Parallel Programming Patterns
or
How to Divvy up the PIE

First, More on the Message Passing Model

Message Passing Machine Model

Concepts in
Partitioning
Load balancing
Placement
Locality

Agarwal
Message Passing Machine Model

Core 0
local cache

local cache

local cache

local cache

Core 1

Main Memory

How to Send a Message

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

But first, “gather” from mem

Network

Flit (flow control unit) Typically a word

Message

Length

Dest addr

Message type (tag)

Src addr

Header

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

Interrupts

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

Network

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

Plate

Boundary condition \( B(x,y) \)

\( A(x,y) \)

E.g., \( A \) is temperature

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

Iterative Jacobi Solution Method

Discretize

\[ A_{i,j} = \frac{A_{i,j}^{n-1} + A_{i+1,j}^{n-1} + A_{i,j+1}^{n-1} + A_{i-1,j}^{n-1}}{4} \]

Next value

Previous iteration values

Iterate till no change.
The ultimate parallel method.
Jacobi is the most commonly used parallel app!
Now, Getting to the Point...
Parallel Implementation

Q: How would you partition the problem? I.e., who does what.
  • Say, on 16 processors?

Communication?

Remember

\[ A_{i,j} = \downarrow \rightarrow \leftarrow \uparrow \]

Partitioning and Communication

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

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: Send and receive rows
Focus on Core1...
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.

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

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

Network

Overhead \( o \) + 1 unit of time to send a word

Overhead \( o \) + 1 unit of time to receive a word

E.g., \( o = 20 \)

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

Runtime for Row-Wise Partitioning in logp

\[
T(\text{iter}) = 16 + 16 + 16 + 16 + 16 \times 4 \text{ cycles}
\]

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

\[ T(\text{iter}) = 20 + 20 + 20 + 20 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 16 \times 4 \text{ cycles} \]

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

Neighborhood relation: Send and receive rows

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

In general, \( L \) units of time in transit.

Network

1 hop, 1 unit of time to send a word

2 hops, 1 unit of time to receive a word

Yes, ring model better captures communication locality.

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 piping 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 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).
Data Partitioning Approach in Networking

Packet inspection module

Sequential Parallel

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

Hybrid Data Partitioning Program Partitioning Approach

Sequential Parallel

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

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

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

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

Each thread performs all functions on different parts of same frame

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

TDP

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 performs all functions on different frames

SDP

Video surveillance example

... Our jacobi 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

TDP

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 performs all functions on different frames
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 the same frame

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 the same frame
## Comparison of Strategies

<table>
<thead>
<tr>
<th>Example</th>
<th>SDP</th>
<th>TDP</th>
<th>SIP</th>
<th>TIP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Split frames among processes</td>
<td>Assign separate frames to processes</td>
<td>Assign eliminate, encode, search, each to a process</td>
<td>Assign eliminate, search to one process and encode to another</td>
</tr>
<tr>
<td>Throughput</td>
<td><img src="image" alt="Comparison" /></td>
<td><img src="image" alt="Comparison" /></td>
<td><img src="image" alt="Comparison" /></td>
<td><img src="image" alt="Comparison" /></td>
</tr>
<tr>
<td>Latency</td>
<td><img src="image" alt="Comparison" /></td>
<td><img src="image" alt="Comparison" /></td>
<td><img src="image" alt="Comparison" /></td>
<td><img src="image" alt="Comparison" /></td>
</tr>
<tr>
<td>Load Balancing</td>
<td><img src="image" alt="Comparison" /></td>
<td><img src="image" alt="Comparison" /></td>
<td><img src="image" alt="Comparison" /></td>
<td><img src="image" alt="Comparison" /></td>
</tr>
<tr>
<td>Communication</td>
<td><img src="image" alt="Comparison" /></td>
<td><img src="image" alt="Comparison" /></td>
<td><img src="image" alt="Comparison" /></td>
<td><img src="image" alt="Comparison" /></td>
</tr>
</tbody>
</table>

The optimum strategy depends on the particular application needs.