Parallel Programming
0024

Mergesort

Spring Semester 2010

Outline

- Discussion of last assignment
- Presentation of new assignment
  - Introduction to Merge-Sort
  - Code Skeletons (see homepage)
  - Issues on Parallelizing Merge-Sort
  - Performance measurements
- Questions/Comments?

Discussion of Homework 3

Part2 – First question

- Why is it not sufficient to add the 'synchronized' keyword to the read() and write() methods to guarantee the specified behavior of the producer/consumer problem?

Part2 – First question

- Why is it not sufficient to add the 'synchronized' keyword to the read() and write() methods to guarantee the specified behavior of the producer/consumer problem?
- Solution: Synchronization ensures that the producer and the consumer can not access the buffer at the same time. But it does not prevent the consumer to read a value more than one time or the producer to overwrite a value that was not read.

Part2 – Second Question

- Would it be safe to use a boolean variable as a "guard" within the read() and write() methods instead of using the synchronized keyword?

Part2 – Second Question

- Would it be safe to use a boolean variable as a "guard" within the read() and write() methods instead of using the synchronized keyword?
- Solution: No, reading *and* writing a value is not atomic! – Can you tell me why, e.g., i++ is not atomic?

Part3 – First Question

- Would it suffice to use a simple synchronized(this) within the run()-method of each, the producer and the consumer to guard the updating of the buffer?

Part3 – First Question

- Would it suffice to use a simple synchronized(this) within the run()-method of each the producer and the consumer to guard the updating of the buffer?
- No, since Producer and Consumer are different objects with different locks ➔ no mutual exclusion guaranteed

Part3 – Second Question

- What is the object that should be used as the shared monitor and (the object upon which the threads are synchronized())?
- Solution: The shared instance of UnsafeBuffer.

- Question: What could you have used instead?

Part 3 – Third Question

- What are the potential advantages/disadvantages of synchronizing the producer/consumer over synchronizing the buffer?

Part 3 – Third Question

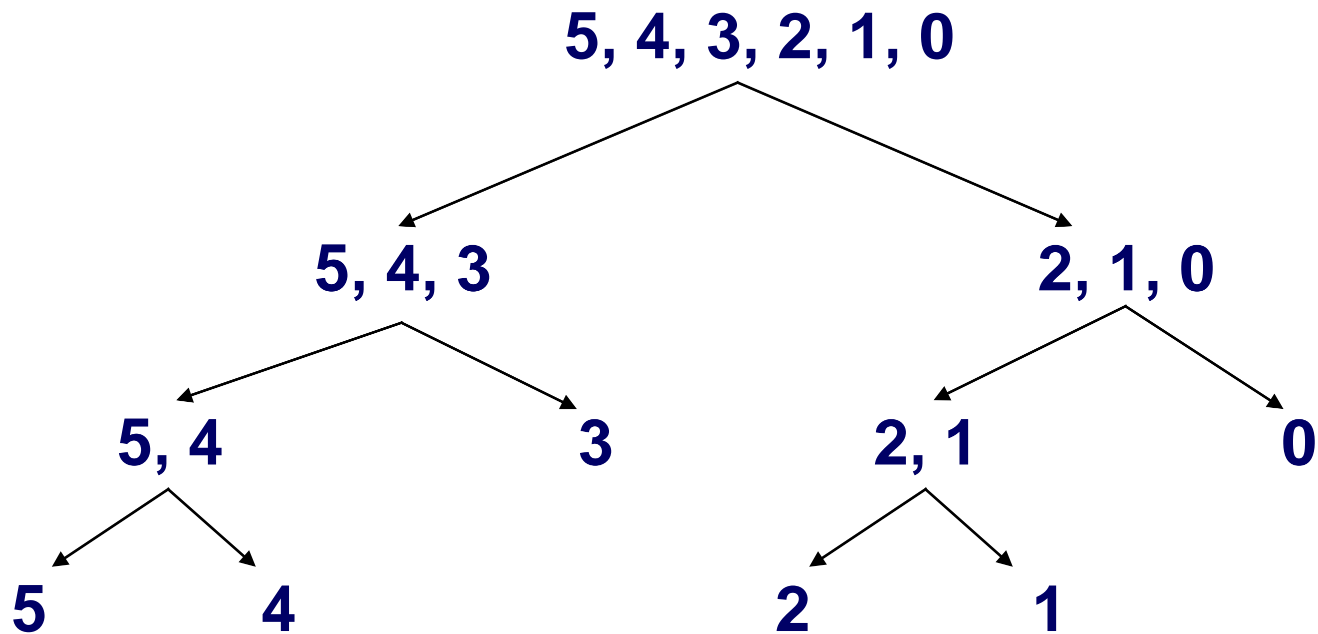
- What are the potential advantages/disadvantages of synchronizing the producer/consumer over synchronizing the buffer?
- Advantages:
  - You can use arbitrary (also unsafe!) buffers
  - You can do things in the Producer/Consumer that need to be done before the other thread can use the buffer. (For example print something to the console).
- Disadvantages:
  - More work to do :-)
  - More error-prone

Presentation of Homework 4

MergeSort

- Problem: Sort a given list 'l' of 'n' numbers
- Example:
  - Input: 9 8 7 6 5 4 3 2 1 0
  - Output: 0 1 2 3 4 5 6 7 8 9
- Algorithm:
  - Divide l into two sublists of size n/2
  - Sort each sublist *recursively* by re-applying MergeSort
  - Merge the two sublists back into one sorted list
- End of recursion:
  - Size of the sublist becomes 1
  - If size of a sublist > 1 => other sorting needed

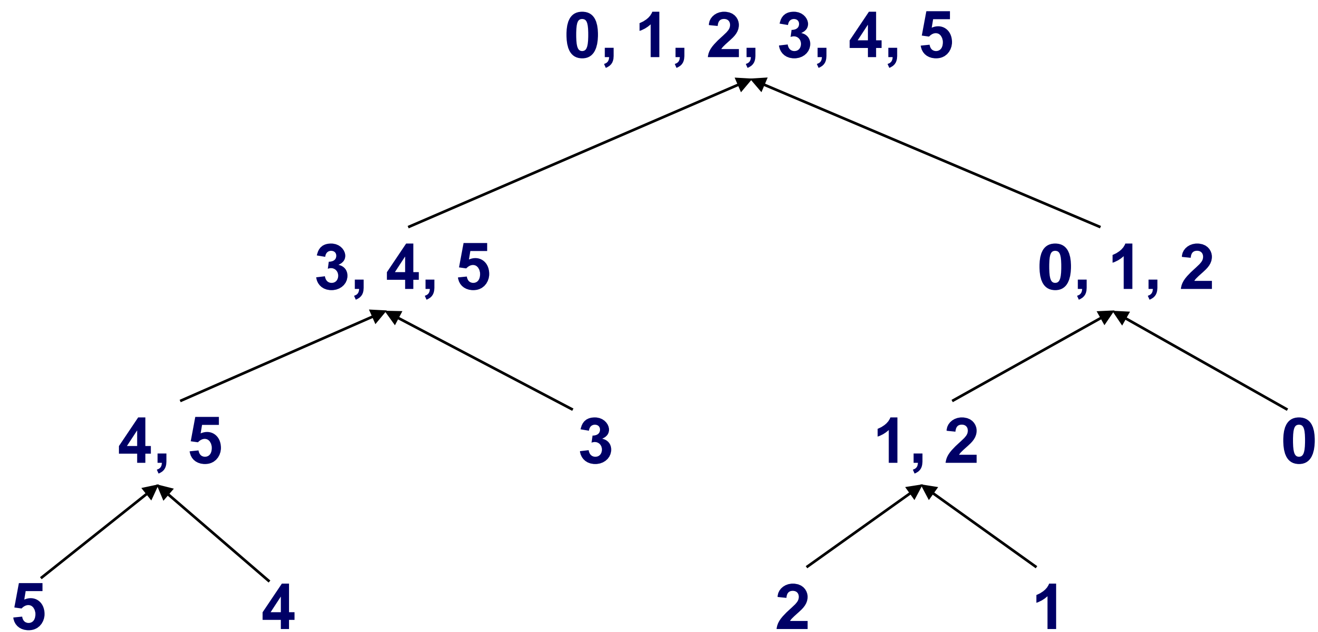
Example: Divide into sublists



Merging

- Combine two sorted lists into one sorted list
- Example:
  - List 1: 0, 5
  - List 2: 3, 4, 45
  - Output: 0, 3, 4, 5, 45
- Merging example:
  - Create a list Output of size 5
  - 0, 5 and 3, 4, 45 — $0 < 3 \rightarrow$ insert 0 in Output
  - 0, 5 and 3, 4, 45 — $3 < 5 \rightarrow$ insert 3 in Output
  - 0, 5 and 3, 4, 45 — $4 < 5 \rightarrow$ insert 4 in Output
  - 0, 5 and 3, 4, 45 — $5 < 45 \rightarrow$ insert 5 in Output
  - Finally, insert 45 in Output

Example: Merging Sorted Sublists



The Code Skeletons (Eclipse)

A Parallel MergeSort

- Which operations can be done in parallel?

A Parallel MergeSort

- Which operations can be done in parallel?
  - Sorting
    - Each sub-list can be sorted by a separate thread

A Parallel MergeSort

- Which operations can be done in parallel?
  - Sorting
    - Each sub-list can be sorted by a separate thread
  - Merging
    - Two ordered sub-lists can be merged by a thread

A Parallel MergeSort

- Which operations can be done in parallel?
  - Sorting
    - Each sub-list can be sorted by a separate thread
  - Merging
    - Two ordered sub-lists can be merged by a thread
- Synchronization issues
  - Limitations in parallelization?

A Parallel MergeSort

- Which operations can be done in parallel?
  - Sorting
    - Each sub-list can be sorted by a separate thread
  - Merging
    - Two ordered sub-lists can be merged by a thread
- Synchronization issues
  - Limitations in parallelization?
    - Merge can only happen if two sublists are sorted
- Performance issues
  - Number of threads?
  - Size of array to sort?

Load balancing

- What if: size of array % numThreads != 0?
- Simple (proposed) solution
  - Assign remaining elements to one thread
- Balanced (more complicated) solution
  - Distribute remaining elements to more threads

Performance Measurement

# of threads /array size	1	2	4	8	16	32	64	…	1024?
100,000	x								
500,000	x								
…									
10,000,000?									

How to Measure Time?

- System.currentTimeMillis() might not be exact
  - Granularity might be higher than a millisecond
  - Might be slightly inaccurate
- System.nanoTime()
  - Nanosecond precision, but not nanosecond accuracy
- For our measurements System.currentMillis() is good enough

How to Measure Time?

```
long start, end;

start = System.currentMillis();

// some action

end = System.currentMillis();

System.out.println("Time elapsed: "
                    + (end - start));
```

Questions to be answered

- Is the parallel version faster?
- How many threads give the best performance?
- What is the influence of the CPU model/CPU frequency?

The Harsh Realities of Parallelization

- **Ideally**
  - **upgrading from uniprocessor to *n*-way multiprocessor should provide an *n*-fold increase in computational power**
- **Real world**
  - **most computations cannot be efficiently parallelized**
    - **Sequential code, synchronization, communication**
- **Speedup**
  - **time(single processor) / time(n concurrent processors)**

Mein Tipp

```
Thread t = …
t.start();
…
try {
    t.join(); // Warten bis t fertig ist
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

Mein Tipp

int[] array

System.out.println(Arrays.toString(array));

Oder für die ersten paar Einträge

System.out.println(Arrays.toString(array).substring(0,20));

StringBuffer und StringBuilder sind schneller als String.

Any Questions?