BITS Pilani
Pilani Campus

DSECL ZG517 - Systems for Data Analytics

Session #3 – Systems Attributes for Data Analytics – Parallel/Distributed System

Murali P
muralip@wilp.bits-pilani.ac.in

[Saturday – 04:30 PM]

Agenda

- Systems Attributes for Data Analytics – Parallel and Distributed Systems
  - Motivation
  - Parallel/Distributed Processing and Data

Review

Locality Example 2: Partitioning in QuickSort

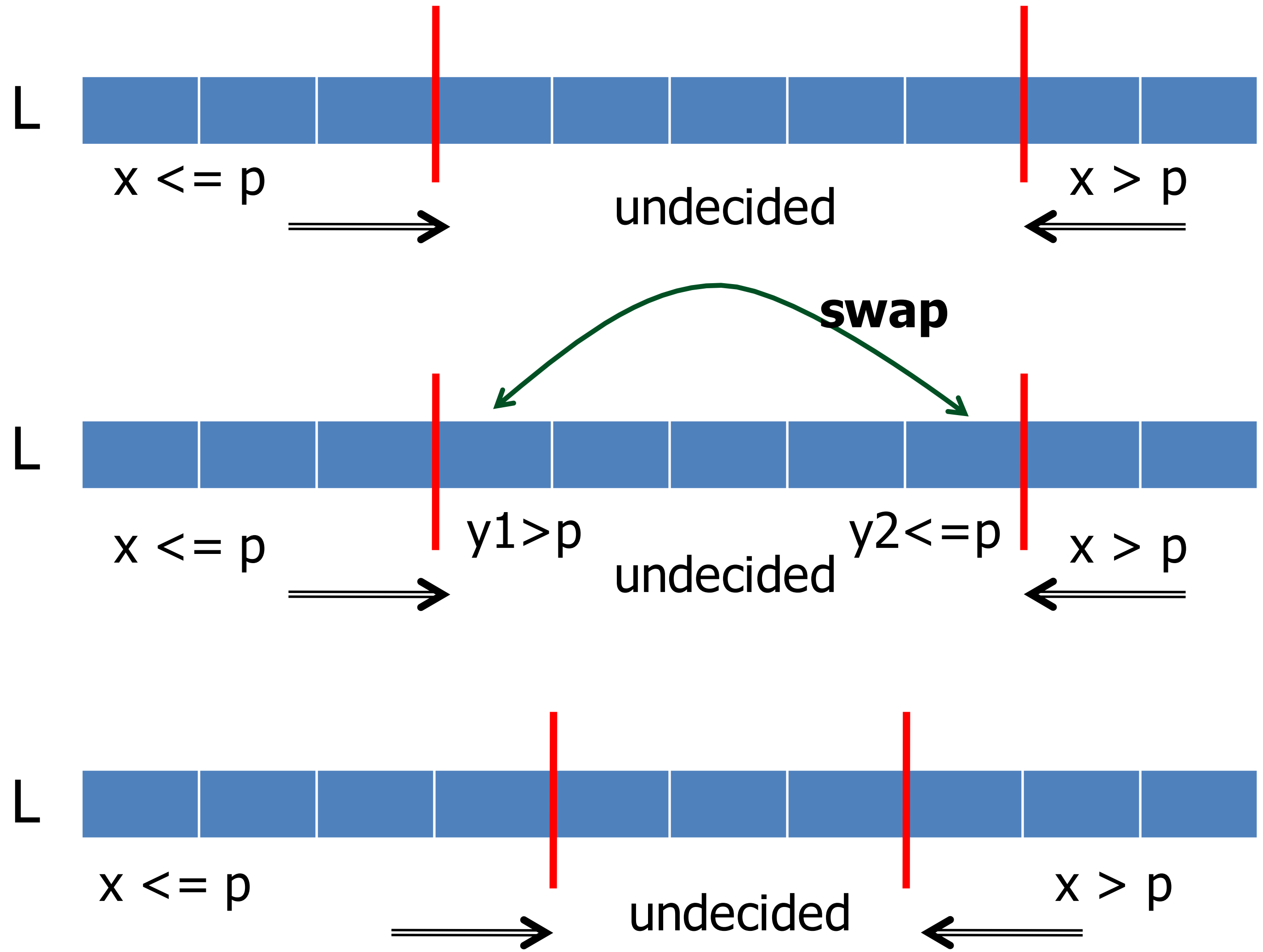
- Partitioning is a step in QuickSort:
  - Given a *pivot **p**,* partition a list ***L*** such that:
    - **p= L[i]** for some ***i*** where **0 <= i < n** and
    - for all ***j*** where **0 < j < n** and **n = length(L)**
      - **j < i** implies **L[j] <= p** and
      - **j > i implies L[j] > p**

Locality Example 2: Partitioning in QuickSort

- Partitioning is a step in QuickSort:
  - Given a *pivot* ***p***, partition a list ***L*** such that:
    - **p= L[i]** for some ***i*** where **0 <= i < n** and
    - for all ***j*** where **0 < j < n** and **n = length(L)**
      - **j < i** implies **L[j] <= p** and
      - **j > i implies L[j] > p**
- Hoare's Partitioning Algorithm:

```
/* assume L[lo] = p */
i=lo+1;  j=hi;
while (i <= j) {
   while (L[i]<=p) i++;   while (L[j]>p) j--;
   if (i<j) swap(L[i], L[j]);
}
```

(Hoare's) Partitioning : (pivot p in L)



Partitioning and Locality

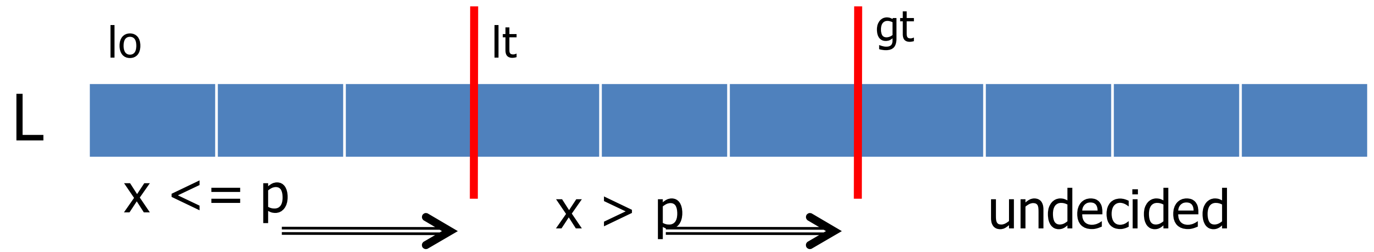
- Recall the inner loops of the Partition algorithm:
  - Array is accessed left-to-right (**L-R**) in one loop, and right-to-left (**R-L**) in the next.
- And these two loops are repeated in the outer loop
  - i.e. the ***locus*** alternates within each iteration of the outer loop, and from the end of one iteration to the start of the next

I1: L-R	X	X								
I1:R-L									Y	Y
I2: L-R			X	X						
I2: R-L								Y		
I3: L-R					X					
I3: R-L							Y			

Back to the Partitioning Algorithm:

- Can you access the array elements from one end instead of both ends?

Locality-Aware Partitioning : (pivot p in L)



Lomuto's partitioning

Maintain these sub-lists (and *the invariants*) i.e.

forall j: lo<=j<=lt --> L[j]<=piv

and

forall j: lt<j<=gt --> L[j]>piv

<u>Exercise</u>:

- Code the locality-aware partitioning algorithm.
- Determine the impact of locality by measuring the time taken – for different data sizes – the classic algorithm and this one.

innovate achieve lead

BITS Pilani
Pilani Campus

Review: Systems Attributes for Data Analytics – Parallel/Distributed

Courtesy: Prof Sundar B slides

Motivation: High Performance

<u>Question</u>

- *Can performance be improved by doing **multiple computations** in **parallel** i.e. at the same time?*

Contexts for High Performance

High Performance requirement may arise due to

- Problem complexity or
- Size of data

or both.

Exercise

In each of the following cases, identify / argue the motivation for high performance requirement:

1. Airline scheduling
2. Summary Statistics of Historic Sales Data of a Retailer
3. Web Crawler

Spectrum of Parallelism

Pipelining

Super-scalar

Message Passing over Custom Interconnect

Clusters

Multi-Threaded

Multi-Core / Many-core

Shared Memory over Custom Interconnect

Grids

Clouds

Pseudo-Parallel

Parallel

Distributed

Instruction – Level

Task – Level
Algorithm - Level

Algorithm - Level
Application - Level

Service -Level

Spectrum of Parallelism

innovate achieve lead

Increasing granularity of parallelism
Decreasing coupling

Pipelining

Super-scalar

Multi-Threaded

Multi-Core / Many-core

Shared Memory over Custom Interconnect

Message Passing over Custom Interconnect

Clusters

Grids

Clouds

Pseudo-Parallel

Instruction – Level

Parallel

Task – Level
Algorithm - Level

Distributed

Algorithm - Level
Application - Level

Service -Level

innovate achieve lead

BITS Pilani

Pilani Campus

Parallel Processing: Memory Access Models

Shared Memory Model:

- Multiple tasks access data via *a shared logical address space* (i.e. a single virtual memory)

Distributed Memory Model:

- Multiple tasks – *executing a single program* – access data from separate (and isolated) address spaces.(i.e. separate virtual memories)

Questions:

1. Does the shared memory model refer to a single program? Why or why not?
2. If a single program is being executed in a distributed memory model, but memories are isolated, how can processors access all the data?

Distributed Memory and Message Passing

In a Distributed Memory model,

- Data has to be moved across Virtual Memories:
  - i.e. a data item in $VMem_1$ produced by task T_1 has to be "communicated" to task T_2 so that
  - T_2 can make a copy of the same in $VMem_2$ and use it.

Whereas in a Shared Memory model,

- task T_1 write the data item into a memory location and T_2 can read the same
  - as that *memory location is part of the logical address space that is shared* between the tasks

Computing Model for Message Passing

Implications:

- Each data item must be located in one of the address spaces
  - i.e. data must be partitioned explicitly and placed (i.e. *distributed*)
- All interactions between processes require explicit communication i.e. ***passing of messages***
  - In the simplest form:
    - a sender (who has the data) and
    - a receiver (who has to access the data)

  must co-operate for exchange of data

Message Passing Model – Separate Address Spaces

Use of separate address spaces complicates programming but this complication is usually restricted to one or two phases:

- Partitioning the input data
  - This improves locality
    - i.e. each process is enabled to access data from within its address space,
      - » which in turn is likely to be mapped to the memory hierarchy of the processor in which the process is running
- Merging / Collecting the output data
  - This is required if each task is producing outputs that have to be combined.

Message Passing Model - Interactions

Use of message passing for interaction complicates programming:

- Process that owns or produces the data must participate in message exchanges
  - even if these have nothing to do with its own flow of computation.
- Communication patterns that are dynamic and/or unstructured result in complex programs:
  - Messaging code – *which may be housekeeping code for data producer* – is scattered and tangled with other code.

Shared Memory Model: Implications for Architecture

- The most straightforward way to realize a shared memory model onto an architecture is a ***shared memory system***:
- i.e.
  - Physical memory (or memories) are *accessible by all processors*
  - A single program is implemented as a *collection of threads* (with one or more threads scheduled in a processor) and
  - The single (logical) address space is mapped onto the physical memory (or memories).

Shared Memory Model: Multi-Threaded Programming

- The most straightforward way to run a task on a shared memory model is a ***multi-threaded program*** i.e.
  - A program is a *collection of threads* (with threads scheduled by the OS or the runtime environment of a language).
- Logical Model
  - Shared address space
- Protection Model
  - Fully shared space
- Running Model (for application Threads)
  - Stack is local
    - (Why?)
  - Rest are shared
    - - typically these include *code area*, *global area*, and *heap*.

Computing Model for Message Passing

- Implications:
  - Each data item must be located in one of the address spaces
    - i.e. <u>data must be partitioned</u> explicitly and placed (i.e. <u>*distributed*</u>)
  - All interactions between processes require explicit communication i.e. ***passing of messages***
    - In the simplest form:
      - a sender (who has the data) and
      - a receiver (who has to access the data)

    must <u>co-operate for exchange of data</u>

Message Passing Model – Separate Address Spaces

- Use of separate address spaces complicates programming
- but this complication is usually restricted to one or two phases:
  - Partitioning the input data
    - This improves locality
      - i.e. each process is enabled to access data from within its address space,
        - » which in turn is likely to be mapped to the memory hierarchy of the processor in which the process is running
  - Merging / Collecting the output data
    - This is required if each task is producing outputs that have to be combined.

Message Passing Model - Interactions

- Use of message passing for interaction complicates programming:
  - Process that owns or produces the data must participate in message exchanges
    - even if these have nothing to do with its own flow of computation.
  - Communication patterns that are dynamic and/or unstructured result in complex programs:
    - Messaging code – *which may be housekeeping code for data producer* – is scattered and tangled with other code.

Structure of Message Passing Programs

- *Synchronous execution of tasks is difficult to achieve* given an environment of separate processes (and address spaces)
  - Tasks may not be the same.
  - Although processes may run on a homogeneous environment (e.g. *a cluster where all nodes are of the same configuration*),
    - execution speeds may vary.
- On the other hand *asynchronous execution is hard to reason* about:
  - Proving properties about the progress of programs is dependent on timing – or at least, precedence, – information.

Thank you !