A Novel Sorting Algorithm and its Application to a Gamma-Ray Telescope Asynchronous Data Acquisition System

Alberto Colavita[†] *Enzo Mumolo*[‡] *Gabriele Capello*[‡]

[†] Microprocessor Laboratory, ICTP-INFN, Via Beirut 31, 34100 Trieste, Italy

[‡] DEEI, Universita' di Trieste, Via Valerio 10, 34127 Trieste, Italy

Tel/Fax: +39.40.676.3861/3460 e-mail: mumolo@univ.trieste.it

ABSTRACT

In this paper we present a novel parallel sorting algorithm, highly suited for VLSI implementation, which works through a cascade of elementary sorting units and leads to a scalable architecture. The paper describes the applications of such device to the asynchronous data acquisition for a gamma ray telescope.

1 Introduction

Sorting of a series of numbers is a very important task in many signal processing areas, such as order statistics non linear filtering [1] or communication switching systems [2]. In this paper we address the common data acquisition problem of building an event out of a set of data read asynchronously, by various part of an extended instrument. We will use GLAST [3] as an example of the new generation of gamma ray telescopes that present the stated data acquisition problem. Obviously the use of the sorting device goes well beyond this application, as it can be very useful in other similar acquisition problems, or in other applications requiring high speed sorting. It is important to note that the important characteristic of this algorithm is its simplicity and suitability for VLSI implementations. Other algorithms, namely the algorithms belonging to the network sorting class, can be more efficient from a computational point of view, but they are less versatile. The Batcher algorithm [5], for example, is slightly faster than the proposed algorithm with an equal number of comparators, but if we want to modify the total number of comparators, a new design of the sorting device would be undertaken. The proposed algoritm, instead, leads to a scalable architecture and the system can be expanded simply by adding other sorting devices in cascade.

2 The sorting algorithm

The sorting device processes a block of N numerical data given at its input. In this section we give a detailed analysis of the algorithm. In order to prepare for this analysis, it is worth describing very briefly some basic issues of the classical sorting algorithms [5].

2.1 Basic issues of the sorting problem

Generally speaking, sorting algorithms can be divided into two classes, namely Internal and External, the first making use of internal RAM memory and the second of some kind of mass memory. The "Internal" algorithms, use some approach to make efficient the sorting task, and can be based on the counting, the insertion, the exchanging, the selection, the merging and the distribution of data. Another different approach is based on the parallelism of the operations, leading to the so called "Sorting Network" [5]. The "External" algorithms, finally, try to efficiently use the external device.

2.2 Description of the proposed algorithm

Usually, applications require that the sorting of data be performed according to the value of a key. With this assumption, a sorting unit is made of two registers, one for data and one for the key. Moreover, each sorting unit consists of two sides sharing the same architecture and a control logic implementing a three state machine: *Reset*, *Shift&Read1* and *Shift&Read2* states. During *Reset*, the contents of the registers are set to zero, while during *Shift&Read*x the content of the key is passed to the output and a new value is read in the 'x' section. The structure of a sorting unit is reported in fig.1.

A sorting device is made out of a concatenation of identical sorting units, thus defining a scalable architecture. Each sorting unit shares its input bus with the output bus of the previous and it shares the output bus with the input bus of the next. The algorithm performed by the control logic is described in fig.2.

In Fig.3 an example of sorting is shown, where the input data enters sequentailly, one data at each clock cycle, into the device. In this example we considered an inverse ordered sequence. From this figure, we can observe that the output sequence has a delay related to the number of sorting units.

2.3 Detailed analysis of the complexity

If we consider a stream of data passing through a single sorting unit, we came out with the observation that the obtained sequence would be equal to a sequence obtained with one pass of the popular "Bubble Sorting"

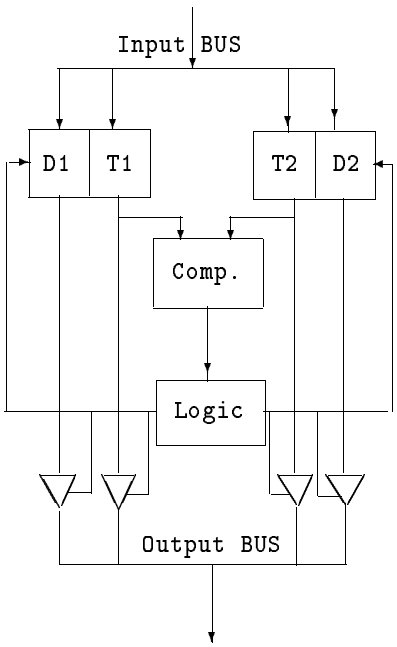

Figure 1: Block diagram of the sorting unit

```
Reset;
While (1) {
        do {
              Shift&Read1;
        } while (T1 ≠ 0)&&(T1 < T2);
        do {
              Shift&Read2;
        } while (T2 ≠ 0)&&(T2 < T1);
}
```

Figure 2: Pseudocode of the sorting algorithm

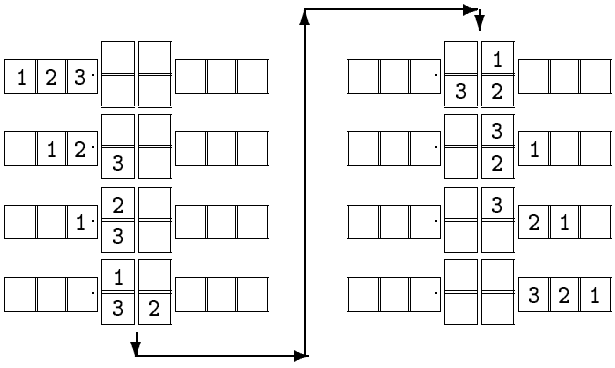

Figure 3: Sorting of a three data sequence with two sorting units

algorithm. That is, the result of a single sorting unit processing a stream of data is the same of the exchanging pass of the bubble sort. Hence, we can use some classical results to our problem. A first result is described in the following Theorem [5].

Theorem 1 *Let $a_1, a_2, \cdots, a_n$ be a permutation and let $b_1, b_2, \cdots, b_n$ be the corresponding inversion table. If one pass of the bubble sort changes $a_1, a_2, \cdots, a_n$ to the permutation $a'_1, a'_2, \cdots, a'_n$, the corresponding inversion table $b'_1, b'_2, \cdots, b'_n$, is obtained from $b_1, b_2, \cdots, b_n$ by decreasing each non zero entry by 1.*

Therefore, we can say that if a stream of data passes through a sorting unit, each non zero element of the relative inversion table is decreased by 1. Since the non zero inversion table elements are, at a maximum, $N-1$, we can conclude that:

Proposition 1 *A cascade of $N-1$ sorting units realizes a complete ordering of a N-long data sequence.*

According to [5], the execution time of the bubble sort algorithm can be represented with $K_1 \cdot A + K_2 \cdot B + K_3 \cdot C + K_4$ where A is the number of passes, B the number of exchanges and C the number of comparisons and $K_1 - K_4$ represent the implementation aspects. The complexity of the proposed algorithm is instead given by $K_1 \cdot A + K_4$, since the exchanges and comparisons are performed in parallel with the input of the data at each clock cycle. In the above relation, the K_4 coefficient keep track of the constant execution times and K_1 is the counterpart of the number of passes of the Bubble Sort. If S is the number of sorting units, let us assume that $S = N - 1$ and that the input data block is N-sized. In general, since $A = N - 1$, we have $K_1 = \frac{N}{S}$ and $K_4 = 2S$. In conclusion, the computational complexity of a cascade of $S = N - 1$ sorting units with a N input data is $3N - 2$. These considerations are summarized in the following

Proposition 2 *The complexity for a (N-1) sorting units device and a N-long input sequence is $O(N)$.*

Let us consider now the case $N - 1 > S$, i.e. the case where the length of the input data sequence is greater than the number of sorting units. In this case we need an external memory in order to store the data which cannot be contained in the S sorting units, that is a $N - 2S$ sized external FIFO. The data sequence must completely pass through the sorting device $(N-1)/S$ times, where the slash represents an integer division, plus $(N-1)\%S$ partial passes. The sorting system is therefore described in Fig.4.

The main problem, in this case, is that if we have a sequence of N sized input data blocks, we must devise a way to separate the different blocks, which otherwise will be merged. To solve this problem, we extended

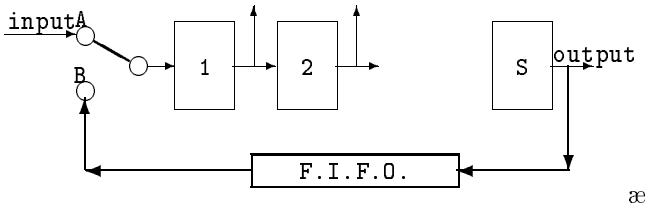

Figure 4: Architecture of the sorting device with external memory

cond.	F1	F2	LE	Oper.
0	0	0	0	NC
1	0	0	1	NC
2	0	1	0	NC1
3	0	1	1	SR1
4	1	0	0	SR2
5	1	0	1	NC2
6	1	1	0	SR2
7	1	1	1	SR1

SR1 : *Shift&Read1*; LE = 1;
SR2 : *Shift&Read2*; LE = 2;
NC : If (T1 $\leq$ T2) SR1 Else SR2;
NC1: If (T1 $\leq$ T2) F1=1; SR1; F2=0; Else SR2;
NC2: If (T1 $\leq$ T2) SR1; Else F2=1; SR2; F1=0;

Figure 5: Description of the revised logic for the sorting algorithm

the sorting algorithm in the following way: every input number is augmented with an additional bit, called in the following F, to be used by the algorithm as a marker to identify the starting point of a sequence. In other words, it is set at the starting of each block sequence, and otherwise it is zeroed. Each sorting unit has another bit, called in the following LE (last entry), which records the position of the last entered data in the sorting unit. The algorithm performed in the logic block of Fig. 2 performs the operations described in Fig. 5. In this way, the different input data blocks remain separated.

Regarding the computational complexity of the algorithm in the case $N > S + 1$, we have to separately consider two cases, namely when N is less or greater than $2S$, because in the first case there is no need for a FIFO external storage. The complexity in these two cases is described in the following Proposition.

Proposition 3 *Let us define the following terms:* $P_i = \frac{N-1}{S}$, *with integer division, and* $P_r = (N-1)\%S$. *The complexity of the device, expressed in terms of number of clock cycles, is the following*

data size	*FIFO size*	*complexity*
$(S+1) < N \leq 2S$	0	$2N + 2P_r$
$N > 2S$	$N - 2S$	$N(P_i + 1) + 2P_r$

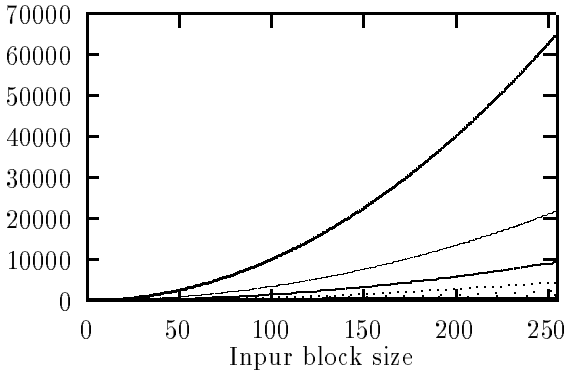

Figure 6: Computational complexity vs. input size varying the number of sorting units for the novel algorithm

The results described in Propositions 2 - 3 are reported in Fig. 6, where a family of curves is shown, from the upper part of the panel to the lower, for 1, 3, 7, 15, 31, 63, 127 and 255 sorting units. In the ordinate the number of clock cycles is represented.

2.4 Comparison with the Batcher algorithm

This algorithm [5] looks for inversion in the data, in such a way that the comparison and exchange steps are executed in parallel. The system is composed by a sorting network and a merging unit and is described in Fig. 7.

In this system, the N long input data sequence is divided into B sub blocks, which is the amount of data simultaneously processed by the sorting network. Fig.8 reports the complexity of such a system versus the input size for different values of B, namely 2 - the straight line-, 8 - represented by the curve with small steps - and 32, which is the curve with the bigger steps. The total number of comparators is a function of B and N and, for each N, it has a minimum for a certain value of B. If we consider the case N=256, we get a minimum at B=8, which corresponds to 50 comparators. From Fig.8 we get that for N=256 and B=8 the complexity is about 700. However, the proposed algorithm with 50 comparators, i.e. with 50 sorting units, leads to a complexity of about 1000. Therefore, a simple Batcher circuit is more efficient than the proposed algorithm. What makes our algorithm highly suitable for applications is its simplicity and scalability.

3 Application of the sorting device to high speed asynchronous data aquisition

The application we are currently developing concernes the realization of an instrument for physics experiments

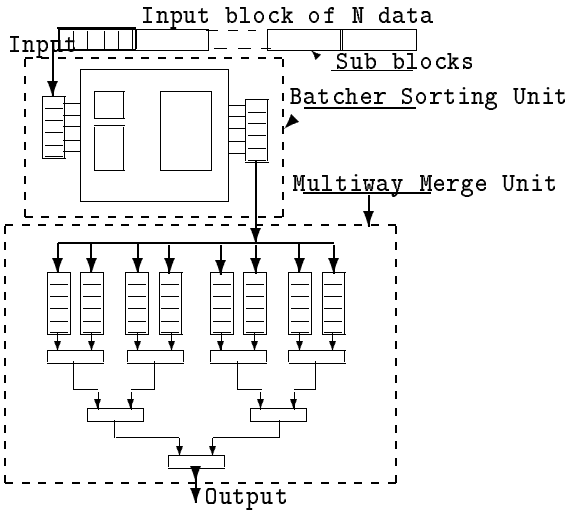

Figure 7: Block diagram of the Batcher algorithm

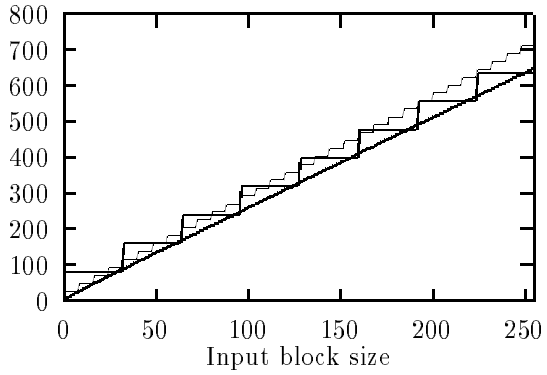

Figure 8: Computational complexity vs. input size varying the number of simultaneously sorted data for the Batcher algorithm

in the field of gamma-ray detection. The system, called 'GLAST', is formed by a number of towers made by several planes, based on silicon strip diodes. Each pixel of the planes lead to a data channel containing information on X-Y coordinates of the point hit by a ray. The data channel then is completed with the plane and tower number and the instant of hit. In total, GLAST has 1.4 million data channels. When a gamma-ray interacts with the instrument it creates a positron-electron pair that triggers signals in many silicon strips. The data generated by the charged particles is sent into the channel. One problem of this system is the high number of channels and the asynchronous nature of the events. Another problem is how to filter out the non relevant data. Gamma-ray and cosmic rays strike the instrument at random time. Cosmic rays have a flux 10000 larger than gamma-rays and we have to distinguish the signal generated by the positron-electron pair created by the gamma-ray from the large background of cosmic rays [4]. If we decide to read the full 1.4 million channel when an event occurs we would find that only a few, may be 10-50 channels have values different from zero. Still, the full instrument will be unavailable to record a new event while reading the data. If we increase the size of the telescope we increase at the same time the number of data channels. Hence, there would be a size (channel count) at which it becomes pointless to improve the aperture without the instrument presenting an unbearable dead time. The solution is to allow each channel to autonomously place a channel number and the time into some sort of pipe. Each channel is unaware of the activity of other channels and hence the data is finally conveyed into a stream of data scrambled in time and channel number. Since time increases continuously the data within the stream is partially sorted in time since the data acquired at t0 cannot be too far away one from the other. To analize if an event was produced by noise, a cosmic ray or a gamma ray we need to sort the channel numbers by time, placing together all the data generated, within the instrument, at a given time slot. In the case of GLAST the instrument generate 1 Mw/sec, were a word is 42 bit long. Sorting such a data stream is beyond the capabilities of present computers while our sort chip is capable of sorting it as it reads it and with a latency proportional to the number of sorting units.

4 Conclusions

A novel parallel sorting algorithm has been described in this paper. Its main advantage is its semplicity, which makes it highly suitable to a VLSI implementation. Moreover, the overall sorting system is scalable, and this is fundamental from a practical point of view. Other faster algorithms do exist, but at the expenses of less versatility and simplicity. A detailed analysis of the computational complexity has been briefly summarized and an application to the field of instrumentation in physics has been introduced.

References

[1] I.Pitas, A.N.Venetsanopulos, *Non Linear Digital Filters*, Kluwer Academic Publishers, 1990

[2] A.Cicuttin, A.Colavita, F.Fratnik, S.Venkataraman, "A novel Algorithm and VLSI Architecture for a Time Switch", *to be published*

[3] "The GLAST proposal", response to the NASA Research Announcement NRA 94-OSS-15, 1994.

[4] Leo, W. R., *Techniques for Nuclear and Particle Physics Experiments*, Springer-Verlag, New York, 1994.

[5] D.E.Knuth, *The art of computer programming. Sorting and searching*, Addison Wesley, 1975