CS601: Software Development for Scientific Computing

Autumn 2021

Week13:

Hierarchical Methods (FMM) and Sparse Matrices

Course Progress..

- Last week
  - Tree-based codes (hierarchical methods)
    - Barnes-Hut
    - Fast Multipole Method (FMM)
- This week
  - FMM
  - Sparse matrices and
  - PA4 discussion

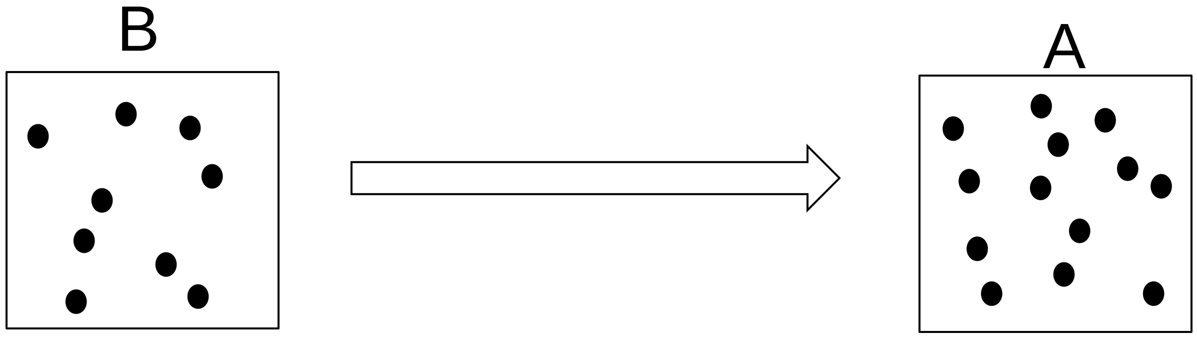
FMM Algorithm

1. Build the quadtree containing all the points.
2. Traverse the quadtree from bottom to top, computing Outer(n) for each square n in the tree.
3. Traverse the quadtree from top to bottom, computing Inner(n) for each square in the tree.
4. For each leaf, add the contributions of nearest neighbors and particles in the leaf to Inner(n)

what is Outer(n) and Inner(n) ?

Well Separated Regions

- Compute the influence of all particles in source region (B) on every particle in target region (A)
  (assumption: A and B are well-separated)



- At each point p_i in A, compute potential:

$$\Phi(x_i, y_i) = \sum_{p_j \in B} m_i \log |p_i - p_j|$$

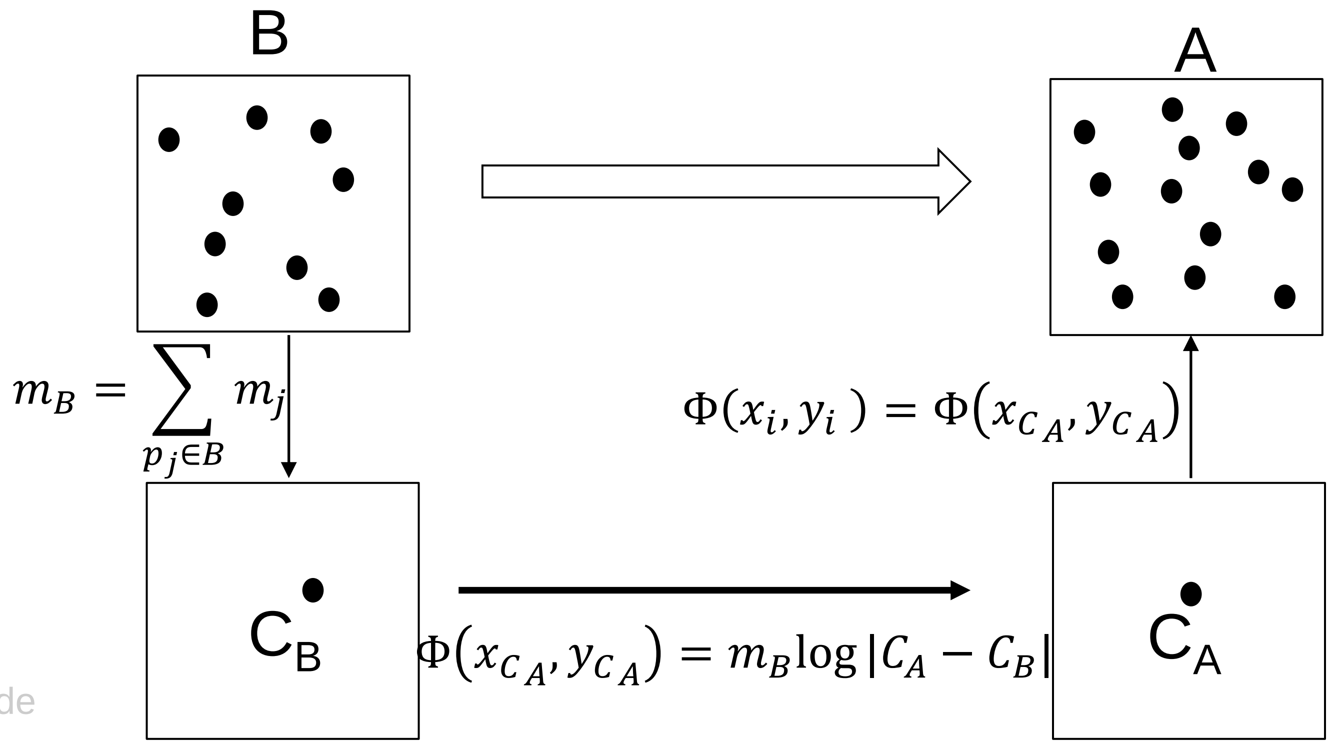
$$i = 1\ to\ N_A, \qquad j = 1\ to\ N_B$$

- Cost: $O(N_A N_B)$

Well Separated Regions

- Compute the influence of all particles in source region (B) on every particle in target region (A)

$$\Phi\left(x_{p_i}, y_{p_i}\right) = \sum_{p_j \in B} m_i \log|p_i - p_j| \text{ , } p_i \in A$$


B
A
$m_B = \sum_{p_j \in B} m_j$
$\Phi(x_i, y_i) = \Phi\left(x_{C_A}, y_{C_A}\right)$
C_B
$\Phi\left(x_{C_A}, y_{C_A}\right) = m_B \log|C_A - C_B|$
C_A

Applying the 3-step Approximation

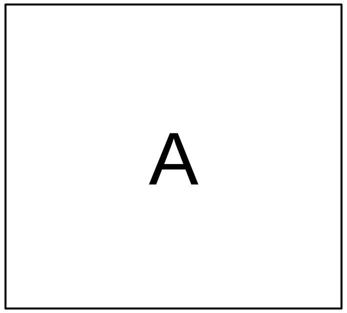
- In N-body simulation every point serves as source as well as target.

  How to identify source and target (boxes A and B in previous slide) i.e. well-separated regions?

  Hierarchical decomposition

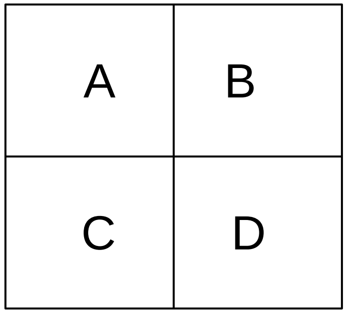
Hierarchical Decomposition

- *Level-0 decomposition*



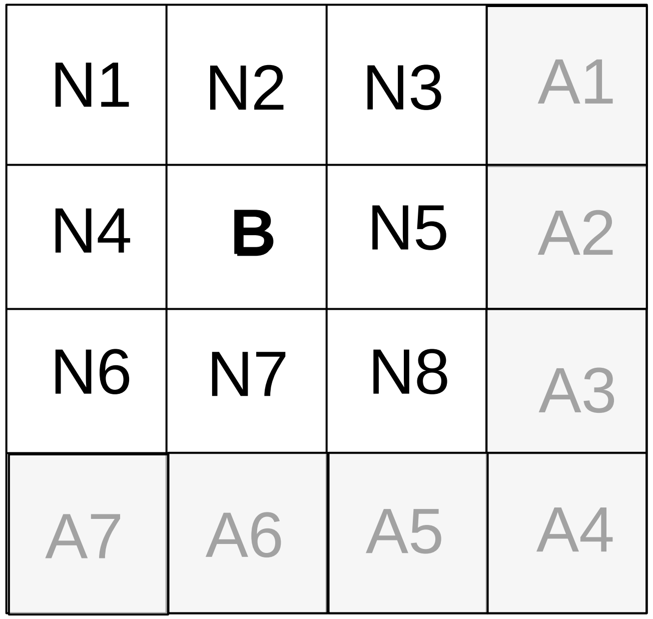
- *Level-1 decomposition*

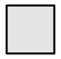
No well-separated boxes



Hierarchical Decomposition

- *Level-2 decomposition*



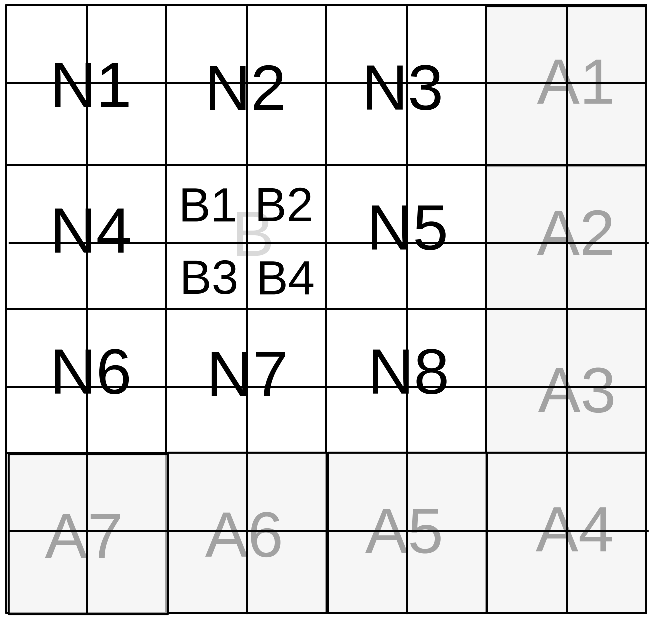


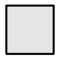
Can approximate the influence of points in `B` on points in `Ai` s

What do we do about **B**’s influence on Ni s?

Hierarchical Decomposition

- *Level-3 decomposition*

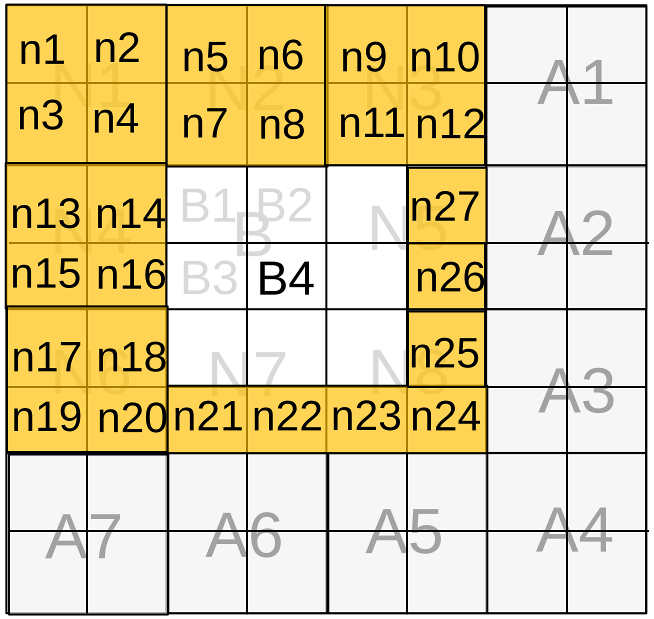




Influence of points in `Bi` s on those in `Ai` s already computed at the previous level (level-2)

Hierarchical Decomposition

- *Level-3 decomposition*



⬜ Influence of points in `Bi` s on those in `Ai` s already computed at the previous level (level-2)

🟨 Well-separated from B4

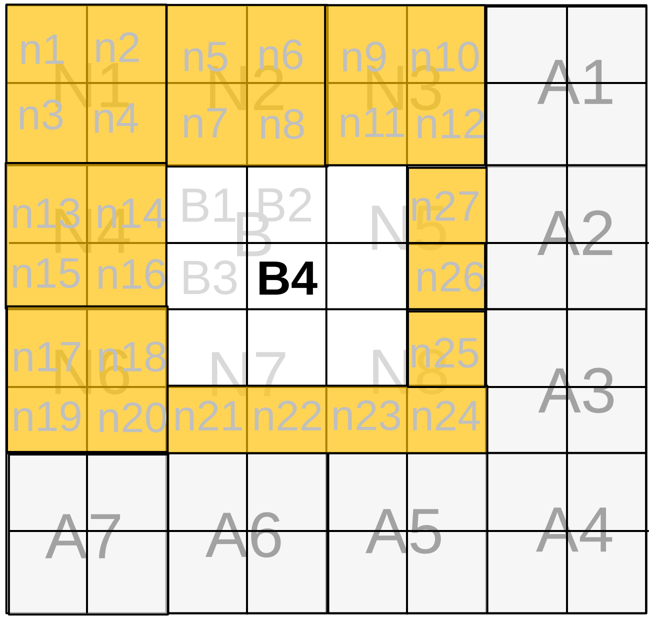
Influence of `B4’s` points on `nx’s` points can be approximated

`nx’s` constitute the <u>interaction list</u> for `B4`.

What is the max size of interaction list? i.e. max number of `nx` *s that we can have for any* `Bi`*?*

Hierarchical Decomposition

- *Level-3 decomposition*



⬜ Influence of points in `Bi` s on those in `Ai` s already computed at the previous level (level-2)

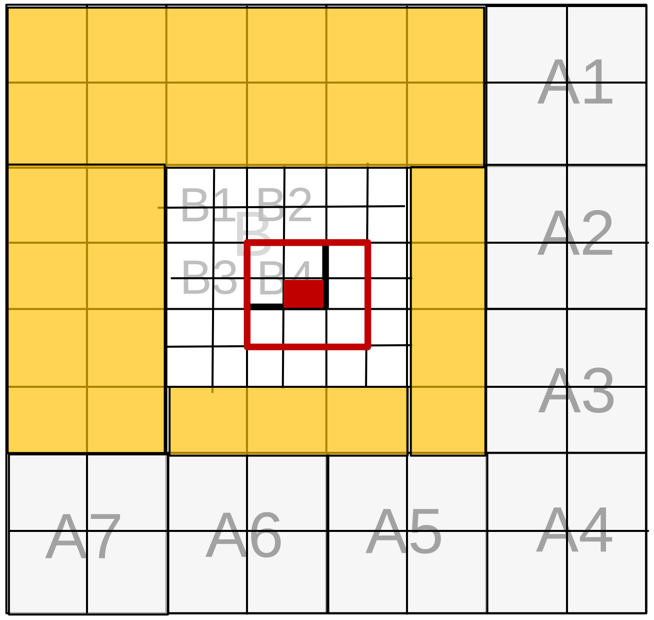
🟨 Well-separated from B4

Influence of `B4's` points on `nx's` points can be approximated

What do we do about **B4**'s influence on its neighbors (white/unshaded boxes)?

Hierarchical Decomposition

- *Level-4 decomposition*



Any unshaded box outside □ can be the *target* for computing the influence of points in 🟥 (*source*)

1. Computing Potential for Well-Separated Regions

```
1. for level L=2 to last_level
2.     for each Box B at level L
3.             iList = GetInteractionList(B)
4.             for each well-separated box A in iList
                       //Compute potential
5.                     potential $= m_B \log |C_A - C_B|$
                       //Accumulate potential
6.                     $\Phi(x_{C_A}, y_{C_A})$ += potential
```

Cost?

1. Computing Potential for Well-Separated Regions

```
1. for level L=2 to last_level
2.     for each Box B at level L
3.             iList = GetInteractionList(B)
4.             for each well-separated box A in iList
                       //Compute potential
5.                      potential $= m_B \log|C_A - C_B|$
                       //Accumulate potential
6.                     $\Phi(x_{C_A}, y_{C_A})$ +=potential
```

Prereqs: we need $\mathrm{m_B}, \mathrm{C_A}, \mathrm{C_B}$ details. (step 0)

2. Assigning Potential to Points

```
1. for each Box A at level L=0 to last_level
2.     $\Phi_{p_i} = \Phi_{p_i} + \Phi_{C_A}$  (where $p_i \in A$ and $C_A$ is A's CM)
```

Cost?

3. Assigning Potential to Points (last level)

```
1. for each Box B at last_level
2.     $\Phi_{p_i} = \Phi_{p_i} + \sum_{p_j \in Neighbors(B)} m_B \log|p_i - p_j|$ (where $p_i \in B$)
```

Cost?

0. Computing Prereqs

```
1. for each Box B at level L=0 to last_level
2.     $m_B = \sum_{p_j \in B} m_j$
3.     //similarly compute $C_B$
```

Cost?

Total Cost (steps 0 + 1 + 2 + 3)

```
O(N log N) + O (N) + O(N log N) + O(N)
```

Can we do better?

0'. Computing Prereqs

- Traverse the tree bottom up instead of top-down

```
for each Box B starting from last_level to L=0
     if B is a leaf box
          $m_B = \sum_{p_j \in B} m_j$
     else
          $m_B = m_{B_1} + m_{B_2} + m_{B_3} + m_{B_4}$
          //$B_1$-$B_4$ are children of B
```

Cost?

2'. Assigning Potential to Points

```
1. for each Box A at level L=0 to last_level
2.     if A is a leaf box
               $\Phi_{p_i} = \Phi_{p_i} + \Phi_{C_A}$  (where $p_i \in A$ and $C_A$ is A's CM)
       else
               $\Phi_{A_1} = \Phi_{A_1} + \Phi_A$
               $\Phi_{A_2} = \Phi_{A_2} + \Phi_A$
               $\Phi_{A_3} = \Phi_{A_3} + \Phi_A$
               $\Phi_{A_4} = \Phi_{A_4} + \Phi_A$
               //$A_1$-$A_4$ are children of A
```

Cost?

Total Cost (steps 0’ + 1 + 2’ + 3)

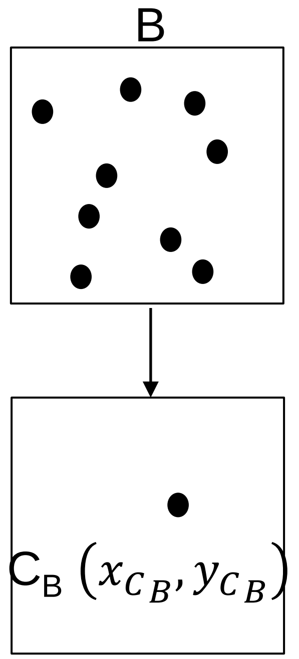
`O(N) + O (N) + O(N) + O(N)`

Problem: low accuracy if source (A) and target (B) are not far away from each other

Solution: more accurate representations for m_B and $\Phi\left(x_{C_A}, y_{C_A}\right)$

Multipole expansion

- Like a Taylor series expansion that is accurate when $x^2 + y^2$ is large (x, y are cartesian coordinates of the point)
- For a quadtree box B centered at $\left(x_{C_B}, y_{C_B}\right)$, we compute and store the terms: $\{\boldsymbol{m_B}, \boldsymbol{\alpha_1}, \boldsymbol{\alpha_2}, \ldots, \boldsymbol{\alpha_p}, \boldsymbol{z_{C_B}}\}$

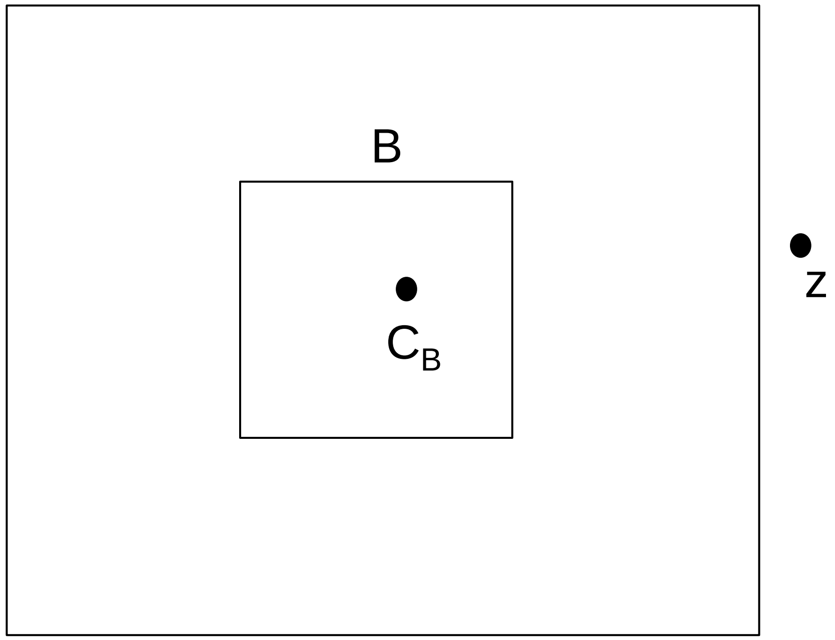


$$\alpha_j = \sum_{i=1}^{N_B} m_i \left(\frac{z_i^j}{j}\right)$$

$z_i$means$|z_i| = |(x_i, y_i)|$

Multipole expansion

- We approximate the potential at point z due to B by:



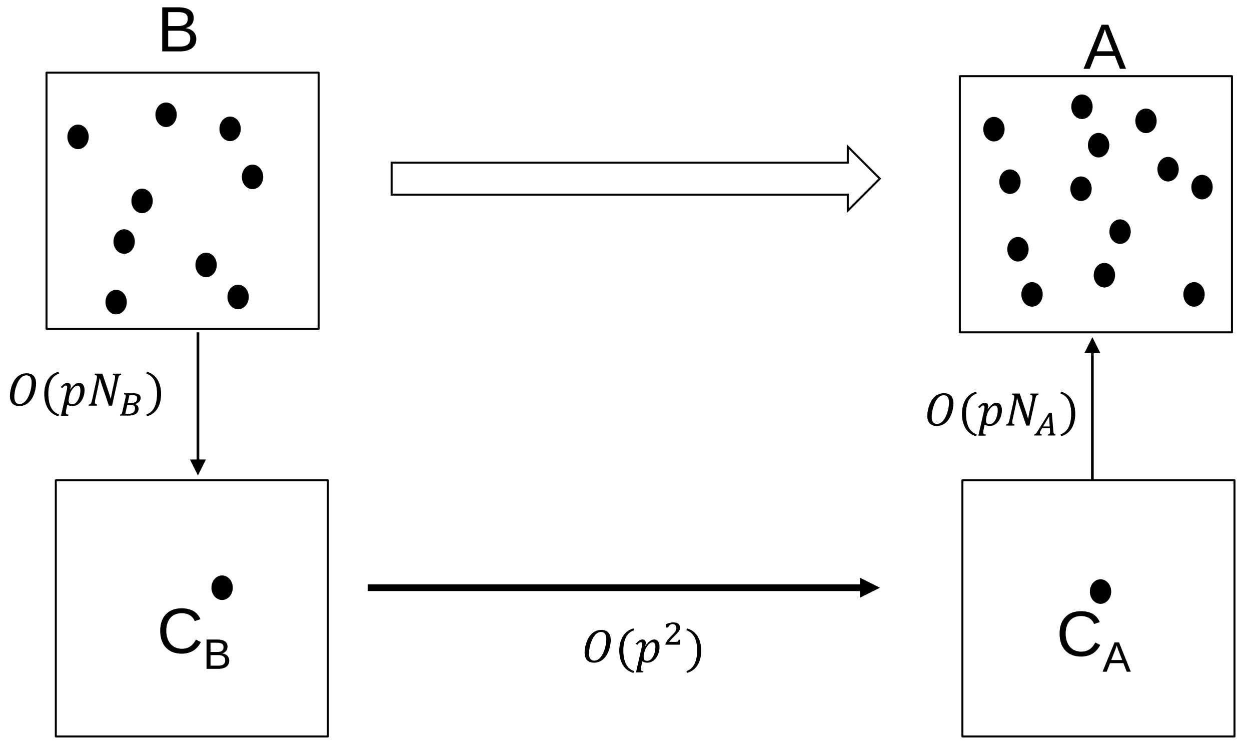
$$\Phi(x_z, y_z) = m_B \log(z - C_B) + \frac{\alpha_1}{z - C_B} + \frac{\alpha_2}{(z - C_B)^2} + \cdots + \frac{\alpha_p}{(z - C_B)^p}$$

- Because $\{\boldsymbol{m_B}, \boldsymbol{\alpha_1}, \boldsymbol{\alpha_2}, \ldots, \boldsymbol{\alpha_p}, \boldsymbol{z_{C_B}}\}$ is used to compute potential outside B, it is called <u>outer expansion</u>

Multipole expansion

- Similarly, we have the <u>inner expansion</u> $\{m_B, \beta_1, \beta_2, \ldots, \beta_p, z_{C_B}\}$ for computing the potential inside the Box due to all other points outside the box
- Computing outer expansions starts from leaf nodes and proceeds upwards in the tree.
- Computing inner expansions starts from root node and proceeds downwards in the tree.

3-Step Approximation (accurate)



FMM Algorithm

1. Build the quadtree containing all the points.
2. Traverse the quadtree from bottom to top, computing Outer(n) for each square n in the tree.
3. Traverse the quadtree from top to bottom, computing Inner(n) for each square in the tree.
4. For each leaf, add the contributions of nearest neighbors and particles in the leaf to Inner(n)

Multipole expansion

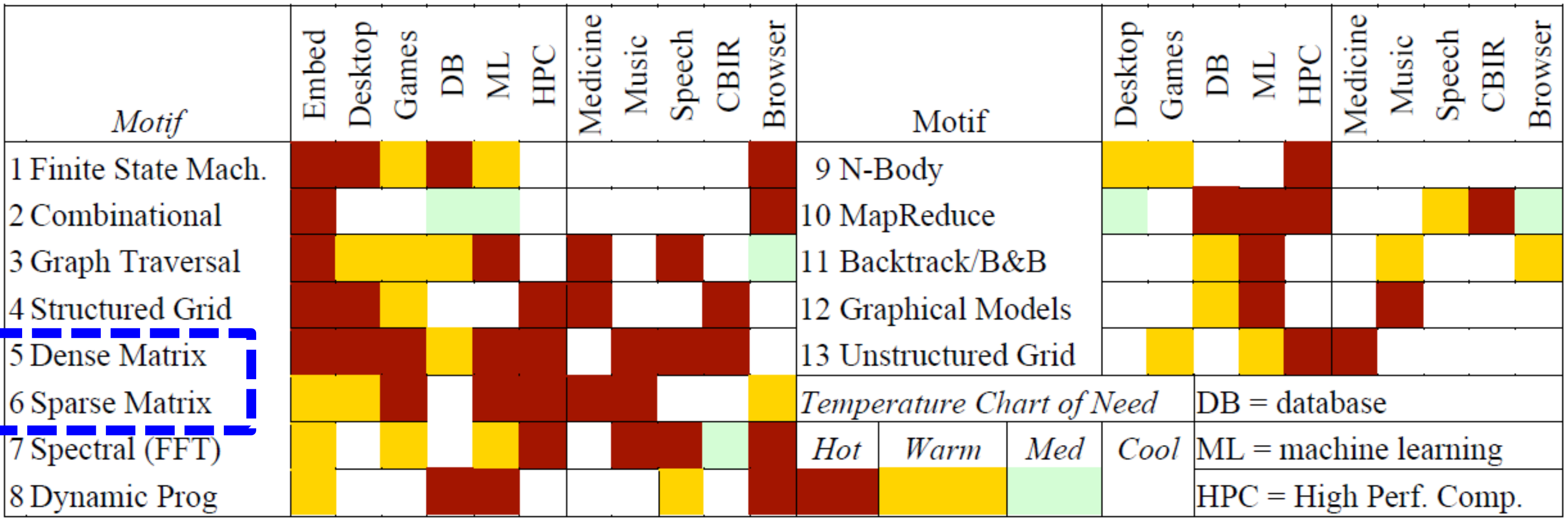
- How to obtain the expression for alpha, beta ?
- What is the value of p?
- How to compute alpha and beta?

- Further reading:

https://people.eecs.berkeley.edu/~demmel/cs267/lecture27/lecture27.html

Matrix Algebra and Efficient Computation

- **Pic source: the Parallel Computing Laboratory at U.C. Berkeley: A Research Agenda Based on the Berkeley View (2008)**



Motif	Embed	Desktop	Games	DB	ML	HPC	Medicine	Music	Speech	CBIR	Browser
1 Finite State Mach.	🟥	🟥	🟨	🟥	🟨						🟥
2 Combinational	🟥			🟩	🟩						🟥
3 Graph Traversal	🟥	🟨	🟨	🟨	🟥		🟥		🟥		🟩
4 Structured Grid	🟥	🟥	🟨			🟥	🟥			🟥	
5 Dense Matrix	🟥	🟥	🟥	🟨	🟥	🟥		🟥	🟥	🟥	
6 Sparse Matrix	🟨	🟨	🟥		🟥	🟥	🟥	🟥			🟨
7 Spectral (FFT)	🟨		🟨		🟨	🟥		🟥	🟥	🟩	🟥
8 Dynamic Prog	🟨			🟥	🟥				🟨		🟥

Motif	Desktop	Games	DB	ML	HPC	Medicine	Music	Speech	CBIR	Browser
9 N-Body	🟨	🟨			🟥					
10 MapReduce	🟩		🟥	🟥	🟥			🟨	🟥	🟩
11 Backtrack/B&B			🟨	🟥			🟨			🟨
12 Graphical Models			🟨	🟥			🟥			
13 Unstructured Grid		🟨		🟨	🟥	🟥				

Temperature Chart of Need				
Hot	*Warm*	*Med*	*Cool*	
🟥	🟨	🟩		

DB = database
ML = machine learning
HPC = High Perf. Comp.

Figure 4. Temperature Chart of the 13 Motifs. It shows their importance to each of the original six application areas and then how important each one is to the five compelling applications of Section 3.1. More details on the motifs can be found in (Asanovic, Bodik et al. 2006).

Seen earlier

Next..

Matrix Multiplication

- Why study?
  - An important “kernel” in many linear algebra algorithms
  - Most studied kernel in high performance computing
  - Simple. Optimization ideas can be applied to other kernels
- Matrix representation
  - Matrix is a 2D array of elements. Computer memory is inherently linear
  - C++ and Fortran allow for definition of 2D arrays. 2D arrays stored row-wise in C++. Stored column-wise in Fortran. E.g.

```
// stores 10 arrays of 20 doubles each in C++
double** mat = new double[10][20];
```

Storage Layout - Example

- Matrix (**2D**):$\mathrm{A} = \begin{bmatrix} A(0,0) & A(0,1) & A(0,2) \\ A(1,0) & A(1,1) & A(1,2) \\ A(2,0) & A(2,1) & A(2,2) \end{bmatrix}$

$A(i,j) = A(row, column)$ refers to the matrix element in the ith row and the jth column

- Row-wise (/Row-major) storage in memory:

$A(0,0)$	$A(0,1)$	$A(0,2)$	$A(1,0)$	$A(1,1)$	$A(1,2)$	$A(2,0)$	$A(2,1)$	$A(2,2)$

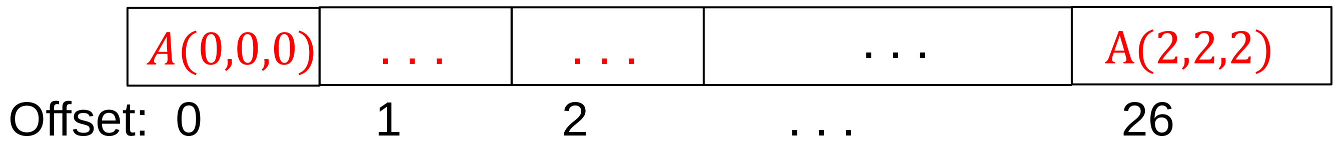
- Column-wise (/Column-major) storage in memory:

$A(0,0)$	$A(1,0)$	$A(2,0)$	$A(0,1)$	$A(1,1)$	$A(2,1)$	$A(0,2)$	$A(1,2)$	$A(2,2)$

- **Generalizing data storage order for ND:** last index changes fastest in row-major. Last index changes slowest in col-major.

Storage Layout - Exercise

- For a 3D array (tensor) assume $A(i, j, k) = A(row, column, depth)$

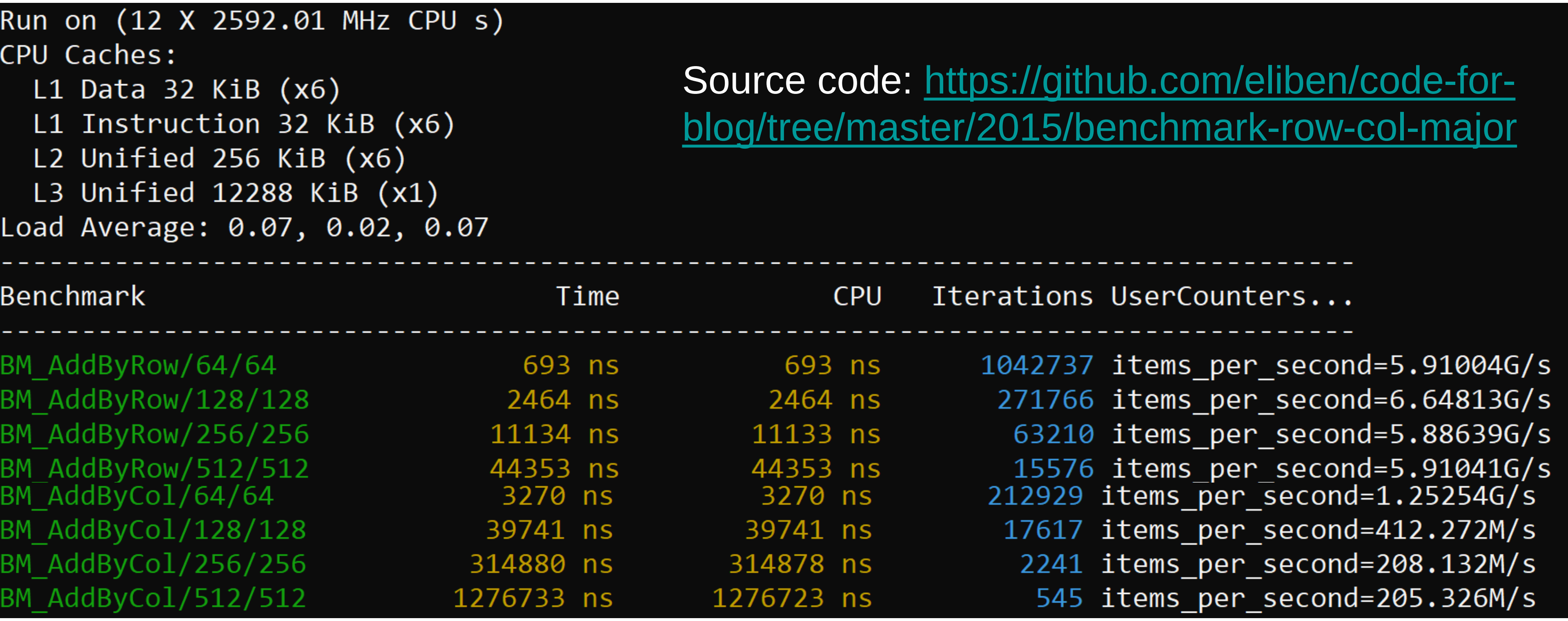


- What is the offset of $A(1, 2, 1)$? as per row-major storage?
- What is the offset of $A(1, 2, 1)$? as per col-major storage?

Storage Layout

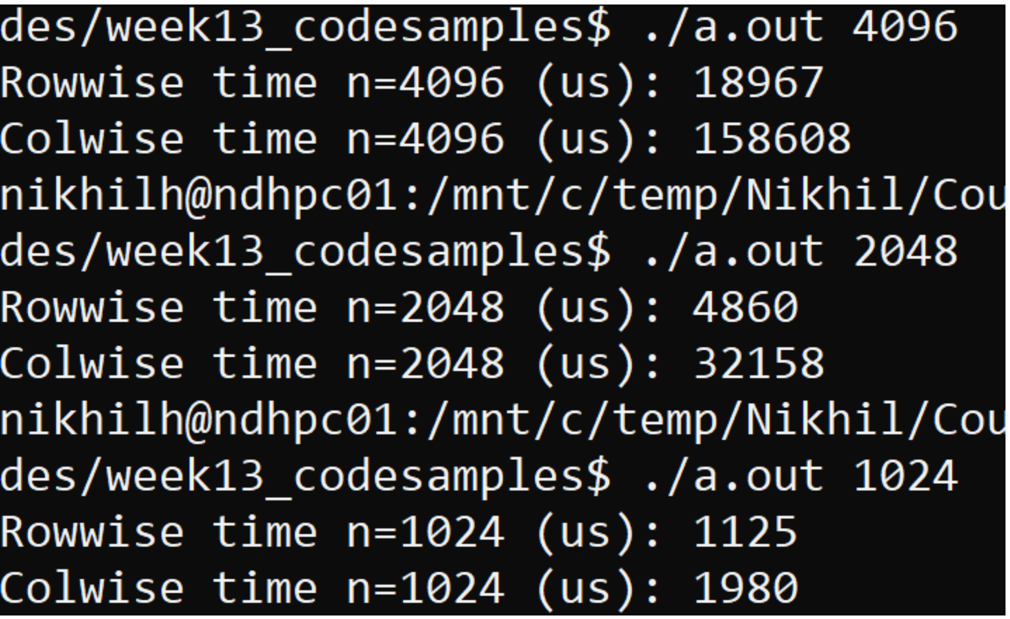
- Layout format itself doesn’t influence efficiency (i.e. no general answer to “is column-wise layout better than row-wise?”)
- However, knowing the layout format is critical for good performance
  - *Always traverse the data in the order in which it is laid out*

How good performance?



```
Run on (12 X 2592.01 MHz CPU s)
CPU Caches:
  L1 Data 32 KiB (x6)
  L1 Instruction 32 KiB (x6)
  L2 Unified 256 KiB (x6)
  L3 Unified 12288 KiB (x1)
Load Average: 0.07, 0.02, 0.07
-------------------------------------------------------------------------------------
Benchmark                              Time             CPU   Iterations UserCounters...
-------------------------------------------------------------------------------------
BM_AddByRow/64/64                    693 ns          693 ns      1042737 items_per_second=5.91004G/s
BM_AddByRow/128/128                 2464 ns         2464 ns       271766 items_per_second=6.64813G/s
BM_AddByRow/256/256                11134 ns        11133 ns        63210 items_per_second=5.88639G/s
BM_AddByRow/512/512                44353 ns        44353 ns        15576 items_per_second=5.91041G/s
BM_AddByCol/64/64                   3270 ns         3270 ns       212929 items_per_second=1.25254G/s
BM_AddByCol/128/128                39741 ns        39741 ns        17617 items_per_second=412.272M/s
BM_AddByCol/256/256               314880 ns       314878 ns         2241 items_per_second=208.132M/s
BM_AddByCol/512/512              1276733 ns      1276723 ns          545 items_per_second=205.326M/s
```

Source code: https://github.com/eliben/code-for-blog/tree/master/2015/benchmark-row-col-major



```
des/week13_codesamples$ ./a.out 4096
Rowwise time n=4096 (us): 18967
Colwise time n=4096 (us): 158608
nikhilh@ndhpc01:/mnt/c/temp/Nikhil/Cou
des/week13_codesamples$ ./a.out 2048
Rowwise time n=2048 (us): 4860
Colwise time n=2048 (us): 32158
nikhilh@ndhpc01:/mnt/c/temp/Nikhil/Cou
des/week13_codesamples$ ./a.out 1024
Rowwise time n=1024 (us): 1125
Colwise time n=1024 (us): 1980
```

⇧

Matrix-Matrix Addition benchmarking

(Source code and further reading)

⇦ Matvec execution time

(we used the source code as a basic example to demonstrate row_major vs. col_major storage.)

Matrix Multiplication

- Three fundamental ways to think of the algorithm
  - Dot product

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\times\begin{bmatrix}5 & 6\\7 & 8\end{bmatrix} = \begin{bmatrix}1.5+2.7 & 1.6+2.8\\3.5+4.7 & 3.6+4.8\end{bmatrix}$$

  - Linear combination of left matrix

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\times\begin{bmatrix}5 & 6\\7 & 8\end{bmatrix} = \begin{bmatrix}5\begin{bmatrix}1\\3\end{bmatrix}+7\begin{bmatrix}2\\4\end{bmatrix} & 6\begin{bmatrix}1\\3\end{bmatrix}+8\begin{bmatrix}2\\4\end{bmatrix}\end{bmatrix}$$

  - Sum of outer products

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\times\begin{bmatrix}5 & 6\\7 & 8\end{bmatrix} = \begin{bmatrix}\begin{bmatrix}1\\3\end{bmatrix}\begin{bmatrix}5 & 6\end{bmatrix} + \begin{bmatrix}2\\4\end{bmatrix}\begin{bmatrix}7 & 8\end{bmatrix}\end{bmatrix}$$