Realistic Calorimeter Feature Extraction: Algorithm, Benchmarks, and Implementation Options

1.0 Introduction

This document describes the algorithm for the data preparation procedure which precedes the electron/gamma feature extraction for the ATLAS calorimeter system [1, 2]. The aim of data preparation is to translate the raw data [3], coming in the packets sent from the readout buffers, into RoI information which the feature extraction code can handle. Benchmarks of both data preparation and feature extraction itself are made in a set of different technologies, covering the range of processors used in the trigger level-2 demonstrator programme [4] in the ATLAS collaboration. We are thankful to A. Bogaerts, G. Crone, R. Dankers, E. Denes, R. Hauser, C. Hortnagl and J. Seixas for helpful discussions concerning this work.

2.0 Subdivisions of Feature EXtraction (FEX)

The entire FEX procedure can be divided into three main phases: Data collection, data preparation and feature extraction itself.

The data collection phase is responsible for collecting all ReadOut Buffers’ (ROBs) packets concerning to the same event and RoI (those packets arrive asynchronously for both events and RoI fragments), and sending them for data preparation. In the data preparation phase, raw data inside the ROBs’ packets are translated into RoI information which can be processed by the feature extraction code.

Complete FEX algorithm

Packet collection ---> Data preparation ---> Feature Extraction itself

Initial ideas for the data collection code have been defined so far, as well as an initial approach for its interface with the data preparation part. In the present work, we use a simplified version of the interface defined by R. Hauser in his collection code, in which a structure of RoID [5] packets (the so called “struct collect”), concerning to the same event and RoI IDs, is sent for data preparation. This is described in figure 1.
struct RoID {
    int roi_eta_index;
    int roi_phi_index;
    int subdetector_no;
    int crate_id;
    int rob_id;
    int data[500];
};

/* max number of fragments per ROI expected */
#define MAX_FRAGMENTS 10 /* For the calorimeter */

/* structure to collect all information and data */
/* for an event/roi pair */
struct collect {
    /* identification */
    int event_id; /* event identifier */
    int roi_id; /* ROI identifier */
    int expected; /* number of packets to follow */

    /* data */
    struct RoID packet[MAX_FRAGMENTS];
};

FIGURE 1. Simplified structure sent from the data collection to the data preparation code.

All fields in the structures of figure 1 are described in details in [3]. The structures shown are a simplified test version (just to allow initial tests and benchmarks using dummy raw data) of the ones being used in the demonstrator programme. Thus, only the useful fields for the data preparation code are included, however other fields would be present in a demonstrator’s situation.

Both data preparation and feature extraction itself have been implemented, here, in the same piece of C code, and the interface between them will not be described. It’s also assumed, so far, that the FEXD [5] packets are sent to the global network by the collector code.

3.0 Generation of dummy raw data files

The readout and data formats definitions used in the present work have been described in [3]. Three different raw data files, in the form of the “struct collect” shown in figure 1, have been generated for tests. The set of tools to generate such files consists of PAW macros, for filling in the data words fields of the different possible data formats corresponding to a ROB packet, and one C program to compose them into the final “collect” structure.
Each of the three files generated corresponds to one specific RoI in the eta-phi space of the calorimeter system. Description follows:

1. **10 ROBs involved**: this file corresponds to the RoI described in figure 11, page 11, of [3]. Such RoI hits all the four calorimeter portions\(^1\), sitting in the overlap region between barrels and endcaps. The RoI eta index is 38 (\(\eta = +1.30\)), being 61 (\(\phi = 61 \pi /32\)) the phi index (which means that the index wraps around in phi). A total of 10 ROBs have to be read out (which means 10 RoID packets sent from the buffers to the FEX node), collected and prepared in this situation (6 in the electromagnetic part, 4 in the hadronic);

2. **8 ROBs involved**: this file corresponds to the upper RoI described in figure 10, page 10, of [3], sitting in the endcaps’ region. The RoI eta index is 43 (\(\eta = +1.80\)), and the phi index 63 (\(\phi = 63 \pi /32\), also wrapping around). 8 ROBs’ packets have to be read out, collected and prepared in this situation (6 in the electromagnetic part, 2 in the hadronic);

3. **3 ROBs involved**: this file corresponds to the lower RoI described in figure 9, page 9, of [3]. The RoI eta index is 19 (\(\eta = -0.6\)), and the phi index is 0 (\(\phi = 0\)). Only the em. (2 ROBs) and had. (1 ROB) barrels are involved in the RoI, which means a total of 3 RoID packets to be collected and prepared.

### 4.0 The data preparation procedure

The data preparation code is composed, basically, by two main loops:

**Loop over packets**
```
{ ... }
```

**Loop over RoI positions**
```
{ ... }
```

**Feature extraction itself**

### 4.1 The *loop over packets*

In the *loop over packets*, all the packets sent from the data collection code (thus, corresponding to one only event/RoI ID) are read. A lookup table is then used to find out, for each packet, the eta and phi index of the corresponding ROB, as well as the granularity index of each trigger tower in the ROB. The granularity index is a number which uniquely defines the subdetector’s portions and the layers of each portion to be taken into account for the given tower, as well as the granularity profile of each layer.

---

1. from now on, a “calorimeter portion” will be defined as being one among four possibilities: the electromagnetic barrel, electromagnetic endcaps, the hadronic barrel or the hadronic endcaps. The field “subdetector_no” (see figure 1) identifies the portion. However, the use of the term “subdetector” is avoided in the text, to prevent misunderstandings.
All the possible granularity indices, and their corresponding eta ranges, are described in figure 2. That corresponds completely to what is described in [3], section 4.0.

Figure 3 illustrates the process of reading the lookup table.

![Granularity index 1](image)

|eta| < 1.40

![Granularity index 2](image)

1.40 < |eta| < 1.50

![Granularity index 3](image)

1.50 < |eta| < 1.60

![Granularity index 4](image)

1.60 < |eta| < 1.80

![Granularity index 5](image)

1.80 < |eta| < 2.00

![Granularity index 6](image)

2.00 < |eta| < 2.40

![Granularity index 7](image)

2.40 < |eta| < 2.50

**FIGURE 2. Possible Granularity indices in the calorimeter (from [3]).**

After reading the lookup table information, each trigger tower inside the given packet receives eta and phi indices, based on the eta and phi indices of the ROB. Thus, the code can recognize which towers in the packet actually belong to the ROI (the valid towers) by simply comparing their indices with the eta and phi indices of the RoI (see figure 1). The valid towers also have their positions identified inside the RoI 4x4 eta-phi matrix. The remaining ones are ignored. Summarizing:
Loop over packets

\{
  \text{Read packet;}
  \text{Call lookup table;}
  \text{Find eta and phi indices of each tower in the given packet;}
  \text{Loop over trigger towers inside the given packet}
  \{
    \text{Find out whether this tower actually belongs to the given RoI;}
    \text{If yes, find out its position inside the RoI 4x4 eta-phi matrix;}
  \}
\}

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{lookup_table}
\caption{Reading the lookup “magic” table.}
\end{figure}

4.2 The loop over RoI positions

In this phase, each of the 16 RoI positions (one position corresponds to one trigger tower in the different calorimeter portions) is “mounted”. The meaning of the term “mounted” depends on the feature extraction algorithm which will be used. In the present case, it’s based on B. Thooris’ algorithm for calorimeter feature extraction, which only considers the radial energy distribution around the particle’s interaction point, in the eta-phi plane, and ignores the information in depth (r direction). Thus, in the present case, to “mount” the RoI position means to project all energy information in the eta-phi plane, for both electromagnetic and hadronic calorimeters, with a granularity of .025 x (2π/256) in eta x phi (the granularity of the middle layer in the electromagnetic barrel) for the em. part, and of .1 x (2π/64) for the had. part.

Depending on the granularity index of a given RoI position, different auxiliary functions have to be called. Once different calorimeter portions will send different packets (they are read out by different ROBs), more than one auxiliary function may be called (each one dealing with a unique packet) for a given RoI position, depending on the number of portions concerned (see figure 2). Table 1 summarizes which functions are called for a given granularity index. The actual functions’ names used in the code
are shown. Their names are suggestive (e.g. “embar1” means “function 1 for the em. barrel”), and could be directly compared to what is described in figure 2.

<table>
<thead>
<tr>
<th>Granularity index of the given RoI positions</th>
<th>Auxiliary functions to be called</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TT_assemb_embar1</td>
</tr>
<tr>
<td></td>
<td>TT_assemb_hadbar1</td>
</tr>
<tr>
<td>2</td>
<td>TT_assemb_emend1</td>
</tr>
<tr>
<td></td>
<td>TT_assemb_hadbar1</td>
</tr>
<tr>
<td>3</td>
<td>TT_assemb_embar1</td>
</tr>
<tr>
<td></td>
<td>TT_assemb_hadbar1</td>
</tr>
<tr>
<td></td>
<td>TT_assemb_hadend1</td>
</tr>
<tr>
<td>4</td>
<td>TT_assemb_embar1</td>
</tr>
<tr>
<td></td>
<td>TT_assemb_hadend2</td>
</tr>
<tr>
<td>5</td>
<td>TT_assemb_emend2</td>
</tr>
<tr>
<td></td>
<td>TT_assemb_hadend2</td>
</tr>
<tr>
<td>6</td>
<td>TT_assemb_emend3</td>
</tr>
<tr>
<td></td>
<td>TT_assemb_hadend2</td>
</tr>
<tr>
<td>7</td>
<td>TT_assemb_emend4</td>
</tr>
<tr>
<td></td>
<td>TT_assemb_hadend2</td>
</tr>
</tbody>
</table>

Summarizing:

**Loop over RoI positions** (4x4 matrix = 16 positions)

{  
  For the given position, check the gran. index (read from the lookup table);  
  Call correspondent auxiliary functions;  
}

4.3 An output example

Presently, there are 3 different versions of the data preparation + feature extraction code: one library, to be integrated with a real collection code, one test version, with a fake collection code, for debugging purposes, and one benchmark version. An example of output of the test version follows:

Main program started...  
File for reading open...  
Data read...  
CALLING DO_FEX!

Initializing lookup table...  
Reading lookup table file...  
Table read!
Starting memory allocation for the table...
Table contents allocated in memory!

Starting main loop over packets...

This is packet 1.
Value of subdetector number: 8.
Calling magic table now...
Information from the magic table has been read...
ROB eta index: 37
ROB phi index: 20
Finished eta_phi indexing of trigger towers...
Tower 6 belongs to ROI in this packet...
Tower 7 belongs to ROI in this packet...
Tower 8 belongs to ROI in this packet...

This is packet 2.
Value of subdetector number: 8.
Calling magic table now...
Information from the magic table has been read...
ROB eta index: 37
ROB phi index: 24
Finished eta_phi indexing of trigger towers...
Tower 5 belongs to ROI in this packet...

This is packet 10.
Value of subdetector number: 11.
Calling magic table now...
Information from the magic table has been read...
ROB eta index: 40
ROB phi index: 24
Finished eta_phi indexing of trigger towers...
Tower 1 belongs to ROI in this packet...
Tower 9 belongs to ROI in this packet...

Starting loop over towers in the formed ROI...

ROI position eta,phi = 1,1...
Granularity index: 1
Entered embar1 function (electromagnetic barrel’s packet) ...
Taking TT# 6 (which belongs to ROI) in the packet and skipping the others ...
Entered hadbar1 function (hadronic barrel’s packet) ...
Taking TT# 54 (which belongs to ROI) in the packet and skipping the others ...

ROI position eta,phi = 4,4...
Granularity index: 4
Entered embar1 function (electromagnetic barrel’s packet) ...
Taking TT# 5 (which belongs to ROI) in the packet and skipping the others ...
Entered hadend2 function (hadronic endcaps’ packet) ...
Taking TT# 9 (which belongs to ROI) in the packet and skipping the others ...

Feature extraction itself starts here...
The program also stores a file with the mounted RoI contents for checking.

## 5.0 Benchmarks results and discussion

From now on, the following conventions are assumed:

- **Total time**: Includes execution time for data preparation and feature extraction itself, but excludes the execution time for packet collection;
- **Loop over packets**: execution time for the “loop over packets”, as described in section 4.1;
- **Feature extraction itself**: Only execution time for the physics code based on B. Thooris’ algorithm;

Table 2 contains a brief description of the stations and boards used for the benchmarks.

### Table 2. Boards’ description.

<table>
<thead>
<tr>
<th>Board</th>
<th>Processor</th>
<th>OS</th>
<th>Compiler</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>CES RIO2 8061</td>
<td>PowerPC 604.96MHz.</td>
<td>LynxOS version 2.3.1</td>
<td>gcc</td>
<td>The PPC communicates to the DRAM via a PCI bridge chip (the IBM “Lanai Kauai” 27-82660)</td>
</tr>
<tr>
<td>AlphaStation 200</td>
<td>ALPHA. 166 MHz.</td>
<td>Digital unix V4.0B</td>
<td>cc</td>
<td>2.95 SPECint95</td>
</tr>
<tr>
<td>AlphaStation 255</td>
<td>ALPHA. 233MHz.</td>
<td>Digital unix V4.0B</td>
<td>cc</td>
<td>4.27 SPECint95</td>
</tr>
<tr>
<td>Alpha Server 4000</td>
<td>ALPHA. 300MHz.</td>
<td>Digital Unix V4.0B</td>
<td>cc</td>
<td>8.11 SPECint95</td>
</tr>
<tr>
<td>Loughborough Sound Images DBV44</td>
<td>Texas Instruments TMS320C40.40MHz.</td>
<td>None</td>
<td>TMS320 floating-point C compiler V4.5</td>
<td>The C40 is directly connected to the external memory through its “local and global memory buses”. The processor operates, internally, at 20MHz.</td>
</tr>
</tbody>
</table>

Table 3 summarizes some results.¹
From table 3, one can realize that variations in the number of packets (or ROBs) involved do not affect proportionally the execution time of the algorithm. This was expected since the code was written in order to ignore the trigger towers which do not belong to the RoI, and only consider the fixed-sized 4x4 matrix of towers. More specifically, only the “loop over packets” (see section 4.1) has execution time dependent on the number of packets, while the “loop over RoI positions” (see section 4.2) has execution time dependent on the RoI size, which is constant for the electron/gamma trigger, and RoI eta position. In order to confirm that the “loop over packets” is not remarkably time consuming when compared to the rest of the code, benchmarks of its execution time have been made. Table 4 summarizes the results.

### TABLE 4. Benchmarks of the “loop over packets”.

<table>
<thead>
<tr>
<th>Input file (as described in 3.0)</th>
<th>CES RIO2 8061</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 ROBS barrel</td>
<td>43 microseconds</td>
</tr>
<tr>
<td>8 ROBs endcap</td>
<td>92 microseconds</td>
</tr>
<tr>
<td>10 ROBs overlap</td>
<td>147 microseconds</td>
</tr>
</tbody>
</table>

1. These tests were made by looping back the same input file as the entry for the algorithm. It’s reasonable to expect that the file contents would remain in the cache memory, making the access to it unrealistically fast. Anyway, we still don’t expect to include access to memory in the benchmarks, leaving this step for a future test. Thus, this effect might not hide what was intended to be measured now: the execution time of the code itself.
One can conclude that the most time consuming part of the data preparation algorithm is the “loop over RoI positions”, where each of the 16 positions of the RoI is mounted by means of a set of auxiliary functions. Thus, variations in the total time will be more related to variations in the eta position where the RoI sits (which defines the auxiliary functions to be called) than on variations on the number of packets. Particularly, the barrel is the region where the RoI mounting is more time demanding, due to the high number of channels to be taken into account per trigger tower.

For the C40, we achieved the results shown in table 5.

<table>
<thead>
<tr>
<th>Measurement and input file (as described in 3.0)</th>
<th>TMS320C40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time, 10 ROBs overlap.</td>
<td>3.7 milliseconds</td>
</tr>
<tr>
<td>Only feature extraction itself.</td>
<td>292 microseconds</td>
</tr>
</tbody>
</table>

The Texas Instruments’ “Parallel Debugger Manager”, with JTAG controlling, was not used for the tests, since it does not support standard I/O functions as “printf()”, “fopen()” and “fclose()”, which turned to be of fundamental importance in an algorithm which has to read a lookup table file. Instead, the NIKHEF C40 Server was used, with VME controlling, which allows the work to be done remotely, in a standard workstation, with standard I/O possibilities.

Some difficulties have been found during the tests, the main one being the shortage of memory in the DBV4x TI modules, as well as in the RHUL C40 boards, for storing the lookup table contents. The solution for this problem was to reduce artificially the size of the table, just keeping the lines which were actually going to be used by the input file selected, and deleting the others. Thus, the tests were limited to a single input file.
6.0 Implementation possibilities

In this section, different implementation possibilities for the FEX phases will be proposed and discussed.

6.1 ROB pre-processing

In the data preparation algorithm, it’s possible to implement the “loop over packets” and the “loop over RoI positions” in different processors, since a satisfactory interface (or packet format) between them is established. To explore the intelligence available in the readout buffers, one could think about running the “loop over packets” in parallel in the ROBs. The possible advantages are: the size of the packets sent to the FEX processors would be smaller, since only the trigger towers which actually belong to the RoI would be transmitted. Also, the processing time taken in the FEXes would be slightly shorter. Possible problems: though the execution time for the “loop over packets” is small compared to the rest of the FEX data preparation algorithm, it could represent a limitation for the ROBs.

6.2 The Alpha subfarm system

The Alpha subfarm FEX implementation [6] is particularly attractive. The idea is to implement the packet collection code in the AlphaServer (the so called “Switch-Farm Interface”, or SFI), and run the data preparation + feature extraction itself in each of the processing nodes (AlphaStations). As soon as a different event/RoI ID is completely collected in the server, it sends the set of packets to one of the processing nodes.

Another possibility, proposed by R. Hauser, is to make the collector to assign one processing node to one particular event/RoI ID pair, as discussed above, but now sending a packet for data preparation as soon as it’s collected in the server and the associated processing node is free. The server does not wait until collecting the complete set of packets concerning to the given pair. It does not mean that the buffering task is being transferred from the server to the processing nodes, since it’s reasonable to expect that new packets belonging to the same pair might arrive in the server before the processing node finishes the data preparation for the previous packet.

This is expected to optimize the FEX execution time. The data preparation code has to be adapted in order to work in a packet by packet basis, and a suitable interface has also to be defined between the collector and the processing nodes.

Figure 4 shows the possible implementation.
6.3 Shared memory

In order to take advantage of possible shared memory implementations, the data preparation code has been written in such a way that all the functions communicate to each other, and to the collector, through pointer passing. The header file of the code is reproduced above.

/*
 * calofex_lib.h
 * Include file for calofex_lib.c library, version 1.2, 02/Feb/1997
 * e/g trigger
 * Author: BK
 *
 */

extern float * do_fex(struct collect *cp); /* main function */
extern float * TT_assemb_embar1(int *cellp, int TT_numb);
extern int * TT_assemb_hadbar1(int *cellp, int TT_numb);
extern float * TT_assemb_emend1(int *cellp, int TT_numb);
extern int * TT_assemb_hadend1(int *cellp, int TT_numb);
extern int * TT_assemb_hadend2(int *cellp, int TT_numb);
extern float * TT_assemb_emend2(int *cellp, int TT_numb, int crate_id);
extern float * TT_assemb_emend3(int *cellp, int TT_numb, int crate_id);
extern float * TT_assemb_emend4(int *cellp, int TT_numb);

One example of such application has been suggested by A. Bogaerts. The idea is to map the ROB data buffers, through a SCI\(^1\) ring [7], in the address space of the FEX processors, and execute the FEX algorithm in shared memory mode. The FEX can, then, read directly in the ROBs the fields and trigger towers which are useful for the given RoI. As in 6.1, this is expected to reduce the amount of information being transmitted, but does not represent extra processing time for the ROBs. Naturally, such a kind of implementation is a so-called “pull architecture”.

### 6.4 Distributed processing implementation

The data preparation algorithm has inherent parallelism. The most obvious parallel processing application in the code is related to the auxiliary functions. As seen in table 1, for each cycle in the “loop over RoI positions”, 2 or 3 auxiliary functions are called. These functions can be parallelized, with little data transfer needed, and this would speed up the overall execution time considerably, as they are the main CP time consumers (see sec. 5.0). One could also execute code on multiple RoI positions in parallel.

One example of such idea, thought in the context of the so called SENNAPE Project [8], is the implementation of the data preparation + feature extraction code in a 16 nodes parallel computer (the Telmat TN310) based on DSPs and DS-Links, to check the possible advantages of a distributed processing environment. Implementations of other FEX algorithms have been underway in the same system [9].

Figure 5 may represent a possible system configuration when 2 RoI positions are being executed in parallel, with granularity indices 5 and 3, respectively.

---

1. “Scalable Coherent Interface”, a high speed interconnection which allows memory sharing.
7.0 Conclusions

The three main phases for the calorimeter feature extraction algorithm (packet collection, data preparation and feature extraction itself) have been described. Benchmarks of the data preparation + feature extraction phases have been made in a set of different processors which are being used in the ATLAS level-2 trigger demonstrator programme. The results have shown that the execution time of the algorithm is mostly dependent on the eta position where the RoI sits, and is not directly proportional to the number of RoID packets which form the RoI. This happens because the algorithm have been designed to ignore all the information, inside the packets, which do not belong to the fixed-sized RoI eta-phi matrix. It was also shown that the data preparation phase is the most time consuming part of the algorithm, if compared to the feature extraction itself.

Different possible implementations of the complete calorimeter FEX procedure have been discussed, including ROB pre-processing, implementation in the Alpha Subfarm System, shared memory applications, and an implementation in a distributed processing environment. The discussed implementations are concerned to a single FEX processing node, or local farm, in a vertical slice of the ATLAS level-2 trigger system.
Only electron/gamma triggers have been considered. We didn’t approach, so far, the jet trigger problem, more critical in which concerns to RoI size and number of ROBs involved per RoI.

8.0 References

   CERN/LHCC/96-42.
   15 December 1996.

   CERN/LHCC/96-41.
   15 December 1996.

   Draft 1.3, 10 February 1997.
   http://sunrans.cern.ch/~kastrup/papers/


   Draft 1.1, 23 October 1996.


   http://www.cern.ch/RD24/


   http://sunrans.cern.ch/~kastrup/papers/
Appendix 1

Follows the flow diagram for the data preparation algorithm.