Lecture 07: Parallelism and Data Structure

Behzad Bordbar
Recap

Algorithms and their importance
They are complex -> Stepwise refinement
Building blocks (5 blocks):
Three main building blocks (sequence, selection, iteration).
Everything can be represented by the three.
Recursion
This lecture: Parallelism and parallel blocks
Lets start with Java
Contents

Parallelism : an introduction
Why it is important?
Parallel blocks and Parallelism in Algorithms
Three sorting algorithms
Data Structures (briefly, for complexity lecture)

Lets start with Java
Who has written a Java program with more than one thread?
Threads in Java

How many thread in running Java program? We all have seen

Exception in thread “main” ....

Helloworld has two thread:

- main()
- Garbage collection Thread.

GUI programs have even more threads!
Why parallelism?

Parallelism allows us to do more than one thing at the same time.

Why?

- Better interaction with user (GUI)
- Easier programming (client/server)
- ...

Parallelism can not be avoided!
Parallelism cannot be avoided!

Moore’s law

*it is not a law!* Observation made by Gordon Moore

number of transistors per square inch on integrated circuits *has doubled* every year since the integrated circuit was invented in 1958 (up to 1965)

Moore prediction has proved to be valid up to now. Moore 2005: cannot be sustained for more than a decade… would eventually reach the limits of miniaturization at atomic level...

How to do many operations? Multicore & parallelism
Parallelism

All algorithms so far are based on assumption that one processor is executing.

- In a computer with several processors, if a task can be broken into a number of parallel tasks, the task can be executed simultaneously by different processor.

The execution time for the whole process can be shortened.

- parallel (concurrent) algorithms vs. sequential algorithms
A sequential algorithm for sorting: Bubble sort.

The basic idea of bubble sort:

- Pass through the list of names by comparing each name with its successor and interchanging them if they are out of order.
- If at the end of the pass all the names are in the correct order, nothing further needs to be done (finished).
- If not, another pass is made through the list until all names are in the order.
Refined bubble sort:

repeat
  start at the top of the list
repeat N-1 times
  if names alphabetically follows its successor
  then exchange name and successor
      remember that an exchange has been performed on this pass
  consider the next name in the list
endrepeat
until no exchange has been performed on this pass
endrepeat
A parallel algorithm for sorting:

module parallelcompare(i, j)

{Each processor executes this with different pair (i, j)}

if Name[i] alphabetically precedes name[j]

then put 0 into the ith row and jth column of the grid

else put 1 into the ith row and jth column of the grid

endmodule
Suppose that you have 8 processors

<table>
<thead>
<tr>
<th></th>
<th>John</th>
<th>Kate</th>
<th>Sam</th>
<th>Fred</th>
<th>Bill</th>
<th>Jill</th>
<th>Marry</th>
<th>Desired Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Kate</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Sam</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Fred</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Bill</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Jill</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Marry</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>
An recursive algorithm for sorting to be used in complexity section:

module sort(list)
{Sorts given list of N names into alphabetically order}
if N > 1
    then divide the list into two halves
    sort(first half of the list)
    sort(second half of the list)
    merge the two halves together
endif
endmodule

How to merge two halves? (clues given in page 49)
Example
Are we really doing multiple things in parallel at the same time?

No! In a single processor machine, a process shares one thread for a short time and switches to running other threads.

Multi-threading is similar to *timesharing*.

Multi-processor machines running multiple things at the same time.

**What is the difference between Program, Process and Thread?** We will answer this question accurately after studying basics of OS.
Algorithms describe processes which transform one data structure (the input) into another data structure (the output). Intermediate data structures may be used.
Sequences (a common type of data structure):

- A sequence is a set of items that is ordered such that every item except last one has exactly one successor, and every item except the first one has exactly one predecessor.

Examples of sequences:
- An English word: a sequence of letters;
- English text: a sequence of words separated by spaces and other punctuation symbols;
- A number (in written form): a sequence of digits;
- A telephone directory: a sequence of records, each of which contains a name, an address, and a telephone number;
- . . .
Different types of sequence

Three kinds of Sequences:

- **Array**: a sequence of fixed length, in which each item is identified by its position. Example: the row of numbers, num[1], …, num[i], …, num[n].

- **Queue**: a sequence of variable length, in which items are always added at one end and removed at the other. So, the order in which items are removed is the same in which they are added (FIFO basis).

- **Stack**: a sequence of variable length, in which items are added and removed at only one end. Thus the order in which items are removed is the reverse in which they are added (LIFO basis).
Trees (another important type of data structure):

- The data in a tree is represented by its nodes.
- Branches represent logical relationships between a data item on one level of the hierarchy and several items at the next level down.
- The node at the highest level of the hierarchy is called the root of the tree.
- The nodes at the lowest extremities are called the leaves.
- A tree is an appropriate data structure for expressing any hierarchical relationships.

Two examples:

- the structure of an English sentence
- the successive refinement of module
Trees are **recursive data structures**, as every node in the tree is the root of a **subtree**.

A tree is either:

1. empty, or
2. a node plus a (possibly empty) set of branches, each leading to a tree

This recursive structure of trees is particularly conducive to recursive algorithms.
Binary trees

- Each node has at most two branches coming from it.
- A binary tree is particularly useful for sorting, since the two branches from each node can be used to represent “before” and “after” relationships.
- A binary tree is sorted if the data at each node follows all the data in the node’s left subtree and precedes all the data in the node’s right subtree.