Image Down Scaling Using MPI

by Shubham Prasad, Pednekar

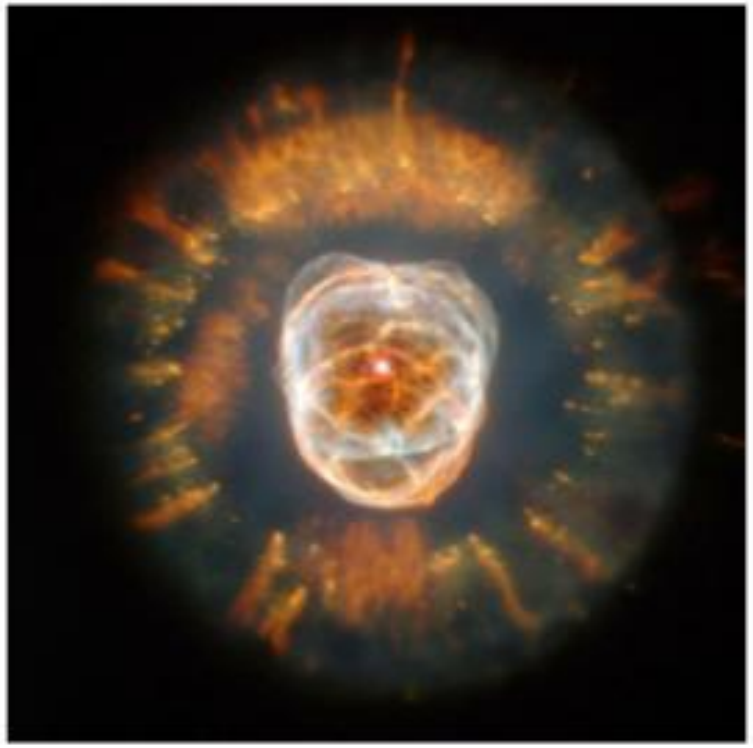
For *CSE708 : Programming Massively Parallel Systems*

Instructed by **Dr. Russ Miller**.

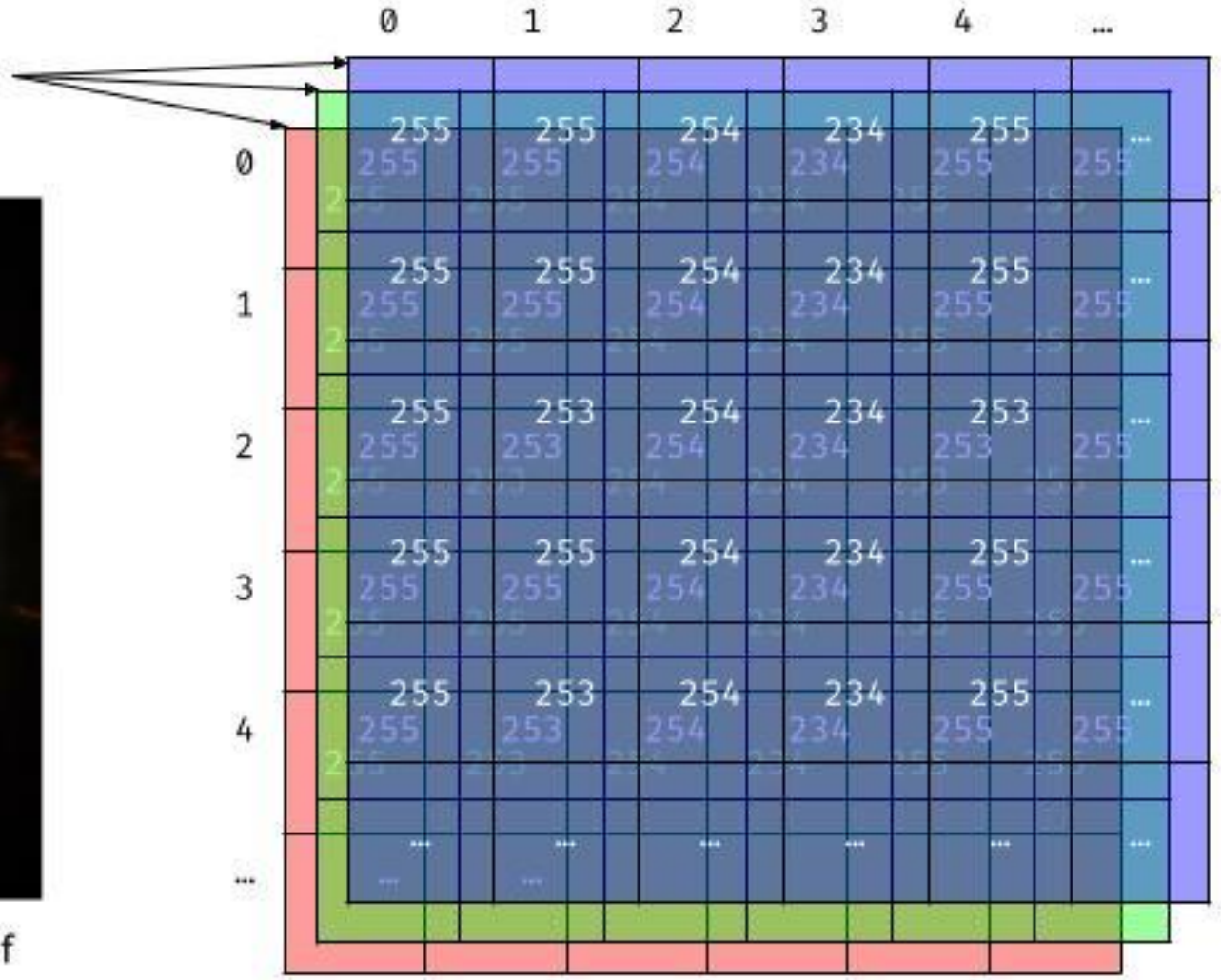
University at Buffalo The State University of New York

University at Buffalo The State University of New York

What is **Image Down-scaling?**



An RGB image is a 2D arrangement of pixels.



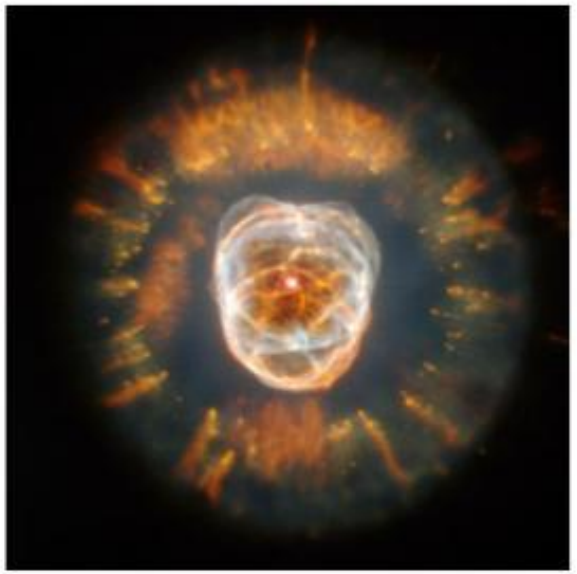
Each pixel is represented by R, G, B values and its indices

≈ 1696**e5** × 1696**e5** × 3 integers

Original Image

DOWNSCALING

1696 × 1696 × 3 integers



Down Scaled Image

Fine-grained algorithm using Cluster of size M*N

Fine-grained Algorithm for Image Down-scaling

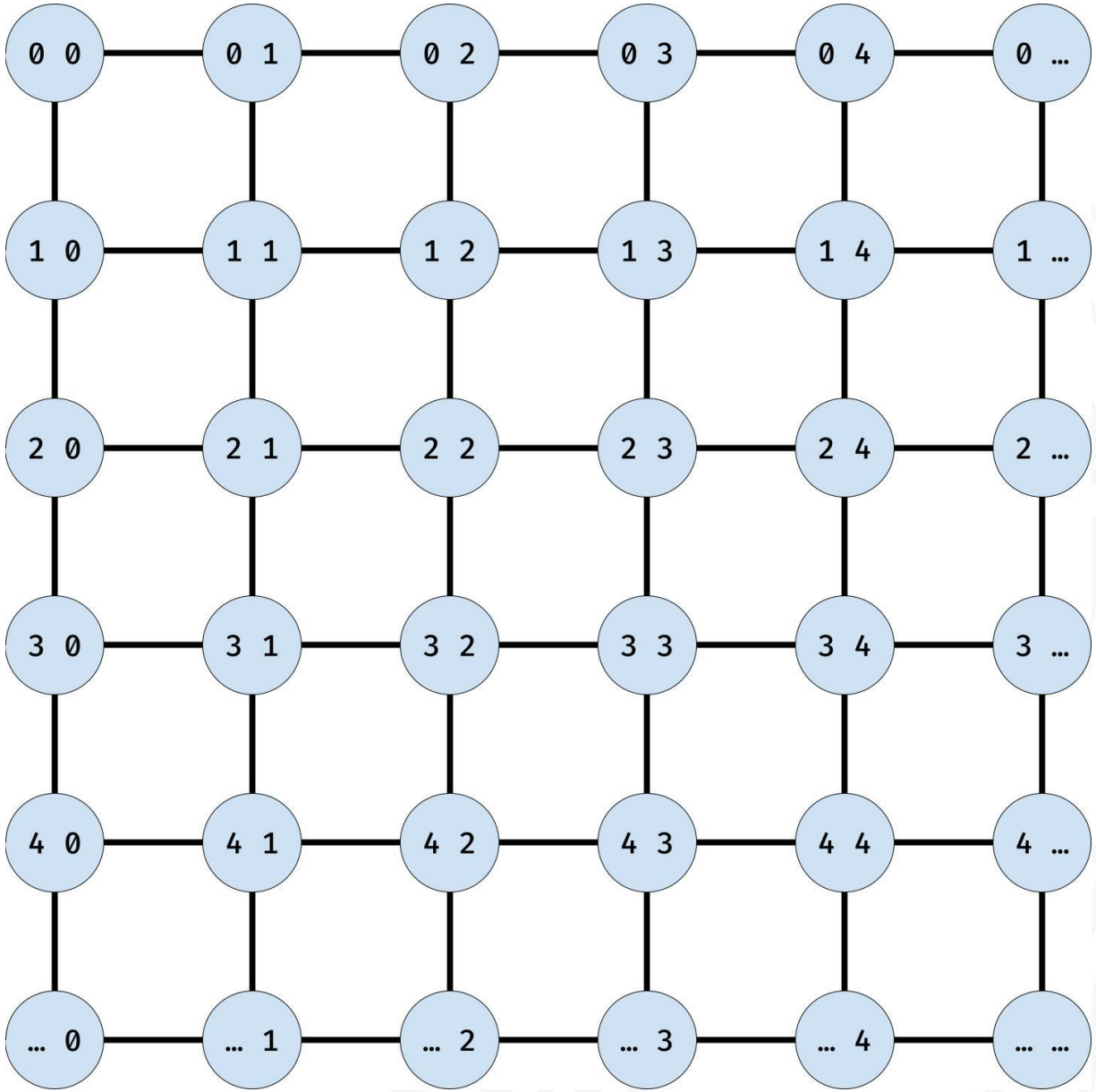
- We approximate X to 2^k. We take 3 mesh like clusters of size M*N with row-major-indexing.
- We distribute the M*N values of R, G and B over the 3 clusters of size M*N respectively. We do this by giving the value of pixel `[x y]` to `P[x y]`.
- Then we keep performing one of the following operations alternatively over all 3 sub-clusters until we reach our desired resolution:
  - Horizontal Merging : All `P[i j]⟵(P[2i j] + P[2i+1 j])/2`
  - Vertical Merging : All `P[i j]⟵(P[i 2j] + P[i 2j+1])/2`
- After the i^{th} iteration, the values on

  `P[0 0]` → `P[0 M/(2^i)]` → `P[N/(2^i) M/(2^i)]` → `P[N/(2^i) 0]`

  for each mesh like clusters represents the down–scaled image!

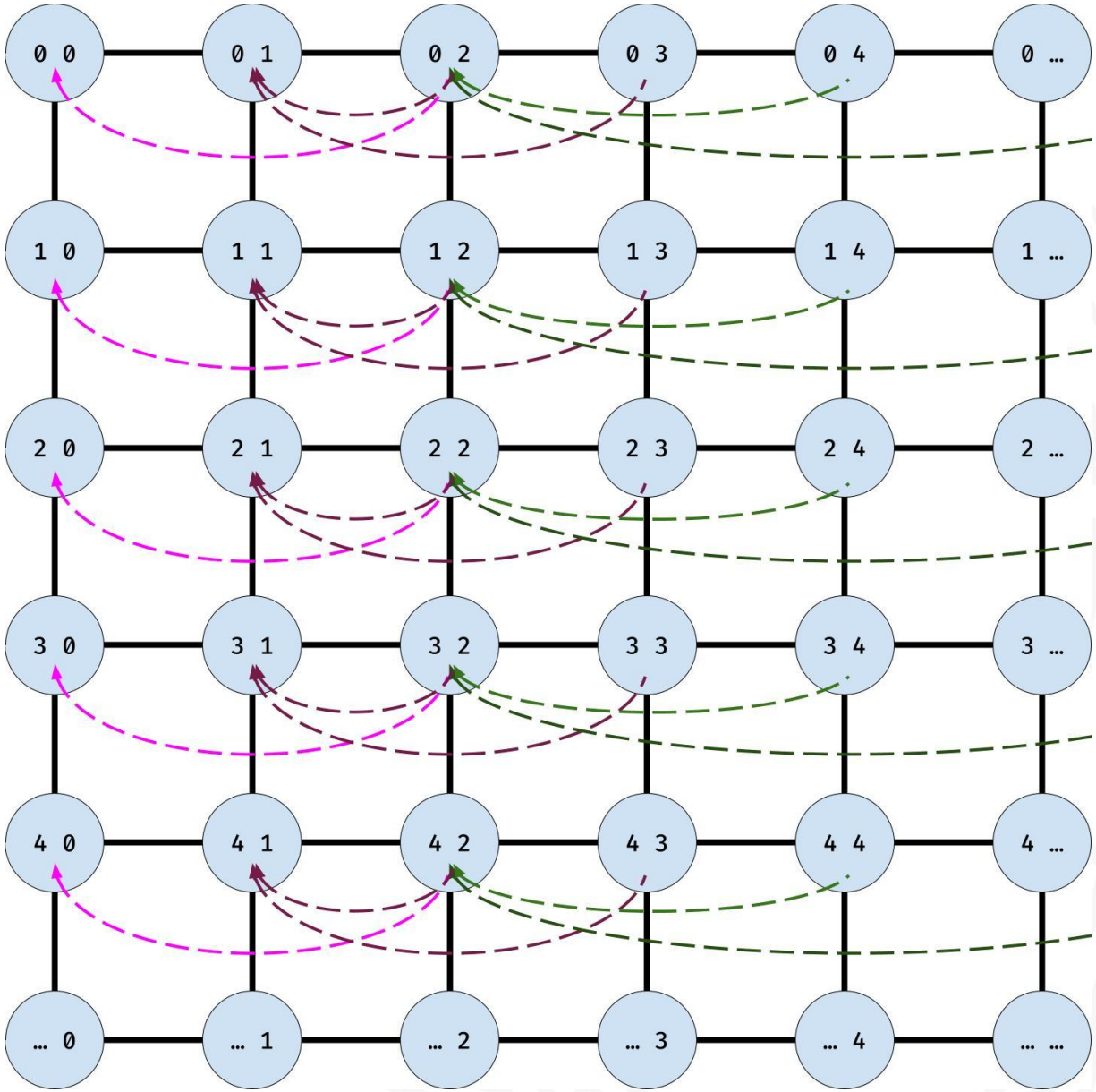
Procedure for Horizontal Merging

1. Every Processor, `P[i  j]`, gets values of `P[2i  j]` and `P[2i+1  j]`
2. `P[i  j]` calculates the average of the 2 values, `(P[2i  j] + P[2i+1  j])/2`
3. `P[i  j]` stores this calculated average as its new value.



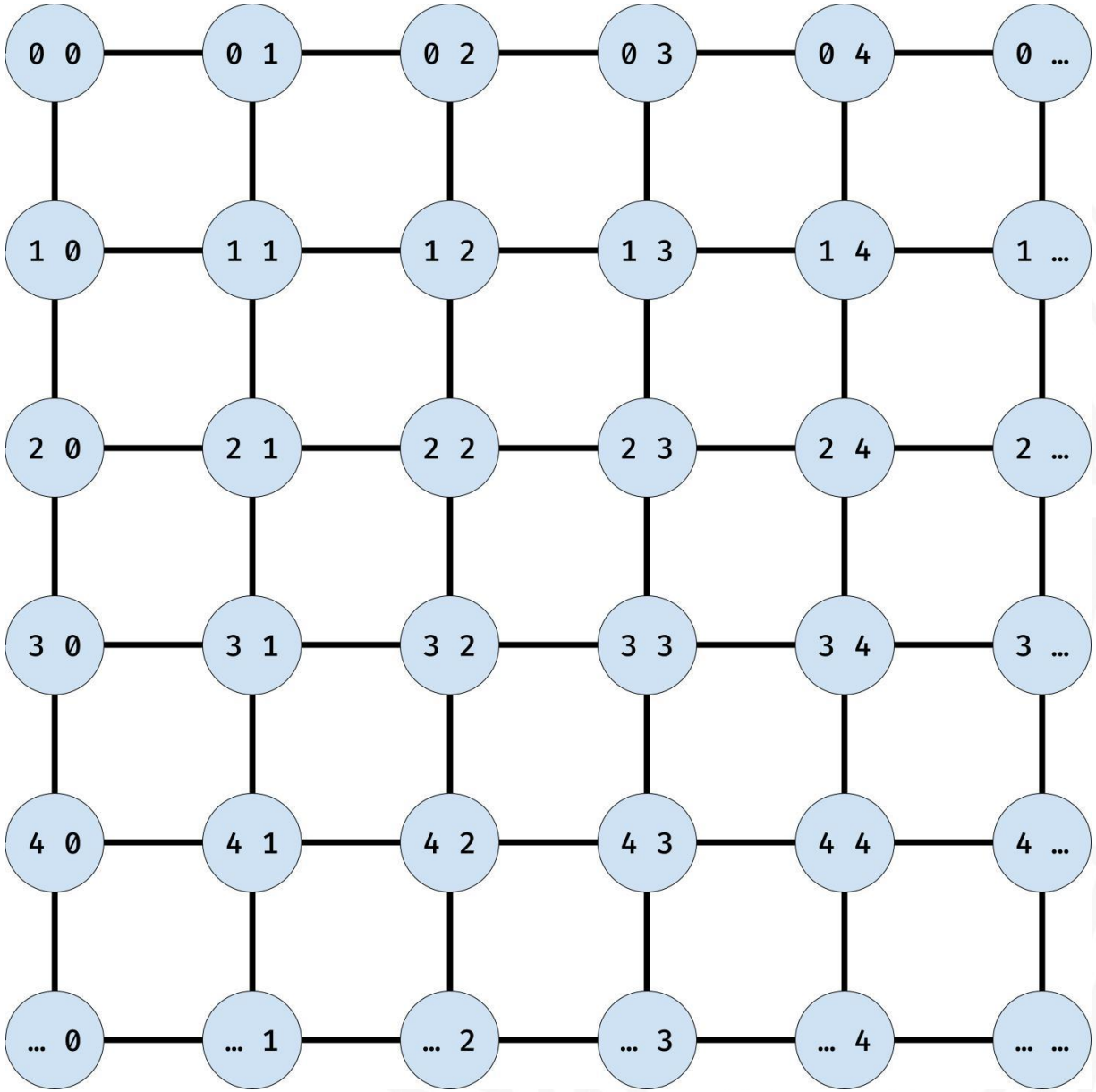
Procedure for Horizontal Merging

1. Every Processor, `P[i  j]`, gets values of `P[2i  j]` and `P[2i+1  j]`
2. `P[i  j]` calculates the average of the 2 values, `(P[2i  j] + P[2i+1  j])/2`
3. `P[i  j]` stores this calculated average as its new value.



Procedure for Vertical Merging

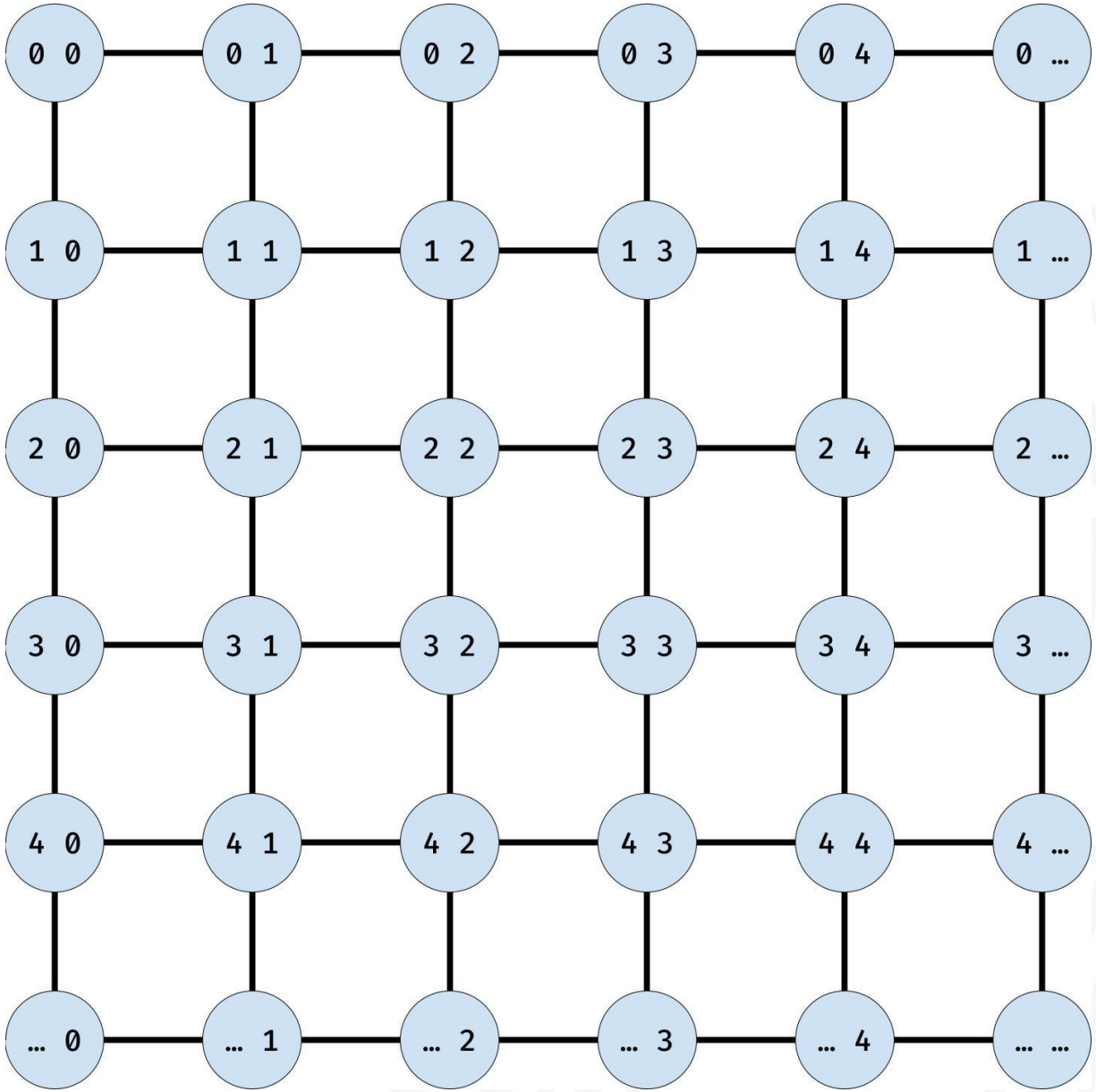
1. Every Processor, `P[i  j]`, gets values of `P[2i  j]` and `P[2i+1  j]`
2. `P[i  j]` calculates the average of the 2 values, `(P[2i  j] + P[2i+1  j])/2`
3. `P[i  j]` stores this calculated average as its new value.



Procedure for Down-scaling

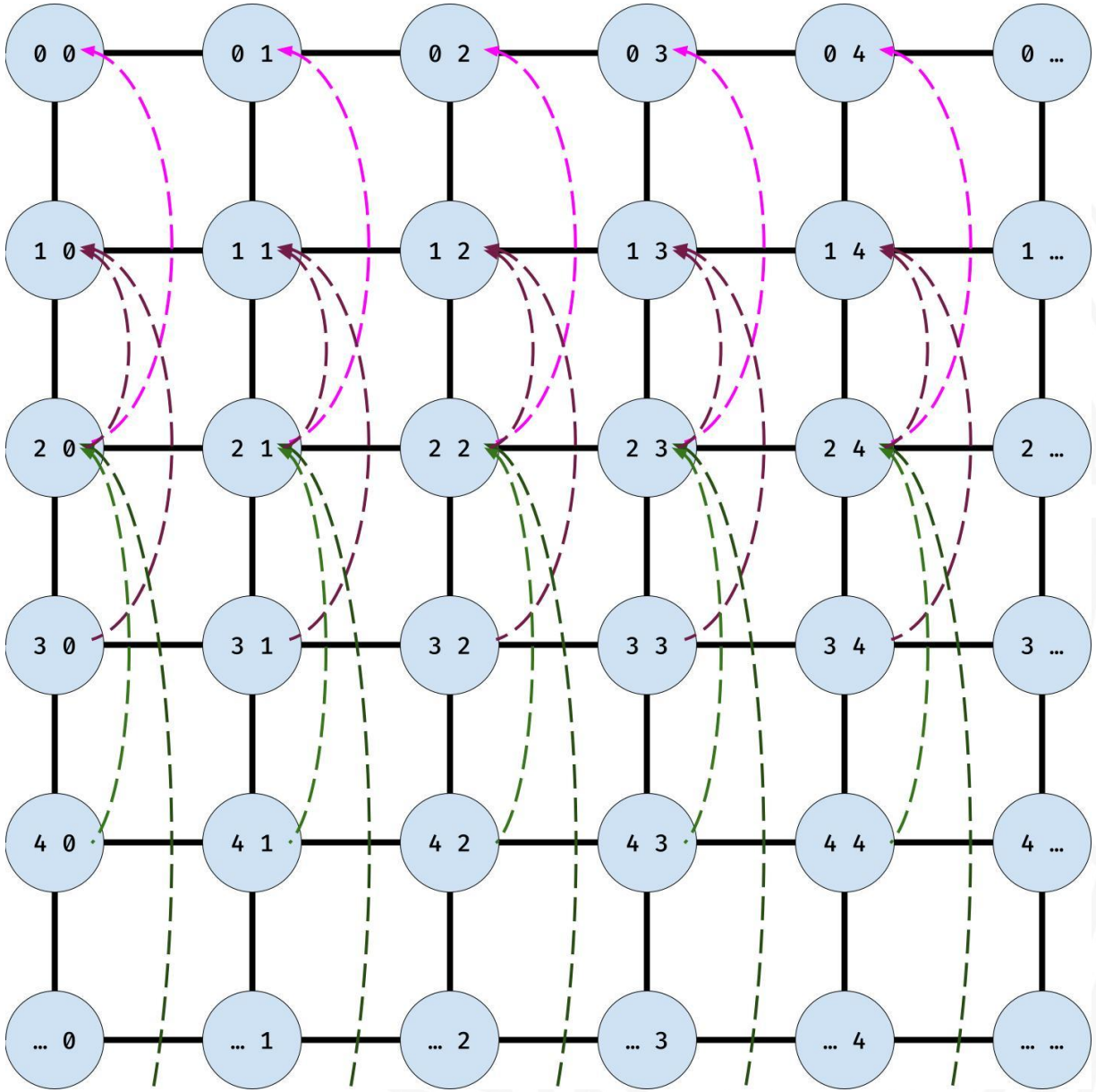
1. Distribute R-values of pixels over the processors s.t. `pixel[x y]` → `P[x y]`
2. For `i` in `0 .. k` do
   a. Perform Horizontal Merging.
   b. Perform Vertical Merging.

   where `k = log(X)`



Procedure for Vertical Merging

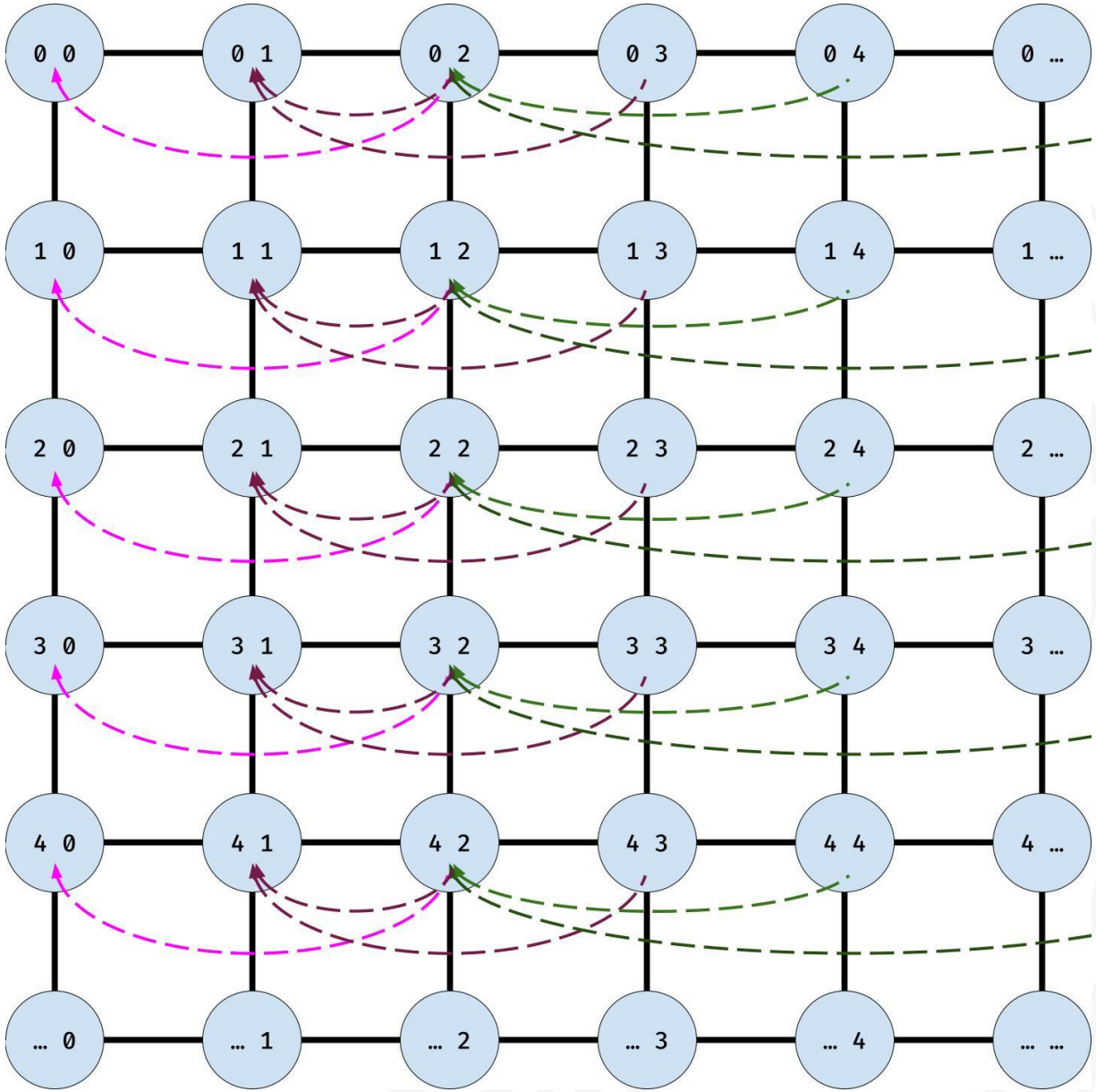
1. Every Processor, `P[i  j]`, gets values of `P[2i  j]` and `P[2i+1  j]`
2. `P[i  j]` calculates the average of the 2 values, `(P[2i  j] + P[2i+1  j])/2`
3. `P[i  j]` stores this calculated average as its new value.



Procedure for Down-scaling

1. Distribute R-values of pixels over the processors s.t. `pixel[x y]` → `P[x y]`
2. For `i` in `0 .. k` do
   a. Perform Horizontal Merging.
   b. Perform Vertical Merging.

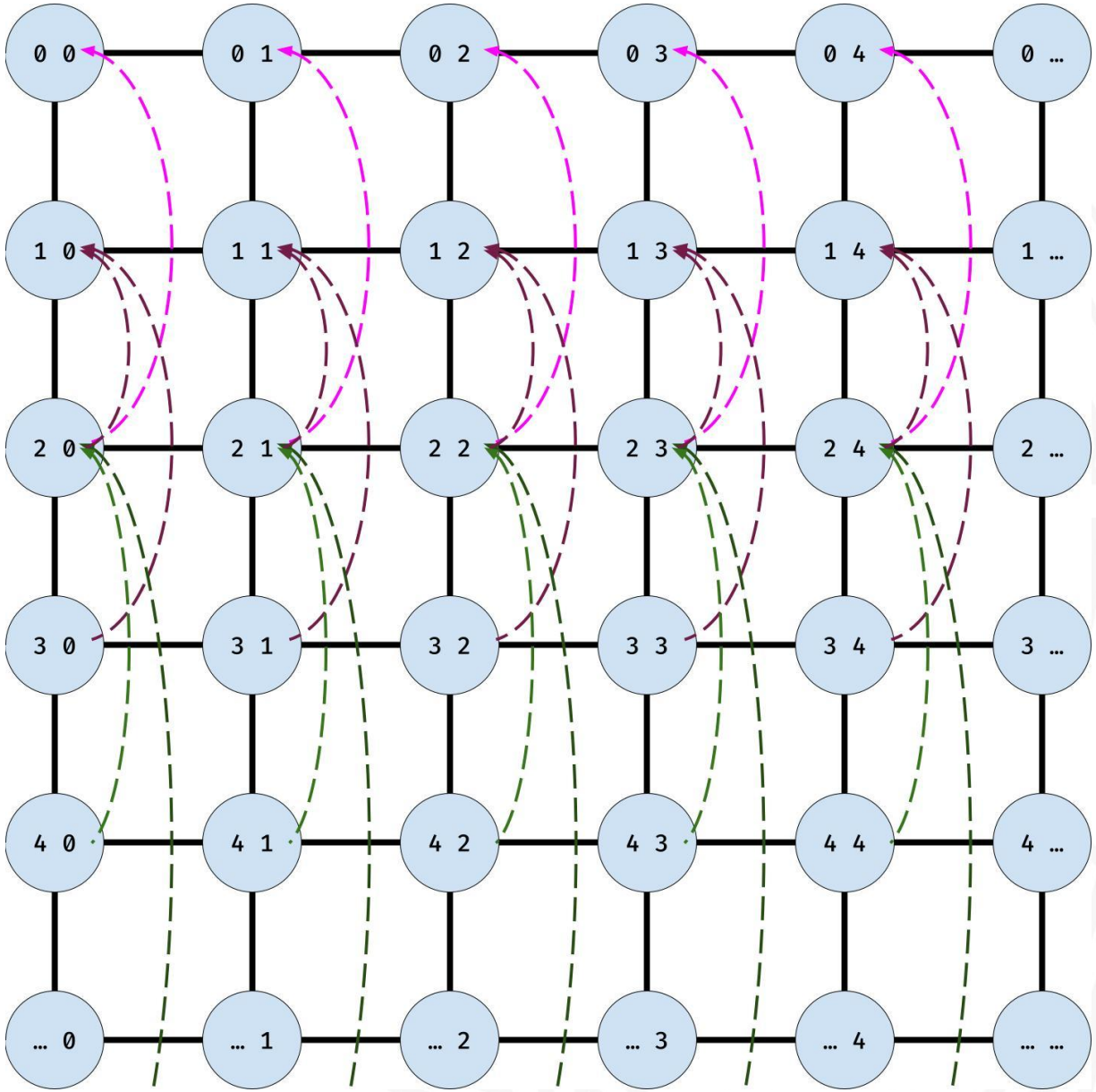
   where `k = log(X)`



Procedure for Down-scaling

1. Distribute R-values of pixels over the processors s.t. `pixel[x y]` → `P[x y]`
2. For `i` in `0 .. k` do
   a. Perform Horizontal Merging.
   b. Perform Vertical Merging.

   where `k = log(X)`



Coarse-grained algorithm

Coarse-grained Algorithm

- Assuming we have X * Y image, and K * K processors.
- We create a (X/K) * (Y/K) sub-image on each processor.
- We assign each pixel of the sub-image global indices (these are the indices of the pixel w.r.t. the original image).
- Now, we can represent a global 2-D array `
- Then we start performing parallel downscaling.
- Each pixel on the global array performs horizontal and vertical merging.
- This happens by sending sequence of messages of pixel values, and new indices from one processor to other processors.

Implementation Details

- When ever 2 processors have to exchange pixel values they exchange pixel values in row major ordering. (i.e. first processor prioritises pixel (0,0) then (0,1) then (0,2) and so on). This way we get a unique index of message exchanges that every 2 pair of processors can follow. (this is because in MPI both receiving and sending processors have to initiate message exchange synchronously).
- But using this method of communication increases the complexity of the communication operation to $O((X/K) * (Y/K)) = O(X^2)$ which is very bad! compared to log time complexity of merge operations.

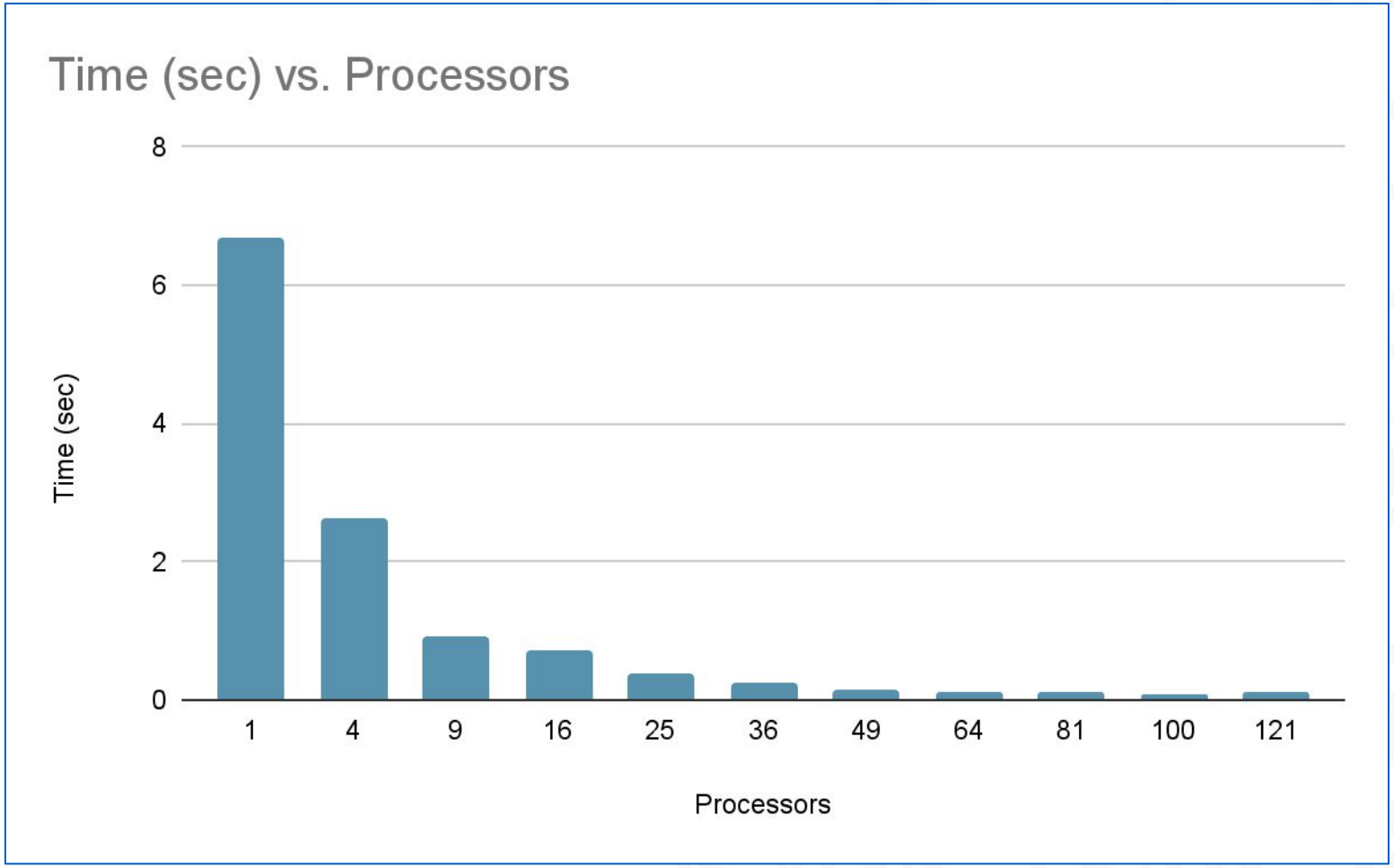
Optimisation!!!

- To counter this, we maintain message queues for each processor that stores all the pixel values it has to exchange with other processors.
- Using message queues, we exchange the entire message queue between processors at once which reduces the time complexity of communication operation to $O(K^2)$.
- This not only reduces the number of messages between processors to one, but it also forces constant number of messages that the whole grid exchanges!!!

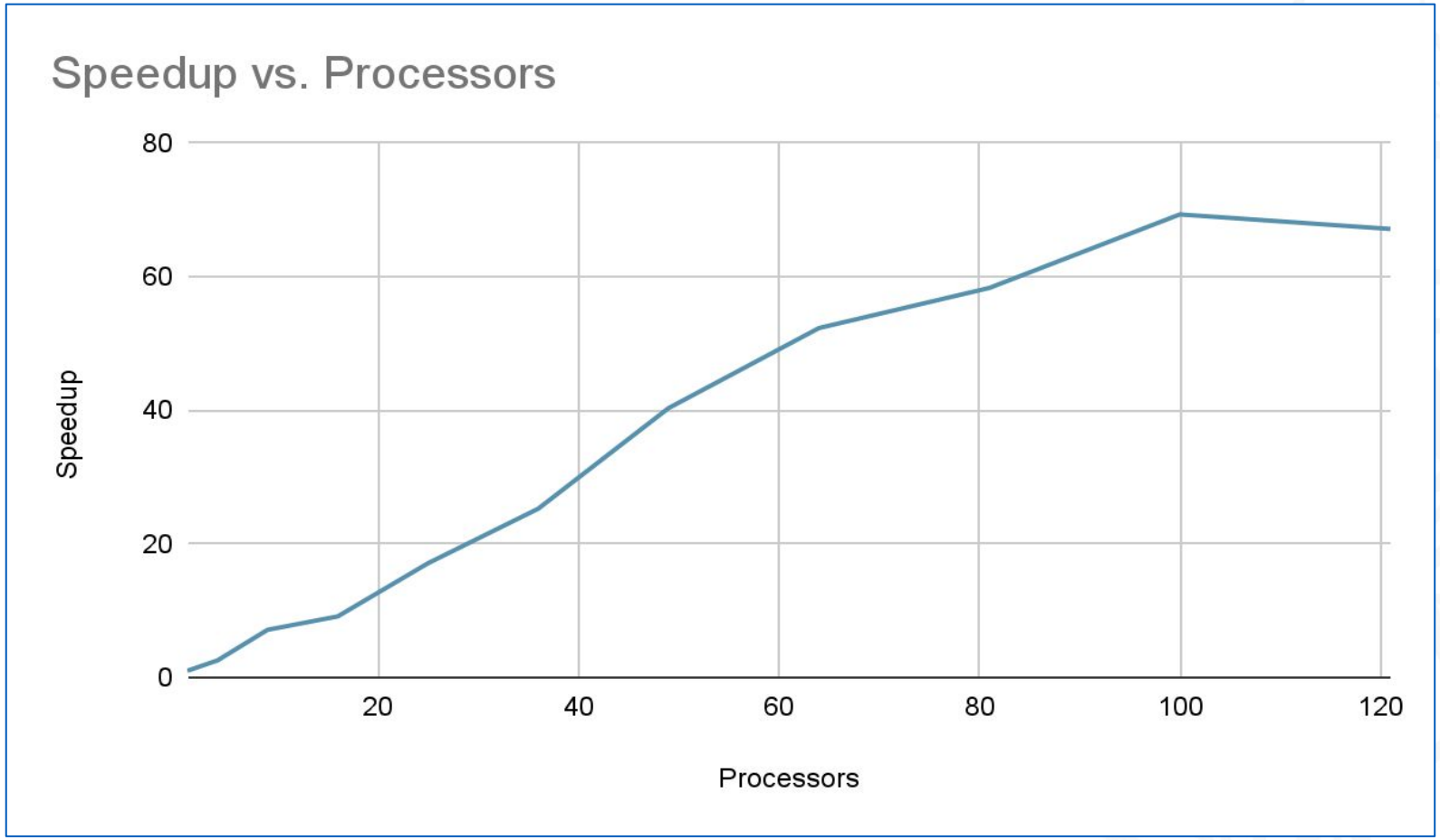
Results

For 100M pixel (10K * 10K)(13 iterations) image with 100 color-dimensions

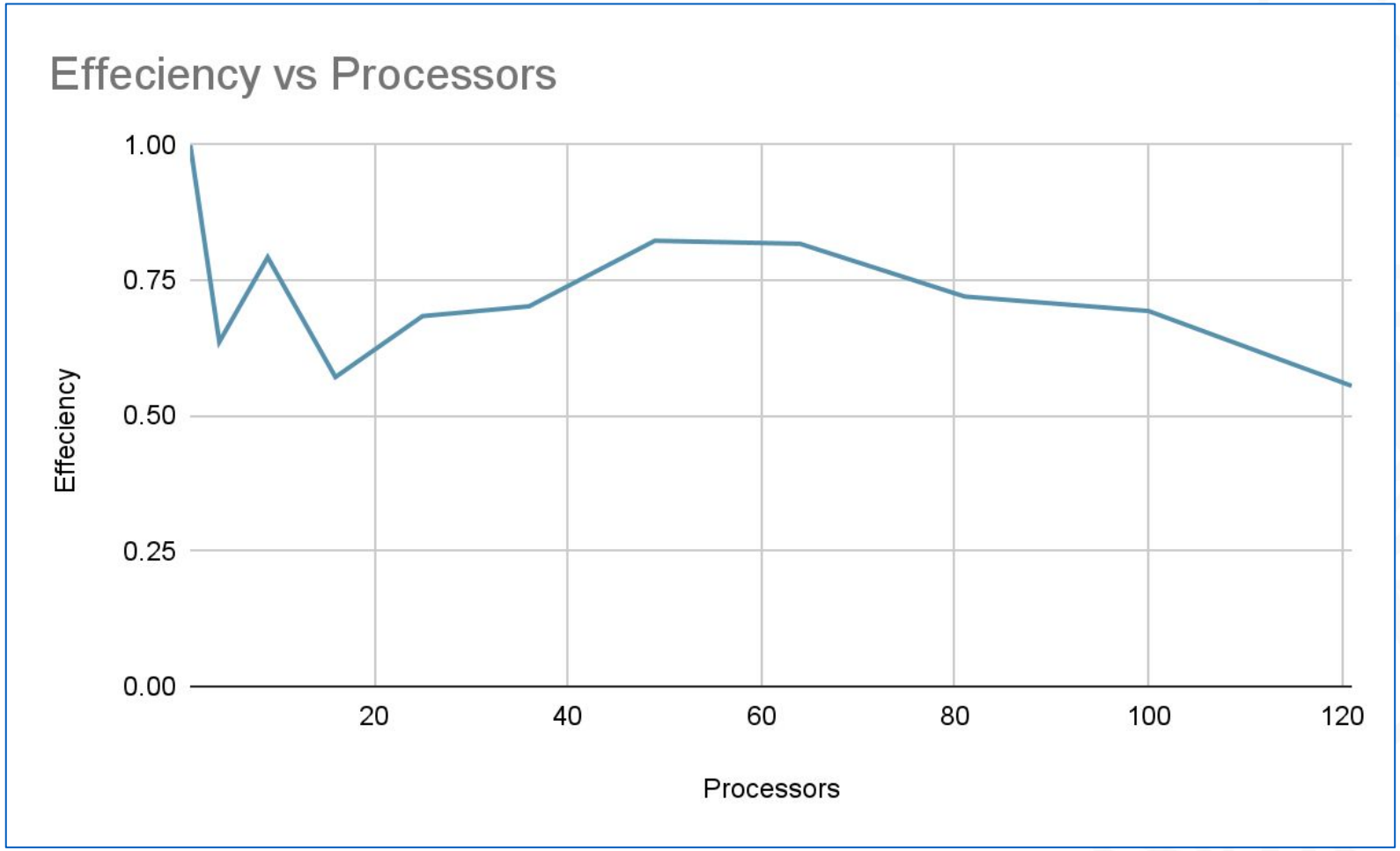
Processors	Avg Time across colors (sec)
1	6.680345
4	2.624891
9	0.936577
16	0.730853
25	0.390719
36	0.264356
49	0.165643
64	0.127688
81	0.11453
100	0.096347
121	0.099478



For 100M pixel (10K * 10K)(13 iterations) image with 100 color-dimensions



For 100M pixel (10K * 10K)(13 iterations) image with 100 color-dimensions

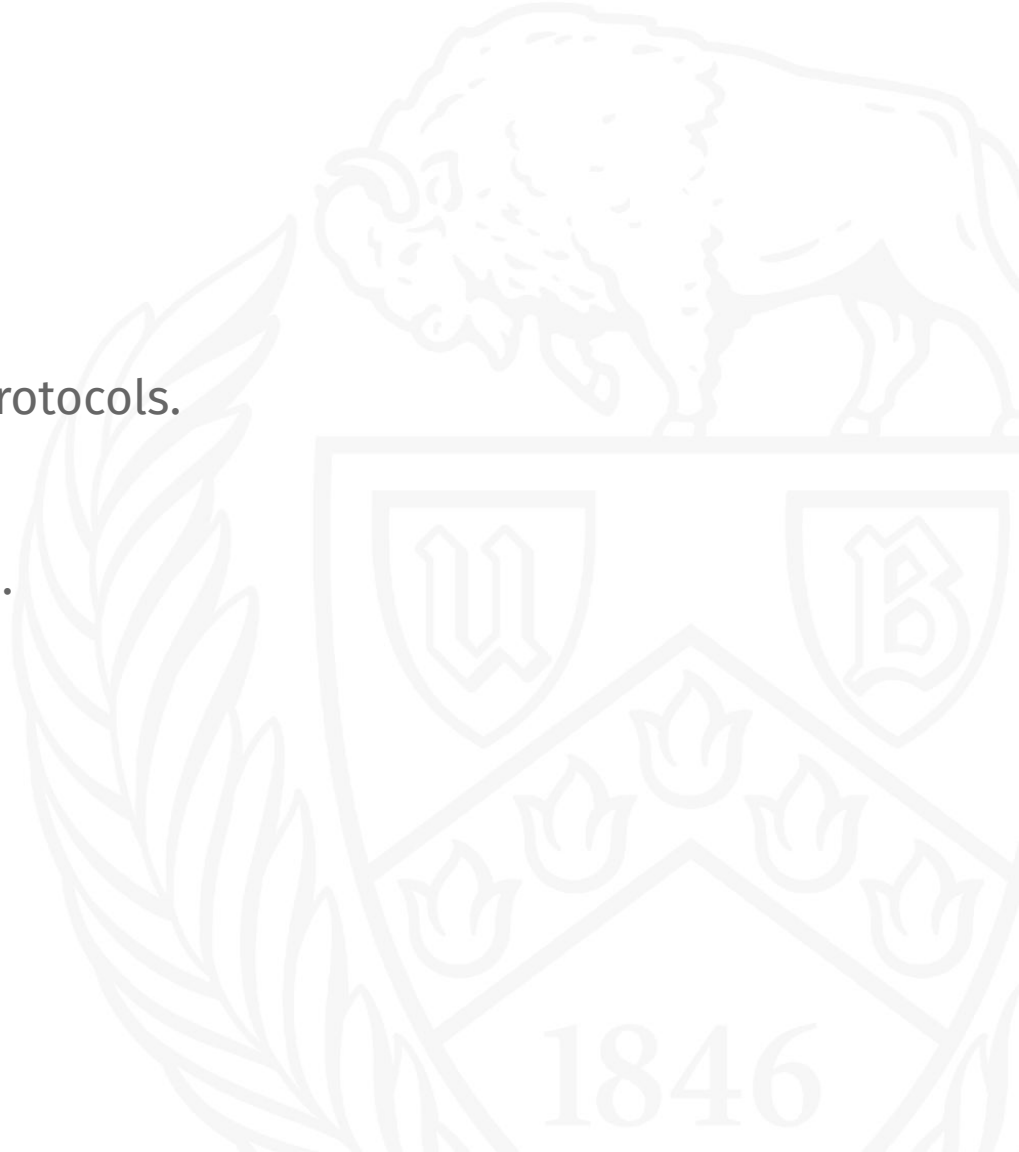


References

1. Algorithms, Sequential and Parallel: A Unified Approach – Russ Miller and Laurence Boxer. 3rd Edition.

Takeaways

1. Programming in MPI.
2. Concurrency Debugging.
3. Optimizing cross process communication using messaging protocols.
4. SLURM and linux CI/CD.
5. Cost optimizing parallel algorithms by data-driven approach.



University at Buffalo The State University of New York

Questions

University at Buffalo The State University of New York

Thank You!