CIS-800-003: Topics in Parallel Programmability

Joe Devietti
9 Jan 2013

and now, your host...

- devietti@cis
- Office hours: by appointment
- Levine 572

Anatomy of a class

- short presentation on paper(s) - 20 minutes
- discussion questions - 45 minutes
- [optional] context for next research topic - 15 minutes

Course Mechanics (1/2)

- paper presentations
- “reading quizzes” on Blackboard
  - a few questions about each paper
  - due the morning before class

Course Mechanics (2/2)

- Future Work™ Fridays!
  - half a page on an idea related to a paper we've read that week
  - due most Fridays, via Blackboard
- Larger future work write-up
  - 2 pages
  - due at the end of the semester
  - upgrade one of your previous ideas, or something new

Course Mechanics (3/2)

- no exams or projects
- no stress!

Sequential Consistency

What is sequential consistency?

What is sequential consistency?

operational

What is sequential consistency?

operational

mathematical

SC = “the most intuitive memory model”

SC = “the most intuitive memory model”

```
byte b = 8;
```

SC = “the most intuitive memory model”

```
byte b = 8;

long x = 8;
```

SC = “the most intuitive memory model”

```
byte b = 8;

long x = 8;

x++;
```

SC sample execution

```
x == 0 && y == 0
```

```
x = 1;                y = 1;
r1 = y;               r2 = x;
```

SC sample execution

```
x == 0 && y == 0
```

```
x = 1;              y = 1;
r1 = y;             r2 = x;
```

can r1 == 0 && r2 == 0?

SC sample execution

```
x == 0 && y == 0
```

```
x = 1;              y = 1;
r1 = y;             r2 = x;
```

can r1 == 0 && r2 == 0?

```
x = 1;         y = 1;
r1 = y;        r2 = x;
y = 1;         x = 1;
r2 = x;        r1 = y;
```

SC sample execution

```
x == 0 && y == 0
```

```
x = 1;            y = 1;
r1 = y;           r2 = x;
```

can r1 == 0 && r2 == 0?

```
x = 1;       y = 1;       x = 1;       x = 1;
r1 = y;      r2 = x;      y = 1;       y = 1;
y = 1;       x = 1;       r2 = x;      r1 = y;
r2 = x;      r1 = y;      r1 = y;      r2 = x;
```

SC sample execution

```
x == 0 && y == 0
```

```
x = 1;              y = 1;
r1 = y;             r2 = x;
```

can r1 == 0 && r2 == 0?

```
x = 1;
r1 = y;
y = 1;
r2 = x;
```

```
y = 1;
r2 = x;
x = 1;
r1 = y;
```

```
x = 1;
y = 1;
r2 = x;
r1 = y;
```

```
x = 1;
y = 1;
r1 = y;
r2 = x;
```

```
y = 1;
x = 1;
r2 = x;
r1 = y;
```

```
y = 1;
x = 1;
r1 = y;
r2 = x;
```

double-checked locking

```
class Foo {
   private Singleton s = null;
   public Singleton getS() {
      if (s == null) {
         s = new Singleton();
      }
      return s;
   }
}
```

double-checked locking

synchronized

```
class Foo {
   private Singleton s = null;
   public Singleton getS() {
      if (s == null) {
         s = new Singleton();
      }
      return s;
   }
}
```

Turn-based mutual exclusion

```
turn = 0;
```

```
while (turn != me) {}
// critical section
turn = (turn+1) % NUM_THREADS;
```

Dekker’s algorithm

```
                    flag[0] = false
                    flag[1] = false
                    turn    = 0

flag[0] = true;
while (flag[1] == true) {
   if (turn ≠ 0) {
      flag[0] = false;

      while (turn ≠ 0) {}

      flag[0] = true;
   }
}

// critical section

turn    = 1;
flag[0] = false;
```

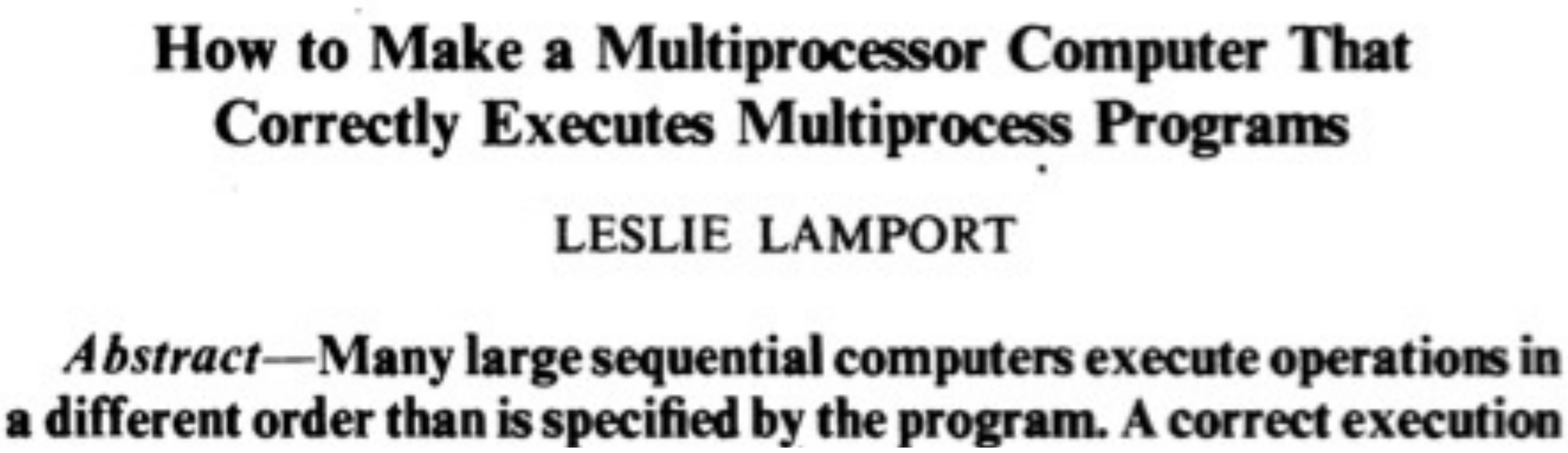
Dekker’s algorithm

```
flag[0] = false
flag[1] = false
turn    = 0
```

```
flag[0] = true;
while (flag[1] == true) {
   if (turn ≠ 0) {
      flag[0] = false;

      while (turn ≠ 0) {}

      flag[0] = true;
   }
}

// critical section

turn    = 1;
flag[0] = false;
```

```
flag[1] = true;
while (flag[0] == true) {
   if (turn ≠ 1) {
      flag[1] = false;

      while (turn ≠ 1) {}

      flag[1] = true;
   }
}

// critical section

turn    = 0;
flag[1] = false;
```

How do we implement SC?

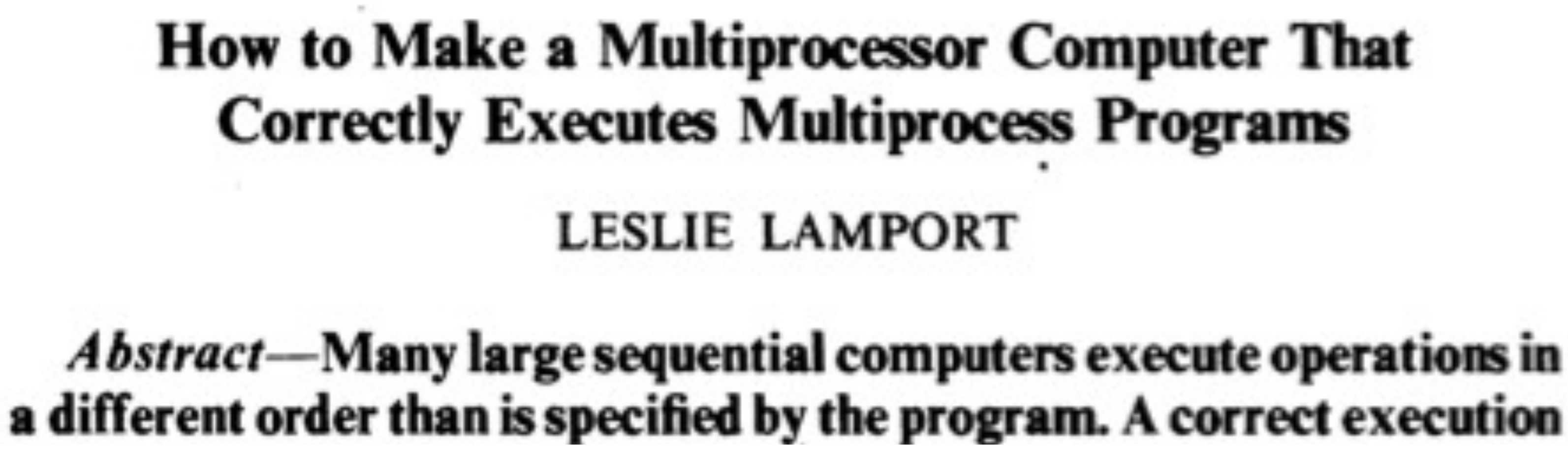


How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs

LESLIE LAMPORT

Abstract—Many large sequential computers execute operations in a different order than is specified by the program. A correct execution

How do we implement SC?



How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs

LESLIE LAMPORT

***Abstract*—Many large sequential computers execute operations in a different order than is specified by the program. A correct execution**

processors issue memory requests in program order

a memory module services requests from a FIFO queue

there may be multiple memory modules