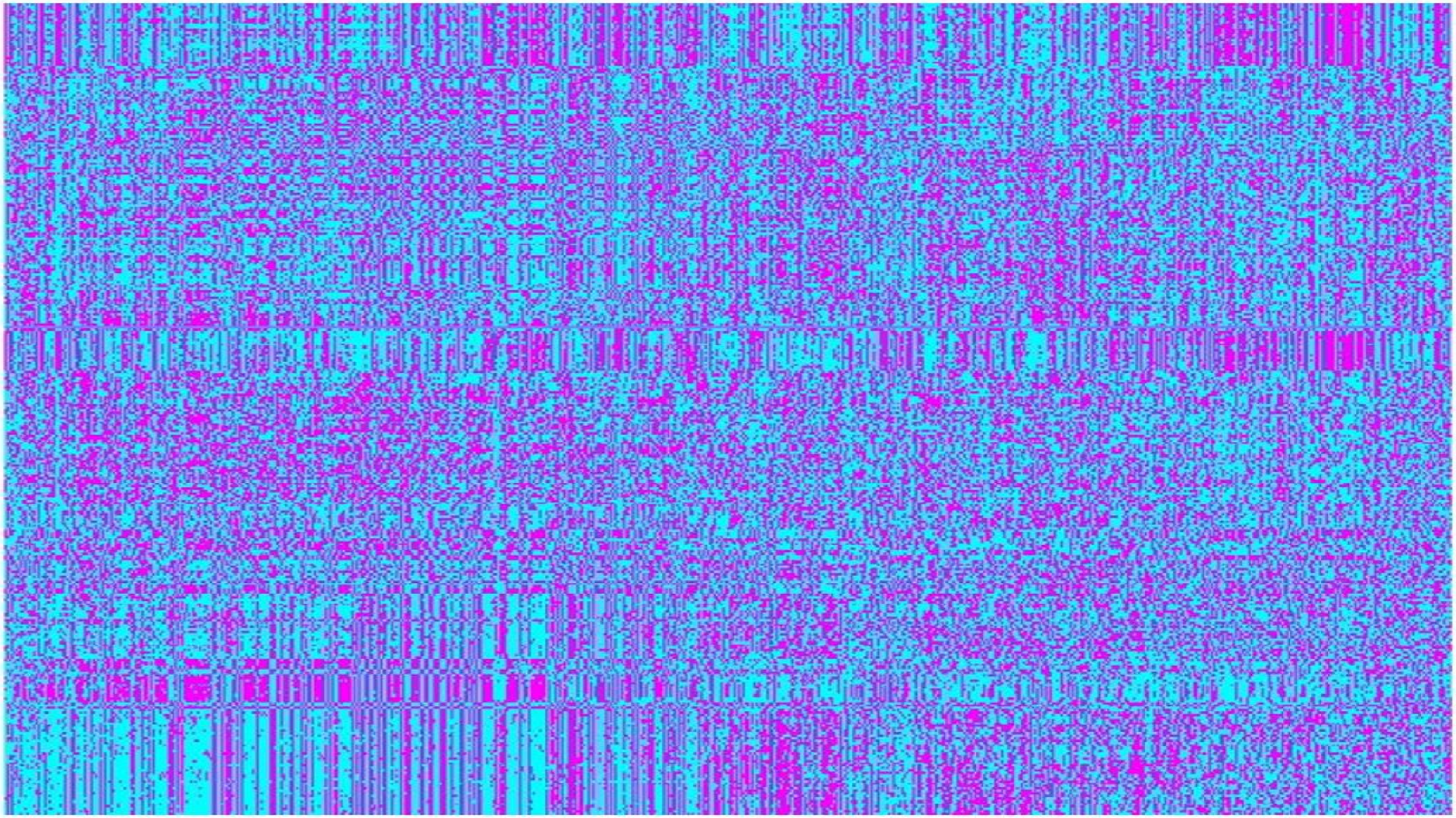
Implementation of Fast Biclustering Algorithm



Outline

- Theoretical Introduction
  - Clustering & Biclustering
  - Applications
  - Previous Attempts
  - Our algorithm
- High Performance Computing
  - Complexity
  - Problem Scale Matters
  - What makes our code fast.
- Results
- Further work

Clustering & Biclustering

- Clustering
- Suppose given the data below

1. REPUBLICAN VOTE FOR PRESIDENT[a]

State	Year																	
	00	04	08	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68
Alabama (AA)	35	21	24	8	22	31	27	48	14	13	14	18	19	35	39	42	70	14
Arkansas (AS)	35	40	37	20	28	39	29	39	13	18	21	30	21	44	46	43	44	31
Delaware (DE)	54	54	52	33	50	56	58	65	51	43	45	45	50	52	55	49	39	45
Florida (FA)	19	21	22	8	18	31	28	57	25	24	26	30	34	55	57	52	48	41
Georgia (GA)	29	18	31	4	7	29	18	43	8	13	15	18	18	30	33	37	54	30
Kentucky (KY)	49	47	48	25	47	49	49	59	40	40	42	45	41	50	54	54	36	44
Louisiana (LA)	21	10	12	5	7	31	20	24	7	11	14	19	17	47	53	29	57	23
Maryland (MD)	52	49	49	24	45	55	45	57	36	37	41	48	49	55	60	46	35	42
Mississippi (MI)	10	5	7	2	5	14	8	18	4	3	4	6	3	40	24	25	87	14
Missouri (MO)	46	50	49	30	47	55	50	56	35	38	48	48	42	51	50	50	36	45
North Car. (NC)	45	40	46	12	42	43	55	29	29	27	26	33	33	46	49	48	44	40
South Car. (SC)	7	5	6	1	2	4	2	9	2	1	4	4	4	49	25	49	59	39
Tennessee (TE)	45	43	46	24	43	51	44	54	32	31	33	39	37	50	49	53	44	38
Texas (TS)	31	22	22	9	17	24	20	52	11	12	19	17	25	53	55	49	37	40
Virginia (VA)	44	37	38	17	32	38	33	54	30	29	32	37	41	56	55	52	46	43
West Virginia (WV)	54	55	53	21	49	55	49	58	44	39	43	45	42	48	47	54	32	40

Clustering & Biclustering

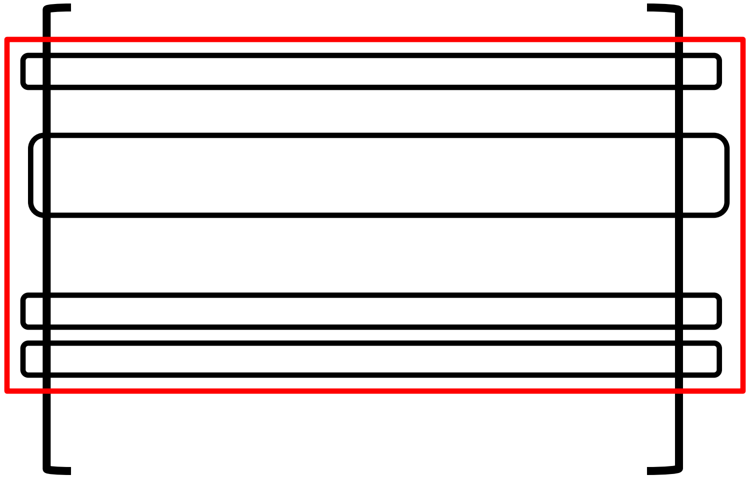
- Clustering
- Suppose given the data below

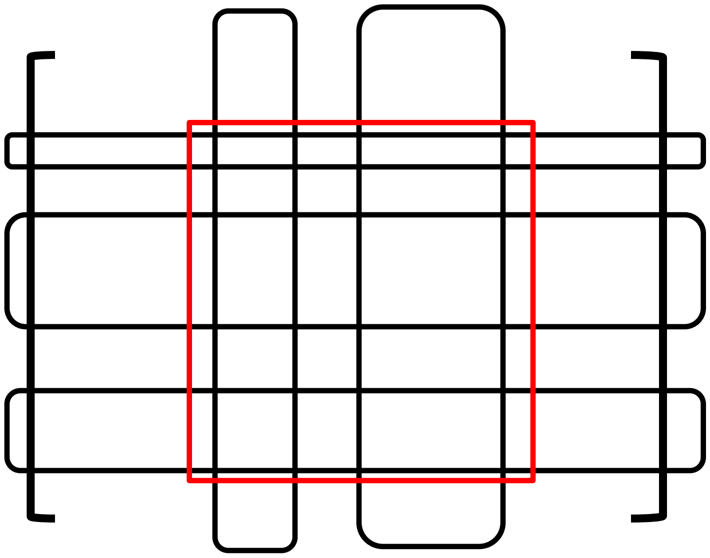
1. REPUBLICAN VOTE FOR PRESIDENT[a]

State	Year																	
	00	04	08	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68
Alabama (AA)	35	21	24	8	22	31	27	48	14	13	14	18	19	35	39	42	70	14
Arkansas (AS)	35	40	37	20	28	39	29	39	13	18	21	30	21	44	46	43	44	31
Delaware (DE)	54	54	52	33	50	56	58	65	51	43	45	45	50	52	55	49	39	45
Florida (FA)	19	21	22	8	18	31	28	57	25	24	26	30	34	55	57	52	48	41
Georgia (GA)	29	18	31	4	7	29	18	43	8	13	15	18	18	30	33	37	54	30
Kentucky (KY)	49	47	48	25	47	49	49	59	40	40	42	45	41	50	54	54	36	44
Louisiana (LA)	21	10	12	5	7	31	20	24	7	11	14	19	17	47	53	29	57	23
Maryland (MD)	52	49	49	24	45	55	45	57	36	37	41	48	49	55	60	46	35	42
Mississippi (MI)	10	5	7	2	5	14	8	18	4	3	4	6	3	40	24	25	87	14
Missouri (MO)	46	50	49	30	47	55	50	56	35	38	48	48	42	51	50	50	36	45
North Car. (NC)	45	40	46	12	42	43	55	29	29	27	26	33	33	46	49	48	44	40
South Car. (SC)	7	5	6	1	2	4	2	9	2	1	4	4	4	49	25	49	59	39
Tennessee (TE)	45	43	46	24	43	51	44	54	32	31	33	39	37	50	49	53	44	38
Texas (TS)	31	22	22	9	17	24	20	52	11	12	19	17	25	53	55	49	37	40
Virginia (VA)	44	37	38	17	32	38	33	54	30	29	32	37	41	56	55	52	46	43
West Virginia (WV)	54	55	53	21	49	55	49	58	44	39	43	45	42	48	47	54	32	40

Contd.

- Clustering:
  - Looking for similar rows
  - Low rank row sets
- Bi-Clustering:
  - Looking for rows and columns simultaneously
  - Low rank submatrices

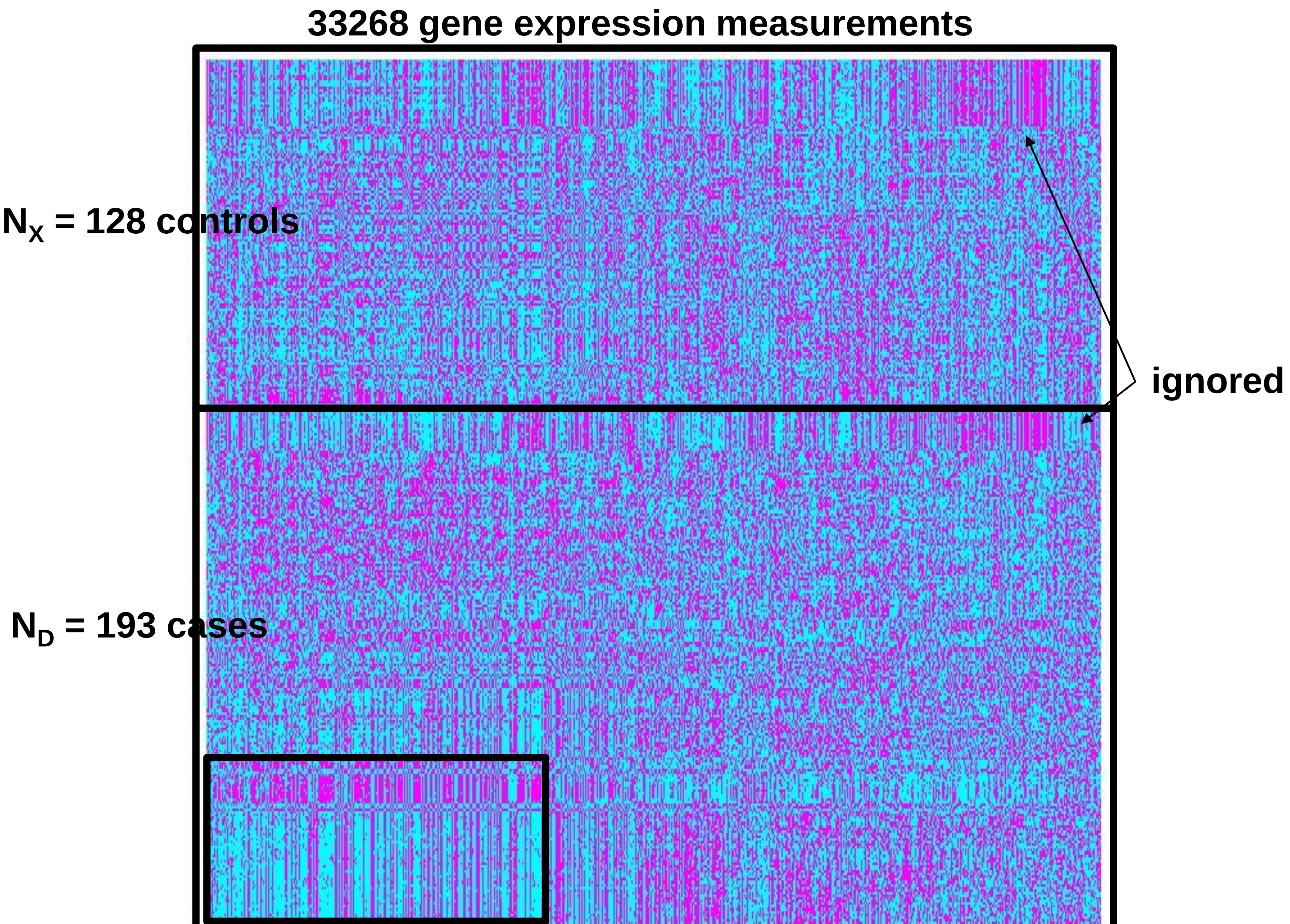




Application –Computational Biology

- Patients and Gene
- Cases and Controls

$$
\begin{array}{lll}
 & & \begin{array}{ccccc} \text{Gene 1} & \text{Gene 2} & \text{Gene 3} & \cdots\cdots & \text{Gene M} \end{array} \\
\textbf{Contrals} & \begin{array}{l} \text{Person 1} \\ \vdots \\ \text{Person L} \end{array} & \left[\qquad Z_{L\times M} \qquad \right] \\
\textbf{Cases} & \begin{array}{l} \text{Patient 1} \\ \text{Patient 2} \\ \vdots \\ \text{Patient N} \end{array} & \left[\qquad A_{N\times M} \qquad \right]
\end{array}
$$


33268 gene expression measurements
N_X = 128 controls
ignored
N_D = 193 cases

Previous Attempts

- Naïve way:
  - check every possible submatrics
  - NP
- Iterative random methods:
  - Start from a initial submatrix and add/remove rows and columns to make submatrix low rank
  - A good Initial Condition is needed
- Spectral biclustering:
  - Iteratively remove rows and columns to get a low rank submatrix ---using SVD in every step
  - $O(N^4) \times$ big constant

Our Algorithm

- For a given N×M matrix A, we are looking for some rows and columns whose intersection is a low rank matrix.
- We give every row and column a score then eliminate rows and columns with lower scores.

Let A be a $n \times m$ $\{1,-1\}$ matrix. Let A_i be the ith row and R_i be its score. Similarly we define B_j as the jth column and C_j as its scores.

Let Z be a $d \times m$ $\{1,-1\}$ matrix and Z_i and K_j be rows and columns.

Define $< \cdot, \cdot >$ as the product of 2 vectors.

Then we can write SCORES as:

$$R_i = \sum_j < A_j, A_i >^2 - \sum_j < Z_j, A_i >^2 \tag{1}$$

$$C_i = \sum_j < B_j, B_i >^2 - \sum_j < B_j, B_i >< K_j, K_i > \tag{2}$$

Algorithm Description

I. Pre-algorithm: Transform A into a binary matrix B

II. Main Algorithm

   1. Give a score to each rows and columns
   2. Remove rows and columns with lowest scores
   3. Recalculate the scores for all remaining rows and columns, go to step 2

III. The rows and columns remains at last forms a low rank submatrix.

Part 2: High Performance Computing

Complexity

- 2N interations
- Calculate 2N scores in every step
- Recall Scores:

$$R_i = \sum_j < A_j, A_i >^2 - \sum_j < Z_j, A_i >^2$$

$$C_i = \sum_j < B_j, B_i >^2 - \sum_j < B_j, B_i >< K_j, K_i >$$

O(N^3) $\Longrightarrow$ **O(N^4)**

Contd.

- Luckily, we found a way to not recalculate the score, but update them
- When a row(col) is removed, there is only a small part in each score changes, we only need to remove these parts from scores.
- When a row is removed:

$$R'_i = \quad R_i - < A_i, a >^2 \ i$$
$$C'_i = \quad C_i - m - 2 \sum_{A_k \neq a} < A_k, a > a_i a_{ki} + \sum_{Z_k} < Z_k, a > a_k z_{ki}$$

- When a column is removed:

$$C'_i = \quad C_i - < B_i, b > [< B_i, b > - < K_i, k >]$$
$$R'_i = \quad R_i - n + d - 2 \sum_{B_k \neq b} b_i a_{ik} [< B_k, b > - < K_k, k >]$$

Contd.

- Luckily, we found a way to not recalculate the score, but update them
- When a row(col) is removed, there is only a small part in each score changes, we only need to remove these parts from scores.

$$
\begin{tikzcd}
O(N^3) \arrow[rr, Rightarrow] \arrow[d, Rightarrow] & & O(N^4) \\
O(N^2) \arrow[rr, Rightarrow] & & O(N^3)
\end{tikzcd}
$$

Complexity

- For a n $\times$ m matrix, the main body of our algorithm is:

1. Initialize scores — $O((n+m)nm)$
2. Remove according to scores — $O(m)$ or $O(n)$
3. Update scores, and go to step 2 — $O(nm)$

Steps 2–3: $O((n+m)nm)$

$O((n+m)nm)$

Memory bound

Problem Scale

- ASD (Sample data) $10^2 \times 10^4$
- Bipolar disorder $10^4 \times 10^6$
- Cancer $10^5 \times 10^7$

Memory Management

- Binary data can be compressed

- 32 entries:

  11101001000101011110101000100011

  $\Downarrow$

  1 unsigned int

- Bipolar disorder $10^4 \times 10^6$ 1.1 GB
- Cancer $\cancel{10^5 \times 10^7}$ 110 GB

- ~~100GB RAM~~ or ~~read 100GB 10^7 times~~ or ~~MPI~~

What makes our code fast !

- How to measure speed

$$\text{Time Per Unit} = \frac{\text{runnint time/s}}{nm(n+m)}$$

$$\text{Speedup} = \frac{\text{sequential time/s}}{\text{parallel time/s}}$$

- Test data:
  - ASD Sample data
  - Random matrix

What makes our code fast !

- Sequential version 0 MATLAB

SUPERSLOW

What makes our code fast !

- Sequential version 1 C
- ASD data:
  - Running time: 120s
  - TPU: 8.56239321e-9

What makes our code fast !

- Sequential version 2 C
- Basic optimization
- Improvement on Bitwise operation
  - Popcount
    - Counting bits in parallel
- ASD data:
  - Running time: 80s
  - TPU: 5.70826214e-9

What makes our code fast !

- Sequential version 3 C
- Loop structure optimization
  - Loop unrolling
    - For(i=0;i<16;i++)

      Operation[i]

    -> Operation[0];

    Operation[1];

    ……

    Operation[15];
- ASD data:
  - Running time: 38s
  - TPU: 2.71142452e-9

What makes our code fast !

- Sequential version 4
- Improvement in memory access
  - Use of cache

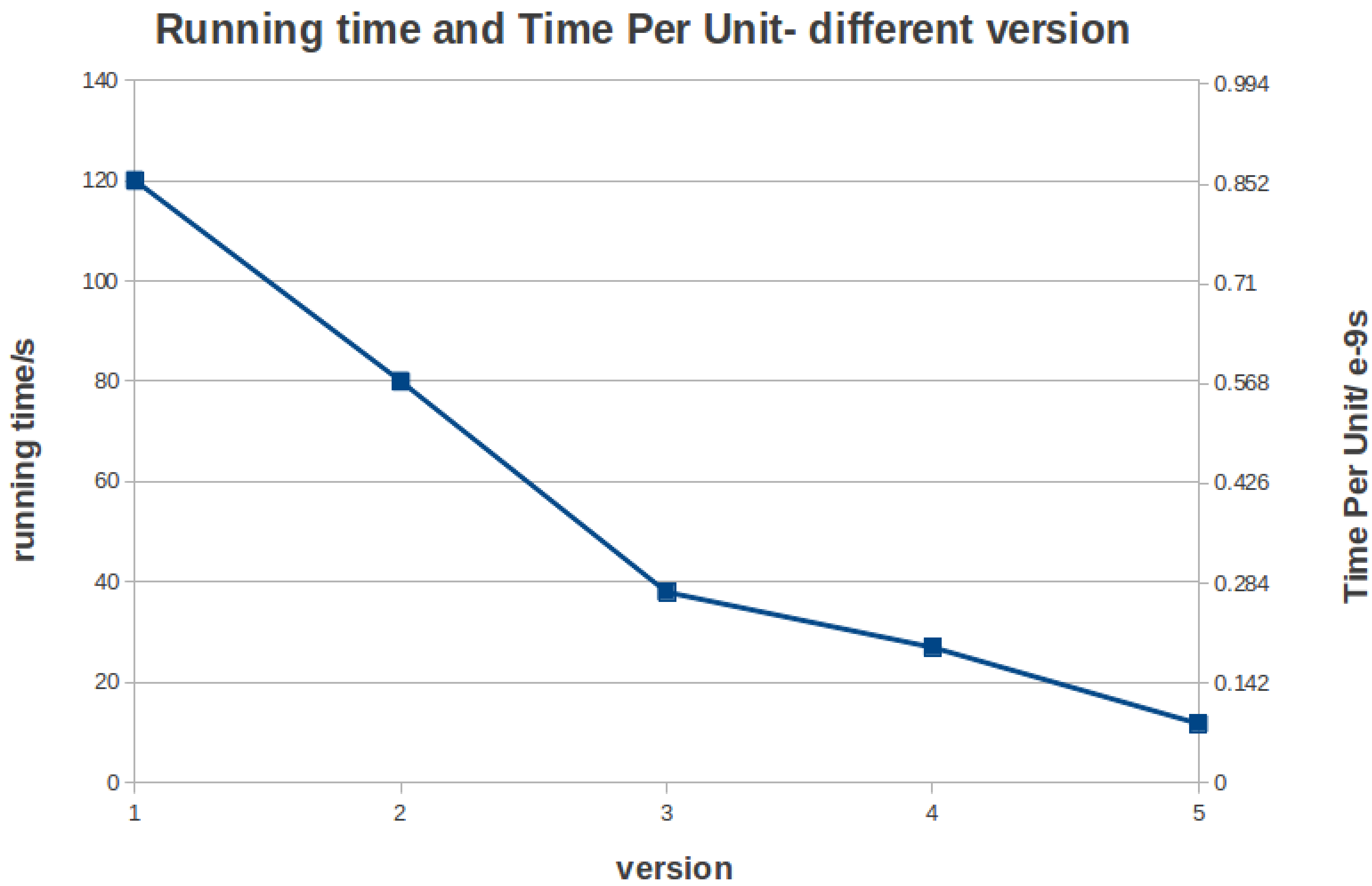
```
        Operation[0,array[s]];
        Operation[1,array[s]];
        ……
        Operation[15,array[s]];
    ->
        temp = array[s];
        Operation[0,temp];
        Operation[1,temp];
        ……
        Operation[15,temp];
```

- ASD data:
  - Running time: 27s
  - TPU: 1.92653847e-9

What makes our code fast !

- Sequential version 5
- Simply add “–O3”
- ASD data:
  - Running time: 17.8s
  - TPU: 1.27008833e-9

Contd.



Running time and Time Per Unit- different version
140
120
100
80
60
40
20
0
0.994
0.852
0.71
0.568
0.426
0.284
0.142
0
1
2
3
4
5
running time/s
Time Per Unit/ e-9s
version

What makes our code fast !

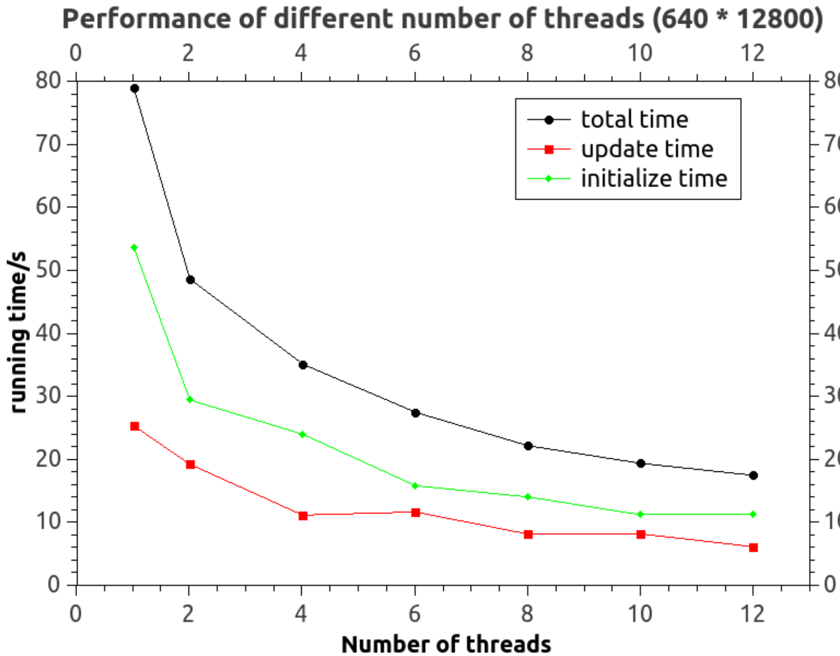
- OpenMP parallel version
- Choose the right loop to parallel
- Avoid synchronization

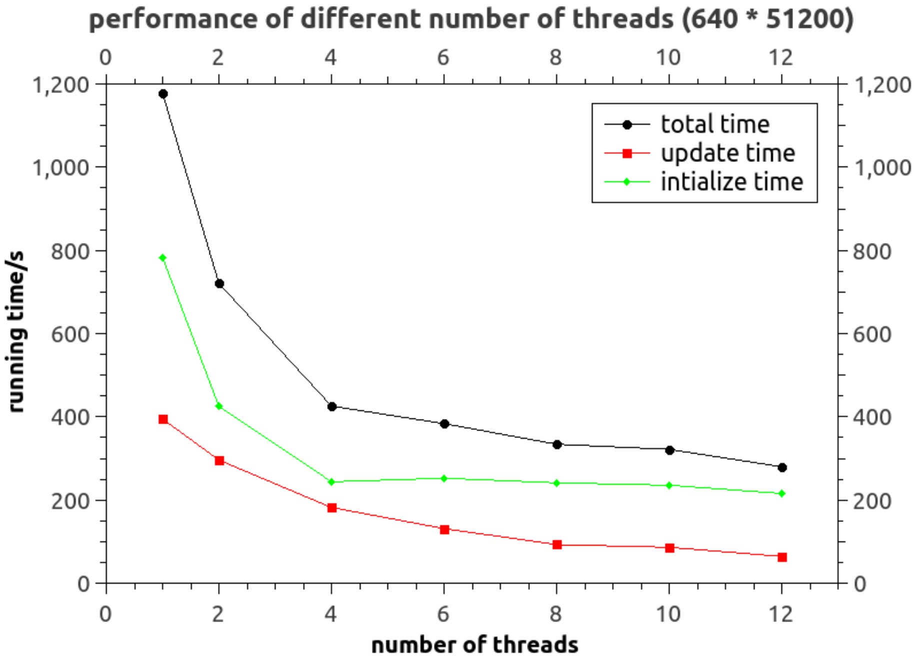
- ASD data:
  - 4-core
  - Running time: 6.01703s
  - TPU: 4.2933267e-10
  - Speedup: 2.9

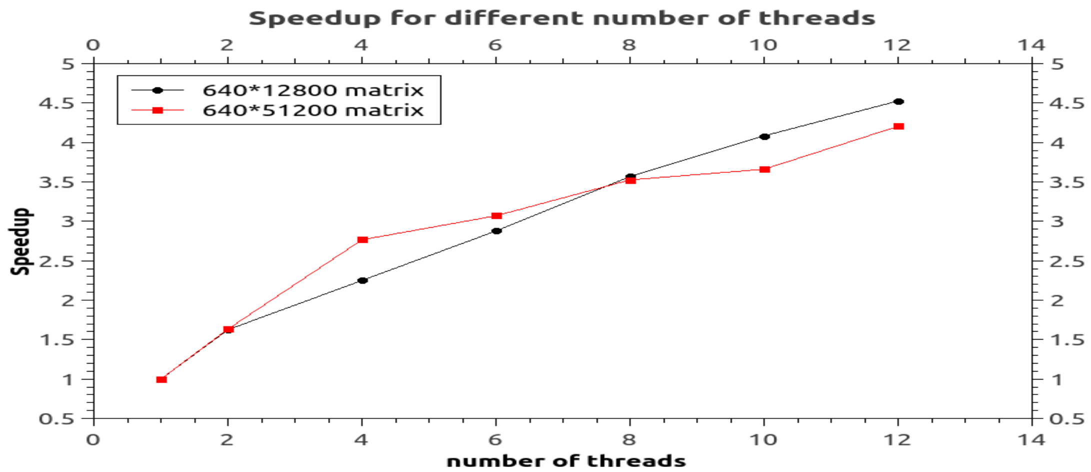
  - 12-core
  - Running time: 3.14378s
  - TPU: 2.2404929e-10
  - Speedup: 5.6

What makes our code fast !

- OpenMP version – core numbers

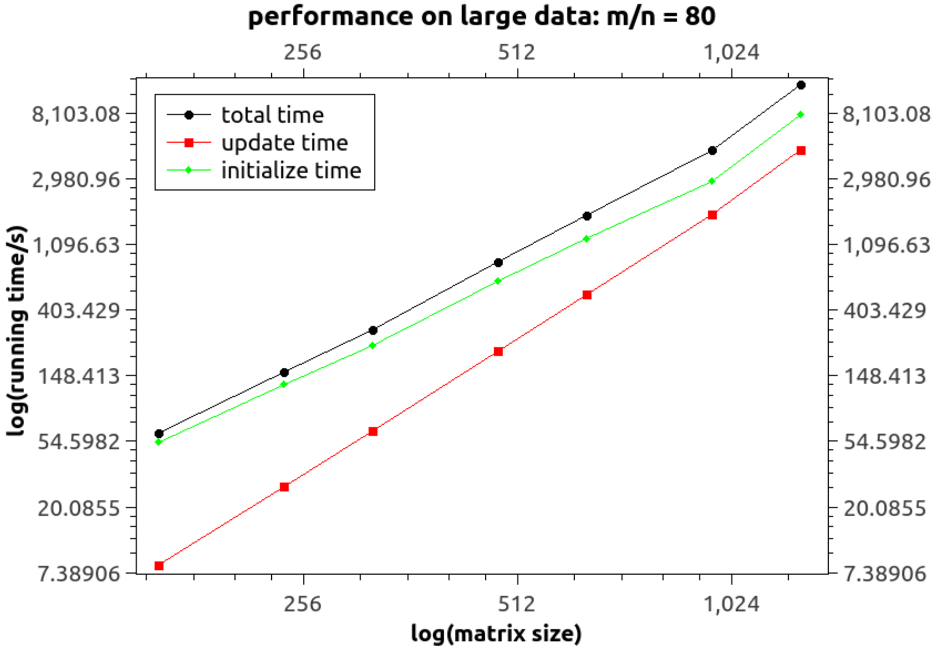


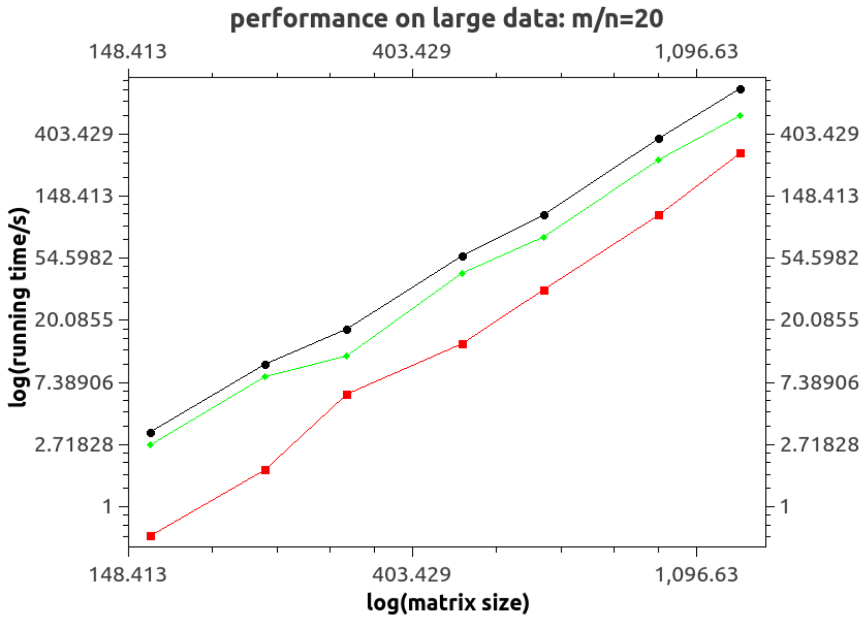




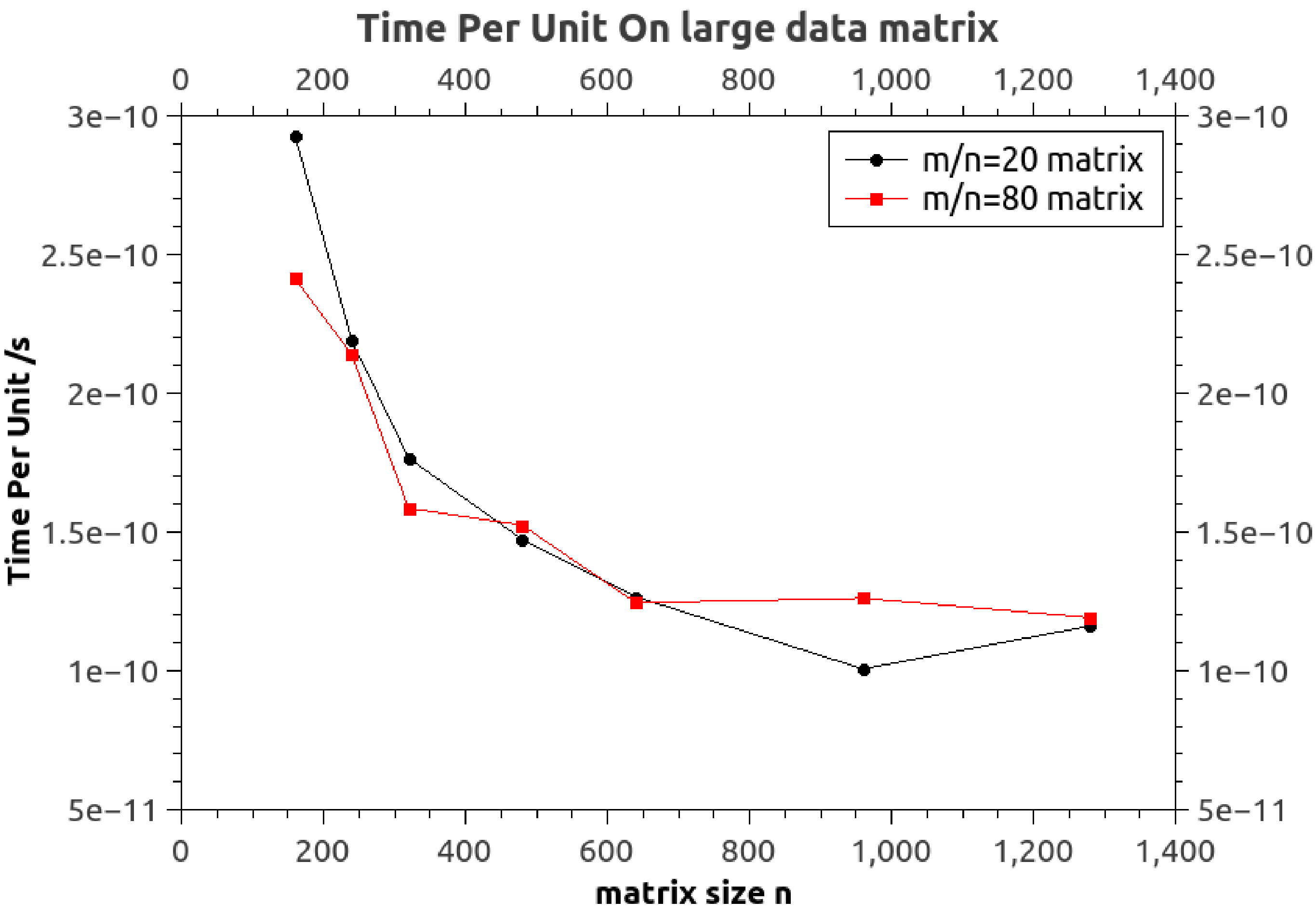
What makes our code fast !

- OpenMP version – Large Data (random matrix)
- Different ratio of n and m





Contd.



Time Per Unit On large data matrix
m/n=20 matrix
m/n=80 matrix
Time Per Unit /s
matrix size n
0
200
400
600
800
1,000
1,200
1,400
3e−10
2.5e−10
2e−10
1.5e−10
1e−10
5e−11

What makes our code fast !

- OpenMP version – Large Data (random matrix)
- Bipolar disorder $10^4 \times 10^6$ 1 week
- Running time estimation

$$\mathrm{TPU} \times nm(n+m) = 1.2 \times 10^{-10} \times 10240 \times 819200 \times 829440$$

$$= 834941\mathrm{s} = 231\mathrm{h} = 9\mathrm{d}$$

What's next…

- Run our code on Bipolar disorder data $10^4 \times 10^6$
- Generalization of the original bicluster algorithm !

- MPI version
- A way has been found

Even more ambitious…

- Possible MPI version
- Just found a way of avoiding to much data transfer between different computer.
  - Only O(n) MPI_Reduction in every iteration
- Separately store data

- Run on $10^5 \times 10^7$ Cancer data **Possible!**
- Still very challenging
  - Ideally, with currant efficiency, given 100 nodes each with 12 core. Estimated:

END

- Thank you.