Lecture 3

Embarrassingly parallel applications; parallel for loops; data sharing; analyzing data dependencies
Announcements

• If you are concurrently enrolled, and need a Moodle account, send your request to Professor Baden
Today’s lecture

• Parallel for loops in PJ
• Analyzing data dependencies to understand synchronization requirements
• Application class: “embarrassingly parallel”
• Application: parallel key search
Parallel for loops

• Most parallelism arises within for loops
• PJ provides for loop classes
Simplest for loop

- Multiply two vectors for all $i = 0:N-1$
  
  $$p[i] = a[i] \times b[i]$$

- Each product $p[i]$ is independent
Threads implementation

- Partition the data into intervals, assign each to a unique thread
- Each thread sweeps over a reduced problem
- New PJ construct: for loop object
PJ Implementation

new ParallelTeam().execute (new ParallelRegion() {
    public void run() throws Exception {
        execute (0, n-1, new IntegerForLoop() {
            public void run (int first, int last) {
                for (int i = first; i <= last; ++ i)
                    p[i] = a[i]*b[i];
            }
        });
    }
});

java EP 5
a: 0 1 2 3 4
b: 5 4 3 2 1
p: 0 4 6 6 4
Flow control

new ParallelTeam().execute (new ParallelRegion() {
  public void run() throws Exception {
    execute (0, n-1, new IntegerForLoop() {
      public void run (int first, int last) {
        ....
      }
    });
  }
});
Control flow

- Implicit barrier at end of ForLoop object
- What if the # of threads doesn’t divide iteration range?
- Do not assume that loop indices are assigned the same way in different parallel contexts

```java
new ParallelTeam().execute (new ParallelRegion() {
    public void run() throws Exception {
        execute (0, n-1, new IntegerForLoop() {
            public void run (int first, int last) {
                int t = getThreadIndex();
                System.out.println ("For #1: thread " + t + " gets " + first + " : " + last);
            }
        });
        execute (0, n-1, new IntegerForLoop() {
            public void run (int first, int last) {
                ......
            }
        });
    }
});
```
Dependences

• Our attempt to parallelize the algorithm succeeded: there were no data dependencies
• What if we have loop carried dependences?
• The value of $u[i]$ computed at iteration $i$ depends on $u[i]$ computed in iteration $i-1$

\[
\text{for } i = 1 : N-1 \\
u[i] = (u[i-1]+u[i+1])/2 \\
\text{end for}
\]
Under the hood

• Create an instance of a concrete subclass of class ParallelRegion (create an anonymous inner class)
• Pass to ParallelTeam::execute()
• ParallelRegion::execute() does the heavy lifting
  ▶ Overloaded for different types of control constructs
  ▶ In this case, we pass the IntegerForLoop describing the control flow we want to realize
  ▶ Barrier executed at the end of ParallelRegion::execute()

```java
class ParallelRegion {
  public void run() throws Exception {
    execute(0, n-1, new IntegerForLoop() {{}})
  }
}
```
Variable scoping and sharing

- Static variables declared in main program’s class are global to all threads
- Variables declared within main() are local to the single thread that executes main()
- Variables declared as instance fields of a ForLoop are local to a single thread
- If one or more threads write shared state, then we have a conflict requiring synchronization (sharing implies at least 1 reader)
AES Key Search
Encryption

• Mozilla Suite, Firefox, and Thunderbird support up to 256-bit encryption (although it’s not commonly used) and is suitable for secure transfers on the Internet. [http://kb.mozillazine.org/128-bit_Encryption](http://kb.mozillazine.org/128-bit_Encryption)

• When you access your account and perform transactions on Citibank Online, we use 128-bit Secure Sockets Layer (SSL) encryption technology — the most widely used method of securing internet transactions available today. [http://www.citibank.com/domain/spoof/sitesecurity.htm](http://www.citibank.com/domain/spoof/sitesecurity.htm)
How encryption works

128 bit blocks

256 bits

128 bit blocks

http://www.cryptographyworld.com/what.htm
How encryption works

- Cleartext "Morpheus"
- CYBERTEXT "*@:$!#1a3"
- KEY
- 256 bits

- e.g. AES
- 128 bit blocks
Cracking the code

• Ciphertext appears random
• What if we knew a plaintext-ciphertext pair?
• Could we guess the key in a known plaintext attack?
• We just iterate through all the keys until we generate the ciphertext from the plaintext
• If we could do $2^{30}$/second ➔ $2^{226}$ seconds = $10^{60}$ years
Another strategy

• We have hope if we know part of the key
  27417fbbab39b8f54333a739b539954da24bf52b3dbd6f58edf0e71e2357****

• Natural embarrassingly parallel algorithm for all possible keys
  encrypt the plain text
  if result matches known ciphertext,
  report the key
end for all
PJ Implementation

Copy 8 lsbs of partialkey[28:31] -> keylsbs;
new ParallelTeam().execute (new ParallelRegion() { // Do trial encryptions in parallel
  public void run() throws Exception {
    execute (0, maxcounter, new IntegerForLoop() { // maxcounter = (1<<n)-1;
      byte[] trialkey, trialCiphertext; AES256Cipher cipher; // Per-thread vars
      // Try every possible combination of low-order key bits.
      public void run (int first, int last) {
        // Set up thread local variables.
        trialkey[0:32] <- partialkey[0:32]; cipher = new AES256Cipher (trialkey);
        for (int cntr = first; cntr <= last; ++cntr) {
          int lsbs = keylsbs | cntr; // Fill in lower bits
          Copy 32-bit lsbs into trialkey[28:31];
          cipher.setKey (trialkey); // Try the key
          cipher.encrypt (plaintext, trialCiphertext);
          // If the result equals the ciphertext, then we found the key
          if (match (ciphertext, trialCiphertext))
            foundkey <- trialkey;
        }
    }
  }
  }
  }
}
public void run (int first, int last) {
    // Set up thread local variables.
    trialkey[0:32] <- partialkey[0:32];
    cipher = new AES256Cipher (trialkey);
    for (int cntr = first; cntr <= last; ++ cntr ) {
        int lsbs = keylsbs | cntr; // Fill in lower bits
        Copy 32-bit lsbs into trialkey[28:31];
        cipher.setKey (trialkey); // Try the key
        cipher.encrypt (plaintext, trialCiphertext);
    // If the result equals the ciphertext, then we found the key
        if (match (ciphertext, trialCiphertext))
            foundkey <- trialkey
    }
}
**Performance**

- Runs on ieng6-203
- Times in milliseconds

<table>
<thead>
<tr>
<th></th>
<th>Seq</th>
<th>NT = 1</th>
<th>NT = 2</th>
<th>NT = 4</th>
<th>NT = 8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>21038</td>
<td>17522</td>
<td>8782</td>
<td>4773</td>
<td>2620</td>
</tr>
<tr>
<td></td>
<td>17312</td>
<td>19035</td>
<td>8743</td>
<td>4608</td>
<td>2416</td>
</tr>
<tr>
<td></td>
<td>17231</td>
<td>17515</td>
<td>8809</td>
<td>4609</td>
<td>2373</td>
</tr>
</tbody>
</table>
Early Loop Exit

• A common performance optimization in search problems

• Once we’ve found what we are looking for, we should stop the other threads and exit the program

```c
for (int c = first; c <= last && foundkey == null; ++c) {
    if (match (ciphertext, trialCiphertext))
        foundkey <- trialkey
}
```
Performance with early loop exit

- Runs on ieng6-203
- Times in milliseconds

<table>
<thead>
<tr>
<th></th>
<th>Early loop exit</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq</td>
<td>5907</td>
<td>17231</td>
</tr>
<tr>
<td>NT = 1</td>
<td>6049</td>
<td>17515</td>
</tr>
<tr>
<td>NT = 2</td>
<td>6036</td>
<td>8743</td>
</tr>
<tr>
<td>NT = 4</td>
<td>1709</td>
<td>4608</td>
</tr>
<tr>
<td>NT = 8</td>
<td>1733</td>
<td>2373</td>
</tr>
</tbody>
</table>
Code for the Curious

• On ieng6-203
• $pub/examples/newKeySearch
• Also see $pub/examples/EP.java
Next Time

• Performance Characterization
• Parallel image handling