Alignment and clustering tools for sequence analysis

Omar Abudayyeh

18.337 Presentation

December 9, 2015

Introduction

- Sequence comparison is critical for inferring biological relationships within large datasets of DNA or protein sequences
- Next generation sequencing has generated **too much** data
- Need for **fast** and **accurate** tools for comparing DNA or protein sequences

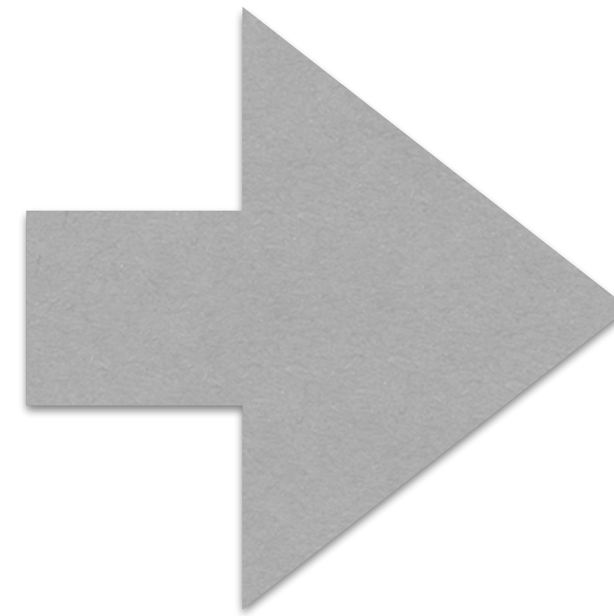
Available sequence comparison tools

Similarity Metrics

edit distance

dynamic programming (needleman-wunsch, smith waterman)

k-tuple (FASTA, BLAST)



Clustering

greedy (UCLUST, CD-HIT)

graph (markov clustering)

vector (k-means)

hierarchical

Outline

- **1. Smith-waterman local alignment**
  - *Serial and parallel implementations in Julia*

- **2. Markov clustering**
  - *Parallelized linear algebra implementation in Julia*

1. Smith-waterman local alignment

Introduction to local Smith-waterman alignment

- Traditional string matching is not useful for comparing DNA or protein sequences due to evolutionary events
- Traditional alignment is assessed through cost function (e.g. edit distance) or stochastic similarity scores (e.g. ML through HMM)
- These approaches all involve dynamic programming, but this can be costly for large problems ~ O(MN)
- Smith-waterman is highly amenable to parallelism due to specific data dependencies in the matrix

Smith-waterman algorithm

- N x M integer matrix, where N and M are sequence lengths

1. Initialize matrix

$$H(i,0) = 0,\ 0 \le i \le m$$
$$H(0,j) = 0,\ 0 \le j \le n$$

	^	A	T	G	C	A	T	G	C	A	T	G	C
^	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0												
T	0												
G	0												
G	0												
G	0												
C	0												
A	0												
T	0												
G	0												

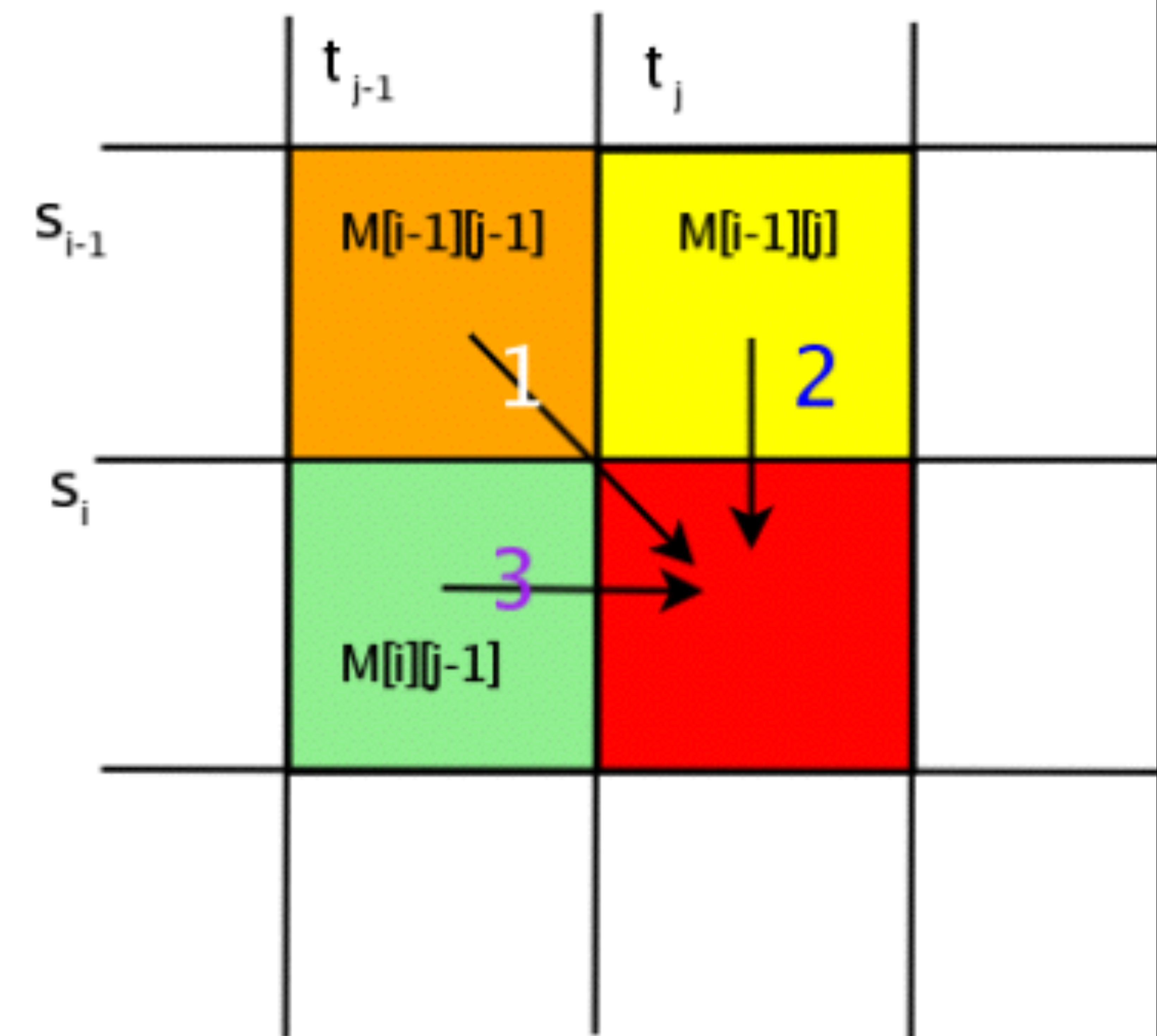
2. Fill Matrix

$$H(i,j) = \max \begin{Bmatrix} 0 & \\ H(i-1,j-1) + \ s(a_i,b_j) & \text{Match/Mismatch} \\ \max_{k\ge1}\{H(i-k,j) + \ W_k\} & \text{Deletion} \\ \max_{l\ge1}\{H(i,j-l) + \ W_l\} & \text{Insertion} \end{Bmatrix},\ 1 \le i \le m, 1 \le j \le n$$

3. Traceback Path

$H_{opt} = \max(H[i,j])$

$\text{traceback}(H_{opt})$



Smith-waterman example

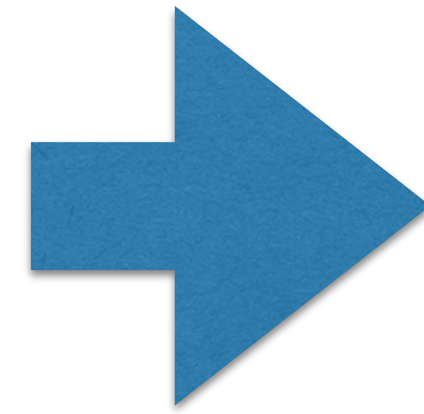
seq1 = "ATGCATGCATGC"
seq2 = "ATGGGCATG"

	^	A	T	G	C	A	T	G	C	A	T	G	C
^	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	2	1	0	0	2	1	0	0	2	1	0	0
T	0	1	4	3	2	1	4	3	2	1	4	3	2
G	0	0	3	6	5	4	3	6	5	4	3	6	5
G	0	0	2	5	5	4	3	5	5	4	3	5	5
G	0	0	1	4	4	4	3	5	4	4	3	5	4
C	0	0	0	3	6	5	4	4	7	6	5	4	7
A	0	2	1	2	5	8	7	6	6	9	8	7	6
T	0	1	4	3	4	7	10	9	8	8	11	10	9
G	0	0	3	6	5	6	9	12	11	10	10	13	12

<table>
<tr><th></th><th>^</th><th>A</th><th>T</th><th>G</th><th>C</th><th>A</th><th>T</th><th>G</th><th>C</th><th>A</th><th>T</th><th>G</th><th>C</th></tr>
<tr><th>^</th><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td></tr>
<tr><th>A</th><td>N</td><td>M</td><td>-</td><td>-</td><td>-</td><td>M</td><td>-</td><td>-</td><td>-</td><td>M</td><td>-</td><td>-</td><td>-</td></tr>
<tr><th>T</th><td>N</td><td>|</td><td>M</td><td>-</td><td>-</td><td>-</td><td>M</td><td>-</td><td>-</td><td>-</td><td>M</td><td>-</td><td>-</td></tr>
<tr><th>G</th><td>N</td><td>|</td><td>|</td><td>M</td><td>-</td><td>-</td><td>-</td><td>M</td><td>-</td><td>-</td><td>-</td><td>M</td><td>-</td></tr>
<tr><th>G</th><td>N</td><td>-</td><td>|</td><td>|</td><td>M</td><td>-</td><td>-</td><td>|</td><td>M</td><td>-</td><td>-</td><td>|</td><td>M</td></tr>
<tr><th>G</th><td>N</td><td>-</td><td>|</td><td>|</td><td>|</td><td>M</td><td>-</td><td>M</td><td>-</td><td>M</td><td>-</td><td>M</td><td>-</td></tr>
<tr><th>C</th><td>N</td><td>-</td><td>|</td><td>|</td><td>M</td><td>-</td><td>-</td><td>|</td><td>M</td><td>-</td><td>-</td><td>-</td><td>M</td></tr>
<tr><th>A</th><td>N</td><td>M</td><td>-</td><td>|</td><td>|</td><td>M</td><td>-</td><td>-</td><td>|</td><td>M</td><td>-</td><td>-</td><td>-</td></tr>
<tr><th>T</th><td>N</td><td>|</td><td>M</td><td>-</td><td>|</td><td>|</td><td>M</td><td>-</td><td>-</td><td>|</td><td>M</td><td>-</td><td>-</td></tr>
<tr><th>G</th><td>N</td><td>|</td><td>|</td><td>M</td><td>-</td><td>|</td><td>|</td><td>M</td><td>-</td><td>-</td><td>|</td><td>M</td><td>-</td></tr>
</table>

Smith-waterman example

seq1 = "ATGCATGCATGC"
seq2 = "ATGGGCATG"

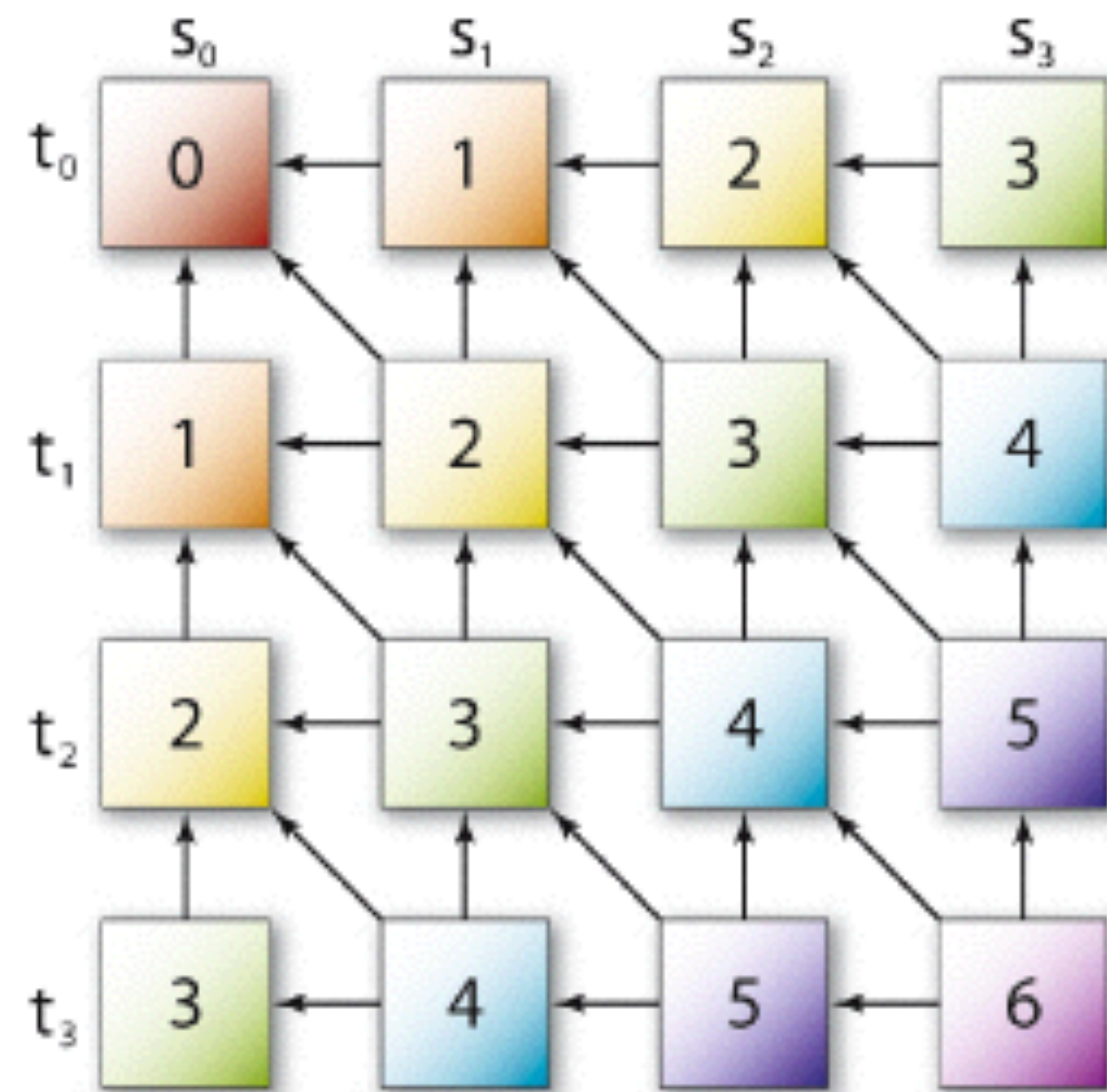


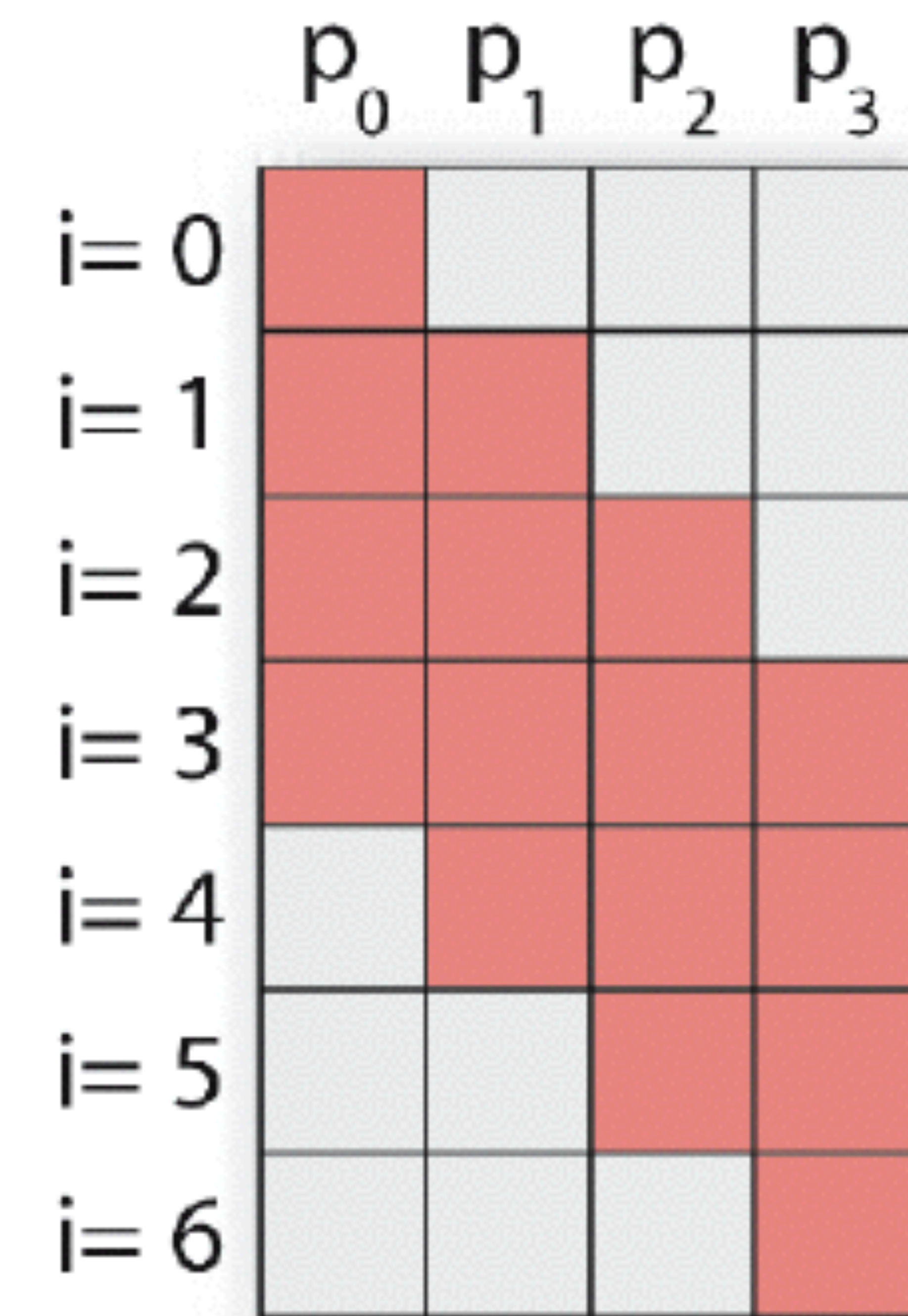
ATGCATGCATG
ATGG—GCATG

	^	A	T	G	C	A	T	G	C	A	T	G	C
^	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	**2**	1	0	0	2	1	0	0	2	1	0	0
T	0	1	**4**	3	2	1	4	3	2	1	4	3	2
G	0	0	3	**6**	5	4	3	6	5	4	3	6	5
G	0	0	2	5	**5**	4	3	5	5	4	3	5	5
G	0	0	1	4	4	**4**	**3**	**5**	4	4	3	5	4
C	0	0	0	3	6	5	4	4	**7**	6	5	4	7
A	0	2	1	2	5	8	7	6	6	**9**	8	7	6
T	0	1	4	3	4	7	10	9	8	8	**11**	10	9
G	0	0	3	6	5	6	9	12	11	10	10	**13**	12

<table>
<tr><th></th><th>^</th><th>A</th><th>T</th><th>G</th><th>C</th><th>A</th><th>T</th><th>G</th><th>C</th><th>A</th><th>T</th><th>G</th><th>C</th></tr>
<tr><td>^</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td></tr>
<tr><td>A</td><td>N</td><td>M</td><td>-</td><td>-</td><td>-</td><td>M</td><td>-</td><td>-</td><td>-</td><td>M</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>T</td><td>N</td><td>|</td><td>M</td><td>-</td><td>-</td><td>-</td><td>M</td><td>-</td><td>-</td><td>-</td><td>M</td><td>-</td><td>-</td></tr>
<tr><td>G</td><td>N</td><td>|</td><td>|</td><td>M</td><td>-</td><td>-</td><td>-</td><td>M</td><td>-</td><td>-</td><td>-</td><td>M</td><td>-</td></tr>
<tr><td>G</td><td>N</td><td>-</td><td>|</td><td>|</td><td>M</td><td>-</td><td>-</td><td>|</td><td>M</td><td>-</td><td>-</td><td>|</td><td>M</td></tr>
<tr><td>G</td><td>N</td><td>-</td><td>|</td><td>|</td><td>|</td><td>M</td><td>-</td><td>M</td><td>-</td><td>M</td><td>-</td><td>M</td><td>-</td></tr>
<tr><td>C</td><td>N</td><td>-</td><td>|</td><td>|</td><td>M</td><td>-</td><td>-</td><td>|</td><td>M</td><td>-</td><td>-</td><td>-</td><td>M</td></tr>
<tr><td>A</td><td>N</td><td>M</td><td>-</td><td>|</td><td>|</td><td>M</td><td>-</td><td>-</td><td>|</td><td>M</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>T</td><td>N</td><td>|</td><td>M</td><td>-</td><td>|</td><td>|</td><td>M</td><td>-</td><td>-</td><td>|</td><td>M</td><td>-</td><td>-</td></tr>
<tr><td>G</td><td>N</td><td>|</td><td>|</td><td>M</td><td>-</td><td>|</td><td>|</td><td>M</td><td>-</td><td>-</td><td>|</td><td>M</td><td>-</td></tr>
</table>

Parallel Implementation of SW





- Sequentially assign anti-diagonal elements to processers
- With p=min(m,n) processors, DP table can be computed in (m + n -1) passes
- Some inefficiency due to processor stalling equal to p(p-1)

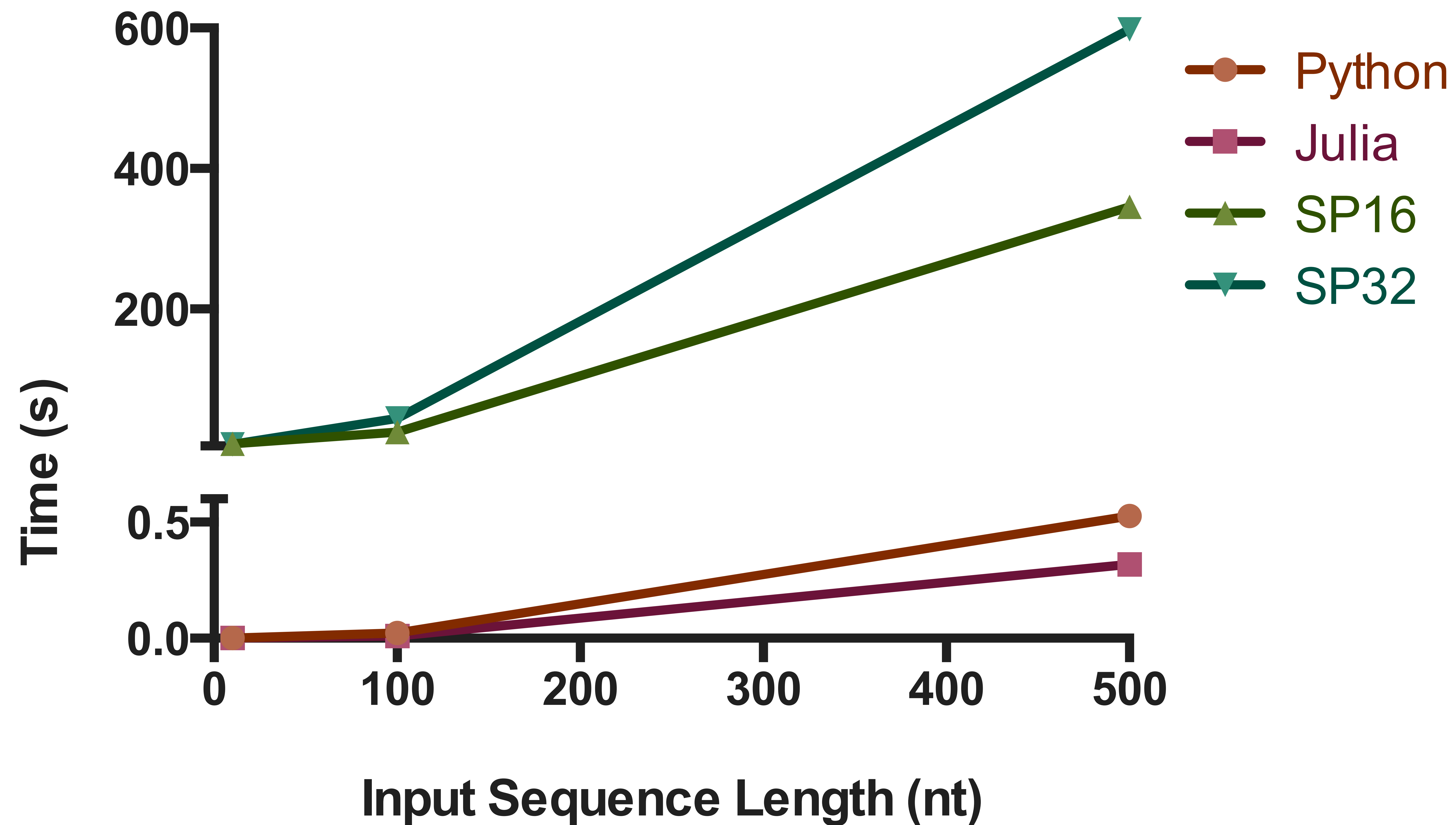
Liu et al. ICCS 2006

Parallel Implementation of SW

```
for j = 2:col
        jcol = j
        irow = 2
        @sync begin
            count = 1
            w = workers()
            while jcol > 1 && irow < row + 1
                    @async remotecall_wait(w[count],shared_get_score!,arguments)
                    jcol -= 1
                    irow += 1
                    count += 1
            end
        end
end
```

- Implemented this with normal arrays and shared arrays on a 40 core machine

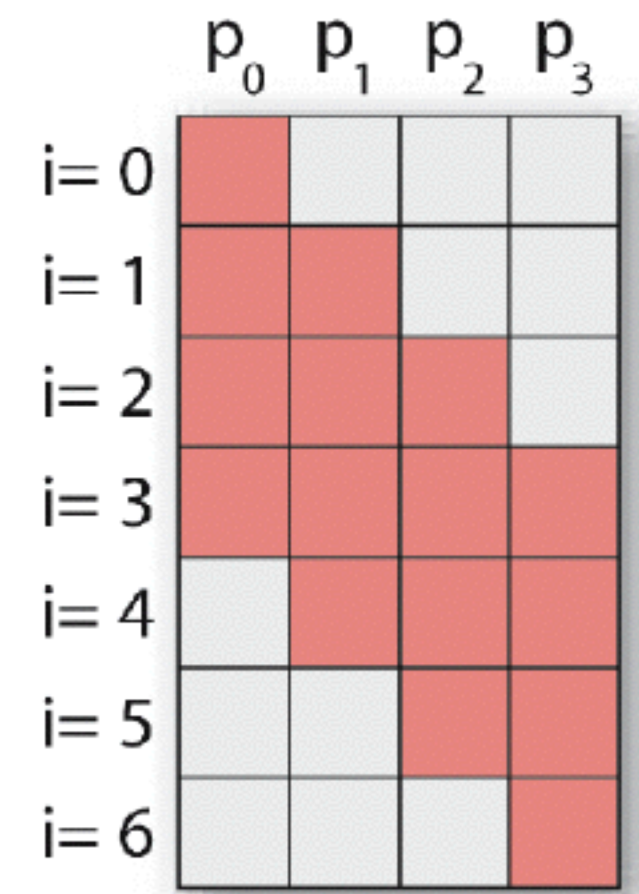
Performance of SW



- Parallel SW is **~1,880x slower**, but Julia serial SW is **~2.5x faster** than python

Outlook

- Overhead too large for parallelism, but serial algorithm in Julia outperforms python
- Try GPU computation with more cores (Julia CUDA and OpenCL)
- Eliminate processor stalling by interleaving requests
- Parallelize other database alignments, such as BLAST
- Add support for protein alignment



2. Markov clustering

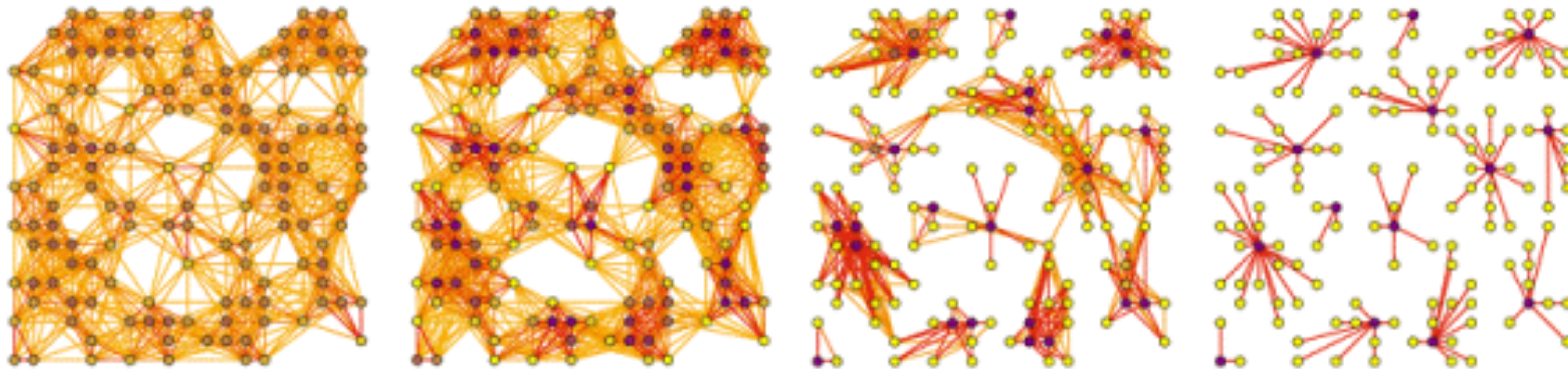
Introduction to markov clustering

- Markov clustering algorithm originally developed for graph clustering and is now a key tool within bioinformatics
- Useful for determining clusters in networks (e.g. protein interactions can help identify genes in disease such as cancer)
- With next generation sequencing technologies, there are vast amounts of data
- **Performance** and **scalability issues** are limiting factors

Van Dongen, S. A cluster algorithm for graphs, *Information Systems*

Markov-clustering overview

- Markov clustering is a simulation of random walks
- After enough walks, flows in the graph become evident and correspond to clusters



Markov-clustering Algorithm

Two step process: where M is the transition matrix of a weighted, undirected graph

1. Expansion

$$Exp(M) = M^p.$$

2. Inflation

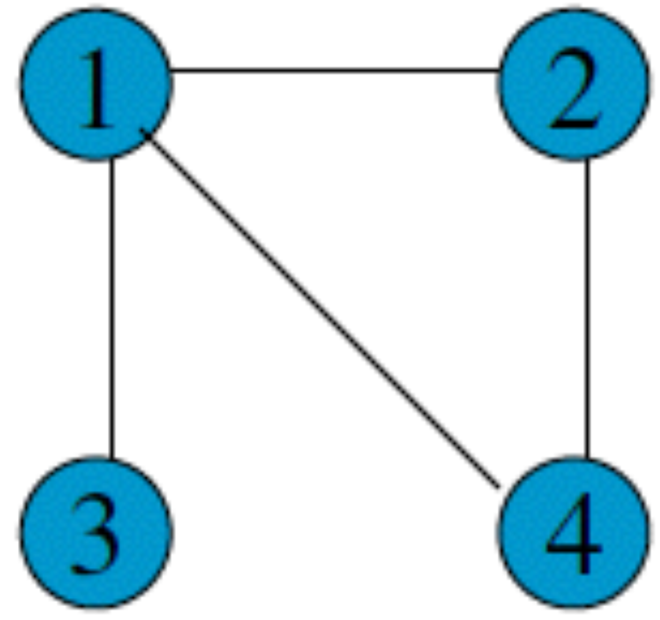
$$(\Gamma_r M)_{ij} = (M_{ij})^r / \sum_{k=1}^{m} (M_{kj})^r; i = 1...m, j = 1...n.$$

Markov-clustering Algorithm

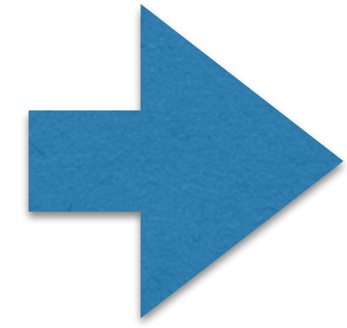
Algorithm:

1. Start with transition matrix
2. Normalize the matrix
3. Expand by taking the pth power of the matrix
4. Inflate by taking the inflation of the matrix with parameter r
5. Repeat steps 3 and 4 until steady state is reached
6. Analyze matrix for clusters

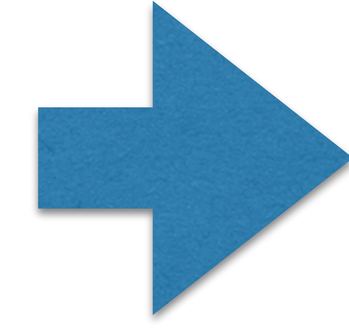
Markov clustering example



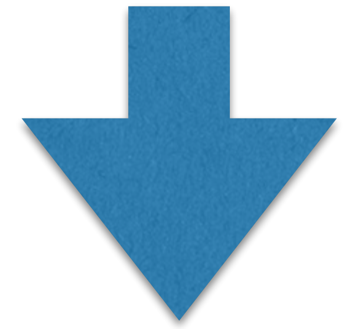
Power of 2
Inflation of 2



$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

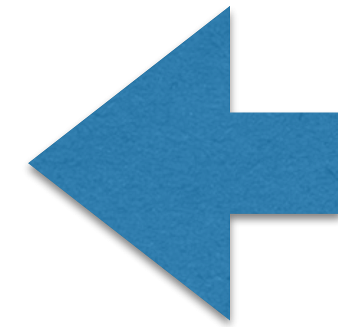


$$\begin{pmatrix} 1/4 & 1/3 & 1/2 & 1/3 \\ 1/4 & 1/3 & 0 & 1/3 \\ 1/4 & 0 & 1/2 & 0 \\ 1/4 & 1/3 & 0 & 1/3 \end{pmatrix}$$



$$\begin{pmatrix} 1/4 & 1/3 & 1/2 & 1/3 \\ 1/4 & 1/3 & 0 & 1/3 \\ 1/4 & 0 & 1/2 & 0 \\ 1/4 & 1/3 & 0 & 1/3 \end{pmatrix} \begin{pmatrix} 1/4 & 1/3 & 1/2 & 1/3 \\ 1/4 & 1/3 & 0 & 1/3 \\ 1/4 & 0 & 1/2 & 0 \\ 1/4 & 1/3 & 0 & 1/3 \end{pmatrix}$$

$$= \begin{pmatrix} .35 & .31 & .38 & .31 \\ .23 & .31 & .13 & .31 \\ .19 & .08 & .38 & .08 \\ .23 & .31 & .13 & .31 \end{pmatrix}$$



$$\begin{pmatrix} .94 & .33 & .50 & .33 \\ .03 & .33 & \text{--} & .33 \\ .01 & \text{--} & .50 & \text{--} \\ .13 & .33 & \text{--} & .33 \end{pmatrix}$$

$$\begin{pmatrix} 1 & .33 & .50 & .33 \\ \text{--} & .33 & \text{--} & .33 \\ \text{--} & \text{--} & .50 & \text{--} \\ \text{--} & .33 & \text{--} & .33 \end{pmatrix}$$

Parallelizing markov clustering

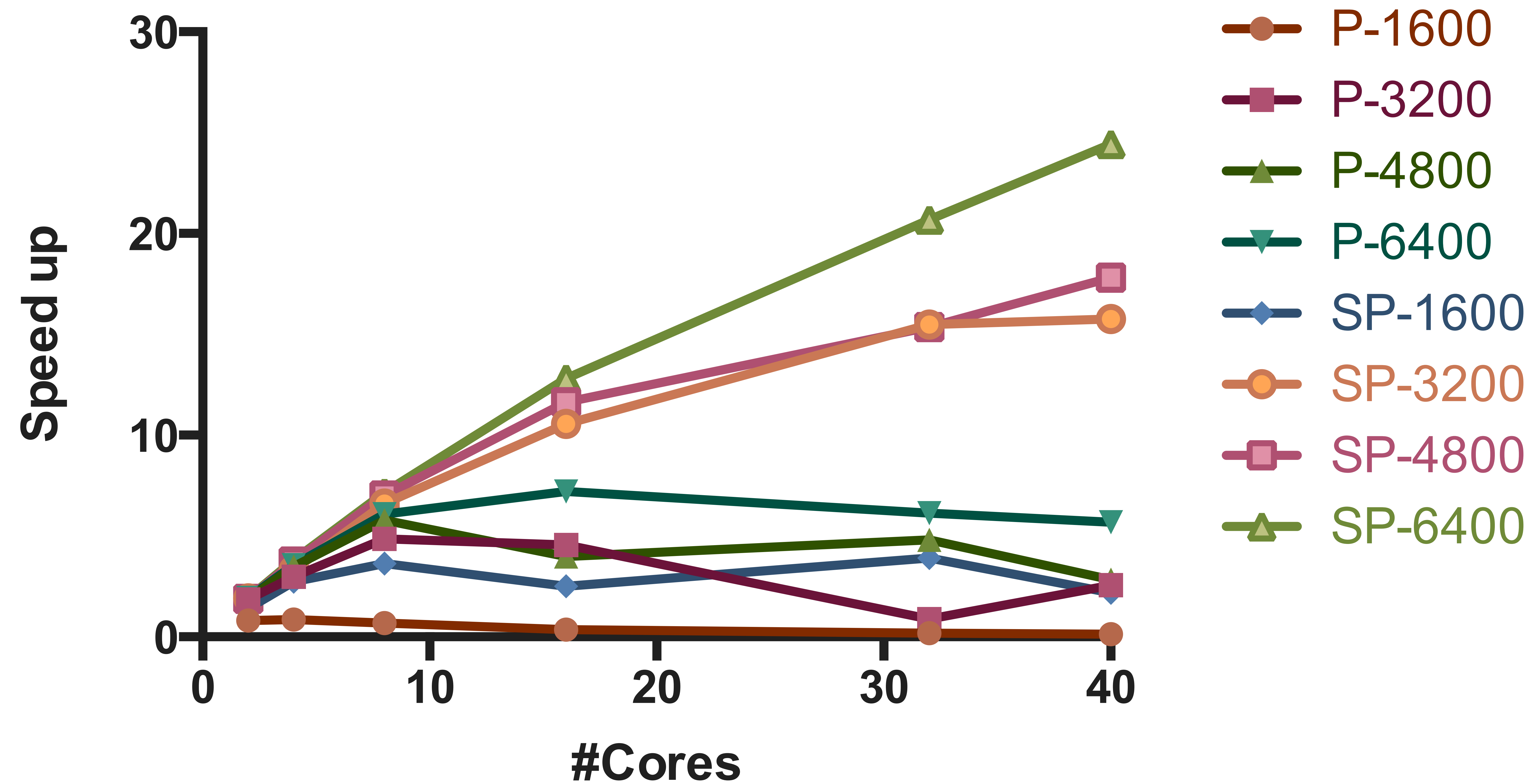
- MCL is $O(N^3)$, where N is number of vertices
  - Cost due to matrix multiplication (inflation can be done in $O(N^2)$)
- Because algorithm is just basic linear algebra operations, it's highly amenable for parallelization
- Implemented parallelized version of expansion and compared performance

Bustamam et al. IEEE 2010 HPC.

MCL algorithm parallelized expansion

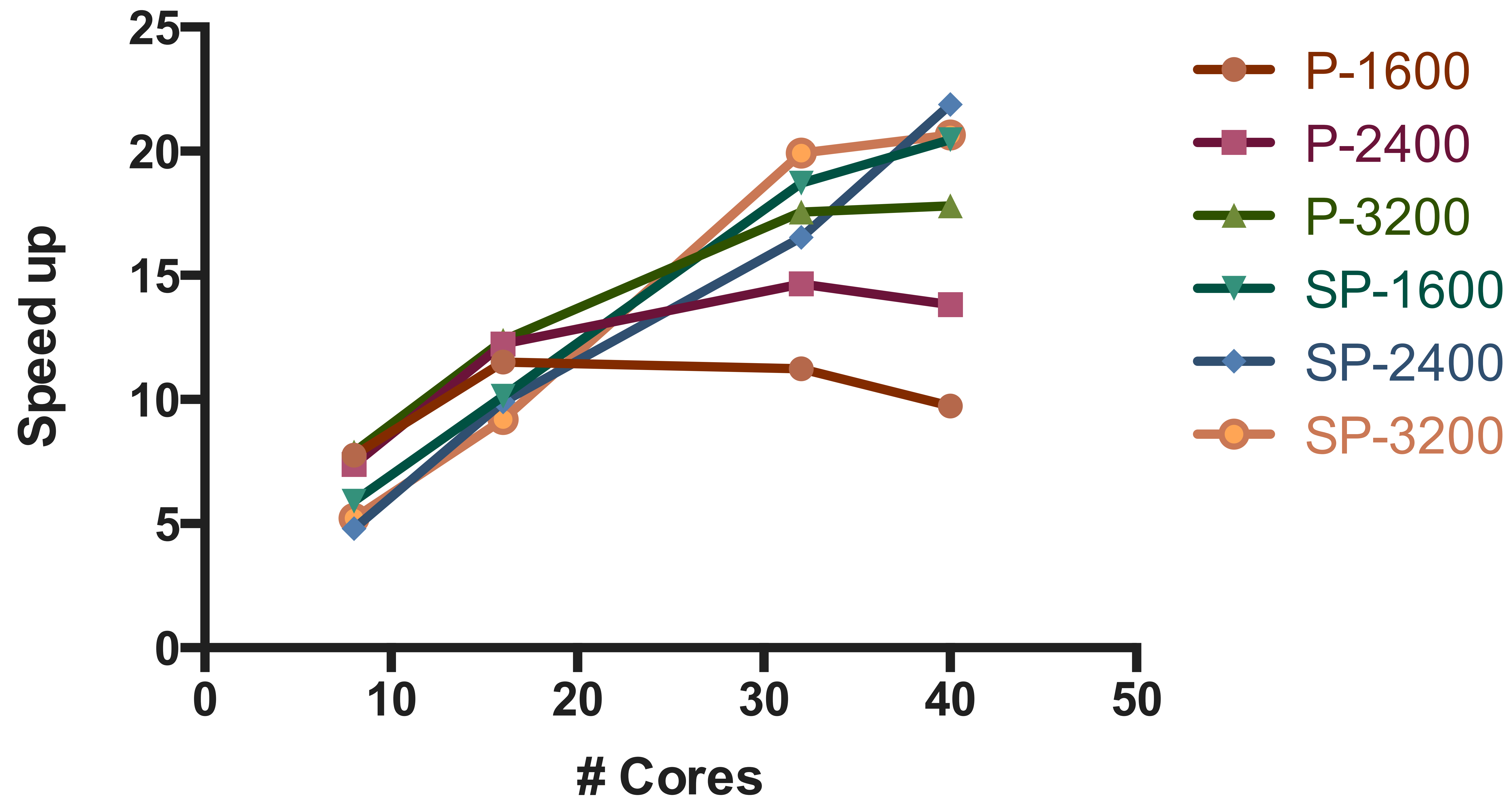
```
@everywhere function mymatmul!(n,w,sa,sb,sc,p)
    range = 1+(w-1) * div(n,p) : (w) * div(n,p)
    sc[:,range] = sa[:,:] * sb[:,range]
end

function sharedmult(n,p,sa,sb,sc)
    @sync begin
        for (i,w) in enumerate(workers())
            @async remotecall_wait(w, mymatmul!, n, i, sa, sb, sc,p)
        end
    end
    return sc
end
```

Performance of parallel matrix multiplication



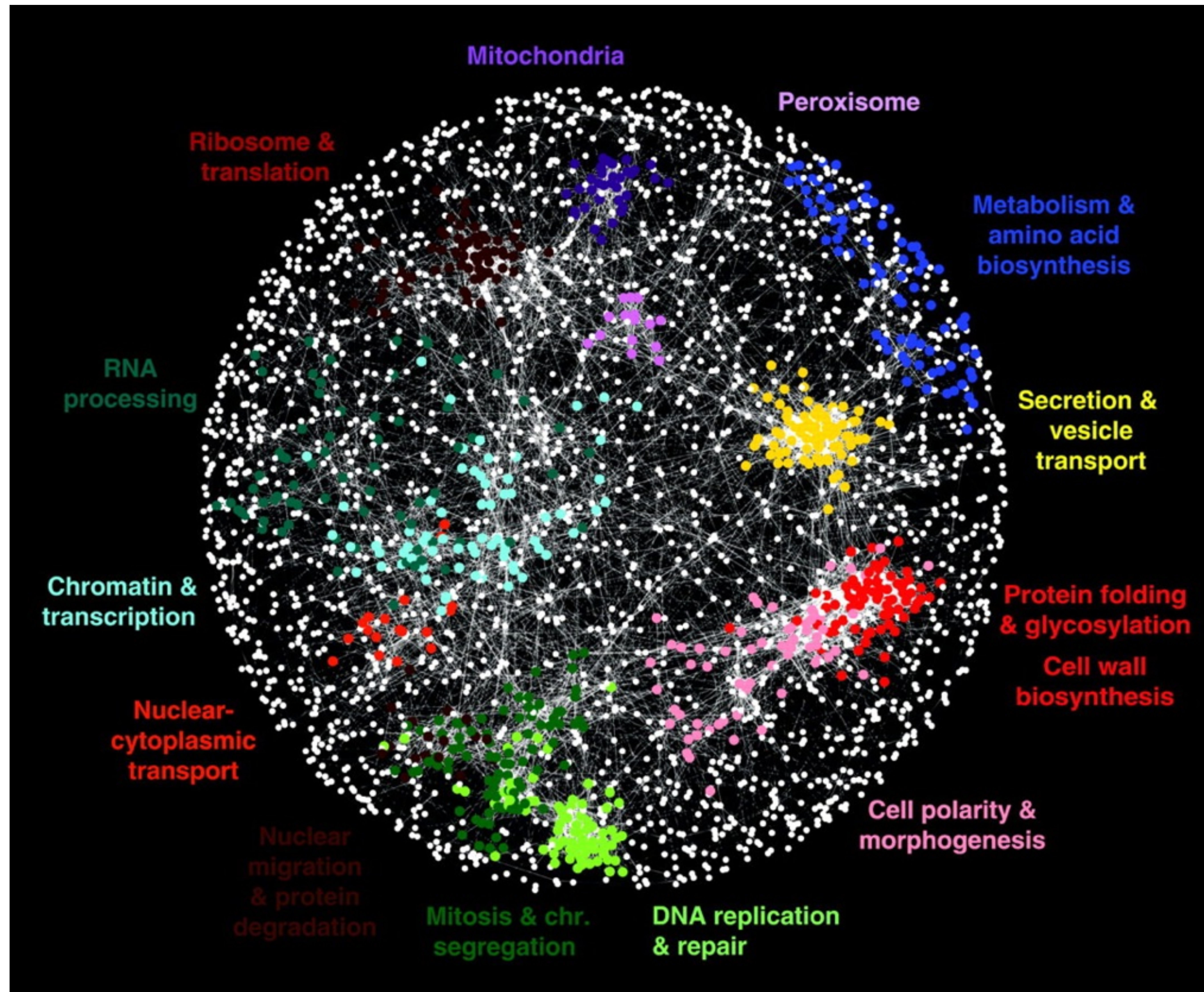
- Shared memory improves performance by **25x**
- **Near linear scaling** is observed

Shared memory MCL has superior performance



- Shared memory MCL improves performance by **21x** and has **linear** scalable performance

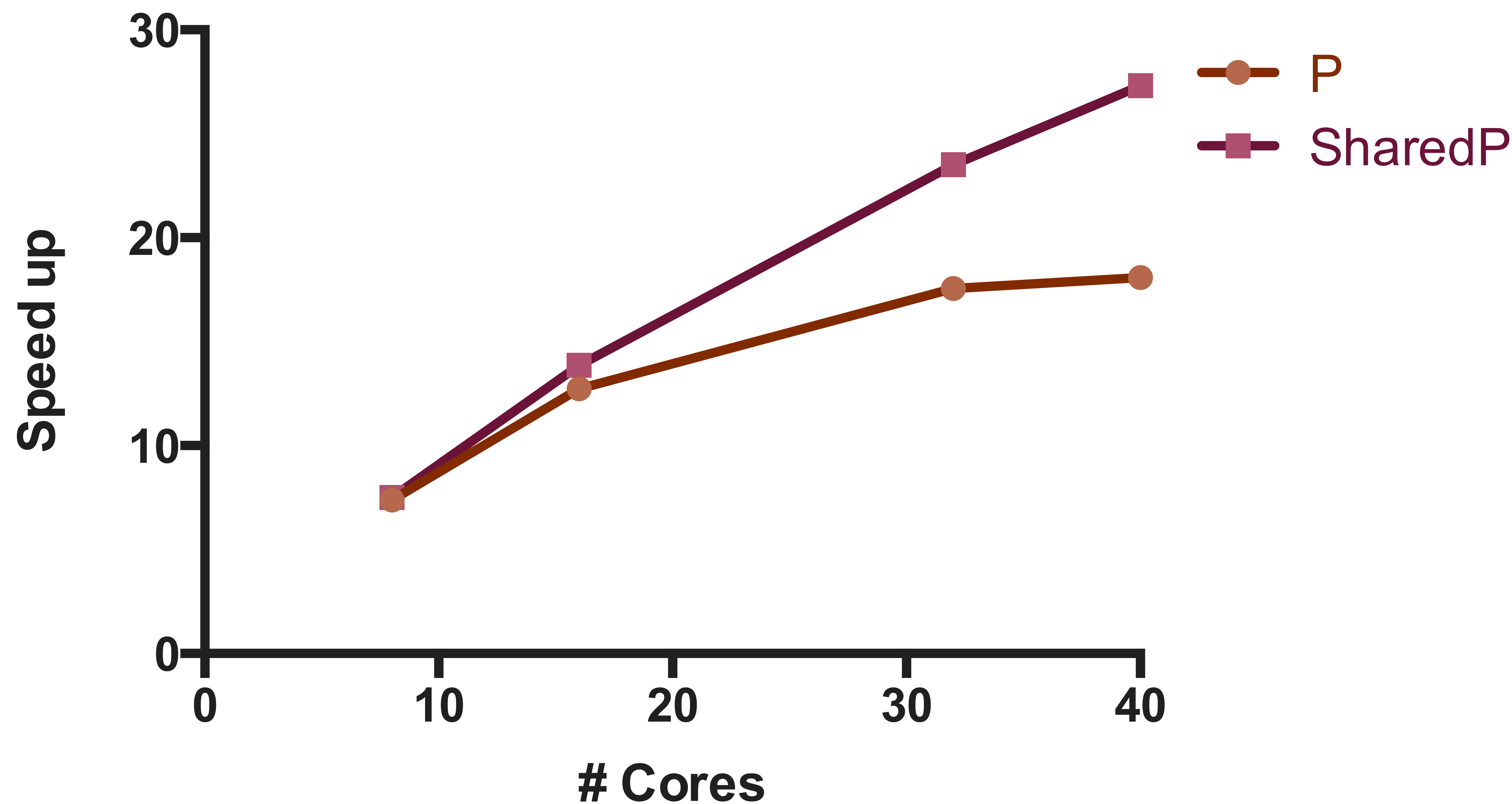
The genetic landscape of a cell



- Dataset created from an interaction map of 5.4 million gene-gene pairs from the budding yeast, Saccharomyces cerevisiae
- 3886 nodes and 15,100,996 edges
- ~26% sparsity

Costanzo et al, *Science, 2010*

MCL successfully clusters 3,886 proteins



Average cluster size: 6.45 proteins
Clusters with >5 members: 229
Singlet Clusters: 253
Total # of clusters: 714

- MCL shared achieved **27x speed increase** and **linear scaling**

Outlook

- Parallelizing in Julia gave superior performance of MCL
- Even better performance was observed on a real, sparse dataset
- Develop a version for GPU computation with Julia
- Implement a sparse version in order to reduce memory usage (such as using CSC format in Julia)

Thank you

Questions?