ECE 669

Parallel Computer Architecture

Lecture 23

Parallel Compilation

Parallel Compilation

- Two approaches to compilation
  - Parallelize a program manually
  - Sequential code converted to parallel code
- Develop a parallel compiler
  - Intermediate form
  - Partitioning
    - Block based or loop based
  - Placement
  - Routing

Compilation technologies for parallel machines

Assumptions:

Input: Parallel program

Output: "Coarse" parallel program & directives for:

- Which threads run in 1 task
- Where tasks are placed
- Where data is placed
- Which data elements go in each data chunk

Limitation: No special optimizations for synchronization -- synchro mem refs treated as any other comm.

Toy example

° Loop parallelization

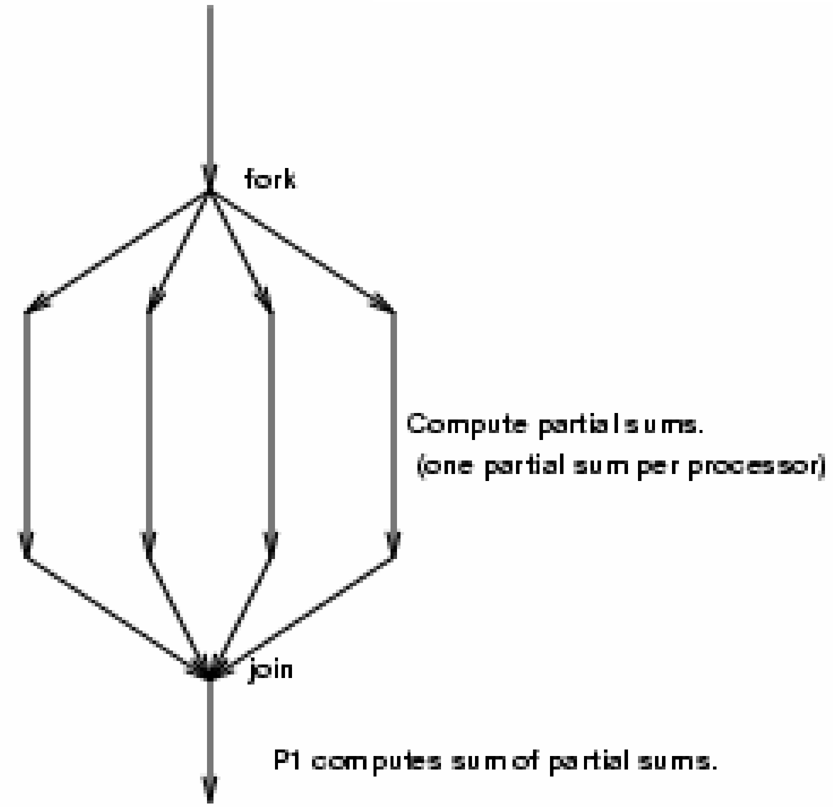
Adding a total of 4n integers, $a_1, a_2, \cdots, a_{4n}$, on a 4-processor system.

Processor 0 will execute $a_0 + a_2 + \cdots + a_{n-1}$.

Processor 0 will execute $a_n + a_{n+2} + \cdots + a_{2n}$

Processor 0 will execute $a_{2n} + a_{2n+2} + \cdots + a$

Processor 0 will execute $a_{3n} + a_{3n+2} + \cdots + a$



Example

- **Matrix multiply**
- **Typically,**

FORALL i

FORALL j

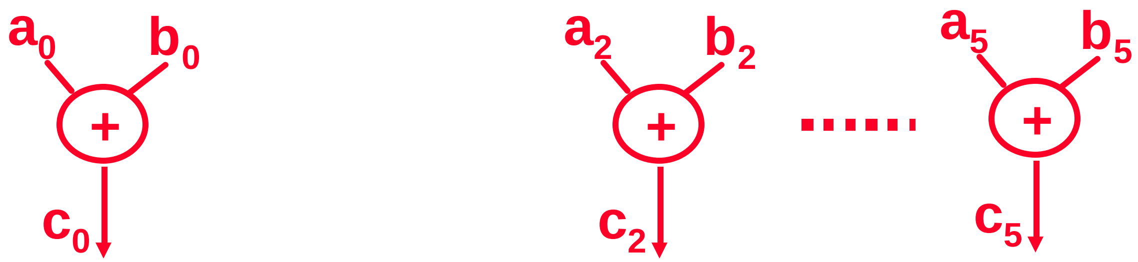
FOR k

$$C[i,j] = C[i,j] + A[i,k] * B[k,j]$$

- **Looking to find parallelism...**

Choosing a program representation...

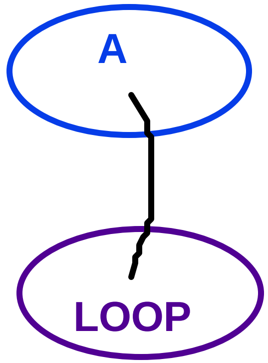
- Dataflow graph



  - No notion of storage ________________ problem
  - Data values flow along arcs
  - Nodes represent operations

Compiler representation

- For certain kinds of structured programs



Data array

Index expressions

LOOP nest

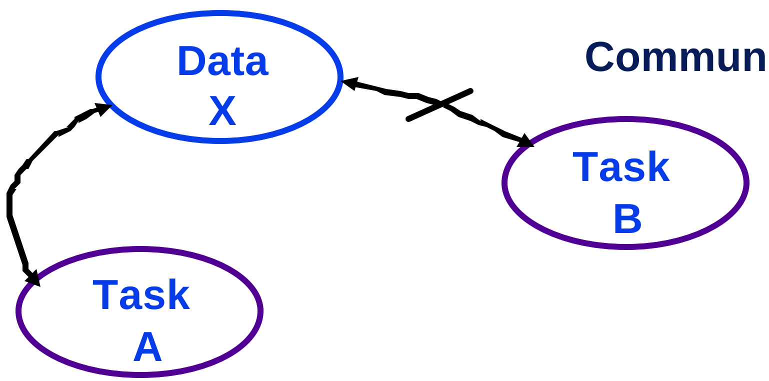
Array

$A[i, j]$

FORALL i ——

  FORALL j ——

- Unstructured programs

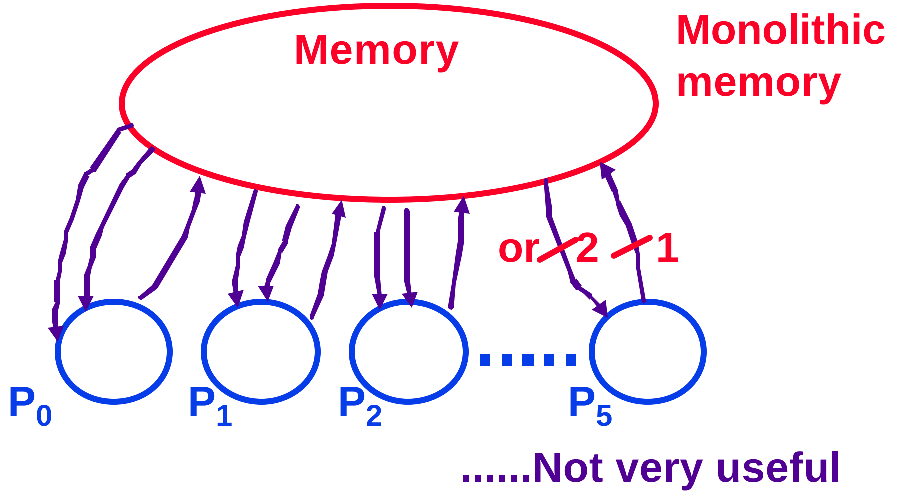


Communication weight

Process reference graph

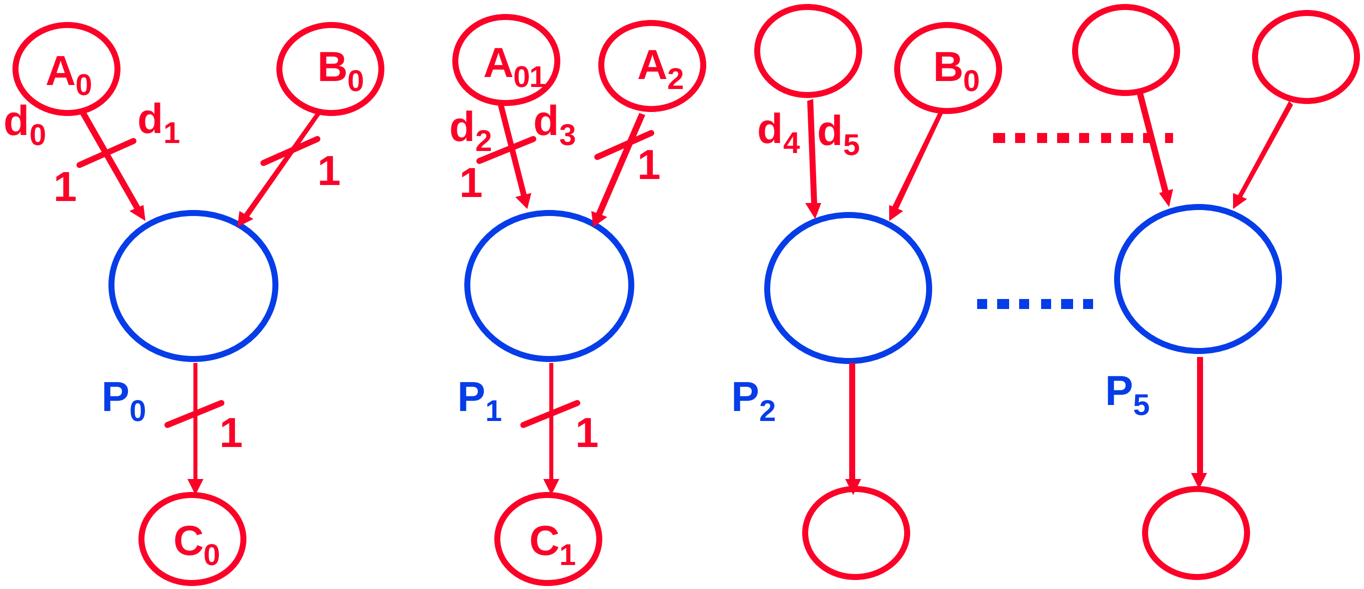
FORALL i FROM 0 to 5

$$C[i] = A[i] + B[i]$$



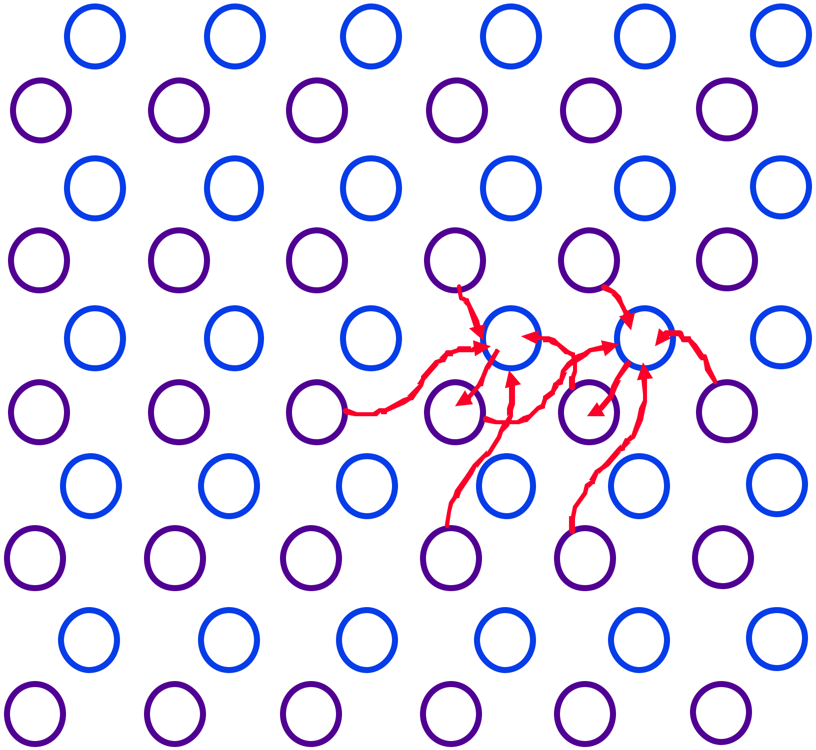
- Nodes represent threads (processes) computation
- Edges represent communication (memory references)
- Can attach weights on edges to represent volume of communication
- Extension: precedence relations edges can be added too
- Can also try to represent multiple loop produced threads as one node

Process communication graph



- Allocate data items to nodes as well
- Nodes: Threads, data objects
- Edges: Communication
- Key: Works for both shared-memory, object-oriented, and dataflow systems! (Msg. passing)

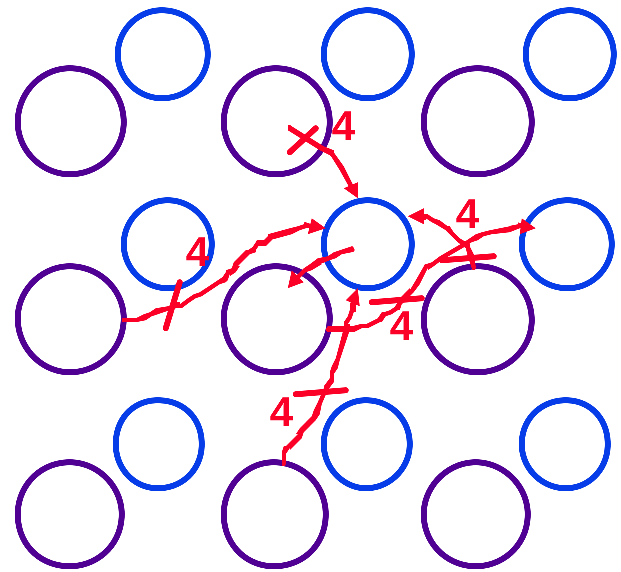
PCG for Jacobi relaxation



Fine PCG

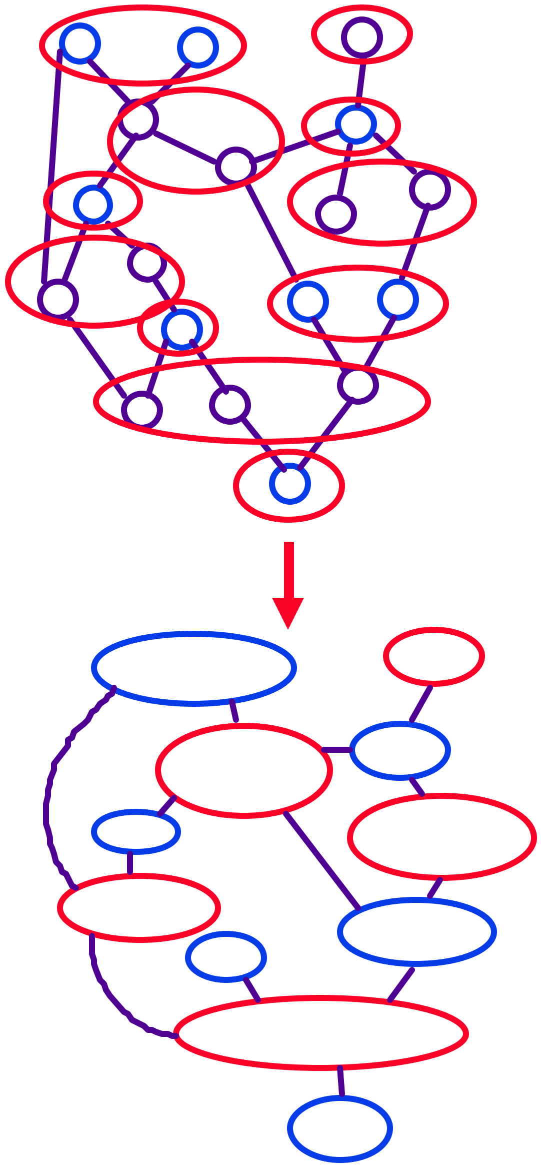
: Computation

: Data



Coarse PCG

Compilation with PCGs

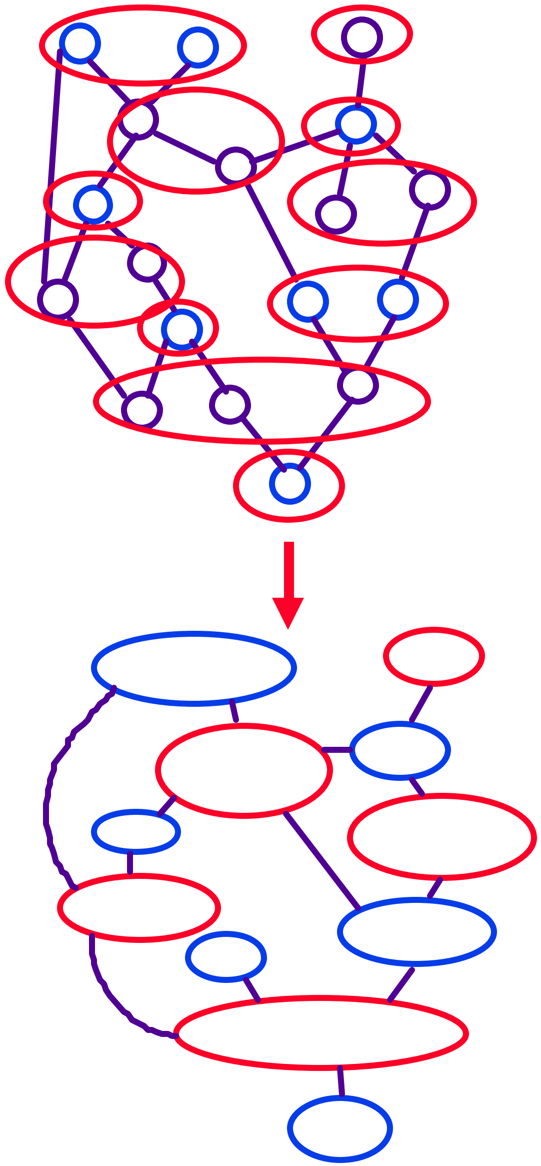


Fine process communication graph

Partitioning

Coarse process communication graph

Compilation with PCGs

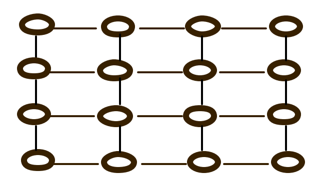


Fine process communication graph

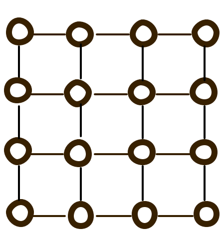
Partitioning

Coarse process communication graph

MP:



Placement



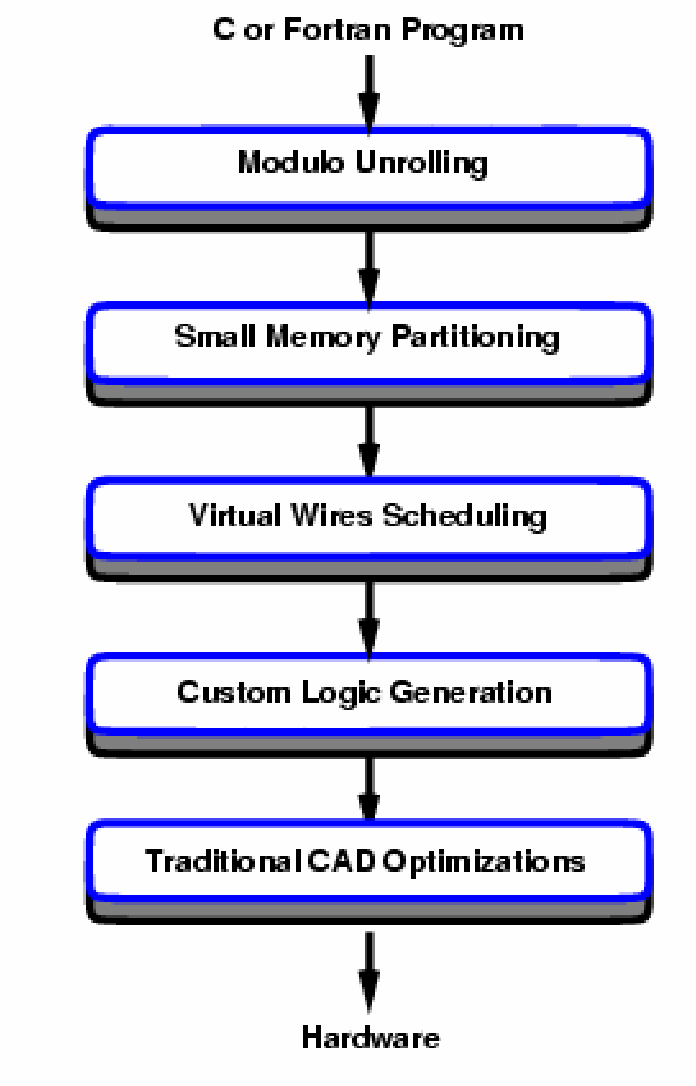
Coarse process communication graph
... other phases, scheduling.
Dynamic?

Parallel Compilation

- Consider loop partitioning
- Create small local compilation
- Consider static routing between tiles
- Short neighbor-to-neighbor communication
- Compiler orchestrated

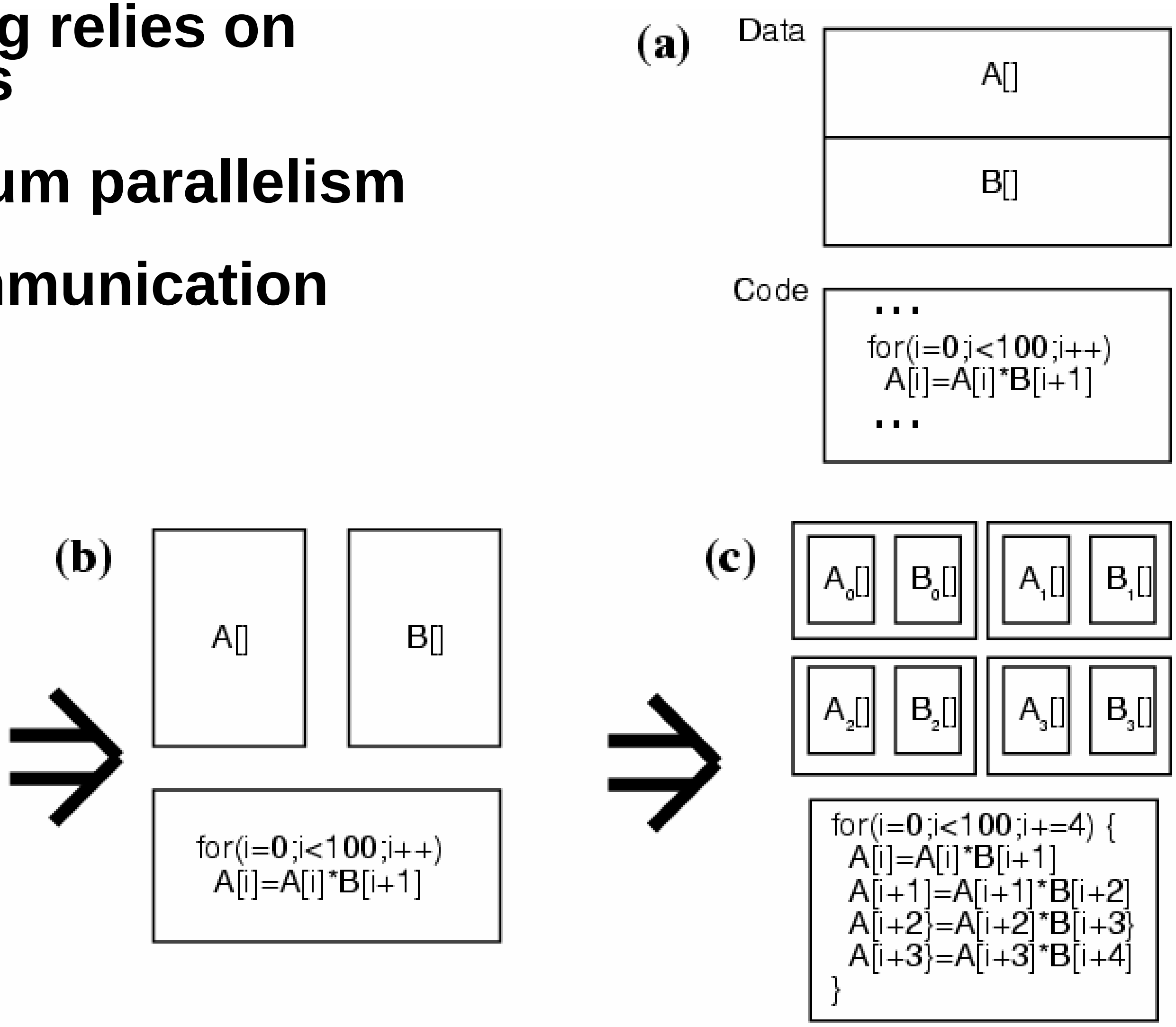
Flow Compilation

- Modulo unrolling
- Partitioning
- Scheduling



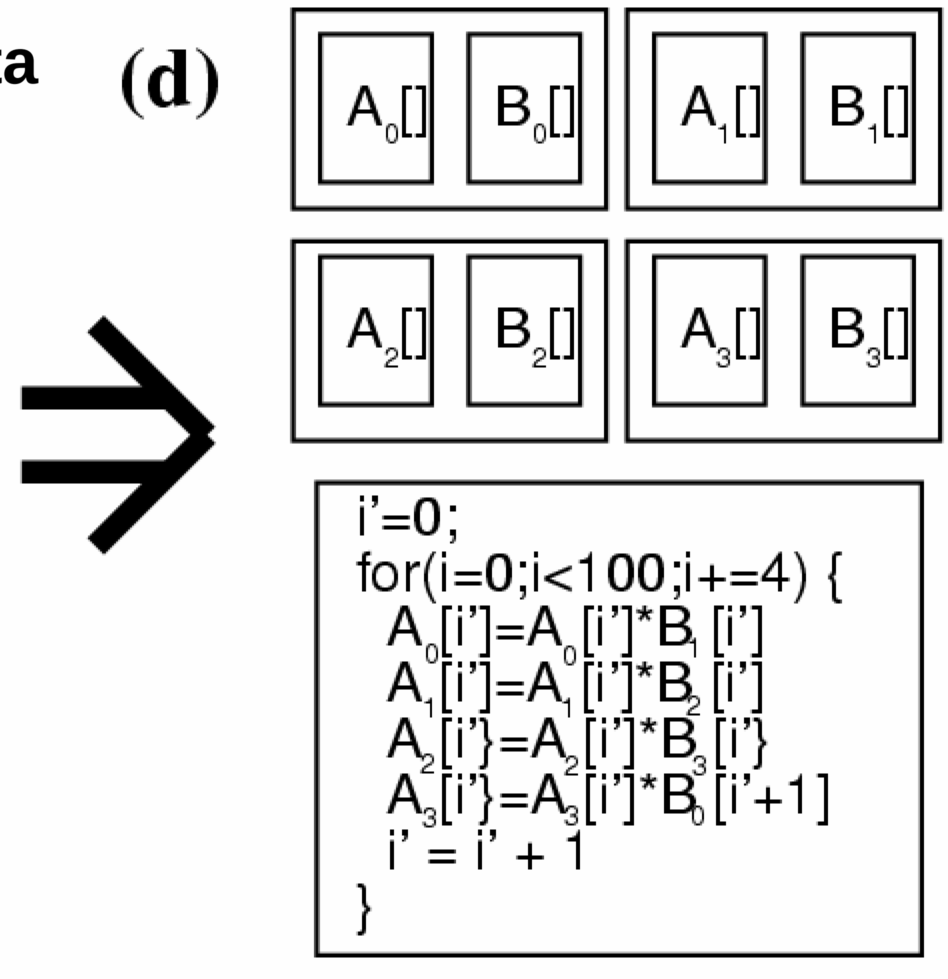
Modulo Unrolling – Smart Memory

- Loop unrolling relies on dependencies
- Allow maximum parallelism
- Minimize communication



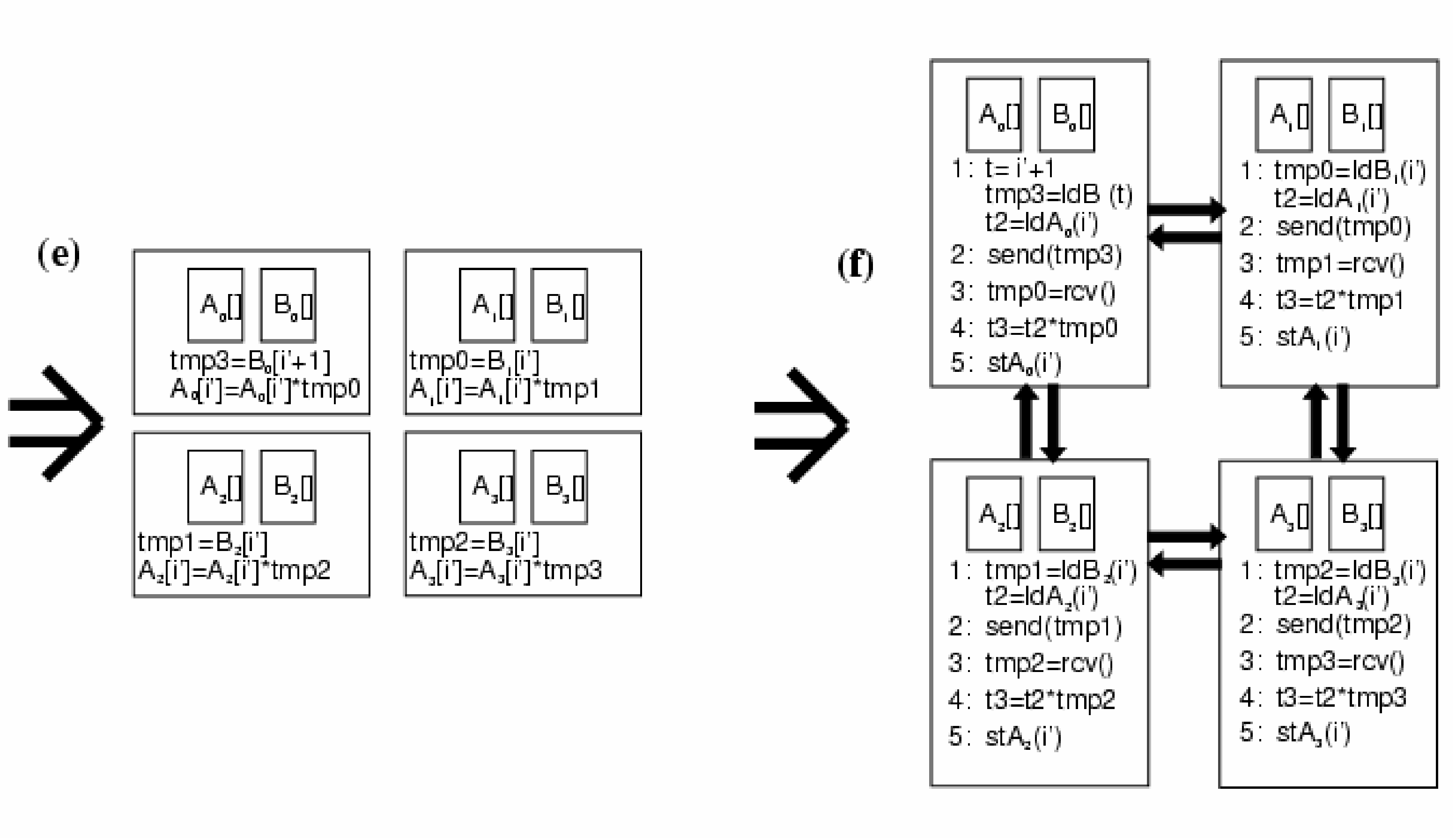
Array Partitioning – Smart Memory

- **Assign each line to separate memory**
- **Consider exchange of data**
- **Approach is scalable**



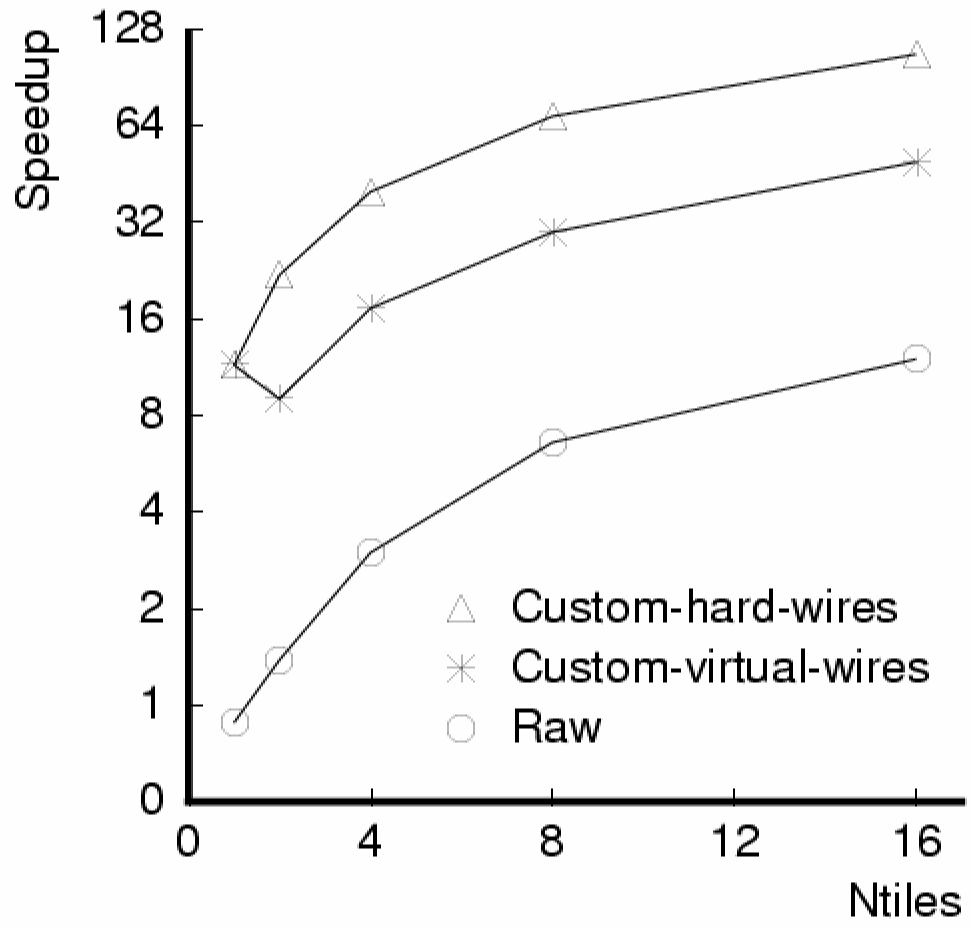
Communication Scheduling – Smart Memory

- Determine where data should be sent
- Determine when data should be sent



Speedup for Jacobi – Smart Memory

- **Virtual wires indicates scheduled paths**
- **Hard wires are dedicated paths**
- **Hard wires require more wiring resources**
- **RAW is a parallel processor from MIT**

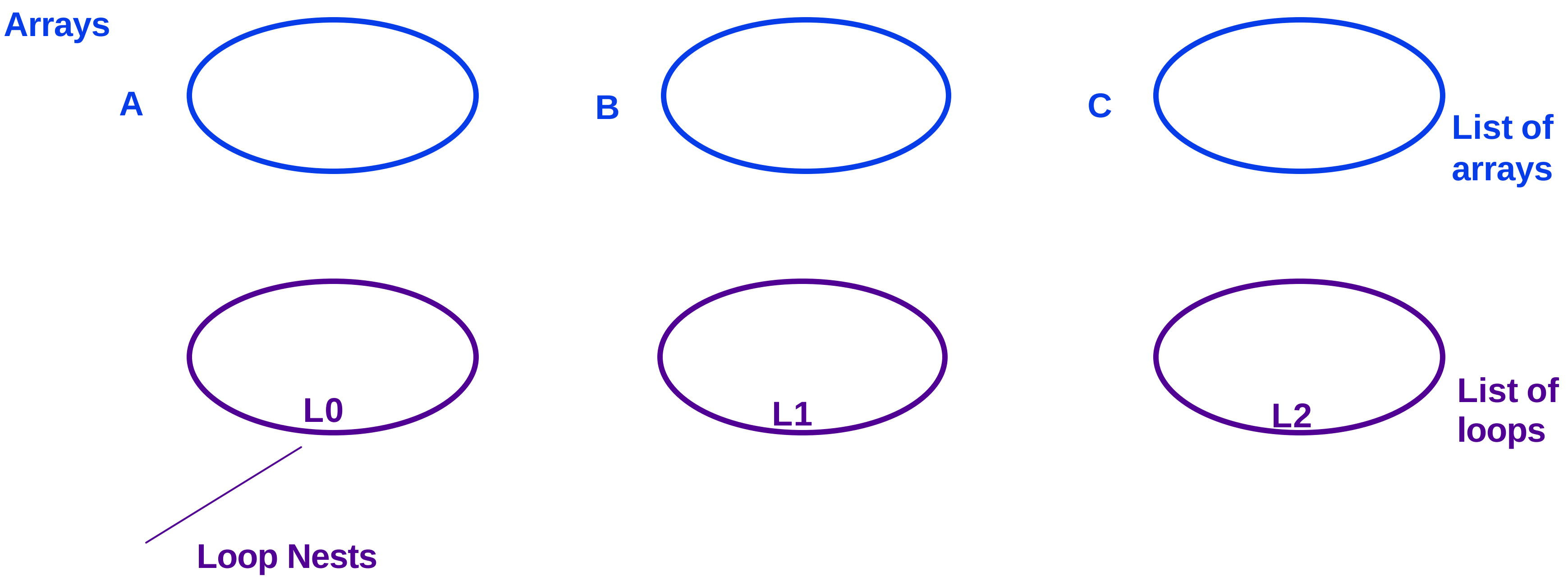


Speedup scalability for jacobi

Partitioning

- Use heuristic for unstructured programs
- For structured programs...

...start from:



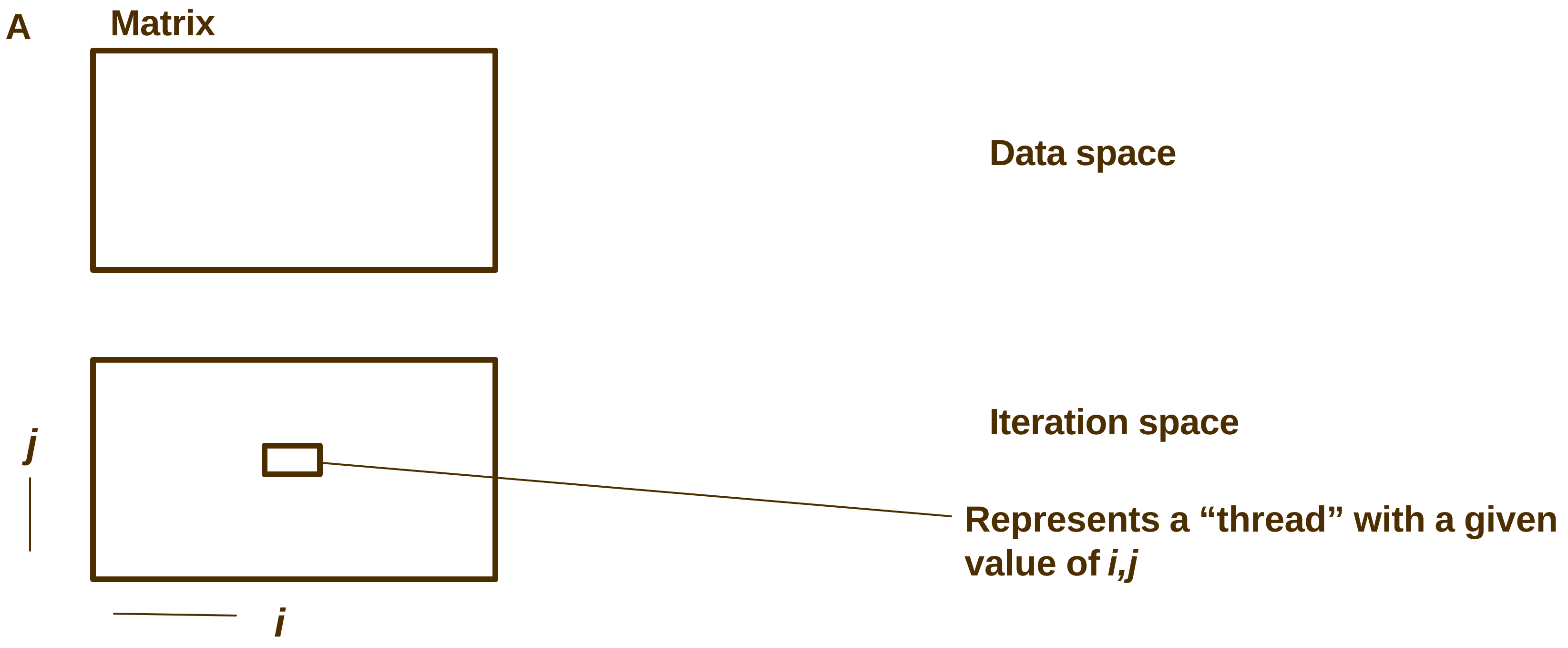
Notion of Iteration space, data space

E.g.

Forall i

Forall j

$$A[i, j] = A[i + 2, j + 1] + A[i, j + 1]$$



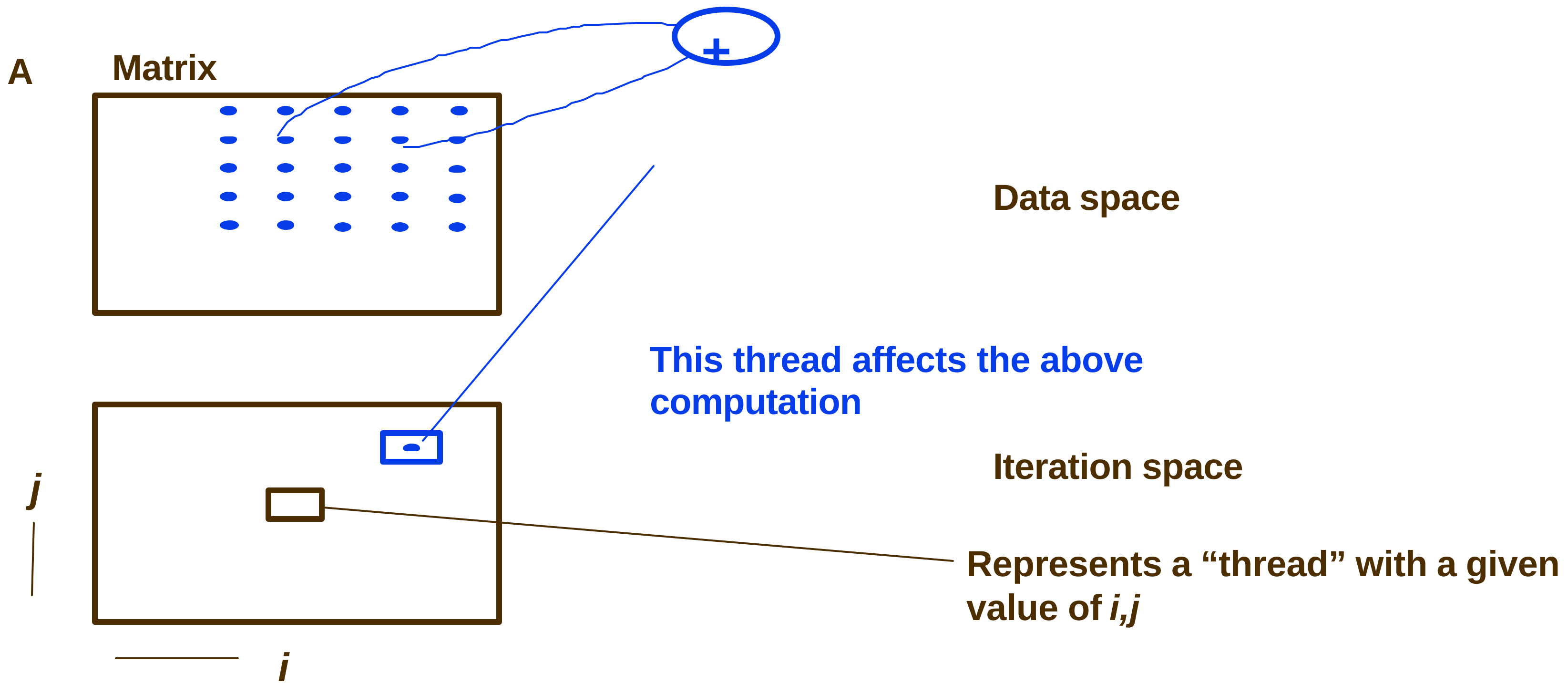
Notion of Iteration space, data space

E.g.

Forall i

Forall j

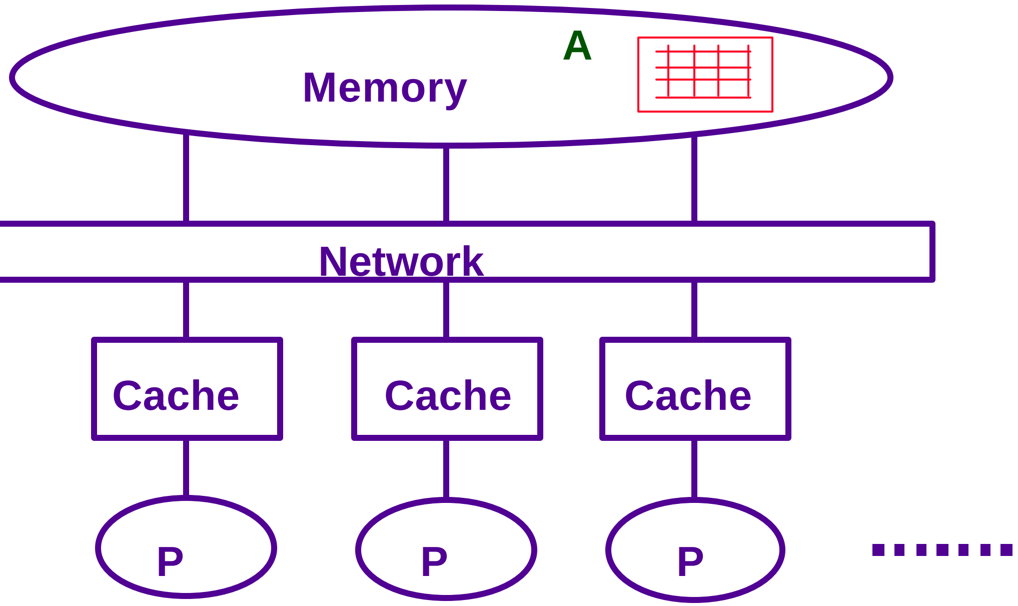
$$A[i,j] = A[i+2, j+1] + A[i, j+1]$$



- **Partitioning: How to "tile" iteration for MIMD M/Cs data spaces?**

Loop partitioning for caches

° **Machine model**



- **Assume all data is in memory**
- **Minimize first-time cache fetches**
- **Ignore secondary effects such as invalidations due to writes**

Summary

- Parallel compilation often targets block based and loop based parallelism
- Compilation steps address identification of parallelism and representations
  - Graphs often useful to represent program dependencies
- For static scheduling both computation and communication can be represented
- Data positioning is an important for computation