

Parallel computing models and their performances

A high level exploration of the
HPC world

George Bosilca
bosilca@icl.utk.edu



THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

Overview

- Definition of parallel application
- Architectures taxonomy
- What is quantifiable ? Laws managing the parallel applications field
- Modeling performance of parallel applications

Formal definition of parallelism

The Bernstein Conditions Let's define:

- $I(P)$ all variables, registers and memory locations used by P
- $O(P)$ all variables, registers and memory locations written by P

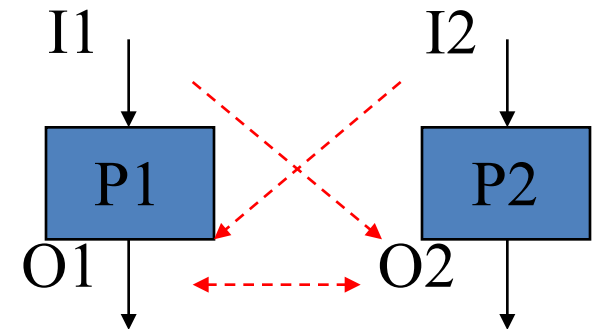
Then $P1; P2$ is equivalent to $P1 \parallel P2$ if and only if

$$\{I(P1) \cap O(P2) = \emptyset \ \& \ I(P2) \cap O(P1) = \emptyset \ \& \ O(P1) \cap O(P2) = \emptyset\}$$

General case: $P1 \dots Pn$ are parallel if and only if
each for each pair Pi, Pj we have $Pi \parallel Pj$.

3 limit to the parallel applications:

1. **Data** dependencies
2. **Flow** dependencies
3. **Resources** dependencies

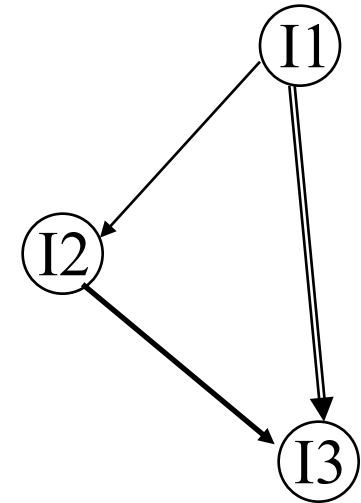
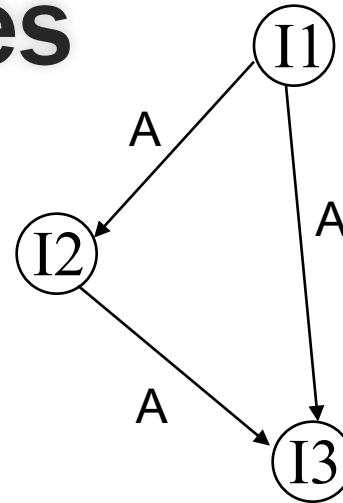


Data dependencies

I1: $A = B + C$

I2: $E = D + A$

I3: $A = F + G$



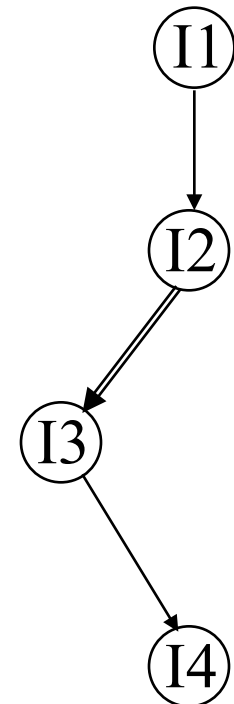
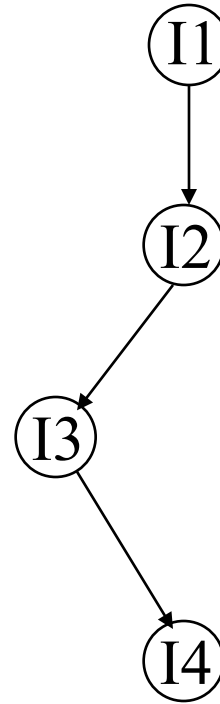
- **Flow dependency (RAW)**: a variable assigned in a statement is used in a later statement
- **Anti-dependency (WAR)**: a variable used in a statement is assigned in a subsequent statement
- **Output dependency (WAW)**: a variable assigned in a statement is subsequently re-assigned

How to avoid them?

Which type of data dependency can be avoided ?

Flow dependencies

I1: $A = B + C$
I2: $\text{if}(A) \{$
I3: $D = E + F \}$
I4: $G = D + H$



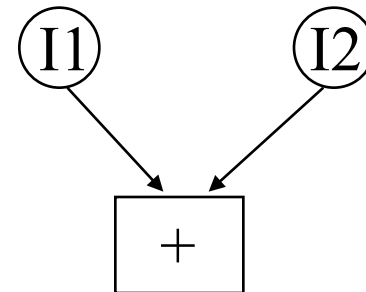
- Data dependency
- = Control dependency

How to avoid ?

Resources dependencies

$$I1: A = B + C$$

$$I2: G = D + H$$



How to avoid ?

A more complicated example (loop)

for i = 0 to 9
 A[i] = B[i]

All statements are **independent**, as they relate to different data. They are concurrent.

for i = 1 to 9
 A[i] = A[i-1]

A[1] = A[0]
A[2] = A[1]
A[3] = A[2]
A[4] = A[3]
A[5] = A[4]
A[6] = A[5]
A[7] = A[6]
A[8] = A[7]
A[9] = A[8]

All statements are **dependent**, as every 2 statements are strictly sequential.

Flynn Taxonomy (1966)

- Computers classified by instruction delivery mechanism and data stream(s)
 - I for instruction, P for program. Conceptually similar, technically at a different granularity.
- 4 characters code: 2 for instruction stream and 2 for data stream

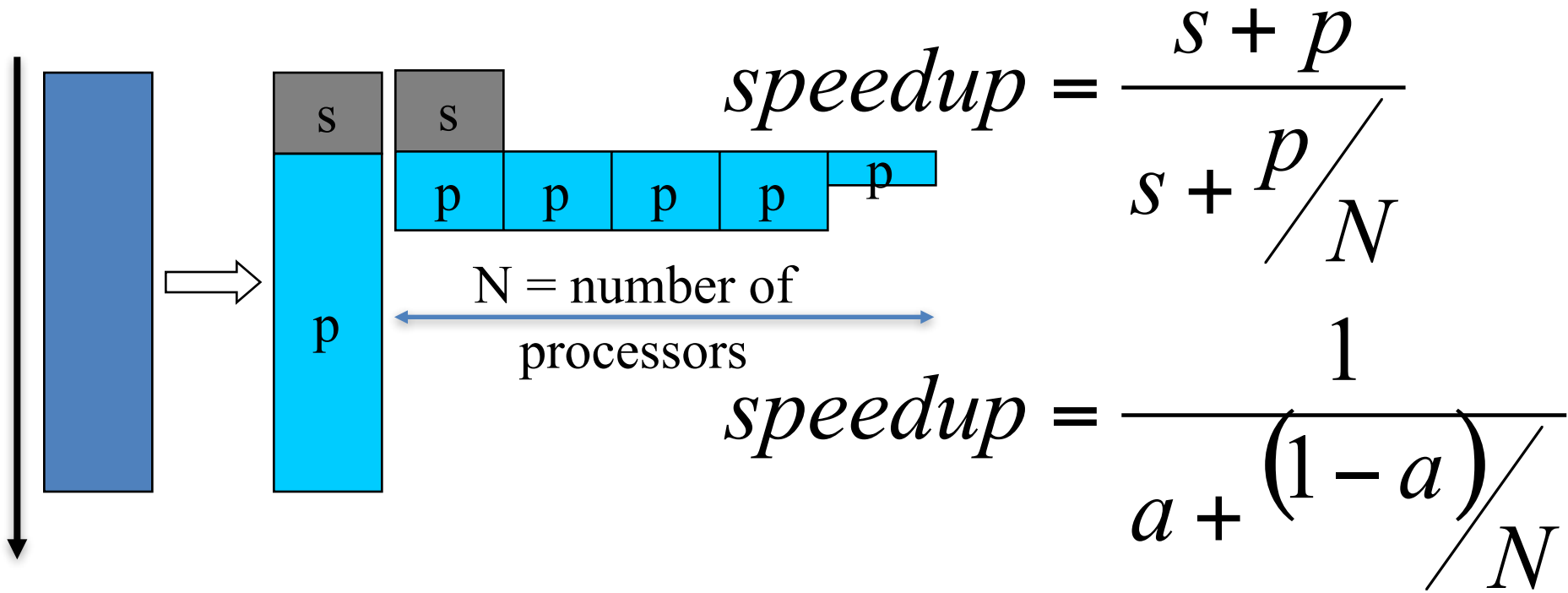
	1 Instruction flow	> 1 Instruction flow
1 data stream	SISD Von Neumann	MISD pipeline
> 1 data stream	SIMD	MIMD

Flynn Taxonomy: Analogy

- SISD: assembly line work (no parallelism)
- SIMD: systolic, GPU computing (vector computing MMX, SSE, AVX)
- MISD: more unusual type. Safety requirements, replication capabilities, think space shuttle.
- MIMD: airport facility, several desks working at their own pace, synchronizing via a central entity (database). Most distributed algorithms, as well as multi-core applications.

Amdahl Law

- First law of parallel applications (1967)
- Limit the speedup for all parallel applications



Amdahl Law

Speedup is bound by $1/a$.

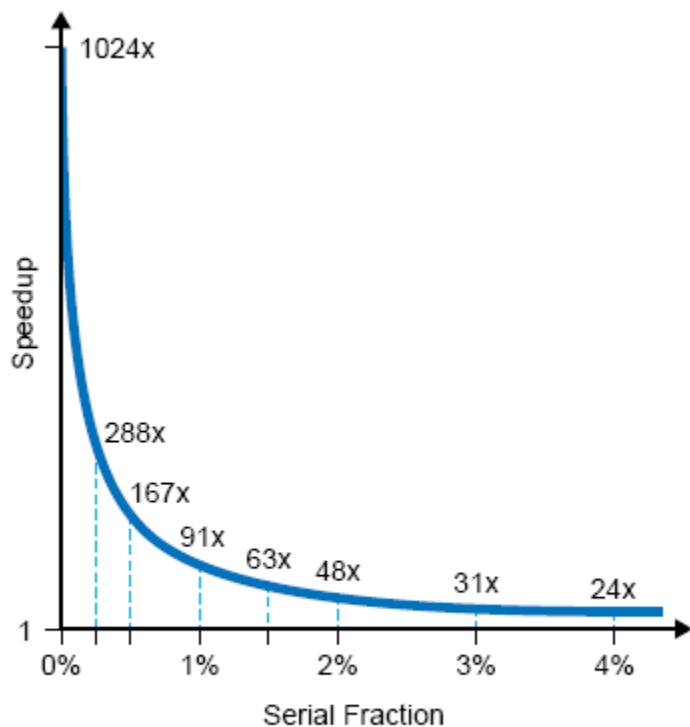


FIGURE 1. Speedup under Amdahl's Law

Amdahl Law

- Bad news for parallel applications
- 2 interesting facts:
 - We should limit the sequential part
 - A parallel computer should be a fast sequential computer to be able to resolve the sequential part quickly
- What about increasing the size of the initial problem ?

Gustafson's Law

- Less constraints than the Amdahl law.
- In a parallel program the quantity of data to be processed increase, so the sequential part decrease.

$$\left. \begin{array}{l} t = s + P / n \\ P = a * n \end{array} \right\} \text{speedup} = \frac{s + a * n}{s + a}$$

$$a \rightarrow \infty \Rightarrow \text{speedup} \rightarrow n$$

Gustafson's Law

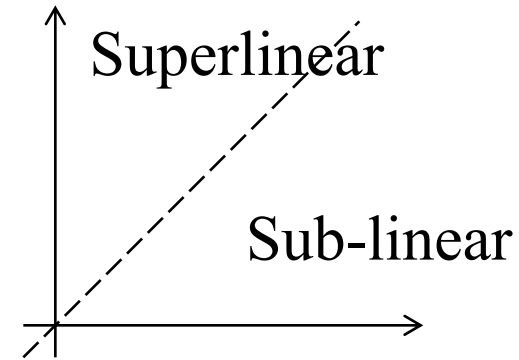
- The limit of Amdahl Law can be transgressed **if** the quantity of data to be processed increase.

$$speedup \leq n + (1 - n)s$$

Rule stating that if the size of most problems is scaled up sufficiently, then any required efficiency can be achieved on any number of processors.

Speedup

- Superlinear speedup ?



Sometimes superlinear speedups can be observed!

- Memory/cache effects
 - More processors typically also provide more memory/cache.
 - Total computation time decreases due to more page/cache hits.
- Search anomalies
 - Parallel search algorithms.
 - Decomposition of search range and/or multiple search strategies.
 - One task may be "lucky" to find result early.

Parallel execution models

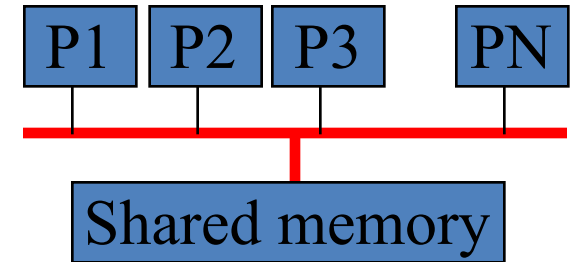
- **Amdahl** and **Gustafson** laws define the limits without taking in account the properties of the computer architecture
- They can only loosely be used to predict (in fact mainly to cap) the real performance of any parallel application
- We should integrate in the same model the architecture of the computer and the architecture of the application

What are models good for ?

- Abstracting the computer properties
 - Making programming simple
 - Making programs portable ?
- Reflecting essential properties
 - Functionality
 - Costs
- What is the von-Neumann model for parallel architectures ?

Parallel Random Access Machine

- World described as a collection of **synchronous processors** which communicate with a **global shared memory unit**.
 - A collection of numbered RAM processors (P_i)
 - A collection of indexed memory cells ($M[i]$)
 - Each processor P_i has its own unbounded local memory (registers) and knows its index (rank)
 - Each processor can access any shared memory cell in unit time
 - Input and output of a PRAM algorithm consist in N distinct items
 - A PRAM instruction consist in 3 synchronous steps: **read** (acquire the input data), **computation**, **write** (save the data back to a shared memory cell).
 - Exchanging data is realized through the writing and reading of memory cells



Parallel Random Access Machine

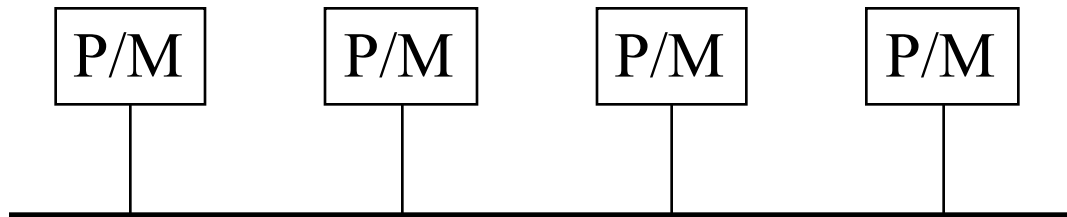
- Algorithmic Complexity:
 - Time = the time elapsed for P_0 computations
 - Space = the number of memory cells accessed
- Specialized in parallel algorithms
 - **Natural**: the number of operations per cycle on N processors is at most N
 - **Strong**: all accesses are realized in a single time unit
 - **Simple**: keep the complexity and correctness overheads low by abstracting all communication or synchronization overheads

The PRAM corresponds intuitively to the programmers' view of a parallel computer: it **ignores** lower level architectural constraints, and details, such as memory access **contention** and **overhead**, **synchronization overhead**, **interconnection network throughput**, connectivity, **speed limits** and **link bandwidths**, etc.

Bulk Synchronous Parallel – BSP

Valiant 1990

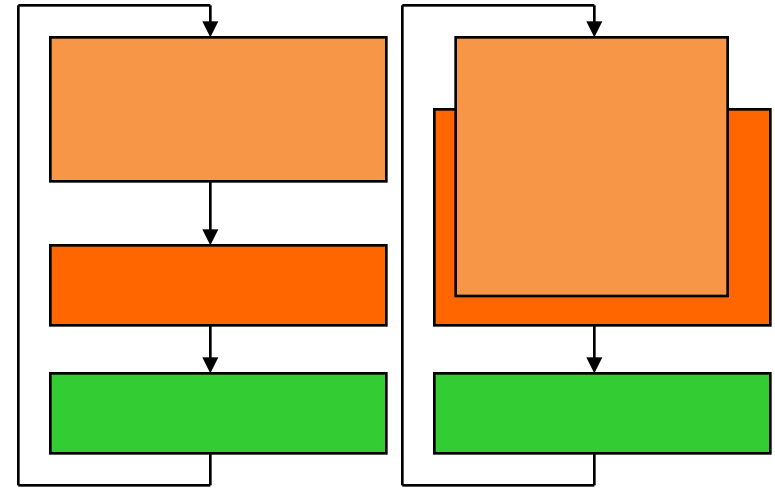
- Differs from PRAM by taking in account communications and synchronizations and by distributing the memory across participants
 - **Compute**: Components capable of computing or executing **local** memory transactions
 - **Communication**: A **network** routing messages between components
 - **Synchronization**: A support for **synchronization** on all or a sub-group of components



- Each processor can access his own memory without overhead and have a uniform slow access to remote memory

BSP - Superstep

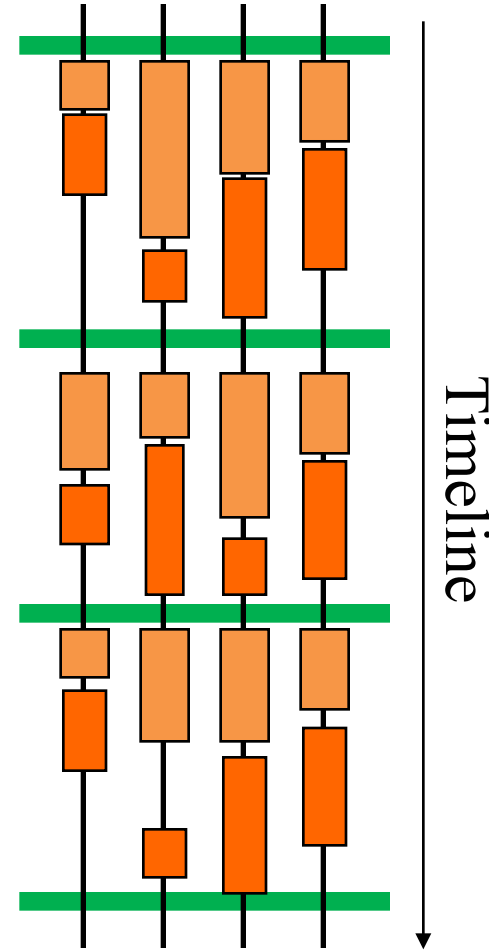
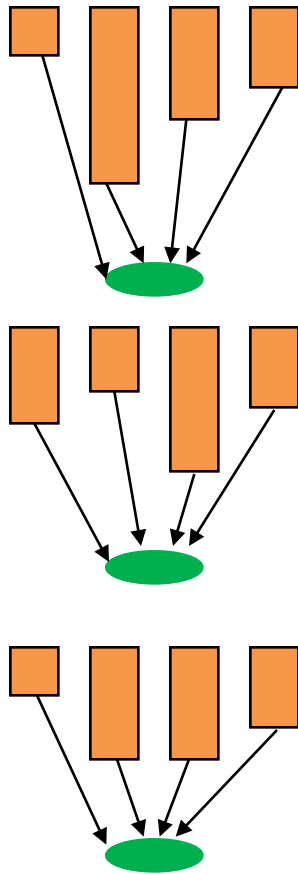
- Applications composed by Supersteps separated by global synchronizations.
- A superstep contains:
 - A computation step
 - A communication step
 - A synchronization step



Synchronization used to insure that all processors complete the computation + communication steps in the same amount of time.

As communications are remote memory accesses (one sided) there are no synchronizations during the computation + communication step

BSP - Global View



BSP – The communication step

- BSP consider communication not at the level of individual actions, but as a whole (per step)
- The goal being to define an upper bound on the time necessary to complete all data movements
- h = the maximum number of messages (incoming or outgoing) per superstep
- g = the network capability to deliver messages
 - It takes hg time for a processor to deliver h messages of size 1
 - A message of size m takes the same time to send as m messages of size 1

BSP – The synchronization cost

- The cost of synchronization is noted by l and is generally determined empirically
- With the increase in scale of the computing resources, the synchronizations are becoming the main bottleneck
 - Removing them might introduce deadlock or livelock
 - Decrease the simplicity of the model

BSP – Compute the cost

$$\begin{aligned} T_{superstep} &= \max_{i=1}^p (w_i) + g * \max_{i=1}^p (h_i) + l \\ &= w + g * h + l \end{aligned}$$

Where:

w = max of computation time

g = 1/(network bandwidth)

h = max of number of messages

l = time for the synchronization

$$T_{total} = \sum_{s=1}^S T_{superstep}$$

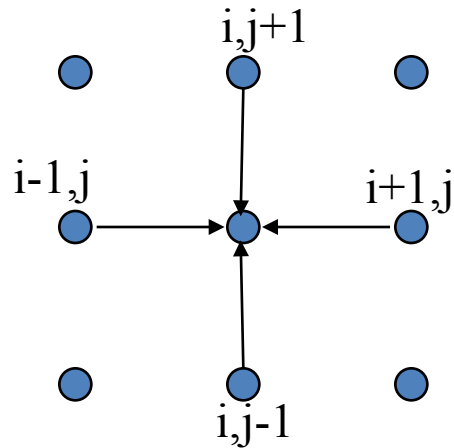
BSP

- An algorithm can be described using only w , h and the *problem size*.
- Collections of algorithms are available depending on the computer characteristics.
 - Small L
 - Small g
- The best algorithm can be selected depending on the computer properties.

BSP - example

- Numerical solution to Laplace's equation

$$U_{i,j}^{n+1} = \frac{1}{4} (U_{i-1,j}^n + U_{i+1,j}^n + U_{i,j-1}^n + U_{i,j+1}^n)$$



for j = 1 to jmax

for i = 1 to imax

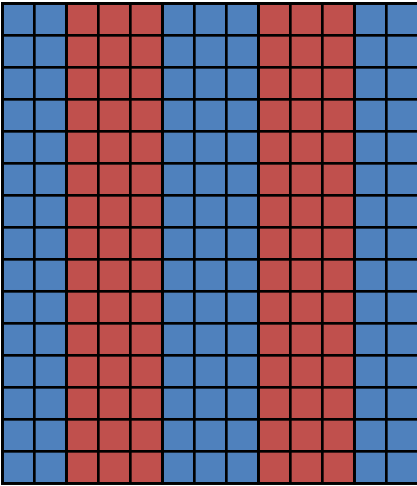
$$U_{\text{new}}(i,j) = 0.25 * (U(i-1,j) + U(i+1,j) \\ + U(i,j-1) + U(i,j+1))$$

end for

end for

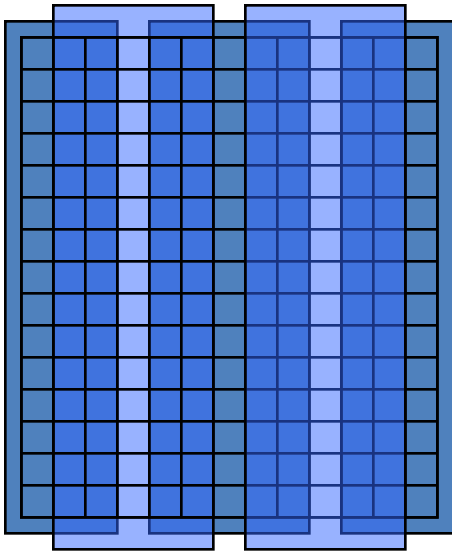
BSP - example

- The approach to make it parallel is by partitioning the data



BSP - example

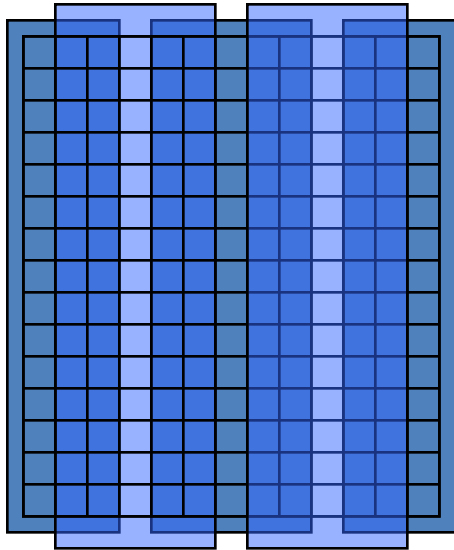
- The approach to make it parallel is by partitioning the data



Overlapping the data boundaries allow computation without communication for each superstep

On the communication step each processor update the corresponding columns on the remote processors.

BSP - example



```
for j = 1 to jmax
  for i = 1 to imax
    unew(i,j) = 0.25 * ( U(i-1,j) + U(i+1,j)
                      + U(i,j-1) + U(i,j+1))
  end for
end for
if me not 0 then
  bsp_put( to the left )
endif
if me not NPROCS - 1 then
  bsp_put( to the right )
Endif
bsp_sync()
```

BSP - example

$$T_{superstep} = w + g * h + l$$

h = max number of messages

= N values to the left +

N values to the right

= $2 * N$ (ignoring the inverse communication!)

$$w = 4 * N * N / p$$

$$T_{superstep} = 4 * \frac{N^2}{p} + 2 * g * N + l$$

BSP - example

- BSP parameters for a wide variety of architectures has been published.

Machine	s	p	l	g
Origin 2000	101	4	1789	10.24
		32	39057	66.7
Cray T3E	46.7	4	357	1.77
		16	751	1.66
Pentium 10Mbit	61	4	139981	1128.5
		8	826054	2436.3
Pentium II 100Mbit	88	4	27583	39.6
		8	38788	38.7