Bubble Sort

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

void swap();

int main (int argc, char \*argv[]) {

int SIZE =1<<8;

int A[SIZE];

for(int i=0;i<SIZE;i++)

{

A[i]=rand()%SIZE;

}

//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( 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",A[i]);

}

printf("\n-------------------------\n Time Parallel= %f",(end-start));

}

void swap(int \*num1, int \*num2)

{

int temp = \*num1;

\*num1 = \*num2;

\*num2 = temp;

}

Merge Sort

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <omp.h>

#define MAX\_SIZE 10

//Function for creating an input array||Update accoorind to your need

void generate\_list(int \* x, int n) {

int i,j,t;

for (i = 0; i < n; i++)

x[i] = i;

for (i = 0; i < n; i++) {

j = rand() % n;

t = x[i];

x[i] = x[j];

x[j] = t;

}

}

void print\_list(int \* x, int n) {

int i;

for (i = 0; i < n; i++) {

printf("%d ",x[i]);

}

}

//Merging 2 sorted subarrays into one tmp array

void merge(int \* X, int n, int \* tmp) {

int i = 0;

int j = n/2;

int ti = 0;

//i will iterate till first half anf J will iterate for 2nd half of array

while (i<n/2 && j<n) {

if (X[i] < X[j]) {

tmp[ti] = X[i];

ti++; i++;

} else {

tmp[ti] = X[j];

ti++;

j++;

}

}

while (i<n/2) { /\* finish up lower half \*/

tmp[ti] = X[i];

ti++;

i++;

}

while (j<n) { /\* finish up upper half \*/

tmp[ti] = X[j];

ti++;

j++;

}

//Copy sorted array tmp back to X (Original array)

memcpy(X, tmp, n\*sizeof(int));

} // end of merge()

void mergesort(int \* X, int n, int \* tmp)

{

if (n < 2) return;

#pragma omp task firstprivate (X, n, tmp)

mergesort(X, n/2, tmp);

#pragma omp task firstprivate (X, n, tmp)

mergesort(X+(n/2), n-(n/2), tmp);

//Wait for both paralel tasks to complete execution

#pragma omp taskwait

/\* merge sorted halves into sorted list \*/

merge(X, n, tmp);

}

int main()

{

int n = 10;

double start, stop;

int data[MAX\_SIZE], tmp[MAX\_SIZE];

generate\_list(data, n);

printf("List Before Sorting...\n");

print\_list(data, n);

start = omp\_get\_wtime();

#pragma omp parallel

{

#pragma omp single

mergesort(data, n, tmp);

}

stop = omp\_get\_wtime();

printf("\nList After Sorting...\n");

print\_list(data, n);

printf("\nTime: %g\n",stop-start);

}

OUTPUT:

samarjeet@samarjeet-VirtualBox:~/codes/OpenMP$ gcc -fopenmp -o p\_Bubble p\_Bubble.c

samarjeet@samarjeet-VirtualBox:~/codes/OpenMP$ ./p\_Bubble

0 1 2 4 5 5 5 5 9 11 11 13 14 15 16 17 17 19 20 21 21 23 24 25 26 26 26 27 27 27 28 28 30 31 33 33 34 35 35 36 37 41 41 41 42 43 46 49 50 50 50 50 51 51 52 54 56 56 58 58 59 59 59 59 60 60 61 61 62 62 65 65 67 70 71 71 72 72 73 74 75 77 78 78 78 79 80 81 84 84 84 85 88 88 90 90 90 92 92 92 92 92 93 94 94 95 97 99 99 100 100 100 102 103 103 105 107 108 111 112 112 112 115 115 115 116 117 118 119 121 121 123 124 124 125 126 127 130 130 134 135 137 141 143 148 148 149 150 151 152 153 154 155 155 158 158 159 161 161 162 163 164 168 168 169 170 170 170 171 171 172 172 175 175 175 176 177 178 179 179 180 181 181 183 186 186 187 187 188 189 190 190 194 197 197 198 198 201 202 202 203 203 205 205 205 208 209 211 212 212 216 219 220 220 220 224 225 225 226 227 227 228 229 230 231 232 232 233 233 233 234 235 236 236 236 239 240 241 242 242 245 247 248 248 248 249 250 250 251 251 251 251 252 253 254 255

-------------------------

Time Parallel= 0.000768

samarjeet@samarjeet-VirtualBox:~/codes/OpenMP$ gcc -fopenmp -o p\_Merge p\_Merge.c

samarjeet@samarjeet-VirtualBox:~/codes/OpenMP$ ./p\_Merge

List Before Sorting...

3 8 2 4 5 0 1 7 9 6

List After Sorting...

0 1 2 3 4 5 6 7 8 9

Time: 1.1549e-05