Parallel Programming Patterns
or
How to Divvy up the PIE

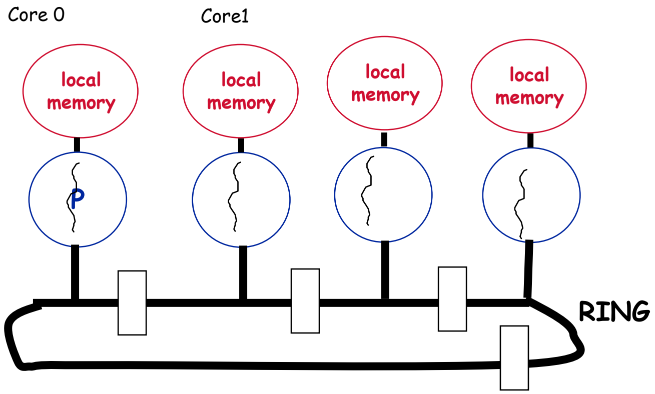
Concepts in
- Partitioning
- Load balancing
- Placement
- Locality

Agarwal

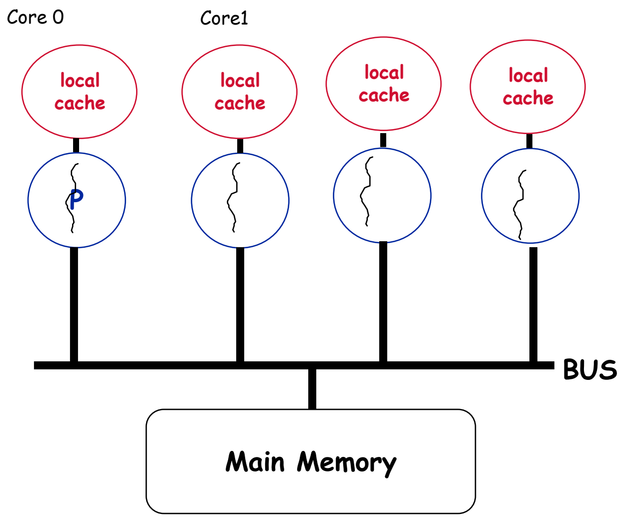
6.173
Fall 2010
L04

First, More on the Message Passing Model

Message Passing Machine Model



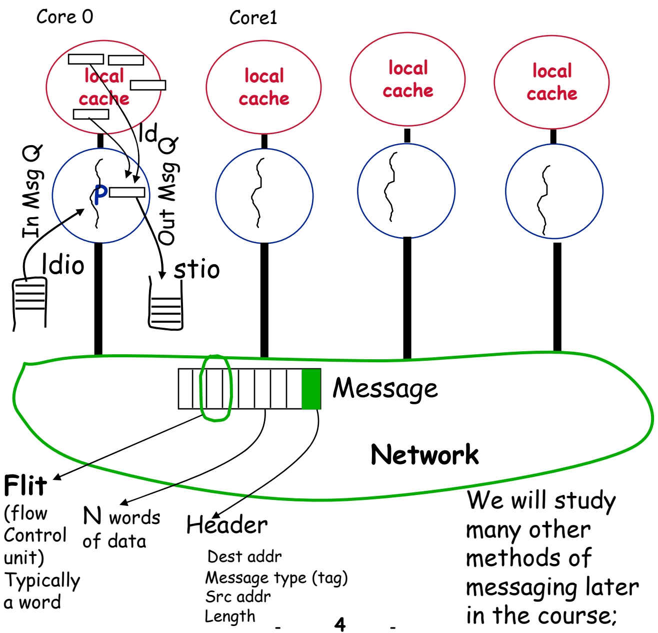
Message Passing Machine Model



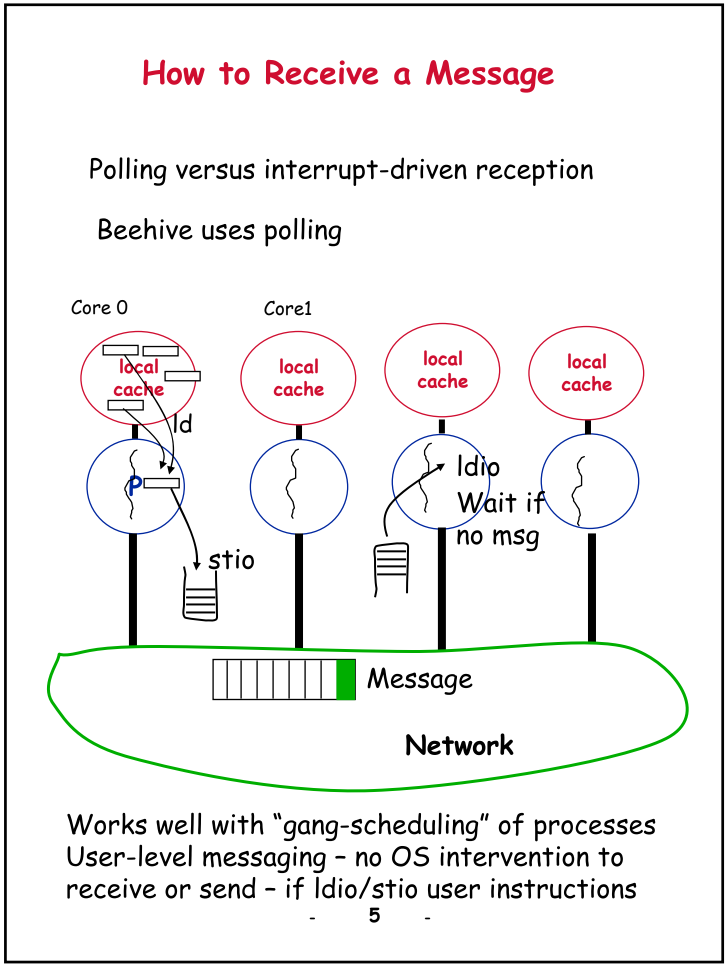
How to Send a Message

In Beehive, message interface
uses read/writes to I/O space

But first, “gather” from mem



We will study many other methods of messaging later in the course; discuss

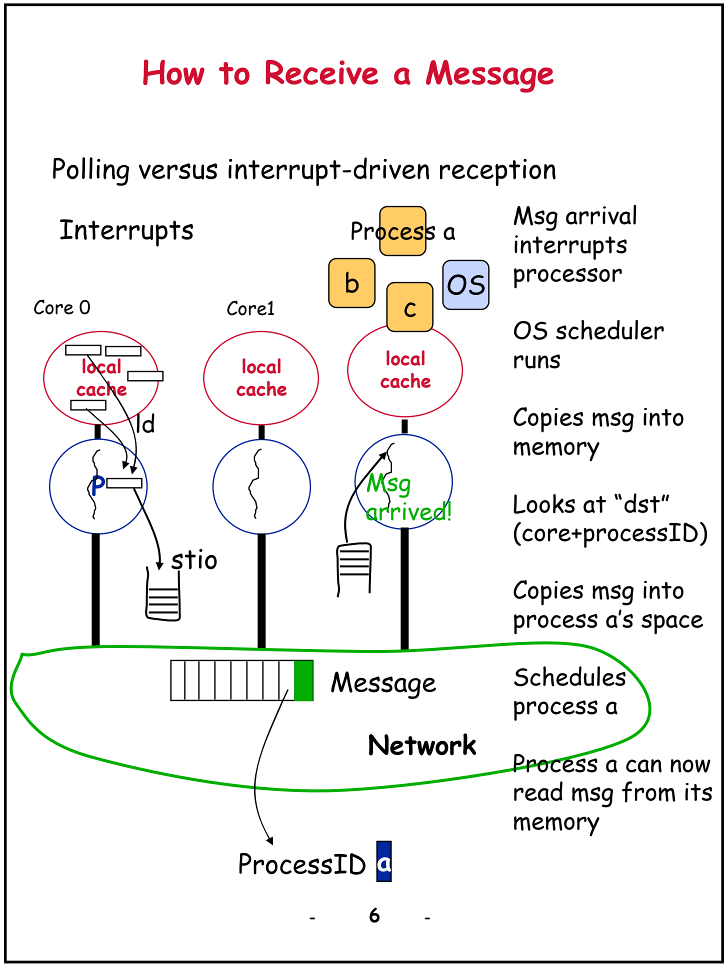


How to Receive a Message

Polling versus interrupt-driven reception

Beehive uses polling

Works well with "gang-scheduling" of processes
User-level messaging – no OS intervention to receive or send – if ldio/stio user instructions



How to Receive a Message

Polling versus interrupt-driven reception

Interrupts

Msg arrival interrupts processor

OS scheduler runs

Copies msg into memory

Looks at "dst" (core+processID)

Copies msg into process a's space

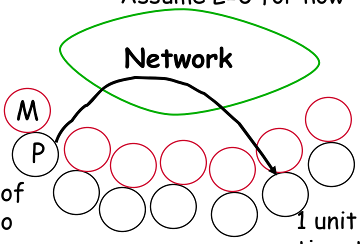
Schedules process a

Process a can now read msg from its memory

Message Passing Algorithm Model

Postal model* for message passing

In general, L units of time in transit.
Assume L=0 for now



1 unit of time to send a word

1 unit of time to receive a word

*[Bar-Noy & Kipnis, SPAA 1992]

Next, How to Break up Problems into Parallel Tasks

Depends on the problem and user requirements

Two major approaches **Aka**

- Data partitioning

  Stripe (Lampson)
  Run to completion (networking)
  Data parallel
  Map

- Instruction (or program) partitioning

  Stream (Lampson)
  Pipelining (networking)

…under postal model

How to Break up Problems into Parallel Tasks

Let's tackle data partitioning first

- Data partitioning

  We will also learn about load balancing, communication volume, and locality along the way

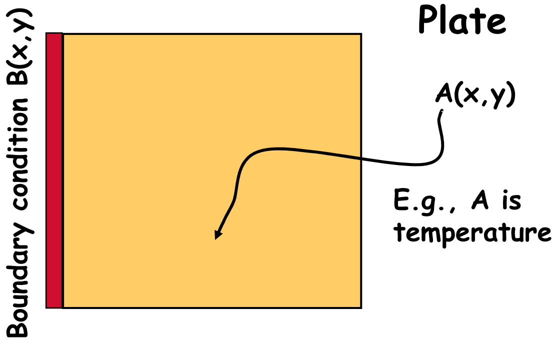
- Instruction (or program) partitioning

Data Partitioning Applies to Most Problems

- Climate modeling
- Heat transfer
- Solving partial differential equations
- Face recognition
- Speech processing
- Finite element solutions
- Fluid flow
- Structure modeling in civil and mechanical engineering
- Networking
  - Deep packet inspection
  - Network routing
  - Switching
- Network security
  - Firewalls
  - Encryption
  - Virus checking
- Genomics
- Data mining
- Web servers and web caching
- Databases
- Travelling salesman problem
- Circuit simulation
- Particle dynamics

An Example Problem

Heat diffusion

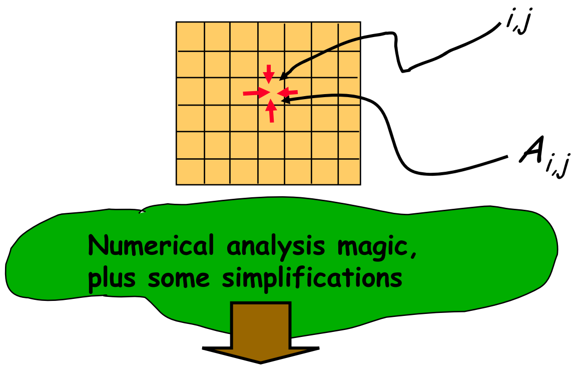


$$\nabla^2 A + B = 0$$ Poisson's equation

Question: Find the steady state A(x,y) at each point on the plate

Iterative Jacobi Solution Method

Discretize



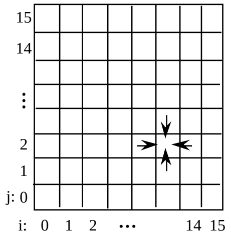
Next value

Previous iteration values

$$A_{i,j} = \frac{A_{i+1,j} + A_{i-1,j} + A_{i,j+1} + A_{i,j-1}}{4}$$

Iterate till no change.
The ultimate parallel method.
Jacobi is the most commonly used parallel app!

Now, Getting to the Point… Parallel Implementation



Remember

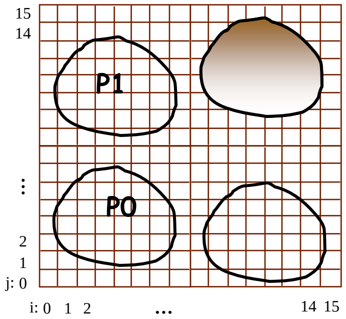
$$A_{i,j} = \frac{\downarrow + \rightarrow + \leftarrow + \uparrow}{4}$$

Q: How would you partition the problem? I.e., who does what.

- Say, on 16 processors?

Communication?

Partitioning and Communication



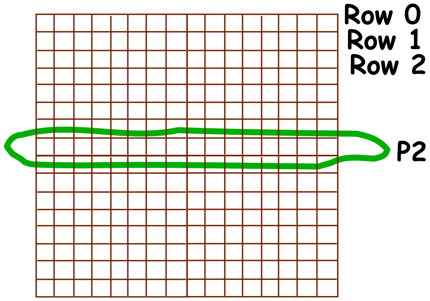
Implies, these data items are kept in P3's local memory, and P3 handles all the computation related to updating these values

Assume 16 processors
Useful to keep the problem picture in mind

Discuss
Issue of *load balancing*

Load balancing: each processor does the same amount of work (compute+comms) – achieved by equal area partitions

Partitioning

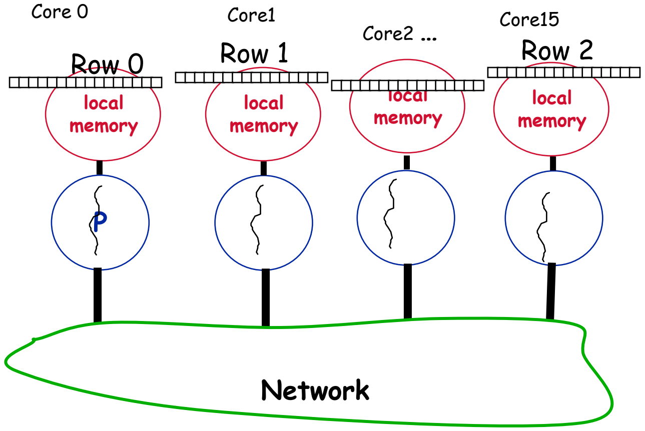


E.g., row-wise partitioning

Load balanced (static)

We will see dynamic load balancing later

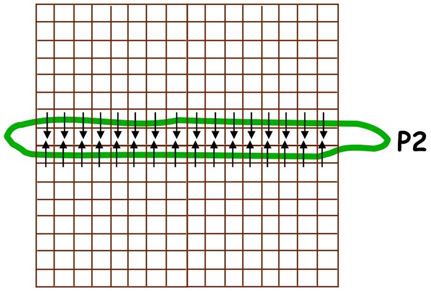
Placement



Above is an example of random placement of rows on cores

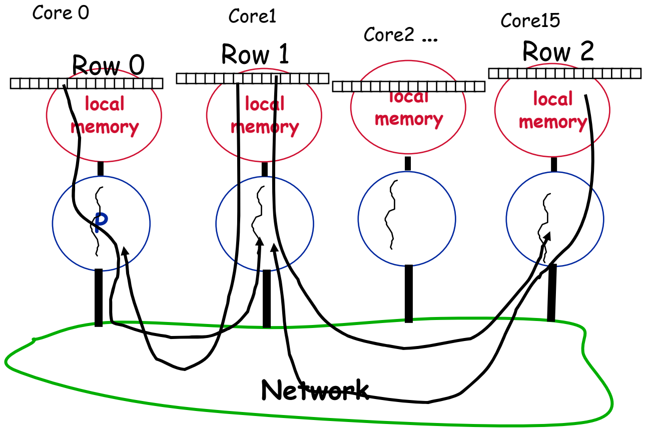
Assume, if $A_{i,j}$ is placed on Core1.
Then, Core1 will handle the update of $A_{i,j}$

Partitioning and Communication



What is the communication pattern?

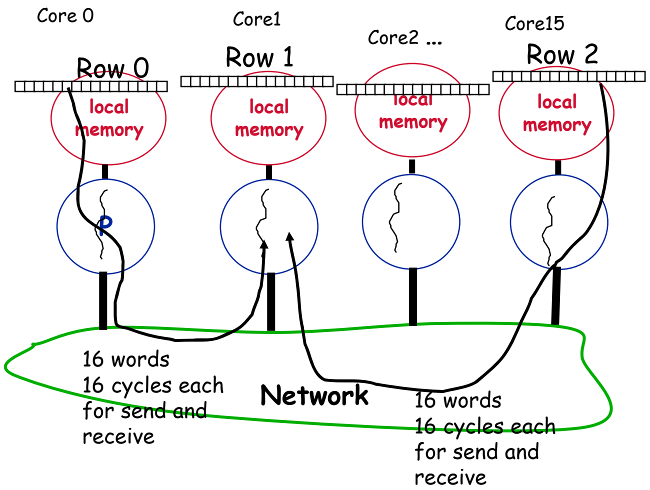
Communication Pattern



Communication pattern: Send and receive rows

Focus on Core1...

Row-Wise Partitioning



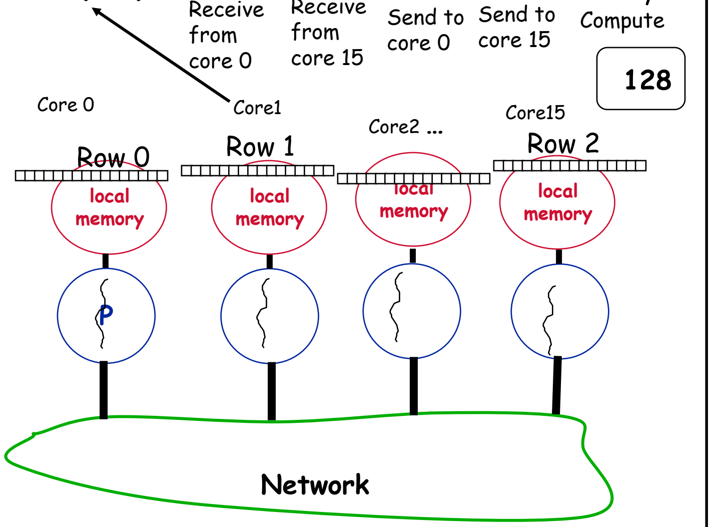
Using the postal model we can compute the communication time

Next, including both the compute time and communication time (assuming each arithmetic op takes 1 cycle), we can compute runtime per iteration

Runtime for Row-Wise Partitioning

T(iter) = 16 + 16 + 16 + 16 + 16x4 cycles

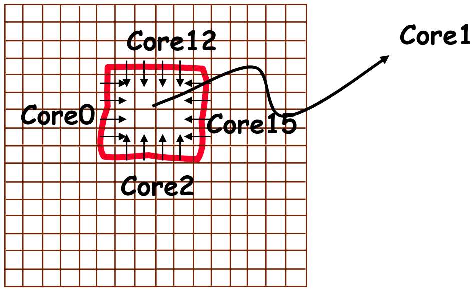
Receive from core 0 | Receive from core 15 | Send to core 0 | Send to core 15 | Compute



Including both the compute time and communication time (assuming each arithmetic op takes 1 cycle), we can compute runtime per iteration

Can we do better?

Communication-Minimized Partitioning



Blocking or tiling

Minimizing perimeter given a constant area

(minimum comm) (for load balance)

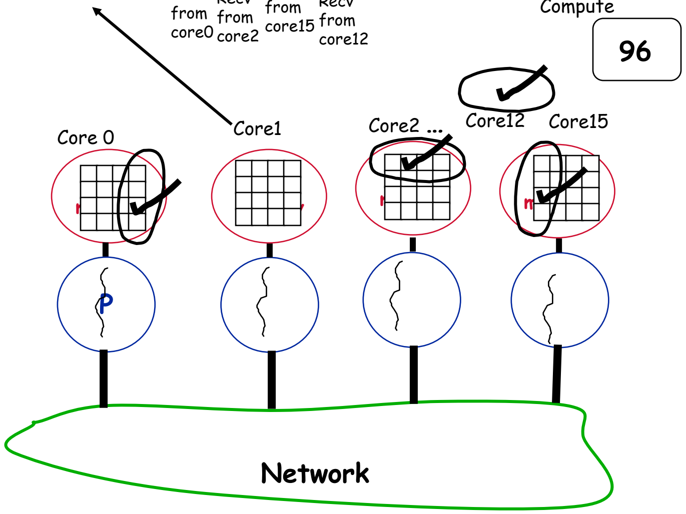
Tradeoffs in startup cost versus total comm volume

Runtime for Tiled Partitioning

T(iter) = 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 16x4 cycles

Recv from core0, Recv from core2, Recv from core15, Recv from core12, Send to core0, Send to core2, Send to core15, Send to core12, Compute

96



Including both the compute time and communication time (assuming each arithmetic op takes 1 cycle), we can compute runtime per iteration

Are there any conditions under which row-wise partitioning is better?

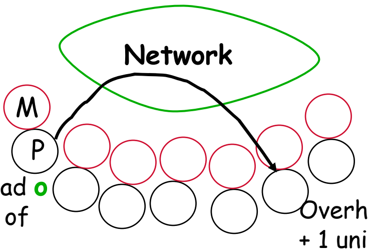
Another Message Passing Algorithm Model

Suppose messages have a large, but fixed, sending or receiving overhead.

Captured by the logp model for message passing

[PPoPP 1993]

In general, L units of time in transit.
Assume L=0 for now



Overhead **o** + 1 unit of time to send a word

E.g., o = 20

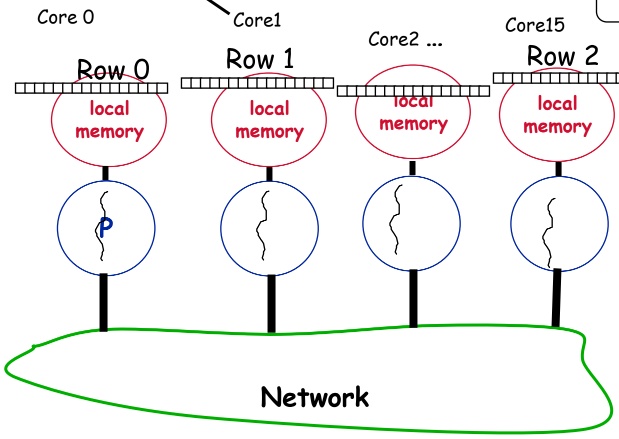
Overhead **o** + 1 unit of time to receive a word

Need to make a tradeoff in startup overhead versus total comm volume

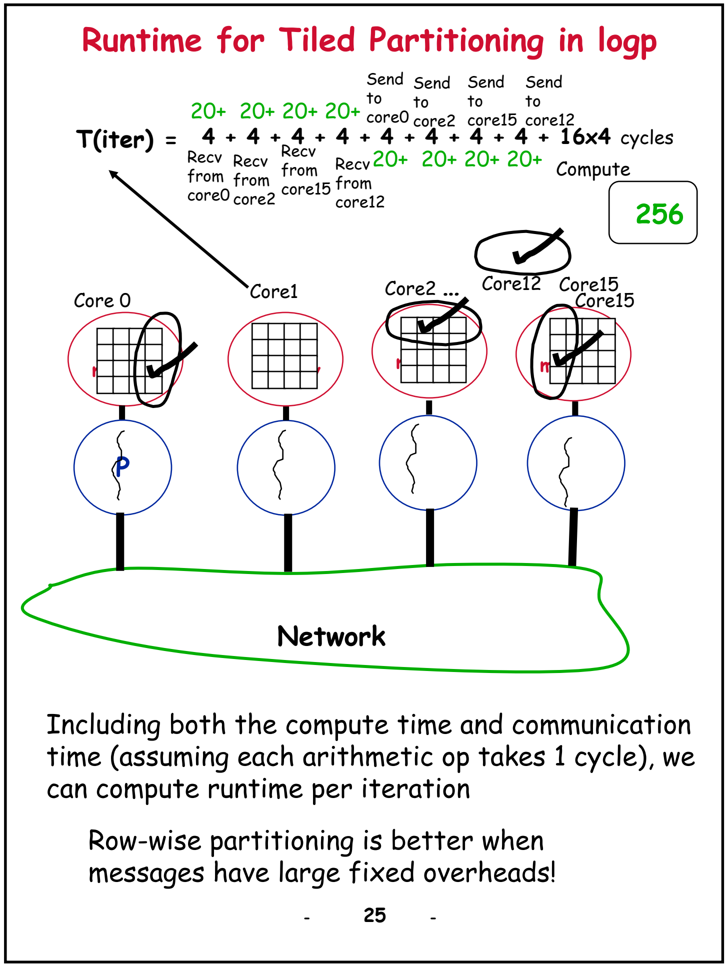
Runtime for Row-Wise Partitioning in logp

$$T(iter) = \overset{20+}{\underset{\text{Receive from core 0}}{16}} + \overset{20+}{\underset{\text{Receive from core 15}}{16}} + \overset{20+}{\underset{\text{Send to core 0}}{16}} + \overset{20+}{\underset{\text{Send to core 15}}{16}} + \underset{\text{Compute}}{16x4} \text{ cycles}$$

$$\boxed{208}$$



Including both the compute time and communication time (assuming each arithmetic op takes 1 cycle), we can compute runtime per iteration



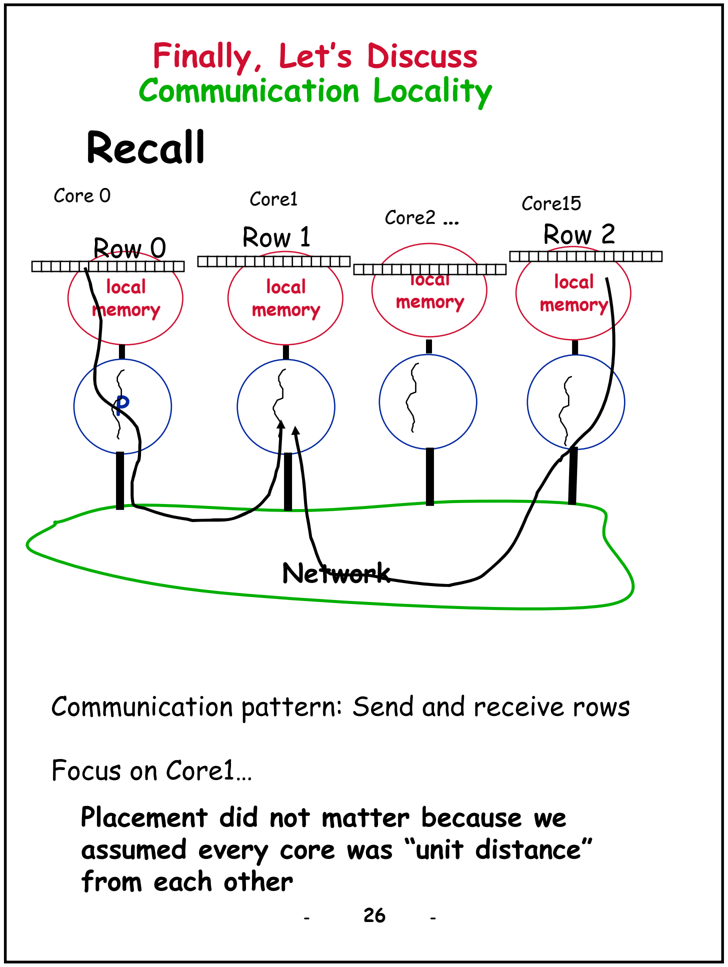
Runtime for Tiled Partitioning in logp

$$T(iter) = \underset{\text{Recv from core0}}{\overset{20+}{4}} + \underset{\text{Recv from core2}}{\overset{20+}{4}} + \underset{\text{Recv from core15}}{\overset{20+}{4}} + \underset{\text{Recv from core12}}{\overset{20+}{4}} + \overset{\text{Send to core0}}{\underset{20+}{4}} + \overset{\text{Send to core2}}{\underset{20+}{4}} + \overset{\text{Send to core15}}{\underset{20+}{4}} + \overset{\text{Send to core12}}{\underset{20+}{4}} + \underset{\text{Compute}}{16x4} \text{ cycles}$$

$\boxed{256}$

Including both the compute time and communication time (assuming each arithmetic op takes 1 cycle), we can compute runtime per iteration

Row-wise partitioning is better when messages have large fixed overheads!



Finally, Let's Discuss Communication Locality

Recall

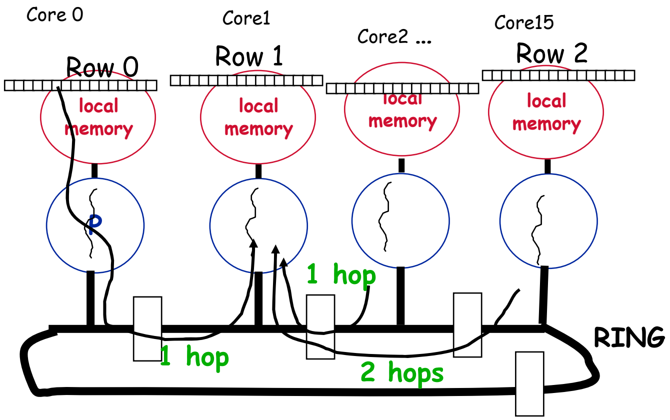
Communication pattern: Send and receive rows

Focus on Core1…

Placement did not matter because we assumed every core was "unit distance" from each other

Communication Locality

What if we replaced ideal communication network with ring



Communication pattern: Send and receive rows

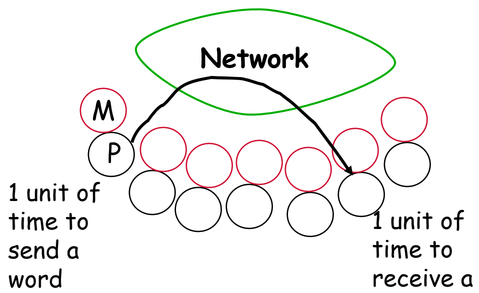
Focus on Core1...

Better to place Row 2 on Core2

Communication Locality

Deos non-zero L in postal model capture communication locality?

In general, L units of time in transit.



1 unit of time to send a word

1 unit of time to receive a word

Nope!

Neither does logp.

Shortcoming of both the postal and logp models.

Need a new spatial algorithmic model.
Nice PhD thesis topic!

Back to
How to Break up Problems
into Parallel Tasks

- Data partitioning

Next, let's tackle instruction partitioning

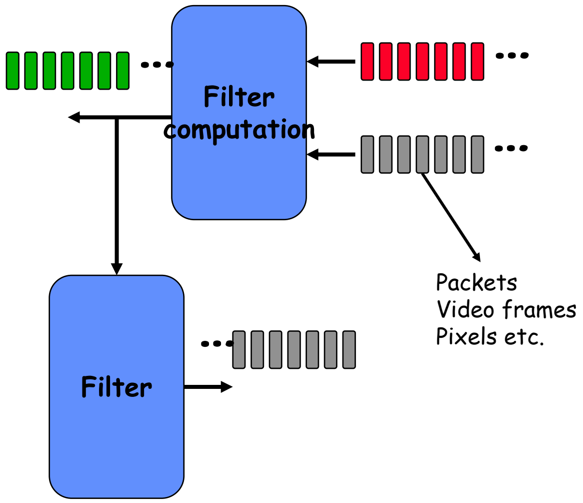
- Instruction (or program) partitioning

Aka pipeling partitioning, or stream partitioning

We will also discuss dynamic approaches to load balancing

Instruction Partitioning Works Well for Stream Problems

Characterized by "eternal" data streams.
Filter style computation on these streams.
Computation times can vary



Stream Application Areas

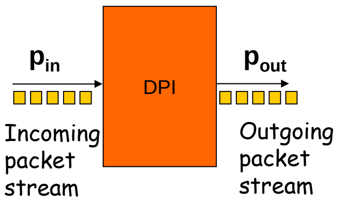
- FIR filters
  - Select a channel in wireless comms
  - Audio filtering
  - Channel selection
- Modems to modulate/demodulate signals
  - Cable modems
  - Cell phones
  - Wireless cards
- Compression
- Search in text and video streams
- Beamforming
  - Directional wireless antennas
  - Tetherless microphones
  - Jammer cancellation
- Video stream computations
- Graphics
- Networking packet streams
  - IP Routing
  - Packet classification
  - Server load balancing (SLB)
- Networking security
  - Deep packet inspection
  - Spam filtering
  - Firewalls

A Stream Application

Networking – Packet Filter using Deep Packet Inspection (DPI)

A Networking Application

Packet Filter using Deep Packet Inspection (DPI)



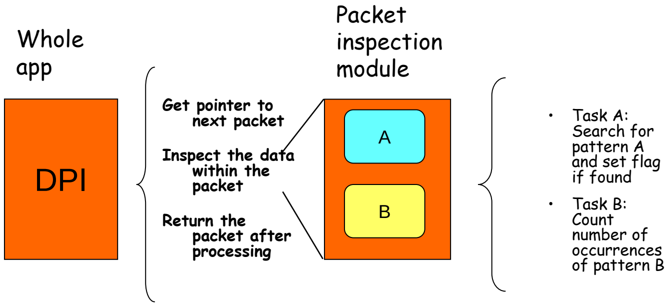
Often, can assume that the processing of each packet is completely independent from the other packets

Other times, in “stateful” applications, the packets in each “flow” are dependent

Similar model used for intrusion prevention systems, spam filters, network performance monitors, firewalls, UTMs, network services, encryption, etc.

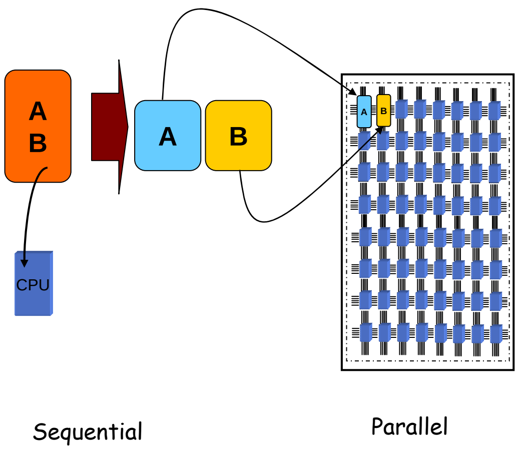
Packet Filter using Deep Packet Inspection (DPI)

Assume that incoming packets have been put somewhere in memory by the packet I/O interface


Whole app
Packet inspection module
DPI
Get pointer to next packet
Inspect the data within the packet
Return the packet after processing
A
B
Task A: Search for pattern A and set flag if found
Task B: Count number of occurrences of pattern B

Similarly, assume that packets must be placed in memory after processing. From there, the packets are sent out on the wire by the packet I/O interface

Instruction Partitioning aka Pipelined



Packet processing for each packet is pipelined across multiple processes (or threads) in a multicore chip with a mesh network (see more detail later in the course)

Instruction Partitioning

First look for data partitioning!
Most apps are data parallel
E.g., multiple network flows
E.g., multiple video streams
E.g., multiple cellphone calls

Pipeline parallelism is harder to code. Trust me!

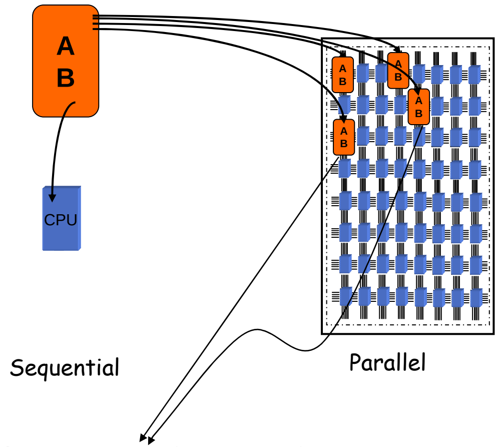
Experience has shown that pipelined partitioning useful in following cases:

1. You want to speed up one instance/flow/stream/call
   E.g., to hit 20Gbps for 1 TCP flow
   E.g., to hit realtime 1080p mainprofile
   E.g., reduce latency for each call

2. You want to get additional speedup after you have done data partitioning
   Hybrid approaches… next

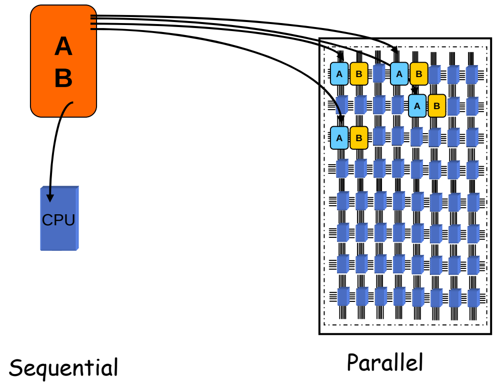
Data Partitioning Approach in Networking

Packet inspection module



Each process (or thread) operates on a separate packet from start to completion

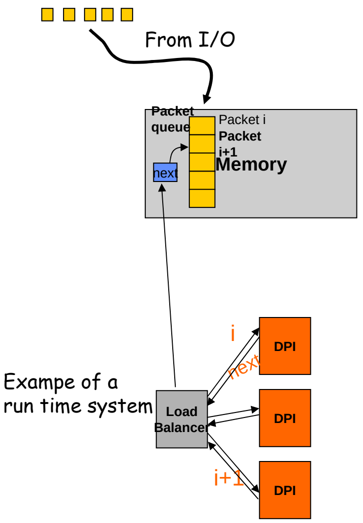
Hybrid Data Partitioning Program Partitioning Approach



Each group of processes (or thread) operates on a separate packet whose processing is pipelined across multiple processes (or threads)

What about load balancing – cannot do so statically!

Load Balancing

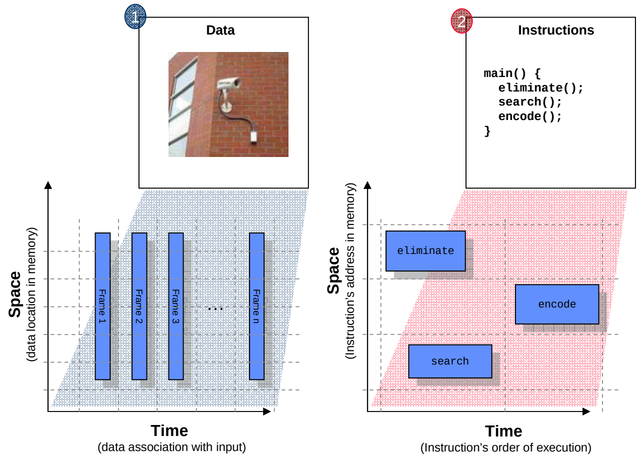


Exampe of a run time system

Assume packets (different lengths) are in memory somehow
E.g., an I/O device stores packets in memory (using DMA – direct memory access)

Generalization of Data and Instruction Partitioning Leads to Following Taxonomy of Partitioning Strategies

Video surveillance example


1
Data
2
Instructions
Space
(data location in memory)
Frame 1
Frame 2
Frame 3
...
Frame n
Time
(data association with input)
Space
(Instruction's address in memory)
eliminate
encode
search
Time
(Instruction's order of execution)

```
main() {
  eliminate();
  search();
  encode();
}
```

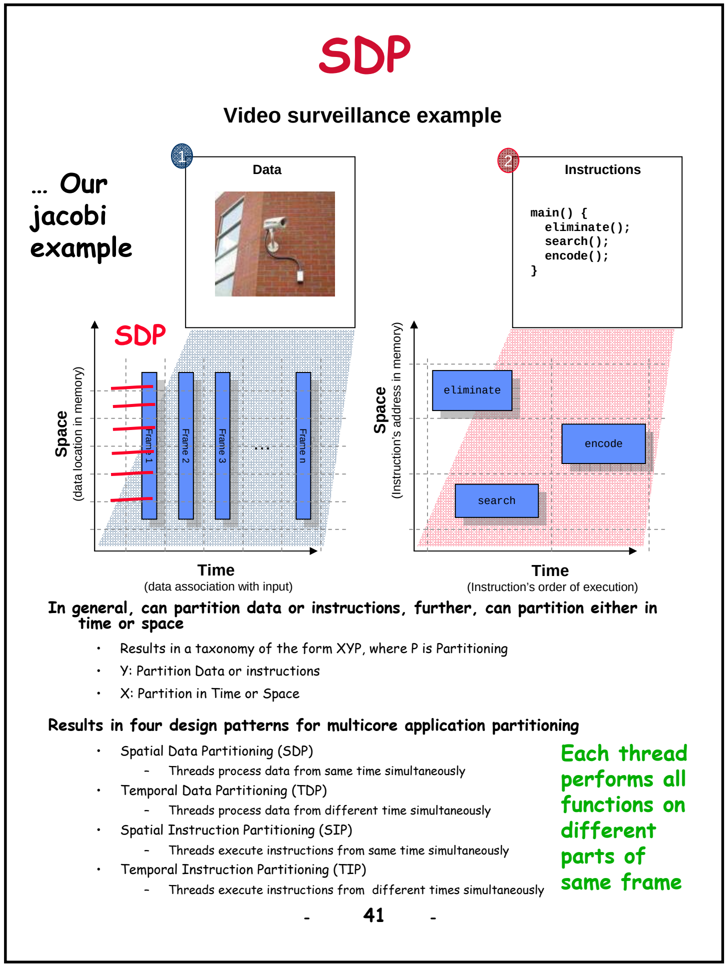
In general, can partition data or instructions, further, can partition either in time or space

- Results in a taxonomy of the form XYP, where P is Partitioning
- Y: Partition Data or instructions
- X: Partition in Time or Space

Results in four design patterns for multicore application partitioning

- Spatial Data Partitioning (SDP)
  - Threads process data from same time simultaneously
- Temporal Data Partitioning (TDP)
  - Threads process data from different time simultaneously
- Spatial Instruction Partitioning (SIP)
  - Threads execute instructions from same time simultaneously
- Temporal Instruction Partitioning (TIP)
  - Threads execute instructions from different times simultaneously

See [Hoffman2010] for details



SDP

Video surveillance example

... Our jacobi example

1
Data
2
Instructions
SDP
Space
(data location in memory)
Frame 1
Frame 2
Frame 3
...
Frame n
Time
(data association with input)
Space
(Instruction's address in memory)
eliminate
encode
search
Time
(Instruction's order of execution)

```
main() {
  eliminate();
  search();
  encode();
}
```

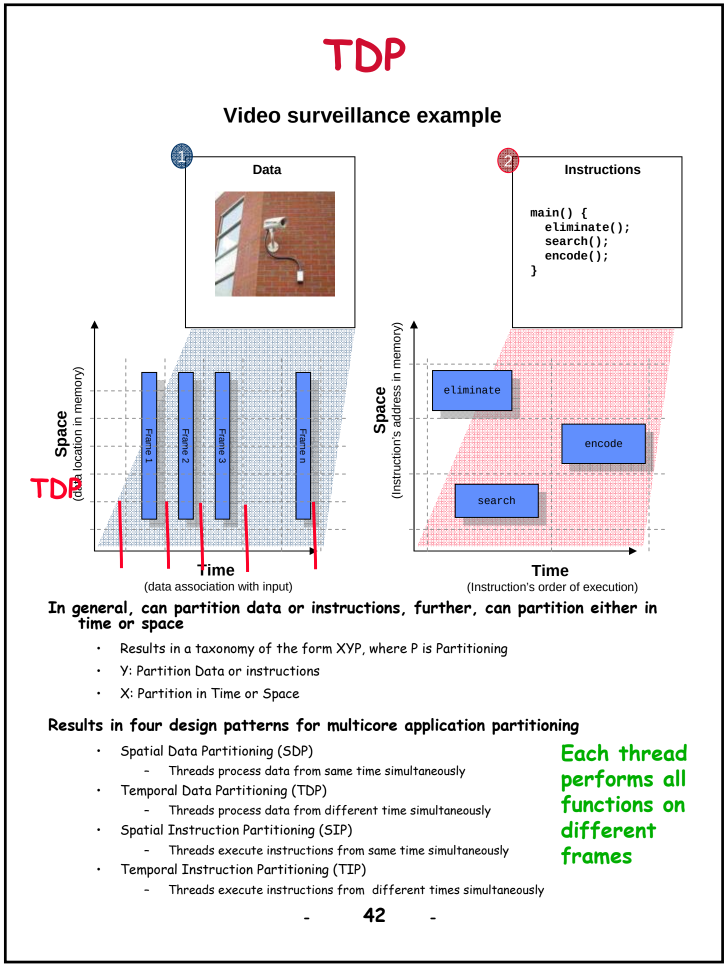
In general, can partition data or instructions, further, can partition either in time or space

- Results in a taxonomy of the form XYP, where P is Partitioning
- Y: Partition Data or instructions
- X: Partition in Time or Space

Results in four design patterns for multicore application partitioning

- Spatial Data Partitioning (SDP)
  - Threads process data from same time simultaneously
- Temporal Data Partitioning (TDP)
  - Threads process data from different time simultaneously
- Spatial Instruction Partitioning (SIP)
  - Threads execute instructions from same time simultaneously
- Temporal Instruction Partitioning (TIP)
  - Threads execute instructions from different times simultaneously

Each thread performs all functions on different parts of same frame



TDP

Video surveillance example

1
Data
2
Instructions
TDP
Space
(data location in memory)
Frame 1
Frame 2
Frame 3
Frame n
Time
(data association with input)
Space
(Instruction's address in memory)
eliminate
encode
search
Time
(Instruction's order of execution)

```
main() {
  eliminate();
  search();
  encode();
}
```

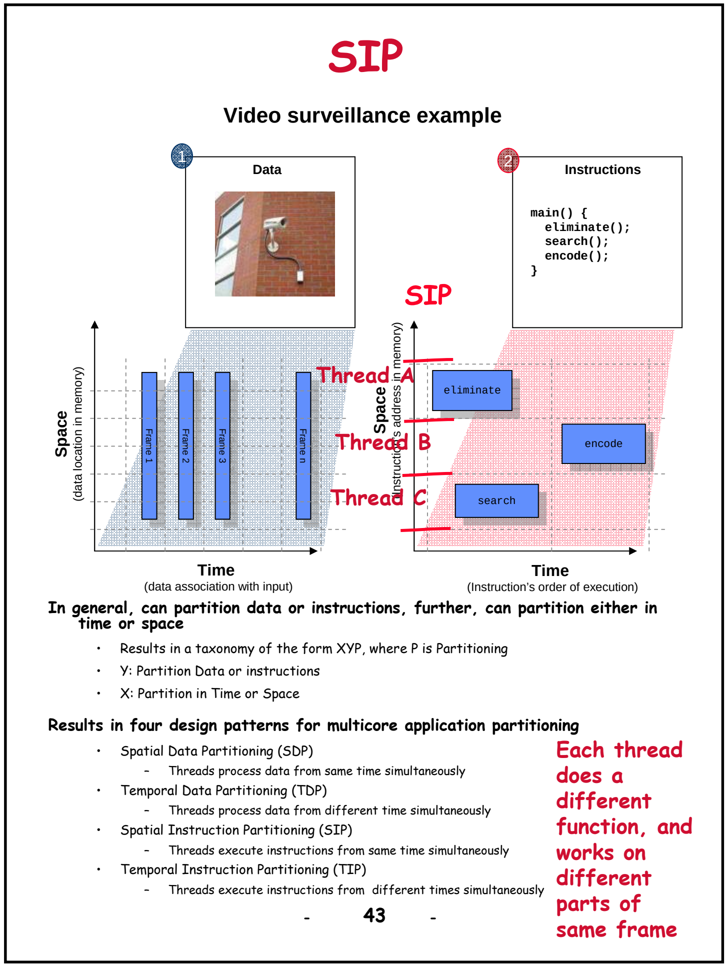
In general, can partition data or instructions, further, can partition either in time or space

- Results in a taxonomy of the form XYP, where P is Partitioning
- Y: Partition Data or instructions
- X: Partition in Time or Space

Results in four design patterns for multicore application partitioning

- Spatial Data Partitioning (SDP)
  - Threads process data from same time simultaneously
- Temporal Data Partitioning (TDP)
  - Threads process data from different time simultaneously
- Spatial Instruction Partitioning (SIP)
  - Threads execute instructions from same time simultaneously
- Temporal Instruction Partitioning (TIP)
  - Threads execute instructions from different times simultaneously

Each thread performs all functions on different frames



SIP

Video surveillance example

In general, can partition data or instructions, further, can partition either in time or space

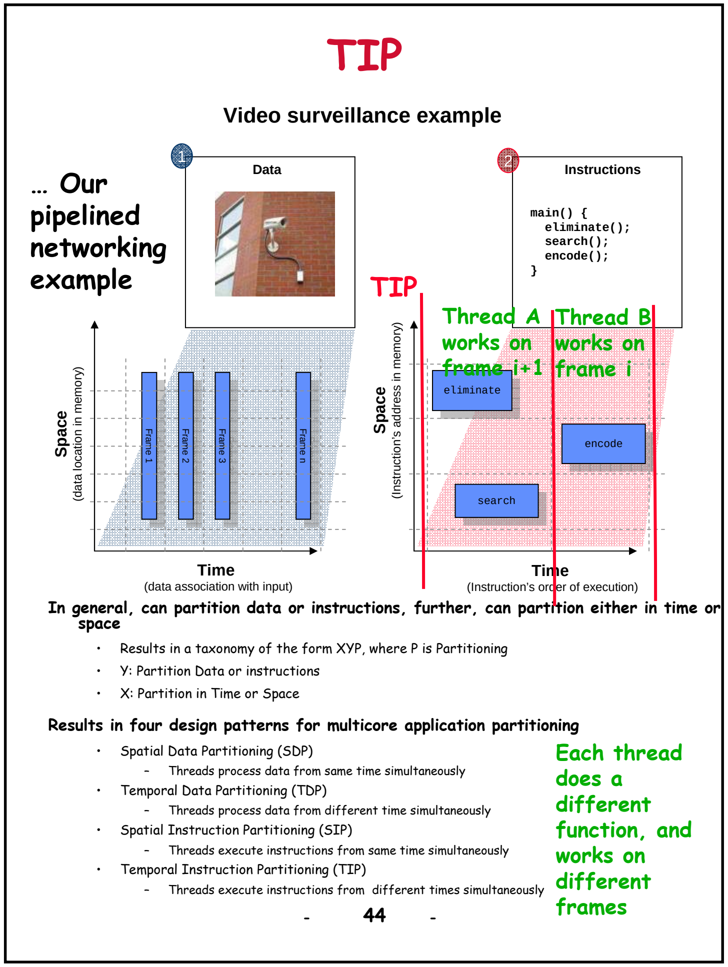
- Results in a taxonomy of the form XYP, where P is Partitioning
- Y: Partition Data or instructions
- X: Partition in Time or Space

Results in four design patterns for multicore application partitioning

- Spatial Data Partitioning (SDP)
  - Threads process data from same time simultaneously
- Temporal Data Partitioning (TDP)
  - Threads process data from different time simultaneously
- Spatial Instruction Partitioning (SIP)
  - Threads execute instructions from same time simultaneously
- Temporal Instruction Partitioning (TIP)
  - Threads execute instructions from different times simultaneously

Each thread does a different function, and works on different parts of same frame

- 43 -


1
Data
2
Instructions
main() {
  eliminate();
  search();
  encode();
}
TIP
Space
(data location in memory)
Frame 1
Frame 2
Frame 3
Frame n
Time
(data association with input)
Space
(Instruction's address in memory)
Thread A works on frame i+1
Thread B works on frame i
eliminate
encode
search
Time
(Instruction's order of execution)

TIP

Video surveillance example

… Our pipelined networking example

In general, can partition data or instructions, further, can partition either in time or space

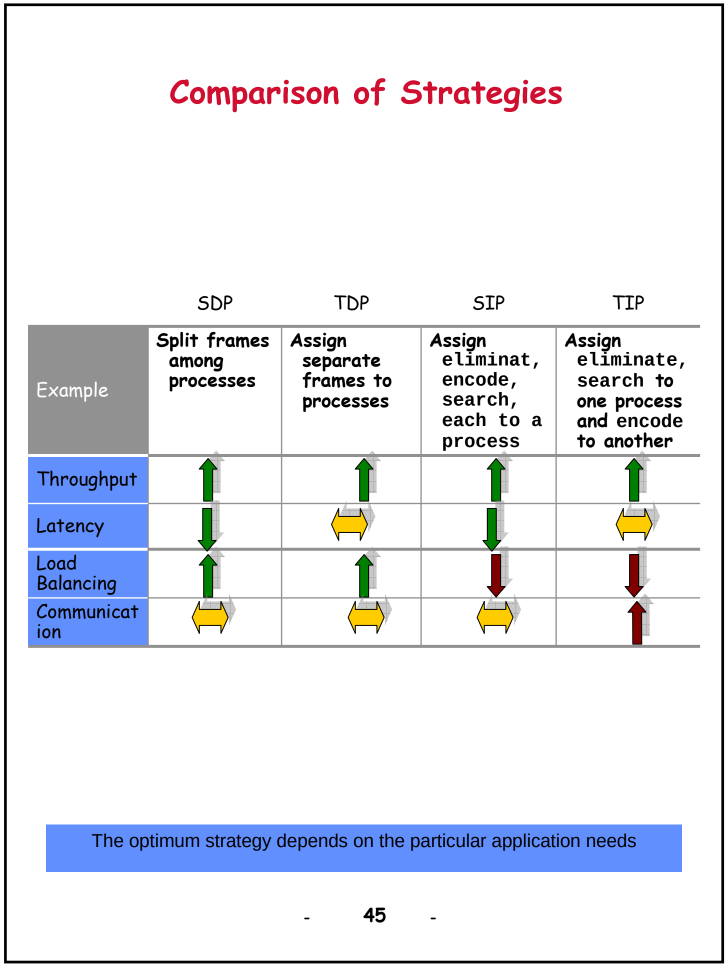
- Results in a taxonomy of the form XYP, where P is Partitioning
- Y: Partition Data or instructions
- X: Partition in Time or Space

Results in four design patterns for multicore application partitioning

- Spatial Data Partitioning (SDP)
  - Threads process data from same time simultaneously
- Temporal Data Partitioning (TDP)
  - Threads process data from different time simultaneously
- Spatial Instruction Partitioning (SIP)
  - Threads execute instructions from same time simultaneously
- Temporal Instruction Partitioning (TIP)
  - Threads execute instructions from different times simultaneously

Each thread does a different function, and works on different frames

- 44 -



Comparison of Strategies

	SDP	TDP	SIP	TIP
Example	Split frames among processes	Assign separate frames to processes	Assign eliminat, encode, search, each to a process	Assign eliminate, search to one process and encode to another
Throughput	↑	↑	↑	↑
Latency	↓	↔	↓	↔
Load Balancing	↑	↑	↓	↓
Communication	↔	↔	↔	↑

The optimum strategy depends on the particular application needs