Parallel/Distributed Computing Assignment #1

최준수 20151619

1.

![A close up of text on a white background

Description automatically generated]()

As the Hint in the question states, if we can find a corresponding rate of increase in the problem size as increase in the number of processors so that the program always has the same efficiency, then problem is (weakly) scalable.

However, number of processors has increase by 2times whereas the problem size needs to be increased by 2.666. Because they don’t correspond to each other, this problem is not weakly scalable.

2.

(1)

첫번째 방법:

For i=0 to ceil(log2n)-1 do

For j=0 to n-1 do

If j < 2i then

xji+1 = xij

else

xji+1 = xji + xij-2i

설명: 첫번째 outer loop을 돌며 각각 자신의 1칸 왼쪽 값을 자신과 더해서 저장한다. 다음 번에는 자신의 2칸 왼쪽 값을 자신과 더해서 저장한다. 이를 log2n만큼 반복하는데 outer loop가 끝날때마다 synchronize해주고 안쪽 loop는 process의 수에 맞게 나눠서 수행하면 병렬적으로 수행할 수 있다

두번째 방법:

First pass:

for(int stride = 1; stride <= n; stride \*= 2)

// thread별로 for loop을 다시 돌림

int index = (thread id + 1) \* stride \* 2 – 1;

if (index < 2\*n)

x[index] += x[index – stride]

stride \*= 2;

Second pass:

for(int stride=n/2; stride >0; stride /=2)

// thread 별로 for loop을 다시 돌림

int index = (thread id +1)\* stride \* 2 -1;

if (index + stride < 2\*n)

x[index + stride] = x[index];

stride /= 2;

짝수번째 index의 값과 홀수번째 index의 값을 구하는 방법을 나눠서 두번 수행한다.

우선, 짝수번째 index의 값들을 구하기 위해 stride=1부터 시작해서 stride의 거리에 있는 값과 더한 후 저장한다. 매 loop마다 2배씩 늘려가며 반복한다. 이 작업이 끝나면 짝수번째 index들은 최종결과 값들을 가지게 되고, 두번째 코드는 stride를 반으로 줄여가며 홀수번째 index를 완성한다.

<http://www.cs.ucr.edu/~nael/217-f15/lectures/217-lec12.pdf>

(2)

3 Nodes

|  |  |  |  |
| --- | --- | --- | --- |
|  | N=10 | N=500 | N=1000 |
| Blocking | 0.000144 | 0.000612 | 0.000984 |
| Non-Blocking | 0.483651 | 0.687662 | 0.007588 |
| MPI\_Scan | 0.000002 | 0.000004 | 0.000003 |

9 Nodes

|  |  |  |  |
| --- | --- | --- | --- |
|  | N=10 | N=500 | N=1000 |
| Blocking | 0.005721 | 0.314907 | 0.029858 |
| Non-Blocking | 0.367673 | 1.251788 | 0.022969 |
| MPI\_Scan | 0.000003 | 0.000005 | 0.000004 |

In theory, executing with more nodes should result in shorter execution time. However, as you can see from the result in the table above, execution with 9 nodes took longer time compared to execution with 3 nodes. It is likely that Tparallel overhead was bigger than actual calculation time because root process needs to communicate with 8 other processes every loop to synchronize. If N was a much bigger number such as 100,000,000, actual execution time would be longer than Tparallel overhead so using more nodes will be beneficial.

(3)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Small/sign\_1.ppm | Small/boxes\_1.ppm | Large/sage\_1.ppm | Large/start\_1.ppm |
| Sequential | 0.00377702 | 0.0033590 | 0.082972 | 0.083533 |
| Parallel | 0.0018301 | 0.0012901 | 0.065376 | 0.072869 |

Parallel showed better performance in both small and large images. Speed-up for small images is around 3. However, for large images, speed-up is around 1. Thus my program isn’t scalable.

I used Macbook Pro to view .ppm files as it supports .ppm as default. But if you are on Linux, based on internet search, you can use “display” and “ImageMagick”