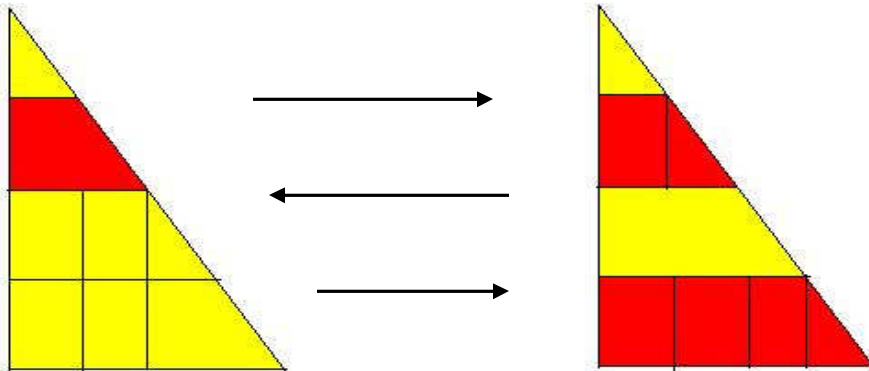
- In the expression ,

$$V[l] = (V[l] - (a[l,1]*v_1 + a[l,2]*v_2 + \dots + a[l,l-1]*v_{l-1})) / a[l,l]$$

Partial sums of expressions ,

$v[l] - a[l,1]*v_1 , a[l,2]*v_2 , \dots$ can be calculated as soon as values of v_1 and v_2 are calculated .

Data Distribution



Processor 1
Calculates
Yellow

Processor 2
Calculates
Red

MPI – MV Results

Size	Speedup
1000	1.981
2000	1.988
4000	1.983
8000	1.984
10000	1.991

(Approximate density 20%)

MPI – Triangular Solve

Size	speedup
1000	1.612
4000	1.723
8000	1.767
10000	1.832

References

- A George, M Heath, J Liu, “Solution of Sparse Positive Definite Systems on a Shared-Memory Multiprocessor”, International Journal of Parallel Programming, Vol 15(4), 1986.
- R D Cunha, T Hopkins, “The Parallel Solution of Triangular Systems of Linear Equations”, 2nd Symposium in High Performance Computing, 1991.

Thank You