CS425: Algorithms for Web Scale Data

Lecture 6: MapReduce Complexity Analysis and Efficient Algorithms

Most of the slides are from the Mining of Massive Datasets book.
These slides have been modified for CS425. The original slides can be accessed at: www.mmds.org

Complexity Analysis of MapReduce Algorithms

Communication Cost Model

- The model we will use:

Communication cost = sum of input sizes to each stage

- Output sizes are ignored
  - *If the output is large, it's likely that it will be input to another stage*
  - *The real outputs are typically small, e.g. some summary statistics, etc.*
- Reading from disk is part of the communication cost
  - *e.g. The input to the map stage can be from the disk of a reduce task at a different node*
- Analysis is independent of scheduling decisions
  - *e.g. Map and reduce tasks may or may not be assigned to the same node.*

Definitions: Replication Rate & Reducer Size

- **Replication rate**: Avg # of key-value pairs generated by Map tasks per input
  - The communication cost between Map and Reduce is determined by this
  - Donated as **r**
- **Reducer size:** Upper bound for the size of the value list corresponding to a *single* key
  - Donated as **q**
  - Choose **q** small enough such that:
    1. there are many reducers for high levels of parallelism
    2. the data for a reducer fits into the main memory of a node
- Typically **q** and **r** inversely proportional
  - Tradeoff between communication cost and parallelism/memory requirements.

Example: Join with MapReduce

- Map:
  - For each input tuple **R(a, b):**

    Generate <**key** = **b**, **value** = **(‘R’, a)**>
  - For each input tuple **S(b, c):**

    Generate <**key** = **b**, **value** = **(‘S’, c)**>

- Reduce:
  - Input: <**b**, **value_list**>
  - In the value_list:
    - Pair each entry of the form **(‘R’, a)** with each entry **(‘S’, c)**, and output:

      <a, b, c>

Replication rate:

r = 1

Communication cost:

2(|R|+|S|)

Reducer size (worst case):

q = |R| + |S|

Example: Single-Step Matrix-Matrix Multiplication

- **Map(input):**

  for each m_{ij} entry from matrix **M**:

  for **k**=1 to n

  generate <**key** = **(i, k)**, **value** = (**'M', j,** m_{ij}) >

  for each n_{jk} entry from matrix **N**:

  for **i**=1 to n

  generate <**key** = **(i, k)**, **value** = (**'N', j,** n_{jk}) >

- **Reduce(key, value_list)**

  sum ← 0

  for each pair **(M, j,** m_{ij}**)** and **(N, j,** n_{jk}**)** in value_list

  sum += $m_{ij} \cdot n_{jk}$

  output **(key, sum)**

*Assume both **M** and **N** have size **nxn***

Replication rate:

r = n

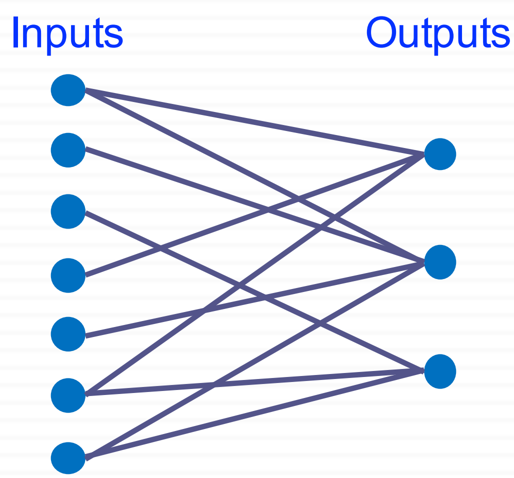
Communication cost:

$2n^2 + 2n^3$

Reducer size:

q = 2n

A Graph Model for MapReduce Algorithms

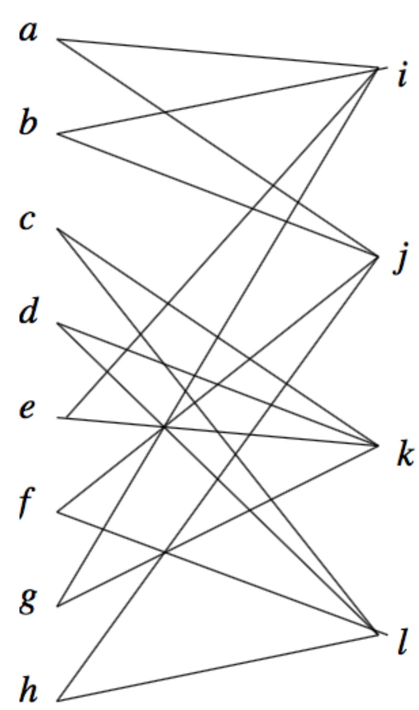


- Define a vertex for each input and output
- Define edges reflecting which inputs each output needs
- Every MapReduce algorithm has a schema that assigns outputs to reducers.

- Assume that max reducer size is **q**.
- Assignment Requirements:
  1. No reducer can be assigned more than **q** inputs.
  2. Each output is assigned to at least one reducer that receives all inputs needed for that output.

Example: Single-Step Matrix-Matrix Multiplication

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} i & j \\ k & l \end{bmatrix}$$



We have assigned each output to a single reducer.
The replication rate $r = n$
The reducer size $q = 2n$

Application: Naïve Similarity Join

Naïve Similarity Join

- **Objective**: Given a large set of elements **X** and a similarity measure **$s(x_1, x_2)$**, output the pairs that have similarity above a given threshold.
  - *Locality sensitive hashing is not used for the sake of this example.*

- Example:
  - Each element is an image of 1M bytes
  - There are 1M images in the set
  - About 5×10^{11} (500B) image comparisons to make

Similarity Join with MapReduce (First Try)

- Let **n** be the # of pictures in the set.

- **Map:**

  for each picture P_i do:

  for each **j=1** to **n** (except **i**)

  generate <**key** = **(i,j)**, **value** = P_i>

- **Reduce (key, value_list)**

  compute **sim**(P_i, P_j)

  output **(i,j)** if similarity is above threshold

Replication rate **r = n-1**
Reducer size **q = 2**
Communication cost = **n + n(n-1)**
# of reducers **= n(n-1)/2**

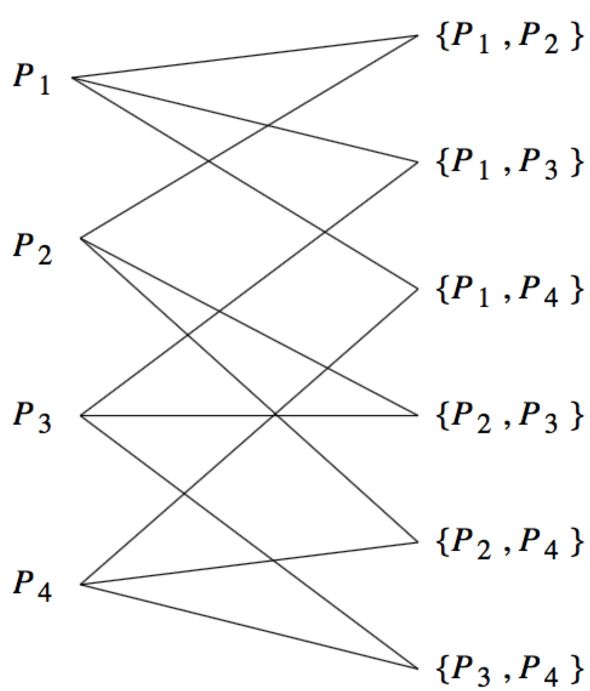
Example: 1M pictures with 1MByte size each

- Communication cost:
  n(n-1) pictures communicated from Map to Reduce
  total # bytes transferred $= 10^{18}$

- Assume gigabit ethernet:
  time to transfer 10^{18} bytes = 10^{10} seconds (~300 years)

- Replication rate **r = n-1**
- Reducer size **q = 2**
- Communication cost = **n + n(n-1)**
- # of reducers **= n(n-1)/2**

Graph Model

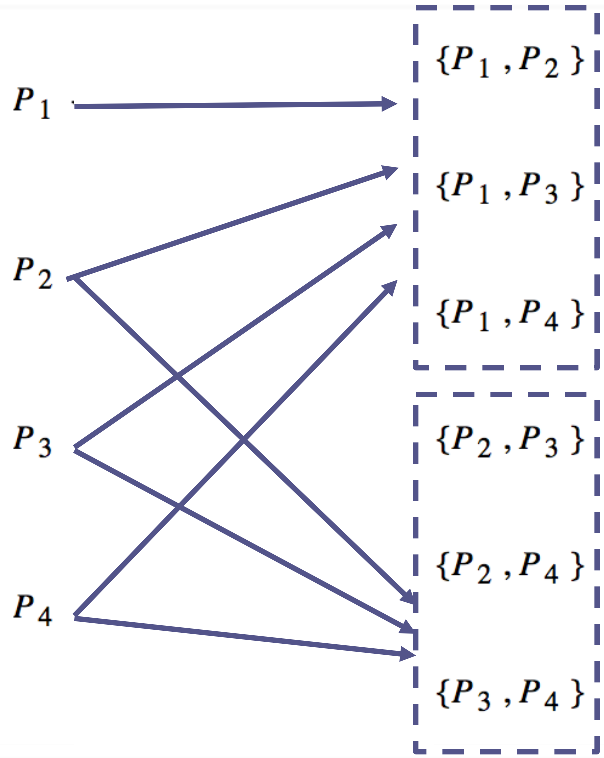


Our MapReduce algorithm:

- One reducer per output.
- P_i must be sent to each output.
- Replication rate $r = n-1$
- Reducer size $q = 2$

What if a reducer *covers* multiple outputs?

Graph Model: Multiple Outputs per Reducer



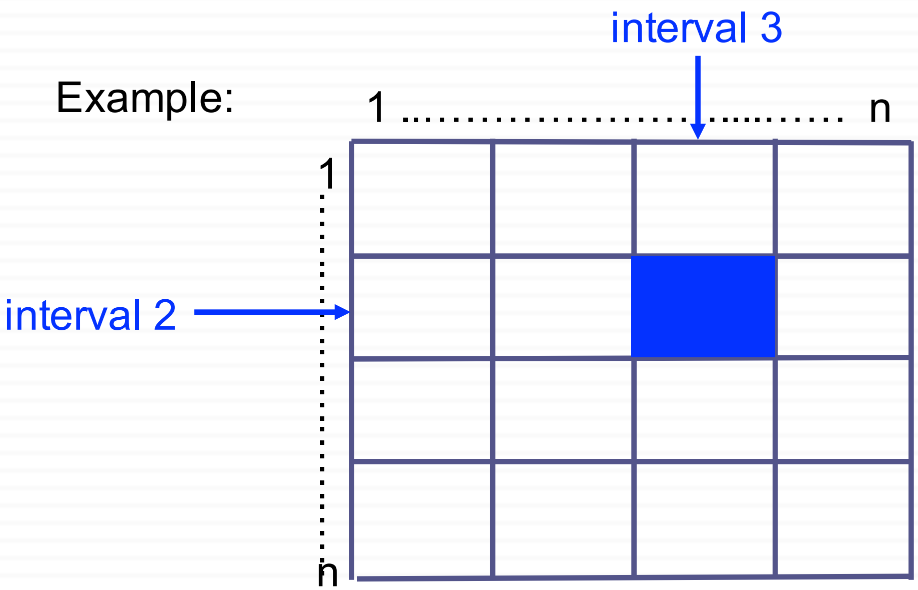
Replication rate & communication cost reduced.

How to do the grouping?

Grouping Outputs

- Define **g** intervals between **1** and **n**.
- **Reducer (u,v)** will be responsible for comparing all inputs in **range u** with all inputs in **range v**.

Example:



Reducer (2, 3) will compare all entries in interval 2 with all entries in interval 3.

Similarity Join with Grouping

- Let **n** be the number of inputs, and **g** be the number of groups.
- **Map:**

  for each P_i in the input

  let **u** be the group to which **i** belongs

  for $v = 1$ to **g**

  generate < **key**=**(u, v)**, **value**=**(i, P_i)** >

- **Reduce(key=(u,v), value_list)**

  for each **i** that belongs to group **u** in **value_list**

  for each **j** that belongs to group **v** in **value_list**

  compute **sim(P_i, P_j)**, and output **(i, j)** if it is above threshold.

Problem:

P_i will be sent to (g_i, g_j)

P_j will be sent to (g_j, g_i)

Similarity Join with Grouping

- Let **n** be the number of inputs, and **g** be the number of groups.
- **Map:**

  for each $\mathbf{P_i}$ in the input

  let **u** be the group to which **i** belongs

  for **v = 1** to **g**

  generate < **key**=[min**(u, v)**, max(**u,v**)], **value**=**(i, $\mathbf{P_i}$)** >

Single key generated for (u,v) and (v,u)

- **Reduce(key=(u,v), value_list)**

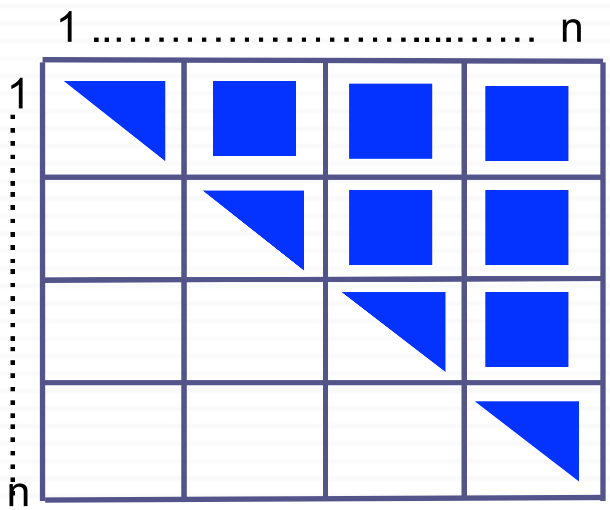
  for each **i** that belongs to group **u** in **value_list**

  for each **j** that belongs to group **v** in **value_list**

  compute **sim($\mathbf{P_i}$, $\mathbf{P_j}$),** and output **(i, j)** if it is above threshold.

Example

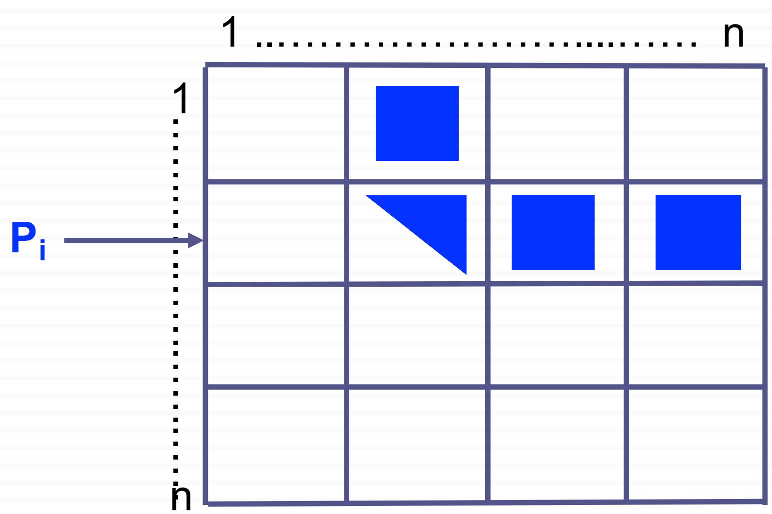
Example: If **g = 4**, the highlighted comparisons will be performed.



There will be a reducer for each key **(u, v),** where **u ≤ v**

Example

Which reducers will receive and use P_i in group 2?



Reducers: (1, 2), (2, 2), (2, 3), (2, 4)

Complexity Analysis

- Replication rate:

  $r = g$

- Reducer size:

  $q = 2n/g$

- Communication cost:

  $n+ng$

- # of reducers:

  $g(g+1)/2$

Example: 1M pictures with 1MByte size each

- Let **$g = 1000$**
- Reducer size **$q = 2n/g$**
  - memory needed for one node: ~2GB *(reasonable)*
- Communication cost = **$n + ng$**
  - total # bytes transferred = **$\sim 10^{15}$** *(still a lot, but 1000x less than before)*
- # of reducers = **$g(g+1)/2$**
  - there are ~**500K** reducers *(enough parallelism for 1000s of nodes)*
- What if g = 100?

Tradeoff Between Replication Rate and Reducer Size

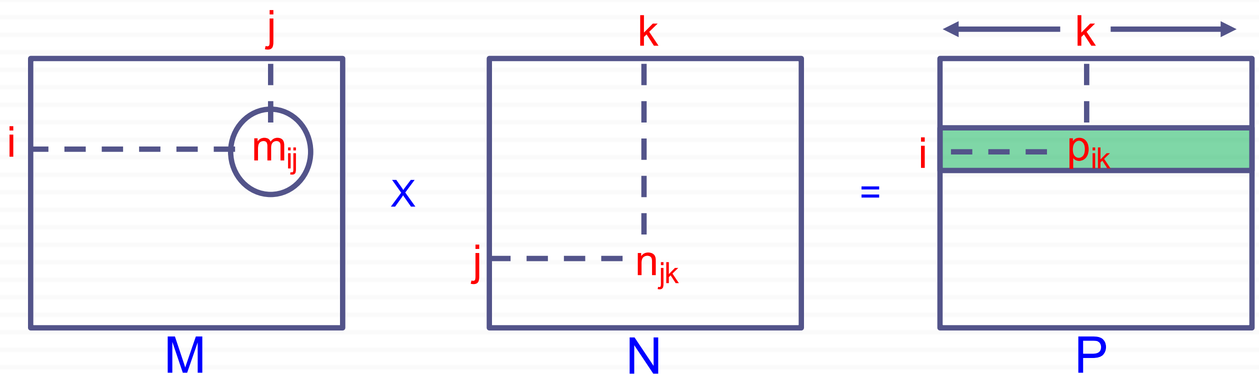
Replication rate $r = g$
Reducer size $q = 2n/g$

⟹ $q = 2n / r$ ⟹ $qr = 2n$

- Replication rate and reducer size are inversely proportional.
- Reducing replication rate will reduce communication, but will increase reducer size.
  - *Extreme case: $r = 1$ and $q = 2n$. There is a single reducer doing all the comparisons.*
  - *Extreme case: $r = n$ and $q = 2$. There is a reducer for each pair of inputs.*
- Need to choose **r** small enough such that the data fits into local DRAM and there's enough parallelism.

Application: Matrix-Matrix Multiplication with 1D Decomposition

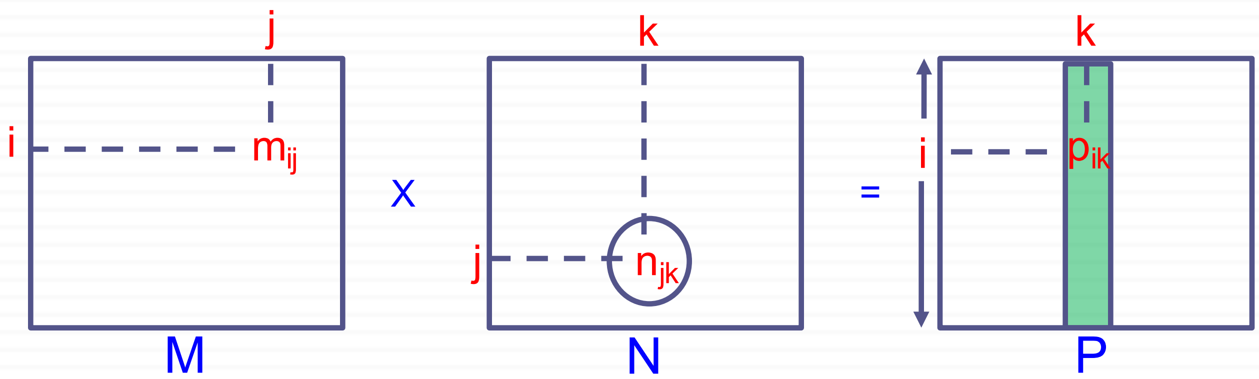
Reminder: Matrix-Matrix Multiplication without Grouping



$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Each $\mathbf{m_{ij}}$ needs to be sent to each reducer **(i, k)** for all **k**

Reminder: Matrix-Matrix Multiplication without Grouping

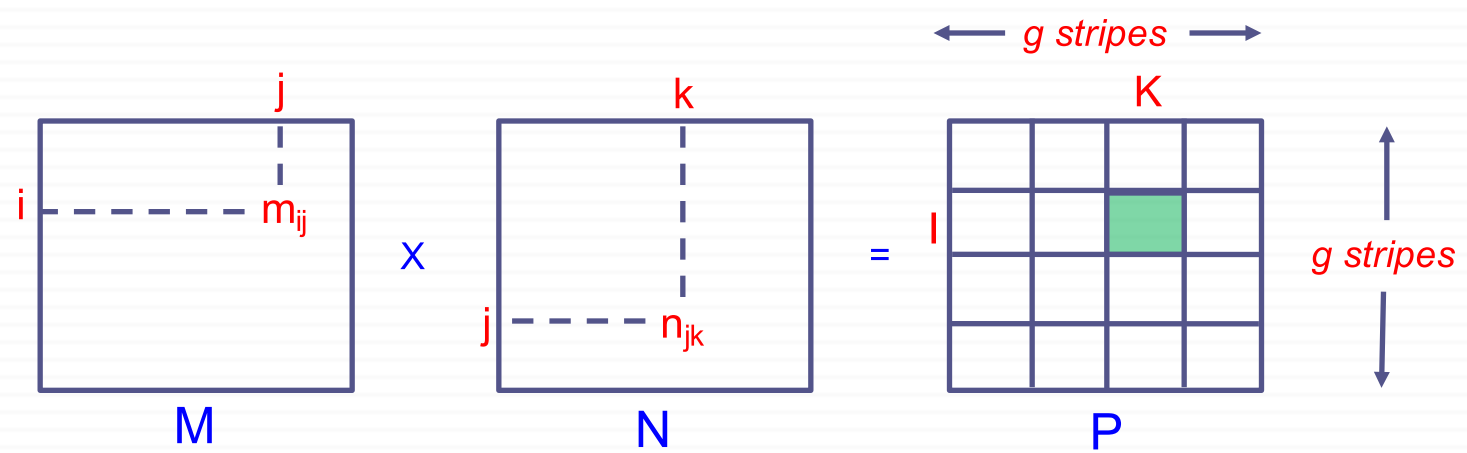


$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Each n_{jk} needs to be sent to each reducer **(i, k)** for all **i**

Replication rate **r = n**

Multiple Outputs per Reducer



Notation:

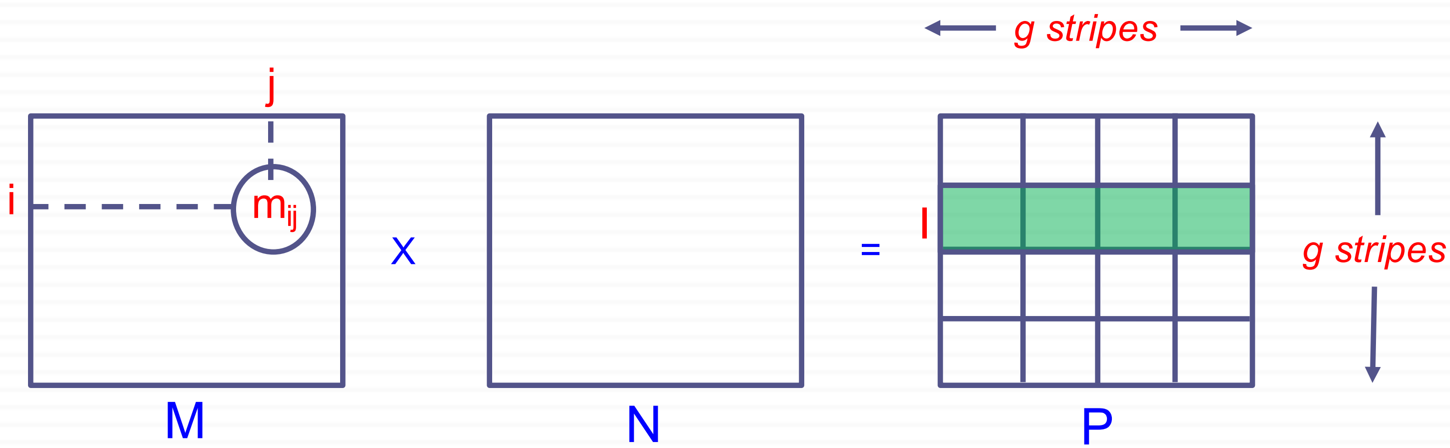
j: row/column index of an individual matrix entry

J: set of indices that belong to the **J**th interval.

Let reducer **(I,K)** be responsible for computing all p_{ik} where:

$i \in I$ and $k \in K$

Multiple Outputs per Reducer

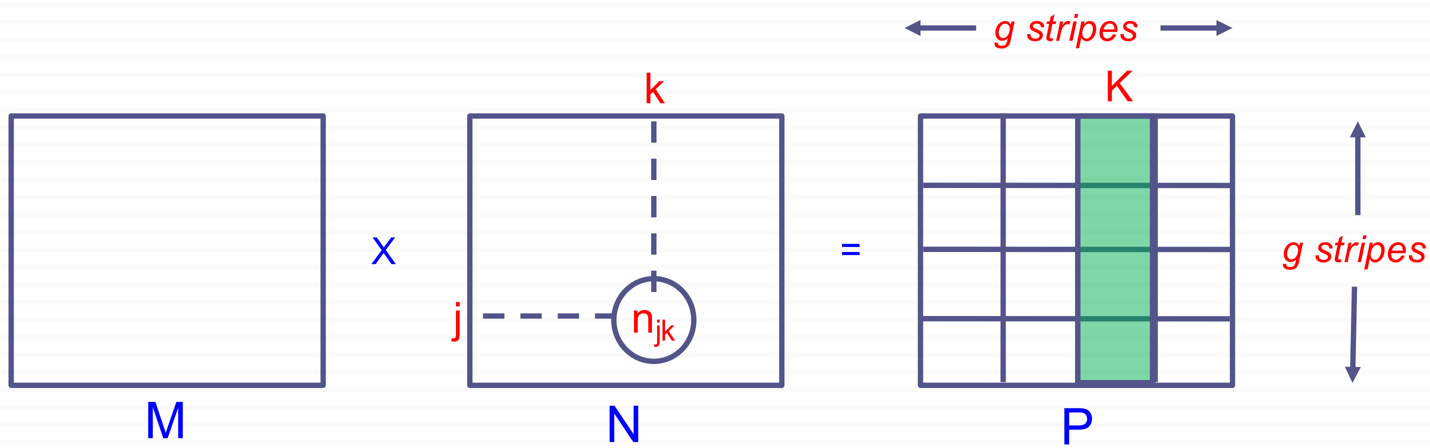


Which reducers need m_{ij}?

Reducers **(I, K)** for all **$1 \leq K \leq g$**

Replication rate **$r = g$**

Multiple Outputs per Reducer

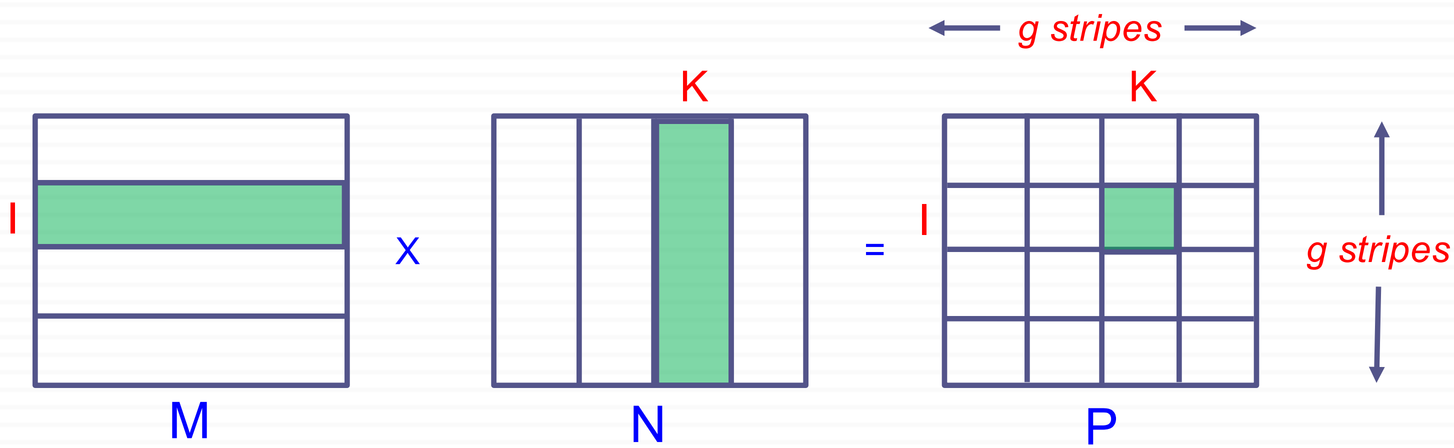


Which reducers need n_{jk}?

Reducers **(I, K)** for all **$1 \leq I \leq g$**

Replication rate **$r = g$**

1D Matrix Decomposition



Which matrix elements will reducer **(I, K)** receive?

Ith row stripe of **M** and **K**th column stripe of **N**

MapReduce Formulation

- **Map**:

  for each element m_{ij} from matrix M

  for $K=1$ to g

  generate <key=(I, K), value = ('M', i, j, m_{ij})>

  for each element n_{jk} from matrix N

  for $I=1$ to g

  generate <key=(I, K), value = ('N', j, k, n_{jk})>

- **Reduce**(key=(I,K), value_list)

  for each $i \in I$ and for each $k \in K$

  $p_{ik} = 0$

  for $j = 1$ to n

  p_{ik} += $m_{ij} \cdot n_{jk}$

  output <key=(i, k), value = p_{ik}>

Replication rate:
$r = g$

Communication cost:
$2n^2 + 2gn^2$

Reducer size:
$q = 2n^2/g$

# of reducers:
g^2

Communication Cost vs. Reducer Size

Replication rate vs. reducer size

$q = 2n^2/g \Rightarrow q = 2n^2/r \Rightarrow qr = 2n^2$

Communication cost vs. reducer size

$$\text{cost} = 2n^2 + 2gn^2$$
$$= 2n^2 + 4n^4/q$$

Inverse relation between communication cost and reducer size.

Reminder: q value chosen should be small enough such that:

Local memory is sufficient

There's enough parallelism

Replication rate:
$r = g$

Communication cost:
$2n^2 + 2gn^2$

Reducer size:
$q = 2n^2/g$

# of reducers:
g^2

Application: Matrix-Matrix Multiplication with 2D Decomposition

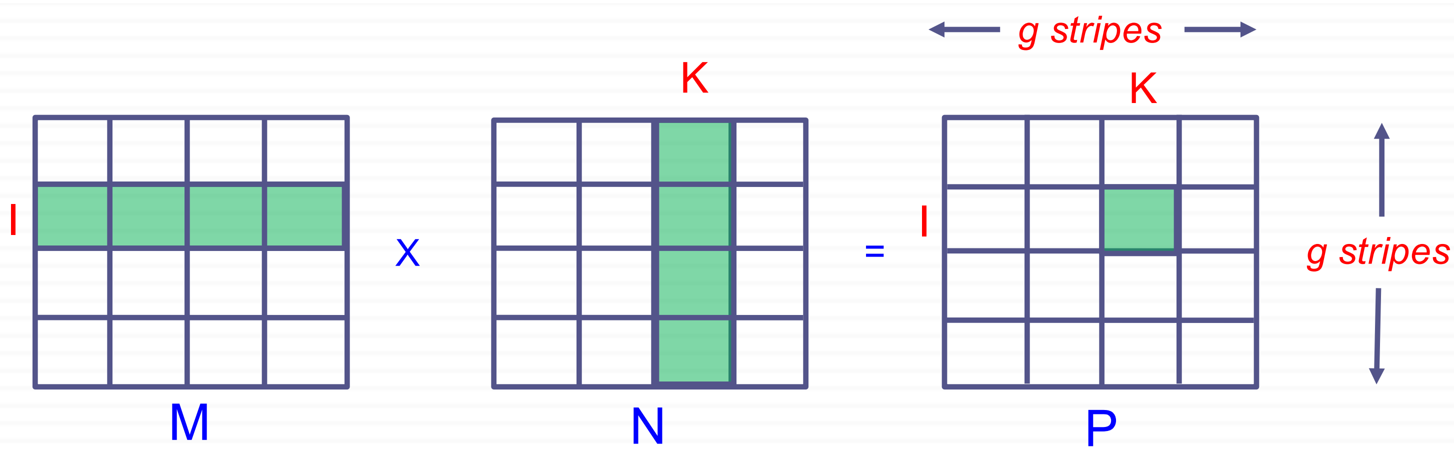
Two Stage MapReduce Algorithm

- What are we trying to achieve?
  - A better tradeoff between replication rate **r** and reducer size **q**
  - The previous algorithm: $qr = 2n^2$
  - We will show that we can achieve $qr^2 = 2n^2$
    - For the same reducer size, the replication rate will be smaller

- Reminder: Two-stage MapReduce without grouping:
  - Stage 1: "Join" matrix entries that need to be multiplied together
  - Stage 2: Sum up products to compute final results

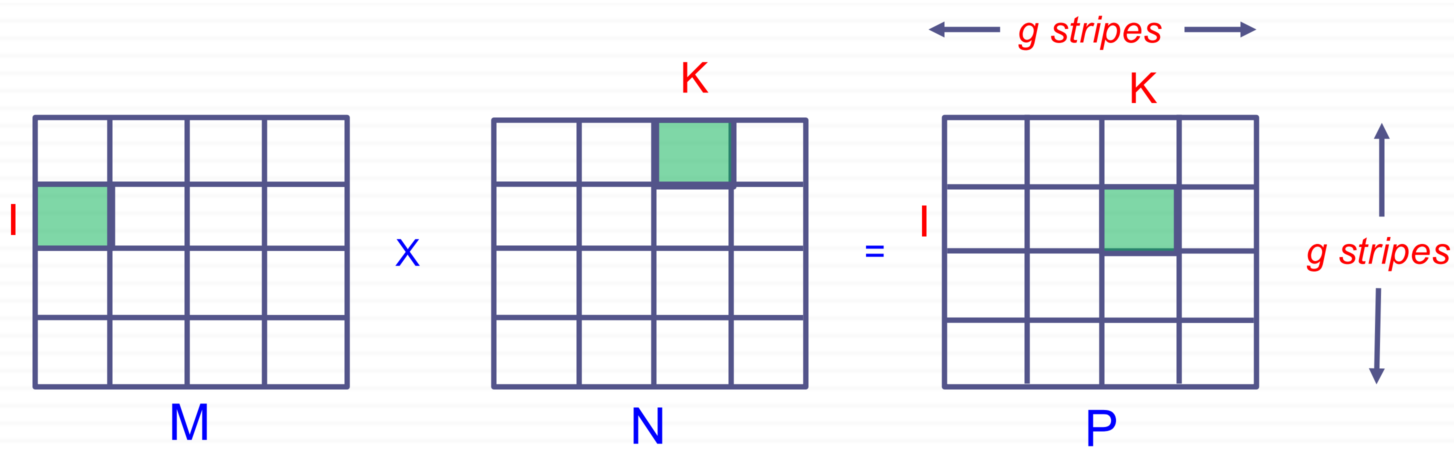
- Use a similar idea, but for sub-blocks of matrices instead of individual elements

2D Matrix Decomposition



Assume that M and N are partitioned to g horizontal and g vertical stripes.

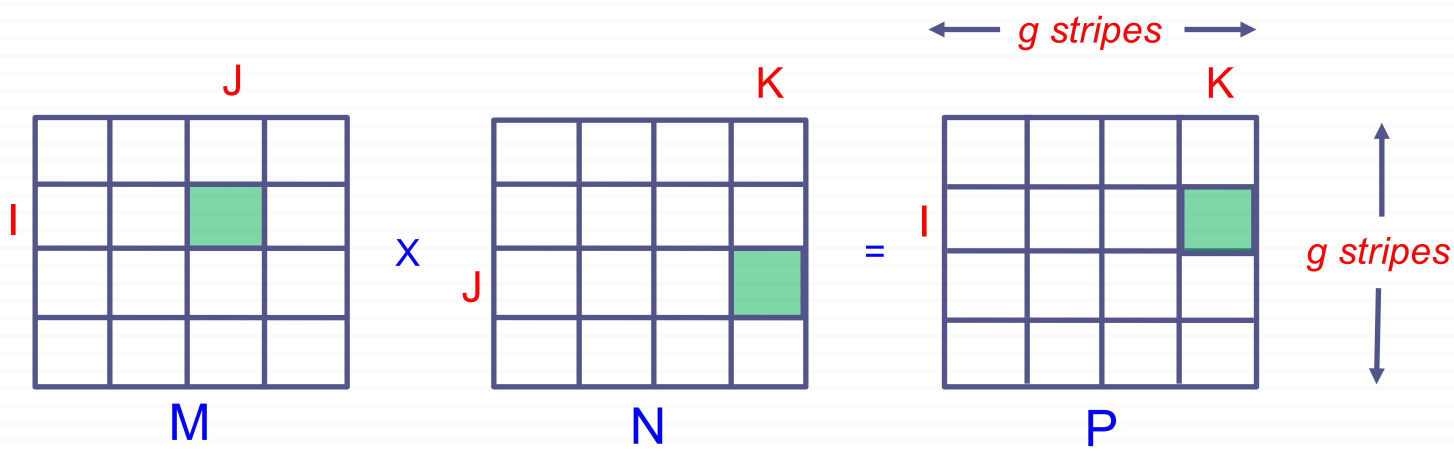
Computing the Product at Stripe (I, K)



$$P_{IK} = \sum_{J=1}^{J=g} M_{IJ} x\, N_{JK}$$

*Note: **M_{IJ} x N_{JK}** is multiplication of two sub-matrices*

How to Define Reducers?



M_{IJ} needs to be multiplied with **N_{JK}** and will produce the partial sum P^{J}_{IK}.

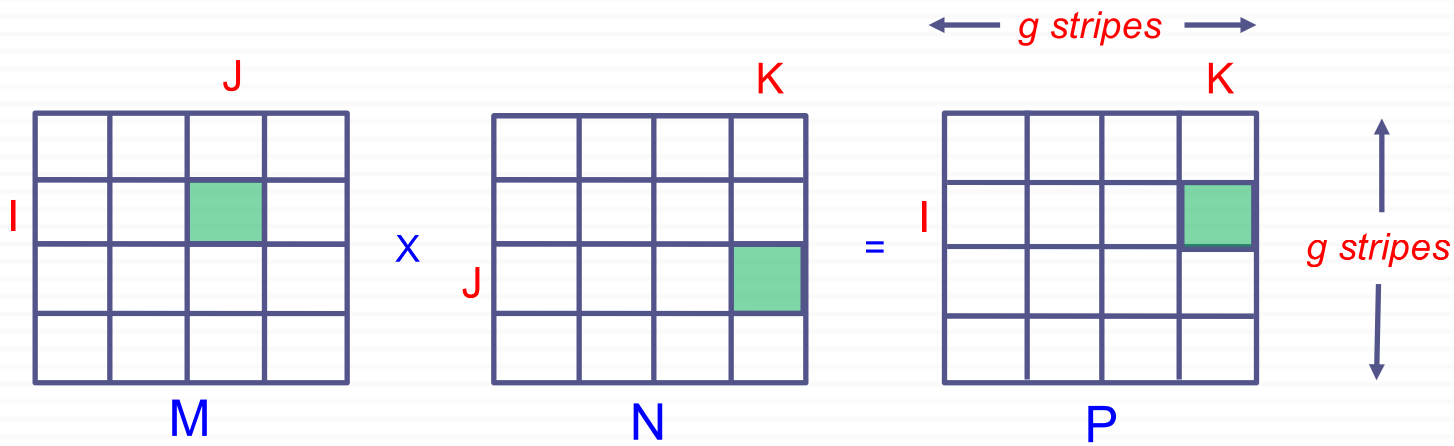
What if we define a reducer for each **(I, K)**?

It would be identical to the 1D decomposition

What if we define a reducer for each **J**?

*Exercise: Derive the communication cost as a function of **n** and **q***

How to Define Reducers?



M_{IJ} needs to be multiplied with **N_{JK}** and will produce the partial sum P^{J}_{IK}.

What if we define a reducer for each **(I, J, K)**?

Smaller reducer size

*Reducer **(I, J, K)** will be responsible for computing the **J^{th}** partial sum for block **P_{IK}***

First MapReduce Step

- **Map**:

  for each m_{ij} in **M**

  for **K** = **1** to **g**

  generate <**key** = **(I, J, K)**, **value** = **('M', i, j, m_{ij})**

  for each n_{jk} in **N**

  for **I** = **1** to **g**

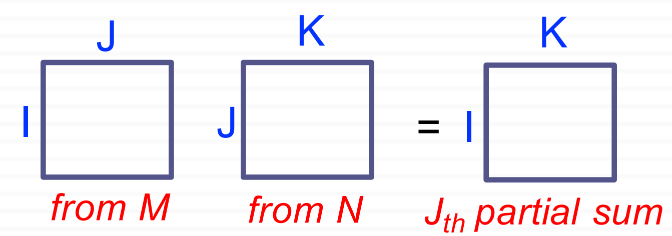
  generate <**key** = **(I, J, K)**, **value** = **('N', j, k, n_{jk})**

- **Reduce**(**key** = **(I, J, K)**, **value_list**)

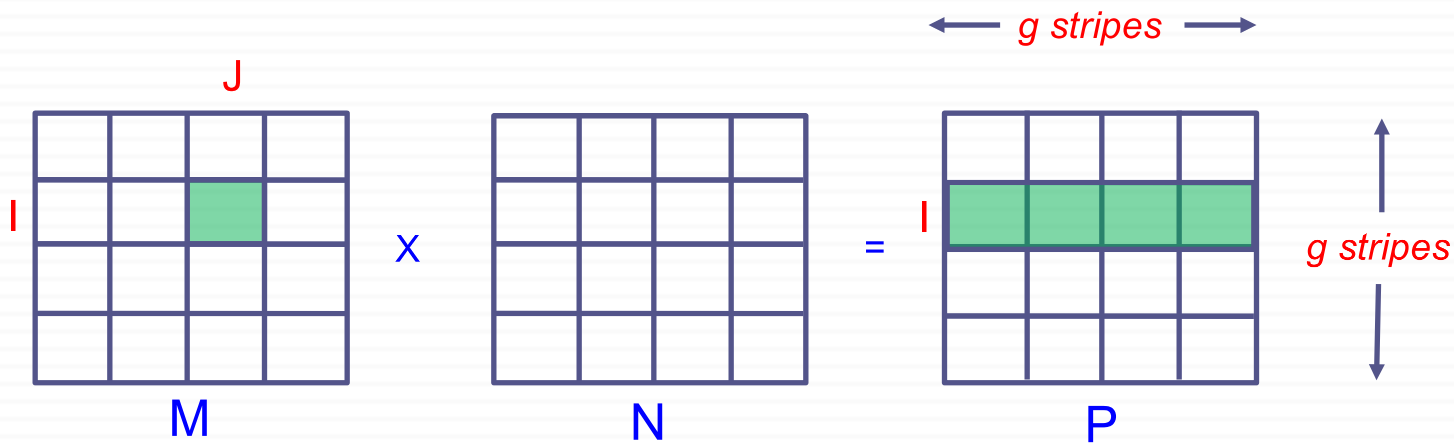
  for each **i** ∈ **I** and **k** ∈ **K**

  compute $x^J_{ik} = \sum_{j \in J} m_{ij} n_{jk}$

  output <**key** = **(i, k)**, **value** = x^J_{ik}>



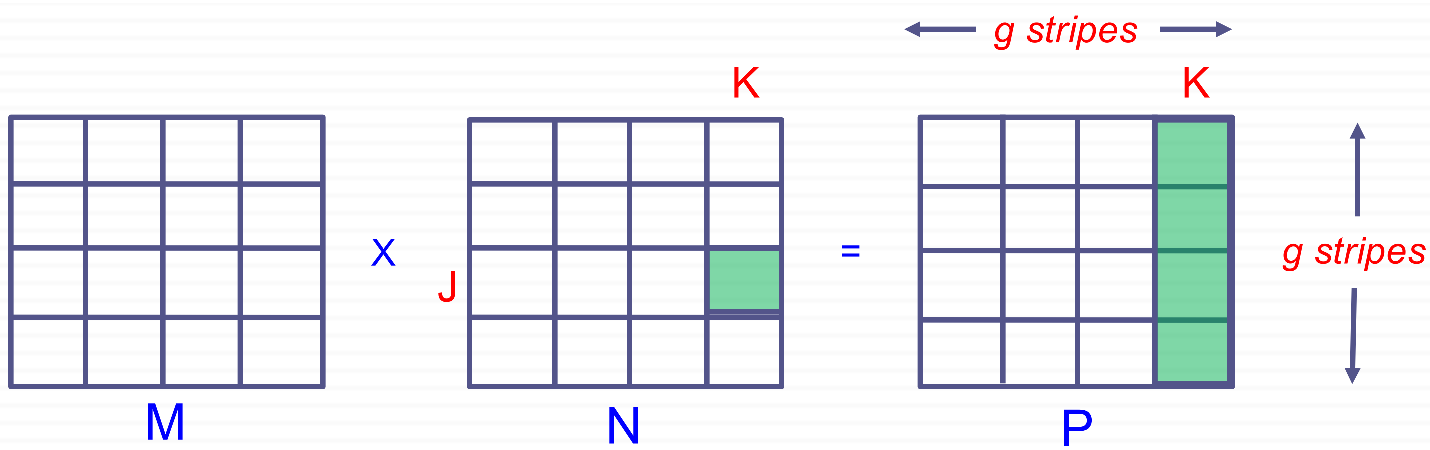
MapReduce Step 1: Map



Block M_{IJ} will be sent to the reducers **(I, J, K)** for all **K**

*Reminder: Reducer **(I, J, K)** is responsible for computing the J^{th} partial sum for block P_{IK}*

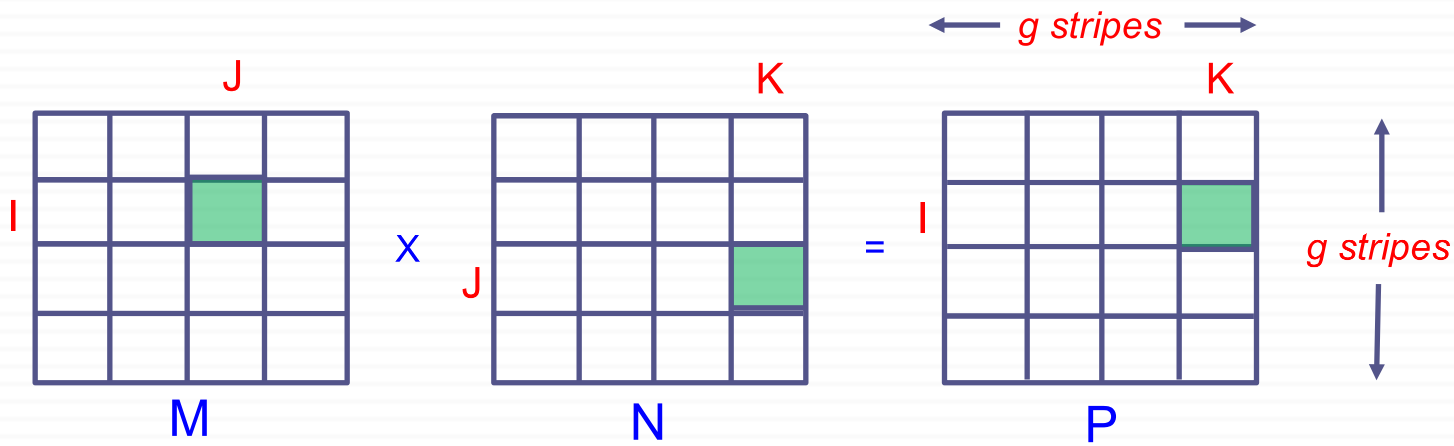
MapReduce Step 1: Map



Block N_{JK} will be sent to the reducers **(I, J, K)** for all **I**

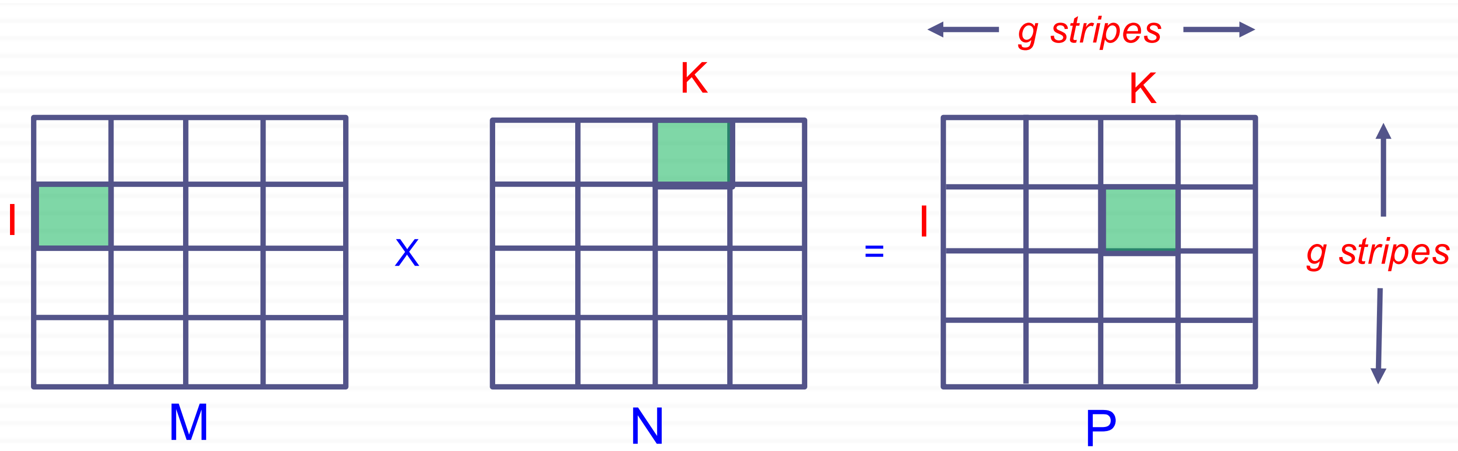
*Reminder: Reducer **(I, J, K)** is responsible for computing the J^{th} partial sum for block P_{IK}*

MapReduce Step 1: Reduce



Reducer ***(I, J, K)*** will receive $\mathbf{M_{IJ}}$ and $\mathbf{N_{JK}}$ blocks and will compute the J^{th} *partial sum for block* P_{IK}

MapReduce Step 1: Reducer Output



$$P_{IK} = \sum_{J=1}^{J=g} M_{IJ} x\, N_{JK}$$

For each **p_{ik}** ∈ **P_{IK}**, there are **g** reducers that compute a partial sum (each with **key**=**(I, J, K)**)

The reduce outputs corresponding to **p_{ik}**: <**key** = **(i, k)**, **value** = **x^J_{ik}**>

MapReduce Step 2

- **Map**:

  for each input <**key** = **(i, k)**, **value** = x^J_{ik}>

  generate <**key** = **(i, k)**, **value** = x^J_{ik}>

- **Reduce**(**key** = **(i, k)**, **value_list**)

  $p_{ik} = 0$

  for each x^J_{ik} in **value_list**

  p_{ik} += x^J_{ik}

  output <**key** = **(i, k)**, **value** = p_{ik}>

Complexity Analysis: Step 1

- **Map**:

  for each m_{ij} in M

  for $K = 1$ to g

  generate <**key** = (I, J, K), **value** = ('M', i, j, m_{ij})>

  for each n_{jk} in N

  for $I = 1$ to g

  generate <**key** = (I, J, K), **value** = ('N', j, k, m_{jk})>

- **Reduce**(**key** = (I, J, K), **value_list**)

  for each $i \in I$ and $k \in K$

  compute $x_{ik}^{J} = \sum_{j \in J} m_{ij} n_{jk}$

  output <**key** = (i, k), **value** = x_{ik}^{J}>

Replication rate:
$r_1 = g$

Communication cost:
$2n^2 + 2gn^2$

Reducer size:
$q_1 = 2n^2/g^2$

# of reducers:
g^3

Complexity Analysis: MapReduce Step 2

- **Map**:

  for each input <**key** = **(i, k)**, **value** = x^J_{ik}>

  generate <**key** = **(i, k)**, **value** = x^J_{ik}>

- **Reduce**(**key** = **(i, k)**, **value_list**)

  $p_{ik} = 0$

  for each x^J_{ik} in **value_list**

  p_{ik} += x^J_{ik}

  output <**key** = **(i, k)**, **value** = p_{ik}>

Replication rate:
$r_2 = 1$

Communication cost:
gn^2

Reducer size:
$q_2 = g$

# of reducers:
n^2

Complexity Analysis

- Total communication cost:

  $2n^2 + 3gn^2$

- Which reducer size is the bottleneck?
  - Typical case: $q_1 \geq q_2$ (when $g^3 \leq 2n^2$)
  - What if this is not the case? (see next slide)

- Communication cost as function of q_1:

$$q_1 = \frac{2n^2}{g^2} \Longrightarrow g = \frac{\sqrt{2}n}{\sqrt{q_1}}$$

$$comm.cost = 2n^2 + \frac{3\sqrt{2}n^3}{\sqrt{q_1}}$$

- Communication cost as function of q_2:

$$comm.cost = 2n^2 + 3n^2 q_2$$

Step 1	Step 2
Replication rate: $r_1 = g$	Replication rate: $r_2 = 1$
Communication cost: $2n^2 + 2gn^2$	Communication cost: gn^2
Reducer size: $q_1 = 2n^2/g^2$	Reducer size: $q_2 = g$
# of reducers: g^3	# of reducers: n^2

Tradeoff Between Communication Cost and Reducer Size

- To decrease communication cost:
  - Choose **g** small enough
- To decrease reducer size:
  - Choose **g** large enough to reduce $\mathbf{q_1}$
  - Size of $\mathbf{q_2}$ is less of a concern. Why?
    - The reduce operation in step 2:
      - Simply accumulate the values
      - The same value is used only once
      - The value_list doesn't have to fit into local memory

$$q_1 = \frac{2n^2}{g^2} \qquad q_2 = g$$

$$comm.cost = 2n^2 + 3gn^2$$

$$comm.cost = 2n^2 + \frac{3\sqrt{2}n^3}{\sqrt{q_1}}$$

$$comm.cost = 2n^2 + 3n^2 q_2$$

- Conclusion: Use the communication cost formula as a function of $\mathbf{q_1}$ to determine the right tradeoff.

Matrix-Matrix Multiplication
1D Decomposition vs. 2D Decomposition

Comparison: Parallelism

1D Decomposition

# of reducers = g_{1D}^2

2D Decomposition

# of reducers = g_{2D}^3 *(step 1)*

n^2 *(step 2)*

For the same # of groups, 2D decomposition has better parallelism

Comparison: Reducer Size

1D Decomposition

$$q_{1D} = \frac{2n^2}{g_{1D}}$$

2D Decomposition

$$q_{2D} = \frac{2n^2}{g_{2D}^2}$$

For the same reducer size:

We need a larger g value for 2D decomposition

$$g_{1D} = g_{2D}^2$$

However, larger ***g*** leads to better parallelism:

\# of reducers for 1D: $g_{1D}^2 = g_{2D}^4$

\# of reducers for 2D: g_{2D}^3 **(step 1)**

n^2 **(step 2)**

Comparison: Communication Costs

1D Decomposition

$$cost_{1D} = 2n^2 + 2n^2 g_{1D}$$

2D Decomposition

$$cost_{2D} = 2n^2 + 3n^2 g_{2D}$$

If the g values are the same:

1D decomposition has lower communication cost

Why would we want to have $g_{1D} = g_{2D}$?

No reason…

More realistically, if the reducer sizes are equal:

$g_{1D} = g_{2D}^2$ (previous slide)

$cost_{1D} = 2n^2 + 2n^2 g_{2D}^2$

$cost_{2D} = 2n^2 + 3n^2 g_{2D}$

Note: We have control over how to choose the g values for 1D and 2D decompositions. However, the max q value is limited by the available local memory size. So, it makes more sense to use the same q value for 1D and 2D decompositions.

Comparison: Communication Costs (when reducer sizes are equal)

1D Decomposition

$$cost_{1D} = 2n^2 + 2n^2 g_{1D}$$

2D Decomposition

$$cost_{2D} = 2n^2 + 3n^2 g_{2D}$$

$$g_{1D} = g_{2D}^2$$

- When does 1D decomposition have less communication cost?

  Only when $g_{1D} = g_{2D} = 1$ (i.e. the serial reduce execution)

- Compare the communication costs for the largest g_{1D} value

  $g_{1D} = n$ and $g_{2D} = \sqrt{n}$

  $cost_{1D} = 2n^2 + 2n^3$

  $cost_{2D} = 2n^2 + 3n^2\sqrt{n}$

  For large # of groups, communication cost of 2D algorithm lower almost by a factor of $\sqrt{n}$

Conclusions

- Complexity analysis:
  - *Replication rate*: Typically determines the communication cost
  - *Reducer size*: Determines the available parallelism and the requirements for local memory sizes
  - Typically tradeoff between communication cost and reducer size
  - We ignored computation costs assuming that the total amount of computation does not change
    - e.g. n^3 multiply-and-add operations for matrix-matrix multiplication
    - However, this is not always the case: There can be parallel implementations that are not work efficient.

- We reduced communication costs by assigning multiple outputs to each reducer. Why?
  - Replication rates reduced (each input needs to be sent to less # of reducers)
  - Grouping may not help algorithms with replication rate = 1
    - e.g. the 2nd step of matrix matrix multiplication with 2D decomposition