Module 3 Quiz

,

People are making progress

154 learners have recently completed this assignment

Submit your assignment
Due DateDec 28, 12:59 AM MST
Attempts3 every 8 hours
Receive grade
To Pass70% or higher
Grade
—

Module 3 Quiz
Graded Quiz • 30 min

Due Dec 28, 12:59 AM MST

Module 3 Quiz

Total points 10
1.
Question 1

Given the following sequential code fragment and assuming N > 2, which of the possible approaches using forall loops (pseudocode introduced in Lecture 3.1) will produce functionally equivalent values in the arrays x and y? Note that a range like "i : [0 : N]" in the forall pseudocode is assumed to be an inclusive range that includes both the lower bound (0) and the upper bound (N).

```
for (i=0; i <= N; i = i + 1) {
  x[i] = x[i] + y[i];
  y[i+1] = w[i] + z[i];
}
```

1 point
A.

```
forall (i : [0 : N]) {
  x[i] = x[i] + y[i];
  y[i+1] = w[i] + z[i];
```

```
}
```

B.

```
forall (i : [0 : N]) {
  x[i] = x[i] + y[i];
}
forall (i : [0 : N]) {
  y[i+1] = w[i] + z[i];
}
```

C.

```
x[0] = x[0] + y[0];
forall (i : [0 : N-1]) {
  y[i+1] = w[i] + z[i];
  x[i+1] = x[i+1] + y[i+1];
}
y[N+1] = w[N] + z[N];
```

2.

Question 2

Given the following sequential code fragment:

```
c = 0;
for (i = 0; i <= N; i++) {
  c = c + a[i];
}
println("c = " + c);
```

and the following attempt to parallelize the fragment using a forall loop (introduced in Lecture 3.1):

```
c = 0;
forall (i : [0 : N]) {
  c = c + a[i];
}
println("c = " + c);
```

Which of the following statements is true, related to the determinism properties introduced in Lecture 2.5?

1 point

A. The parallel code exhibits data races and structural determinism, but not functional determinism.

B. The parallel code exhibits data races and functional determinism, but not structural determinism.

C. The parallel code exhibits functional and structural determinism, and no data races.

D. The parallel code exhibits data races, but not functional or structural determinism.

3.
Question 3

Assume that forall is implemented using a finish scope containing a sequential for loop in which each iteration is implemented as a parallel async task.

Given the following two versions of code that attempt to parallelize a matrix multiplication computation (introduced in Lecture 3.2). We now use a slightly different notation for forall loops that corresponds to actual code (in the PCDP library) rather than pseudocode. The lower and upper bound parameters for the forall constructs still represent inclusive ranges.

```
// Version 1
forall(0, N - 1, 0, N - 1, (i, j) -> {
  C[i][j] = 0;
  for (int k = 0; k < N; k++) {
    C[i][j] += A[i][k] * B[k][j];
  }
});
```

And

```
// Version 2
forall(0, N - 1, (i) -> {
  forall(0, N - 1, (j) -> {
    C[i][j] = 0;
    for (int k = 0; k < N; k++) {
      C[i][j] += A[i][k] * B[k][j];
    }
  });
```

});

Which of the following statements are true for Versions 1 and 2?

1 point

A. Version 2 will create fewer finish scopes than version 1

B. Version 2 will create fewer async tasks than version 1

C. Version 1 and Version 2 will perform the same total number of multiply operations (from line 5 in version 1, and line 6 in version 2)

D. Version 2 will have a longer critical path than Version 1, if we assume that each multiply operation corresponds to 1 unit of work

4.

Question 4

Recall that barriers were introduced in Lecture 3.3. True or false, the following code snippet that uses barriers can exhibit a data race?

```
sum = 0;
forall (0, 2, (i) -> {
  if (i == 0) {
    sum += i;
  }
  barrier;
  if (i == 1) {
    sum += i;
  }
  barrier;
  if (i == 2) {
    sum += i;
  }
});
```

1 point

True

False

5.

Question 5

Recall that barriers were introduced in Lecture 3.3. Which of the choices is a legal ordering of the print statements in the code snippet below?

```
forall (0, 1, (i) -> {
  println(“Hello from iteration “ + i);
  barrier;
  println(“Continuing iteration “ + i);
  barrier;
  println(“Finishing iteration “ + i);
});
```

1 point

A.

Hello from iteration 0

Continuing iteration 0

Finishing iteration 0

Hello from iteration 1

Continuing iteration 1

Finishing iteration 1

B.

Hello from iteration 0

Hello from iteration 1

Continuing iteration 0

Finishing iteration 0

Continuing iteration 1

Finishing iteration 1

C.

Hello from iteration 1

Hello from iteration 0

Continuing iteration 1

Continuing iteration 0

Finishing iteration 0

Finishing iteration 1

6.
Question 6

Consider the code below, and recall that barriers were introduced in Lecture 3.3. Which of the choices is a functionally equivalent barrier-based parallel program?

```
forall (0, 3, (i) -> {
  sum[i] = i;
});
forall (0, 3, (i) -> {
  sum[i] += sum[i + 1];
});
```

1 point

A.

```
forall (0, 3, (i) -> {
  sum[i] = i;
  barrier;
  sum[i] += sum[i + 1];
});
```

B.

```
forall (0, 3, (i) -> {
  sum[i] = i + sum[i + 1];
});
```

C.

```
forall (0, 3, (i) -> {
  sum[i] = sum[i + 1];
  barrier;
  sum[i] += i;
});
```

7.
Question 7

What was the primary benefit of using barriers in the one-dimensional iterative averaging example studied in Lecture 3.4?

1 point

A. Barriers were necessary for the correct implementation of one-dimensional iterative averaging. It could not be implemented without them.

B. Fewer tasks had to be created when we made use of barriers, leading to lower overhead

C. Barriers reduced the amount of non-determinism caused by data races in the version of the code that did not use barriers.

D. Barriers led to the creation of more tasks, allowing the parallel runtime more flexibility in scheduling tasks.

8.
Question 8

Which of the following is true about iteration grouping/chunking for parallel loops (as introduced in Lecture 3.5)?

1 point

A. Loop chunking increases the amount of work performed by the parallel loop and reduces the number of tasks created.

B. Loop chunking reduces the amount of work performed by the parallel loop and reduces the number of tasks created.

C. Loop chunking does not affect the amount of work performed by the parallel loop and increases the number of tasks created.

D. Loop chunking does not affect the amount of work performed by the parallel loop and reduces the number of tasks created.

9.
Question 9

Given the sequential code snippet below:

```
for (i = 1; i <= 100; i++) {
  a[i] = b[i] + c[i + 10];
}
```

And four parallel versions of the above code snippet, which of the provided parallel versions is correct?

A.

```
// No chunking/grouping is performed in this version
forall (1, 100, (i) -> {
  a[i] = b[i] + c[i + 10];
});
```

B.

```
// Chunking is performed, but the chunk size is unknown
forallChunked (1, 100, (i) -> {
  a[i] = b[i] + c[i + 10];
 });
```

C.

```
chunkSizeDivides100 = 10;
// Chunk size is provided in the third parameter below
forallChunked (1, 100, chunkSizeDivides100, (i) -> {
  a[i] = b[i] + c[i + 10];
});
```

D.

```
chunkSizeDoesNotDivide100 = 23;
// Chunk size is provided in the third parameter below
forallChunked (1, 100, chunkSizeDoesNotDivide100, (i) -> {
  a[i] = b[i] + c[i + 10];
});
```

1 point

A. Snippets A, B, and C are correct but not D.

B. Snippets A and C are correct but not B and D.

C. Only snippet C is correct.

D. All of the parallel snippets are correct. (ALL OTHERS CHOICES WERE MARKED INCORRECT)

10.

Question 10

In general, recalling the contents of Lecture 3.5, what is a good heuristic for setting the number of chunks in a forall parallel loop?

1 point

A. The number of chunks should be similar to the number of hardware cores in the platform.

B. The number of chunks should group together as many iterations in to a single chunk as there are hardware cores in the platform.

C. The number of chunks should be an order of magnitude smaller than the number of loop iterations.

D. The number of chunks should always be between 10 and 20.

Honor Code Agreement