Connected-Component Labeling Using HLS for High-Energy Particle Physics Instruments

Nick Song
Dept. of Computer Science and Engineering
Washington University
St. Louis, Missouri, USA
qinzhounick@wustl.edu

Marion Sudvarg
Dept. of Physics
Washington University
St. Louis, Missouri, USA
msudvarg@wustl.edu

Roger D. Chamberlain
Dept. of Computer Science and Engineering
Washington University
St. Louis, Missouri, USA
roger@wustl.edu

ABSTRACT

Many instruments used in high-energy particle physics observations, e.g., gamma-ray telescopes, use FPGAs for front-end signal processing of raw sensor data. The use of high-level synthesis (HLS) to express the signal processing algorithms has the potential to significantly reduce development time for new instruments of this type. We describe our experience with one of the computational stages in the signal processing pipeline, *island detection*, exploring its implementation across multiple configurations: 1D versus 2D islands, and 4-way versus 8-way connected-component labeling (CCL) in the 2D configuration. We report resource usage and performance for both configurations of 2D island detection, including the required optimizations necessary for HLS to be effective. Results indicate that our implementation can perform 4-way CCL on 15k images per second even for 43×43 pixel arrays.

CCS CONCEPTS

• **Applied computing** → **Astronomy**; • **Computer systems organization** → **Sensors and actuators**; • **Hardware** → **Reconfigurable logic applications**.

KEYWORDS

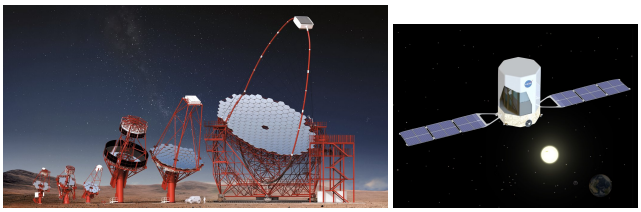
high-level synthesis, particle physics, CCL

ACM Reference Format:
Nick Song, Marion Sudvarg, and Roger D. Chamberlain. 2025. Connected-Component Labeling Using HLS for High-Energy Particle Physics Instruments. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3731599.3767413

1 INTRODUCTION

High-energy particle physics instruments, including gamma-ray telescopes (Fig. 1) like VERITAS [28], CTA [10], COSI [25], APT [3], and ADAPT [7]; neutrino detectors like IceCube [1]; and general-purpose particle detectors like the ATLAS experiment at CERN [24] produce large volumes of sensor data that must be processed and analyzed with high throughput.

(a) The VERITAS telescopes [26]. (b) COSI rendering [9].



(c) CTA Observatory rendering [11]. (d) APT rendering [22].

Fig. 1: Gamma-ray telescopes.

On such instruments, field-programmable gate arrays (FPGAs) are widely deployed to read out data from front-end electronics. Examples include the Flash ADC electronics system for VERITAS [5], the focal plane module readout on NUSTAR [13], and the NECTAr0 high-speed digitizer ASIC [12] for CTA. Increasingly, to meet their scientifically-crucial throughput requirements, they also employ FPGA logic to pre-process raw sensor readout values, reducing the volume of data transmitted, analyzed, and stored by downstream CPU- and GPU-based computational platforms.

Across instruments, these FPGA-based data processing pipelines share common functionality. However, diverse instruments require unique implementations of the constituent algorithms, and the logic is often rewritten largely from scratch for each new instrument that is developed. Traditionally, this logic is specified in a hardware description language (HDL) such as VHDL, Verilog, or SystemVerilog. Writing, simulating, and hardware debugging of HDL-based firmware is rarely straightforward and introduces significant overhead to the development cycle. As instrument designs increase in complexity and scale, this presents a tremendous challenge toward on-time and cost-effective project completion.

High-level synthesis (HLS) presents an attractive alternative. As an approach to expressing hardware designs at a conceptual level higher than that of register-transfer level (RTL) descriptions [8], HLS has proved its advantages over traditional HDL. It has a shorter learning curve for engineers and automates a significant amount of the workload in FPGA design and development. Sudvarg et al. [21, 23] have successfully developed HLS implementations of several algorithms for real-time data processing on board ADAPT [7],

SC Workshops '25, November 16–21, 2025, St Louis, MO, USA

ACM ISBN 979-8-4007-1871-7/2025/11
https://doi.org/10.1145/3731599.3767413

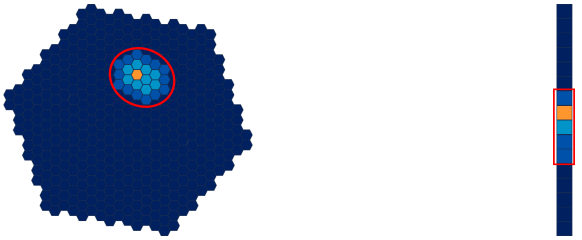


Fig. 2: Left: an island of signals detected in a 2D pixel array (e.g., for CTA [6]). Right: an island of signals detected in a 1D pixel array (e.g., for ADAPT [20]). Image from [23].

a suborbital gamma-ray and cosmic-ray telescope currently under development. These algorithms compose a pipeline that performs (among other stages) channel-level readout and waveform integration, then centroids over identified "islands" of consecutive nonzero integrals in a 1D array (see Fig. 2, Right) to identify the positions and energies of particle interactions within the detector.

The future of HLS-driven FPGA firmware design in high-energy instruments therefore seems promising, especially as a paradigm to improve reusability, enabling fast prototyping and deployment on new instruments. To this end, we are interested in exploring possibilities to leverage the current ADAPT computational pipeline to a broader context and making it more generic and suitable for other instruments in this domain.

In this paper, we focus on extending ADAPT's 1D island detection to 2D pixel arrays (see Fig. 2, Left), such as those found in the photomultiplier-based imaging sensors used by the Cherenkov Telescope Array (CTA) [10]. Pursuant to this, we propose a generic method that implements both 4-way and 8-way connected-component labeling (CCL). This is integrated into the pipeline of [23], enabling compile-time specification of 1D or 2D sensor arrangements.

Section 2 of this paper describes common features that drive requirements across many high-energy particle detectors, focusing on the ADAPT and CTA gamma-ray telescopes. It uses reported limitations of the existing CTA computational platform, a CPU-based server cluster, to motivate the use of FPGAs and an extension of the pipeline in [23]. Section 3 provides background on CCL and discusses related FPGA implementations. Section 4 outlines our HLS-based CCL design. Section 5 details our development and optimization efforts using the AMD Vitis HLS synthesis platform. It presents reported resource usage and throughput results along the way to highlight the impact of careful pragma and data structure selection on performance, even when authoring HLS in a high-level language (C++, in this case). We demonstrate that these efforts enable throughput of 15k events per second even for pixel arrays of size 43×43, meeting the stated goals of CTA. Finally, Section 6 concludes the paper and proposes directions for future work.

2 MOTIVATION

Our work is motivated by a class of particle physics instruments that trigger sensor readouts when they detect high-energy interactions. For many such instruments, fast triggering rates and extensive sensor arrays produce large volumes of data, which must be processed with high throughput. FPGAs, which can be programmed with application-tailored logic that supports wide parallelism and deep pipelining, are well-suited to such applications. This paper builds upon our earlier work to develop an FPGA-based data processing pipeline for the Antarctic Demonstrator for the Advanced Particle-astrophysics Telescope (ADAPT) instrument [21, 23].

ADAPT serves as a balloon-borne pathfinder for the Advanced Particle-astrophysics Telescope (APT, Fig. 1d), with an anticipated suborbital flight from Antarctica in December 2026 [4, 7, 27]. Due to the limited communication bandwidth typical of space-borne missions, the majority of the data filtering and analysis must be performed directly onboard the instrument. This drives the need for computationally efficient, real-time data processing capabilities. ADAPT mimics this environment, allowing our team to prototype these capabilities on a high-altitude platform.

ADAPT's detector has 4 interleaved layers of scintillating fiber trackers and imaging CsI calorimeters (ICC), allowing for gamma-ray and cosmic-ray measurements. Particle interactions in the tracker and ICC scintillators produce photons in the visible spectrum, which are transported via optical fibers and detected by silicon photomultipliers (SiPMs) arranged in 1D arrays. The SiPMs are continuously sampled by analog waveform digitizer ASICs that, when triggered, perform the analog to digital conversion (ADC).

Digital packets are read out by FPGAs which perform per-channel preprocessing and filtering, including data acquisition, pedestal subtraction, photon counting, and zero-suppression; as well as cross-channel analysis and event building stages, including island detection and centroiding to estimate the position and energy of each interaction within the detector. An overview of the pipeline is illustrated in Fig. 3. To ease prototyping and enable rapid revisions in parallel with ADAPT's design iterations during its ongoing development, we implemented the pipeline primarily in HLS [21, 23]. Through careful pragma selection and other optimizations, it can process 300k events per second.

Generality of Design Unlike ADAPT, which detects particles within the instrument itself, an Imaging Atmospheric Cherenkov Telescope (IACT) measures optical light emitted from high-energy gamma-ray interactions in the atmosphere. Nonetheless, these are members of a broader class of instruments that share several core data-processing steps in common.

For instance, CTA employs a three-stage pipeline in which the R0→DL1 phase (data calibration, image aggregation, and event cleaning) mirrors ADAPT's channel-level readout, pedestal subtraction, integration, and zero-suppression stages. While CTA's DL1→DL2 phase (energy, direction, and "gammaness" estimation) shares similar goals with ADAPT's island detection and centroiding [6], it involves some materially different computational modules. Relevant to this work, ADAPT's 2D spatial reconstruction uses perpendicular 1D arrays of optical fibers, whereas CTA has true 2D imaging capabilities using arrays of photomultiplier tubes (PMTs) as "pixels." Although both identify clusters ("islands") of adjacent "lit" pixels (those with integrated waveforms from SiPM/PMT readouts above a pre-defined threshold), the semantics of 1D versus 2D island detection are fundamentally different, as illustrated in Fig. 2.

Thus, we are interested in identifying and extracting the similar parts of the computation to create a generic pipeline so that other instruments might benefit from the existing work. Our eventual

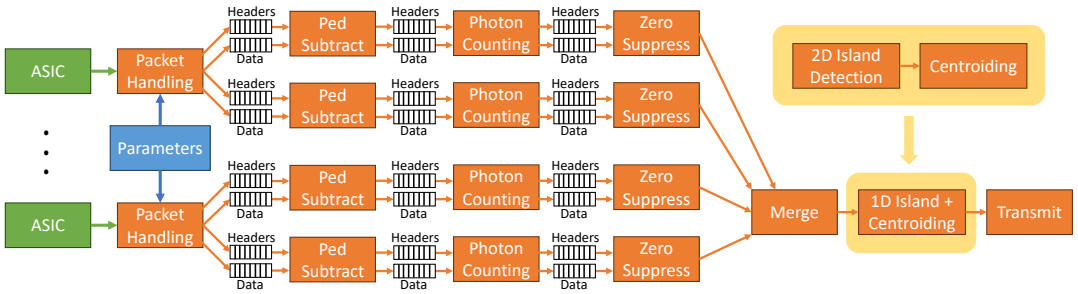


Fig. 3: ADAPT's prototype FPGA-based data processing pipeline, illustrating our contribution. Adapted from [21, Figure 2].

aim is to explore the potential of using HLS, along with generic programming techniques, to allow compile-time specification of computational stages and data stream formats. We believe this will ease adoption and enable rapid development compared to traditional HDL-based designs.

CTA The Cherenkov Telescope Array (CTA) Observatory [10], currently under advanced development, will consist of 64 IACT telescopes in Spain and Chile. The real-time data analysis pipeline for the observatory's multiple telescopes runs on a server cluster that aims to support up to 15k events per second [6]. Each server runs 8 threads of the R0→DL1 phase described above; each thread supports a 1.25 kHz event rate, for 10 kHz in total. The DL1→DL2 phase processes each event in about 1.3 ms. These reported results suggest that CTA's throughput target is not yet being met.

Pursuant to this, we focus our efforts on 2D island detection, a crucial stage of CTA's DL1→DL2 phase. We propose a generic method that implements connected-component labeling (CCL), supporting both 4-way and 8-way connected components. We implement it using HLS and integrate it into the pipeline of [21, 23], enabling compile-time switching between 1D and 2D signal arrays. The goal is to support the possibility of eventual integration of FPGA-based implementations of portions of the R0→DL1 and DL1→DL2 phases into CTA's real-time data analysis pipeline.

3 BACKGROUND AND RELATED WORK

Custom-tailored computational pipelines for image processing and pattern recognition in high-energy physics experiments have a long and storied history. Indeed, the ILLIAC III, built by the University of Illinois in 1966, was designed to scan, preprocess, and perform list-mode graph analysis of bubble chamber photographs [17].

In modern particle-physics instruments, connected-component labeling (CCL) [15] can play a critical role in **island detection**, clustering spatially correlated sensor activations that correspond to physical events, e.g., particle interactions in a scintillator. CCL is a fundamental image segmentation algorithm that identifies and labels groups of connected pixels in binary images. In 4-way CCL, pixels must be adjacent across an edge (top, right, bottom, or left) to be grouped. In 8-way CCL, pixels are also connected across their corners. This difference is illustrated in Fig. 4.

This work applies CCL in a real-time FPGA pipeline for:

- Identifying clusters of detections in 2D sensor arrays.
- Assigning unique labels to each spatially distinct region.
- Enabling efficient downstream tracking of interactions.

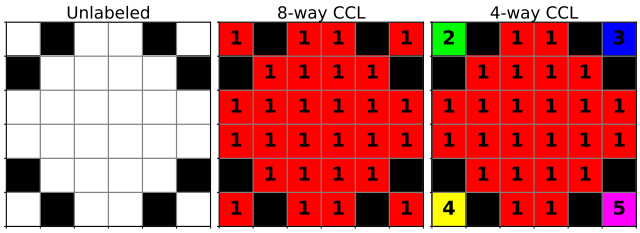


Fig. 4: 4-way vs. 8-way CCL for a 6×6 pixel image. Colors and numbers reflect the final label assigned to each component.

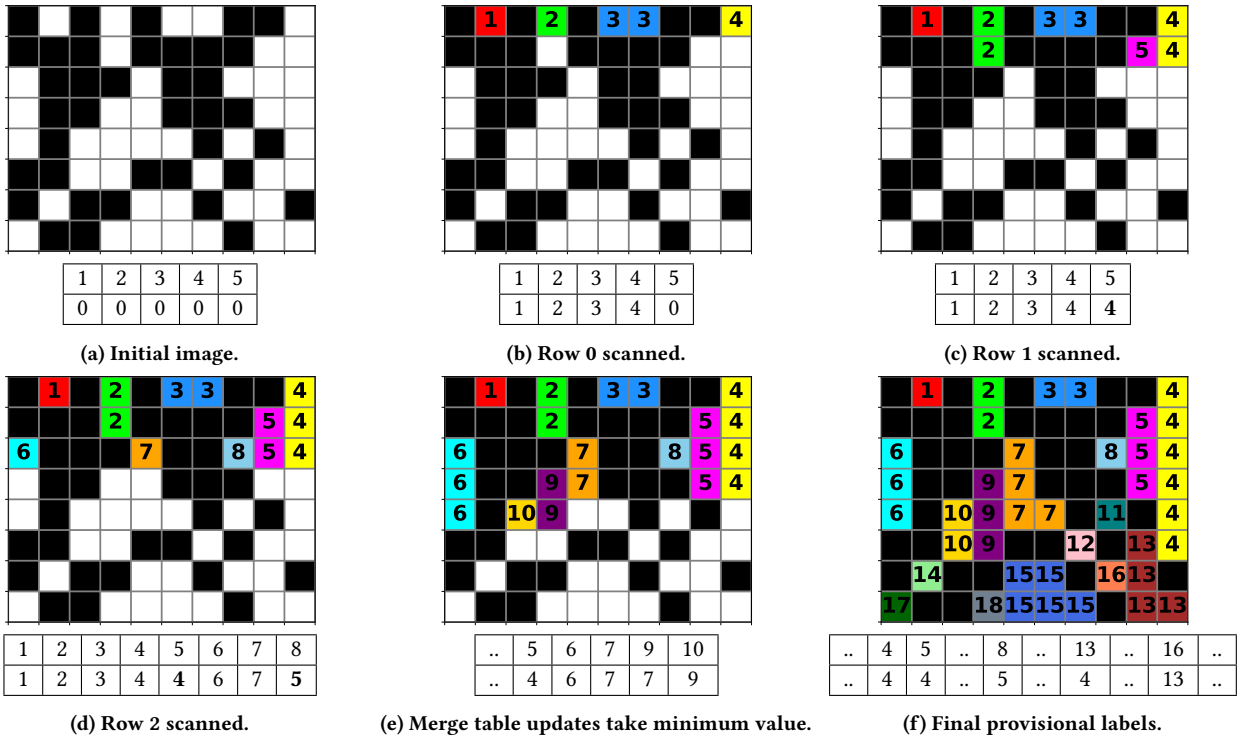
Several CCL strategies exist, including multi-pass, two-pass, and single-pass algorithms. A widely adopted method is the two-pass solution proposed by Rosenfeld and Pfaltz [19]. This approach first assigns provisional labels while recording label equivalences, then resolves them in a second pass to assign final labels.

CCL has been implemented in both software and hardware, with many optimizations aimed at improving throughput and latency. For instance, He et al. [14] proposed a fast two-pass algorithm that achieves high performance by:

- Reducing conditional logic through a 14-case analysis for 8-way connectivity.
- Optimizing label equivalence resolution using a flat union-find data structure with a representative label table.
- Minimizing memory access using case tables and hard-coded logic, avoiding runtime branching.

Kowalczyk et al. [16] successfully implemented a hardware-accelerated two-pass CCL on FPGAs for real-time 4K image processing at 60 FPS. In contrast, Bailey and Johnston [2] proposed a single-pass CCL algorithm that resolves label equivalences on-the-fly during the first scan, eliminating the need for a second relabeling pass. While this approach can reduce latency, it introduces significant control complexity and data dependencies, which are challenging to manage in a pipelined FPGA implementation.

Our design adopts a more balanced 1.5-pass approach: during the first raster scan, we assign provisional labels and record equivalence relationships in the merge table. After this pass, we resolve the merge table to collapse transitive label chains and assign compact, final island IDs. Rather than performing a second scan to rewrite labels, we access the resolved merge table directly to output final labels. This method retains the controlled memory access and modular structure of a two-pass algorithm while avoiding redundant data traversal, making it more efficient and better suited to real-time, HLS-based FPGA synthesis.



(a) Initial image. (b) Row 0 scanned. (c) Row 1 scanned.

(d) Row 2 scanned. (e) Merge table updates take minimum value. (f) Final provisional labels.

Fig. 5: 4-way CCL algorithm example for a 10×8 image. The top row of the merge table (below each image) indicates the group number. In our implementation, this is inferred from the 1-indexed array position. The bottom row indicates which group it will resolve to. A value of 0 indicates the group does not yet exist, i.e., no pixels are yet labeled with that group number.

4 DESIGN

This section describes the design of our connected-component labeling (CCL) algorithm for **2D island detection** in high-energy particle physics instruments. We note that this is a high-level overview, with examples showing the step-by-step procedural logic, that remains platform-agnostic. Details of our HLS-based implementation and optimizations are provided in Section 5.

Our algorithm consists of three major stages:

(1) **Data Acquisition and Preprocessing**: Input data is retrieved (in our case, from the upstream Merge component shown in Fig. 3) and structured into a 2D format.
(2) **CCL**: Labels are assigned using a **1.5-pass** labeling approach with a **merge table** to resolve equivalences.
(3) **Final Label Output**: The resolved labels are written to the output stream for downstream processing.

4.1 Data Structuring

Our algorithm begins by reading pixel intensity values from upstream processing modules. In our implementation, which integrates 2D CCL as an alternative method for island detection within ADAPT's prototype FPGA pipeline, these reads are from a Merge module (see Fig. 3) that merges zero-suppressed waveform integrals from multiple 16-channel digitizer ASICs. The output of the Merge module is a 16-channel-wide data FIFO. Our algorithm refactors the wide vectors into a single array (`data`) of channel values of size `NROWS×NCOLS`; these are user-configured at compile time with preprocessor macros. In HLS, we implement this as a 1D array, converting a specified row/column into an address according to `row×NCOLS+col`. Once pixel values are in `data`, we proceed with a raster scan.

4.2 Raster Scan and Provisional Labeling

Next, the CCL algorithm performs a row-major raster scan across pixels. As the scan progresses, provisional group labels are assigned to lit pixels according to the minimum of the group labels already assigned to its neighbors. For 4-way CCL, the top and left neighbors are checked (right and bottom neighbors have not yet been processed in the scan). 8-way CCL also checks the top-left neighbor. If no neighbors are lit, then the pixel is assigned to a new group.

Example 4.1. Consider 4-way CCL for the image shown in Fig. 5a. After row 0 (the top row) is scanned, group labels are assigned according to Fig. 5b. Notice that the two adjacent lit pixels are given the same label.

All lit neighbors of a lit pixel belong to the same group. Therefore, if a pixel has multiple lit neighbors that have already been assigned different group labels, this discrepancy will need to be resolved. We use a merge table to track equivalencies. The merge table is a simple 1-dimensional array with each entry indicating equivalency for the group of the corresponding index (1-indexed). A value of 0 indicates that the group does not yet exist.

Example 4.2. Notice in Fig. 5c that the second-to-last pixel of row 1 does not have top or left neighbors lit. It is therefore provisionally assigned to its own new group; in this case, group 5. However, it

should be assigned to group 4. Our algorithm identifies this when the last pixel of row 1 is reached: its top neighbor is part of group 4, but its left neighbor is labeled 5. In this case, the last pixel is assigned the minimum value (4) and the merge table entry for group 5 is updated to indicate equivalency with 4.

Example 4.3. When row 2 is scanned, Fig. 5d shows that the third-to-last pixel does not have top or left neighbors lit, so it is provisionally assigned to a new group (8). In reality, it is a member of group 4. When the next pixel (labeled 5) is reached, as in Example 4.2, our algorithm identifies that it belongs to the same group as the previous pixel (labeled 8). The merge table entry for group 8 is therefore updated to point to group 5. Group 5 itself points to group 4; transitive label chains are later resolved (Section 4.3).

We note that merge table entries are updated to equal the minimum among the neighboring pixel group labels ***and the existing entry value, if non-zero***. This avoids overwriting earlier merge table entries pointing to smaller labels, which would break label equivalence resolution.

Example 4.4. In Fig. 5e we see that after a pixel is provisionally assigned to group 10, the next pixel in the row is assigned to group 9 (since this is the minimum among its top and left neighbors). Thus, group 10 in the merge table will be updated to point to group 9. However, group 9 in the merge table remains pointing to group 7 (the smaller value it was assigned when scanning the row above).

4.3 Resolve Merge Table

To collapse transitive group label chains, our algorithm resolves the merge table in ascending order of label number, until a zero-value entry (indicating no more groups) is reached. When a group merge is detected, i.e., when an entry does not point to itself, it is resolved by pointing all of the child groups (those detected later) to the root group (the one initially detected). This is done by double-dereferencing to access the root: `mt[idx]` is set to `mt[mt[idx]]`.

Example 4.5. In Fig. 5f, we see that provisional group labels 4, 5, 8, 13, and 16 all belong to the same group. The merge table entries after the raster scan are shown below the figure. Entries 4, 5, and 13 remain the same, since but `mt[8]` is set to `mt[mt[8]] = mt[5] = 4`, and similarly for `mt[16]`.

Resolving the merge table in ascending order naturally resolves any chains of equivalencies since later merges automatically refer to already-resolved entries.

4.4 Final Label Output

The final group labels for each pixel are passed downstream by directly indexing the resolved merge table according to the provisional label, and passing the group label to which it points. This avoids writing back to the data array.

5 IMPLEMENTATION AND EXPERIMENTS

This section presents the stepwise optimization of our 2D CCL implementation using AMD's (previously, Xilinx) Vitis HLS for FPGA-based island detection. We begin with a naïve baseline design, then progressively introduce **storage binding**, **array partitioning with loop unrolling**, and **loop pipelining**. Each stage addresses specific bottlenecks observed in synthesis, building on the previous stage's improvements. Performance at each stage is summarized in Tables 1 and 2 for 4-way and 8-way connectivity, respectively.

Table 1: Island Detection Results for Size 8 × 10 (4-way).

Stage	Latency	II	BRAM	FF	LUT
Baseline	998	998	4	1076	2257
Bind Storage	1158	1158	7	1014	2303
Unrolled	1018	1018	5	1068	2629
Pipelined	340	340	5	4229	4096

Table 2: Island Detection Results for Size 8 × 10 (8-way).

Stage	Latency	II	BRAM	FF	LUT
Baseline	1398	1398	4	1196	2746
Bind Storage	1718	1718	7	1200	2863
Unrolled	1578	1578	5	1254	3189
Pipelined	406	406	3	7041	6583

5.1 Naïve Implementation

The baseline design implements the 1.5-pass CCL algorithm described in Section 4 without tool-specific optimization pragmas. Fig. 6 shows a more detailed implementation.

```
void island_detection_2d(...) {
  loop through (row,col) in NROWS × NCOLS:
    if pixel_val[row][col] != 0:
      top_valid   = 0 < row < NROWS && pixel_val[row-1][col] != 0;
      left_valid  = 0 < col < NCOLS && pixel_val[row][col-1] != 0;
      top  = top_valid  ? top_label  : INVALID;
      left = left_valid ? left_label : INVALID;
      assigned = top_valid || left_valid;
      curr_label = assigned ? min(top,left) : new_label;
      mt[top] =  min(mt[top],curr_label);
      mt[left] = min(mt[left],curr_label);
}
```

Fig. 6: CCL implementation.

As described in Section 4.3, data is received as a flattened 1D array from the `merged_integrals` stream, reconstructed into an `NROWS` × `NCOLS` matrix, and processed in row-major order. The top-level function configures the dataflow pipeline to operate in 1D/2D mode via a compile-time switch `TWO_DIMENSION`. When set, the `island_detection_2d` function is compiled into the pipeline instead of the original `island_detection_and_centroiding`, which adds flexibility to the pipeline without touching the core design. Connectivity is compile-time selectable between 4-way (top and left neighbors only) and 8-way (adding top-left and top-right neighbors). A single manifest constant, `EIGHTWAY_NEIGHBORS`, is used with preprocessor #if statements to enable or disable the resources needed for top-left and top-right neighbors.

Each pixel's label is determined according to the procedure described in Section 4.2 by:

(1) Checking visited neighbors for labels.
(2) Assigning the minimum neighbor label if any exist.
(3) Allocating a new label otherwise.
(4) Recording equivalences in the `merge_table`.

Observed bottlenecks:

- *Merge table access* used default register storage, consuming FFs and LUTs.

- *Inner loop initiation interval (II)* matched the total loop trip-count, as the scheduler serialized all operations.

Resource utilization and timing results for the baseline implementation of 4-way and 8-way CCL are shown in the *Baseline* rows of Tables 1 and 2, respectively.

5.2 Bind Storage

To reduce the merge table's FF/LUT utilization and to allow concurrent reads and writes across it, we bound the merge table to dual-port BRAM using `#pragma HLS bind_storage`.

This mapping supports either two simultaneous reads or one read and one write per cycle, conserving logic resources. However, it also introduces a fixed one-cycle read latency for every merge table access and limits total concurrent accesses to two per cycle. In this non-pipelined design, the extra cycle per iteration increased the overall loop latency (998 → 1158 cycles for 4-way connectivity). This slowdown is expected, because in the baseline design, merge table lookups in registers were effectively combinational, whereas in BRAM they become sequential operations.

Result Discussion: Baseline → Bind Storage These results are summarized in the *Bind Storage* rows of Tables 1 and 2.

- **Latency/II:** +16.0% (998 → 1158). Dual-port BRAM introduces a fixed 1-cycle read latency that is not hidden without pipelining.
- **BRAM:** +75.0% (4 → 7) due to mapping the merge table to `RAM_2P`.
- **FF:** −5.8% (1076 → 1014) as register-based storage is reduced.
- **LUT:** +2.0% (2257 → 2303).

Takeaway: Storage binding saves logic but slows the design due to one-cycle BRAM latency. In Section 5.4, this latency becomes fully hidden inside a pipeline, allowing the II of the inner loop to reach 1 without exhausting FF/LUT resources.

5.3 Array Partitioning and Loop Unrolling

To exploit parallelism in the input data, we unrolled the channel-processing loop by a factor of 16, matching the 16 input channels per ALPHA ASIC, and applied cyclic partitioning to the `data` array. This is shown in Fig. 7.

```
static int32_t data[ROW*COL];
#pragma HLS ARRAY_PARTITION variable=data
        cyclic factor=num_channels dim=1

loop through a in 5 ALPHA ASICs:
    integral = merged_integrals.read();
    loop through c in 16 channels:
    #pragma HLS UNROLL FACTOR=num_channels
        data[a*NUM_CHANNELS + c] = integral[c];
```

Fig. 7: Array partitioning + loop unrolling.

This enabled parallel processing of 16 pixels per cycle and eliminated the memory-port bottleneck caused by single-bank storage.

Note that `num_channels` is not a preprocessor macro, but rather a template parameter. In Vitis HLS, preprocessor macros cannot be passed directly to HLS pragmas. However, template parameter *can* be, and a preprocessor macro can then be passed as a template argument. This distinction is shown in Fig. 8.

```
#define NCHANNELS 16

//Not allowed:
#pragma HLS UNROLL FACTOR=NCHANNELS

//Allowed:
template <int num_channels>
island_detection_2d(...) {
    loop through 16 channels:
    #pragma HLS UNROLL FACTOR=num_channels
}
island_detection_2d<num_channels=NCHANNELS>(...);
```

Fig. 8: Preprocessor macros in HLS pragmas.

Result Discussion: Bind Storage → Unrolled These results are summarized in the *Unrolled* rows of Tables 1 and 2.

- **Latency/II:** −12.1% (1158 → 1018). Parallel channel handling masks part of the BRAM delay, even without pipelining.
- **BRAM:** −28.6% (7 → 5) due to layout/pruning differences after unrolling/partitioning.
- **FF:** +5.3% (1014 → 1068) from replicated control/data paths.
- **LUT:** +14.2% (2303 → 2629) for the same reason.

Takeaway: Unrolling partially recovers performance but cannot overcome loop-carried dependencies alone.

5.4 Loop Pipelining

We applied loop pipelining to the inner column loop of the first labeling pass to achieve `II=1`, as shown in Fig. 9. To remove read-after-write hazards between the current pixel and its left neighbor, we used a buffer (`prev`) to store the left neighbor's label. Merge table updates were decoupled from the labeling loop using `hls::stream` queues for the top and left neighbors (and diagonals in the 8-way case), which reduced BRAM port contention.

```
loop through (row,col) in NROWS × NCOLS:
#pragma HLS PIPELINE II=1
    if data != 0:
        . . .
        left = left_valid ? prev : INVALID;
        curr_label = assigned ? min(top,left) : new_label;
        if  !assigned: stream_top  << (row,   col,   new_label);
        if  top_valid: stream_top  << (row-1, col,   curr_label);
        if left_valid: stream_left << (row,   col-1, curr_label);
```

Fig. 9: Pipelined CCL first pass.

Because the merge table was already bound to dual-port BRAM in the previous stage, the one-cycle read latency no longer affected total loop time: the pipelined design could issue the next iteration before the previous BRAM access completed. This is where the storage binding paid off: it preserved low FF/LUT usage while achieving maximum throughput.

Unrolled → Pipelined These results are summarized in the *Unrolled* rows of Tables 1 and 2.

- **Latency/II:** −66.6% (1018 → 340) with `II=1` for the inner loop. Pipeline overlap hides the earlier BRAM read latency.
- **BRAM:** 0% (5 → 5).
- **FF:** +295.8% (1068 → 4229) due to buffering, stream interfaces, and pipeline registers.
- **LUT:** +55.8% (2629 → 4096).

Table 3: Scalability Analysis (4-way Connectivity).

Array Size	Latency	II	BRAM	FF	%	LUT	%
8×10	340	340	5	4,229	1	4,096	2
16×16	956	956	5	9,885	2	6,003	2
24×24	2,076	2,076	21	19,682	4	10,133	4
32×32	3,644	3,644	21	34,029	8	15,485	7
43×43	6,668	6,668	23	63,358	15	26,416	12
64×64	14,396	14,396	28	132,369	32	41,588	20

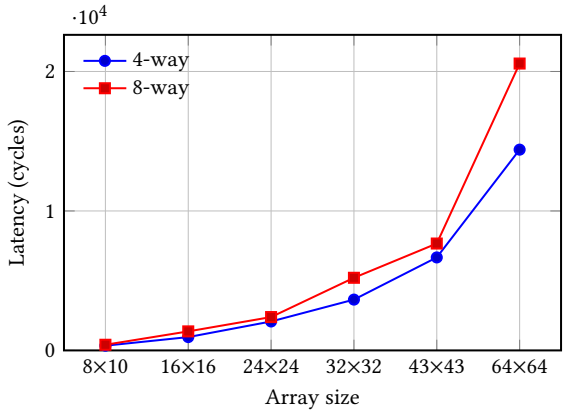


Fig. 10: Latency scaling of the fully optimized pipelined design for 4-way vs. 8-way connectivity across array sizes. All runs meet inner-loop `II=1`; latency increases nearly linearly with pixel count, with 8-way incurring additional cost from diagonal-neighbor checks and wider merge-resolution logic.

Takeaway: Pipelining delivers the step-change in throughput; prior storage binding now helps by keeping logic pressure manageable while the BRAM latency is fully hidden.

Notably, applying the same pipelining strategy to the **8-way connectivity** implementation yields an even larger relative speedup: a −74.3% latency reduction (1578 → 406). This is because the additional diagonal neighbor checks introduce more independent operations that the pipeline can parallelize.

During this optimization, Vitis HLS reported a false memory dependency on the `stream_top` FIFO in the 4-way connectivity design. The algorithm guarantees that `stream_top` is written either for a new island initialization or for a top-neighbor merge, but never both in the same iteration. However, the compiler could not prove this exclusivity at compile time and serialized the loop, which would have increased the II.

To resolve this, we made the exclusivity explicit by restructuring the update logic into an `if/else if` form, ensuring that only one write to `stream_top` occurs per iteration (Fig. 12). This removes the false dependency and allows the inner loop to meet `II=1` without adding another FIFO.

```
if      !assigned: stream_top  << (curr_idx, new_label);
else if top valid: stream_top  << (top_idx,  curr_label);
```

Fig. 12: False dependency removed via single-write pattern.

Table 4: Scalability Analysis (8-way Connectivity).

Array Size	Latency	II	BRAM	FF	%	LUT	%
8×10	406	406	3	7,041	1	6,583	3
16×16	1,365	1,365	3	15,631	3	10,031	4
24×24	2,392	2,392	25	30,303	7	17,128	8
32×32	5,208	5,208	25	51,989	12	26,860	13
43×43	7,664	7,664	25	95,729	23	46,001	22
64×64	20,570	20,570	32	199,694	48	75,641	37

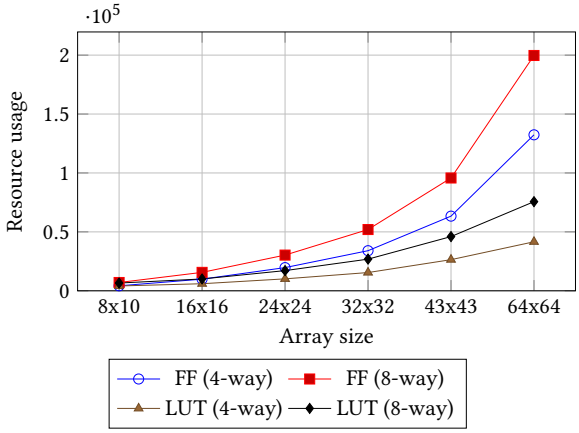


Fig. 11: Flip-flop (FF) and LUT scaling for 4-way vs. 8-way CCL. FF usage increases *approximately linearly* with pixel count, while LUT usage grows *sublinearly*, reflecting control-logic dominated growth with limited datapath replication.

5.5 Performance and Scalability Results

All designs were synthesized in Vitis HLS 2022.1 for a Xilinx Kintex-7 XC7K325T-2FBG676I at 100 MHz. Simulations used C and RTL co-simulation to verify correctness on representative event data from the ADAPT pipeline. We synthesized the fully optimized pipelined design for a range of input sizes to evaluate scalability. The merge table is sized to accommodate the maximum number of provisional group labels, growing proportionally to the array dimensions:

$$\texttt{MERGETABLE\_SIZE} = \frac{\texttt{ROW}+1}{2} \times \frac{\texttt{COL}+1}{2}.$$

Tables 1 and 2 summarize the results for 4-way and 8-way CCL, respectively, on an 8×10 pixel array. Tables 3 and 4 illustrate scalability, showing timing and resource utilization numbers as the image size increases. Figs. 10 and 11 show the same results graphically, directly comparing 4-way and 8-way CCL latency and resource usage. We note that the array size of 43×43 roughly corresponds to CTA's Large Size Telescope (LST), which has 1855 pixels [18]. With 4-way CCL, we achieve CTA's target 15 kHz event rate:

$$\frac{1}{6668}\frac{\text{events}}{\text{cycle}} \cdot \frac{1}{10}\frac{\text{cycles}}{\text{ns}} \cdot 10^9 \frac{\text{ns}}{\text{s}} = 14{,}997 \frac{\text{events}}{\text{s}}$$

Latency and FF usage scale almost linearly with the total number of pixels, while BRAM usage grows in discrete steps due to the merge table capacity and HLS FIFO buffering. LUT usage also

increases with input size but at a slower rate relative to FFs, as control logic growth dominates over datapath replication. Under ideal conditions without resource or timing constraints, throughput scaling suggests that the pipelined 8-way design could handle images up to 813×813 pixels at 30 fps, while the 4-way design could process 975×975 pixels at the same frame rate.

Observations:

- **Latency and II:** For all image sizes, the inner-loop `II=1`, and latency increases nearly linearly with the number of pixels.
- **Flip-Flops:** FF usage shows a strong linear correlation with the total pixel count due to per-pixel control and state tracking logic.
- **LUTs:** LUT usage grows sublinearly relative to FFs, reflecting that most new pixels reuse existing datapath structures while adding proportionally less control logic.
- **BRAM:** Stepwise increases are caused by merge table and FIFO sizing; jumps occur when storage exceeds a BRAM block threshold.

Additional 8-Way Connectivity Observations:

- **Throughput/latency overhead vs. 4-way:** For the same array sizes, 8-way increases latency by **+15–43%**.
- **Resource overhead:** Adding diagonal checks increases FF usage by **+51–67%** and LUT usage by **+61–82%**, due to a wider tree for determining the minimum among neighbor labels, additional comparator logic, and more merge-update streams.
- **Scaling trends:** 8-way maintains inner-loop `II=1` across sizes and preserves near-linear latency scaling, but with greater FF and LUT demand compared to 4-way.
- **Timing dips:** Small negative slack at certain sizes (8×10, 24×24, and 32×32) is likely caused by FIFO implementation changes (LUTRAM ↔ BRAM), increased comparator width, and extra mux/compare stages on the critical path. These effects are size-dependent and disappear at 16×16 and 64×64.

6 CONCLUSIONS AND FUTURE WORK

We have presented a generalized FPGA-based 1.5-pass connected-component labeling (CCL) design, implemented in High-Level Synthesis (HLS), for island detection in astrophysical instruments. The design extends the original ADAPT preprocessing pipeline with a compile-time switch between 1D and 2D processing modes, enabling deployment across multiple instrument architectures without redesign.

On a Kintex-7 target, the optimized 2D pipeline reduced latency (4-way connectivity) compared to the naïve baseline by 66%. The design can process 15k events per second with array size (43×43), similar to pixel count from the CTA Large Size Telescope. For the more complex 8-way connectivity case, pipelining achieved an even greater relative improvement, reducing latency by 74%. These results demonstrate that the benefits of pipelining become more pronounced with increasing neighborhood complexity.

Scalability tests up to 64×64 pixels on a single FPGA confirmed that the design sustains inner-loop `II=1` for both 4-way and 8-way connectivity. Latency and FF usage scale almost linearly with the total number of pixels, BRAM usage grows in discrete steps, and LUT usage increases sublinearly relative to FFs. To realize integration into CTA's real-time analysis pipeline, we will also need to consider system scalability concerns, e.g., communication costs.

Future work should investigate a single-pass CCL approach to reduce latency by removing the need for a second scan. The main challenge lies in efficiently resolving label equivalences in a single traversal while avoiding dynamic memory dependencies. We also intend to evaluate a two-pass implementation.

Currently, the inner loop is fully pipelined (`II=1`) while the outer loop remains serialized due to unresolved dependencies between top, top-left, and top-right neighbors. Achieving a fully pipelined first pass could further reduce latency, but may require additional buffering and logic replication. As future work, we intend to develop this using both HLS and an RTL-optimized design, and compare their performance, area, and maintainability of implementation.

Scalability tests showed nearly linear growth in latency and FF usage with array size. Future work should focus on breaking this scaling trend by:

- Optimizing control logic to reduce per-pixel FF overhead.
- Exploring hierarchical or tiled processing to limit merge table and FIFO growth.
- Using compressed or dynamically allocated label storage to reduce BRAM and FF requirements.
- Investigating selective pipelining or partial unrolling to balance performance and resource usage at larger scales.

The current design writes one pixel label per cycle to the output FIFO, which becomes a major latency contributor at larger image sizes. Widening the interface to output multiple labels per cycle could reduce this overhead, with future work focusing on keeping the enhancement generic across instruments. We believe a number of data processing pipeline stages that are frequently employed in high-energy particle physics instruments are amenable to the type of generalization we have demonstrated here.

We also intend to more rigorously and formally verify the correctness of the presented algorithms to ensure that all edge cases are handled and result in correct labels. During final testing, we identified one corner case where some transitive merge table label chains are not resolved correctly for 4-way CCL (though the issue does not arise in 8-way CCL). While the image patterns that cause this do not arise in the relatively concave island shapes present in our target application, this needs to be addressed for better generality. Although we believe we have identified the logical fix, integrating it into our optimized, pipelined HLS design requires additional effort.

ACKNOWLEDGMENTS

Support provided by NASA award 80NSSC21K1741, the McDonnell Center for the Space Sciences, the Peggy and Steve Fossett Foundation, and a Washington University OVCR seed grant.

REFERENCES

[1] Rasha Abbasi, Yasser Abdou, T Abu-Zayyad, M Ackermann, J Adams, JA Aguilar, M Ahlers, MM Allen, D Altmann, K Andeen, et al. 2012. The design and performance of IceCube DeepCore. *Astroparticle Physics* 35, 10 (2012), 615–624. https://doi.org/10.1016/j.astropartphys.2012.01.004

[2] D. G. Bailey and C. T. Johnston. 2007. Single Pass Connected Components Analysis. In *Proc. of Image and Vision Computing New Zealand*. 282–287.

[3] James Buckley et al. 2021. The Advanced Particle-astrophysics Telescope (APT) Project Status. In *Proc. of 37th Int'l Cosmic Ray Conference*, Vol. 395. Sissa Medialab, 655:1–655:9. https://doi.org/10.22323/1.395.0655

[4] James H. Buckley, Jeremy Buhler, and Roger D. Chamberlain. 2024. The Advanced Particle-astrophysics Telescope (APT): Computation in Space. In *Proc. of 21st Int'l Conference on Computing Frontiers Workshops and Special Sessions*. ACM, 122–127. https://doi.org/10.1145/3637543.3652980

[5] James H Buckley, P Dowkontt, K Kosack, and P Rebillot. 2003. The VERITAS flash ADC electronics system. In *Proc. of 28th Int'l Cosmic Ray Conference*. 2827–2830.

[6] Sami Caroff, Pierre Aubert, Enrique Garcia, Gilles Maurin, Vincent Pollet, and Thomas Vuillaume. 2023. The Real Time Analysis Framework of the Cherenkov Telescope Array's Large-Sized Telescope. In *Proc. of 38th Int'l Cosmic Ray Conference*, Vol. 444. Sissa Medialab, 616:1–616:9. https://doi.org/10.22323/1.444.0616

[7] Wenlei Chen, James Buckley, et al. 2023. Simulation of the instrument performance of the Antarctic Demonstrator for the Advanced Particle-astrophysics Telescope in the presence of the MeV background. In *Proc. of 38th Int'l Cosmic Ray Conference*, Vol. 444. Sissa Medialab, 841:1–841:9. https://doi.org/10.22323/1.444.0841

[8] Jason Cong, Bin Liu, Stephen Neuendorffer, Juanjo Noguera, Kees Vissers, and Zhiru Zhang. 2011. High-level synthesis for FPGAs: From prototyping to deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 4 (2011), 473–491. https://doi.org/10.1109/TCAD.2011.2110592

[9] Michael F. Corcoran. 2021. COSI: Compton Spectrometer and Imager. https://heasarc.gsfc.nasa.gov/docs/objects/heapow/archive/technology/cosi.html. HEASARC "High Energy Astrophysics Picture of the Week" archive. Last modified February 27, 2024. NASA's Goddard Space Flight Center..

[10] CTA Consortium, M Actis, G Agnetta, F Aharonian, A Akhperjanian, J Aleksić, E Aliu, D Allan, I Allekotte, F Antico, et al. 2011. Design concepts for the Cherenkov Telescope Array CTA: an advanced facility for ground-based high-energy gamma-ray astronomy. *Experimental Astronomy* 32 (2011), 193–316. https://doi.org/10.1007/s10686-011-9247-0

[11] CTAO. 2018. Proposed CTA Telescopes (eso1841f). https://www.eso.org/public/images/eso1841f/. Licensed under Creative Commons Attribution 4.0 International (CC BY 4.0); image created by the European Southern Observatory (ESO), credit CTAO..

[12] E. Delagnes, J. Bolmont, P. Corona, D. Dzahini, F. Feinstein, D. Gascón, J.-F. Glicenstein, F. Guilloux, C. L. Naumann, P. Nayman, F. Rarbi, A. Sanuy, J.P. Tavernet, F. Toussenel, P. Vincent, and S. Vorobiov. 2011. NECTAr0, a new high speed digitizer ASIC for the Cherenkov Telescope Array. In *Nuclear Science Symposium Conference Record*. IEEE, 1457–1462. https://doi.org/10.1109/NSSMIC.2011.6154348

[13] Fiona A. Harrison, Steve Boggs, Finn Christensen, William Craig, Charles Hailey, Daniel Stern, William Zhang, et al. 2010. The Nuclear Spectroscopic Telescope Array (NuSTAR). In *Space Telescopes and Instrumentation 2010: Ultraviolet to Gamma Ray*, Vol. 7732. SPIE, 189–196. https://doi.org/10.1117/12.858065

[14] Longin He, Yuyan Chao, Kenji Suzuki, and Kesheng Wu. 2009. Fast Connected-Component Labeling. In *Proc. of the Int'l Conference on Pattern Recognition*. IEEE, 1977–1980. https://doi.org/10.1016/j.patcog.2008.10.011

[15] Lifeng He, Xiwei Ren, Qihang Gao, Xiao Zhao, Bin Yao, and Yuyan Chao. 2017. The connected-component labeling problem: A review of state-of-the-art algorithms. *Pattern Recognition* 70 (2017), 25–43. https://doi.org/10.1016/j.patcog.2017.04.018

[16] Marcin Kowalczyk, Paweł Ciarach, Dominika Przewłocka-Rus, and Tomasz Kryjak. 2021. Real-Time FPGA Implementation of Parallel Connected Component Labelling for a 4K Video Stream. *Journal of Signal Processing Systems* 93 (2021), 481–498. https://doi.org/10.1007/s11265-021-01636-4

[17] Bruce H. McCormick. 1963. The Illinois Pattern Recognition Computer-ILLIAC III. *IEEE Transactions on Electronic Computers* EC-12, 6 (1963), 791–813. https://doi.org/10.1109/PGEC.1963.263562

[18] C. Perennes, M. Doro, D. Corti, L. Lessio, M. Mallamaci, M. Mariotti, R. Rando, and I. Salmaso. 2020. Optical feasibility of an upgrade of the CTA LST camera to SiPM. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 984 (2020), 164485. https://doi.org/10.1016/j.nima.2020.164485

[19] Azriel Rosenfeld and John L. Pfaltz. 1966. Sequential Operations in Digital Picture Processing. *J. ACM* 13, 4 (1966), 471–494. https://doi.org/10.1145/321356.321357

[20] Marion Sudvarg et al. 2023. Front-End Computational Modeling and Design for the Antarctic Demonstrator for the Advanced Particle-astrophysics Telescope. In *Proc. of 38th Int'l Cosmic Ray Conference*, Vol. 444. Sissa Medialab, 764:1–764:9. https://doi.org/10.22323/1.444.0764

[21] M. Sudvarg et al. 2025. FPGA-Based Data Processing using High-Level Synthesis on the Antarctic Demonstrator for the Advanced Particle-astrophysics Telescope (ADAPT). In *Proc. 39th Int'l Cosmic Ray Conf.*, Vol. 501. Sissa Medialab, 852:1–852:9.

[22] Marion Sudvarg, Jeremy Buhler, Roger Chamberlain, Chris Gill, and James Buckley. 2022. Work in Progress: Real-Time GRB Localization for the Advanced Particle-astrophysics Telescope. In *Proc. of 15th Wkshp. on Operating Systems Platforms for Embedded Real-Time Applications*. 57–61.

[23] Marion Sudvarg, Chenfeng Zhao, Ye Htet, Meagan Konst, Thomas Lang, Nick Song, Roger D. Chamberlain, Jeremy Buhler, and James H. Buckley. 2024. HLS Taking Flight: Toward Using High-Level Synthesis Techniques in a Space-Borne Instrument. In *Proc. of 21st Int'l Conference on Computing Frontiers*. ACM, 115–125. https://doi.org/10.1145/3649153.3649209

[24] The ATLAS Collaboration, G Aad, et al. 2008. The ATLAS Experiment at the CERN Large Hadron Collider. *Journal of Instrumentation* 3, 08 (Aug. 2008), S08003. https://doi.org/10.1088/1748-0221/3/08/S08003

[25] J. Tomsick et al. 2023. The Compton Spectrometer and Imager. In *Proc. of 38th Int'l Cosmic Ray Conference*, Vol. 444. Sissa Medialab, 745:1–745:9. https://doi.org/10.22323/1.444.0745

[26] VERITAS. 2009. The VERITAS array, an air Cherenkov telescope designed to detect low-energy cosmic rays. https://commons.wikimedia.org/wiki/File:VERITAS_array.jpg. Public domain. Image courtesy of the National Science Foundation. Original source: https://www.nsf.gov/news/mmg/mmg_disp.jsp?med_id=64717&from=.

[27] Daisy Wang, Ye Htet, Marion Sudvarg, Roger Chamberlain, Jeremy Buhler, and James Buckley. 2025. Coordinating Instruments for Multi-Messenger Astrophysics. In *Proc. of 22nd Int'l Conference on Computing Frontiers Workshops and Special Sessions*. ACM, 213–218. https://doi.org/10.1145/3706594.3727948 arXiv:whscbb25.pdf

[28] TC Weekes et al. 2002. VERITAS: the very energetic radiation imaging telescope array system. *Astroparticle Physics* 17, 2 (2002), 221–243. https://doi.org/10.1016/S0927-6505(01)00152-9