Parallel Programming
0024

Week 05

Thomas Gross

Spring Semester 2009

Mar 19, 2009

Outline

- Discussion of Solution(s) to Mergesort
  - Performance measurement
- Presentation of Assignment 6

Measurments

```
public static final int K = 10;
public static final int EPS = 5; // %
public static final int HUNDRED_DIV_EPS = 100/EPS;
```

```
long [] best_times = new long[4]; long start, stop;
System.out.print("# elements  \t 1 thread ");
for(int i=2; i<=64; i<<=1) System.out.print("\t"+i+" threads ");
System.out.println();

for (int noi=0; noi<2; noi++) {
  if(noi==0){ no_elements = 100000; } else{ no_elements = 10000000; }
  System.out.print(no_elements+"\t");

  for(int no_threads = 1; no_threads<=64; no_threads <<= 1){
    for(int i=0; i<4; i++) best_times[i] = Long.MAX_VALUE;
    int i=0;
    do{
      int_to_sort = createRandIntArray(no_elements);
      MTMergeSort m = new MTMergeSort(int_to_sort, no_threads);
      start = System.currentTimeMillis(); // start timing
      m.sort();
      stop = System.currentTimeMillis(); // stop timing
      best_times[3] = stop - start;
      Arrays.sort(best_times);
    }while( best_times[2] - best_times[0] >
            best_times[2] / HUNDRED_DIV_EPS || ++i < K );
    long average = (best_times[0] + best_times[1] + best_times[2]) / 3;
   System.out.print(" \t\t"+average);
  }
  System.out.println();
}
```

Notes

Extra keys for Java VM: -Xms1024M -Xmx1024M

Constants in Java:

public static final int K = 10;

Results

Intel Pentium M @ 1 GHz / 512 MB / XP

Threads	1	2	4	8	16	32	64
100 000	70	70	60	70	70	80	80
10 000 000	9947	10728	10241	10128	10124	-	-

Intel Core2 Duo CPU E8500 @ 3.16GHz / 4 GB / Ubuntu 8.10 x64

Threads	1	2	4	8	16	32	64
100 000	13	7	7	7	8	10	12
10 000 000	1951	1034	1029	1040	1050	1036	1054

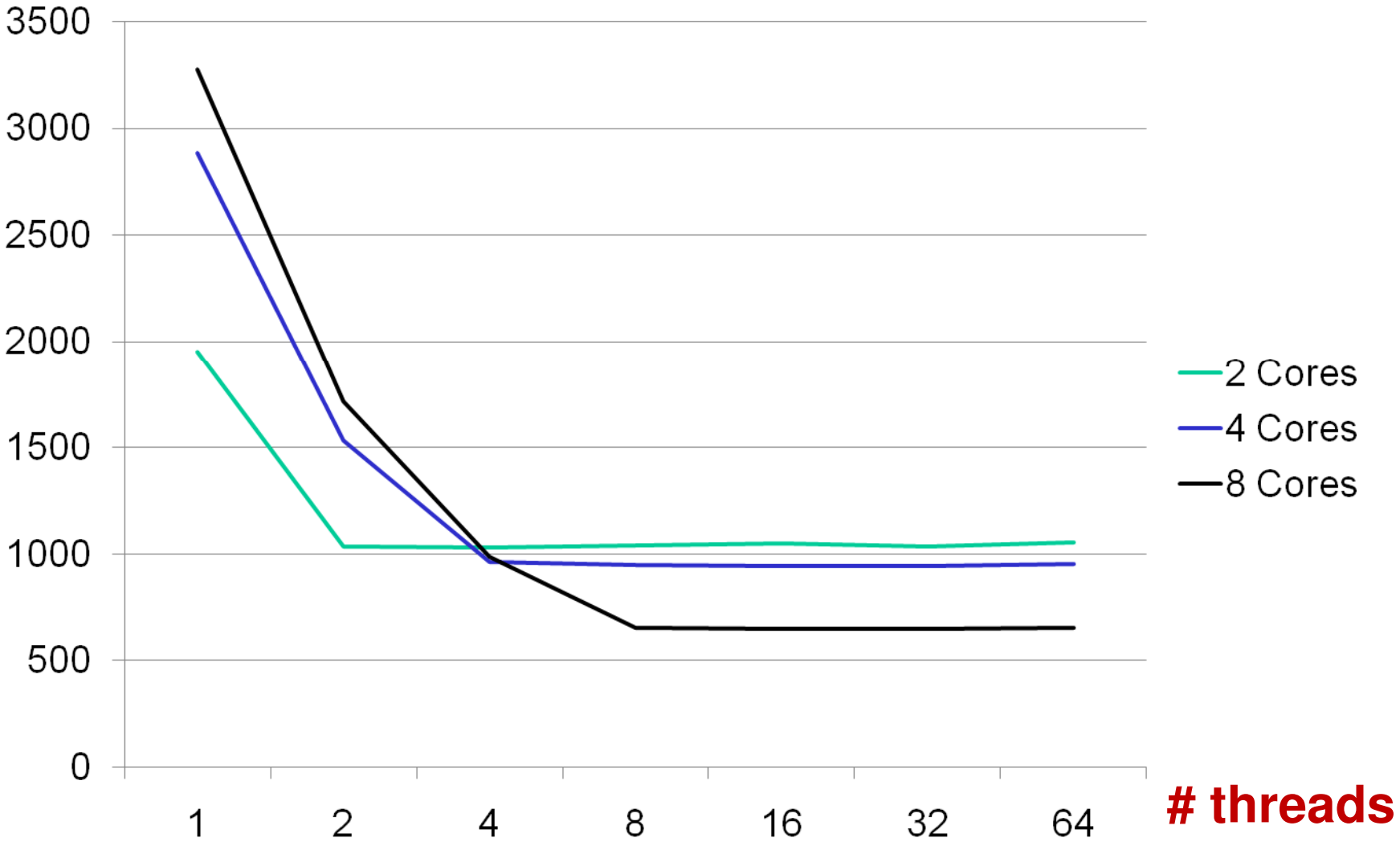
Intel Core2 Quad CPU Q9400 @ 2.66GHz / 4 Gb / 64 bit Vista

Threads	1	2	4	8	16	32	64
100 000	21	11	7	7	8	9	12
10 000 000	2883	1530	958	946	941	943	950

Intel Xeon 8 Core E5345 @2.33 / 2.47 Gb visible due to 32 bit XP

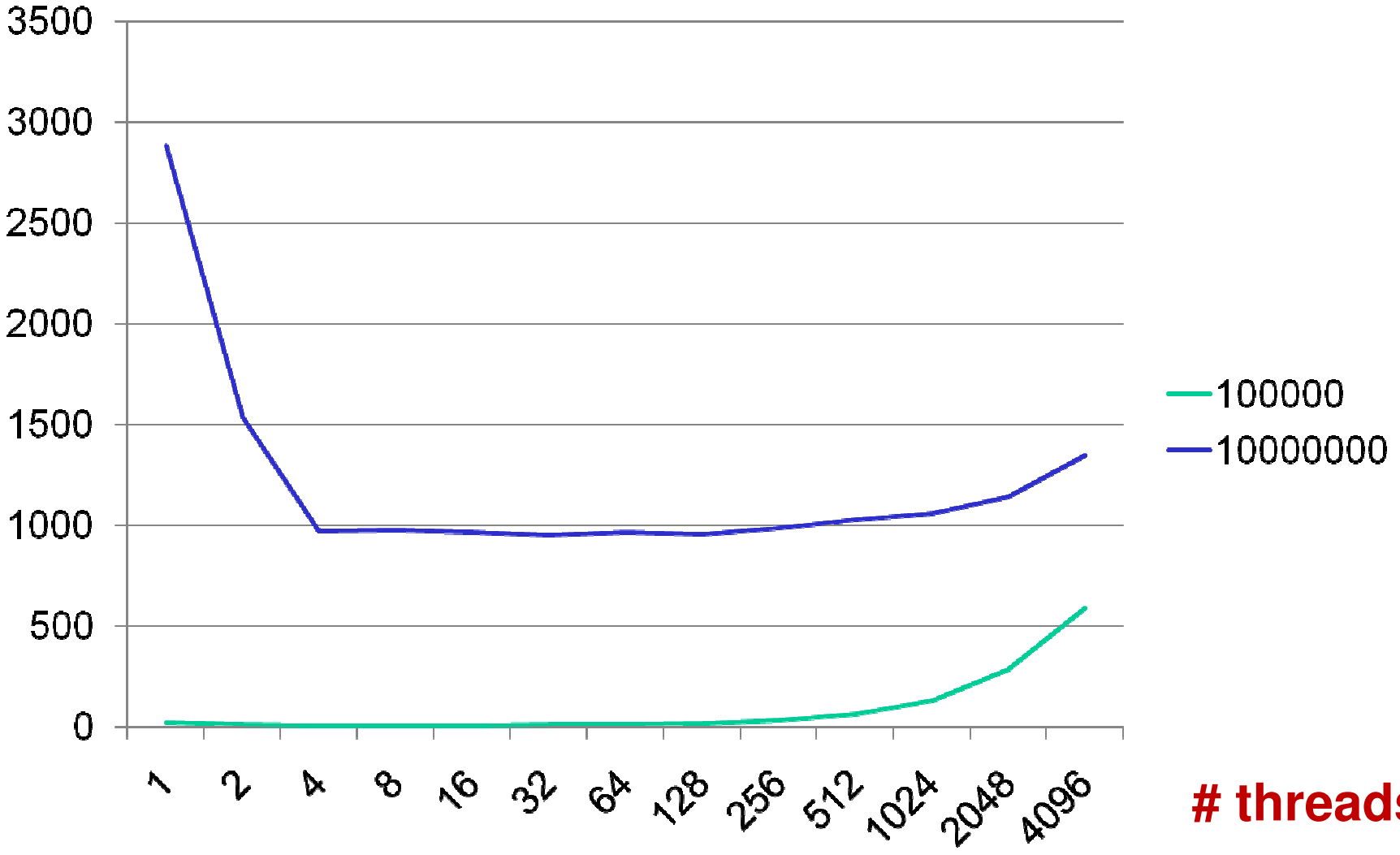
Threads	1	2	4	8	16	32	64
100 000	15	0	0	0	0	0	0
10 000 000	3276	1718	989	656	650	650	656

Compare


3500
3000
2500
2000
1500
1000
500
0
1
2
4
8
16
32
64
2 Cores
4 Cores
8 Cores
threads

More threads

Intel Core2 Quad CPU Q9400 @ 2.66GHz / 4 Gb / 64 bit Vista



Classroom Exercise

Class room exercise

Consider this program (fragment) [PingPong] for thread A (myid == 0) and thread B (myid == 1)

```
// thread A

public void run() {

      while (true) {

   A1: non_critical section

   A2:   while ( !(signal.turn == 0)){}

    A3: critical_section

    A4: signal.turn = 1;

        }

    }
```

Class room exercise, continued

```
// thread B
public void run() {
      while (true) {
   B1: non_critical section
   B2:   while ( !(signal.turn == 1)){}
     B3: critical_section
     B4: signal.turn = 0;
         }
    }
```

Your task (now!)

Show that these threads will never be both in their critical section at the same time.

You should prove this property in a manner that's similar to the proof given in class.

Some thoughts on how to proceed

We introduced already labels for statements and produced two distinct versions for thread A and thread B.

Now you should formulate the invariant.

Invariant(s)

(i) at(A3) -> turn == 0

(ii) at(B3) -> turn== 1

(iii) not [at(A3) AND at(B3)]

Proof strategy

Proof by induction on the execution sequence.

Base case: does (i) hold at the start of the execution of the program (threads at A1 and B1)

Induction step: Assume that (i) holds. Will execution of an additional step invalidate (i)?

Proof (i)

at(A1): condition (i) is false => do not care about signal

at(A2): condition (i) is false => do not care about signal

at(A3): condition (i) is true => turn == 0, follows from the fact that turn was 0 at(A2) AND the transition from A2->A3 did not change value of turn

at(A4): condition (i) is false ==> do not care about turn

Now, we consider:

at(B1) : no change to turn

at(B2) : no change to turn

at(B3) : no change to turn

at(B4) : changes turn to 0

=> Invariant 1 is true

Proof (ii)

Same way (please do it if you had trouble with proof of i)

Proof (iii)

Induction start trivial.

Proof of induction step by contradiction.

Assume thread A entered CS (A3) at time t1

Assume thread B entered CS (B3) at time t2, where t2 = t1 + delta

--> CONTRADICTION: since we are in A3 signal MUST be 0 (cannot be 0 and 1 at the same time)

Assume thread B entered CS (B3) at time t1

Assume thread A entered CS (A3) at time t2, where t2 = t1 + delta

--> CONTRADICTION: since we are in B3 signal MUST be 1 (cannot be 0 and 1 at the same time)

Next exercise…

Show that these threads maintain the mutual exclusion property, i.e. these threads will never be both in their critical section at the same time.

```
public void run() {
        while (true) {
         mysignal.request();
            while (true) {
                if (othersignal.read() == 1) break;
                mysignal.free();
                mysignal.request();
            }
         // critical section
         mysignal.free();
        }  }
```

Any Questions?