Parallel and Distributed Systems

Bitonic Sorter

Marios Mitalidis
Aristotle University of Thessaloniki
mariosam@ece.auth.gr

5/11/2016

Abstract

This project is about a parallel implementation of the Bitonic Sort algorithm. The algorithm was implemented using pthreads, OpenMP and CilkPlus in C language and the results was compared against *qsort* function from *stdlib.h* library and a serial implementation of bitonic sort.

Algorithm

The program takes as input the parameters p and q. It then creates 2^p threads, and an array of random elements, with length 2^q. Finally, it sorts the elements and checks the correctness of the result.

The pseudo-code of the algorithm, to implement the actual sorting, is presented in the following box.

```
1.      Make some initial calculations.
2.      If the current number of threads < max number of threads
3.              Create a new thread.
4.              Assign to new thread: rec_bitonic_sort( left )
5.      else
6.              Execute rec_bitonic_sort( left )
7.      end_if
8.
9.      Execute rec_bitonic_sort( right )
10.     Execute bitonic_merge
```

In the critical section of the code, that is access to shared memory, a mutex variable was used.

Furthermore, an optimized version of the algorithm was implemented by combining bitonic sort with qsort. More specifically, for the sorting of sub-arrays with length less than 2^{21}, qsort was used. This resulted in an important decrease in execution time.

Finally, in addition to pthreads, the equivalent OpenMP and Cilk Plus implementations were written and we compared the results.

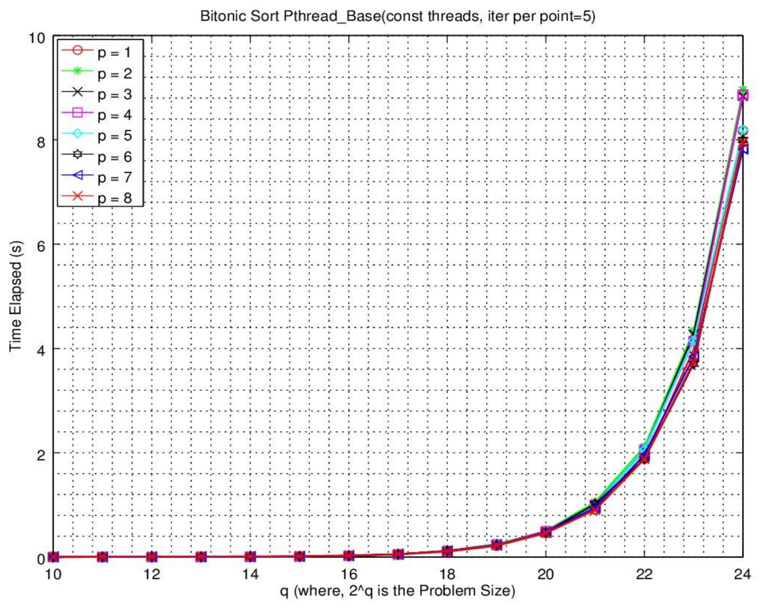
Correctness check

Functional correctness was confirmed, after comparing the output of the algorithm with the output of *qsort* function from *stdlib.h*.

Results

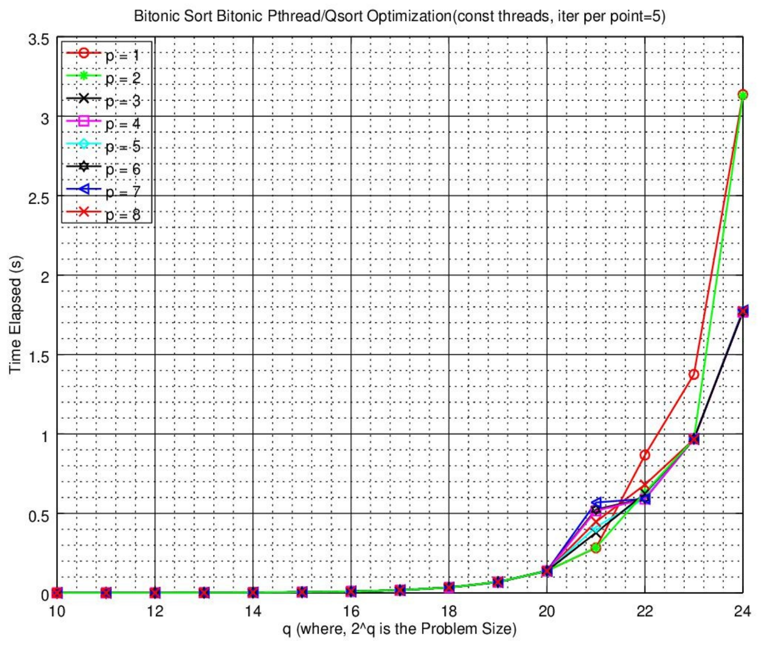
The experiments were carried out on diades system (Intel Xeon CPU @2.5GHz, 8 cores) of AUTh. Each experiment was executed 5 times and we calculated the average executation time.

First, we present the diagram of time/probem size for curves of constant p, and for the case of pthreads.



Drawing 1: Bitonic Sort - Pthreads Recursive (basic)

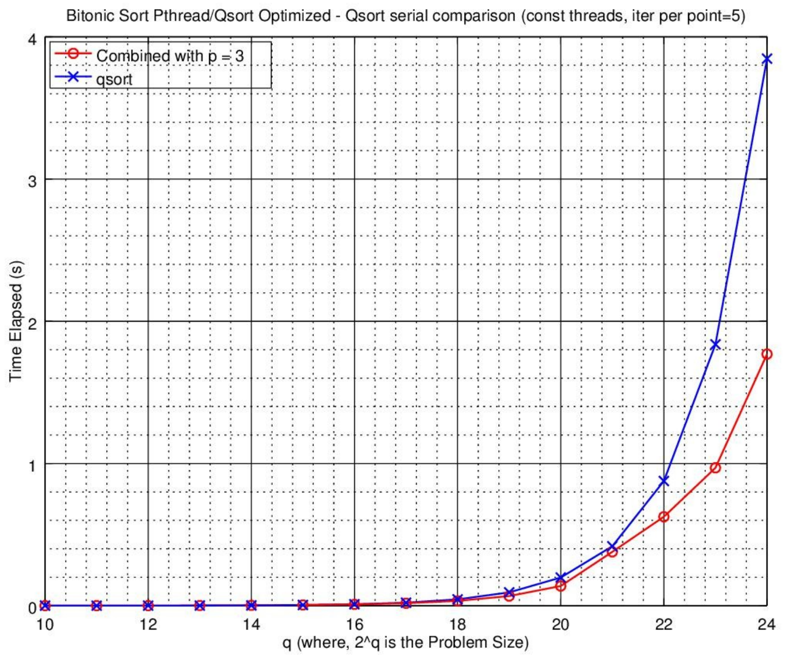
When we use the combination of bitonic sort and qsort the time is reduced, as can be seen from the next diagram.



Drawing 2: Bitonic Sort - Pthreads Recursive (optimized)

From the corresponding OpenMP and Cilk Plus diagrams we can see a similar trend. They are attached in the system files.

Moreover, we present a comparison with qsort and the optimal solution which was achieved for 8 threads and with the combination of bitonic sort and qsort, where we see a clear reduction in execution time.



Drawing 3: Bitonic Sort (Pthreads Optimal) - Qsort comparison

Finally we present the table with execution time for q = 24, και p = 3. The parallel_threshold (combination of bitonic sort with qsort), was 2^{21}.

Algorithm	Time (s)
Bitonic Serial imperative	16.8350
Bitonic Serial recursive	11.4940
Bitonic Parallel pthreads - basic	8.8353
Bitonic Parallel pthreads – qsort optimization	1.7692
Bitonic Parallel OpenMP – qsort optimization	5.3690
Bitonic Parallel CilkPlus – qsort optimization	1.9072
Quick Sort stdlib.h	3.8450

Conclusion

A parallel bitonic sort algorithm was implemented and compared against the serial counterpart and qsort from stdlib.h. For the optimum case that both bitonic sort and qsort were combined the result was improved execution time in comparison with qsort.