ANALYSIS OF THREE–DIMENSIONAL NUCLEAR DATA

ON A TRANSPUTER–BASED MULTIPROCESSOR SYSTEM

R.J.R. Back[1], *M. Aspns*[1], *J. Granlund*[1], *J. Hattula*[2],
R. Julin[2], *A. Lampinen*[2], *T. Lnnroth*[3] *and P. Waxlax*[1]

Abstract

A prototype for administration and analysis of huge amounts of data, mainly nuclear multiple coincidences, is described. The prototype is implemented on a 100−unit transputer-based multiprocessor system, *Hathi-2*. First, the performance of the software is tested on artificially generated three–dimensional data. Then a set of real data is analyzad to estimate the rate of sort that can be performed. The feasibility of using such a system to analyse large nuclear-physics data spaces is discussed.

Contents: Page:

[1] *Department of Computer Science, bo Akademi, Lemminkinengatan 14, 20520 Turku, Finland*
[2] *Department of Physics, University of Jyvskyl, Seminaarinkatu 15, 40100 Jyvskyl, Finland*
[3] *Department of Physics, bo Akademi, Porthansgatan 3, 20500 Turku, Finland*

1. Introduction: Some Basic Nuclear Physics

1.1 Properties of a $\gamma-$ray Coincidence Experiment

A nucleus that is in an excited state, as a result of a collisional excitation for example, de-excites via emission of charged particles (p), neutrons (n) and gamma-rays ($\gamma-$rays). The product nucleus, reached after the $p,n-$emission, passes via it's excited states until it eventually reaches the ground state. Since the nucleus obeys quantum mechanics it's level spectrum is discrete, and hence is also the finally emitted $\gamma-$ray radiation discrete. The experimental nuclear physicist observes, that is detects via a detector and collects, these quanta. Since the probe, or target, contains a large number of nuclides (about 10^{22}) the emitted radiation from the various levels arrives at the detector at completely random rate. In addition the intensity af a single transition also varies greatly. This intensity difference may be as large as 1-1000.

The detector transforms the energy of the impinging quantum to an electric pulse or analog signal, which is brought to an ADC (acronym for Analog–to–Digital Converter), which converts the analog pulse to a binary address. Thus the original transition energy, which contains the physical information, is given a digital address. If the system were ideal a certain transition would correspond to a single address, as shown in Fig. 1a. In reality it is not, because there are different physical mechanisms present for the energy-to-pulse transformation, and instead of a single address the single transition gives a distribution, exemplified by the monoenergetic $\gamma-$ray of ^{137}Cs in Fig. 1b. The spectrum of Fig. 1 is a unidimensional $(1-D)$ $\gamma-$ray distribution, where only the peaks contain the physical information, whereas the tails are unwanted noise. As a negative feature it can be anticipated that the areas of the peak and the tail are comparable in size. In addition the peak has a width which is about 5-10 channels, or addresses, wide at the root. This tail is what constitutes the 'background', to be discussed further below.

Figure 1. a) Ideal response of a single transition. b) The truth. The peak dominates by height with a height ratio of about 300:1, but the area of the 'tail' is of comparable magnitude. This tail is due to Compton scattered photons. Note the effect of using Compton suppression, as illustrated in the bottom part of the figure.

The goal is to construct an "energy level scheme" of the nucleus in question by analyzing the recorded data. The result of such an analysis is shown schematically in Fig. 2. It is a simple level scheme, where the labels for each arrow, or transition, are from left to right: $\gamma-$ray energy (address) E_γ, intensity I_γ and transition number N_γ. The most relevant data of an experiment is in the form of $\gamma - \gamma$ coincidences. The question of 'coincidences' giving the input for the construction can be illustrated as follows: Taking transition number 2 in Fig. 2 we see that it can be coincident with transitions numbers 1, 3, 4, 5, 6, 9, 11 and 12. Here the 'coincident with' means that only arrows *in* the two streams up-down and sideways-down with respect to '2', that is taking them along a path, can form coincidences, i.e., in this case the pairs 2-1, 2-3, ... 2-12.

Figure 2. A simple level scheme. The labels for each arrow, or transition, are from left to right: address (energy), intensity and transition number.

The $\gamma - \gamma$ coincidences constitute a two-dimensional array built from all pairs of coincident events, where one partner is a peak (true address) and the other a peak or tail event, or even from pairs both of which are tail events. In Fig. 3 a corresponding two-dimensional $(2 - D)$ spectrum is shown. The spikes of Fig. 1 are now peaks (hills) above the plane, and various combinatils of peak-tail or tail-tail constitute the background, which is now a "plane". If this matrix is projected onto one of it's axes we obtain a $(1 - D)$ spectrum like the one shown in Fig. 1, but there is a loss is resolution: from Fig. 3 it can be seen that the peaks are separated within the plane but in the projection the peaks may fall on top of each other. In addition the projection tends to accumulate background under the peaks. This background is generally fairly weak in the plane. The accumulation of background reduces the observability of weak peaks in the projection, i.e. the sensitivity is decreased.

1.2 Multidetector Arrays: NORDBALL

During the last few years multi-detector arrays have been developed in international collaboration projects. An example of such a collaboration equipment is the NORDBALL[1], which started runnning in early 1988 at the Risø laboratory of the Niels Bohr Institute in Roskilde, Denmark. The schematic construction of the NORDBALL set-up is shown in Fig. 4. The main purpose of

Figure 3. Part of a schematic $2-D$ coincidence matrix.

such an equipment is to increase the so-called coincidence efficiency of the experiments. Especially the increase of triple coincidences $(\gamma-\gamma-\gamma)$ is of importance when high sensitivity is required.

For detector arrays the data *size* becomes a parameter of importance. The NORDBALL detector $(1-D)$ data is usually in the form of 12 bit words, which are colloquially referred to as '4 kilo' spectra, since $2^{12}=4096\sim 4K$. The corresponding two-dimensional matrix of a standard $2-D$ coincidence experiment then has an address space of $4K\times 4K=16M$. If this matrix is projected onto one of it's axes, or if so-called gating is done, we obtain a $1-D$ "gated spectrum", but there is a loss of sensitivity. Gating is the classical method of coincidence analysis. A better way, which reduces the statistical and Compton noise, is to make a linear fit [2], instead of taking peak-background differences. Also here there is a loss of sensitivity, and recently a two-dimensional fit in the concidence plane has been proposed and tested [3]. All this is done to keep the data space in a manageable size.

A large efficiency allows for observation of $(3-D)$ triple coincidences. Such $\gamma-\gamma-\gamma$ triplets can in Fig. 4 be found five, namely 123, 134, 234, 125 and 146. However, when recording triples, or collecting three-dimensional spectra, the space needed increases very fast. If each dimension is $4K$, the total space needed is $4K\times 4K\times 4K=64G=6.4\cdot 10^{10}$! As for the two-dimensional case one should not project out, in this case $4K\times 4K$ matrices, on any of the sides, but save the total space information analogously to the matrix. In the $3-D$ case the interesting information (which was spikes in $(1-D)$, peaks in $(2-D)$) is clustered into "stars" in the cube. These

Figure 4. Schematic representation of NORDBALL, cf. ref. [1]. *Each detector unit consists of a central Ge detector surrounded by a BGO Compton suppression shield. There are 32 openings, 20 hexagons and 12 pentagons.*

stars, or more precisely, clusters, correspond to the peaks rising above the background plane in the two-dimensional case, see Fig. 3.

There is one intrinsic way of reducing the required space. One might note that each "triple event", formally abc, is present in six different addresses, namely as the permutations abc, acb, bac, bca, cab and cba (intensities not included). This is a six-fold degeneracy, which permits one to 'fold' the cube and retain only one sixth of it, which reduces the number of points to $\sim 10^{10}$. This suggests that the events should be sorted according to some rule before it is stored so that the space to be analysed reduces to 1/6 of the original space. The symmetry in a three-dimensional space is illustrated in Fig. 5.

In present-day experiments the data in collected as event-moded data, i.e. as address lists, and subsequently sorted into, usually, an $x-y-$matrix. This $2-D$ matrix is strightforwardly analyzed on a normal university machine, typically a VAX785 — VAX 8800, and it uses some 10 hours of CPU totally. When the third dimension is added, i.e. triples coincidences are considered, this CPU time is multiplied by a factor of 4000, or, since one in fact has some 300 spectra per dimension and the third dimension adds sensitivity, maybe a factor of 500. The CPU time is then $\sim 500 \times 10h = 5.000h \sim$250 days, or more than 1/2 a year! If instead of running on one computer, one could divide the task on, say, 100 computers, one is down to about 4 days for each of them, which a least looks reasonable. But a system of 100 VAXes is not that readily available.

Today the processing requires both automatic and manual work. The manual work consists mainly of tape mounting which is time consuming. The problem of sorting the data has been studied by Urban et al. in [4] where a method to handle the data using a VAX 780 computer and two magnetic discs with capacities of 200 and 256 MBytes and a tape station was proposed.

Figure 5. The six-fold degeneracy of a three-dimensional $\gamma - \gamma - \gamma-$cube. The points labelled '1' through '6' are equivalent in each sixth of the divided cube.

2. Principles of the $\gamma - \gamma - \gamma$ Analysis

2.1 Structure of the Data

Due to the detector properties, cf. Fig. 1b., there exists so called true clusters and a non-true data. The true clusters consists of events originating from photopeaks whereas the rest of the events are called background, and is accumulated in the space with a different distribution. To separate it from the true clusters, the surrounding area and propagation has to be studied. We are only interested in the true clusters arisen, their position and intensity. The intensity is defined as the number of events which are considered to belong to a cluster, whereas the position in this case consists of a two- or three-dimensional address which denotes the centre of the cluster in space. The clusters intensity can be between 1-1000 events. This makes it very difficult to detect a real cluster with small intensity and to preclude the detection of non-true clusters where the accumulation of events is very high.

Analysing the material is not trivial and requires both time and memory space. Special care should be taken when choosing data structures due to the large amount of data and the data properties. Although the amount of data is large it would only fill 0.15 % of a matrix with the dimensions 4096*4096*4096. It has been shown in [4] that the cube is essentially empty – out of $1 \cdot 10^{10}$ points $\sim 2 \cdot 10^8$ are filled by events, and these are clustered in such a way that a part is in 'stars' and part is in the random background. Suppose that there are 10000 stars in the cube, and that they have equal intensity for simplicity, and that the width of a star is 4 points/dimension. The $1 \cdot 10^4$ stars occupy only $6.4 \cdot 10^5$ places, so totally only each 100th point is occupied, and the physical memory to store all events is $\sim 10^8$ which is $\sim 100M$. Note that the mean number of counts in each point of a star is about 150 if one assumes an even distribution (which is not totally correct) and that the total 'brightness' is 10^4. This means that the stars, even if their intensities in reality vary, essentially are on an 'empty' background, and this is in fact the strength of multidimensional events.

Of all data received approximately 90 % of the events are background and are not useful for the final task. An ideal algorithm would distinguish relevant events from background saving time and

memory. To find such an algorithm is probably impossible because we can not say whether an event is going to be in a cluster or if it is only background until all events are stored. At that point already 90 % of all the useless events are stored. To detect a cluster in real time there has to be a number which indicate when an accumulation of events in an area will be interpreted as a cluster. To find a cluster as soon as possible and remove it from the storage of events, this number must be low. The information on cluster intensities and positions can then be stored separately. On the other hand, a low number increases the possibility to find non-true clusters and increases the manual work to be done later.

The result, e.g the clusters found, must be presented in a proper way. Because of the amount of clusters, over 10000, an enumeration of the position and intensity of the clusters is not sufficient. The sorting and analysation of the data suggests an efficient computer with access to a large random access memory. For storage of the final results and the sorted data a harddisk is needed. We have not studied the analysing algorithm nor have we made any attempt to solve the problem of distinguishing the true clusters from the non-true clusters. This must at the moment be done manually.

During a typical experiment at NORDBALL, which might take several days, usually $\geq 1 \cdot 10^8$ events are collected. The collection speed is about 1000-2500 events per second. Each event has a length of about 30 bytes, and the rate is thus about 30-75 kbytes/s. The data is normally stored on magnetic tape for subsequent analysis. If one can use a multiprocessor system it is also possible to perform operations on-line on the addresses, e.g. calibration (which is a linear transformation).

2.2 Decomposition of the Data Space

Since the data points are independent of each other, except within a cluster, the whole data set may be divided into disjoint parts, or subcubes. We have studied two different approaches for storage of the events from the experiment on a distributed-memory multiprocessor system, a list-based structure and a bit map representation. There are both advantages and disadvantages to these two solutions. The main structure of the program is the same in both approaches. The problem suggests that the space to be investigated is to be divided into subspaces which can be handled separately, thus can be administrated by different processors. Two possibilities have been considered:

The subspaces are disjoint. In this case the processors are not aware of the events on the other side of the boarder between two subspaces. When investigating an event close to the boarder, a reqest must be made to the neighbour for investigation of events that belong to the subspace of a neighbour. In the worst case, in a two dimensional space, four neighbours can be involved in such a task. On the other hand an event is only registered on one processor and redundancy can be avoided.

There is a shared area between neighbouring processors. If the radius of a cluster is approximately known, and it's within reasonable limits, the second approach can be used. The boarder area common can then be chosen to be the radius of a cluster. With this presumption a processor can analyse the surroundings of an event without consulting any other processor. This speeds up the analysation phase, although there will be some redundancy in the data storage. To minimize and simplify the communication between processors we have chosen the latter solution. The implementation of this method on a multiprocessor system is described in sections 4.2 and 4.3 below.

3. Description of a Transputer, and of Hathi-2

3.1 The Transputer

A transputer is a single $VLSI$ device with a processor, a local memory and a number of communication links for connection to other transputers. The architecture of the IMS T800 transputer is illustrated in Fig. 6.

The transputer is a family of microprocessor developed by Inmos Ltd in Bristol, UK. The name stems from the combination "*trans*istor com*puter*", which suggests that the microprocessor could be used to build powerful multiprocessor systems in the same way as computers are built from transistors, see e.g. [5] for details, and Fig. 7 for an example of an array of transputers.

Figure 6. The IMS T800 transputer architecture cf. ref. [5].

Figure 7. Four transputers organized in an array.

The T800 transputer is a 32-bit *RISC*-type microprocessor which also contains a 64-bit floating-point processor, four 20 Mbit/s *DMA* communications links, 4 Kbytes of on–chip *RAM* and a 32-bit memory interface on a single $1.5\mu m$ *CMOS* chip. The 4 Kbytes of on-chip *RAM* provide a data rate of $\leq$ 80 Mbytes/s. The processor can directly access a linear address space of $\leq$ 4 Gbytes. The 32-bit wide memory interface uses multiplexed data and address lines, and provides a data rate of up to 25 Mbytes/s. For a processor clock speed of 20 MHz, the T800 processor has a performance of about 10 MIPS or 1.5 Mflops.

3.2 The *Hathi-2* Multiprocessor System

The *Hathi-2* multiproccessor system was built in a joint project between the Department of Computer Science at bo Akademi and the Technical Research Centre in Oulu. The system consists of 100 T800 transputers, each currently with 1.25 Mb of RAM, but expandable to 4.25 Mb per processor. The *Hathi-2* system can be characterized as a modular, reconfigurable, general-purpouse multiprocessor system.

The *Hathi-2* system consists of 25 identical processor boards, illustrated in Fig. 8a. Each board contains 4 T800 transputers with up to 4.25 Mb of local memory, a C004 link crossbar switch and a T212 transputer to control the setting of the switch and perform other system service tasks. The boards in *Hathi-2* are connected to each other in a static torus connection, illustrated in Figure 8b. This two-level interconnection scheme makes it possible to change the topology of the system by software, i.e. by controlling the C004 switches. The T212 transputers are connected into a ring and form a distributed control system (see Fig. 9), that controls the crossbar switches and performs other system adminastration tasks. See ref. [6] for more details.

Figure 8. a. The board architecture of Hathi-2. b. The board connections of Hathi-2.

The *Hathi-2* system is connected to the national (and international) networks via a Sun workstation, and can be accessed independent of where the user is located (see Fig. 10). The performance of the *Hathi-2* system is about 150 MFLOPS. The total memory capacity of the system is currently 125 MBytes, but this can be easily extended to 425 MBytes. *Hathi-2* can be configured or partitioned in several ways, depending on the user's need, i.e. depending on the topology of the problem. For the standard partitionings of *Hathi-2*, see ref. [7].

Figure 9. Schematic view of the Hathi-2 control system.

Figure 10. Hathi-2 host computers and peripherals.

The system is programmed in the high-level languages Occam, Fortran or C. Experience gained from a number of application programs implemented on *Hathi-2* has showed that the most efficient use of this system is for problems where a large number of processes run (almost) independently of each others. Examples of such problems are Monte Carlo calculations, fluid dynamic calculations and multidimensional data analysis.

4. Implementation on *Hathi-2*

4.1 Representation of Data

The structure and the decomposition of the data was described in section 2. We have investigated three different methods for storing and analyzing the observations on a distributed-memory multiprocessor.

By representing the observations as a directed graph where for each point is stored the (x, y, z) coordinates and a pointer to the next observation in each dimension, the memory requirement depends only on the number of observations stored. For each observation one needs three 16-bit integers for the (x, y, z) coordinates and three 32-bit pointers, i.e., 18 bytes for each observation. Additionally, there is a need of a small fixed amount of memory for storing the pointer heads. For 10^8 observations, the total memory requirement will be about 1.8 Gbytes, which is currently an unrealistic amount of memory.

An alternative solution is the usage of a direct access bit-map, which however, can also be more memory consuming. The direct access bit-map consists in this case of a contiguous memory area divided into locations which represents the space of all possible addresses. The address of an event denotes the location in memory where the event is going to be registered. The computation can be done using arithmetics as addition, multiplication, logical or and logical and. This speeds up the investigation and registration of incoming events. Furthermore this method is not dependent on the number of events to be sorted or the data already sorted and analysed because no lists must be traversed and no pointers have to be updated. The computation and registration can be done in constant time throughout the whole experiment.

The direct access bit-map is dependent on how many events we consider to be held in one location before we are determined that a cluster has been formed in the area around that location. The size of the location can not be too small due to the possibility to find a huge amount of non true clusters, however, if it is chosen to be too large the memory requirement grows very fast and we soon face an unrealistic memory demand. A compromise must be found, where memory requirements and accuracy can both be within tolerable limits. If such compromise is found, it is possible from a small sample of data to compute and estimate the behaviour of the system in a large full scale experiment. This saves the user from surprises in a later state and precludes the possibility that all available memory is used and the experiment is not yet finished.

In this model there are two possibilities to detect a cluster. The criterion is either a high frequency of events registered in a single location or a high proportion of events in the surrounding neighbourhood. Instead of registering all events that are detected, a mechanism which use statistics and probabilities can be used to save time and memory. As opposed to the previous method where all events where registered, only some of the events will be registered. The frequency of registration depends on the number of previous registered events and a function. With this method we can minimize the number of bits we have to allocate per address. Of course the accuracy will suffer somewhat, but the overall pattern will be found.

In the solution adopted in our implementation, the observations are stored in a two-dimensional array consisting of linked lists of observations. Each element in the array consists of the (x, y) coordinates of the observations and a pointer to a list of z-values. The elements in the z-list consist of the z-coordinate, a counter indicating the number of observations in this point and a pointer to the next observation with the same (x, y) coordinates. The data structure is illustrated in Fig. 11.

With this representation, we need one pointer for each (x, y) coordinate pair. For each observation, we need to store the z value, the counter and a pointer to the next observation, giving a total of 7 bytes per observation. For 10^8 observations, the maximum memory requirement will be about 764 Mbytes. However, measurements show that the average number of observations for the points that occur in the observations is between 4 and 5, so the actual memory requirement can be reduced by a factor of 4, giving a total memory requirement of about 190 Mbytes.

Figure 11. Data structure for storing the transmitted observations.

Because of the limited amount of memory in *Hathi-2*, the implementation has been scaled down to a size of $400 \times 400 \times 4000$ points. The problem is decomposed using geometrical parallelism, where each processor is responsible for a part of the space. The data domain is divided in equally large blocks in the (x, y) plane. Each processor keeps record of the observations falling inside it's own domain, and whenever a new observation arrives, the processor inserts the observation into the appropriate z-list and checks, by searching the z-lists of the neighbouring points, if a cluster was formed. When a cluster is found, the observations belonging to this cluster are removed from the z-lists and only the positions and intensities of the cluster is stored in a separate list. New observations are also checked against this cluster list to see if the observation falls into some already detected cluster.

4.2 Processes Used

The implemented system consists of two different components, one main process called the root process, for producing, sending and gathering data and presentation of results. The other component is a node process for registering events and analysing subspaces. Both the root process and the node processes are composed of parallel processes. We here describe the different processes and their function.

The host, or root, process. The root process sends the observations via a communication link to the node processes to be analyzed. It consists of an Input/Output process, which interacts with the user and the file system, a Data generation process for generating artificial observations used for test purposes, and a Communication process which communicates with the node processes. The result of the computations, i.e the centres of the clusters and their intensity (number of events in a certain region), is presented on the screen of the host computer. The result is also saved in a file for later examination. The processes on the root processor are illustrated in Fig. 12a.

The node processes. The node process is composed of four processes. The Input process receives observations and checks whether the observation is administrated by this processor. If so, the observation is sent to the Calculation process, where it is stored in a z-list or cluster list. If the

Figure 12. Schematic illustration of processes. a. The root process. b. The node process.

observation does not belong to this processor domain, it is send to the Output process and passed on to the next node process.

Between the Input process and the Calculation process there is a need for a buffer. The buffer is a temporary storage for events belonging to the processor but not yet registered and analysed. Without the buffer the data flow would easily stagnate and processes in the network would get idle. The purpouse of this buffer is to enable a constant flow of data in the network without possible standstill in one processor.

4.3 Configuration, Communication and Storage

The present experiment was designed for maximum processor capacity, and, as stated above, since the data space is disjoint and very large it should use all processors. For this reason *Hathi-2* was partitioned in it's maximum form, i.e. as 96 processors. The task of each processor is to search a given data package for events containing the coordinates of it's own data space. If found, these events are analyzed and stored in the z-lists or cluster-lists of this processor. The remaning events are sent to the next processor and the procedure is repeated until all data is sorted. The processes used are those described above.

The network used for testing the algorithms is a ring consisting of a number of node processors and one root processor. There is no disadvantage of such a simple solution if all the events are initially coming through one link from the experiment. This first link constitutes the constraint. However, if it is possible to get the input from more than one source with different links, a better solution should be considered. In the network used the load of the links is highest on the first link, because all data have to be transmitted through the first link. Eventually, the data reaches the processor which administrates it and it will, if it's not needed by further processors, be removed from the data stream. The configuration is illustrated in Fig. 13.

The communication between the processors is unidirectional, with one link to and one link from a processor. To distinguish different kinds of data being used, a tag which indicate the type of data sent, is used. The data originates from the root processor which resides on the host computer. From there the data is send with the external links between the processors. The internal communication between processes executing in parallel on a processor, is handled by the channel construction available in the occam language.

Figure 13. Configuration used in the test runs.

4.4 File Handling

Since the task is to sort a large amount af data the file handling turns out to be of major importance. It should be possible to read data from secondary memory, in this case a hard disk, with at least the same rate as the node processes are able to analyze, otherwise the processors are not efficiently utilized.

A local problem presently is that *Hathi-2* does not have a secondary memory of it's own, but uses the hard disk of the host computer Sun3. The Sun3 is also a time-sharing unit, so efficient disk access also implies denying access for other users, which is not desirable.

The present tests were made reading data from the hard disk of the host. This means that the data on magnetic tape in JYFL format (JYFL stands for Department of Physics, University of Jyvskyl) first must be translated to a suitable format which can be read by the host process. This conversion is impossible in a real situation if high speed is needed.

5. Results

The tests described above were run for different data sets, both for artificially generated and for real data. The present documentation (see also [8]) does not contain a presentation of a final product ready to be used solving all problems of collecting, sorting and analysis of three-dimensional data. The emphasis was more on designing a prototype system and finding the main problems handling such a data stream on a multiprocessor system.

These preliminary tests indicate that lack of primary memory produces big problems. Although the processor capacity is large the programs suffer from lack of memory. However, it has been clearly shown that multiprocessor systems are *a priori* well suited for these kinds af analyses, since the data space of the problem can be partitioned into disjoint subspaces.

One of the main problems, which has not been tackled is this work, is the storage of the total data. This needs very large hard disk units, and constitutes a project on it's own, and it will not be discussed further here.

The test runs were made in two different ways:
— In the first test, data was read from the Sun3 hard disk, transmitted to the *Hathi-2* system and

sorted on the individual transputers.
— The second test consisted of sorting data on the transputer system only, having the data in memory.
In this way we could compare the capacity of the implemented system both with and without disk access.

Sorting Data from Hard disk. First, only the disk input capacity of the system was tested, without any other users on the Sun3 host. A total of 1,523,168 events were read from the disk in 731 seconds total. This corresponds to an input rate of 2084 events/second or cps (counts-per-second).

In the second test, the same amount of events was both read from the disk and sorted on the multiprocessor system. The following rates were observed: for a package size of 1000 events, 1,523,168 events were sorted in 965 seconds, and for a package size of 5000, the time was 981 seconds. Conversely, this means expressed as events/seconds the following rates: 1578 cps at 1000 package size, and 1553 cps at 5000. As is seen, there is no big difference in performance when varying the package size.

Sorting in Memory. In this test the data was first transferred to the primary memory of the transputers, and thereafter the sorting started. Here 500 events were sent to the prosessor network to be sorted 2000 times, that is, a total of 1,000,000 events. The sorting time was about 35.5 seconds which corresponds to 28,169 cps!

It was quite evident, and, in fact, expected, that the main bottleneck is at the transmission from host to transputer system. The presented figures imply that the system has a very large capacity for analyzing and storing observations, but that the input rate from the hard disk to the multiprocessor system is slow and constitutes the main bottleneck in the implemented system.

6. Conclusions

The system was first tested with artificially generated input data. A separate processor was used to produce input data with the same distribution as data generated in the experiment. However, the random number generator used to produce the input data was unable to calculate more than about 3000 observations per second, so these test could not be carried out for input rates higher than this. The program was able to process about 3000 observations per second.

As the next step, the system was tested with real data from the experiment. In this case, the bottleneck proved to be the interface to the data files on the user host processor, in this case a Sun 3/160 workstation. Observations could be read from the disk and analyzed at a maximum rate of about 1570 observations per second.

Some experiments were also carried out to measure the performance of the system without any input or output. These tests were carried out by processing the same input data a large number of times, so that data had to be input only once. The results from this test indicated that the system, provided that the input rate from the hard disk was large enough, could handle over 28,000 observations per second. This corresponds to a data rate of about 328 Kbytes/s.

The conclusion is that a processor system of the *Hathi-2* kind is well capable of analyzing triples $\gamma-$coincidence data, i.e. (x, y, z) coordinates, at collecting speed. As mentioned in section 2.1, the typical event rate at an experiment is 1000-2500 cps, and our tests show that with the current implementation on *Hathi-2* about 1500-4000 cps can be analyzed. The rate is restricted by the slow communication with the hard disk.

By using a dedicated fast hard disk with a data rate of over 328 Kbytes/s, directly connected to a transputer link, we expect to be able to analyze the total amount of data produced during an experiment, i.e. 10^8 events in a few hours.

References

1. B. Herskind, Nuclear Physics **A447**, 395 (1985)

2. T. Lnnroth and P.P. Jauho, Nuclear Instruments and Methods in Physics Research **A261**, 549 (1987)

3. D.C. Radford et al., Workshop on Nuclear Structure (Slide Report), The Niels Bohr Institute, Copenhagen (May 16-20, 1988) p. 124

4. W. Urban, W. Gast, R.M. Lieder, T. Rzaca-Urban, G. Hebbinghaus and A. Krämer-Flecken, Annual Report 1986, Institut für Kernphysik, Kernforschungsanlage Jülich, 1987, p. 128

5. Inmos Limited, *Transputer Reference Manual*, Prentice Hall (New York, 1988)

6. M. Aspns, R.J.R. Back and T.-E. Malén, *Hathi-2 Multiprocessor System*, Microprocessors and Microsystems **14**, 457-466 (1990)

7. M. Aspns and T.-E. Malén, *Hathi-2, User's Guide, version 1.0*, Reports on Computer Science & Mathematics, Ser. B, No. 6, 1989

8. J. Granlund och P. Waxlax, *Underskning av klustercentrum med ett multiprocessorsystem*, Institutionen fr informationsbehandling, bo Akademi, 1988