Parallel Programming
0024

Week 10

Thomas Gross

Spring Semester 2010

May 20, 2010

Outline

- Evaluation
- Discussion of Homework 09
- Presentation of Homework 10
  - OpenMP revisited
  - JOMP
  - Block Matrix Multiplication
- Questions?

Evaluation – Vielen Dank ☺

- 16 Fragebögen

+ gut vorbereitet

+ kompetent

+ begeistert

+ freundlich und hilfsbereit

+ gute Fragen

- Englische Aussprache

+ interessante und informative Beispiele

+ nicht nur Powerpoint

+ gutes Arbeitsklima

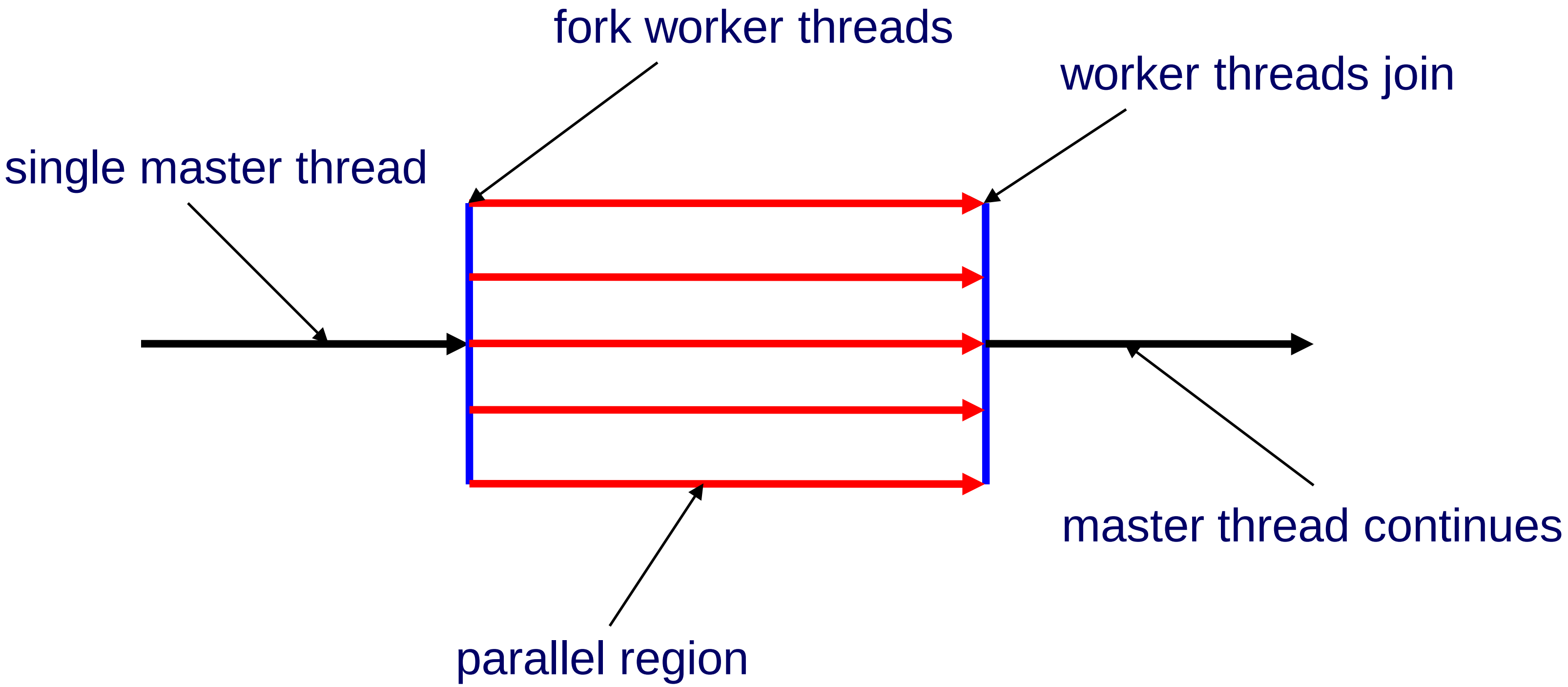
+ gute Erklärungen

OpenMP in a Nutshell

- OpenMP is an API that consists of three parts
  - Directive-based language extension
  - Runtime library routines
  - Environment variables
- Three categories of language extensions
  - Control structures to express parallelism
  - Data environment constructs to express communication
  - Synchronization constructs for synchronization

Parallel Control Structures

- Alter flow of control in a program
  - fork/join model



Parallel Control Structures

- Two kinds of parallel constructs
  - Create multiple threads (parallel directive)
  - Divide work between an existing set of threads
- Parallel directive
  - Start a parallel region
- For directive
  - Exploit data-level parallelism (parallelize loops)
- Sections directive
  - Exploit thread-level parallelism (parallelize tasks)
- (Task directive (OpenMP 3.0))
  - Task with ordering (not possible with sections)

Communication & Data Environment

- Master thread (MT) exists the entire execution
- MT encounters a parallel construct
  - Create a set of worker threads
  - Stack is private to each thread
- Data Scoping
  - Shared variable: single storage location
  - Private variable: multiple storage locations (1 per thread)

Synchronization

- Co-ordination of execution of multiple threads
- Critical directive: implement mutual exclusion
  - Exclusive access for a single thread
- Barrier directive: event synchronization
  - Signal the occurrence of an event

Exploiting Loop-Level Parallelism

- Important: program correctness
- Data dependencies:
  - If two threads read from the same location and at least one thread writes to that location
    - Data dependence

Exploiting Loop-Level Parallelism

- Important: program correctness
- Data dependencies:
  - If two threads read from the same location and at least one thread writes to that location
    - Data dependence
  - Example

Loop carried dependence

```
for (i = 1; i < N; i++)
   a[i] = a[i] + a[i - 1];
```

Can the loops be parallelized?

```
for (i = 1; i < n; i+= 2)
  a[i] = a[i] + a[i – 1]
```

Can the loops be parallelized?

```
for (i = 1; i < n; i+= 2)
  a[i] = a[i] + a[i – 1]
```

No dependence

Can the loops be parallelized?

```
for (i = 1; i < n; i+= 2)
  a[i] = a[i] + a[i – 1]
```

No dependence

```
for (i = 0; i < n/2; i++)
  a[i] = a[i] + a[i + n/2]
```

Can the loops be parallelized?

```
for (i = 1; i < n; i+= 2)
   a[i] = a[i] + a[i – 1]
```

No dependence

```
for (i = 0; i < n/2; i++)
   a[i] = a[i] + a[i + n/2]
```

No dependence

Can the loops be parallelized?

```
for (i = 1; i < n; i+= 2)
  a[i] = a[i] + a[i – 1]
```

No dependence

```
for (i = 0; i < n/2; i++)
  a[i] = a[i] + a[i + n/2]
```

No dependence

```
for (i = 0; i < n/2+1; i++)
  a[i] = a[i] + a[i + n/2]
```

Can the loops be parallelized?

```
for (i = 1; i < n; i+= 2)
   a[i] = a[i] + a[i – 1]
```

No dependence

```
for (i = 0; i < n/2; i++)
   a[i] = a[i] + a[i + n/2]
```

No dependence

```
for (i = 0; i < n/2+1; i++)
   a[i] = a[i] + a[i + n/2]
```

Dependence:
read(0+n/2)
write(n/2)

Important directives for the assignment

```
//omp parallel shared (a,b)
  private (c,d)
```

- Starts a parallel region
- Shared: variable is shared across all threads
- Private: each thread maintains a private copy

Important directives for the assignment

`//omp parallel shared (a,b) private (c,d)`

- Starts a parallel region
- Shared: variable is shared across all threads
- Private: each thread maintains a private copy

`//omp for schedule(dynamic or static)`

- Distribute loop iterations to worker threads
- Dynamic: loop-chunks are assigned to threads at runtime
- Static: loop-chunk assignment before the loop is executed

Important directives for the assignment

- //omp critical
  - Code section is executed by a single thread at a time

Assignment 10

- Task 1
  - Parallelize an existing implementation with OpenMP
  - Which loop nest would you parallelize?
- Do you need a critical section?
- Task 2
  - Implement a Block Matrix Multiplication
  - Divide the source matrices into sub-matrices
  - Assign a thread to each sub-matrix
- Which one performs better?
- Due: 1 Week

OpenMP in Java

- Not natively supported by Java
- JOMP: source to source compiler
- How to use?
  - Download jar file from course page
  - Import external jar to your project (classpath)
  - Perform the following steps
    - java jomp.compiler.Jomp file(.jomp) -> file.java
    - javac file.java
    - java file

Any questions?