LP1

High Performance Computing

Assignment 3

Date of Completion: - 4.9.2020

Title :- Parallel Sorting Algorithms

Problem Statement :- For Bubble sort and Merge sort, based on existing sequential algorithms, design & implement parallel algorithm utilizing all resources available.

Objectives :- Study parallel execution of sorting algorithm
Study open MP for parallel computing.

Outcomes :- students will be able to
- Implement sorting algorithms in open MP.

Software / Hardware Requirement :- Ubuntu OS, Open MP API, editor.

Theory

Parallel Sorting :-

a) Parallel Bubble Sort

i) Implement as a pipeline
ii) Let localsize = n/no of processor
iii) We divide the array into blocks and each process

executes the bubble sort on the past including comparing the last element with the first one belonging to the next thread

iv) Implement with loop for j=0 to n-1.

v) For each iteration of j each thread needs to wait until previous thread has finished that iterations

vi) Synchronization mode to be used is 'barrier'.

b) Parallel Merge Sort.

i) These steps are preformed
   1) Divide
   2) Conquer
   3) Combine

ii) Locket sort list onto one processor

iii) Merge implements as they come together

iv) Simple tree structure is obtained.

Algorithm

Bubble Sort

```
for k = 0 to n-2
    if k is even then
        for i = 0 to n/2 - 1 do in parallel
            if A[2i] > A[2i+1] then
                swap A[2i] & A[2i+1]
        end for
    else for i = 0 to n/2 - 2 do in parallel
        if A[2i+1] > A[2i+2] then
            swap A[2i+1] & A[2i+2]
        end for
    end if
end for
```

Parallel Merge Sort

1. mid > size/2
   if both children present in tree then
   send mid, firstchild
   send size-mid secondchild
   send list, mid, firstchild
   send list from mid, size mid, secondchild
   call merge (list, 0, mid, list, mid+1, size, temp, 0, size)
   Store temp in another array List
   else
   call parallelMergeSort (List, 0, size)
   if i>0 then
   send list, size, parent

Analysis

	Time Complexity	Bubble Sort	Merge Sort
Sequential	Best	O(n)	O(nlogn)
	Averge	O(n²)	O(nlogn)
Parallel	Best	O(n)	O(n)
	Averge	O(nlogn)	O(nlogn)

Test Case:-

for n = 1000	Bubble Sort	Merge Sort
Time for parallel execution:-	0.0039	0.00017
Time for serial execution:-	0.0021	1.8

for n = 100		
Parallel execution time:-	0.0009 3	0.00018
Serial execution time:-	0.00003	1.6

Conclusion:- Thus I completed the sorting algorithms using parallel reduction and understood the algorithms.

Assignment 3

1) Merge Sort

CODE:

```
code = """
#include<iostream>
#include<omp.h>

using namespace std;

void printArray(int *arr, int size) {
for(int i=0; i<size; i++) {
cout<<arr[i]<<" ";
}
cout<<endl;
}

void merge(int* arr, int start, int mid, int end) {
int len = (end - start) + 1;
int temp[len];
int cur = 0;
int i = start;
int j = mid + 1;
while(i <= mid && j <= end){
if(arr[i] < arr[j]) {
temp[cur] = arr[i];
cur++;
i++;
}
else {
temp[cur] = arr[j];
cur++;
j++;
}
}
if(i <= mid) {
while(i <= mid) {
temp[cur] = arr[i];
i++;
cur++;
}
}
else if(j <= end) {
while(j <= end) {
temp[cur] = arr[j];
j++;
cur++;
}
}
cur = 0;
for(i=start; i<=end; i++) {
arr[i] = temp[cur];
cur++;
}

}

void mergeSort(int *arr, int start, int end) {
if(start < end) {
int mid = (start+end) / 2;
```

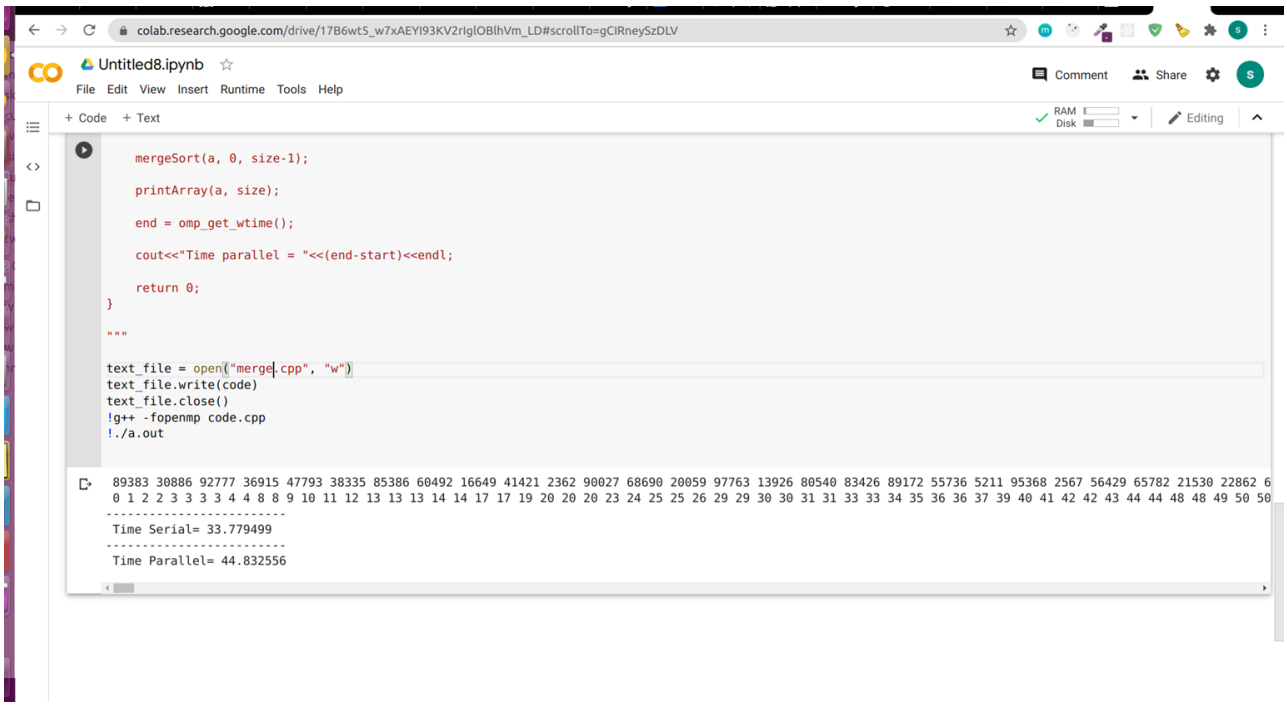
```
#pragma omp parallel sections
{
#pragma omp section
mergeSort(arr, start, mid);
#pragma omp section
mergeSort(arr, mid+1, end);
}
merge(arr, start, mid, end);
}
}

int main(int argc, char *argv[]) {
int size = 100;
int a[size];
double start, end;
omp_set_num_threads(2);
for(int i=0; i<size; i++) {
a[i] = rand()% 100;
}
//int a[]= {7,33,5,5,23,111,75,34,77,121,120};
for(int i=0; i<size; i++)
cout<<" "<<a[i];
cout<<endl;
start = omp_get_wtime();
mergeSort(a, 0, size-1);
printArray(a, size);
end = omp_get_wtime();
cout<<"Time parallel = "<<(end-start)<<endl;
return 0;
}

"""

text_file = open("code.cpp", "w")
text_file.write(code)
text_file.close()
!g++ -fopenmp code.cpp
!./a.out
```

OUTPUT:


colab.research.google.com/drive/17B6wtS_w7xAEYI93KV2rIglOBlhVm_LD#scrollTo=gCIRneySzDLV
Untitled8.ipynb
File Edit View Insert Runtime Tools Help
Comment
Share
+ Code + Text
RAM
Disk
Editing
mergeSort(a, 0, size-1);
printArray(a, size);
end = omp_get_wtime();
cout<<"Time parallel = "<<(end-start)<<endl;
return 0;
}
"""
text_file = open("merge.cpp", "w")
text_file.write(code)
text_file.close()
!g++ -fopenmp code.cpp
!./a.out
89383 30886 92777 36915 47793 38335 85386 60492 16649 41421 2362 90027 68690 20059 97763 13926 80540 83426 89172 55736 5211 95368 2567 56429 65782 21530 22862 6
0 1 2 2 3 3 3 3 4 4 8 8 9 10 11 12 13 13 13 14 14 17 17 19 20 20 20 23 24 25 25 26 29 29 30 30 31 31 33 33 34 35 36 36 37 39 40 41 42 42 43 44 44 48 48 49 50 50
Time Serial= 33.779499
Time Parallel= 44.832556

2) Bubble sort

CODE:

```
code = """#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
void swap(int *,int *);
int main (int argc, char *argv[]) {
int SIZE =1000;
int A[SIZE],B[SIZE];
for(int i=0;i<SIZE;i++)
{
A[i]=rand()%SIZE;
B[i]=A[i];
printf(" %d",A[i]);
}
printf("\\n");
//int A[5] = {6,9,1,3,7};
int N = SIZE;
int i=0, j=0;
int first;
double start,end;
start = omp_get_wtime();
for (int i = 0; i < N-1; i++) {
for (int j = 0; j < N-i-1; j++) {
if (B[j] > B[j+1])
{
// swap arr[j+1] and arr[i]
int temp = B[j];
B[j] = B[j+1];
B[j+1] = temp;
}
}
}
end = omp_get_wtime();
double temptime = end -start;
start=omp_get_wtime();
for( i = 0; i < N-1; i++ )
{
first = i % 2;
#pragma omp parallel for default(none),shared(A,first,N)
for( j = first; j < N-1; j += 1 )
{
if( A[ j ] > A[ j+1 ] )
{
swap( &A[ j ], &A[ j+1 ] );
}
}
}
end=omp_get_wtime();
for(i=0;i<N;i++)
{
printf(" %d",B[i]);
}
printf("\\n-------------------------\\n Time Serial= %f",(temptime));
printf("\\n-------------------------\\n Time Parallel= %f",(end-start));
}
void swap(int *num1, int *num2)
{
int temp = *num1;
*num1 = *num2;
*num2 = temp;}"""
```

```
text_file = open("code.cpp", "w")
text_file.write(code)
text_file.close()
!g++ -fopenmp code.cpp
!./a.out
```

OUTPUT: