Parallel Programming

Recitation Session 4

Thomas Weibel <weibelt@ethz.ch>

Laboratory for Software Technology,
Swiss Federal Institute of Technology Zürich

March 25, 2010

Executive Summary

- Write parallel MergeSort together
- Parallel MergeSort performance
- Classroom exercise: Parallel Matrix multiplication

Outline

Parallel MergeSort

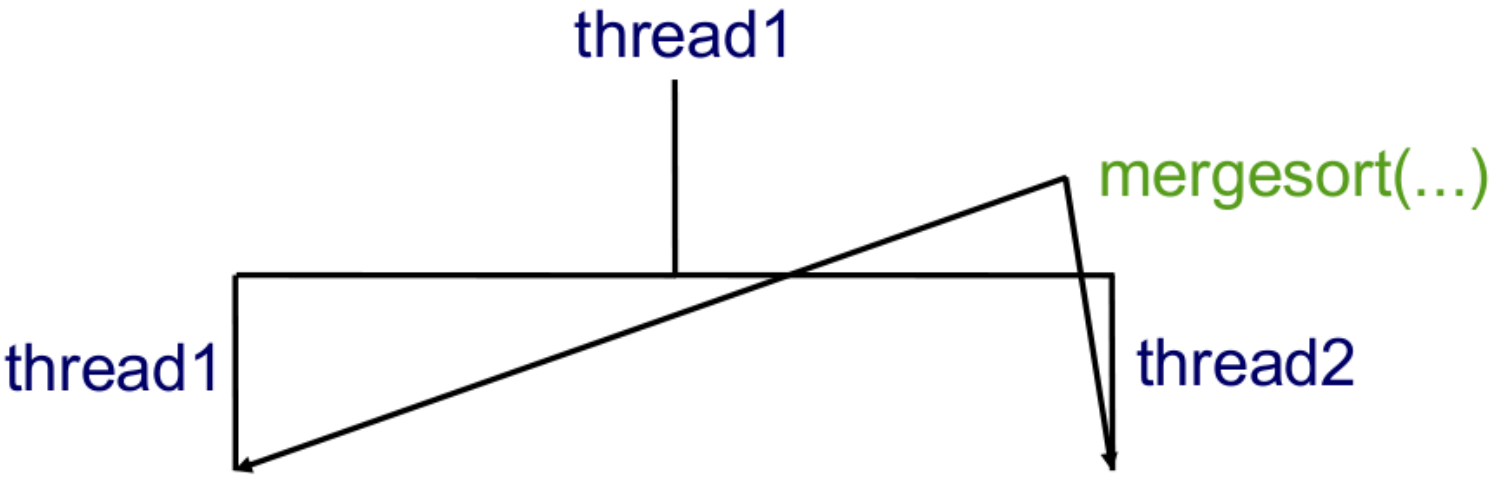
```
void mergesort (int start, int end) {
  if (end - start >= 1) {
    int middle = (start + end) / 2;
    mergesort(start, middle);
    mergesort(middle+1, end);
    merge(start, middle, middle+1, end);
  }
}
```

Recursive Thread Creation

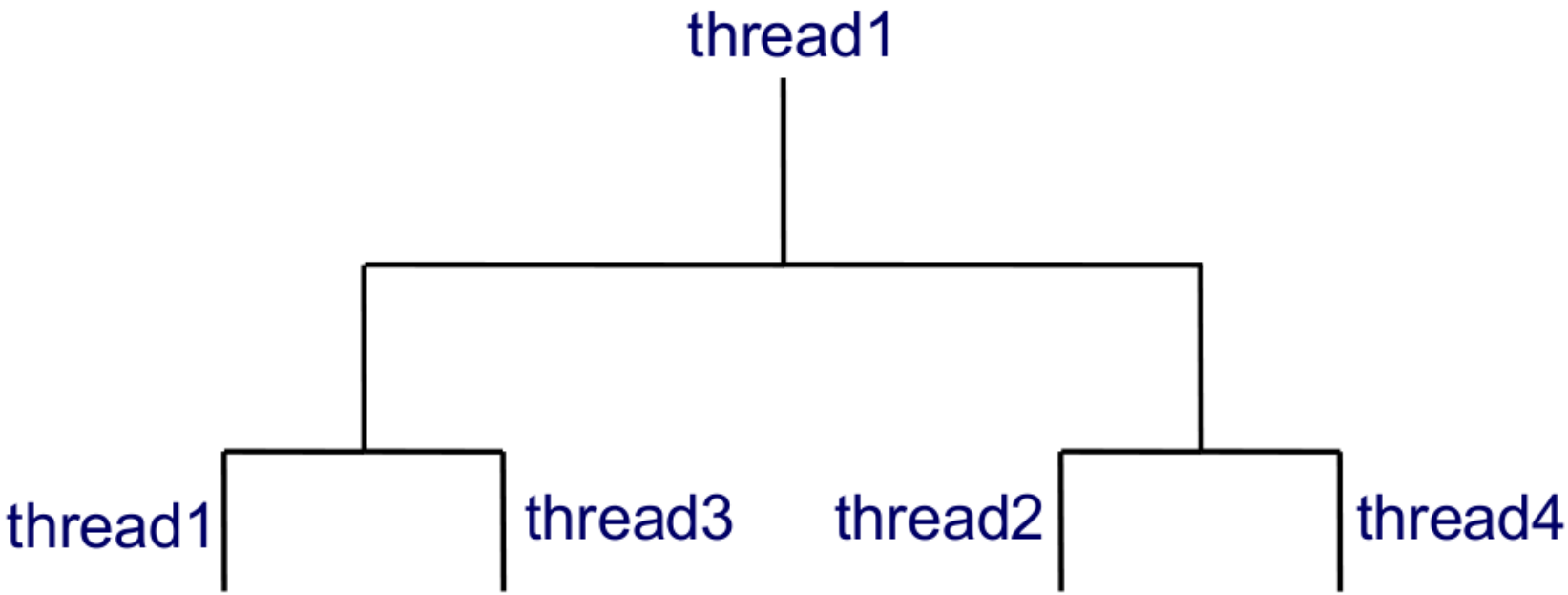
```
void mergesort (int start, int end) {
  if (end - start >= 1) {
    int middle = (start + end) / 2;

    // thread 1 executes the first mergesort
    mergesort(start, middle);

    // thread 2 executes the second mergesort
    mergesort(middle+1, end);

    // join, remaining thread executes merge
    merge(start, middle, middle+1, end);
  }
}
```

Example: Graph with 4 Threads

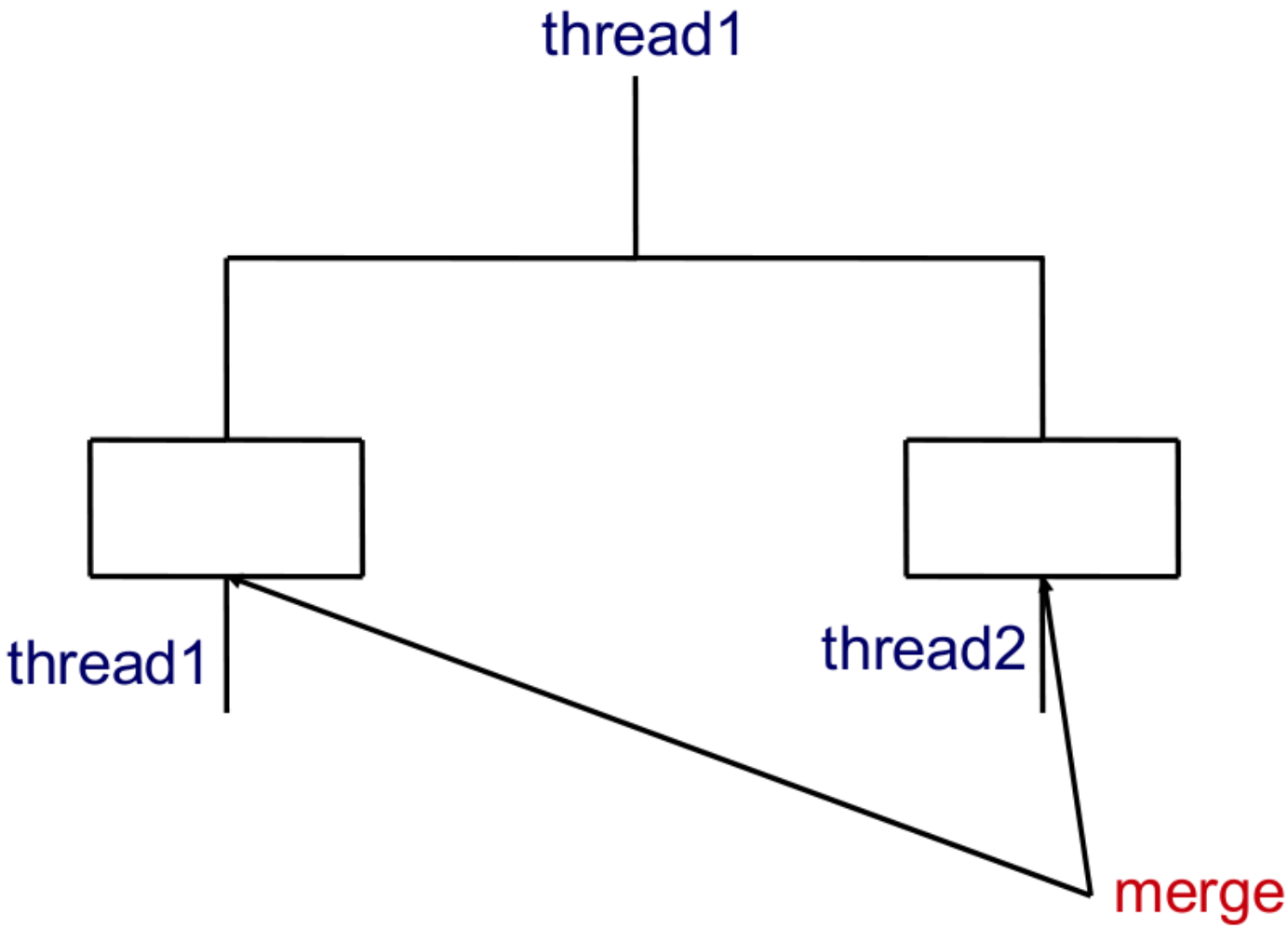
Example: Graph with 4 Threads



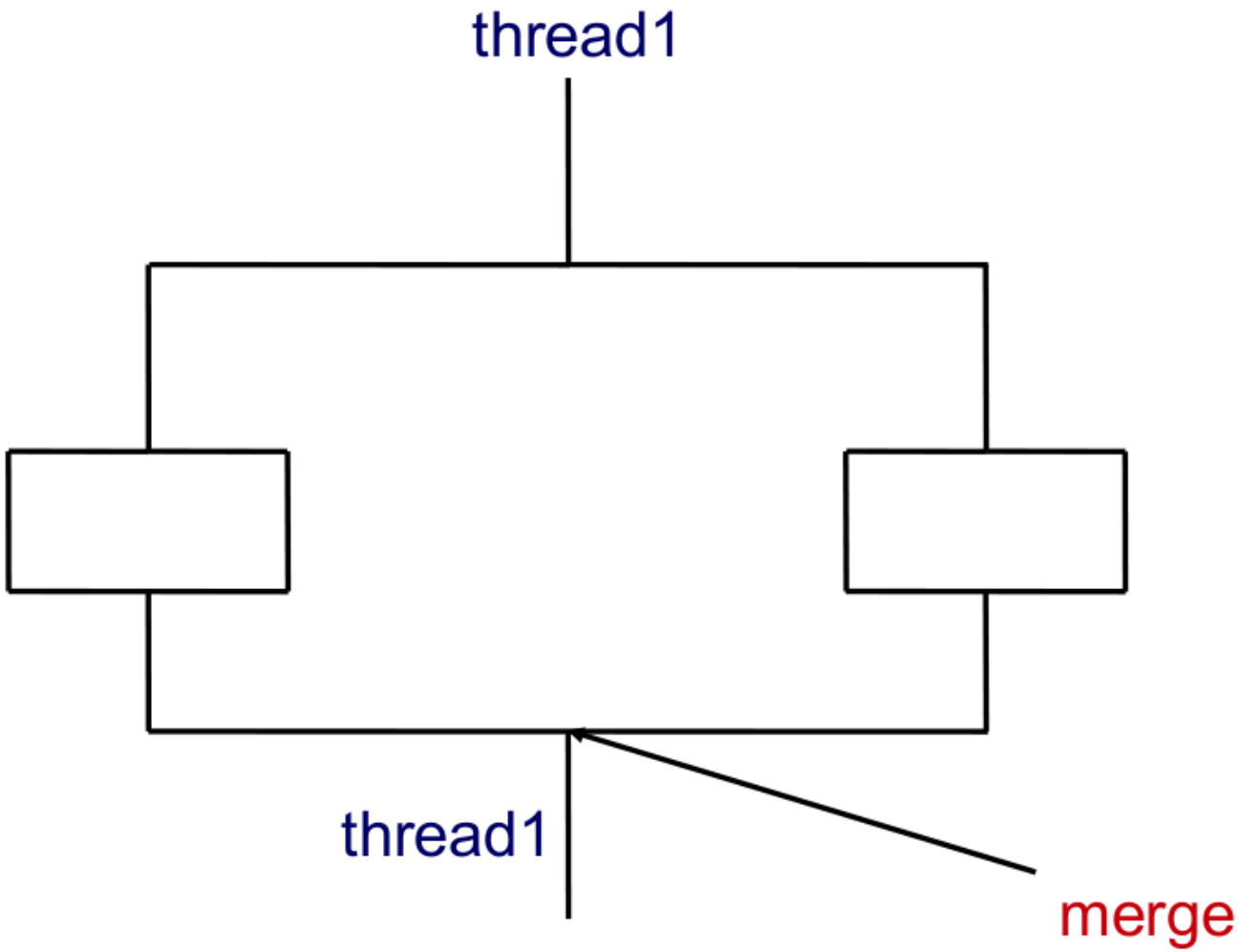
Example: Graph with 4 Threads



Example: Graph with 4 Threads

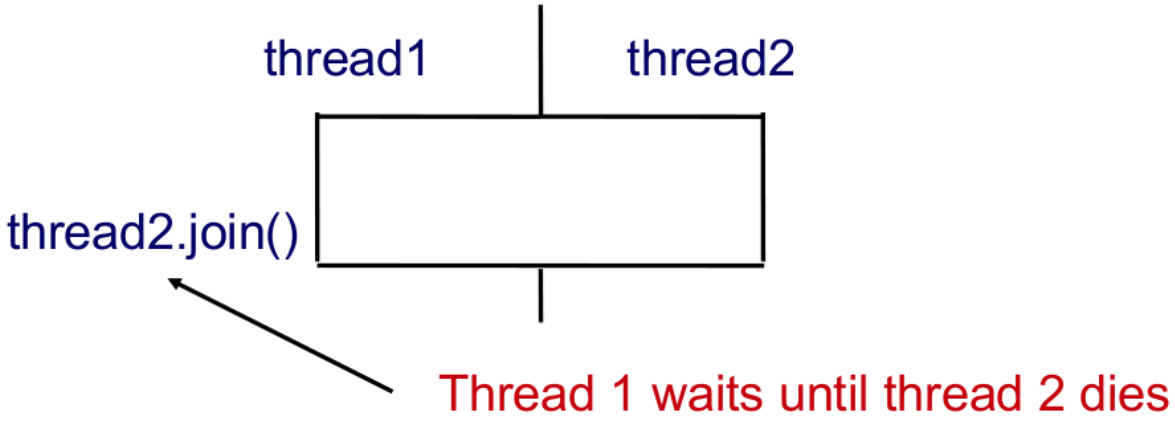


Example: Graph with 4 Threads



Synchronization: Join

- `join()` waits for this thread to die
- Exception: If another thread has interrupted the current thread



Solution

Let's solve it together!

Outline

Measurements

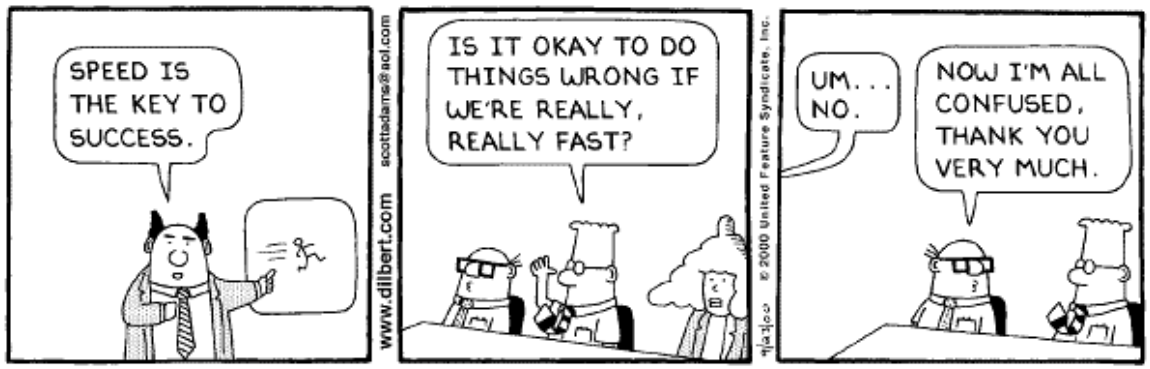
```
int_to_sort = createRandIntArray(no_elements);
MTMergeSort m = new MTMergeSort(int_to_sort,
                                no_threads);
// start timing
long start = System.currentTimeMillis();
m.sort();
// stop timing
long stop = System.currentTimeMillis();
long time = stop - start;
```

Extra keys for Java VM:

- -Xms<size>: set initial Java heap size (eg. to 1024M)
- -Xmx<size>: set maximum Java heap size (eg. to 2048M)

Questions

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



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

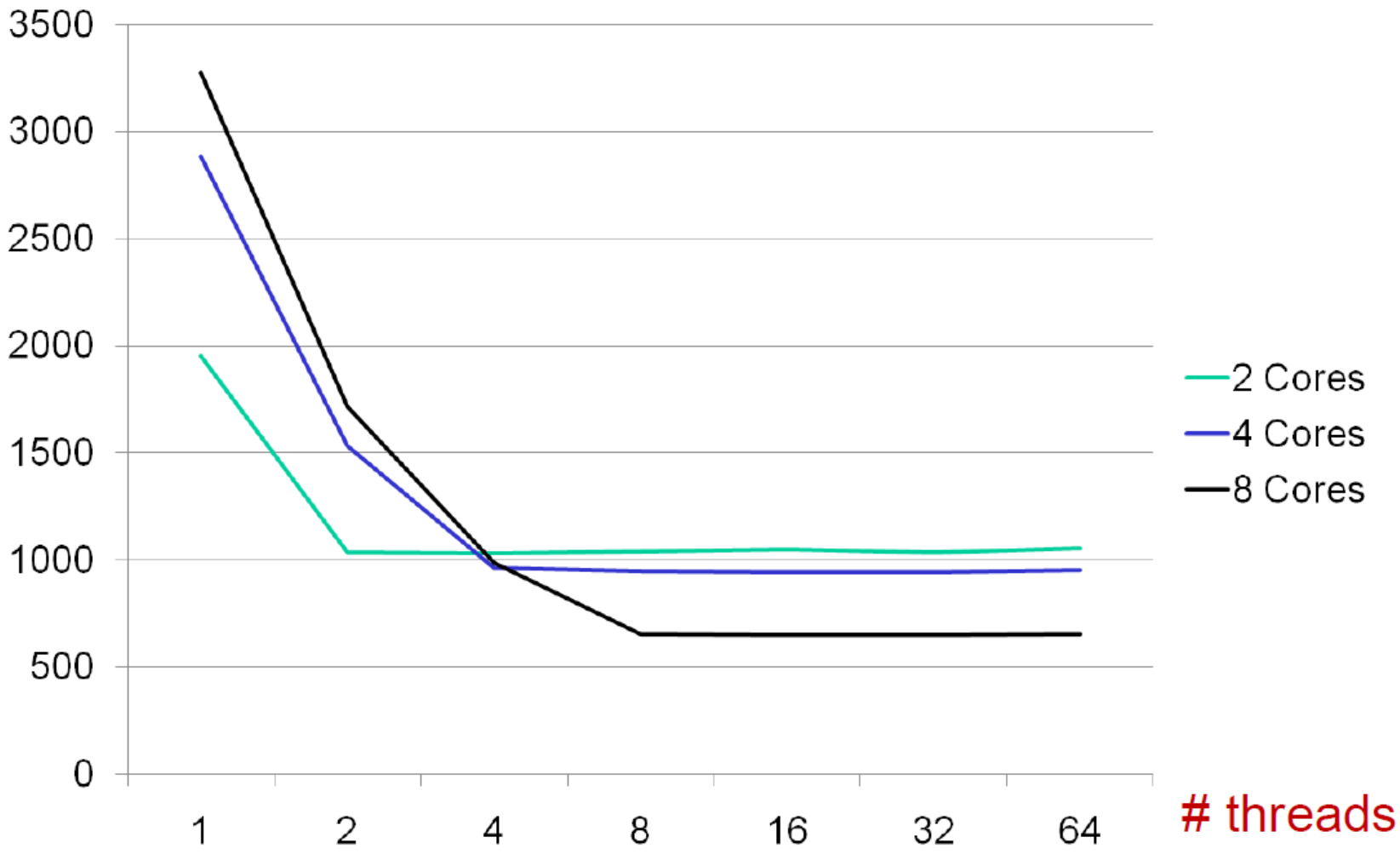
Threads	1	2	4	8	16	32	64
100 000	13	7	7	7	8	10	12
10 000 000	1951	1034	1029	1040	1050	1036	1054

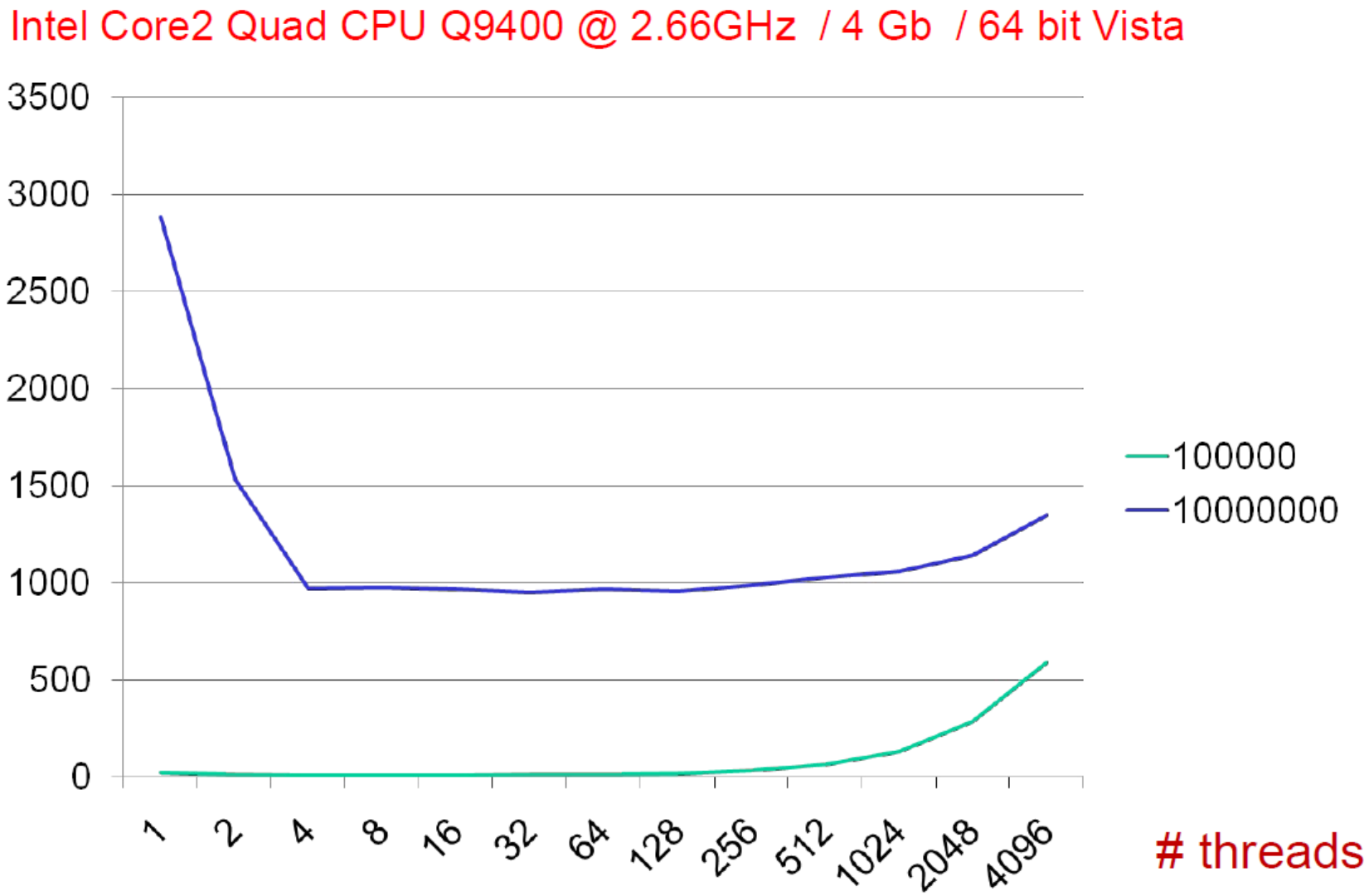
Intel Core2 Quad CPU Q9400 @ 2.66GHz / 4 Gb / 64 bit Vista

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


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


Intel Core2 Quad CPU Q9400 @ 2.66GHz / 4 Gb / 64 bit Vista
3500
3000
2500
2000
1500
1000
500
0
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
100000
10000000
threads

Outline

Matrix Multiplication

- Problem: Given two matrices **A**, **B** of size $N \times N$.
- Compute the matrix product $\mathbf{C} = \mathbf{AB}$ with

$$\mathbf{C}_{ij} = \sum_{k=0}^{N-1} \mathbf{A}_{ik} \cdot \mathbf{B}_{kj}$$

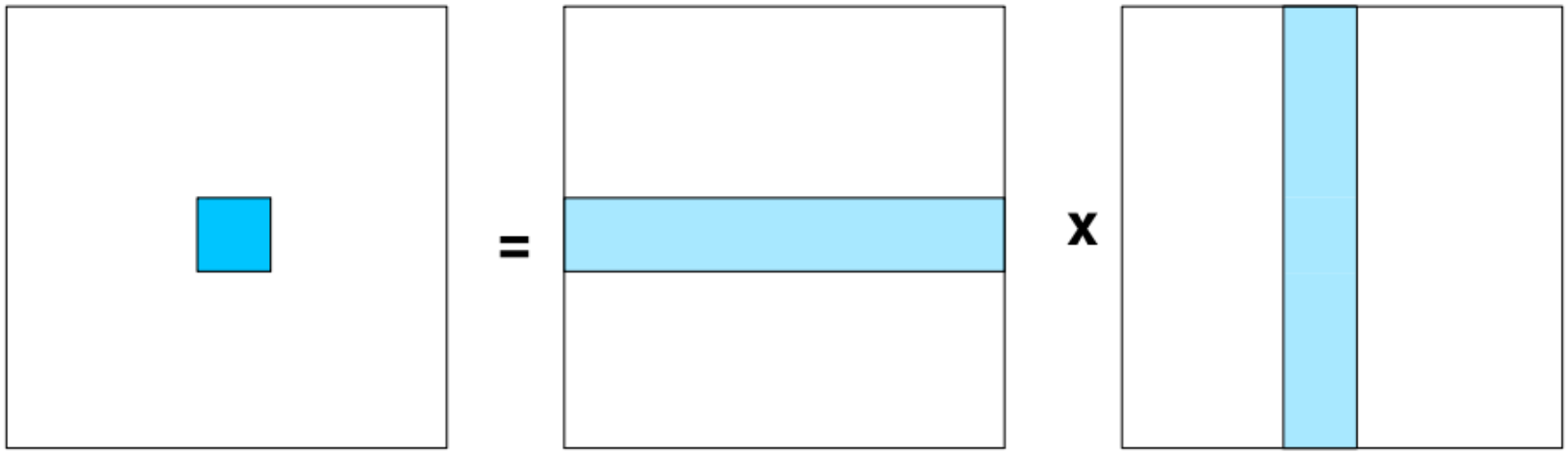
- **A**, **B** elements are double-precision floating point numbers (`double`)

Dense Matrices

- Assume that **A** and **B** are dense matrices
- Sparse matrices have many zero elements
  - Only the non-zero elements are stored
- Dense matrices have mostly non-zero elements
- Each matrix requires N^2 storage cells

Parallel Matrix Multiplication

Which operations can be done in parallel?



Programming Matrix Multiplication

Java code for the loop nest is easy:

```
double[][] a = new double[N][N];
double[][] b = new double[N][N];
double[][] c = new double[N][N];

for (i=0; i<N; i++) {
  for (j=0; j<N; j++) {
    a[i][j] = rand.nextDouble();
    b[i][j] = rand.nextDouble();
    c[i][j] = 0.0;
  }
}

for (i=0; i<N; i++) {
  for (j=0; j<N; j++) {
    for (k=0; k<N; k++) {
      c[i][j] += a[i][k] * b[k][j];
    }
  }
}
```

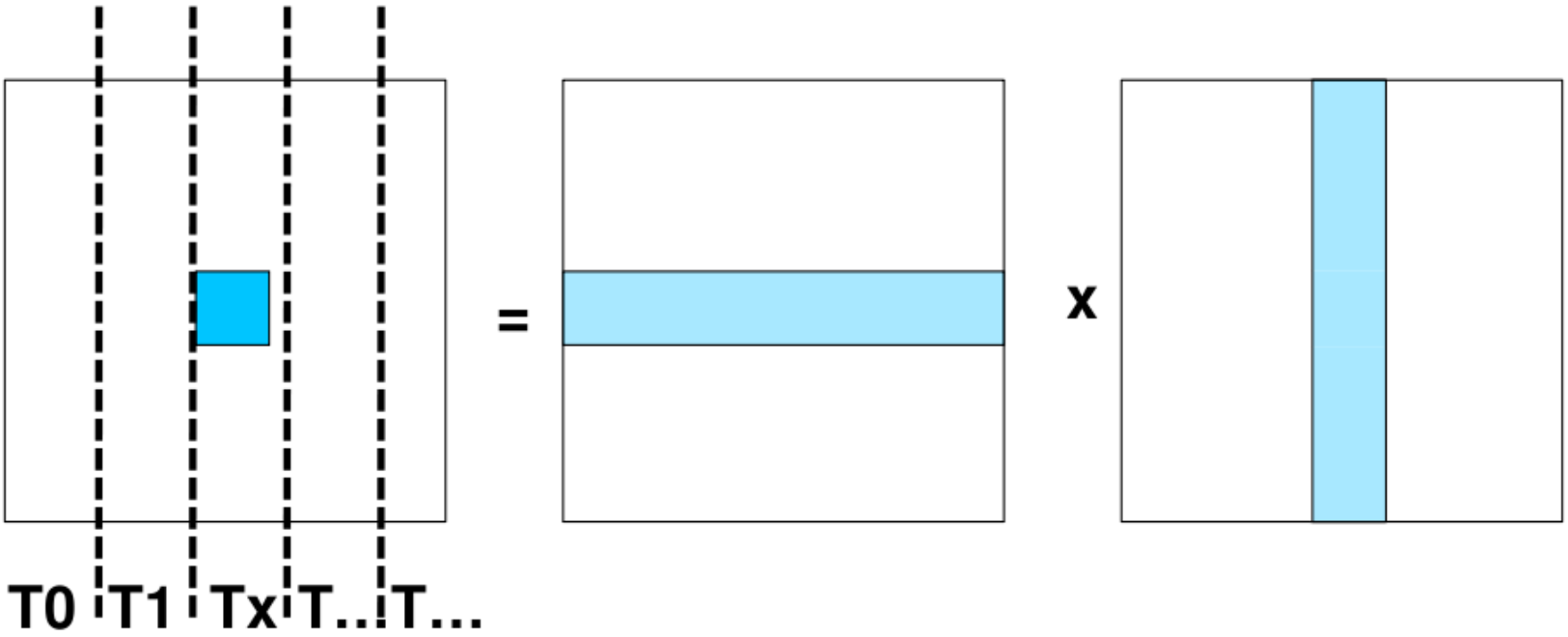
Parallel Matrix Multiplication

- Data partitioning based on
  - Input matrix **A**
  - Input matrix **B**
  - Output matrix **C**
- We assume that all threads can read inputs **A** and **B**
  - Start with partitioning of output matrix **C**
  - No need to use `synchronized`!

Parallel Matrix Multiplication

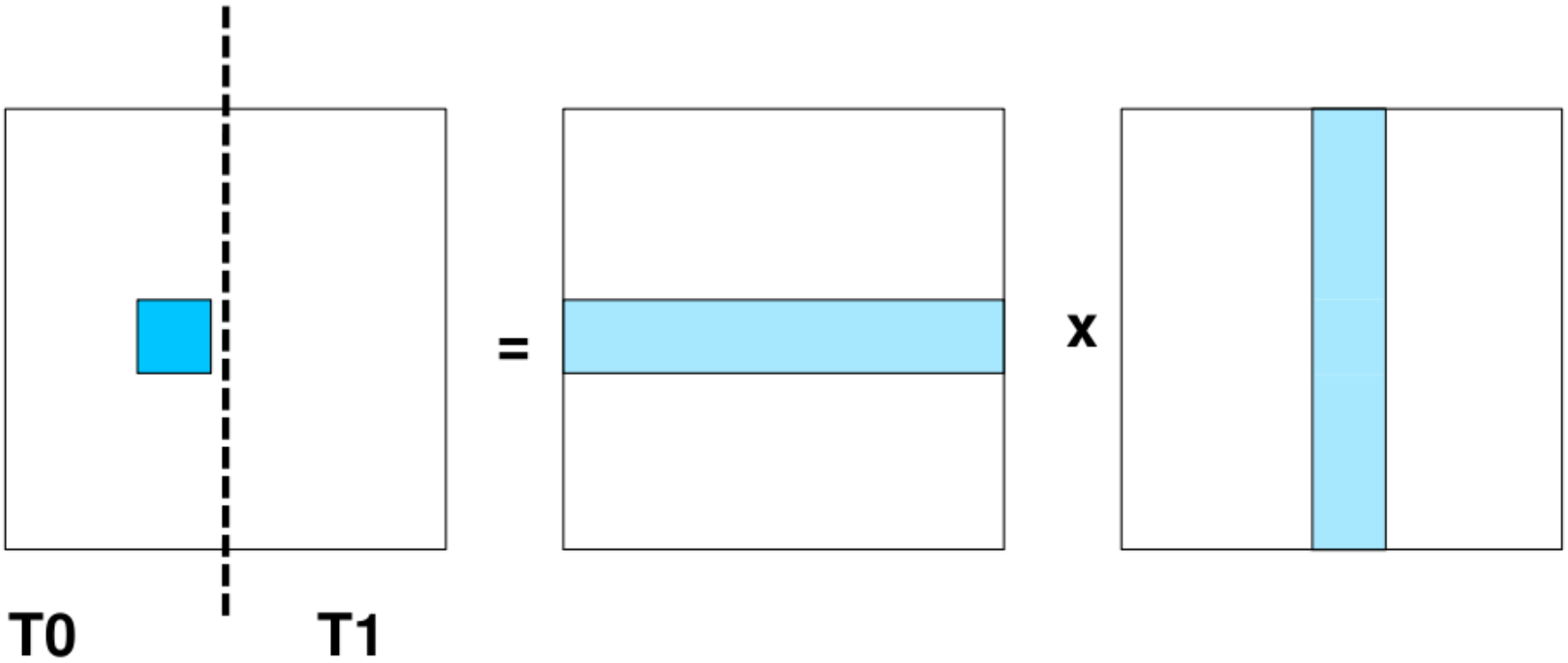
Each thread computes its share of the output **C**

Partition **C** by columns



Two Threads

One thread computes columns $0 \ldots \frac{N}{2}$, the other columns $\frac{N}{2} + 1 \ldots N - 1$

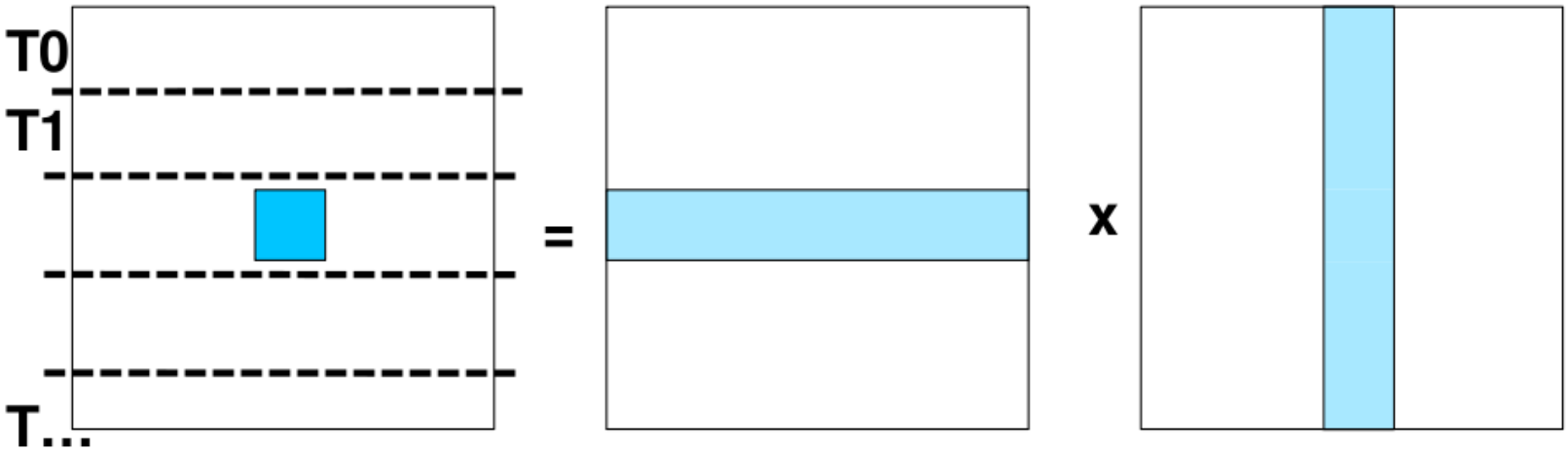


Two Threads

```
// Thread 0
for (i=0; i<N; i++) {
  for (j=0; j<N; j++) {
    for (k=0; k<N/2; k++) {
      c[i][j] += a[i][k] * b[k][j];
    }
  }
}
// Thread 1
for (i=0; i<N; i++) {
  for (j=0; j<N; j++) {
    for (k=N/2; k<N; k++) {
      c[i][j] += a[i][k] * b[k][j];
    }
  }
}
```

Other Aspects

Partition **C** by columns or by rows?



Other Aspects

What should be the order of the loops?

```
for (i=0; i<N; i++) {
  for (j=0; j<N; j++) {
    for (k=0; k<N; k++) {
      c[i][j] += a[i][k] * b[k][j];
    }
  }
}

for (k=0; i<N; i++) {
  for (i=0; j<N; j++) {
    for (j=0; k<N; k++) {
      c[i][j] += a[i][k] * b[k][j];
    }
  }
}
```

Performance Measurement

Matrix size	Number of threads								
	1	2	4	8	16	32	64	...	1024
100									
200									
...									
$10'000$									

One Thread per Matrix Element

Classroom exercise

Don't try this at home!

- Threads Require resources
  - Memory for stacks
  - Setup, teardown
- Scheduler overhead
- Worse for short-lived threads

Summary

- Parallel MergeSort
- Performance issues
- Parallel Matrix multiplication