typescript
Script started on Wed, Apr 19, 2017 5:25:50 PM
←]0;/cygdrive/c/Users/essy/desktop/classes/Parallel Processing/Prog4•

←[32messy@essy-HP ←[33m/cygdrive/c/Users/essy/desktop/classes/Parallel Processing/Prog4←[0m

$ cat mergesort.cpp
//// Kenneth Willeford

//// Prog4 - Parallel MergeSort

////

//// This program runs a parallel merge sort as defined

//// in Prog4.pdf based on the options provided.

////

//// compile and run:

//// $> g++ -fopenmp -o xxx mergesort.cpp

//// $>./xxx 100 4 trace

////   //4 is the number of threads to use - any

////   //100 is the size of the array to be sorted - any

////   //trace is to show the output with tracing - optional

#include<iostream>

#include<cstdlib>

#include<ctime>

#include<omp.h>

using namespace std;

long* list;   // The Global Array that will be sorted.

long N;   // The Length of the Global Array, first command line argument.

long p;   // The number of threads to create, second command line argument.

bool traceOutput;   // Whether or not to trace merge sort output

// Derived from gnu.org example

// For additional information see: http://www.gnu.org/software/libc/manual/html_node/Comparison-Functions.html#Comparison-Functions

int CompareLongs (const void *a, const void *b){

```
  const long *da = (const long *) a;
  const long *db = (const long *) b;
  return (*da > *db) - (*da < *db);
}


// Based on rank and core_difference merges two sorted sub-arrays into a combined sorted array.
void Merge(long r,long d){
        long lbegin = r*N/p;                    // Get Start of Left Array
        long lendrbegin = (r+d)*N/p;    // Get Start of Right Array / End of Left Array
        long rend = (r+2*d)*N/p;                // Get End of Right Array

        // Copy the subarrays into their own arrays for processing.
        long lcopyi = lbegin;
        long rcopyi = lendrbegin;
        long* leftArray  = new long[d*N/p];
        long* rightArray = new long[d*N/p];
        #pragma parallel for
        for(int i = 0; i < d*N/p; i++){
                leftArray[i]  = list[lcopyi++];
                rightArray[i] = list[rcopyi++];
        }

        // Merge the created arrays into the main array by <.
        lcopyi = 0;
        rcopyi = 0;
        for(int i = lbegin; i < rend; i++){
                // If there is nothing left in the leftArray
                if(lcopyi >= d*N/p){
                        // Copy contents from the rightArray
                        list[i] = rightArray[rcopyi++];
                // If there is nothing left in the rightArray
```

```
                } else if (rcopyi >= d*N/p){
                        // Copy contents from the leftArray
                        list[i] = leftArray[lcopyi++];
                // If the leftArray element is less than the rightArray
                } else if(leftArray[lcopyi] < rightArray[rcopyi]){
                        // Copy contents from the leftArray.
                        list[i] = leftArray[lcopyi++];
                }else{
                        // If the right array element is equivalent or less then
copy the contents from the rightArray.
                        list[i] = rightArray[rcopyi++];
                }
        }
}


// Performs the merge sort in parallel
void MergeSort(){
        // Special Case, only 1 thread being made. In this case just run qsort on
list and exit.
        if(p == 1){ qsort(list,N,sizeof(long),CompareLongs); return;}


        long MyRank = omp_get_thread_num();             // Get Thread ID
        long* LocalList = new long[N/p];            // Create Evenly Partitioned
Local List
        long Start = N/p * MyRank;                              // Get the start
position of the list.


        // Copy Relevant Global Array Contents to Local Array
        #pragma omp parallel for
        for(long i = 0; i < N/p; i++)
                LocalList[i] = list[Start++];


        // Print local list information if tracing.
```

```
	if(traceOutput){
		#pragma omp critical
		{
			cout << "Thread_" << MyRank << ", local_list: ";
			for(long i = 0; i < N/p; i ++)
				cout << LocalList[i] << " ";
			cout << endl;
		}
		// Used to syncrhonize so that multiple threads wont have access to
cout. (Should probably have made it its own function with critical section.)
		#pragma omp barrier
	}
	// Sort Local Array.
	qsort(LocalList,N/p,sizeof(long),CompareLongs);

	// Print sorted local list information if tracing.
	if(traceOutput){
		#pragma omp critical
		{
			cout << "Thread_" << MyRank << ", sorted local_list: ";
			for(long i = 0; i < N/p; i ++)
				cout << LocalList[i] << " ";
			cout << endl;
		}
	}

	// Copy Local Array Contents to Global Array
	Start = N/p * MyRank;
	#pragma omp parallel for
	for(long i = 0; i < N/p; i++)
		list[Start++] = LocalList[i];
```

```
        // Determine who is sender or reciever and terminate thread / merge where
necessary.
        long divisor = 2;
        long CoreDifference = 1;
        while(divisor <= p){
                #pragma omp barrier
                if(MyRank % divisor == 0) Merge(MyRank,CoreDifference);
                CoreDifference *= 2;
                divisor *= 2;
        }
}

// Retrieves the command line arguments
void GetCommandLineArguments(int argc ,char* argv[]){
        N = atoi(argv[1]);      // get size of array
        p = atoi(argv[2]);      // get number of threads to make
        // determine if a trace should occur
        if(argc > 3) traceOutput = string(argv[3]) == "trace";
}

// Creates a random array based on N.
void MakeRandomArray(){
        list = new long[N];
        srand(time(NULL));
        // rand() isn't thread safe so parallel for causes unintended behavior, on
my machine elements will cycle like so [7,1,7,1,7,1,7...]
        for(long i = 0; i < N; i++){
                list[i] = rand() % N;
        }
}

// Executes the MergeSort then returns the execution time.
double BenchmarkMergeSort(){
```

```
        double time_elapsed = omp_get_wtime();          // Start Time
        #pragma omp parallel num_threads(p)
        MergeSort();                                                    //
Perform the merge sort.
        // If tracing output then print the sorted list.
        if(traceOutput){
                cout << "sorted list:" << endl;
                for(int i = 0; i < N; i++){
                        cout << list[i] << " ";
                }
                cout << endl;
        }
        return omp_get_wtime() - time_elapsed;          // Final Time
}

// Function to verify if the list is sorted.
bool ListIsSorted(){
        for(long i = 1; i < N; i++)
                if(list[i] < list[i-1]) return false;
        return true;
}

// Prints the final output. Requires the execution time.
void PrintResult(double t){
        if(ListIsSorted()){
                cout << "**verified that list is sorted." << endl;
        } else {
                cout << "**list is NOT sorted." << endl;
        }
        cout << "Using P=" << p << ", n=" << N << ", Time taken: " << t << " sec" <<
endl;
}
```

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

        GetCommandLineArguments(argc,argv);

        MakeRandomArray();

        PrintResult(BenchmarkMergeSort());

}←]0;/cygdrive/c/Users/essy/desktop/classes/Parallel Processing/Prog4•

←[32messy@essy-HP ←[33m/cygdrive/c/Users/essy/desktop/classes/Parallel
Processing/Prog4←[0m

$ g++ -fopenmp -o xxx mergesort.cpp
←]0;/cygdrive/c/Users/essy/desktop/classes/Parallel Processing/Prog4•

←[32messy@essy-HP ←[33m/cygdrive/c/Users/essy/desktop/classes/Parallel
Processing/Prog4←[0m

$ ./xxx 40 4 trace
Thread_0, local_list: 38 29 15 24 18 16 37 20 28 32

Thread_1, local_list: 26 31 26 6 26 18 20 15 39 12

Thread_2, local_list: 21 38 0 6 10 17 6 31 25 8

Thread_3, local_list: 33 22 20 35 3 30 30 35 35 15

Thread_0, sorted local_list: 15 16 18 20 24 28 29 32 37 38

Thread_1, sorted local_list: 6 12 15 18 20 26 26 26 31 39

Thread_2, sorted local_list: 0 6 6 8 10 17 21 25 31 38

Thread_3, sorted local_list: 3 15 20 22 30 30 33 35 35 35

sorted list:

0 3 6 6 6 8 10 12 15 15 15 16 17 18 18 20 20 20 21 22 24 25 26 26 26 28 29 30 30 31
31 32 33 35 35 35 37 38 38 39

**verified that list is sorted.

Using P=4, n=40, Time taken: 0.00300002 sec

←]0;/cygdrive/c/Users/essy/desktop/classes/Parallel Processing/Prog4•

←[32messy@essy-HP ←[33m/cygdrive/c/Users/essy/desktop/classes/Parallel
Processing/Prog4←[0m

$ ./xxx 40 4 trace□←[K□←[K□←[K□←[K□←[K□←[K□←[K□←[K□←[K□←[K□←[K□←[K□←[K□←[K□←[K□←[K•••••echo
""□I"□'"□m"□ "□u"□s"□i"□n"□g"□ "□1"□/"□1"□0"□□←[1P"□□←[1P"□□←[1P"□□←[1P"□9"□□←[1P"□□←[1P"□
"□1"□□←[1P"□a"□ "□h"□u"□n"□d"□r"□e"□d"□ "□m"□i"□l"□l"□i"□o"□n"□
"□v"□e"□r"□s"□□←[1P"□□←[1P"□□←[1P"□□←[1P"□i"□n"□s"□t"□e"□a"□d"□ "□o"□f"□ "□a"□
"□b"□i"□l"□l"□i"□o"□n"□ "□e"□l"□e"□m"□e"□n"□t"□s"□ "□b"□e"□c"□a"□u"□s"□e"□ "□t"□h"□e"□ "□l"
←
[A←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C
←
[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C
←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[Ca"
"
```

t"□t"□e"□r"□ "□t"□h"□r"□o"□w"□s"□ "□a"□ "□b"□a"□d"□_"□a"□l"□l"□o"□c"□ "□o"□n"□ "□m"□y"□
"□m"□a"□c"□h"□i"□n"□e"□←[C
I'm using a hundred million instead of a billion elements because the latter throws
a bad_alloc on my machine
←]0;/cygdrive/c/Users/essy/desktop/classes/Parallel Processing/Prog4•

←[32messy@essy-HP ←[33m/cygdrive/c/Users/essy/desktop/classes/Parallel
Processing/Prog4←[0m

$ echo "I'm using a hundred million instead of a billion elements because the lat
tter throws a bad_alloc on my machine"←[A
←[C←[C./xxx 40 4 trace←[K

←[K←[A←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C←[C□□□□□□□←[1P trace□□□□□□1
trace□□□□□□←[C←[C←[C←[C←[C←[C□←[K□←[K□←[K□←[K□←[K□←[K□←[K□□□←[1P 1□□□←[1P 1□□1 1□□0 1□□0 1□□0
1□□□←[1P 1□□0 1□□0 1□□0 1□□0 1□□0 1□□0 1□□□←[1P 1□□□←[1P 1□□□←[1P 1□□□←[1P 1□□□←[1P 1□□□←[1P
1□□0 1□□0 1□□0 1□□0 1□□0 1□□0 1□□
**verified that list is sorted.

Using P=1, n=100000000, Time taken: 40.488 sec

←]0;/cygdrive/c/Users/essy/desktop/classes/Parallel Processing/Prog4•

←[32messy@essy-HP ←[33m/cygdrive/c/Users/essy/desktop/classes/Parallel
Processing/Prog4←[0m

$ ./xxx 100000000 1□←[K2
**verified that list is sorted.

Using P=2, n=100000000, Time taken: 13.465 sec

←]0;/cygdrive/c/Users/essy/desktop/classes/Parallel Processing/Prog4•

←[32messy@essy-HP ←[33m/cygdrive/c/Users/essy/desktop/classes/Parallel
Processing/Prog4←[0m

$ ./xxx 100000000 2□←[K4
**verified that list is sorted.

Using P=4, n=100000000, Time taken: 10.925 sec

←]0;/cygdrive/c/Users/essy/desktop/classes/Parallel Processing/Prog4•

←[32messy@essy-HP ←[33m/cygdrive/c/Users/essy/desktop/classes/Parallel
Processing/Prog4←[0m

$ ./xxx 100000000 4□←[K8
**verified that list is sorted.

Using P=8, n=100000000, Time taken: 12.257 sec

←]0;/cygdrive/c/Users/essy/desktop/classes/Parallel Processing/Prog4•

←[32messy@essy-HP ←[33m/cygdrive/c/Users/essy/desktop/classes/Parallel
Processing/Prog4←[0m

$ exit
exit

Script done on Wed, Apr 19, 2017 5:29:32 PM