High-Performance Computing

Find Largest Prime Gap with Binary Tree

Anand Kamble
Department of Scientific Computing
Florida State University

---

1 Introduction

In this project, we are trying to parallelize the code for finding the largest prime gap between the prime numbers which are stored in a binary tree.

2 Implementation

In this program, we are using one thread which will serve as a master and will go through the tree creating the tasks.
It is implemented like this in the main function:

```
#pragma omp parallel
    {

// Letting one thread act as the master thread to create the tasks.
#pragma omp single nowait
        {
            gap = insertPrimes(root, n, m, 0, firstPrime);
        }
    }
```

2.1 Tasks

While going through the tree, this master thread creates the child tasks which the other threads will execute.
These tasks look like the following, ensuring that all the nested executions of the recursive function are done in parallel.

```
#pragma omp task
{
    R_a = insertPrimes(root->left, n, rightPrime - 1, level + 1, prev);
    R_b = insertPrimes(root->right, rightPrime + 1, m, level + 1,
       prev);
    right_gap = max(R_a, R_b);
}
```

Also, before creating the tasks, we are checking if we have enough threads to handle that many tasks. Since each node of the binary tree will result in two tasks, one for the left side and another for the right. And we know that the number of nodes doubles every level we go down in a tree. We are applying the following condition:

```
    if (2 * root->level > omp_get_max_threads())
    {
    //...Keep the execution serial.
    }
```

```
    else
    {
    //...spawn new tasks.
    }
```

3 Errors and Debugging

While programming this, I was not able to get the correct answers for the largest gap. Not waiting for the child processes to complete their execution was causing this. So adding taskwait before calculating the gap helped.

```
#pragma omp taskwait
    return max((rightPrime - leftPrime), max(left_gap, right_gap));
```

Also, I am not getting a speed-up as expected, the time required for the program stays relatively similar after 4 threads, and until 4 threads it does speed up.

To solve this, I tried using 2 master threads, i.e. the master thread created 2 more threads that were generating the tasks, one generating all the left side tasks and the other generating the right side. Doing this I saw a speed-up of factor 30, which did not look right. That's why I have commented that part in the code.

4 Results

This program was benchmarked with up to 6 cores and here are the plots for scalability and time.

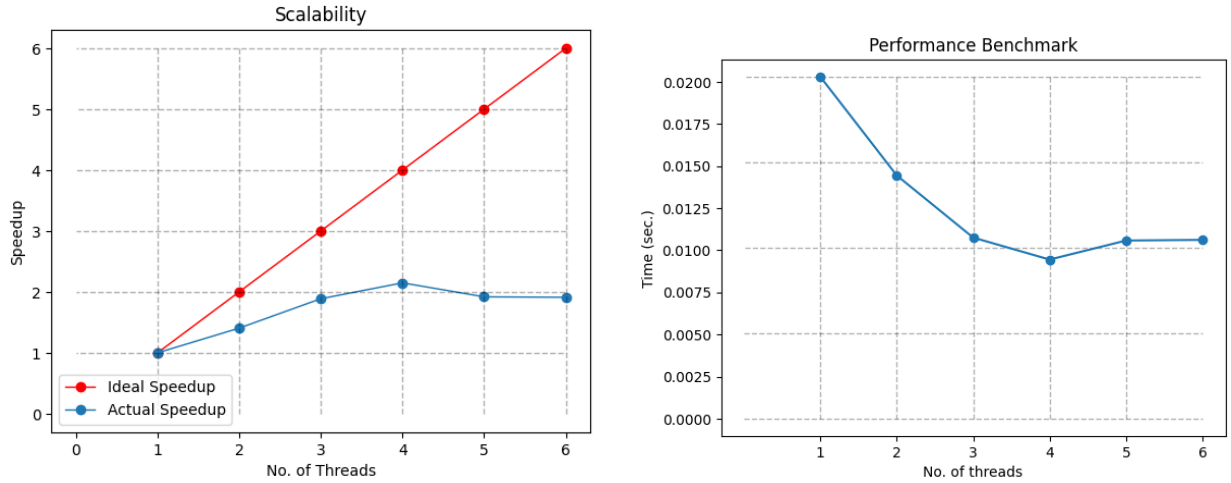

Figure 1: Timing and Scaling of the program

References

[1] NoseKnowsAll. *OpenMP - Binary tree* , *https://stackoverflow.com/a/34620077/22647897*. Jan 5, 2016.

[2] OpenAI. *Parallelized Binary Tree Reduction* , *https://chat.openai.com/share/b414c84e-5c52-4bbc-9538-dd1d5082a7fb*. Feb 20, 2024.