A4 Parallel Prefix Design – Zuoting Xie

Threads Creation: passing 3 parameters to the constructor – stage of operation, index of the array list & cyclic barrier that records the number of threads generated at this stage.

```
void main(String[] args){
	for (int k = 1; k < toSumList.length; k = k * 2) {
		CyclicBarrier barrier = new CyclicBarrier(numOfThreadsCreatedForThisK);
		for (int i = 0; i < toSumList.length; i = i + 2 * k) {
			if (i + k < toSumList.length) {
				ParaPrefix thread = new ParaPrefix(i,k,barrier);
			}
		}
	}
}
```

Threads Running: the basic idea is that threads will be categorized into different stages and hence run stage by stage. he stage will eventually goes one step further but there is no threads running anymore so it doesn't matter.

1. Synchronized static integer filed stage that records the stage of operation. Check if the stage the thread is at is the same as the current stage – if yes, continue execute; if not, wait. 2. Synchronized static double list, do the actual calculation. Wait until all the threads at the same stage finish calculation. 3. Change the stage to next level and notify all the threads waiting on stage. 4. The process repeats 1-2-3 with stage changed (in new stage, Boolean stageChanged is to set back to false)

ParaPrefix Class {

```
	static int stage = 1;
	static boolean stageChanged = false;
	static double[] valueList;

	int i, k;
	CyclicBarrier barrier;

	public paraPrefix(int i, int k, CyclicBarrier barrier) {
		this.i = i;
		this.k = k;
		this.barrier = barrier;
	}

	run(){
```

```
                synchronized(stage){
                        while(k != stage){
                                stage.wait();
                        }
                }

                synchronized(valueList){

                        stageChanged = false;

                        if(i+k < valueList().length){
                                valueList[i] = valueList[i]+valueList[i+k];
                        }
                }

                barrier.await;

                synchronized(stage){
                        if(stageChanged == false){
                                stage = stage*2;
                                stageChanged = true;
                        }
                        stage.notifyAll();
                }
        }
}
```

Potential problem of the pseudo code I have: For example, stage 1 threads finish running and notifyAll; there is a chance that threads waken up are all of stage 3 or later, then all waken up threads go back to wait state again and stage 2 threads never gets to run (deadlock).

Potential solution: notifyAll before wait, so that eventually stage 2 threads will get notified. But this is much more like all threads constantly notifying and waiting – like looping constantly checking the condition