Visualizing Parallelism in CS 2: Report from a Spring 2012 Early Adopter
Sean Massung and Cinda Heeren, [massung1,c-heeren]@illinois.edu
University of Illinois at Urbana-Champaign, College of Engineering, Department of Computer Science

Introduction
We describe the incorporation of the IEEE-TCPP Curriculum Initiative into CS 2 at the University of Illinois.
We focus on three main parallel programming concepts, each delivered during a two hour discussion/lab session. Material from these lab sections is graded, and some key topics are evaluated on exams in the form of multiple choice questions.
The three lessons explore the basics of parallelism in a visual manner: using OpenMP, race conditions, and reductions.

Lab 1: Intro to Parallelism
In the coding portion of the lab, we expose students to computation across threads.
For one task, students simply write a function that removes the green color component from an image.

\begin{verbatim}
int i = 0; i < width; ++i
output(i, j) = source(i, j);
output(i, j) = output(i, j) - green;
\end{verbatim}

To illustrate the progress of the parallelized code we give the students augmented code.

\begin{verbatim}
for(int i = 0; i < width; ++i) {
#pragma omp parallel for
    for(int j = 0; j < height; ++j) {
        temp = *image(i, j);
        output(i, j) = output(i, j) - temp;
    }
}
\end{verbatim}

Lab 2: Race Conditions
Race conditions are the main topic in the second lab section. Students learn that correctly parallelizing programs does not just consist of blindly pasting a \#pragma on an outer for loop. An incorrect parallel image flipper is below.

\begin{verbatim}
RGBAPixel temp;
#pragma omp sep parallel for
for(int i = 0; i < width; ++i) {
    for(int j = 0; j < height / 2; ++j) {
        temp = *image(i, j);
        *image(i, j) = *image(i, height - j - 1);
        *image(i, height - j - 1) = temp;
    }
}
\end{verbatim}

Provided code creates a histogram of colors, broken down by which thread counted them (0 to 3) and its use in a parallelized code is:

\begin{verbatim}
map<RGBAPixel, int> ret_freq;
#pragma omp critical
for(int i = 0; i < width; ++i) {
    for(int j = 0; j < height; ++j) {
        RGBAPixel temp = *image(i, j);
        int local_freq = ++local_freq[*image(i,j)];
    }
}
\end{verbatim}

Another typical question simply tests knowledge of the term reduction.

1. A reduction performs the same instruction on data across multiple threads.
2. A reduction occurs when private data on individual threads is assembled into a general solution.
3. Reduction is a technique wherein parallelism is applied to the portion of a program that requires the most computation.
4. Reduction is just another term for parallel. Then the speedup for the parallelized code is:
5. None of these is the correct choice.

Lab 3: Reductions
The last lab section introduces a paradigm for solving complex data dependency issues, namely reductions. We present reductions as a general algorithmic technique, so as to provide a stepping stone to understanding the MapReduce programming model.

\begin{verbatim}
map<RGBAPixel, int> local_freq;
#pragma omp parallel for
for(int i = 0; i < width; ++i) {
    for(int j = 0; j < height; ++j) {
        local_freq[*image(i,j)] = ++local_freq[*image(i,j)];
    }
}
#pragma omp critical
map<RGBAPixel, int> ret_freq;
for (int i = 0; i < width; i++)
for (int j = 0; j < height/2; j++)
    ret_freq[*image(i,j)] = local_freq[*image(i,j)];
\end{verbatim}

We ask students to create a PNG color histogram. To do so we simply record the number of pixels of each color. This is trivial in serial, but requires a slightly different approach when applied across many threads, since the sub-problems on each thread must be combined into the whole.

\begin{verbatim}
for(int i = 0; i < 10; ++i)
    for(int j = 0; j < 10; ++j)
        for(int k = 0; k < 10; ++k)
            table[i][j] = (i+1)*(j+1);
\end{verbatim}

Evaluation
Student performance on lab exercises was exemplary with average scores of 95% on all three assignments. Besides grading lab work, questions regarding parallelism were included on exams. Below are examples of the questions we have assessed on various tests over the past two semesters.

Which of the code examples above is/are NOT correctly parallelized?
(a) Only item (i) is incorrect.
(b) Only item (ii) is incorrect.
(c) Only item (iii) is incorrect.
(d) Two of the above examples are incorrect.
(e) All statements (i), (ii), and (iii) are correct.

Another typical question simply tests knowledge of the term reduction, and its use in a parallel context. Again, student performance is not nearly perfect—typically only 50 to 60% of students answer correctly.

What is the best definition of the term reduction? Suppose an algorithm takes 7 seconds to run serially, and 2 seconds to run in parallel. Then the speedup for the parallelized code is:

1. A reduction performs the same instructions on data across multiple threads.
2. A reduction occurs when private data on individual threads is assembled into a general solution.
3. Reduction is a technique wherein parallelism is applied to the portion of a program that requires the most computation.
4. Reduction is just another term for speedup.
5. None of these answers is correct.

Spring 2012 Early Adopter
Third NSF/TCPP Workshop on Parallel and Distributed Computing Education
http://cs.gsu.edu/~tcp/curriculum/1-qredupar