CS 385 Lab 4 – Wed., Feb. 25, 2009

Summary

In this lab, you will parallelize an algorithm that has two properties we haven't seen before:

- The possibility of significant load imbalance between threads
- Some inter-thread dependencies (the iterations of the loop are not perfectly independent).

Your job is to parallelize a program that scans, character by character, through an array of input text and does the following:

- If the ith character is 'q' or 'Q', it runs an AES-encryption-like algorithm on the 16 characters starting with character i, then copies the result into an output array, and then continues the main loop with character (i+16).
- Otherwise, it copies the character into the output buffer, unmodified.

You should also add timing routines around the main for-loop to evaluate the performance of your code. I strongly recommend redirecting the output to a file for large values of N or turning it off altogether.

Preparation

Copy the following three files into your directory:

```
~srivoire/cs385/pickup/aes.h
~srivoire/cs385/pickup/aes.cpp
~srivoire/cs385/pickup/l4-stub.cpp
```

Of these files, you should only have to look at l4-stub.cpp.

The starter code

You can skim this section; it just explains the code in l4-stub.cpp, from top to bottom.

The constants at the beginning of the program include the familiar N and P. I recommend that you test your program with N=16. The larger value of N took me 2 minutes to run sequentially on cwolf, so you will probably need a slightly larger value (no more than 10 times the current value) to get the best timing results.

The final constant, `padding`, pads the array with 15 NULL characters. This ensures that we don't run off the end of the array if the last character is a 'Q' and we try to encrypt the next 15 characters. You don't need to modify this constant.

The main program declares an input buffer (character array) and an output buffer, both with size (N + padding). Notice that the buffers are unsigned characters and that the output buffer will not necessarily end up with a null terminator. Therefore,

we must operate on them and print them out character by character rather than as strings.

Next, we initialize the arrays by calling InitBuf. Scroll down to InitBuf now. InitBuf shows 3 ways to initialize the buffer; currently, the first two are commented out. The first one is just for testing with N=16, but you will evaluate your code using both of the remaining methods, plus any other patterns you think are interesting.

Next, we print out the two arrays, first in plaintext and then hexadecimal values of the ASCII characters.

The next part of the program, with the commented-out OpenMP directive, is the main loop, which performs the operation described in the Summary section.

Finally, we print the output array and free the memory used by the arrays.

Deliverables

1. You should get your code working correctly with multiple threads. Because a 'Q' will affect the value of the next 16 characters, which could potentially be operated upon by another thread, you need to think about how to ensure that the correct (encrypted) characters will get stored in the output array. Remember that the array itself is a shared variable accessible to all threads.
2. You should evaluate the performance (timing) of your code on a suitably large array for random input data (the third option in InitData). In a comment at the beginning of your code or in a separate file, report 3 data points, the value of N you used, and the average. Experiment with another scheduling option (either changing the chunk size, static/dynamic, or both) and report the timing results from this option. Can you explain the relative performance? (it's OK if it's just random noise).
3. Repeat Step 2 for the second option in InitData, where the first chunk of the array is initialized to 'q'.

You should submit *yourlastname*L4.cpp to ~srivoire/cs385/submit. If you are successful, you will see your file listed at
http://rivoire.cs.sonoma.edu/cs385/lab4sub.txt

If you did your writeup in a separate file, copy it to the same directory as *yourlastname*L4.txt.

Acknowledgments

The AES encryption code is based on the reference encryption code for AES provided at:
http://www.hoozi.com/Articles/AESEncryption.htm
The principal modifications were to make it reentrant (http://en.wikipedia.org/wiki/Re-entrant_code) and to change the way the encryption routines are invoked.