

Bulk Synchronous Parallel

What with PRAM?

- PRAM can emulate a message-passing computer by dividing the memory into private memories with each processor
- Several PRAM based papers (fine-grained) algorithmic techniques
 - Results seem irrelevant, **posterior time is away**
 - Performance predictions are **inaccurate**
 - Hasn't lead to programming languages
 - Hardware doesn't have fine-grained synchronous steps

BSP

- The **Bulk Synchronous Parallel (BSP)** abstract computer is a bridging model for designing parallel algorithms.
- It serves a purpose similar to the Parallel Random Access Machine (PRAM) model. It is generalization of PRAM model.
- BSP does not take communication and synchronization for granted.
- An important part of analyzing a BSP algorithm rests on **quantifying** the **synchronization and communication** needed.

BSP-History

- **BSP: Bulk-Synchronous Parallel**

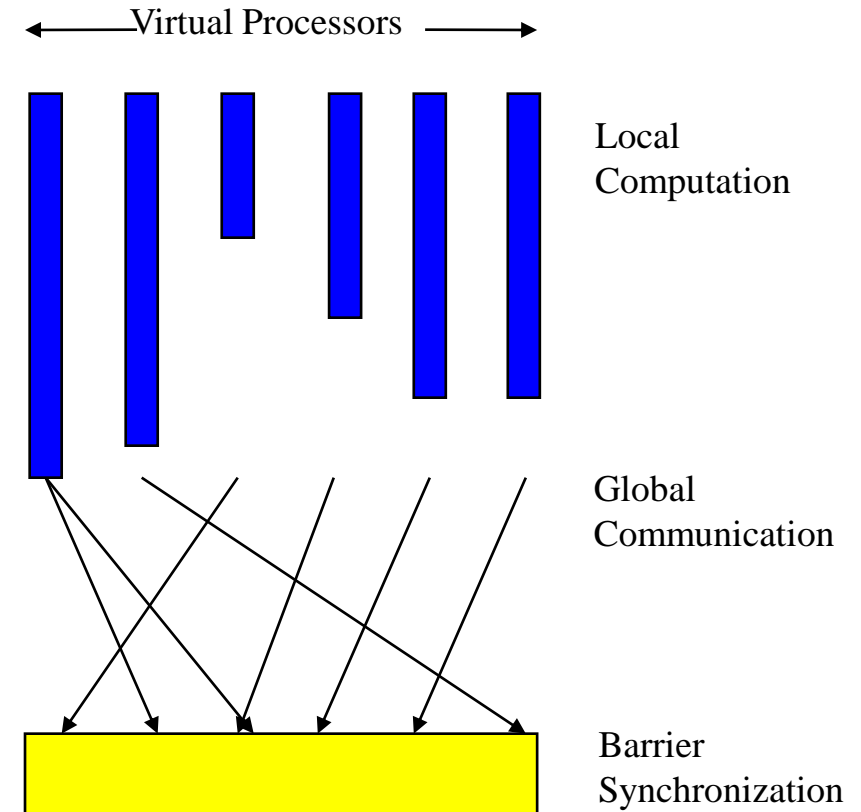
- Valiant, Leslie G., “A Bridging Model for Parallel Computation”, Communications of the ACM, Aug., 1990, Vol. 33, No. 8, pp. 103-111.
- BSP is designed to be **architecture independent**
 - Portable programs
- BSP considers at a global level (*bulk*) computation and communication
- Execution time of a BSP program is computed by the local execution time and from few parameters tied to the particular architecture that is used

BSP

- A BSP computer consists of
 - **Components** capable of processing and/or local memory transactions
 - a **network** that **routes** messages between pairs of such components, and
 - a **hardware facility** that allows for the *synchronization* of all or a subset of components. I.e. **Periodicity parameter L** : to facilitate synchronization at regular intervals of L time units.

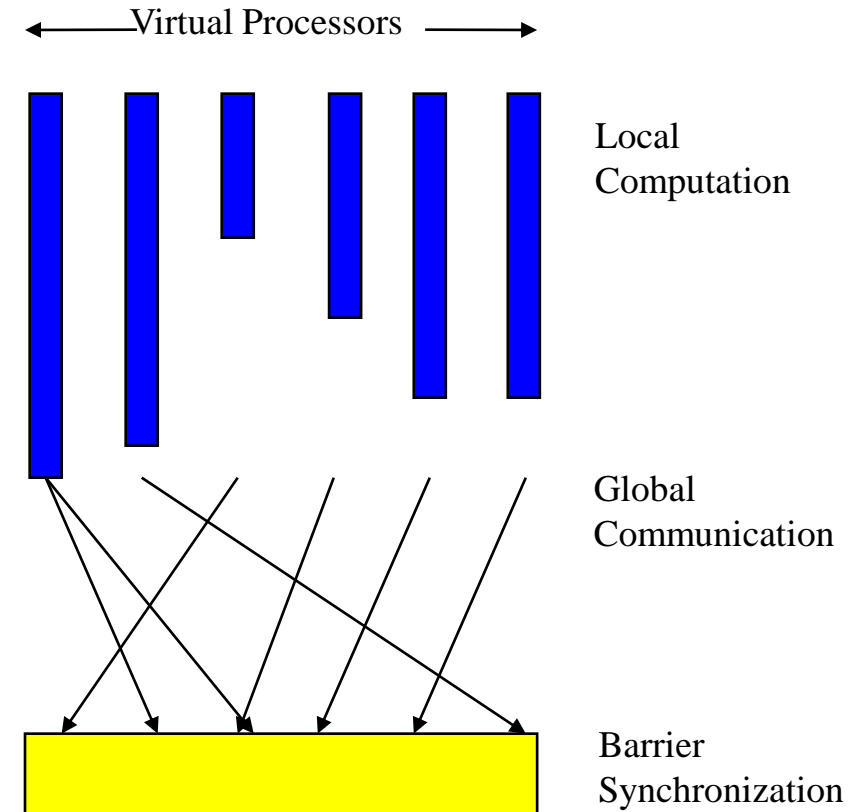
Continued..

- The *components* could be processors
- *The inter-connection network* could be router
- The *periodicity* parameter could be barrier.



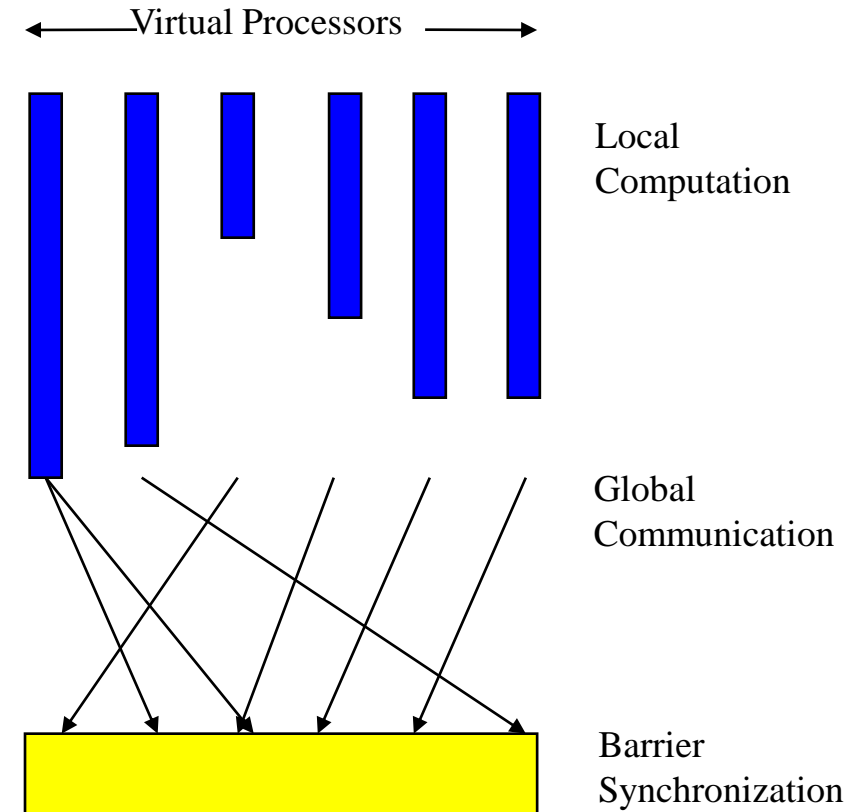
Computation on BSP Model

- A computation consists of *several supersteps*
- A *superstep* consists of:
 - A computation where each processor **uses only locally held values**
 - A global message transmission from each processor to any subset of others
 - A barrier synchronization



Computation on BSP Model

- At the end of a superstep, the transmitted messages become available as local data for the next *superstep*

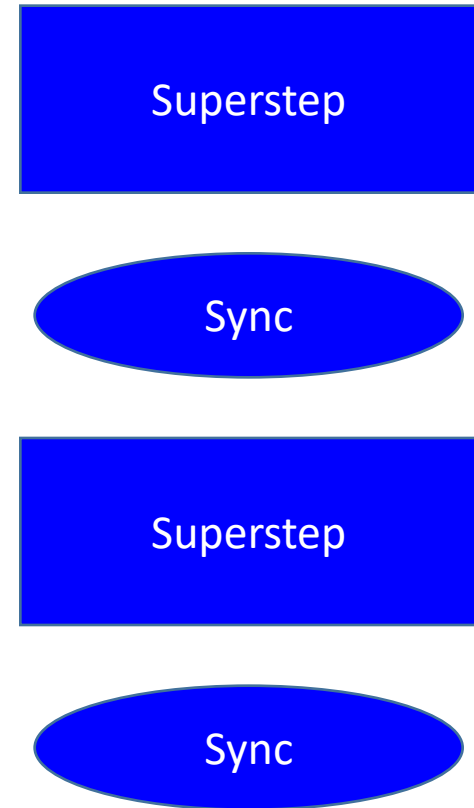


Communication on BSP Model

- A communication is always realized in a point-to point manner
 - Thus it is **not allowed** for multiple processes to read or write the same memory location **in the same cycle**
- All memory and communication operations in a *superstep* must completely finish before any operation of the next *superstep* begins


Communication on BSP Model

- In BSP, each processor has local memory
 - “One-sided”^{*} communication style is advocated
 - There are globally-known “symbolic addresses”
 - Data may be inconsistent until next barrier synchronization



^{}allow a process to access another process address space without any explicit participation in that communication operation by the remote process. One-sided **put** and **get** Direct Remote Memory Access (DRMA) calls, rather than paired two-sided **send** and **receive** message passing calls*

The BSP Model

- 
- Compute → Communicate → Synchronize → repeat
 - The BSP computer is a MIMD system
 - It is **loosely synchronous** at the *superstep* level
 - While the PRAM model was synchronous atwhich level??

The BSP Model

- Compute → Communicate → Synchronize → repeat
- The BSP computer is a MIMD system
- It is **loosely synchronous** at the *superstep* level
 - While the PRAM model was synchronous at **instruction level**
- Within a superstep, different processes execute asynchronously at their own paces

BSP Basics

- A BSP program runs in **supersteps**:
 1. Do local work (computation)
 2. Send/receive messages (communication)
 3. Wait until everyone is done (synchronization)
- The cost of a BSP program = **computation**
+ communication
+ synchronization.

Components (Processors)

- No need for programmers to manage memory, assign communication and perform low-level synchronization.
- This is achieved by programs written with **sufficient parallel slackness**.
- When programs written for **v virtual processors** are **run on p real processors** with $v \gg p$ (e.g. $v = p \log p$) then there is parallel slackness.
- Parallel slackness makes work distribution more balanced (than in cases such as $v = p$ **OR** $v < p$).

The BSP Model – w

- To account for load imbalance, the computation time **w is the maximum time** spent on computation operations by any processor

The BSP Model – h

- The BSP model **abstracts the communication operations** in a BSP superstep by the **h -relation** concept
- An **h -relation** is an abstraction of any communication operation, where each node **sends at most h words** to various nodes and each node **receives at most h words**

The BSP Model – gh

- Parameter g measures the permeability of the network to continuous traffic to uniformly random destinations
 - The parameter g is defined such that an h -relation will be delivered in time gh
 - The communication overhead is gh cycles, where g is the proportional coefficient for realizing an h -relation
- The value of g is platform-dependent, but independent of the communication pattern
 - In other words, gh is the time to execute the most time-consuming h -relation

The BSP Model – mg

- BSP does not distinguish between sending 1 message of length m , or m messages of length 1
 - Cost is mg

The BSP Model – l

- The **synchronization overhead** is l , which has a lower bound of the communication network latency (i.e., the time for a word to propagate through the physical network) and is always greater than zero

Barrier

- “Often expensive and should be used as sparingly as possible”
 - Developers of BSP claim that barriers are not as expensive as they are believed to be in high performance computing community
- The cost of a barrier synchronization **has two parts**
 - The cost caused by the variation in the completion time of the computation steps that participate
 - **The cost of reaching a globally-consistent state in all processors**

Barrier

- The parameter β captures the latter of these costs
 - Lower bound on β is the diameter of the network
 - However, it is also affected by many other factors, so that, in practice, an accurate value of β for each parallel architecture is obtained empirically

The two parts of barrier cost

1. Variation in completion times (load imbalance):

1. If one processor is slower (more work, more messages, slower hardware), others must **wait**.
2. This is “waiting for the slowest processor.”

2. Reaching a globally consistent state:

1. Even if all processors finish at the same time, the system must ensure that:
 1. All **messages** sent in this superstep are delivered to the right processors.
 2. All processors agree that “**everyone is done**” and it’s safe to start the next superstep.
2. This requires **synchronization overhead**: exchanging small control signals, acknowledgments, or using a global clock.
3. In real systems, this is the **latency (/)** part of the BSP cost model.

Parameters (in simple words)

p → number of processors (workers).

w → **work** per processor in one superstep.

- **Example**: how many additions/multiplications each processor does locally.

h → number of **messages** a processor sends or receives in a superstep.

g → **gap per message** = cost of sending one word of data.

- If each message is 100 words long, cost = $100g$.

l → **latency** = time for barrier synchronization (the “global consistency overhead” cost).

- Like waiting for the slowest worker to arrive before moving on.

gh → total communication cost for a processor in a superstep.

- If a worker sends h words, each word costs g , total = gh .

The BSP Model

- h : communication time
- w : computation time
- l : synchronization time
- gh : communication overhead
- The time for a superstep is estimated by the sum
- $????$

The BSP Model

- ***h***: communication time
- ***w***: computation time
- ***l***: synchronization time (2nd part)
- ***gh***: communication overhead
- The time for a superstep is estimated by the sum
 - ***Max_i w_i + Max_i gh_i + l***

The BSP Model

- The BSP model **allows** the overlapping of the computation, the communication, and the synchronization operations within a superstep
 - If all three types of **operations are fully overlapped**, the time for a superstep becomes **$\max(w, gh, l)$**
 - However, the more conservative **$w + gh + l$** is typically used

Example: Maximum of n element

- Algorithm to compute the maximum of a n-elements array. On a BSP, since there is **no shared memory**, we have to say where the data are
 - $A[0..n-1]$ is distributed block-wise across **p** processors
 - For instance, each processor can have a portion of the array
 - n/p elements
- To describe an algorithm on a BSP machine, we have to define all supersteps
 - Local computing operations
 - Communication operations
 - Synchronization barrier

Maximum

- Superstep1

- Local computation phase

- $m = -\infty$;
 - for all $A[i]$ in my local partition of A , $m = \max(m, A[i])$;

- Communication phase

- if $\text{myPID} \neq 0$ send (m , 0);
 - else // on P_0 :
 - for each i in $\{1..p-1\}$ recv (m_i , i);

- Superstep2

- if $\text{myPID} = 0$ for each i in $\{1..p-1\}$ $m = \max(m, m_i)$

Maximum

- Superstep1

- Local computation phase

Time?

- $m = -\infty$;
 - for all $A[i]$ in my local partition of A , $m = \max(m, A[i])$;

- Communication phase

Time?

- if $\text{myPID} \neq 0$ send (m , 0);
 - else
 - for each i in $\{1..p-1\}$ recv (m_i , i);

// on P_0 :

- Superstep2

- if $\text{myPID} = 0$ for each i in $\{1..p-1\}$ $m = \max(m, m_i)$

Time?

Maximum

- Superstep1

- Local computation phase

(n/p)

- $m = -\infty$;
 - for all $A[i]$ in my local partition of A , $m = \max(m, A[i])$;

- Communication phase

Time?

- if myPID $\neq 0$ send (m , 0);
 - else
 - for each i in $\{1..p-1\}$ recv (m_i , i);

// on P_0 :

- Superstep2

- if myPID = 0 for each i in $\{1..p-1\}$ $m = \max(m, m_i)$

Time?

Maximum

- Superstep1

- Local computation phase (n/p)

- $m = -\infty$;
 - for all $A[i]$ in my local partition of A , $m = \max(m, A[i])$;

- Communication phase $(gh, \text{ with } h = p-1)$

- if myPID $\neq 0$ send $(m, 0)$;
 - else // on P_0 :
 - for each i in $\{1..p-1\}$ recv (m_i, i) ;

- Superstep2

- if myPID = 0 for each i in $\{1..p-1\}$ $m = \max(m, m_i)$ Time?

Maximum

- Superstep1

- Local computation phase (n/p)

- $m = -\infty$;
 - for all $A[i]$ in my local partition of A , $m = \max(m, A[i])$;

- Communication phase (gh , with $h=p-1$, P_0 receives $p-1$ messages)

- if myPID $\neq 0$ send (m , 0);
 - else // on P_0 :
 - for each i in $\{1..p-1\}$ recv (m_i , i);

- Superstep2

- if myPID = 0 for each i in $\{1..p-1\}$ $m = \max(m, m_i)$ Time?

Maximum

- Superstep1

- Local computation phase (n/p)

- $m = -\infty$;
 - for all $A[i]$ in my local partition of A , $m = \max(m, A[i])$;

- Communication phase (gh , with $h=p-1$, P_0 receives $p-1$ messages)

- if myPID $\neq 0$ send ($m, 0$);
 - else // on P_0 :
 - for each i in $\{1..p-1\}$ recv (m_i, i);

- Superstep2

- if myPID = 0 for each i in $\{1..p-1\}$ $m = \max(m, m_i)$ p

Maximum

- Total
- $\Theta(n/p + g(p-1) + l + p) = \Theta(n/p + gp + l)$

Example

- Algorithm for inner-product with 8 processors
- Given two arrays **x** and **y**, we want to compute $\sum x_i y_i$
- In a BSP program, it is crucial to define **how data are split among processors**
 - For instance, in this example, **how the vectors' elements can be divided?**

Example

- Algorithm for inner-product with 8 processors
- Given two arrays **x** and **y**, we want to compute $\sum x_i y_i$
- In a BSP program, it is crucial to define **how data are split among processors**
 - For instance, in this example, the vectors' elements can be **divided cyclically or in blocks**

	0	1	2	3	4	5	6	7	8	9
Cyclic:	P0	P1	P2	P3	P0	P1	P2	P3	P0	P1
Block:	P0	P0	P0	P1	P1	P1	P2	P2	P2	P3

- In any case, it is better having both x_i and y_i on the same processor!

Example

- Algorithm for inner-product using 8-processor BSP computer in 4 supersteps (“small” communication):
- Superstep 1
 - Computation?
 - Communication?
 - Barrier synchronization

Example

- Algorithm for inner-product using 8-processor BSP computer in 4 supersteps (“small” communication):
- Superstep 1
 - **Computation:** Each processor computes its local sum in $w = 2N/8$ time (actually $2N-1/8$) (*N multiplications + $N-1$ additions*)
 - **Communication:** Processors 0, 2, 4, 6 send their local sums to processors 1, 3, 5, 7 respectively
 - Apply 1-relation here
 - **Barrier synchronization**

Example

- Superstep 2
 - Computation?
 - Communication?
 - Barrier synchronization

Example

- Superstep 2
 - **Computation:** Processors 1, 3, 5, 7 each perform one addition ($w = 1$)
 - **Communication:** Processors 1 and 5 send their intermediate results to processors 3 and 7 respectively
 - 1-relation is applied here
 - **Barrier synchronization**

Example

- Superstep 3
 - Computation?
 - Communication?
 - Barrier synchronization

Example

- Superstep 3

- Computation: Processors 3 and 7 each perform one addition ($w = 1$)
- Communication: Processor 3 sends its intermediate result to processor 7
 - Apply 1-relation here
- Barrier synchronization

Example

- Superstep 4
 - Computation?
 - Communication?


Example

- Superstep 4
 - Computation: Processor 7 performs one addition ($w=1$) to generate the final sum
 - No more communication or synchronization is needed

Example

- The total execution time (8 processors) is?

Example

- The total execution time is (8 processors) is $2N/8 + 3g + 3l + 3$ cycles
- In general, the execution time is  supersteps on an p -processor BSP
 - **How much is the parallel time on PRAM computer with p processors?**

Example

- The total execution time is (8 processors) is $2N/8 + 3g + 3l + 3$ cycles
- In general, the execution time is $2N/p + (g+l+1)\log p$ cycles on an p -processor BSP
 - **How much is the parallel time on PRAM computer with p processors?**

Example

- The total execution time is (8 processors) is $2N/8 + 3g + 3l + 3$ cycles
- In general, the execution time is $2N/p + (g+l+1)\log p$ cycles on an p -processor BSP
- This is in contrast to the time $2N/p + \log p$ on a PRAM computer
 - The two extra terms, $\log p$ and $l \log p$ correspond to communication and synchronization overheads, respectively

Matrix Multiplication

- We want to multiply two matrices, A and B
 - $A_{(n \times n)} \times B_{(n \times n)} = C_{(n \times n)}$
- The standard algorithm uses $p \leq n^2$ processors
 - If $p = n^2$, then each processor can compute the **value of a single element** in C

Matrix Multiplication

- **Each element** of C can be computed in parallel using n processors on a CREW PRAM
 - $O(\log n)$ parallel time
 - Basically, it's a SUM in parallel
- **All** c_{ij} can be computed in parallel using n^3 processors in $O(\log n)$ time

Matrix Multiplication

- In the BSP model we need to find a way of dividing the input among processors, and to optimize the communication
- Since we have only p processors every processor gets n^2/p elements.
- To each processor we assign the sub-problem of computing a sub-matrix of C , of size $n/\sqrt{p} \times n/\sqrt{p}$
 - Each processor computes $n/\sqrt{p} \times n/\sqrt{p} = n^2/p$ elements of C
- Thus, each processor receives in input n/\sqrt{p} rows of A and n/\sqrt{p} columns of B

Matrix Multiplication

Let $n = 4$

$$A = \begin{array}{|c|c|c|c|} \hline a_{11} & a_{12} & a_{13} & a_{14} \\ \hline a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ \hline a_{41} & a_{42} & a_{43} & a_{44} \\ \hline \end{array}$$

$$B = \begin{array}{|c|c|c|c|} \hline b_{11} & b_{12} & b_{13} & b_{14} \\ \hline b_{21} & b_{22} & b_{23} & b_{24} \\ \hline b_{31} & b_{32} & b_{33} & b_{34} \\ \hline b_{41} & b_{42} & b_{43} & b_{44} \\ \hline \end{array}$$

$$C = A \times B =$$

$$\begin{array}{|c|c|c|c|} \hline c_{11} & c_{12} & c_{13} & c_{14} \\ \hline c_{21} & c_{22} & c_{23} & c_{24} \\ \hline c_{31} & c_{32} & c_{33} & c_{34} \\ \hline c_{41} & c_{42} & c_{43} & c_{44} \\ \hline \end{array}$$

Matrix Multiplication

Let $p = 4$ (p_1, p_2, p_3, p_4)

p_1 computes

c_{11}	c_{12}
c_{21}	c_{22}

p_2 computes

c_{13}	c_{14}
c_{23}	c_{24}

p_3 computes

c_{31}	c_{32}
c_{41}	c_{42}

p_4 computes

c_{33}	c_{34}
c_{43}	c_{44}

Matrix Multiplication

p_1 computes

c_{11}	c_{12}
c_{21}	c_{22}

with input

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}

b_{11}	b_{12}
b_{21}	b_{22}
b_{31}	b_{32}
b_{41}	b_{42}

p_2 computes

c_{13}	c_{14}
c_{23}	c_{24}

with input

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}

b_{13}	b_{14}
b_{23}	b_{24}
b_{33}	b_{34}
b_{43}	b_{44}

Matrix Multiplication

p_3 computes

c_{31}	c_{32}
c_{41}	c_{42}

with input

a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

b_{11}	b_{12}
b_{21}	b_{22}
b_{31}	b_{32}
b_{41}	b_{42}

p_4 computes

c_{33}	c_{34}
c_{43}	c_{44}

with input

a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

b_{13}	b_{14}
b_{23}	b_{24}
b_{33}	b_{34}
b_{43}	b_{44}

Matrix Multiplication

- Let us compute the number of local operations performed by a processor, say p_4
 - Given a local row and a local column of p_4
 - » How many sums does it perform?
 - » How many multiplications does it perform?

p_4 computes

c_{33}	c_{34}
c_{43}	c_{44}

with input

a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

b_{13}	b_{14}
b_{23}	b_{24}
b_{33}	b_{34}
b_{43}	b_{44}

Matrix Multiplication

- Let us compute the number of local operations performed by a processor, say p_4
 - Given **a local row and a local column** of p_4
 - »How many sums does it perform **$n-1$**
 - »How many multiplications does it perform **n**

p_4 computes

c_{33}	c_{34}
c_{43}	c_{44}

with input

a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

b_{13}	b_{14}
b_{23}	b_{24}
b_{33}	b_{34}
b_{43}	b_{44}

Matrix Multiplication

- Let us compute the number of local operations performed by a processor, say p_4
 - How many row-by-column inner products p_4 does perform locally?

p_4 computes

c_{33}	c_{34}
c_{43}	c_{44}

with input

a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

b_{13}	b_{14}
b_{23}	b_{24}
b_{33}	b_{34}
b_{43}	b_{44}

Matrix Multiplication

- Let us compute the number of local operations performed by a processor, say p_4
 - Summing **over all inner products** performed by p_4
 - » How many sums does it perform?
 - » How many multiplications does it perform?

p_4 computes

c_{33}	c_{34}
c_{43}	c_{44}

with input

a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

b_{13}	b_{14}
b_{23}	b_{24}
b_{33}	b_{34}
b_{43}	b_{44}

Matrix Multiplication

- Let us compute the number of local operations performed by a processor, say p_4
 - Summing **over all inner products** performed by p_4
 - » How many sums does it perform **$(n-1) \times n/\sqrt{p} \times n/\sqrt{p} = (n-1)n^2/p$**
 - » How many multiplications does it perform **$n \times n/\sqrt{p} \times n/\sqrt{p} = n^3/p$**

p_4 computes

c_{33}	c_{34}
c_{43}	c_{44}

with input

a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

b_{13}	b_{14}
b_{23}	b_{24}
b_{33}	b_{34}
b_{43}	b_{44}

Matrix Multiplication

- Thus, each processor executes locally $(n-1)n^2/p$ sums + n^3/p multiplications
- That is, $(2n-1)n^2/p$ operations

Matrix Multiplication

- Now, let us analyze the complexity of the communication phase
 - In order to execute its local operations, how many messages does each processor needs to receive?

p_4 computes

c_{33}	c_{34}
c_{43}	c_{44}

with input

a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

b_{13}	b_{14}
b_{23}	b_{24}
b_{33}	b_{34}
b_{43}	b_{44}

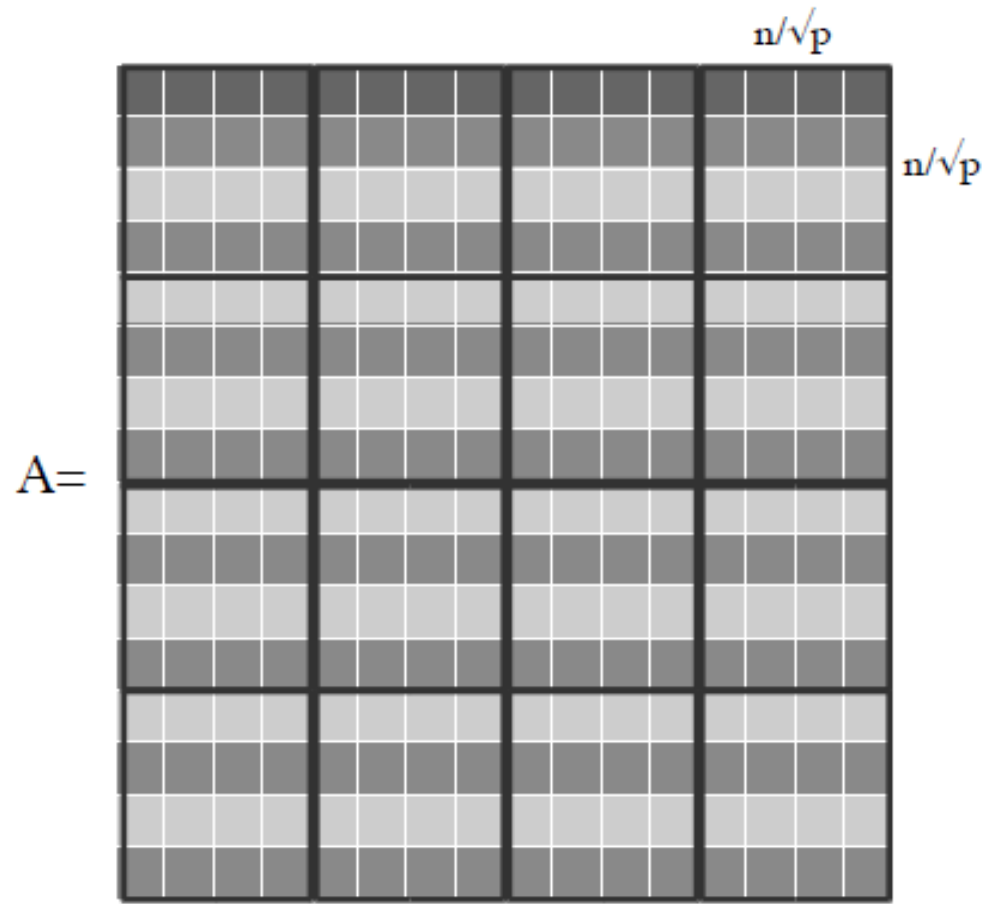
Matrix Multiplication

- How many of its local elements each processor needs to send, so that the other processors can receive the elements they need?
- For instance, to which processor p_2 has to send the elements of A it locally has?
 - To p_1

$A =$

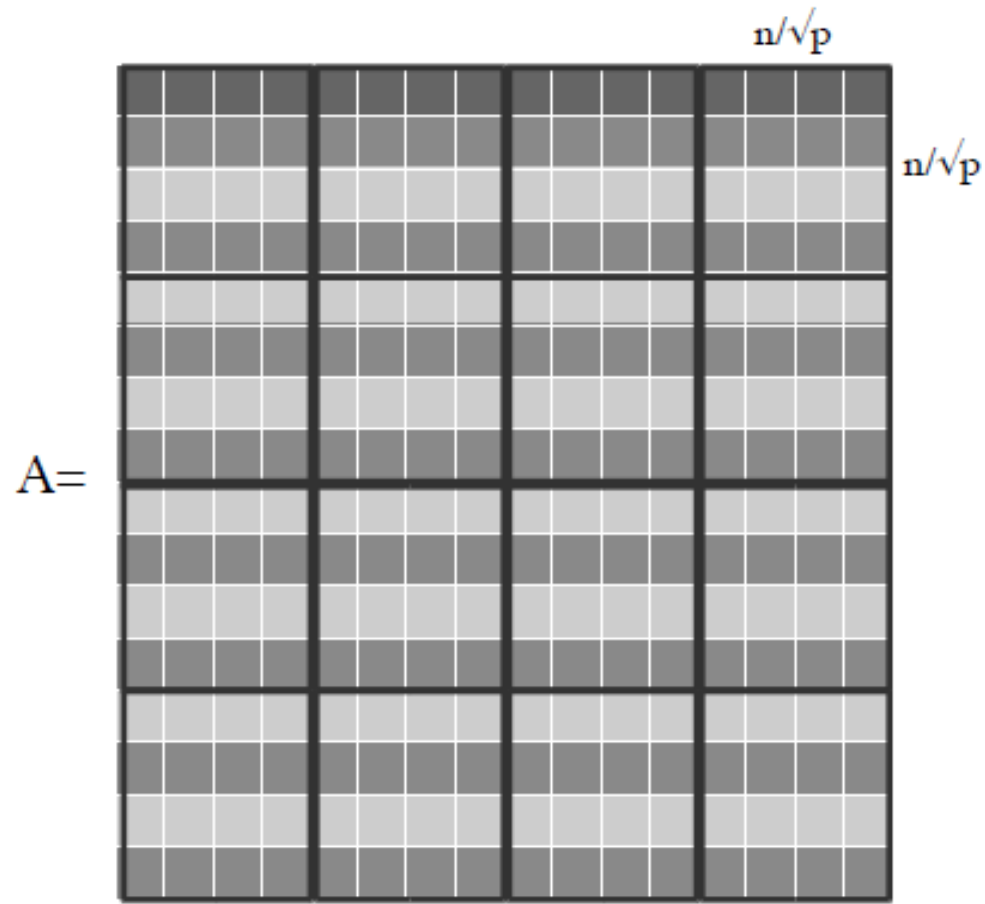
a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

Matrix Multiplication



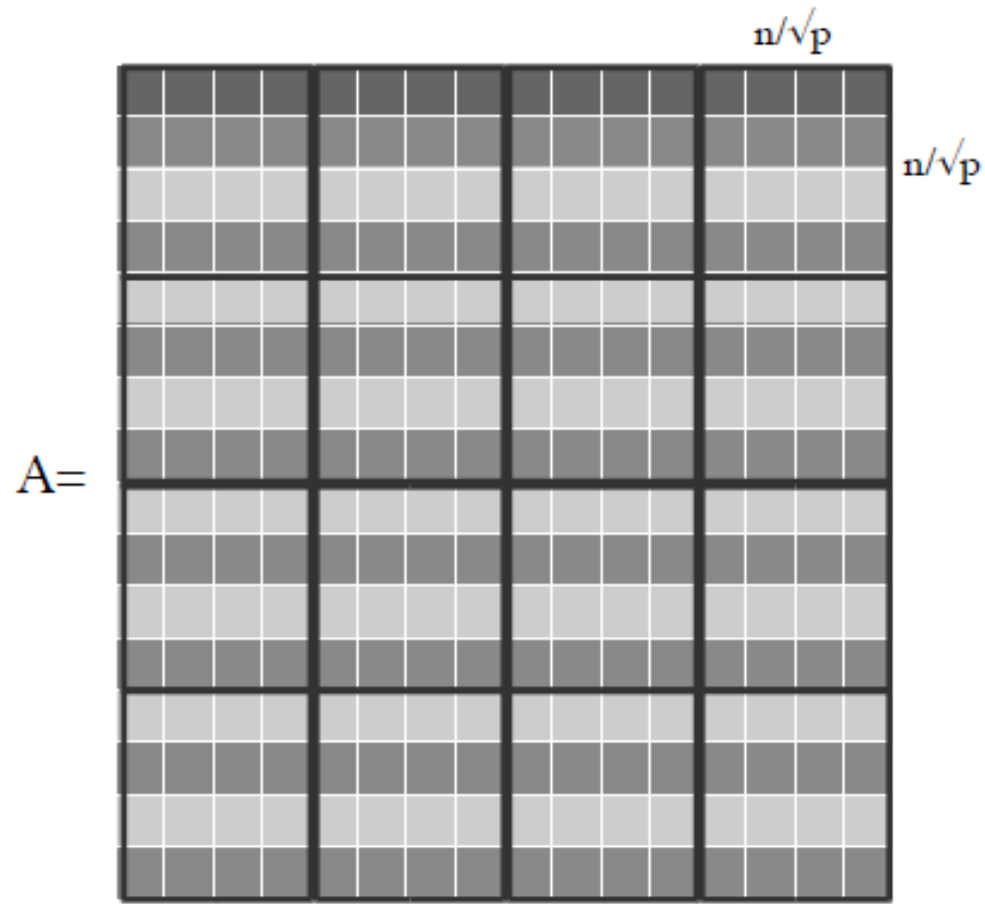
- In general, each processor has to send each one of its local values to how many processors?

Matrix Multiplication



- In general, each processor has to send each one of its local values **to how many processors?**
- **\sqrt{p} (at most)**
- So, how many messages will send each processor in total?

Matrix Multiplication



- In general, each processor has to send each one of its local values to how many processors?
- **\sqrt{p} (at most)**
- So, how many messages will send each processor in total?
- **$(2n^2/p) \times \sqrt{p}$ (From A and B)**

Matrix Multiplication

- Clearly, we cannot expect to have the elements spread over the processors exactly as we need!!
- Thus, we can assume that the elements of A and B are uniformly distributed among processors
 - $2n^2/p$ for each processor
- Each processor *replicates locally* each one of its elements at most **\sqrt{p}** times

Matrix Multiplication

- Finally, \sqrt{p} processors send the appropriated replicated elements to the processors that need them
- Thus, the number of transmissions, for each processor, is this number of messages: $(2n^2/p) \times \sqrt{p} = 2n^2/\sqrt{p}$

Matrix Multiplication

- The cost of this BSP algorithm is
 - $(2n-1)n^2/p + (2n^2/p^{1/2})g + l$
- The optimal cost $O(n^3/p)$, with n^2/p memory for each processor, is achieved when
 - $g = O(n/p^{1/2})$
 - $l = O(n^3/p)$

Example

- let's take your BSP model matrix multiplication example with **n = 8** (matrix size 8×8) and **p = 4** (4 processors).

A

a11	a12	a13	a14	a15	a16	a17	a18
a21	a22	a23	a24	a25	a26	a27	a28
a31	a32	a33	a34	a35	a36	a37	a38
a41	a42	a43	a44	a45	a46	a47	a48
a51	a52	a53	a54	a55	a56	a57	a58
a61	a62	a63	a64	a65	a66	a67	a68
a71	a72	a73	a74	a75	a76	a77	a78
a81	a82	a83	a84	a85	a86	a87	a88

B

b11	b12	b13	b14	b15	b16	b17	b18
b21	b22	b23	b24	b25	b26	b27	b28
b31	b32	b33	b34	b35	b36	b37	b38
b41	b42	b43	b44	b45	b46	b47	b48
b51	b52	b53	b54	b55	b56	b57	b58
b61	b62	b63	b64	b65	b66	b67	b68
b71	b72	b73	b74	b75	b76	b77	b78
b81	b82	b83	b84	b85	b86	b87	b88

Step 1: Divide the work

- We have a matrix $C=A \times B$, size **8×8** , **$p = 4$ processors**, so each gets $n^2/p = 64/4 = 16$ elements of C .
- Each processor computes a sub-matrix of size $(n/\sqrt{p}) \times (n/\sqrt{p}) = (8/2) \times (8/2) = 4 \times 4$. So each processor works on a **4×4 block of C** .

P1	a11	a12	a13	a14	a15	a16	a17	a18
	a21	a22	a23	a24	a25	a26	a27	a28
	a31	a32	a33	a34	a35	a36	a37	a38
	a41	a42	a43	a44	a45	a46	a47	a48
P3	a51	a52	a53	a54	a55	a56	a57	a58
	a61	a62	a63	a64	a65	a66	a67	a68
	a71	a72	a73	a74	a75	a76	a77	a78
	a81	a82	a83	a84	a85	a86	a87	a88

P2	b11	b12	b13	b14	b15	b16	b17	b18
	b21	b22	b23	b24	b25	b26	b27	b28
	b31	b32	b33	b34	b35	b36	b37	b38
	b41	b42	b43	b44	b45	b46	b47	b48
P4	b51	b52	b53	b54	b55	b56	b57	b58
	b61	b62	b63	b64	b65	b66	b67	b68
	b71	b72	b73	b74	b75	b76	b77	b78
	b81	b82	b83	b84	b85	b86	b87	b88

Step 2: Input required

- To compute its 4×4 block of C , each processor needs:
 - 4 rows of A , 4 columns of B
- Processor **P1** (top-left block of C) needs: rows 1–4 of A and columns 1–4 of B .
- Processor **P2** (top-right block of C) needs: rows 1–4 of A and columns 5–8 of B .
- Processor **P3** (bottom-left block of C) needs: rows 5–8 of A and columns 1–4 of B .
- Processor **P4** (bottom-right block of C) needs: rows 5–8 of A and columns 5–8 of B .

P1	a11	a12	a13	a14	a15	a16	a17	a18
	a21	a22	a23	a24	a25	a26	a27	a28
	a31	a32	a33	a34	a35	a36	a37	a38
	a41	a42	a43	a44	a45	a46	a47	a48
P3	a51	a52	a53	a54	a55	a56	a57	a58
	a61	a62	a63	a64	a65	a66	a67	a68
	a71	a72	a73	a74	a75	a76	a77	a78
	a81	a82	a83	a84	a85	a86	a87	a88

P2	b11	b12	b13	b14	b15	b16	b17	b18
	b21	b22	b23	b24	b25	b26	b27	b28
	b31	b32	b33	b34	b35	b36	b37	b38
	b41	b42	b43	b44	b45	b46	b47	b48
P4	b51	b52	b53	b54	b55	b56	b57	b58
	b61	b62	b63	b64	b65	b66	b67	b68
	b71	b72	b73	b74	b75	b76	b77	b78
	b81	b82	b83	b84	b85	b86	b87	b88

Step 2: Input required

- To compute its 4×4 block of C , each processor needs:
 - 4 rows of A , 4 columns of B
- Processor **P1** (top-left block of C) needs: rows 1–4 of A and columns 1–4 of B .
- Processor **P2** (top-right block of C) needs: rows 1–4 of A and columns 5–8 of B .
- Processor **P3** (bottom-left block of C) needs: rows 5–8 of A and columns 1–4 of B .
- Processor **P4** (bottom-right block of C) needs: rows 5–8 of A and columns 5–8 of B .

P1	a11	a12	a13	a14	a15	a16	a17	a18	a11	a12	a13	a14	a15	a16	a17	a18	b11	b12	b13	b14	b15	b16	b17	b18
	a21	a22	a23	a24	a25	a26	a27	a28	a21	a22	a23	a24	a25	a26	a27	a28	b21	b22	b23	b24	b25	b26	b27	b28
	a31	a32	a33	a34	a35	a36	a37	a38	a31	a32	a33	a34	a35	a36	a37	a38	b31	b32	b33	b34	b35	b36	b37	b38
	a41	a42	a43	a44	a45	a46	a47	a48	a41	a42	a43	a44	a45	a46	a47	a48	b41	b42	b43	b44	b45	b46	b47	b48
P3	a51	a52	a53	a54	a55	a56	a57	a58	a51	a52	a53	a54	a55	a56	a57	a58	b51	b52	b53	b54	b55	b56	b57	b58
	a61	a62	a63	a64	a65	a66	a67	a68	a61	a62	a63	a64	a65	a66	a67	a68	b61	b62	b63	b64	b65	b66	b67	b68
	a71	a72	a73	a74	a75	a76	a77	a78	a71	a72	a73	a74	a75	a76	a77	a78	b71	b72	b73	b74	b75	b76	b77	b78
	a81	a82	a83	a84	a85	a86	a87	a88	a81	a82	a83	a84	a85	a86	a87	a88	b81	b82	b83	b84	b85	b86	b87	b88
C								A								B								

Step 2: Input required

- To compute its 4×4 block of C , each processor needs:
 - 4 rows of A , 4 columns of B
- Processor **P1** (top-left block of C) needs: rows 1–4 of A and columns 1–4 of B .
- Processor **P2** (top-right block of C) needs: rows 1–4 of A and columns 5–8 of B .
- Processor **P3** (bottom-left block of C) needs: rows 5–8 of A and columns 1–4 of B .
- Processor **P4** (bottom-right block of C) needs: rows 5–8 of A and columns 5–8 of B .

P1	a11	a12	a13	a14	a15	a16	a17	a18	a11	a12	a13	a14	a15	a16	a17	a18	b11	b12	b13	b14	b15	b16	b17	b18
	a21	a22	a23	a24	a25	a26	a27	a28	a21	a22	a23	a24	a25	a26	a27	a28	b21	b22	b23	b24	b25	b26	b27	b28
	a31	a32	a33	a34	a35	a36	a37	a38	a31	a32	a33	a34	a35	a36	a37	a38	b31	b32	b33	b34	b35	b36	b37	b38
	a41	a42	a43	a44	a45	a46	a47	a48	a41	a42	a43	a44	a45	a46	a47	a48	b41	b42	b43	b44	b45	b46	b47	b48
P3	a51	a52	a53	a54	a55	a56	a57	a58	a51	a52	a53	a54	a55	a56	a57	a58	b51	b52	b53	b54	b55	b56	b57	b58
	a61	a62	a63	a64	a65	a66	a67	a68	a61	a62	a63	a64	a65	a66	a67	a68	b61	b62	b63	b64	b65	b66	b67	b68
	a71	a72	a73	a74	a75	a76	a77	a78	a71	a72	a73	a74	a75	a76	a77	a78	b71	b72	b73	b74	b75	b76	b77	b78
	a81	a82	a83	a84	a85	a86	a87	a88	a81	a82	a83	a84	a85	a86	a87	a88	b81	b82	b83	b84	b85	b86	b87	b88
C								A								B								

Step 2: Input required

- To compute its 4×4 block of C , each processor needs:
 - 4 rows of A , 4 columns of B
- Processor **P1** (top-left block of C) needs: rows 1–4 of A and columns 1–4 of B .
- Processor **P2** (top-right block of C) needs: rows 1–4 of A and columns 5–8 of B .
- Processor **P3** (bottom-left block of C) needs: rows 5–8 of A and columns 1–4 of B .
- Processor **P4** (bottom-right block of C) needs: rows 5–8 of A and columns 5–8 of B .

P1	a11	a12	a13	a14	a15	a16	a17	a18	a11	a12	a13	a14	a15	a16	a17	a18	b11	b12	b13	b14	b15	b16	b17	b18
	a21	a22	a23	a24	a25	a26	a27	a28	a21	a22	a23	a24	a25	a26	a27	a28	b21	b22	b23	b24	b25	b26	b27	b28
	a31	a32	a33	a34	a35	a36	a37	a38	a31	a32	a33	a34	a35	a36	a37	a38	b31	b32	b33	b34	b35	b36	b37	b38
	a41	a42	a43	a44	a45	a46	a47	a48	a41	a42	a43	a44	a45	a46	a47	a48	b41	b42	b43	b44	b45	b46	b47	b48
P3	a51	a52	a53	a54	a55	a56	a57	a58	a51	a52	a53	a54	a55	a56	a57	a58	b51	b52	b53	b54	b55	b56	b57	b58
	a61	a62	a63	a64	a65	a66	a67	a68	a61	a62	a63	a64	a65	a66	a67	a68	b61	b62	b63	b64	b65	b66	b67	b68
	a71	a72	a73	a74	a75	a76	a77	a78	a71	a72	a73	a74	a75	a76	a77	a78	b71	b72	b73	b74	b75	b76	b77	b78
	a81	a82	a83	a84	a85	a86	a87	a88	a81	a82	a83	a84	a85	a86	a87	a88	b81	b82	b83	b84	b85	b86	b87	b88
C								A								B								

Step 2: Input required

- To compute its 4×4 block of C , each processor needs:
 - 4 rows of A , 4 columns of B
- Processor **P1** (top-left block of C) needs: rows 1–4 of A and columns 1–4 of B .
- Processor **P2** (top-right block of C) needs: rows 1–4 of A and columns 5–8 of B .
- Processor **P3** (bottom-left block of C) needs: rows 5–8 of A and columns 1–4 of B .
- Processor **P4** (bottom-right block of C) needs: rows 5–8 of A and columns 5–8 of B .

P1	a11	a12	a13	a14	a15	a16	a17	a18	a11	a12	a13	a14	a15	a16	a17	a18	b11	b12	b13	b14	b15	b16	b17	b18
	a21	a22	a23	a24	a25	a26	a27	a28	a21	a22	a23	a24	a25	a26	a27	a28	b21	b22	b23	b24	b25	b26	b27	b28
	a31	a32	a33	a34	a35	a36	a37	a38	a31	a32	a33	a34	a35	a36	a37	a38	b31	b32	b33	b34	b35	b36	b37	b38
	a41	a42	a43	a44	a45	a46	a47	a48	a41	a42	a43	a44	a45	a46	a47	a48	b41	b42	b43	b44	b45	b46	b47	b48
P3	a51	a52	a53	a54	a55	a56	a57	a58	a51	a52	a53	a54	a55	a56	a57	a58	b51	b52	b53	b54	b55	b56	b57	b58
	a61	a62	a63	a64	a65	a66	a67	a68	a61	a62	a63	a64	a65	a66	a67	a68	b61	b62	b63	b64	b65	b66	b67	b68
	a71	a72	a73	a74	a75	a76	a77	a78	a71	a72	a73	a74	a75	a76	a77	a78	b71	b72	b73	b74	b75	b76	b77	b78
	a81	a82	a83	a84	a85	a86	a87	a88	a81	a82	a83	a84	a85	a86	a87	a88	b81	b82	b83	b84	b85	b86	b87	b88
C								A								B								

Step 3: Local computation (per processor)

- Each processor has to compute **16 elements (4×4)**.
- Each element of C is an inner product of 8 terms.
- So, per processor:
 - Multiplications = $n \times 16 = 8 \times 16 = 128$
 - Additions = $(n-1) \times 16 = 7 \times 16 = 112$
- Total = **128 + 112 = 240 operations** per processor.
 - Multiplications = $n^3/p = 8^3/4 = 512/4 = 128$.
 - Additions (sums) = $(n-1)n^2/p = 7 \cdot 16 = 112$

Step 4: Communication

- Each processor does not initially have all the rows/columns it needs.
- So, processors must **share rows of A and columns of B** with others.
- **Who sends to whom?**
 - **P1**: Has rows 1–4 of A. Must share them with **P2** (since P2 also needs rows 1–4).
 - **P3**: Has rows 5–8 of A. Must share them with **P4**.
 - **P1**: Has columns 1–4 of B. Must share them with **P3**.
 - **P2**: Has columns 5–8 of B. Must share them with **P4**.

Step 4: Communication

- Each processor does not initially have all the rows/columns it needs.
- So, processors must **share rows of A and columns of B** with others.
- So:
- Each processor shares with **$\sqrt{p} = 2$ processors at most**.
- Messages per processor = $(2n^2/p) \times \sqrt{p} = (2 \times 64/4) \times 2 = 32 \times 2 = 64$
- 8 sends total; each processor sends exactly two blocks.
- Each processor shares with **$\sqrt{p} = 2$ processors at most**.

Matrix Multiplication

- There exists a more sophisticated algorithm, by McColl and Valiant, that solves the problem with less messages
 - $n^3/p + (n^2/p^{2/3})g + l$
- That is optimal when
 - $g = O(n/p^{1/3})$
 - $l = O(n^3/(p \log n))$