High Performance Parallel Computing (4.) parallelization

Imre Szeberényi

BME IIT

<szebi@iit.bme.hu>

Some of the figures are from Ian Foster:
Designing and Building Parallel Programs
(Addison-Wesley).
https://www.mcs.anl.gov/~itf/dbpp/

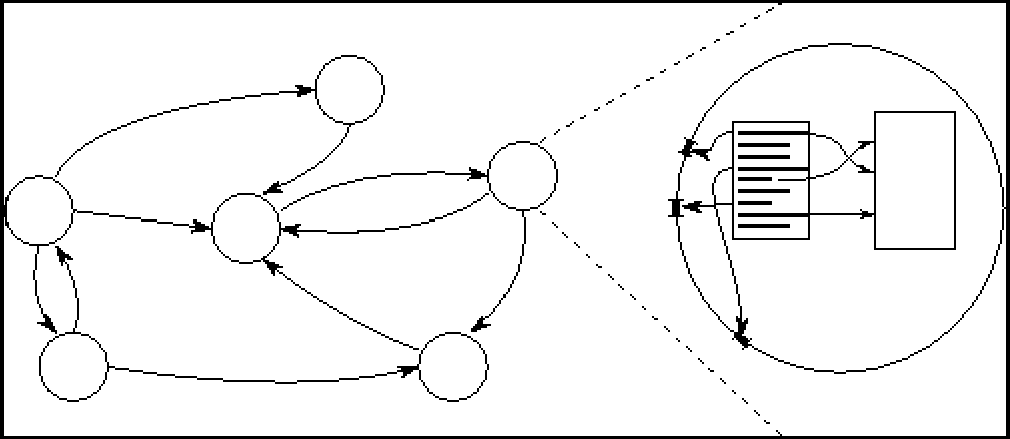
MŰEGYETEM 1782

Short summary

- Models of parallel computers
  - Flynn’s taxonomy. idealized parallel computer model
- Programming models
  - Shared memory. Distr. shared mem. Message passing
- Classes (types) of parallel computers
  - computers with vector processors . Symmetric Multiprocessors (SMP). Massively Parallel(MPP), Cluster
- Tools
  - MPI, OpenMP

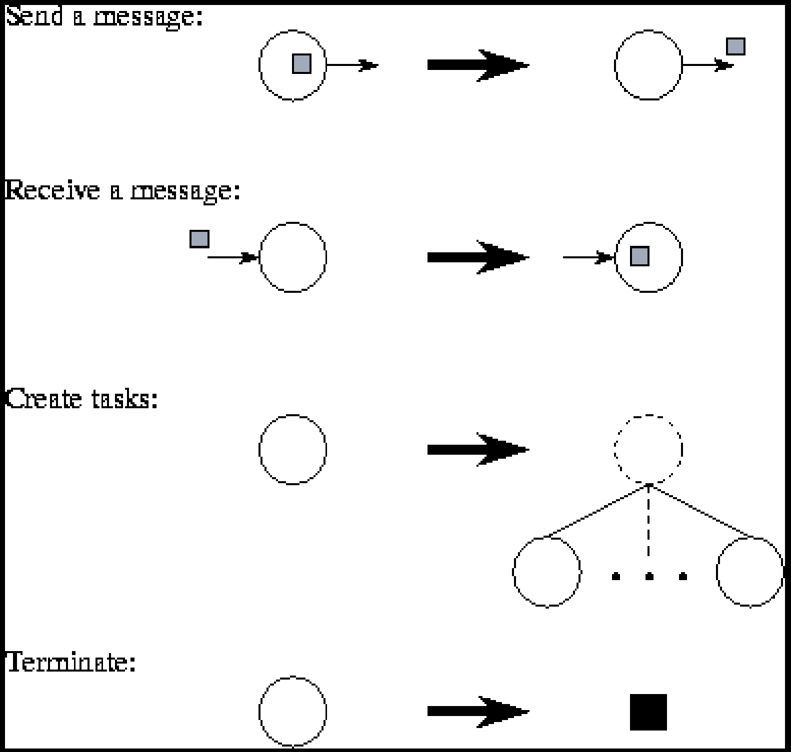
Task/Channel model /1

- Each task runs a sequential algorithm
- Each task has own local memory
- Tasks are connected through channels
- The channels are realized by messages queues



Task/Channel model /2

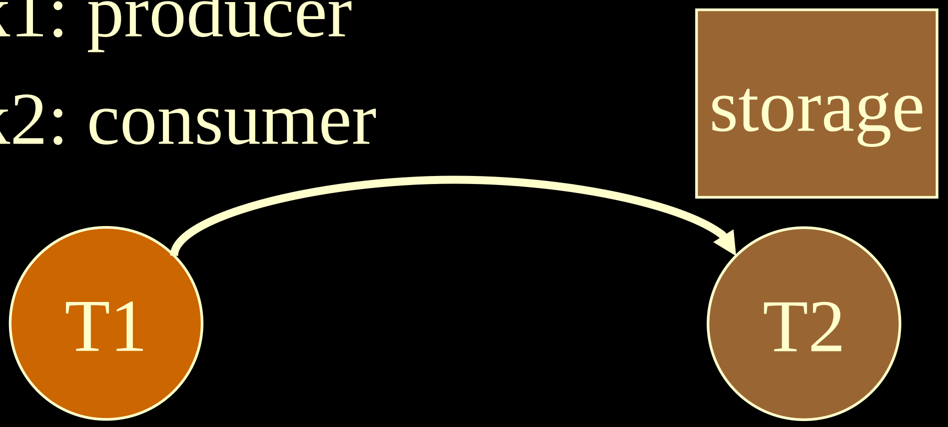
- Tasks are concurrent
- The send is synchronised
- The receive is asynchronous
- Tasks are connected to the channels through in/out ports



- Task can be associated with processors in any manner

Task/Channel model /3

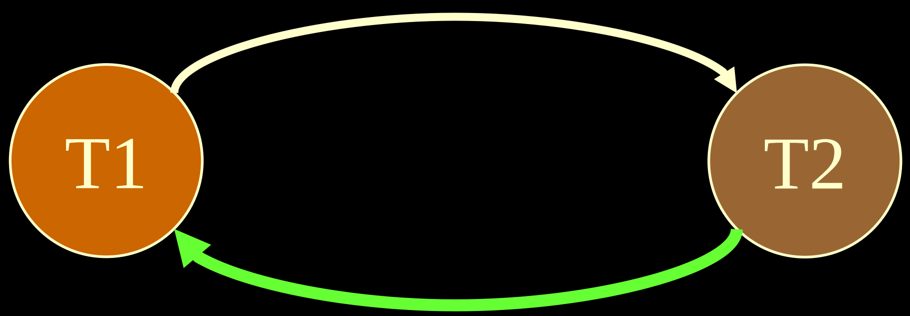
- Example: producer-consumer problem
  - task1: producer
  - task2: consumer



- if the consumer is slower, the product are stored
- if the producer is slower, the consumer waits

Task/Channel model /4

- Example: producer-consumer problem
  - task1: producer
  - task2: consumer



- The second channel used for triggering the producer

Attributes of the model

- Can be associated to the idelalized parallel machine.
- The task represents a serial program
- The channel realize the communication between the tasks
- The tasks are independent from the processor mappings.
- Enables a modular setup

Task/Channel vs. message

- The message addressed to a specific task, so it is not enough abstract.
- In the general message passing model does not allow dynamic task creation.
- One processor can have only one task in the the message passing model.

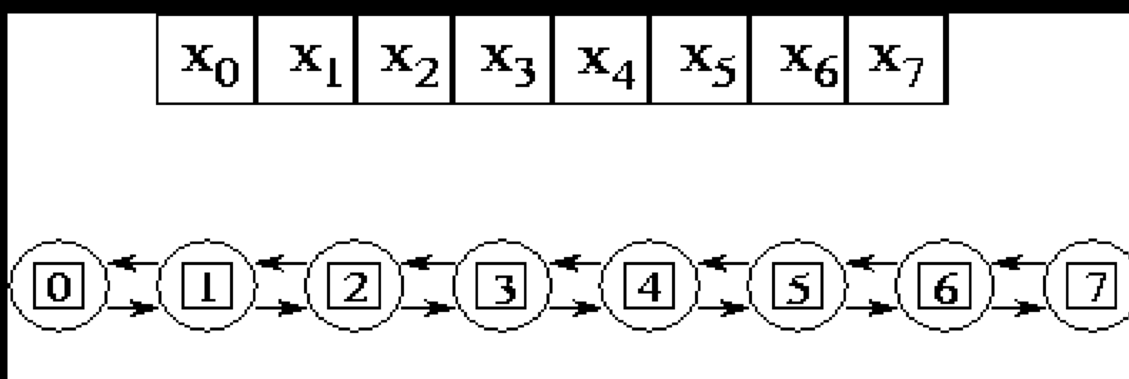
The last two bullet points are not real restrictions in many message passing environments.

Examples /1

- Finite difference:
  - Operation on each vector element in T times:

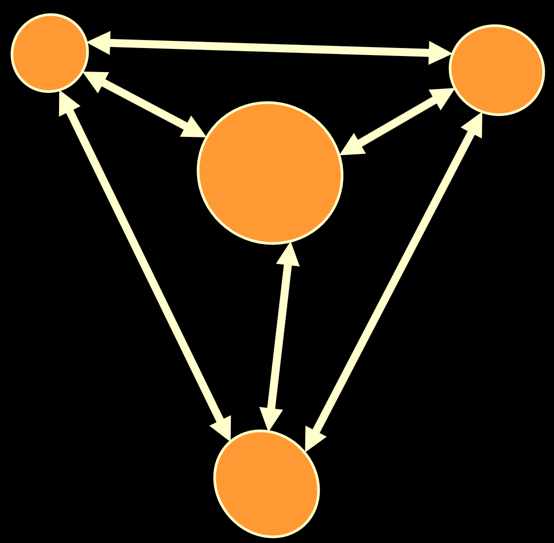
$$0 < i < N-1,\ 0 \leq t < T\ :\ X_i^{(t+1)} = \frac{X_{i-1}^{(t)} + 2X_i^{(t)} + X_{i+1}^{(t)}}{4}$$

  - Each element computed by different tasks:



Examles /2

- Pairwise iteration (ex: interactions of atoms)



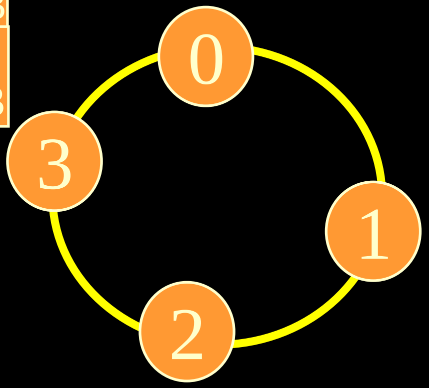
$$f_i = \sum_{j=0}^{N-1} F(X_i, X_j)$$

  - N*(N-1) messages, or
  - N*(N-1)/2 taking advantage of symmetry

Examples /3

- Circular connection (channel) gives more effective solution
  - Each task puts own data to a vector and send it.
  - Each task receives the vector and completes the data
  - After N-1 steps each task knows everything.
  - The force (F) can be computed after each step.

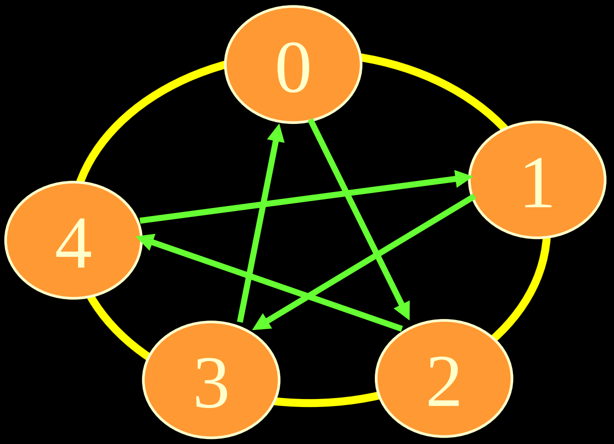
L_0	L_1	L_2	L_3
F_0	F_1	F_2	F_3



Examples /4

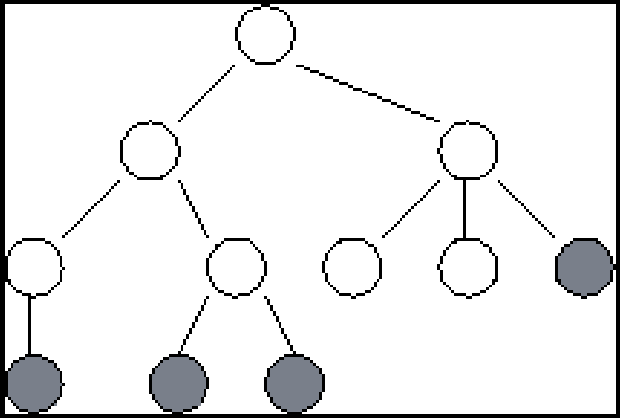
- The modell can be simplified by N additional channels:
  - Create a new communication channel between the task i and task i+N/2.
  - Compute the forces and distribute the results in each steps.
  - Only N/2 iteration needed.

L_0	L_1	L_2	L_3	L_4
F_0	F_1	F_2	F_3	F_4

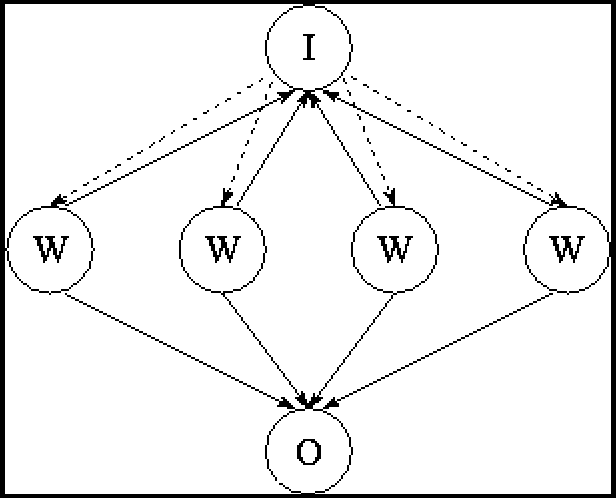


Examples /5

- Parallel search:
  - simple task distribution in the three

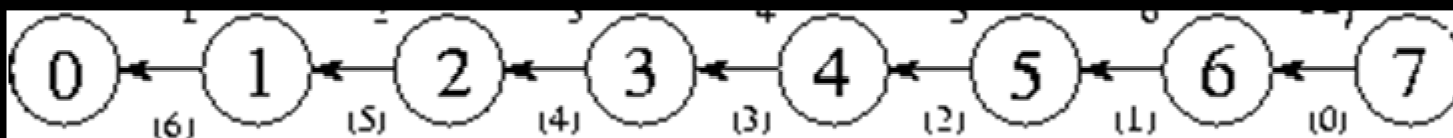


- Parameter scan:
  - master-worker algorithm

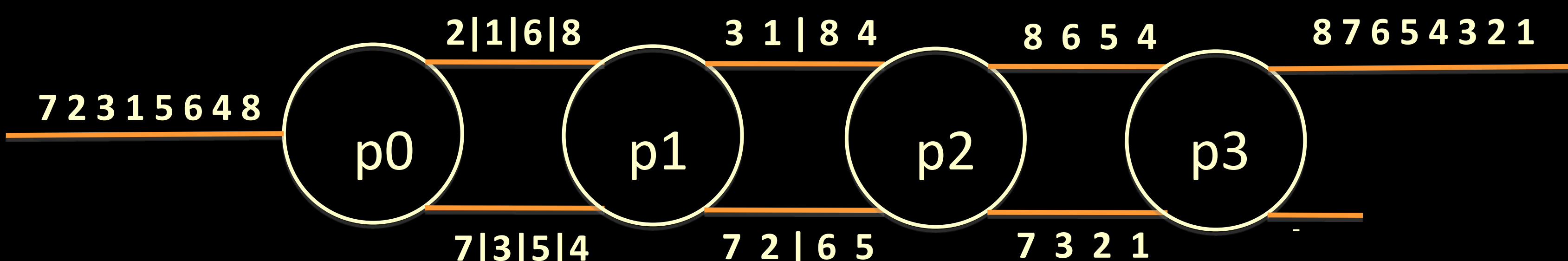


Examples /6

- Pipeline sort:
  - each node keep the bigger value
  - the smaller is sent to the next node

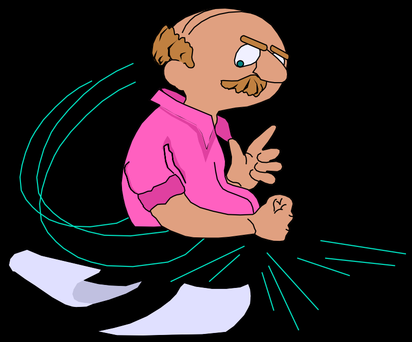


- Pipeline merge sort



Designing Parallel Algorithms

- Designing a parallel algorithm is not easy.
- There is no recipe or magical ingredient
  - Except creativity
- We can benefit from a systematic approach.
  - Framework for algorithm design
- Most problems have several parallel solutions which may be totally different from the best sequential algorithm.

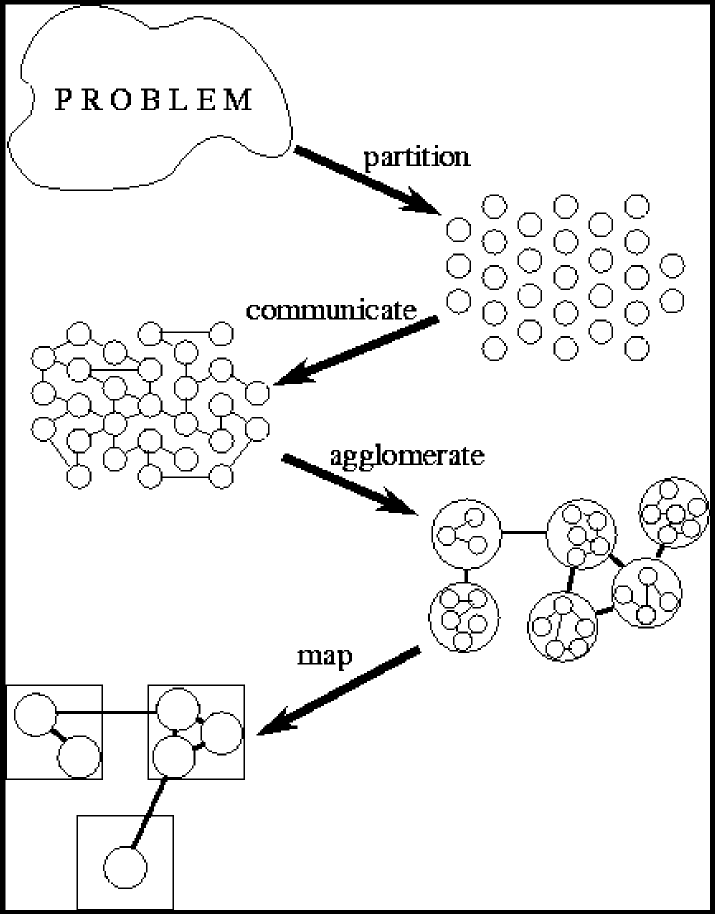


PCAM Algorithm Design

- 4 Stages to designing a parallel algorithm
  – Partitioning
  – Communication
  – Agglomeration
  – Mapping
- P & C focus on concurrency and scalability.
- A & M focus on locality and performance.

PCAM Algorithm Design

- Partitioning
  - Computation and data are decomposed.
- Communication
  - Coordinate task execution
- Agglomeration
  - Combining of tasks for performance
- Mapping
  - Assignment of tasks to processors



Partitioning

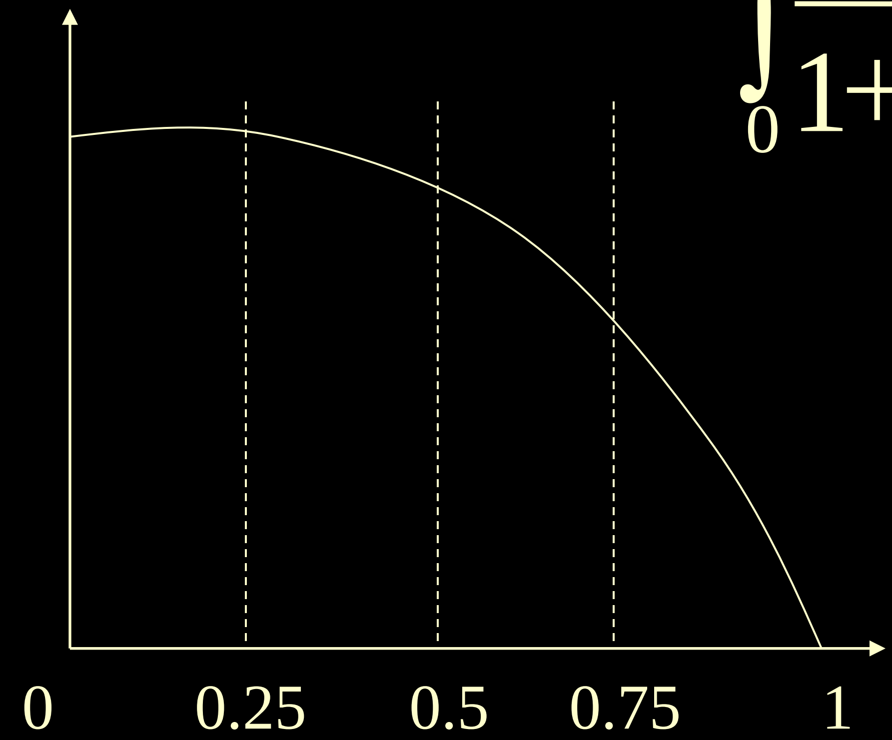
- Ignore the actual number of processors and the target architecture.
- Expose opportunities for parallelism.
- Divide up both the computation and data
- Can take two approaches
  - domain decomposition
  - functional decomposition

Domain Decomposition

- Start algorithm design by analyzing the data
- Divide the data into small pieces
  - Approximately equal in size
- Then partition the computation by associating it with the data.
- Communication issues may arise as one task needs the data from another task.

Domain Decomposition

- Evaluate the definite integral. $\int_0^1 \frac{4}{1+x^2}$

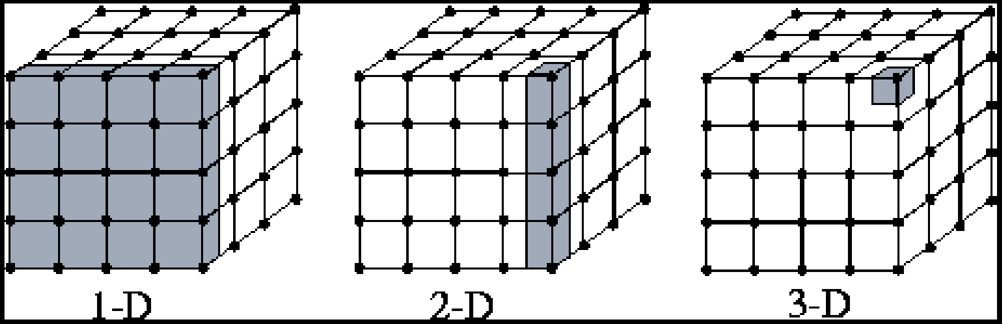


Now each task simply evaluates the integral in their range.

All that is left is to sum up each task's answer for the total.

Domain Decomposition

- Consider dividing up a 3-D grid
  - What issues arise?



- Other issues?
  - What if your problem has more than one data structure?
  - Different problem phases?
  - Replication?

Functional Decomposition

- Focus on the computation
- Divide the computation into disjoint tasks
  - Avoid data dependency among tasks
- After dividing the computation, examine the data requirements of each task
- Typical method when the data partitioning is not possible (ex. searching in a tree)

Partitioning Checklist

- Define a LOT of tasks?
- Avoid redundant computation and storage?
- Are tasks approximately equal?
- Does the number of tasks scale with the problem size?
- Have you identified several alternative partitioning schemes?

Communication

- The information flow between tasks is specified in this stage of the design

- Remember:
  - Tasks execute concurrently.
  - Data dependencies may limit concurrency.

Communication

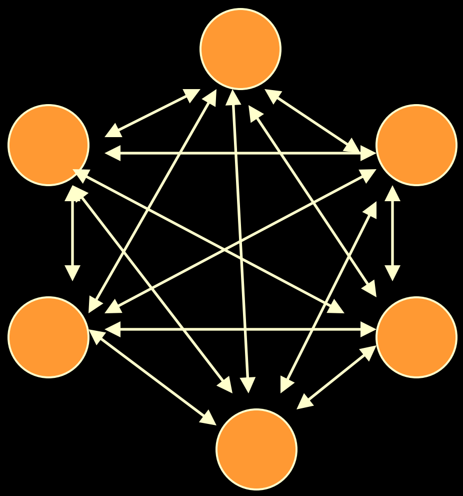
- Define Channel
  - Link the producers with the consumers.
  - Consider the costs
    - Logical
    - Physical
  - Distribute the communication.
- Specify the messages that are sent.

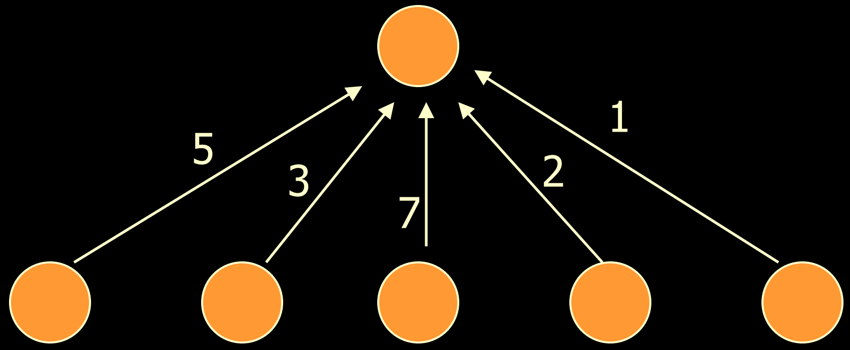
Communication Patterns

- Local vs. Global
- Structured vs. Unstructured
- Static vs. Dynamic
- Synchronous vs. Asynchronous

Global Communication

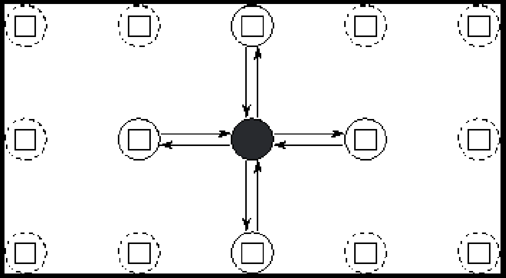
- Not localized.
- Examples
  - All-to-All
  - Master-Worker





Communication examples /1

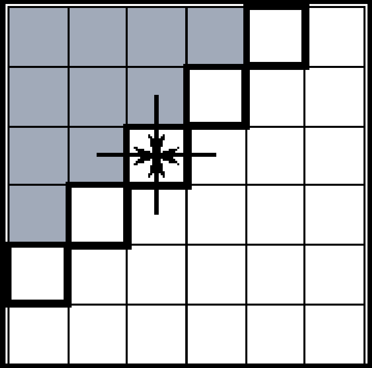
Boundary value problems width FDM



Jakobi FDM:

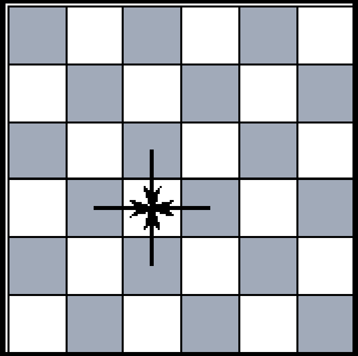
$$X_{i,j}^{(t+1)} = \frac{4X_{i,j}^{(t)} + X_{i-1,j}^{(t)} + X_{i+1,j}^{(t)} + X_{i,j-1}^{(t)} + X_{i,j+1}^{(t)}}{8}.$$

(Gauss-Seidel):



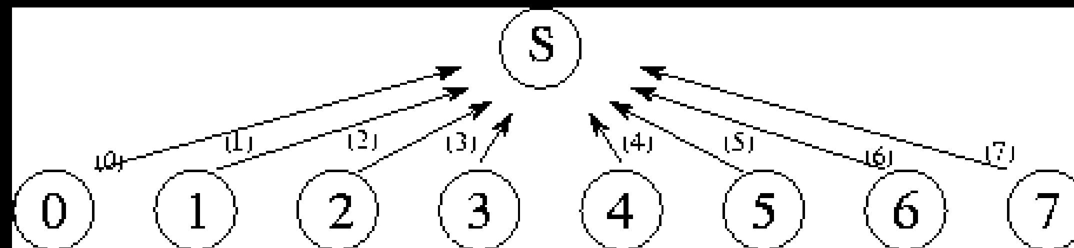
$$X_{i,j}^{(t+1)} = \frac{4X_{i,j}^{(t)} + X_{i-1,j}^{(t+1)} + X_{i+1,j}^{(t)} + X_{i,j-1}^{(t+1)} + X_{i,j+1}^{(t)}}{8}.$$

Red-Black ordering:

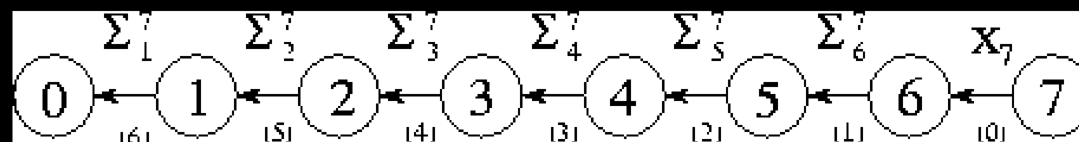


Communication examples /2

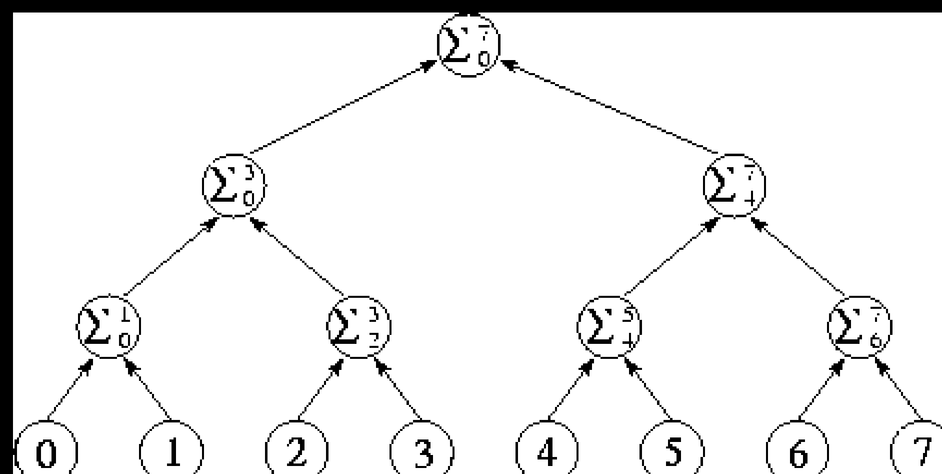
Summ:



Pipeline:



Divide and conquer



Problems to Avoid

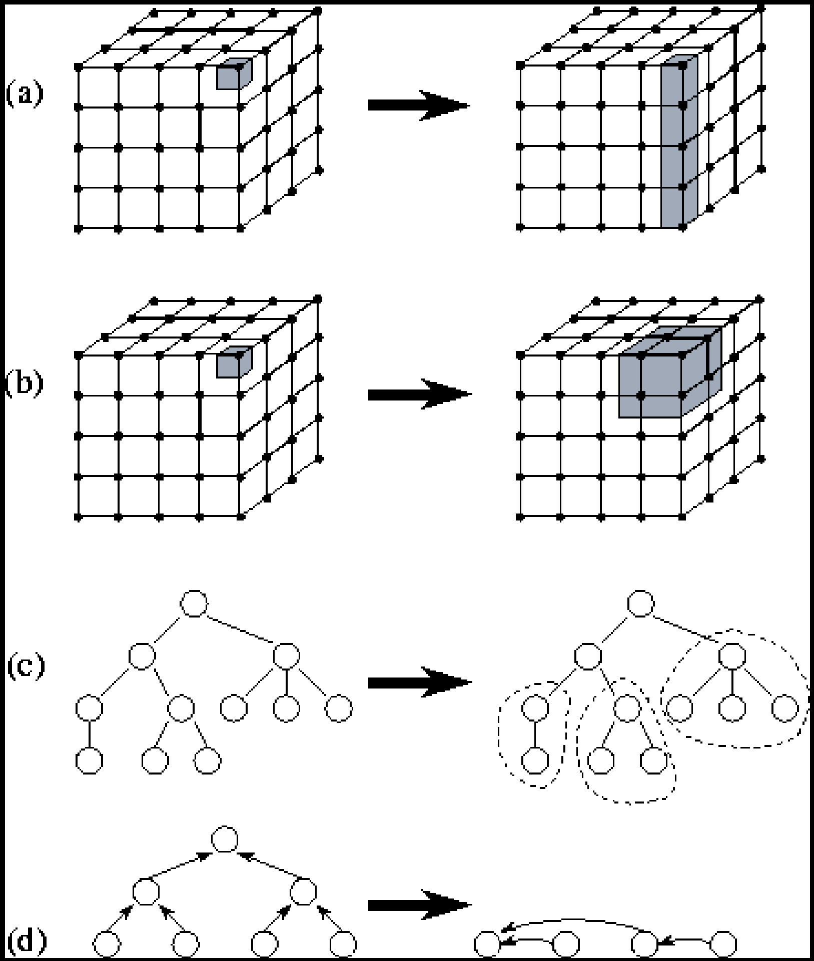
- A centralized algorithm
  - Distribute the computation
  - Distribute the communication
- A sequential algorithm
  - Seek for concurrency
  - Divide and conquer
    - Small, equal sized subproblems

Communication Design Checklist

- Is communication balanced?
  - All tasks size about the same
- Is communication limited to neighborhoods?
  - Restructure global to local if possible.
- Can communications proceed concurrently?
- Can the algorithm proceed concurrently?
  - Find the algorithm with most concurrency.
    - Be careful!!!

Agglomeration

- Partition and Communication steps were abstract
- Agglomeration moves to concrete.
- Combine tasks to execute efficiently on some parallel computer.



Agglomeration Goals

- Reduce communication costs by
  - increasing computation
  - decreasing/increasing granularity
- Retain flexibility for mapping and scaling.
- Reduce software engineering costs.

Changing Granularity

- A large number of tasks does not necessarily produce an efficient algorithm.
- We must consider the communication costs.
- Reduce communication by
  - having fewer tasks
  - sending less messages (batching)

Surface to Volume Effects

- The Communication /Computation rate should be kept in low level.
- Communication is proportional to the surface of the subdomain.
- Computation is proportional to the volume of the subdomain.
- Remember: the sphere has the lowest surface/volume rate.
- Increasing computation will often decrease communication.

Avoid Communication

- Look for tasks that cannot execute concurrently because of communication requirements.
- Replication can help accomplish two tasks at the same time.

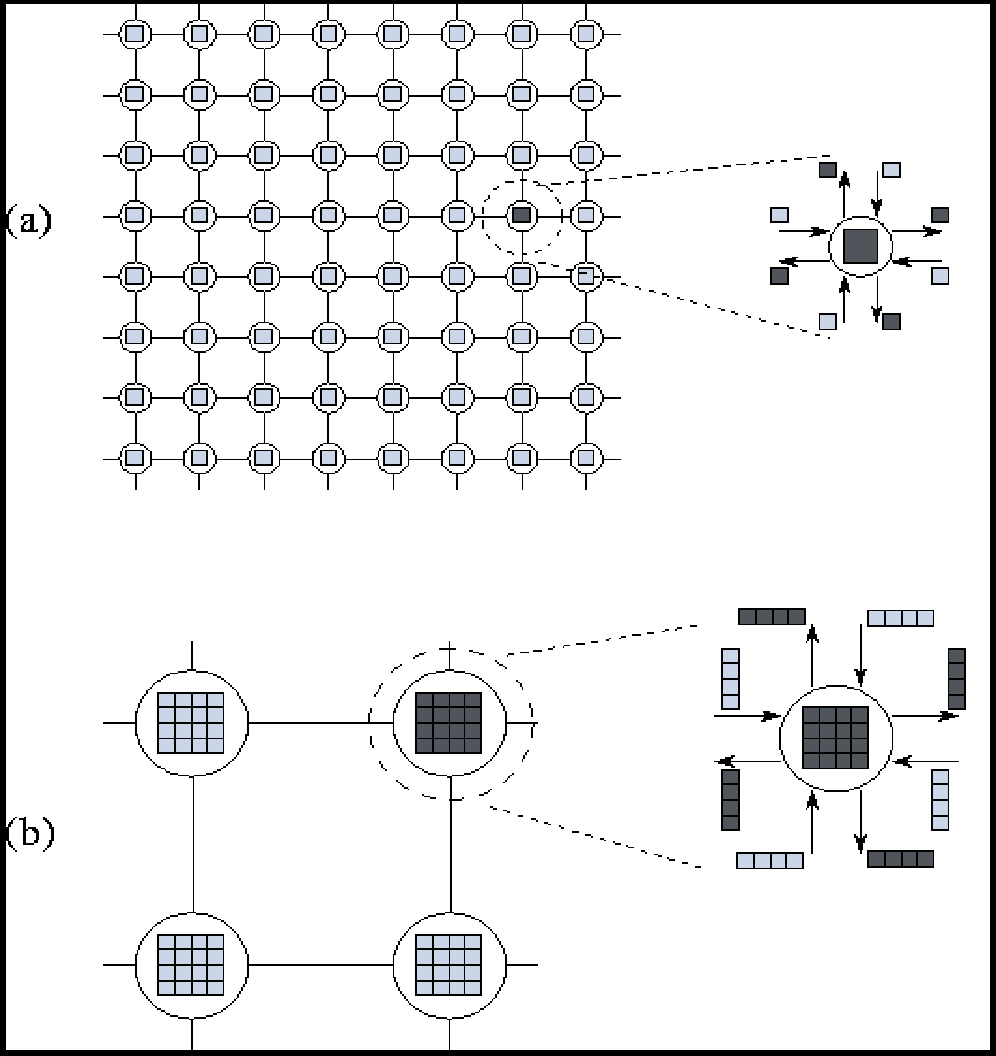
Preserve Flexibility

- Create more tasks than processors.
- Overlap communication and computation.
- Don't incorporate unnecessary limits on the number of tasks.

Agglomeration Checklist

- Reduce communication costs by increasing locality.
- Do benefits of replication outweigh costs?
- Does replication compromise scalability?
- Does the number of tasks still scale with problem size?
- Is there still sufficient concurrency?

Agglomeration example



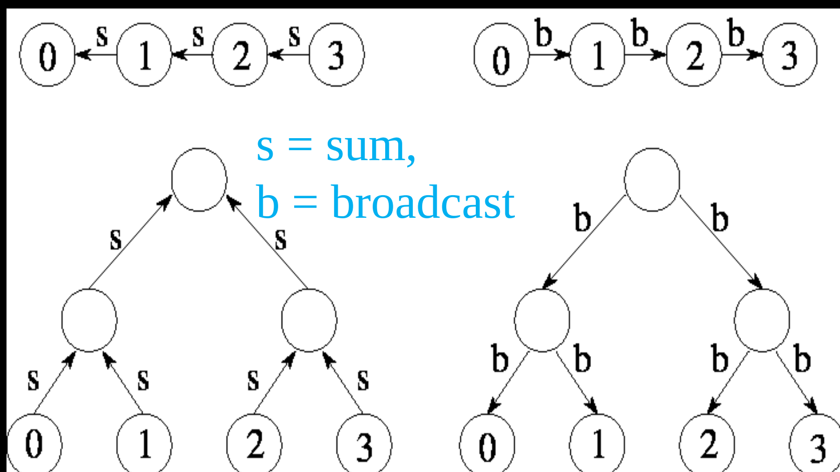
(a) 8*8=64 task,
64*4=256 mesages
256*1=256 data
S/V = 4/1

(b) 2*2=4 task,
4*4=16 messages
16*4=64 data
S/V = 16/16

Surface/Volume ratio ⇒
Comm./Computation

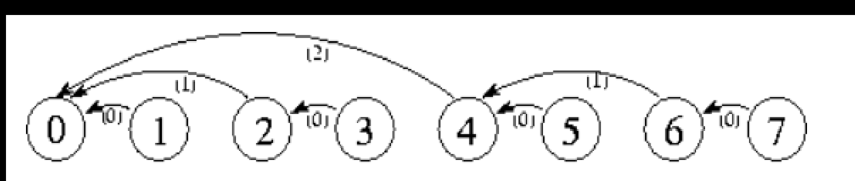
Summ of N Integers

The result should be sent to each node.

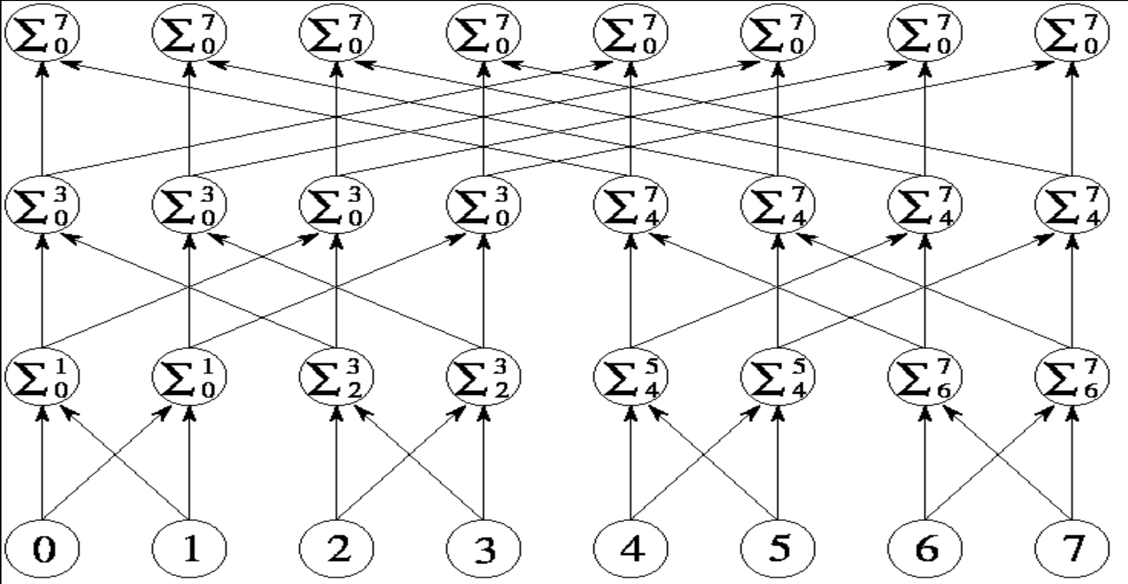


	Steps	Messages
Array:	2(N-1)	2(N-1)
Tree:	2 log N	4(N-1)

Communication structure in tree version:

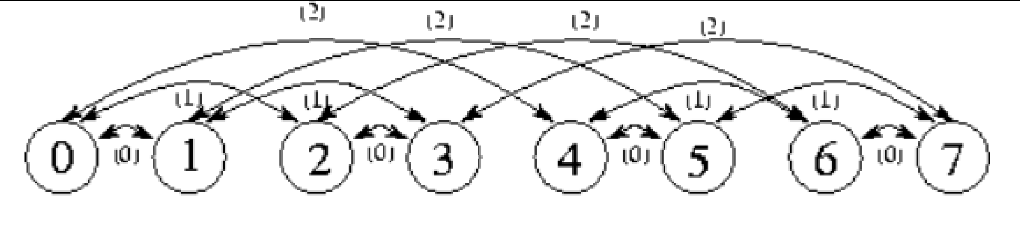


Using Replication (Butterfly)



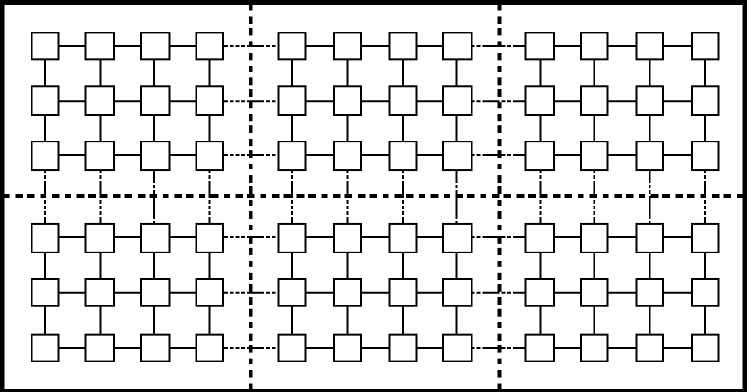
Steps	Messages
log N	2N log N

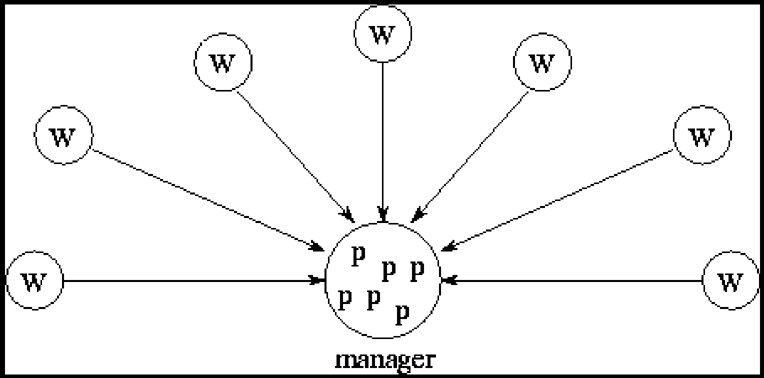
Communication structure:



Mapping

- Specify where each task is to operate.
- Mapping may need to change depending on the target architecture.
- Mapping has a big influence to the load balancing and scheduling.





Mapping

- Goal: Reduce Execution Time
  - Concurrent tasks ---> Different processors
  - High communication ---> Same processor
- Mapping is a game of trade-offs.

Other Mapping Problems

- Variable amounts of work per task
- Unstructured communication
- Heterogeneous processors
  - different speeds
  - different architectures

- Solution: LOAD BALANCING

Load Balancing

- Static
  - Determined a priori
  - Based on work, processor speed, etc.
- Probabilistic
  - Random
- Dynamic
  - Restructure load during execution
- Task Scheduling (functional decomp.)

Static Load Balancing

- Based on a priori knowledge.
- Goal: Equal WORK on all processors
- Algorithms:
  - Basic
  - Recursive Bisection

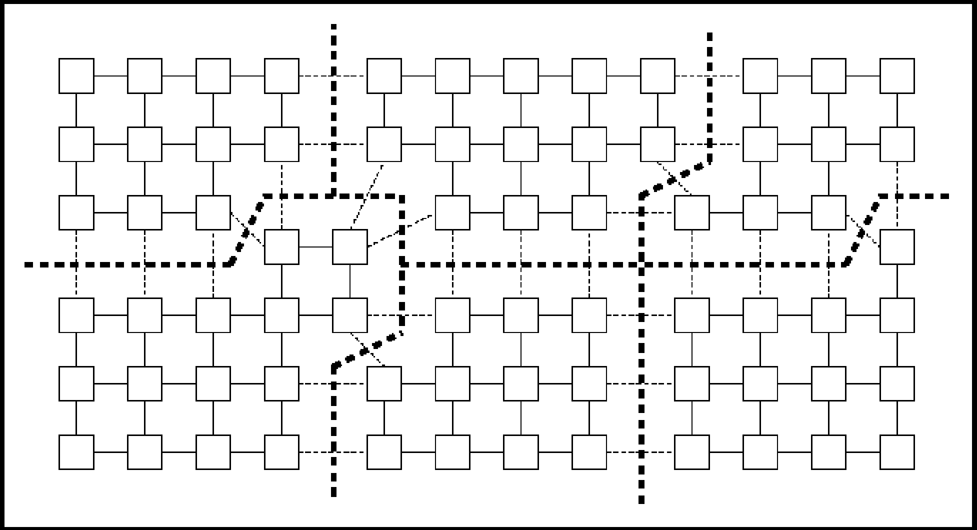
Basic

- Divide up the work based on
  - Work required (R)
  - Processor speed (p_i)

$$r_i = R\left(\frac{p_i}{\sum_i p_i}\right)$$

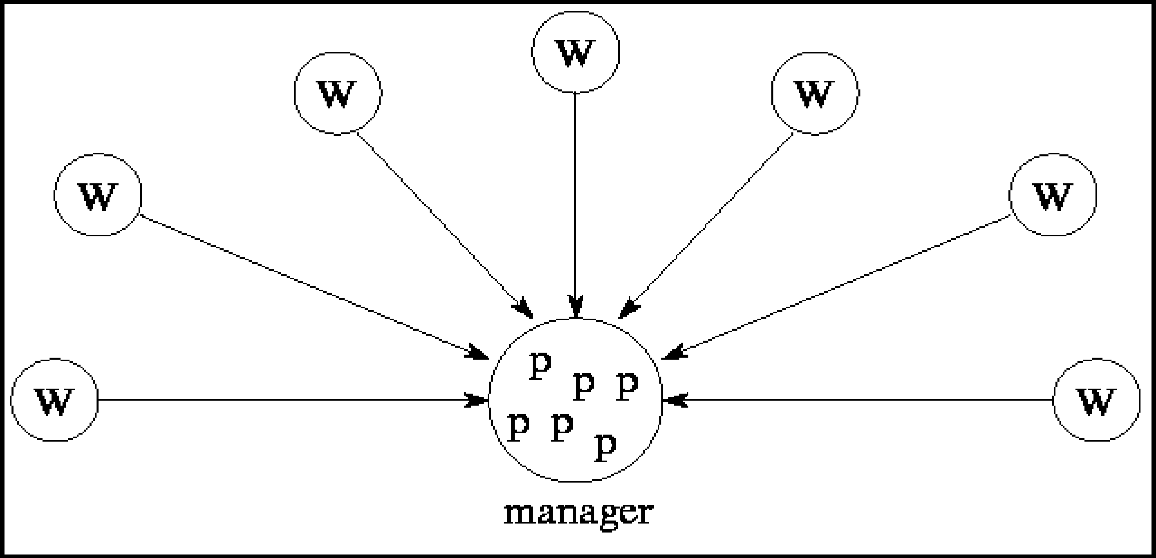
Dynamic Algorithms

- Adjust load when an imbalance is detected.
- Local or Global



Task Scheduling

- Many tasks with weak locality requirements.
- Manager-Worker model.



Task Scheduling

- Manager-Worker
- Hierarchical Manager-Worker
  - Uses submanagers
- Decentralized
  - No central manager
  - Task pool on each processor
  - Less bottleneck

Mapping Checklist

- Is the load balanced?
- Are there communication bottlenecks?
- Is it necessary to adjust the load dynamically?
- Can you adjust the load if necessary?
- Have you evaluated the costs?

PCAM Summary

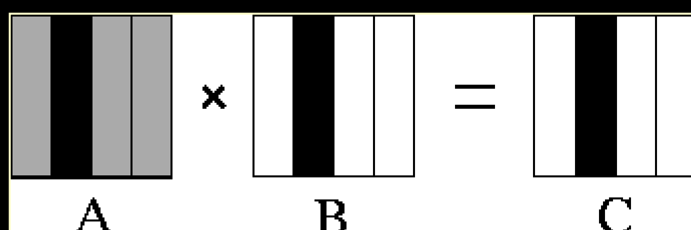
- Partition
  - Domain or Functional Decomposition
- Communication
  - Link producers and consumers
- Agglomeration
  - Combine tasks for efficiency
- Mapping
  - Divide up the tasks for balanced execution

Example 1: matrix multiplication

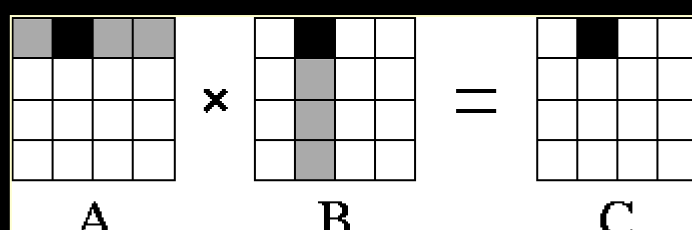
Matrix-matrix mul: $O(N^3)$

$$C_{ij} = \sum_{k=0}^{N-1} A_{ik} \cdot B_{kj}$$

- Partitioning:
  - each C_{ij} should be computed by only one proc
  - domain decomposition
  - 1D:

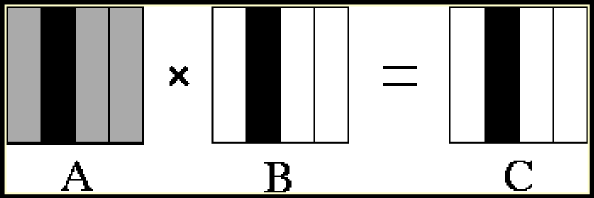


  - 2D:



Example 1: matrix mul /2

- Communication
  - 1D: Each proc. uses the mat A.



  - there is a process which broadcasts and collects the data
  - the communication

$$T_{1d} = (P\text{-}1)*(N^2+2*N^2/P)*t_w + (P\text{-}1)*t_s$$

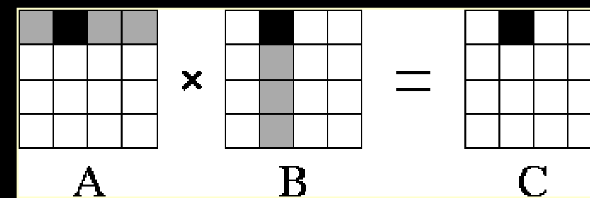
t_w – time for transferring data

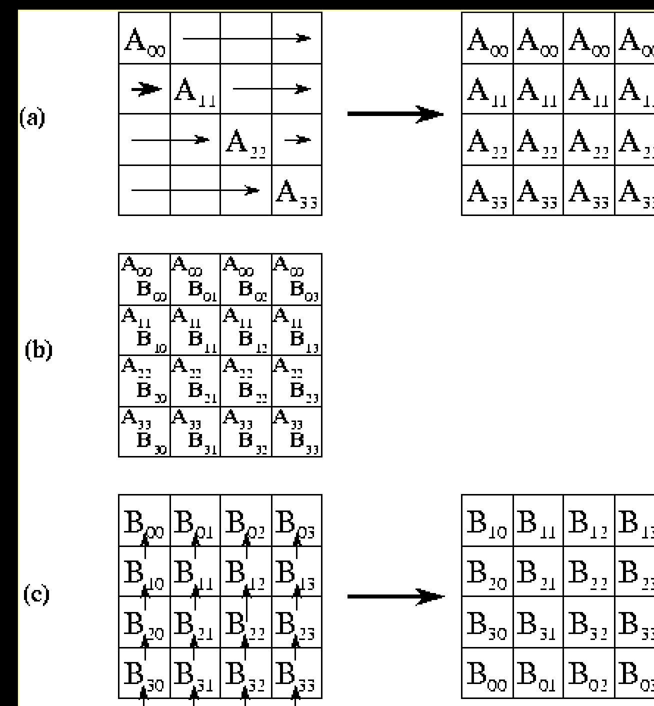
t_s – time for setting up the communication

$T_{1d} \approx O(P*N^2)$

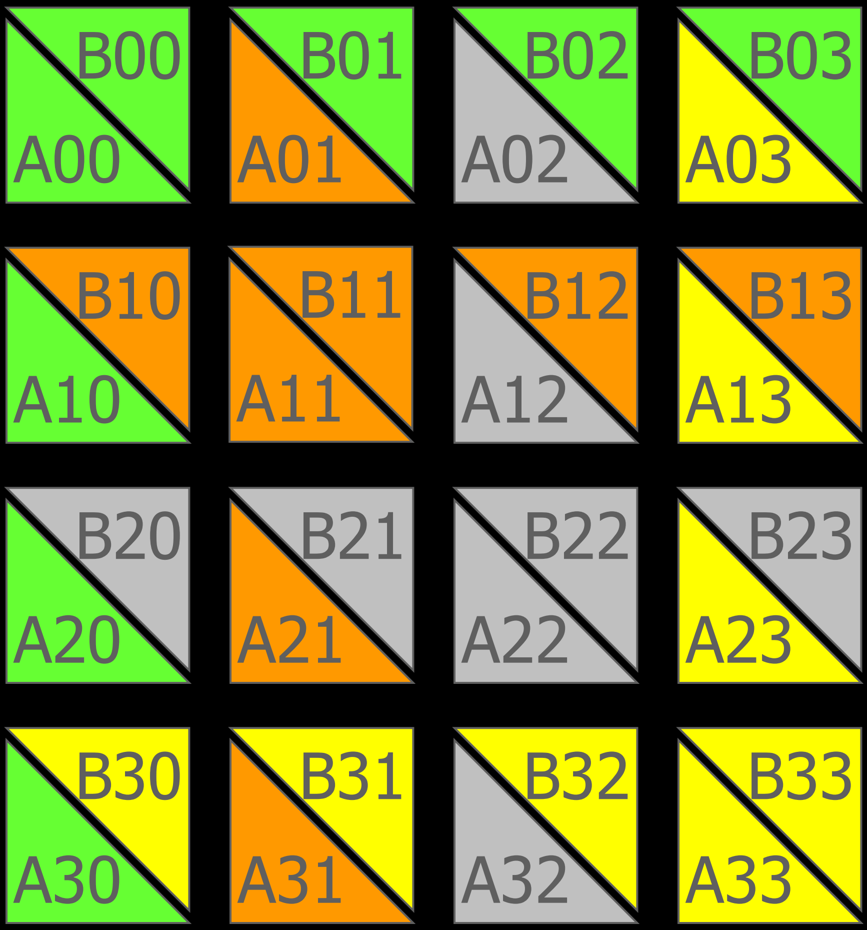
Example 1: matrix mul /3

- Communication
  - 2D: The same row and same column needed from A & B
  - Algorithm (Fox's):
    1. C = 0
    2. Repeat N-1-times
    3. Put the diagonals of A into rows A'
    4. $C_{ij} = C_{ij} + B_{ij} * A'_{ij}$
    5. B cyclic shift up
  - $T_{2d} = 3*(\sqrt{P}-1)*N^2/P*t_w + 3*(\sqrt{P}-1)*t_s \approx O(N^2/\sqrt{P})$





Eliminating the broadcast (Cannon's)

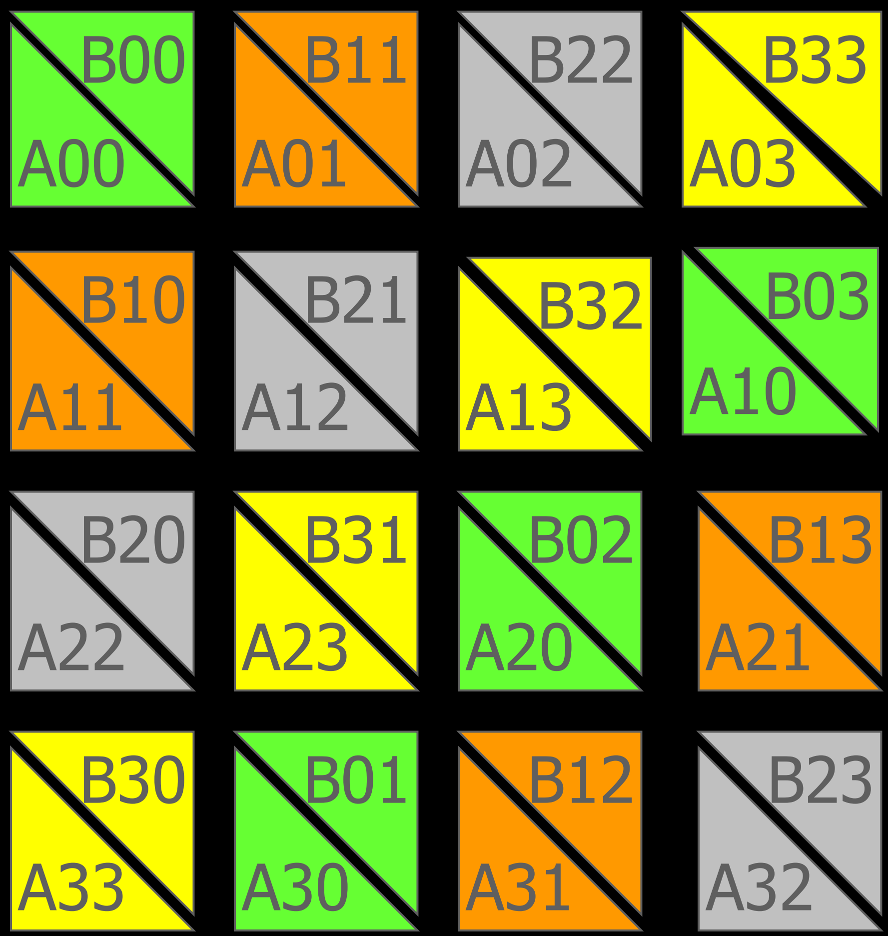


Each triangle denotes one block of the matrix.

Only blocks width same colour can be multiplied.

Trick: reorder the block!

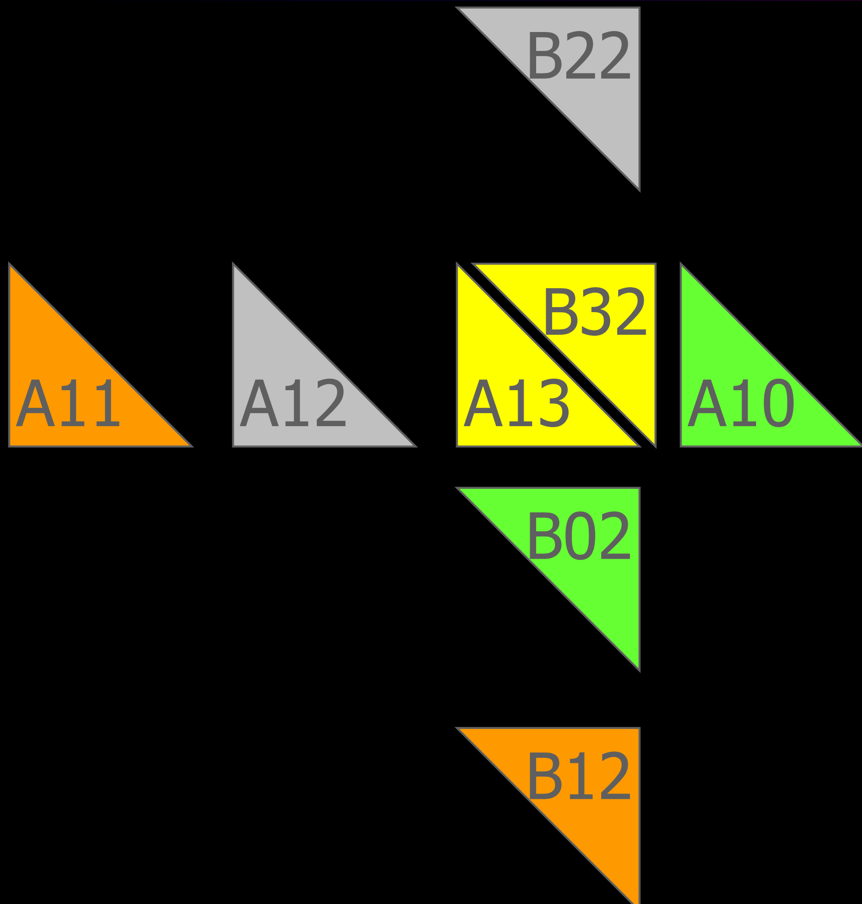
Reordering the blocks



Block $\mathbf{A}_{\mathbf{ij}}$ cyclic shifted left by **i** positions

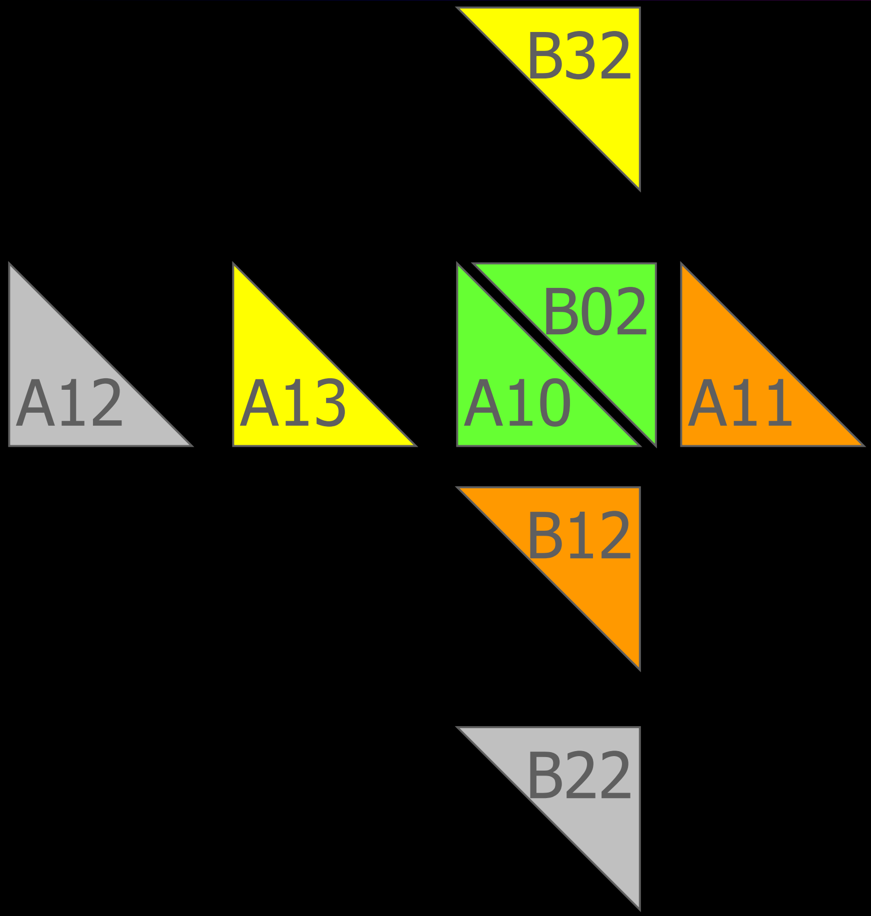
Block $\mathbf{B}_{\mathbf{ij}}$ cyclic shifted up by **j** positions

Step 1



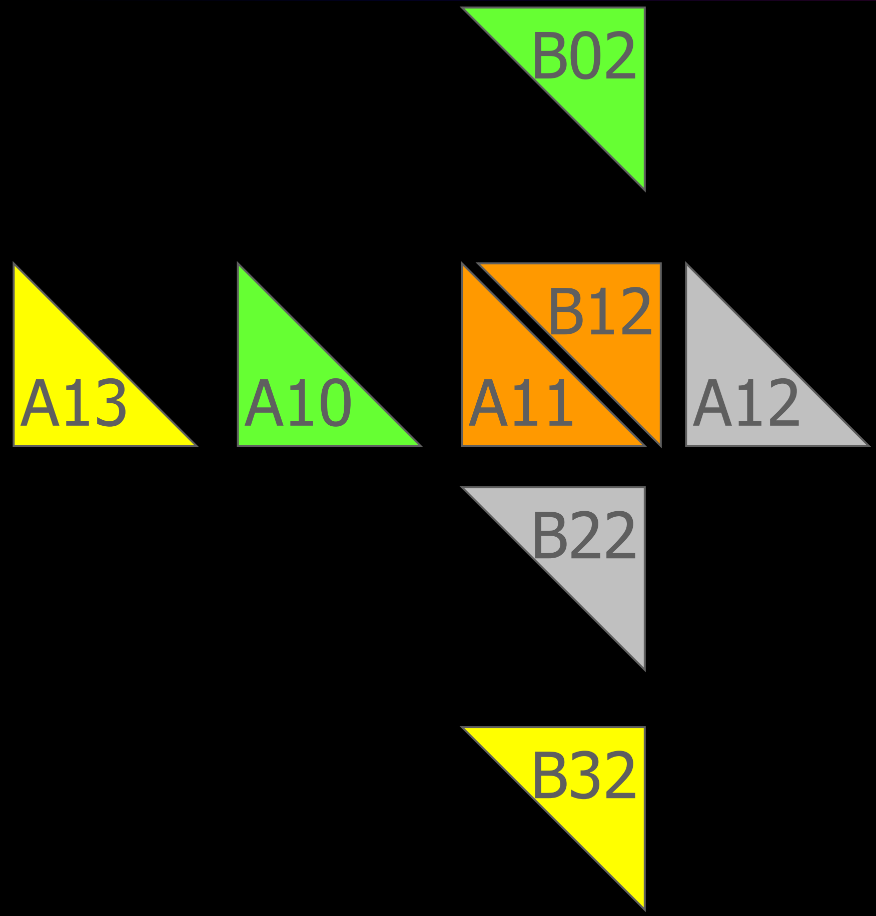
C12 = A10*B02 + A11*B12 + A12*B22 + A13*B32

Step 2



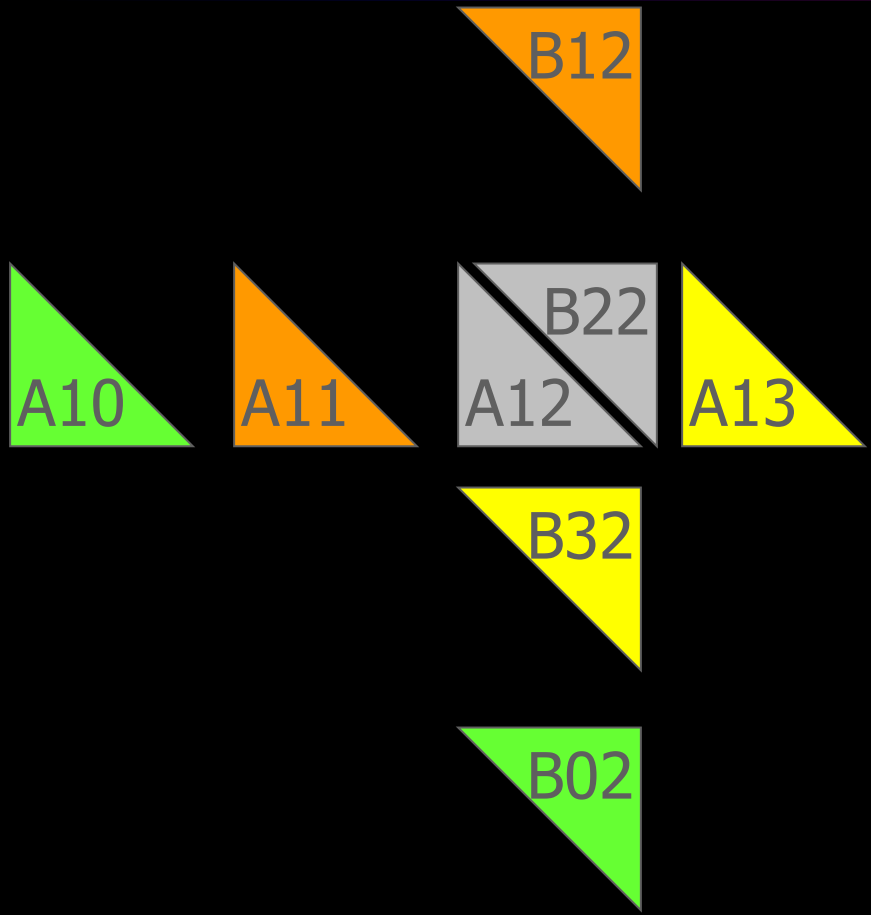
C12 = A10*B02 + A11*B12 + A12*B22 + A13*B32

Step 3



C12 = A10*B02 + A11*B12 + A12*B22 + A13*B32

Step 4



C12 = A10*B02 + A11*B12 + A12*B22 + A13*B32

The implemenation

```
// PE(i , j)
k := (i + j) mod N;
receive a[i][k] as a from coordinator;
receive b[k][j] as b from coordinator;
c[i][j] := 0;
for (l := 0; l < N; l++) {
    c[i][j] := c[i][j] + a * b;
        concurrently {
            send a to PE(i, (j + N – 1) mod N);
            send b to PE((i + N – 1) mod N, j);
        } with {
            receive a' from PE(i, (j + 1) mod N);
            receive b' from PE((i + 1) mod N, j );
        }
    a := a';
    b := b';
}
```