Homework 1

MacMillan, Kyle

September 28, 2018

Contents

Title

Table of Contents **i**

List of Figures **ii**

1 Problem 1 **1**

2 Problem 2 **1**

3 Problem 3 **3**

3.1 Problem 3a 3
3.2 Problem 3b 3
3.3 Problem 3c 4
3.4 Problem 3d 4
3.5 Problem 3e 4
3.6 Problem 3f 4
3.7 Problem 3g 4
3.8 Problem 3h 4

4 Problem 4 **4**

5 Problem 5 **4**

6 Graduate Assignment **4**

List of Figures

1 Example debug output. 3
2 Better performance without reduction. 3

1 Problem 1

What are the identity values for the operators: &&, ||, |, ^?

&& : 1
|| : 0
| : 0
^ : 0

2 Problem 2

Suppose OpenMP did not have the reduction clause. Show how to implement an efficient parallel reduction by adding a private variable and using the critical pragma.

```
/* File:    problem2.cpp
 * Purpose: Alternates sign of integer added to sum
 *
 *              sum = 0 + 1 + -2 + 3 + -4...
 *
 * Compile: g++ -Wall -fopenmp -o problem2 problem2.cpp -std=c++11
            g++ -Wall -fopenmp -o problem2 problem2.cpp -DDEBUG -std=c++11
 * Run:     ./problem2
 *
 * Input:   none
 * Output:  Times for each of the three runs
 *
 * Notes:
 *    1.    If ran with the -DDEBUG flag you can see what the sum should
            be based on n
 *
 */

#include <inttypes.h>   // Better integer functionality
#include <stdio.h>      // Printing to console
#include <omp.h>        // Multithreading
#include <chrono>       // High precision clock

using namespace std::chrono;

// Global
uint8_t     thrds    = omp_get_num_procs();


int main(int argc, char* argv[]) {
    uint8_t times = 20;
    high_resolution_clock::time_point t1 = high_resolution_clock::now();
    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    duration<double> no_omp_time = duration_cast<duration<double>>\
        (high_resolution_clock::now() - high_resolution_clock::now());
    duration<double> omp_time = duration_cast<duration<double>>\
        (high_resolution_clock::now() - high_resolution_clock::now());
    duration<double> no_reduc_time = duration_cast<duration<double>>\
        (high_resolution_clock::now() - high_resolution_clock::now());
    for(uint8_t j = 0; j < times; ++j)
```

```
    {
        uint64_t    n       = 80000000,
                    k       = 0;
        int64_t     sum     = 0;

        // RESET for baseline
        t1 = high_resolution_clock::now();

        for (k = 0; k < n; ++k)
        {
            sum += ((k & 1) == 0 ? 1.0 : -1.0) * k;
        }

        t2 = high_resolution_clock::now();
        no_omp_time += duration_cast<duration<double>>(t2 - t1);
#ifdef DEBUG
        if (j == 0){
            printf("No OMP sum   : %" PRIi64 "\n", sum);
        }
#endif

        // RESET for reduction + omp
        sum = 0;
        t1 = high_resolution_clock::now();

        #pragma omp parallel for num_threads(thrds) reduction(+: sum) private(k)
        for (k = 0; k < n; ++k)
        {
            sum += ((k & 1) == 0 ? 1.0 : -1.0) * k;
        }

        t2 = high_resolution_clock::now();
        omp_time += duration_cast<duration<double>>(t2 - t1);
#ifdef DEBUG
        if (j == 0){
            printf("OMP sum      : %" PRIi64 "\n", sum);
        }
#endif

        // RESET for no reduction
        sum = 0;
        k = 0;
        t1 = high_resolution_clock::now();

        #pragma omp parallel num_threads(thrds)
        {
            int64_t thread_sum = 0;
            #pragma omp for
            for(uint64_t i = k; i < n; ++i){
                // Locally (privately) runs this
                thread_sum += ((i & 1) == 0 ? 1.0 : -1.0) * i;
            }
```

```
            #pragma omp critical
            sum += thread_sum;
        }

        t2 = high_resolution_clock::now();
        no_reduc_time += duration_cast<duration<double>>(t2 - t1);
#ifdef DEBUG
        if (j == 0){
            printf("No Reduc sum : %" PRIi64 "\n", sum);
        }
#endif
    }
    printf("Averages over %" PRIu8 " runs:\n", times);
    printf("No OMP    : %.14f\n", no_omp_time.count() / times);
    printf("OMP       : %.14f\n", omp_time.count() / times);
    printf("No Reduc  : %.14f\n", no_reduc_time.count() / times);

    return 0;
}
```

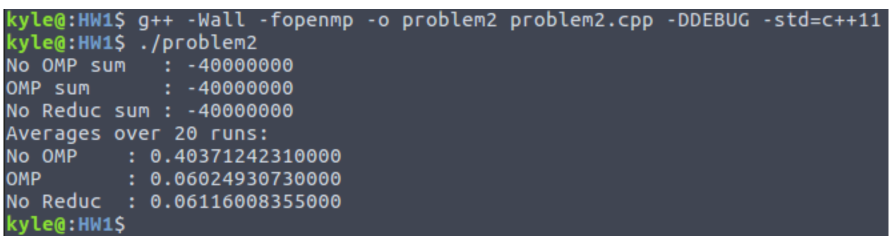

Figure 1: Example debug output.

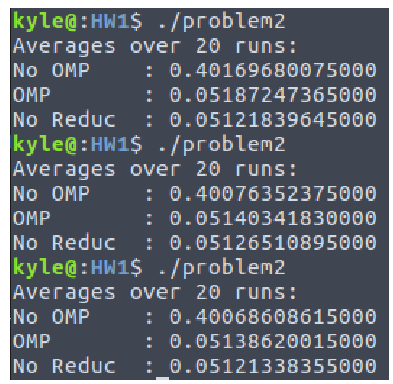

Figure 2: Better performance without reduction.

As can be seen in the figures the sums are performing as expected. An interesting, and expected outcome is that in Figure 1 it takes 0.06 seconds to run *OMP* and *No Reduc* but in Figure 2 it takes 0.05 seconds. The *No OMP* takes 0.40 seconds regardless. The reason for this behavior is that *OMP* uses the cores you give it and at the time of recording the first figure the browser was open and running a video. When I recorded the second Figure I had closed my browser to maximize performance for multi-core processing. The *No OMP* section of code was only running on one core, so it did not care that I had a video playing.

3 Problem 3

3.1 Problem 3a

3.2 Problem 3b

This code section is not suitable for OpenMP because of the && operator in comparison. Per the OpenMP documentation §2.6, p53:

test-expr	One of the following: *var relational-op b* *b relational-op var*
relational-op	One of the following: < <= > >=

3.3 Problem 3c

This code can be ran with OMP but it is dependent on whether or not $foo()$ is threadsafe.

3.4 Problem 3d

asdf

3.5 Problem 3e

asdf

3.6 Problem 3f

asdf

3.7 Problem 3g

asdf

3.8 Problem 3h

asdf

4 Problem 4

asdf

5 Problem 5

If the address of the nodes in a hypercube has n bits. How many nodes can it be at the most and how many edges does each node have? Give an algorithm that routes a message from node u to node v in this k-node hypercube in no more than log(k) steps.

Hypercubes are generalized by dimensionality. A hypercube address containing n-bits will exist in n dimensions (d). The maximum number of nodes is determined by d, and is defined as 2^d. A trait of hypercubes is that each node will have d edges. So the total amount of edges would be defined as $d * 2^{d-1}$.

6 Graduate Assignment

asdf