BLOCK INDIRECT

A new parallel sorting algorithm

Francisco Jose Tapia
fjtapia@gmail.com

BRIEF

Modern processors obtain their power increasing the number of “cores” or HW threads, which permit them to execute several processes simultaneously, with a shared memory structure.

SPEED OR LOW MEMORY

In the parallel sorting algorithms, we can find two categories .

SUBDIVISION ALGORITHMS
Filter the data and generate two or more parts. Each part obtained is filtered and divided by other threads, until the size of the data to sort is smaller than a predefined size, then it is sorted by a single thread. The algorithm most frequently used in the filter and sort is quick sort.

These algorithms are fast with a small number of threads, but inefficient with a large number of HW threads. Examples of this category are :

- Intel Threading Building Blocks (TBB)
- Microsoft PPL Parallel Sort.

MERGING ALGORITHMS
Divide the data into many parts at the beginning, and sort each part with a separate thread. When the parts are sorted, merge them to obtain the final result. These algorithms need additional memory for the merge, usually an amount equal to the size of the input data.

With a small number of threads, these algorithms usually have similar speed to the subdivision algorithms, but with many threads they are much faster . Examples of this category are :

- GCC Parallel Sort (based on OpenMP)
- Microsoft PPL Parallel Buffered Sort

SPEED AND LOW MEMORY

This new algorithm is an unstable parallel sort algorithm, created for processors connected with shared memory. This provides excellent performance in machines with many HW threads, similar to the GCC Parallel Sort, and better than TBB, with the additional advantage of lower memory consumption.

This algorithm uses as auxiliary memory a 1024 element buffer for each thread. The worst case memory usage for the algorithm is when elements are large and there are many threads. With big elements (512 bytes), and 32 threads, the memory measured was:

- GCC Parallel Sort 1565 M
- Threading Building Blocks (TBB) 783 M
- Block Indirect Sort 814 M

INDEX

1.- OVERVIEW OF THE PARALLEL SORTING ALGORITHMS

Among the unstable parallel sorting algorithms, there are basically two types:

1.- SUBDIVISION ALGORITHMS

As Parallel Quick Sort. One thread divides the problem in two parts. Each part obtained is divided by other threads, until the subdivision generates sufficient parts to keep all the threads buys. The below example shows that this means with a 32 HW threads processor, with N elements to sort.

Step	Threads working	Threads waiting	Elements to process by each thread
1	1	31	N
2	2	30	N / 2
3	4	28	N / 4
4	8	24	N / 8
5	16	16	N / 16
6	32	0	N / 32

Very even splitting would be unusual in reality, where most subdivisions are uneven

This algorithm is very fast and don't need additional memory, but the performance is not good when the number of threads grows. In the table before, until the 6th division, don't have work for to have busy all the HW threads, with the additional problem that the first division is the hardest, because the number of elements is very large.

2.- MERGING ALGORITHMS,

Divide the data into many parts at the beginning, and sort each part with a separate thread. When the parts are sorted, merge them to obtain the final result. These algorithms need additional memory for the merge, usually an amount equal to the size of the input data.

These algorithms provide the best performance with many threads, but their performance with a low number of threads is worse than the subdivision algorithms.

2.- *BENCHMARKS*

2.1.- INTRODUCTION

To benchmark this we use the implementation proposed for the Boost Sort Parallel Library. It's pending of the final approval, due this can suffer some changes until the final version and definitive approval in the boost library. You can find in https://github.com/fjtapia/sort_parallel.

If you want run the benchmarks in your machine, you can find the code, instructions and procedures in https://github.com/fjtapia/sort_parallel_benchmark

For the comparison, we use these parallel algorithms:

1. GCC Parallel Sort
2. Intel TBB Parallel Sort
3. Block Indirect Sort

2.2.- DESCRIPTION

This benchmark was run on a Dell Power Edge R520 12 G 2.1 GHz with two
Intel Xeon E5-2690, with the Hyper Threading activate and with 32 HW threads.

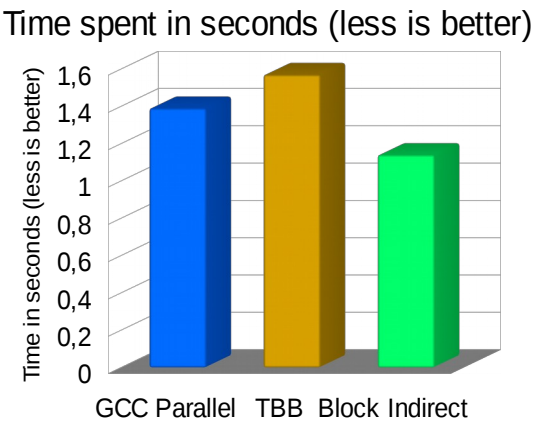
The compiler used was the GCC 5.2 64 bits

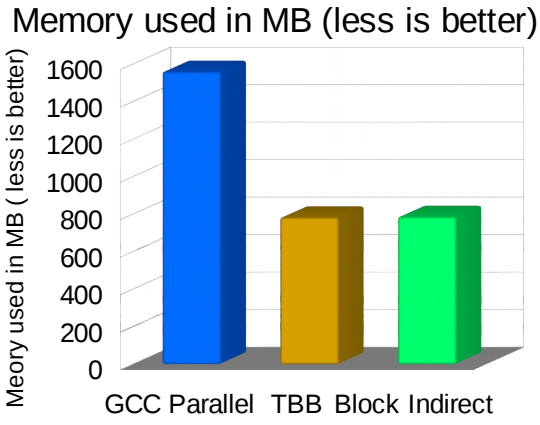
The benchmark have 3 parts:

- Sorting of 100 000 000 64 bits numbers
- Sorting of 10 000 000 strings
- Sorting of objects of different sizes, with different comparison methods.

2.3.- NUMBERS

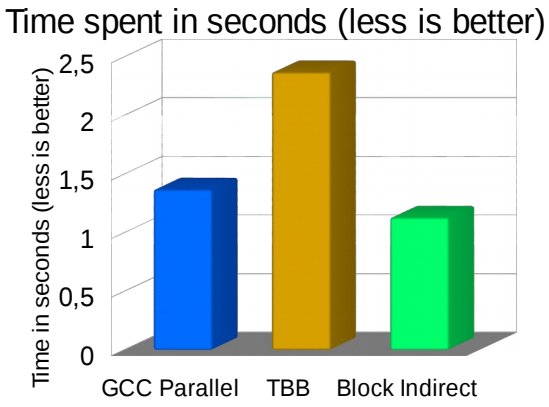
Sorting of 100 000 000 random 64 bits numbers.

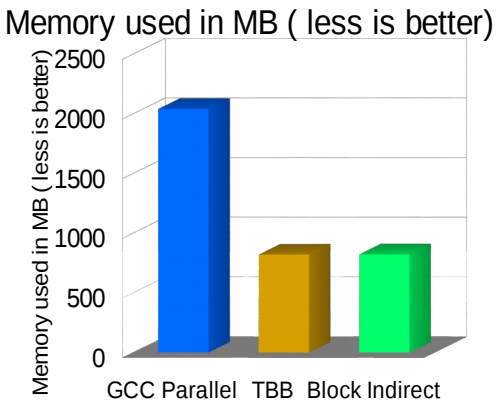




2.4.- STRINGS

Sorting 10 000 000 strings, randomly filled



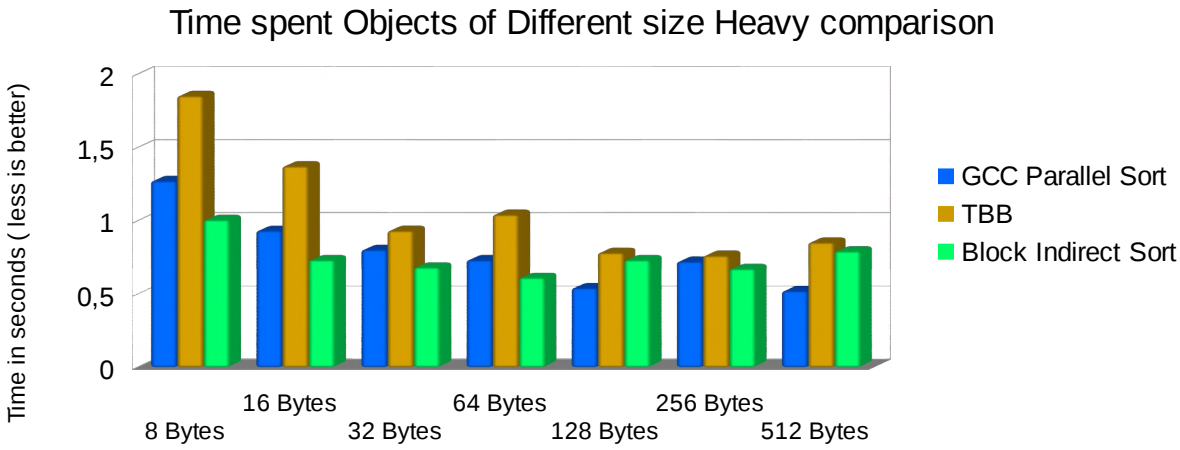


2.5.- OBJECTS

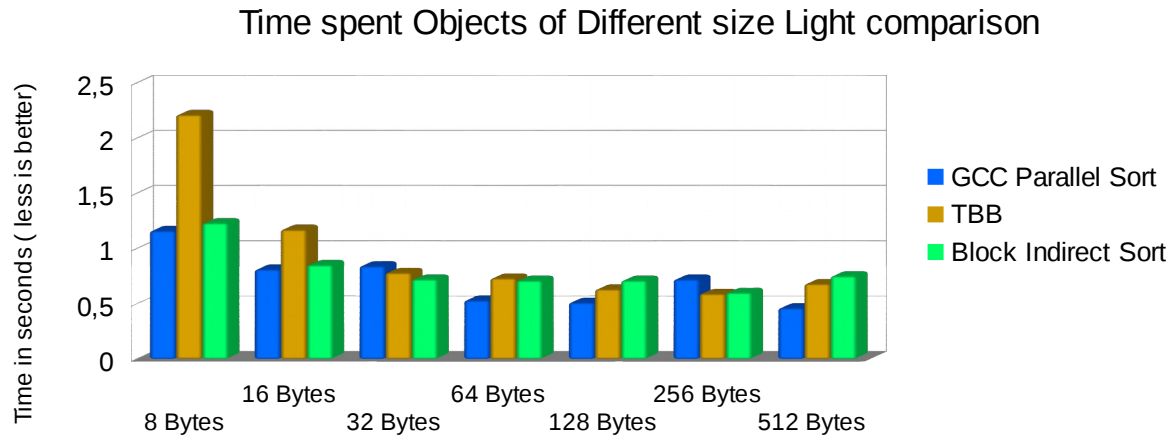
The objects are arrays of 64 bits numbers, randomly filled. An 8 byte object is an array of 1 element, a 16 bytes object have 2 numbers and successively , until the object of 512 bytes, which have 64 numbers.

The benchmark had been done using two kind of comparison:

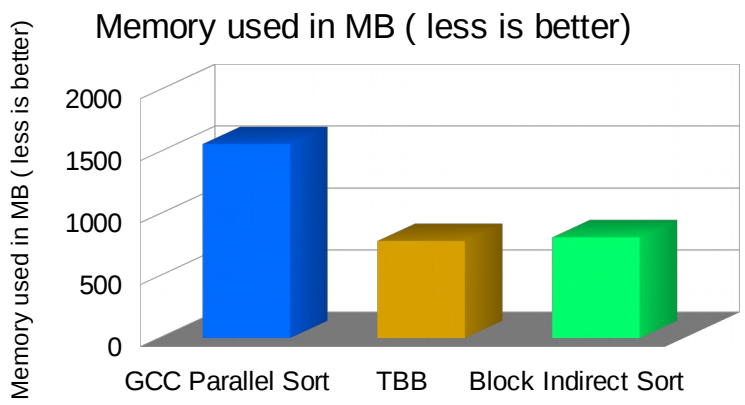
1.- Heavy comparison. The comparison is the sum of all the numbers in the array. In each comparison, the sum is done



2.- Light comparison. The comparison is done comparing the first element of the array as a key.



3.- Memory used



3.- BIBLIOGRAPHY

- **Introduction to Algorithms,** 3rd Edition (Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein)
- **Structured Parallel Programming: Patterns for Efficient Computation** (Michael McCool, James Reinders, Arch Robison)
- **Algorithms + Data Structures = Programs** (Niklaus Wirth)

4.- GRATITUDE

To **CESVIMA** (http://www.cesvima.upm.es/), **Centro de Cálculo de la Universidad Politécnica de Madrid.** When need machines for to tune this algorithm, I contacted with the investigation department of many Universities of Madrid. Only them, help me.

To **Hartmut Kaiser**, Adjunct Professor of Computer Science at Louisiana State University. By their faith in my work,

To **Steven Ross**, by their infinite patience in the long way in the develop of this algorithm, and their wise advises.