Camlin Page

Assignment 3.

- Title: Parellel Sorting Algorithms.

- Problem Statement:
  Parellel sorting algorithms for bubble sort & merge sort based on existing sequential algorithms & design & implement utilizing all resources

- Objective:
  - To study parellel execution of sorting algo
  - To study OpenMP for parellel computing

- Outcomes:-
  We will be able to
  - understand parellel sorting algorithms
  - implement algorithms using openMP.

- Requirements:-
  - OS: Fedora 20 / Ubuntu (64bit)
  - openMP API
  - Editor: gedit
  - G++ compiler
  - RAM: 4GB
  - HDD - 500GB

- Theory:

→ Parellel sorting:

(A.) Parellel Bubble Sort:-
- Implement as a pipeline
- let local size = n / no_of_processors
we divide array into blocks, and each processors executes the bubble sort on this part including comparing the last element with the first one belonging to next thread
- Implemented with the loop
for (j=0 ; j< n-1 ; j++)
- For every iteration of j, each thread needs to wait until previous threads has finished that iteration
- Synchronization mode to be used as 'barrier'.

(B.) Parellel Merge Sort:-
- Three steps are performed
(1.) Divide (2.) Conquer (3.) Combine
- Collect sorted list in one processor
- Merge elements as they come together
- Simple tree structure is obtained

Camlin Page

Algorithms:

(A) Parellel Bubble sort :-

(1.) for k=0 to n-2
(1.1) if k is even then
(1.1.1) for i=0 to n/2 -1 do in 11el
(1.1.1.1) if A[2i] > A[2i+1] then
(1.1.1.1.1) swap
(1.1.2) end for

(1.2.) else
(1.2.1) for i=0 to n/2-2 do in 11el
(1.2.1.1) if A[2i+1] > A[2i+2] then
(1.2.1.1.1) swap
(1.2.2) end for
(1.3) end if
(2.) end for

(B) Parellel Merge Sort :-

(1) mid = size/2
(2) if both children present in tree then
(2.1) send mid, firstchild
(2.2) send size-mid, secondchild
(2.3) send list, mid, firstchild
(2.4) send list from mid, size-mid, secondchi
(2.5) call merge (list, 0, mid, list, mid+1,
size, temp, 0, size)

(2.6) store temp in another arraylist
(3) else
(3.1) call parallel Merge Sort (list, 0, size)
(4) end if
(5) if i>0 then
(5.1) send list, size, parent
(6) end if

- Analysis:

Time complexity of both Algorithms.

Type	Case	Bubble sort	Merge sort
Sequential	Best	O(n)	O(nlogn)
	Worst	O(n²)	O(nlogn)
	Average	O(n²)	O(nlogn)
Parallel	Best	O(n)	O(n)
	Worst	O(nlogn)	O(nlogn)
	Average	O(nlogn)	O(nlogn)

- Test cases & Analysis

Sorting	I/p size	Sequential	Parallel	Efficiency
Bubble	n=256	0.020	0.05	0.4
	n=1024	0.070	0.011	6.36
Merge	n=1024	0.003	0.03	0.1
	n=2048	0.002	0.02	0.1

Camlin Page

$$\text{Efficiency} = \frac{WCSA}{WCPA}$$

We observe that for bubble sort there was improvement in performance of algorithm but for merge sort, sequential algorithm proves better.

Input:-

Input array → 5, 9, 5, 2, 4, 11, 5, 1, 5, 0

Output:-

Sorted array → 0, 1, 2, 4, 5, 5, 5, 5, 9, 11

- Conclusion.

Thus, we successfully implemented parellel bubble sort & merge sort using openMP.