Grain - A Java Data Analysis System for Total Data Readout

P. Rahkila

Department of Physics, University of Jyväskylä, P.O. Box 35 (YFL), FI-40014 University of Jyväskylä, Finland

---

Abstract

Grain is a data analysis system developed to be used with the novel Total Data Readout data acquisition system. In Total Data Readout all the electronics channels are read out asynchronously in singles mode and each data item is timestamped. Event building and analysis has to be done entirely in the software post-processing the data stream. A flexible and efficient event parser and the accompanying software system have been written entirely in Java. The design and implementation of the software are discussed along with experiences gained in running real-life experiments.

Key words: Data Analysis, Total Data Readout, Recoil Decay Tagging, Java
PACS: 29.85.+c, 07.05.Kf, 07.05.Rm

---

1 Introduction

Nuclear physics experiments are usually instrumented using conventional common dead time data acquisition systems which are triggered by an event in a pre-defined detector. In decay spectroscopy and Recoil Decay Tagging (RDT) [1–3] experiments (in which decay spectroscopy is combined with in-beam spectroscopy) these systems inherently suffer from dead time losses since rather wide common gates have to be used in order to collect all the required information. These problems grow worse if either the focal plane count rate or the common gate width is increased. The former condition often arises from the fact that the reaction channel under study may form only a minor fraction of the total counting rate of the implantation detector. In the latter case one is usually either studying isomeric decays with half-lives of the order of

---

Email address: `panu.rahkila@phys.jyu.fi` (P. Rahkila).

Preprint submitted to Elsevier *28 July 2008*

tens of microseconds or using them as a “tag” in Recoil Isomer Tagging (RIT) experiments. To overcome these problems a novel Total Data Readout (TDR) method [4] was developed by the GREAT collaboration as part of a project to build a highly sensitive tagging spectrometer [5] .

TDR is a triggerless data acquisition system in which all the electronics channels operate individually in free running singles mode. All the information is read out asynchronously by the front-end electronics consisting of gated analog-to-digital converters (ADCs) and bit-pattern registers. Data items are time-stamped with 10ns precision using a 100 MHz clock signal, which is distributed throughout the whole system. The data are subsequently ordered within the TDR DAQ in a collate and merge software layer, after which the data forms a single time-ordered stream ready to be processed by the analysis programs. Unlike the data emerging from a conventional data acquisition system, the data from the TDR collate and merge layer is not structured or filtered in any way, apart from the time ordering. Temporal and spatial correlations required to form events out of the raw data stream and the filtering to remove unwanted or irrelevant data have to be done entirely in the software which processes the data stream.

Grain was developed to provide a complete, self-contained, cross platform data analysis system which could be used to analyse the raw TDR data stream. The core of the system lies in three functions: in the ability to form physically meaningful events from the data stream, in providing a relatively simple way for the experimentalist to run a physics based ”sorting code” to extract the results for each experiment from these events and in allowing the user to visualise and analyse the data using histograms and n-tuples through a simple graphical user interface. A schematic of the Grain functionality is shown in figure 1. The main purpose of the software is to provide a tool for the online analysis at the RITU separator at the Accelerator Laboratory of the University of Jyväskylä (JYFL), where the TDR system along with the GREAT spectrometer are currently located, and to facilitate the subsequent offline analysis of the experimental data. The TDR DAQ currently consists of 480 independent spectroscopy channels each capable of running at 10kHz. The analysis system should thus be able to process data up to the theoretical maximum counting rate of 4.8MHz (corresponding to about 38MB/s data rate as 64bit data words are used) over the whole duration of an experiment, which can span up to a month of runtime. The GREAT TDR system also includes an event builder software, TDREB [6]. In normal online running conditions TDREB is used to filter data before it is passed on to Grain for online analysis and to the tape server for storage. Grain is used in offline analysis as a completely stand-alone system, regardless whether data under analysis has been filtered by the TDREB or not, and can be used online as stand-alone solution if required.

Grain has been implemented entirely in Java. The portability, object-oriented

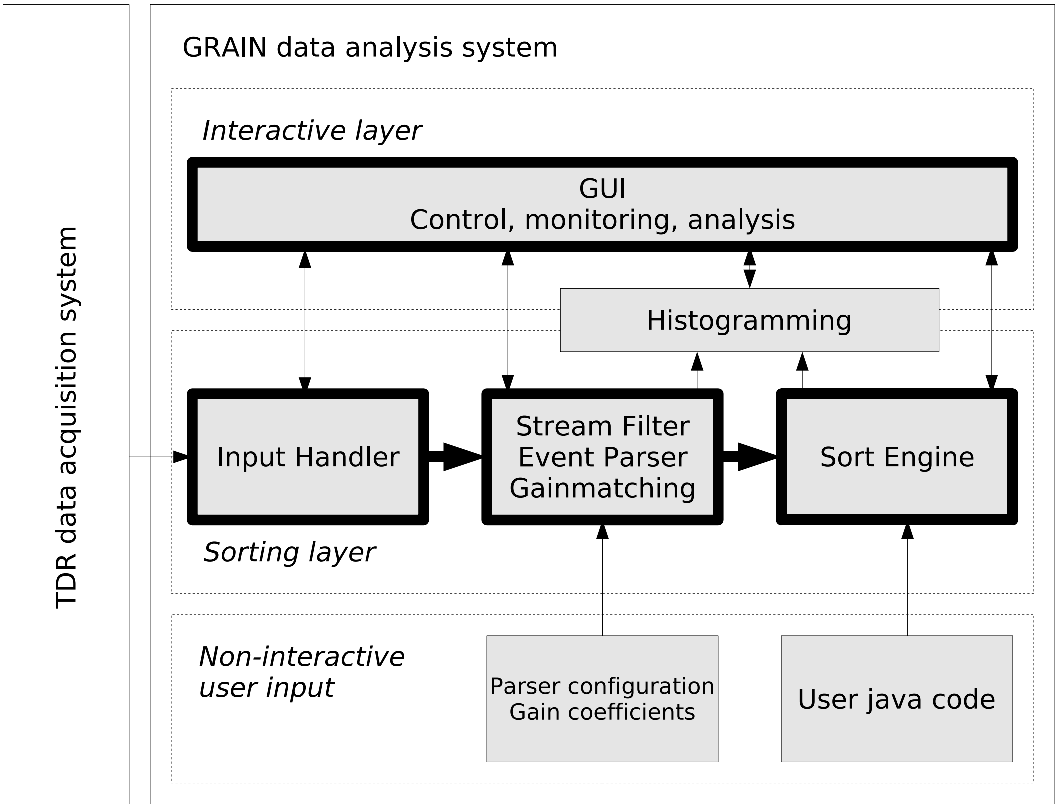


Fig. 1. Schematic of the data processing in Grain. Each data processing sub-task (thick boxes) runs on a separate thread which are interconnected by FIFOs (thick arrows) and supervised by the master thread running the GUI.

programming language and the incorporated, extensive user interface and networking libraries were the main motives behind the decision. Java has had a reputation of being too slow for calculation intensive tasks, such as data analysis, but in the recent years the arrival of just-in-time (JIT) compilers have lifted the performance to the same level as native, compiled languages (see e.g. [7]). Previous reports on the usage of Java in similar tasks [8,9] were also found to be mostly positive. The Grain executable is available for download at the development web page [10].

2 Stream filtering and event parsing

2.1 Stream filtering

Prior to building the events the TDR data stream must be filtered against unwanted data, which usually consists of vetoed and piled up signals. In traditional systems the vetoing and pile-up detection was incorporated into the front-end electronics and the data acquisition system would normally never see these data. In the TDR system the data analysis software is required to perform these tasks, though the TDR ADCs have a limited hardware-veto capability. For example, in the current JYFL TDR setup events from the Compton suppression shields of the target array are read out as bit-pattern data. Thus, the data from the target array germanium detectors and their BGO shields must be correlated pairwise in software in order to perform the suppression.

Pile-up rejection is based on the TDR ADCs capability to detect gates arriving at the ADC during the processing of the previous gate. These data are included in the stream as separate special data items and thus each channel needs to be self-correlated in time to find the piled-up data. Vetoed or pile-up data can be either discarded or marked and included in the events.

2.2 Event parsing

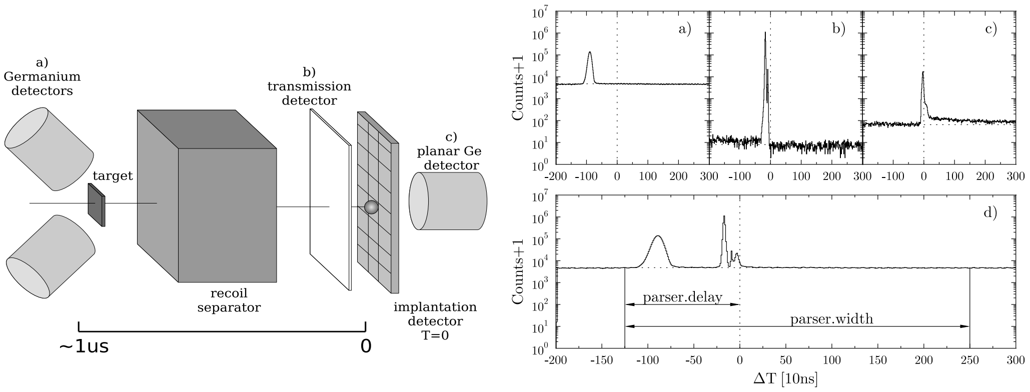


Fig. 2. Schematic illustration of a typical RDT setup and the time structure of a typical RDT experiment data stream with respect to any signal in the implantation detector. Panels a),b) and c) represent the typical response of some individual detector groups. Panel d) shows the summed structure and the main event builder timing parameters.

Two types of event parsers have been developed so far. Decay spectroscopy and tagging experiments require a parser which constructs events in which the trigger is any signal from the implantation detector. Stand-alone in-beam experiments require a trigger based on the multiplicity of hits in the detector array. In both cases time domain correlations were selected as the first stage of the event builder strategy. This was mainly done in order to maximise the input capacity of the system as the conditions used in the first stage of event parsing require only a single dynamic parameter, the time stamp, and a static definition of which data acquisition channels constitute the triggering detector group.

The decay/RDT event parser is almost entirely based on the time structure of the stream. A typical time structure of the stream, with respect to any signal in the implantation detector, taken from a tagging experiment at RITU is presented in figure 2. The individual components forming the structure can be roughly divided into three groups depending on the placement and role of the detector groups: a) preceding, b) prompt and c) delayed events. Typical examples of these are presented in the upper panels of figure 2. In panel a) the time spectra of the germanium array at the target position is shown. Flight time of the recoils through the separator is $\sim 1 \mu$s. Panel b) shows the timing of the transmission detector (a multiwire proportional counter) placed 240 mm upstream from the implantation detector. Panel c) shows time structure of events in the planar germanium detector placed next to the implantation detector exhibiting prompt and delayed components.

In decay or RDT experiments events can be simply defined at the first stage as a time slice of the stream, which is triggered by any datum from a predefined group of ADC channels. As the data is buffered in time order, it is possible to easily extend the slice to cover also data in the past and in the future with respect to the triggering data. The parameters needed to construct the slice are the address of the triggering channel, offset of the slice (delay) and the extent of the slice (width). By varying these parameters the parser can be configured for different types of requirements of RDT, RIT or decay spectroscopy experiments.

Pure in-beam experiments usually use a hit-multiplicity trigger where a certain number of coincident hits is required in a defined time window. In the case of TDR the event parsing is rather straight forward. As the data is already time ordered and filtered, and can be easily buffered in memory, one can simply count the number of hits over a given period after each individual hit. The input parameters required are the width of the coincidence window, the set of channels over which the multiplicity is calculated and the minimum required multiplicity. Figure 3 shows a typical time structure of the TDR data stream in a stand-alone γ-ray experiment.

Both event parsers have been implemented around a ring-buffer, which holds the data objects. Access to the data preceding and following a certain data item can be done by iterations in the buffer. On insertion and removal the data item is checked whether it is a triggering item, a vetoing item or a piling-up item. In any of these cases corresponding data is searched in the buffer and flagged if found. All the data is checked on removal whether it is flagged as triggered and if so, dispatched to the next level of event parser. Data is also gainmatched at this stage using user provided gainmatching coefficients.

Once the group of data forming an event has been identified, the internal event structure needs to be assigned. At the second stage of event building the data

Fig. 3. Time structure of the TDR stream from a stand-alone γ-ray experiment. Data is histogrammed if more than one hit is registered in 1μs window. In normal running conditions a 70 ns window would be used, as indicated in the figure. Oscillations in the background are caused by the structure of the beam produced by the JYFL K130 cyclotron.

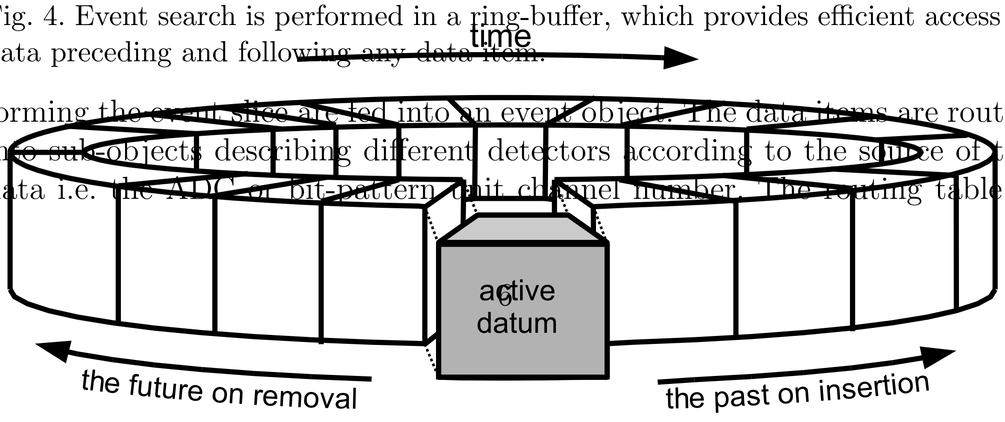


Fig. 4. Event search is performed in a ring-buffer, which provides efficient access to data preceding and following any data item.

forming the event slice are fed into an event object. The data items are routed into sub-objects describing different detectors according to the source of the data i.e. the ADC or bit-pattern unit channel number. The routing table is

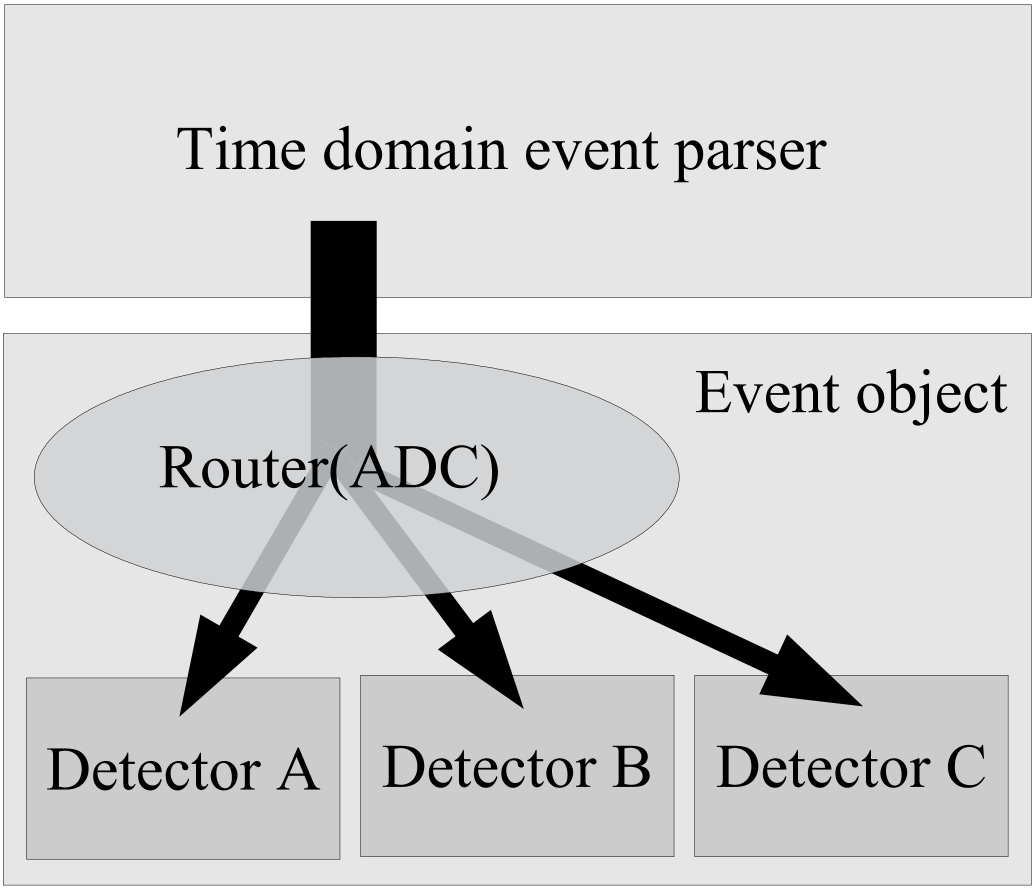


Fig. 5. Data is routed to the detector objects within the event object based on the origin of the data.

defined in the sub-detector objects at the implementation time. A schematic drawing of the routing is shown in figure 5. Several different event types have been predefined and users select the type which is appropriate for their analysis from the user interface.

3 System Implementation

The general design of the analysis system is shown in figure 1. In order to benefit from modern computer hardware with multiple processor cores available on most machines, the data processing system was designed to be multithreaded. Each individual main subsystem thread is indicated in the figure with a thick border. Users interact with the graphical user interface (GUI) running the master thread and providing interactive analysis functions as well as serving as the control thread for data sorting jobs. The sort layer consists of an input handler thread reading raw TDR data from a disk, tape or network source, an event builder thread which filters and gainmatches the data and constructs the events and a sort engine thread which uses the user code to extract the relevant data from the events. The data is relayed in the sorting layer from thread to thread using first-in, first-out buffers (FIFOs) which are indicated in the figure by thick arrows. The GUI thread starts the worker threads at the beginning of each data sorting job.

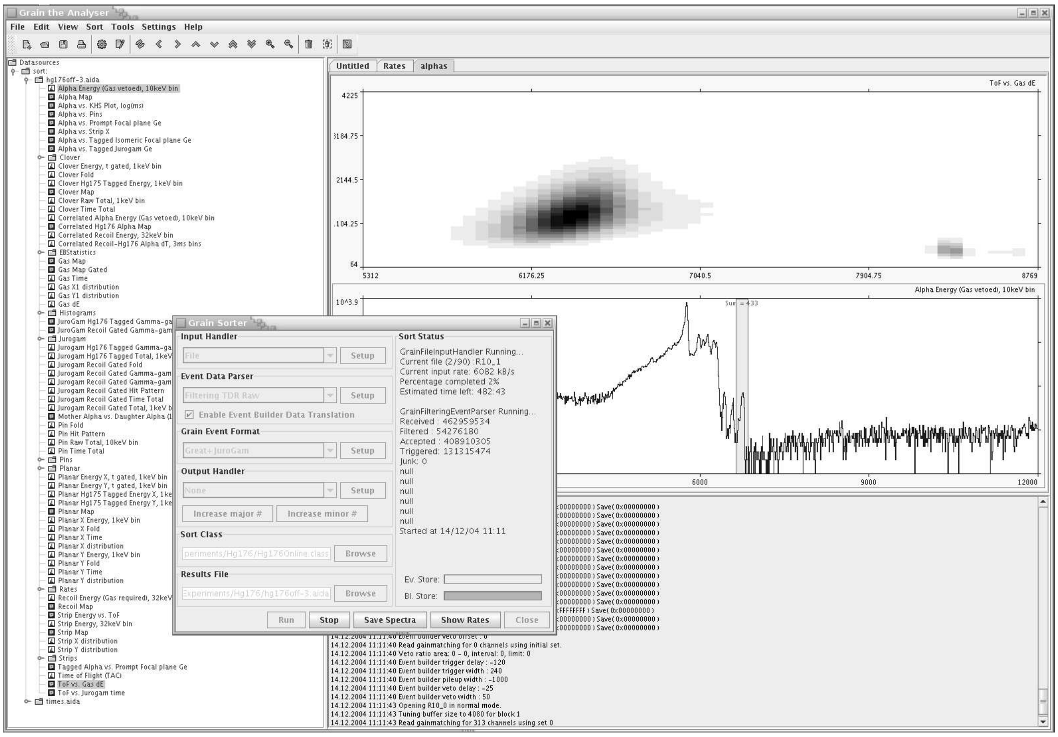
3.1 Sort Engine

The sort engine uses the Java dynamical class loading capability. Grain provides abstract (skeleton) sorter classes which the user needs to implement and which provide access to the event data. Users can thus write their own data reduction routines in Java using all the features of the language as long as this inheritance relationship is fulfilled. Compiled classes can be loaded into the Java Virtual Machine (JVM) dynamically at runtime. Histogramming and other basic analysis services are provided via JAIDA [11], the Java implementation of the AIDA (Abstract Interfaces for Data Analysis) definition [12]. A new binner had to be added to the JAIDA histogrammer since rather large multidimensional histograms are required in nuclear physics analysis. The histograms and n-tuples created in the sort engine are available through the GUI at runtime.

3.2 Correlation Framework

During the last decade the RDT technique [1–3] has been widely used in the studies of the structure of neutron deficient nuclei and super-heavy nuclei (see e.g. review articles [13,14]). In RDT the identification is based on spatial and temporal correlations of the recoiling reaction product and the subsequent, often discrete, decay events. Similar correlation tasks are used also in pure decay spectroscopy.

A large amount of information per event needs to be stored in a concerted manner often for several hours and over several event generations in order to perform these correlations. The Grain correlation framework is based on the discrete position sensitivity provided by the double-sided silicon strip detectors used in GREAT and the fact that all the event information is already encapsulated in the event object. The framework consists of a container object which provides a time ordered, time constrained stack of event objects per implantation detector pixel and routines to insert an event into the container and to retrieve the history of any given pixel based on the current event. The framework is presented to the user as a class library which is available to be used in the sort code. This framework simplifies correlation analysis



Input Handler: File

Event Data Parser: Filtering TDR Raw

☑ Enable Event Builder Data Translation

Grain Event Format: Great+JuroGam

Output Handler: None

Sort Class: periments/Hg176/Hg176Online.class

Results File: Experiments/Hg176/hg176off-3.aida

Sort Status

```
GrainFileInputHandler Running...
Current file (2/90) :R10_1
Current input rate: 6082 kB/s
Percentage completed 2%
Estimated time left: 482:43

GrainFilteringEventParser Running...
Received : 462959534
Filtered : 54276180
Accepted : 408910305
Triggered: 131315474
Junk: 0
null
null
null
null
null
null
Started at 14/12/04 11:11
```

Ev. Store:

Bl. Store:

```
00000000 ) Save( 0x00000000 )
00000000 ) Save( 0x00000000 )
00000000 ) Save( 0x00000000 )
00000000 ) Save( 0x00000000 )
00000000 ) Save( 0x00000000 )
00000000 ) Save( 0x00000000 )
00000000 ) Save( 0x00000000 )
00000000 ) Save( 0x00000000 )
00000000 ) Save( 0x00000000 )
00000000 ) Save( 0x00000000 )
00000000 ) Save( 0x00000000 )
00000000 ) Save( 0x00000000 )
FFFFFFFF ) Save( 0x00000000 )
00000000 ) Save( 0x00000000 )
00000000 ) Save( 0x00000000 )
14.12.2004 11:11:40 Read gainmatching for 0 channels using initial set.
14.12.2004 11:11:40 Veto ratio area: 0 - 0, interval: 0, limit: 0
14.12.2004 11:11:40 Event builder trigger delay : -120
14.12.2004 11:11:40 Event builder trigger width : 240
14.12.2004 11:11:40 Event builder pileup width : -1000
14.12.2004 11:11:40 Event builder veto delay : -25
14.12.2004 11:11:40 Event builder veto width : 50
14.12.2004 11:11:43 Opening R10_0 in normal mode.
14.12.2004 11:11:43 Tuning buffer size to 4080 for block 1
14.12.2004 11:11:43 Read gainmatching for 313 channels using set 0
```

Fig. 6. Screenshot of the Grain GUI. Main window displaying two- and one-dimensional histograms, histogram tree and logger window along with the sort control window are shown.

greatly as the user does not need to implement event accounting and memory management and they are handled in a consistent manner for all the users.

3.3 User Interface and Analysis Functions

The Grain graphical user interface has been implemented using the standard Java Swing toolkit and Java2D graphics (see fig. 6). A standard GUI design, with menus and toolbars, was selected as it is already familiar to most users. Histograms can be browsed with a tree widget and displayed on the main panel, several at a time if required. Information about the progress of the analysis jobs and the results for the interactive analysis are displayed in the logger window. Standard zooming and scrolling functions are provided for both one- and two-dimensional histograms. Peak-area integration and fitting of gaussian peak-shapes and exponential decay-curves are provided for one-dimensional histograms. Two-dimensional histograms (matrices) can be sliced either on the standard GUI or on a separate widget geared towards coincidence analysis. N-tuples can also be used in the interactive analysis through AIDA evaluators and filters. Histograms can be exported to ASCII and Radware [15] formats. The ASCII format can also be imported along with ROOT histograms through the hep.io.root package [16]. The spectrum view can be printed using the printing system provided by the operating system or exported to a variety of formats. The AIDA XML data format used by Grain through JAIDA libraries to store histograms and n-tuples is an open standard. Several other AIDA-compliant tools [9,17–19] can be used to read, view and analyse the files instead of Grain if so required.

4 Performance and Usage

4.1 Performance

The data sorting performance of the whole analysis system has been analysed in two ways. First, only simple through-put tests were run on a modern computer with a dual-core AMD processor running 64-bit Linux operating system and 64-bit Java version 1.6 from Sun Microsystems. Later, the performance of different parts of the data sorting chain have been analysed using the Netbeans Java Profiler [20].

To demonstrate the performance of the sorting a typical RDT experiment was selected as a test case. Actual data from an experiment using a heavy ion fusion evaporation reaction ^{36}Ar + ^{144}Sm → ^{180}Hg* to produce light Hg isotopes was used. In optimum operating conditions the total counting rate of the detectors was about 400 kHz, mainly originating from the target array germanium detectors, corresponding to a data rate of about 3.2 MB/s from the TDR. Data was written to disk without any TDREB prefiltering, and later analysed offline. The throughput was derived from the time it took to analyse a 10GB portion of the data using different triggering schemes. The results are shown in Table 1. It should be noted that the RDT triggering would be used to extract data from reaction channels produced with very low cross sections whereas the gamma-multiplicity triggering would be used for different reaction channels with higher cross sections, making comparison between the trigger types somewhat meaningless. In early experiments the histogramming of raw data was noticed to have a serious impact on the sorting performance on-line. This is clearly reflected in the current results and is likely to be caused by the high memory bandwidth required to constantly update the histograms. A buffering histogramming subsystem is used to alleviate the problem somewhat. Online histogramming of the raw data is also performed in the TDR DAQ, so it can be safely turned off if the performance degradation is too high.

The current implementation of the system can effectively use two processor cores per sort, while in uniprocessor systems the threading model causes higher overheads and the performance is degraded. An upgrade to allow the use of several parallel sorting pipelines to utilise new multicore chips is in planning.

Apart from the threading issues the performance scales rather linearly with the integer operation performance of the processor core in question. As especially the correlation analysis can be quite memory-intensive, high bandwith memory architecture and large CPU cache can result in substantial increases in the performance.

As can be seen, the throughput without raw histogramming is over an order of magnitude higher than a typical RDT experiment currently requires and close to that for stand-alone experiments. In decay experiments data rates are always much lower as the target area detectors are not used. In the case the input rate would exceed the rate Grain can process data real-time, a safety feature which starts skipping input data buffers is implemented in the input handler receiving data over the network in the online configuration. The complete data stream can thus still be written to storage by the TDR acquisition system at the expense of loss of statistics in the online analysis.

Table 1
Maximum throughput of the Grain sorter in different configurations. See text for details.

Trigger	with raw histogramming	w.o. raw histogramming
RDT	19 MB/s	44 MB/s
$\gamma\gamma$	13 MB/s	22 MB/s
$\gamma\gamma\gamma$	17 MB/s	28 MB/s

The event parser has been found to be the bottleneck in the sorting performance by using the Java profiler. About 65% of the execution time is spent in the event parser thread, out of which about a half is spent in the actual event search in the ring-buffer. This bottleneck is partly alleviated by the multithreading as the parser utilises a single processor core and the other parts of the system run in the other available cores.

4.2 Usage

Grain has been used as an on-line analysis tool in over 50 experiments since 2002, catering for very different experiments ranging from decay spectroscopy of very heavy elements [21] to RDT studies in the A∼100 region [22]and the development of the novel β-tagging technique [23]. In vast majority of cases Grain has also been the main tool in offline analysis. In total 23 peer-reviewed papers using Grain were published 2004-2007.

5 Conclusions

Analysis of the triggerless, TDR generated data has been implemented in a flexible, efficient manner. The use of Java language and platform has been a major contributor to the success of the system. Platform independence has granted simple installation and operation on the three current major personal computer operation systems, making it easy for users to deploy the software where required. Java language and the use of object oriented techniques has not only simplified development of the system itself, but has simplified the users task of sort code writing, especially when complicated correlation schemes have to be used. The amount and variety of users and experiments utilising the system as well as the amount of publications for which the data analysis has been mostly conducted using Grain also demonstrates the success of the system.

Acknowledgments

I would like to thank all the people who have used the software and contributed through bug reports and suggestions. This work has been supported by the EU 5th Framework Programme "Improving Human Potential - Access to Research Infrastructure". Contract No. HPRI-CT-1999-00044 and the EU 6th Framework programme "Integrating Infrastructure Initiative - Transnational Access", Contract Number: 506065 (EURONS) and by the Academy of Finland under the Finnish Centre of Excellence Programmes 2000-2005 (Nuclear and Condensed Matter Physics Programme at JYFL) and 2006-2011 (Nuclear and Accelerator Based Physics Programme at JYFL).

References

[1] K. -H. Schmidt, et al. Phys. Lett. 168B (1986) 39.

[2] R. S. Simon, et al., Z.Phys. A325 (1986) 197.

[3] E. S. Paul, et al., Phys.Rev. C51 (1995) 78.

[4] I. Lazarus, et al., IEEE Trans. Nucl. Sci. NS-48 (2001) 567.

[5] R. D. Page, et al., Nucl. Instrum. Methods Phys. Res. B204 (2003) 634.

[6] Great TDR software, http://ns.ph.liv.ac.uk/tdr_software.html

[7] J. M. Bull, L. A. Smith, L. Pottage, R. Freeman, Benchmarking java against c and fortran for scientific applications, in: JGI '01: Proceedings of the 2001 joint

ACM-ISCOPE conference on Java Grande, ACM, New York, NY, USA, 2001, pp. 97–105.

[8] K. B. Swartz, D. W. Visser, J. M. Baris, Nucl. Instrum. Methods Phys. Res. A 463 (2001) 354.

[9] Jas3 software, http://jas.freehep.org/jas3

[10] Grain software, http://trac.cc.jyu.fi/projects/grain

[11] Jaida project, http://java.freehep.org/jaida

[12] Aida project, http://aida.freehep.org

[13] R. Julin, K. Helariutta, M. Muikku, J.Phys.(London) G27 (2001) R109.

[14] R. D. Herzberg, J.Phys.(London) G30 (2004) R123.

[15] Radware software, http://radware.phy.ornl.gov

[16] Rootio package, http://java.freehep.org/freehep-rootio

[17] Openscientist software, http://openscientist.lal.in2p3.fr/

[18] Physicist interface (pi) project, http://lcg-pi.web.cern.ch/lcg-pi/

[19] Paida software, http://paida.sourceforge.net/

[20] Netbeans profiler, http://www.netbeans.org/products/profiler/

[21] R. D. Herzberg, et al., Nature(London) 442 (2006) 896.

[22] M. Sandzelius, et al., Phys.Rev.Lett. 99 (2007) 022501.

[23] A. N. Steer, et al., Nucl.Instrum.Methods Phys.Res. A565 (2006) 630.