IceCube String Trigger Simulation Summary

Doug Cowen and Kael Hanson
University of Pennsylvania
Department of Physics & Astronomy
209 S. 33rd St., Philadelphia, PA 19104
cowen@hep.upenn.edu, kaeld@hep.upenn.edu

February 12, 2001

The proposal for data acquisiton in IceCube calls for an event model in which single hits from each OM channel are written into memory buffers as they occur without any event structure being imposed on the hits until much later in the pipeline. This model has the advantage that arbitrarily complex trigger systems can be built which examine the data stream for interesting patterns. The ceiling on the complexity of the trigger is set by its processing time. Given the number of channels in IceCube, approximately 5000, and among a noise rate in each channel of 500 Hz, a trigger which simultaneously examined the inputs over the whole detector would need to process of order 10^6 hits per second. Many interesting events, especially low-energy ones, will be localized on scales small compared to IceCube. It has been proposed, therefore, to divide the triggering among subsections of the detector. A logical division occurs at string boundaries, giving rise to the concept of the string trigger. Each proposed string contains only 60 OMs so that the data rate is reduced by two orders of magnitude.

In what follows, a description of one particular software implementation of the string trigger is presented. An emphatic note is in order. While a reasonable attempt has been made to ensure a realistic simulation of the physics involved, the primary focus of this investigation is software simulation of the string trigger for the purpose of obtaining an estimate of the computational cost of the algorithm. Detailed studies of optimal configurations of string triggers for extracting low-energy physics events in the geometries of AMANDA and IceCube are discussed elsewhere [2, 1].

1 Requirements of the String Trigger

The Monte Carlo studies have shown that the minimal requirement of a string trigger is that it must be able to find local coincidences of hits according to the generic template:

> **N** hits within a group of **M** contiguous OMs
> *and* within a time window **ΔT**.

This is demonstrated schematically in Figure 1. Any implementation must be capable of completing the required computations in *real-time*. It is not, on average, allowed to fall behind the rate at which new hits, both real and background, are entering the system.

2 A Simple Software Simulator

Simulator input was derived from the AMANDA Monte Carlo program AMASIM2. It was used to create events in a mockup detector — this a single IceCube string, 900 m long with an OM spacing of 15 m. The AMASIM2 input was generated with MUO0, a

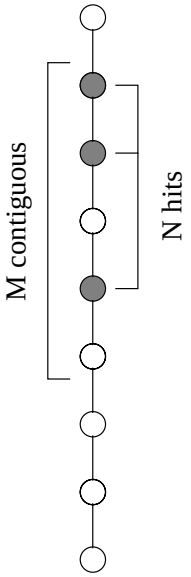


Figure 1: A schematic illustration of the topology of the string trigger.

simple program which generates muon vertices. The muons were generated in "volume mode" within 50 m of the string's center, with an E^{-1} energy spectrum. The AMASIM2 events were fed into a program which placed the hits without any event information associating them to one another into a continuous time buffer, where they were mixed with a noise background of varying levels to form the simulated data stream of an IceCube string.

Because of the continuous nature of the incoming data stream we had to simulate the passage of time in a somewhat realistic manner when mixing data and background. It was assumed in all of the simulations that the isotropic muon flux was impinging on the model detector at a rate of 100 Hz. There were 1549 muons in the input sample which resulted in buffer lengths of approximately 17 s.

In reality, the data from a string are likely to be segregated into buffers private to the front-end controller for each channel rather than feeding into one large list. The string trigger algorithm requires as input a globally time-ordered sequence of hits. This results in the overhead of merging the sixty buffers, a non-negligible component of the total processing. The cost of this preprocessing will be evaluated in a separate program since it is not an integral part of the string trigger. It has not been included in the time estimates of this report. Nevertheless, it must eventually be factored into the proposed readout scheme.

The software was written in a high-level language, C++, to run on a general-purpose digital computer. This choice was motivated by the urgency of an answer, the availability of a general-purpose digital computing device (two actually, a 700 MHz Pentium-III and a 400 MHz Pentium-II), and the existence of the C++ *Standard Template Library* which greatly simplified list management. One proposal is indeed to implement the trigger on a crate-resident computer. This simulation probably gives an upper bound on the processing time since the alternatives, such as C or machine coding on an embedded processor or further still the use of FPGAs, would likely incur much lower computational costs. As will be demonstrated by the results of this simulation, even the lower speed software trigger is able to cope with expected data rates without any problem.

The algorithm that was developed attacks the problem in a very straightforward way: it searches first for $\mathbf{N}$ hits in a time window $\mathbf{\Delta T}$ over the whole string. This is accomplished by maintaining two iterators (the C++ analogs of pointers) into the list of hits that mark the boundaries of a sliding window which encompasses all hits within a specified time interval, here 1 µs. (Consult Figure 2 for an illustrative example.) The search pauses when the number of hits within the window exceeds threshold $\mathbf{N}$. Now it is necessary to determine whether the hits are clustered in space. This again uses a sliding window concept. Hits identified by the time cluster finder are projected onto their positions on the string and a search is done in each of $N_{\mathrm{str}} - \mathbf{M} + 1$ overlapping boxes of width $\mathbf{M}$ to see whether any exceed $\mathbf{N}$ (Figure 3). If so, then a string trigger is generated. Trigger generation causes the rightmost iterator to seek forward to expand the time window to 10 µs, to simulate the effect of a hardware trigger gate. The time cluster then restarts beyond the right edge of this gate.

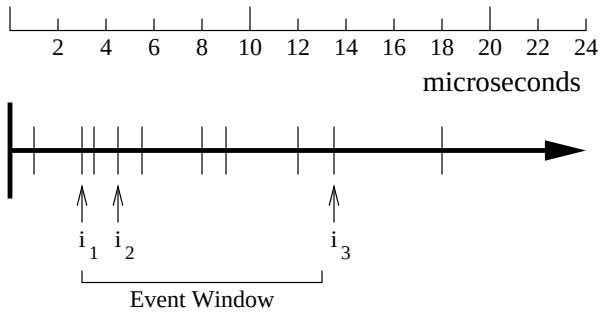


Figure 2: Example of time clustering algorithm. Two iterators search over events, pictured here as ticks along a timeline. The right iterator moves forward until $\mathbf{\Delta T}$ between the two is $> 1\,\mu$s. If the number of ticks in the window $[i_1, i_2) > \mathbf{M}$ then the algorithm branches to a spatial search. Otherwise, the left iterator moves forward one tick. Here iterators i_1 and i_2 at the position shown have found a cluster starting at $t = 3\,\mu$s. Assuming the hits are clustered in space, the right iterator at i_2 moves until $\mathbf{\Delta T}$ is $> 10\,\mu$s at position i_3. The iterator at i_1 is then brought to this position and the search begins again.

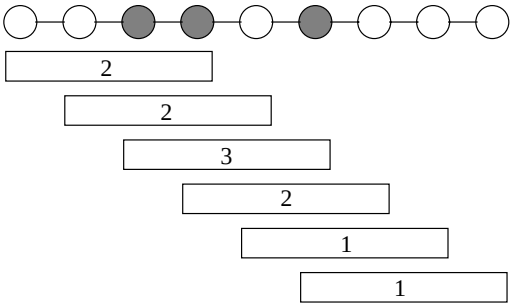


Figure 3: Diagram of space cluster finder on a string. The boxes represent sliding windows which accumulate counts of hits within them. Here for example, the third box would satisfy a trigger of (3:4) (three hits within four contiguous spaces) but no others would.

It may be necessary to bypass the space cluster search for a particular class of events. A vertical or near vertical muon can produce sparse hits along a nearby string (see, for an example, Figure 4). In these cases the event fails the space cluster portion of the search and is rejected by the string trigger, perhaps other string triggers and global triggers as well since there is considerable distance between strings. These events can be salvaged with minimal impact on noise rejection by inventing another multiplicity threshold in the time cluster algorithm. Above this threshold, time clusters are accepted without performing a space cluster search. This also has the computational benefit of obviating the spatial cluster code for a large fraction of the events which produce many hits on a string (Figure 5).

It should be noted that this algorithm requires very little of the host processor; simple integer operations are all that are needed making it feasible to implement on embedded processors or FPGAs, &c.

3 Results

The results of the simulation are shown in Table 1. Two "runs" were performed using the same Monte Carlo data but with differing levels of background: (1) 500 Hz intrinsic PMT noise rate, and (2) 1 kHz intrinsic PMT noise rate. Each run was processed with a variety of trigger configurations, hereafter referred to as $\mathbf{N} : \mathbf{M} : \mathbf{N'}$ which is read as $\geq \mathbf{N}$ hits within $\mathbf{M}$ contiguous OMs *or* $\geq \mathbf{N'}$ hits on string. These must be clustered within a time interval of $1\,\mu$s.

	500Hz Background			1kHz Background		
Trigger	T_{700}	T_{400}	R	T_{700}	T_{400}	R
2 : 5 : 5	190	390	130	410	770	480
3 : 8 : 5	130	230	0.42	250	340	4.3
X : X : 6	120	170	0	240	330	0

Table 1: Processing times and efficiencies/rejection powers of the various software trigger simulations. The numbers are grouped according to the two levels of background simulated. T_{700} and T_{400} are processing times, in milliseconds, on a 700 MHz P-III and a 400 MHz P-II, respectively. R is background rate admitted by trigger, in Hz. The trigger efficiency was also estimated by these programs; however, a fuller simulation with more strings is required to calculate the signal efficiency for these various string triggers.

As is clearly demonstrated by this simple exercise, the string trigger can be implemented at a minimal cost of execution: software triggering on approximately 17 s of data requires less than a second of CPU process time [1] even assuming a high background noise rate of 1 kHz per OM. In the worst case of the $2:5:5$ trigger operating over the noisy 1kHz background, the processing took less than 2.5% of the total time available in order to keep up with the event rate. It seems that dedicating 1 single board computer (SBC) per string solely for the purpose of local triggering is more than adequate. One possibility is implementing much more complex triggers using the available resources. Another is to decrease the processing power per string either by feeding several strings into one SBC or exploring other solutions such as cheaper embedded processors or FPGAs to do the job.

References

[1] M. Leuthold *IceCube Performance Studies.* Available at Matthias Leuthold's web page `http://www-zeuthen.desy.de/~leuthold`

[2] T. Feser and L. Köpke *Study of Multiplicity and String Trigger.* AMANDA-IR/2000903.

[1] These numbers were obtained by enclosing the trigger simulation in calls to the UNIX system call `clock()` which measures only the time the processor spent in the thread executing the software trigger and not the total time spent multitasking.

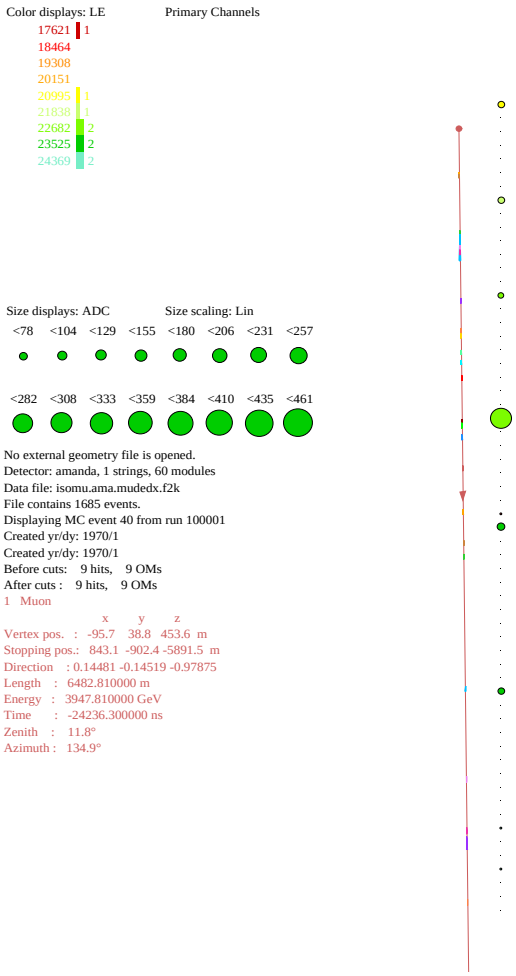


Figure 4: Monte Carlo event depicting a muon which passes nearby a string but is far enough away such that it fails to produce a localized cluster of hits. This class of event topologies advocate using an additional threshold level beyond which the spatial clustering is not checked. The muon shown here is about 50 m distant from the string: for the proposed inter-string spacing of 125 m in IceCube, this muon is unlikely to trigger another string.

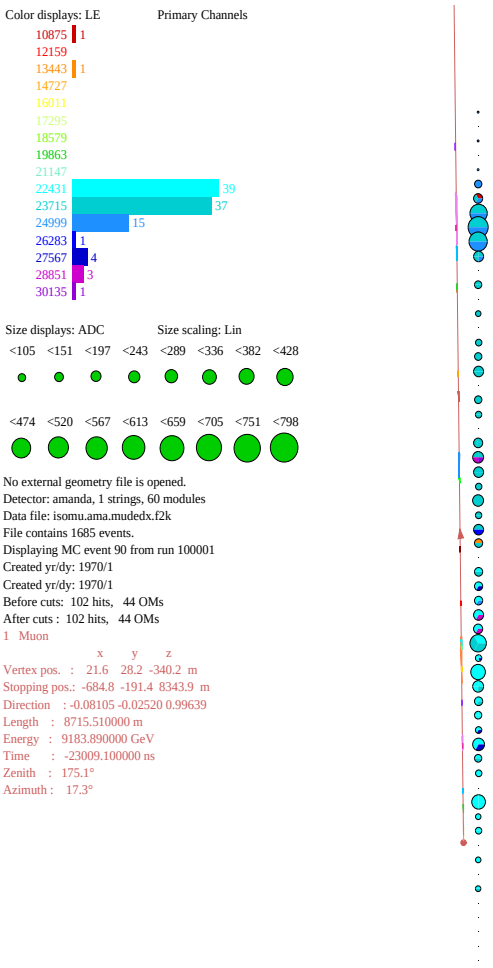


Figure 5: Monte Carlo event demonstrating situation where secondary threshold can be used to eliminate the need to run a spatial cluster search.