Visualizing Parallelism in CS 2

Sean Massung & Cinda Heeren

University of Illinois at Urbana-Champaign

May 20th 2013
IEEE-TCPP Curriculum Initiative for CS 2 at UIUC

Three main parallel programming concepts, each delivered during a two hour discussion/lab section

Material is graded, and key topics are evaluated on exams

Three lessons explore the basics of parallelism in a visual manner: using OpenMP, race conditions, and reductions

Tethering images to understanding parallelism enhances student interest and provides an alternative route to mastery of the material
Image manipulation

- 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.

```c
#pragma omp parallel for
for(int i = 0; i < width; ++i) {
    for(int j = 0; j < height; ++j) {
        *output(i,j) = *source(i,j);
        output(i,j)->green = 0;
    }
}
```
A visual partition of work

- Students are given augmented code that stops execution midway through the operation.
- They see four threads operating on the image removing the green component.
- The threads are partitioning the image along the width as shown in the code snippet.
Students learn that correctly parallelizing programs does not just consist of blindly pasting a `#pragma` on an outer for loop.

Image output from the provided code is shown to the right.
The buggy code

```c
RGBAPixel temp;
#pragma omp 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 - 1 - j);
        *image(i, height - 1 - j) = temp;
    }
}
```

Moving the declaration of temp inside the for loop fixes the problem, creating a local temp variable for each thread.
We want to create a color histogram

- 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.
Creating the histogram

```cpp
map<RGBAPixel, int> ret_freq;

#pragma omp parallel
{
    map<RGBAPixel, int> local_freq;
#pragma omp for
    for(int i = 0; i < width; ++i)
    {
        for(int j = 0; j < height; ++j)
            ++local_freq[*image(i,j)];
    }
#pragma omp critical
    {
        map<RGBAPixel, int>::iterator curr;
        for(curr = local_freq.begin(); curr != local_freq.end(); ++curr)
        {
            int count = curr->second;
            ret_freq[curr->first] += count;
        }
    }
}
return ret_freq;
```
Evaluating the students

- 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.
- The following slides contain examples of the questions we have included on various tests over the past two semesters.
Pop quiz!

Which of the code examples 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.
Reductions

What is the definition of the term \textit{reduction}?

(a) A reduction performs the same instructions on data across multiple threads.

(b) A reduction occurs when private data on individual threads is assembled into a general solution.

(c) Reduction is a technique wherein parallelism is applied to the portion of a program that requires the most computation.

(d) Reduction is just another term for \textit{speedup}.

(e) None of these is the correct choice.
Suppose an algorithm takes 7 seconds to run serially, and 2 seconds to run in parallel. Then the speedup for the parallelized code is:

(a) \( \frac{2}{7} \)

(b) \( \frac{7}{2} \)

(c) \( \frac{7-2}{7} \)

(d) The speedup cannot be determined because the number of processors is not known.

(e) None of these answers is correct.
Conclusions

- Foundations for including parallelism in introductory and second-level courses exist, but should be presented in a more approachable way.
- Applying parallelism to images gives a useful purpose to students’ work and immediate, visual feedback.
- Watching threads work on images increases student understanding and allows them to connect with the assignment on a tangible level.
Future Work

- Due to course reform, we will have more space added to the class: this time can be used to deepen the parallel concentration of the course in addition to examining existing topics in more depth.

- A fourth lab section is under development that deals with OpenMP tasks and their use in parallel sorting algorithms.