EE 382C/361C: Multicore Computing **Fall 2016**

Lecture 17: October 25

Lecturer: Vijay Garg *Scribe: Xin Xu*

17.1 Introduction

The topics covered in this lecture are:

1. Merge Sort
2. Wait-Free Synchronization

17.2 Merge Sort

Decompose one task into multiple tasks:

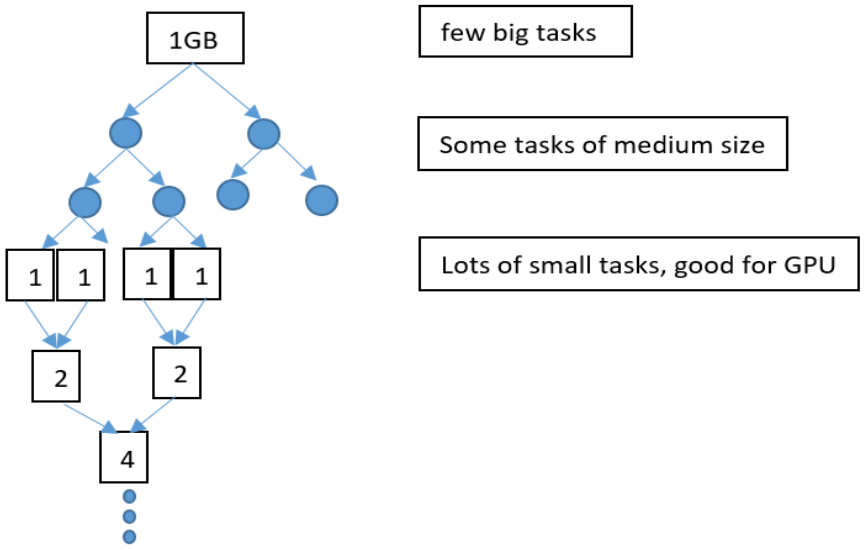


Figure 17.1: Merge Sort

17.2.1 Algorithm for Multiple Merging tasks

1. Determine rank of each element(mentioned in Lecture9). Rank(x) is equal to the sum of number of elements in B less than x and the number of elements in C less than x. Then merge array B and array C to get array D.
2. Cascaded Algorithm(mentioned in Lecture10). Divide n array into n/logn groups. Fill in only splitters.

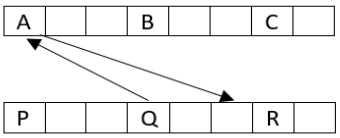


Figure 17.2: Merge Sort

17.3 Wait-Free Synchronization

It is possible to build a multiple reader multiple writer(MRMW) atomic multivalued register from single-reader single-writer(SRSW) safe Boolean registers. This can be achieved by the following chain of constructions:

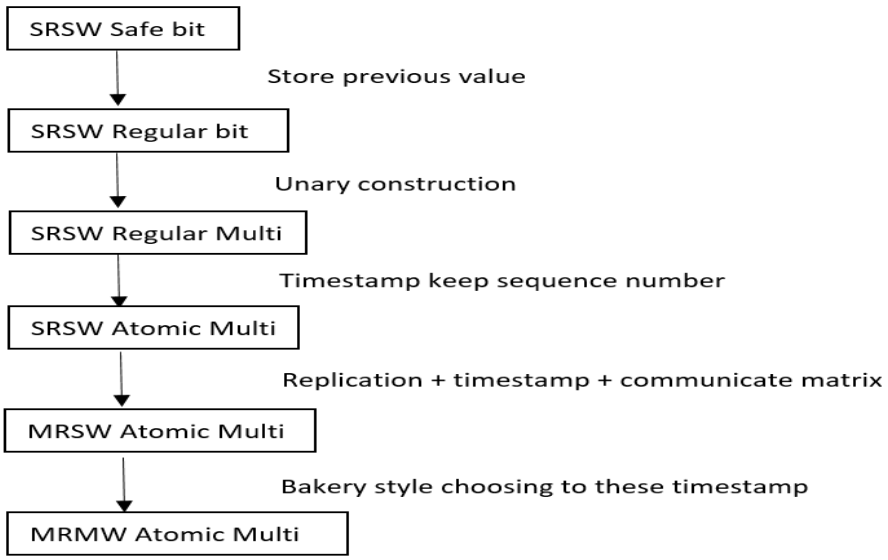


Figure 17.3: construction

17.3.1 Regular SRSW Register

We construct a regular SRSW register from a safe SRSW register using smarter writer which can remember last value.

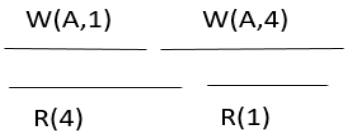
17.3.2 SRSW Multivalued Register

For read: scan the array, looking for the first non-zero bit. The idea is that the reader should return the index of the first true bit. The straightforward solution for writer: Updating the array in the forward direction until it reaches the required index. But it does not work.
Reason: Suppose firstly we set A[3] to 1, and now we need to write A[5] to 1. After the writer executes, the array becomes 0 0 0 0. Before writer set A[5] to 1, the reader comes in, it will find all zeros.
For write, the right operation:

Algorithm 1 RegularBoolean

```
1:  class RegularBoolean {
2:      boolean prev; // not shared
3:      SafeBoolean value;
4:      public boolean getValue() {
5:          return value.getValue();
6:      }
7:      public void setValue(boolean b) {
8:          if (prev != b) {
9:              value.setValue(b);
10:             prev = b;
11:         }
12:     }
13: }
```

•Set A[x] = 1
•Traverse backward resetting bit to zeros
But it is not atomic. This situation can happen:



Code in the handout: the reader first does a forward scan and then does a backward scan at line 14 to find the first bit that is true. Two scans are sufficient to guarantee linearizability.

17.3.3 MRSW Register

We now build a MRSW register from SRSW registers. The straightforward solution is to have an array of n SRSW registers and one writer write to all n registers. This is not linearizable. Instead, we use a sequence number associated with each value. The writer maintains the sequence number and writes this sequence number with any value that it writes. Thus we view our SRSW register as consisting of two fields: the value and ts (for timestamp). In order to build MRSW Register, we use communication matrix. Comm[i][j] is used by the reader i to inform the value it read to the reader j. Reader communicate the value it read to all other readers by writing in the corresponding row.

17.3.4 MRMW Register

The construction of an MRMW register from MRSW registers is to use n MRSW registers for n writers. We use the approach of the Bakery algorithm to assign unique sequence number to each writer.

17.3.5 Atomic Snapshots

Lock-free constructions can be turned into wait-free constructions by using the notion of helping moves. The main idea is that a thread tries to help pending operations. For example, the thread wanting to perform an update operation helps another concurrent thread.

References

[1] V.K. GARG Introduction to Multicore Computing

[2] https://github.com/vijaygarg1/UT-Garg-EE382C-EE361C-Multicore