WIDAR Prototypes: Correlator Backend & Fast Formatter

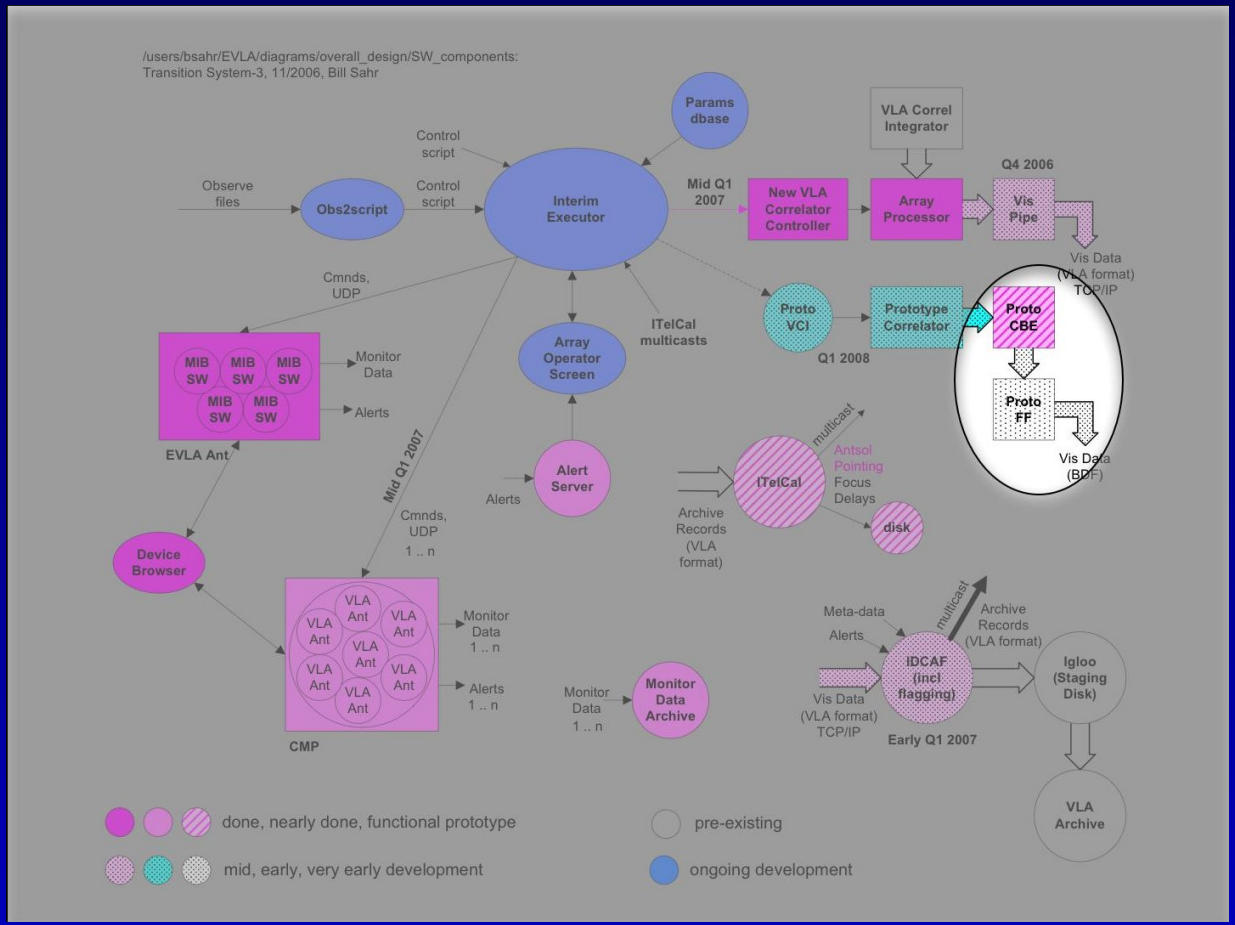
EVLA M&C Transition System Software CDR — 2006–12–05

M. Pokorny

Outline

- Subsystem overview
- Implementation environment
- Architecture
- CBE/FF interfaces
- Node processes
- CBE program designs
- Design details
- Fast formatter design
- Current status

CBE/FF in transition system software



Implementation environment

Development tools

- C language, GNU libc, gcc
- CVS
- GNU autotools
- dejagnu, check (with gcov)
- oprofile

Implementation environment

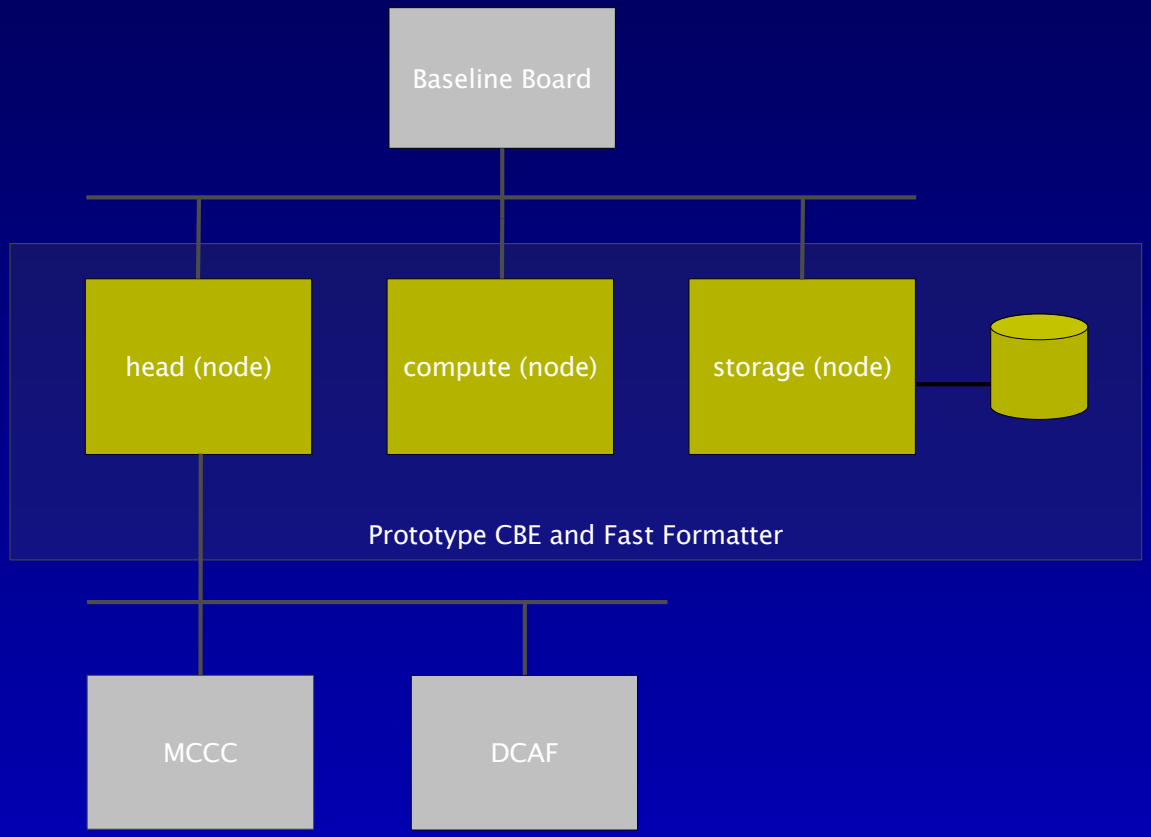
Development tools

- C language, GNU libc, gcc
- CVS
- GNU autotools
- dejagnu, check (with gcov)
- oprofile

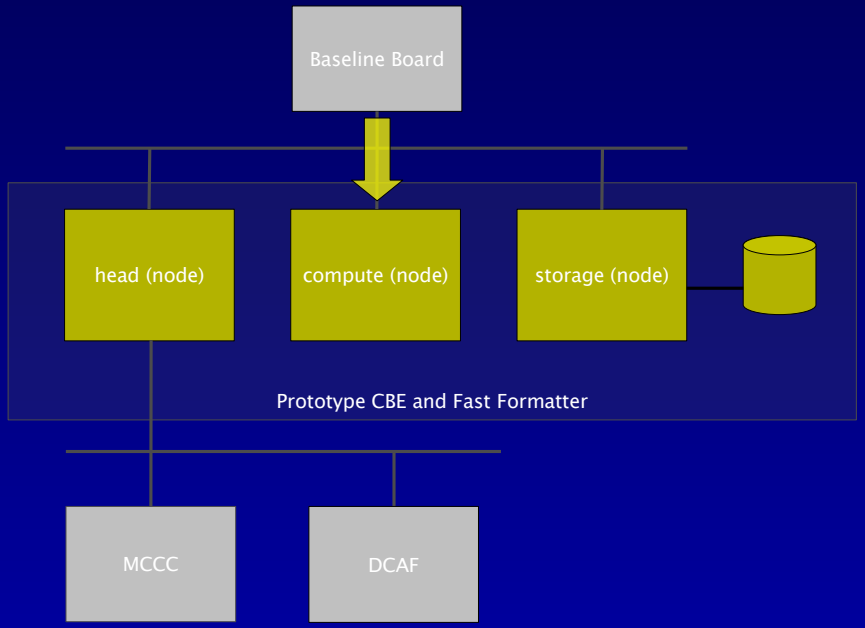
Library usage

glib, popt, fftw, cblas, *atlas*, *check*

Architecture

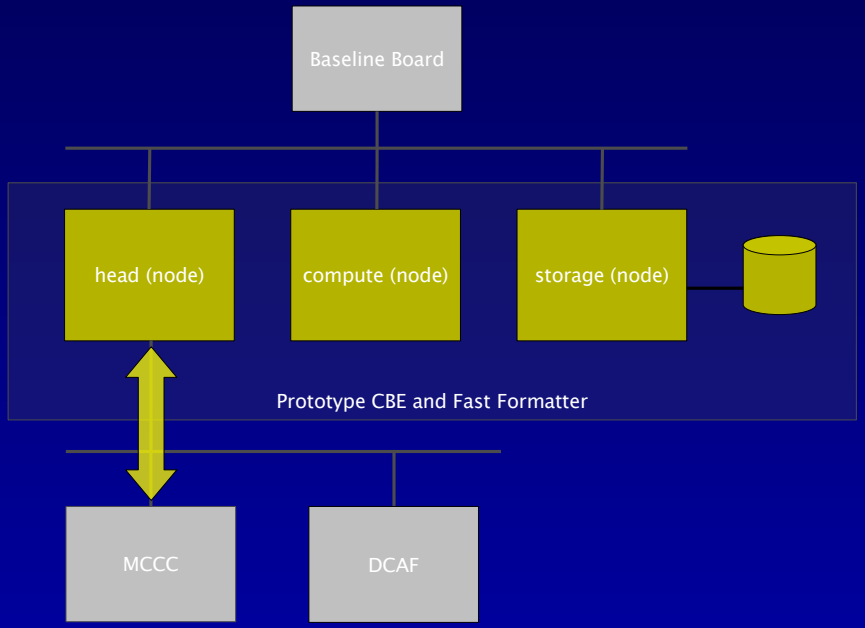


Baseline board interface



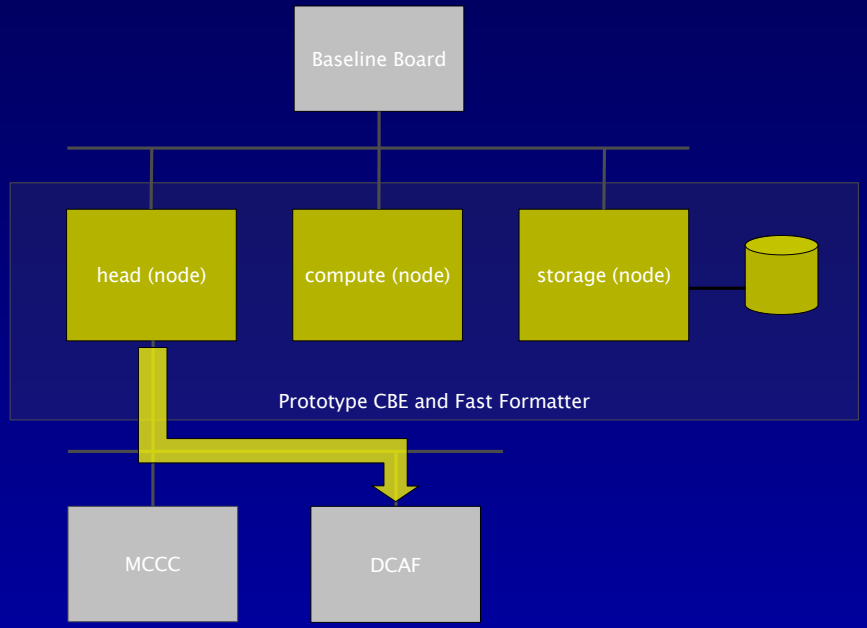
- lag frames delivered *via* UDP packets
- read-only
- compute node

MCCC interface



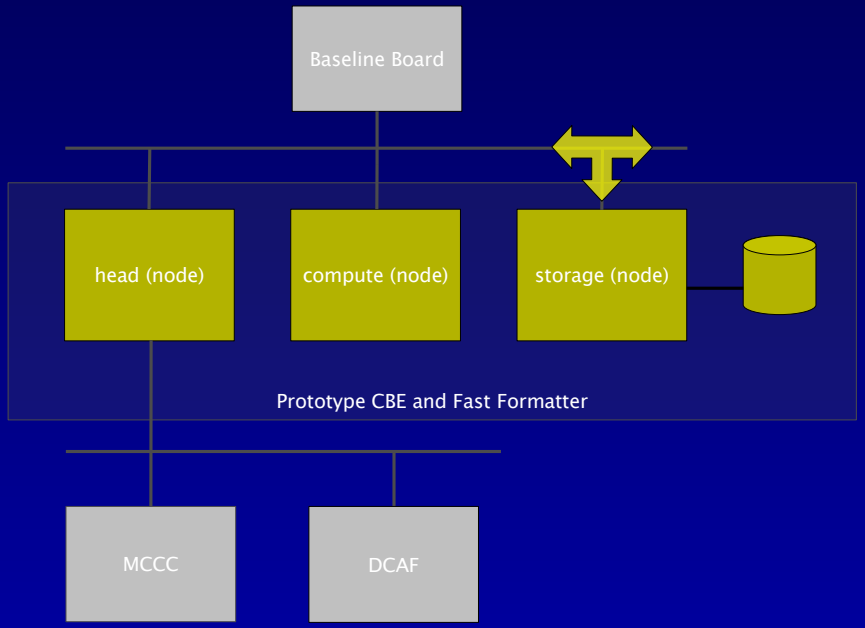
- interface to M & C
- head node

DCAF interface



- SDM (science data model) data
- write-only
- head node

Archive interface



- correlator binary data format files
- storage node
- head node for signalling

Head node processes

- Control process
  - spawn Monitor process
  - communicate with MCCC
  - configure backend for sub-scan

Head node processes

- Control process
  - spawn Monitor process
  - communicate with MCCC
  - configure backend for sub-scan
- Monitor process
  - monitor compute node status and performance
  - send monitor data to Control process

Head node processes

- Control process
  - spawn Monitor process
  - communicate with MCCC
  - configure backend for sub-scan
- Monitor process
  - monitor compute node status and performance
  - send monitor data to Control process
- Output process
  - initialize correlator output binary data format files, and write headers
  - send data (*e.g.*, science data model main table) to DCAF

Compute node processes

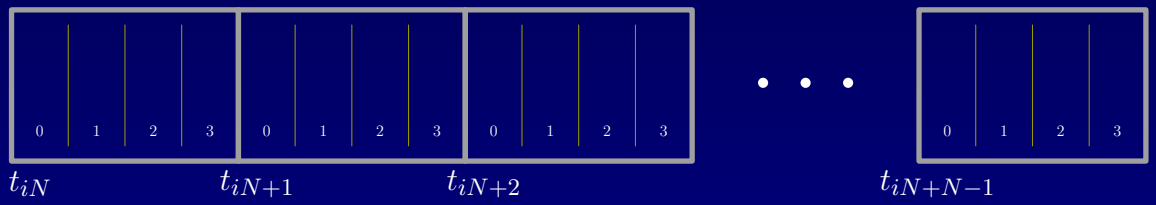
- Input process
  - configure compute node for sub-scan
  - spawn lag set processing processes
  - receive lag frames
  - sort lag frames into lag sets

Compute node processes

- Input process
  - configure compute node for sub-scan
  - spawn lag set processing processes
  - receive lag frames
  - sort lag frames into lag sets
- Lag set processing processes (`lagset_proc`)
  - data reduction of lag sets including normalization, windowing, Fourier transformation, integration
  - write data to binary data format files

Single-threaded, event-driven design

event	**action**
receive data on command socket	input XML document data, parse when document is complete
SIGINT, SIGTERM, SIGHUP	set flag for termination
SIGCHLD	clean up resources associated with child
low frequency timer	act on termination flag
receive UDP packet on lag frame input socket	sort lag frames as they are read from socket, continue reading until no packets are available



- Each element of a lag set sequence contains a partially decoded lag frame.
- Lag sets in the lag set sequence are indexed by time.
- Lag frames within a lag set are indexed by an offset.
- Each lag set sequence is implemented in shared memory accessible by a single lag set processing process.

Sort, *v. tr.*

1. To map a lag frame to a lag set and frame offset in a lag set sequence.

Sort, *v. tr.*

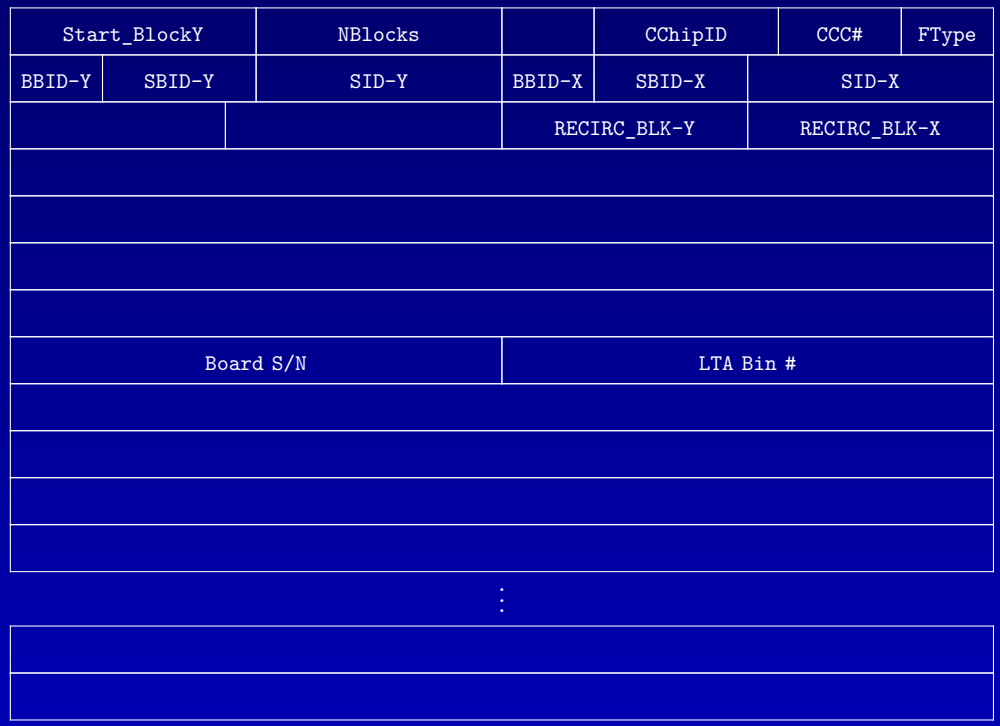
1. To map a lag frame to a lag set and frame offset in a lag set sequence.

Start_BlockY		NBlocks	nlags	CChipID	CCC#	FType
BBID-Y	SBID-Y	SID-Y	BBID-X	SBID-X	SID-X	
STATUS_BITS		FRAME_COUNT	RECIRC_BLK-Y		RECIRC_BLK-X	
TIMESTAMP-0						
TIMESTAMP-1						
DVCOUNT-Center						
DVCOUNT-Edge						
Board S/N			LTA Bin #			
Lag0-In_phase accumulator						
Lag0-Quadrature accumulator						
Lag1-In_phase accumulator						
Lag1-Quadrature accumulator						
⋮						
Lag127-In_phase accumulator						
Lag127-Quadrature accumulator						

LTA data frame

Sort, *v. tr.*

1. To map a lag frame to a lag set and frame offset in a lag set sequence.



Lag frame sorting fields

Lag frame (masked) header words used as sorting key

Start_BlockY						NBlocks							CChipID			CCC#	FType
BBID-Y			SBID-Y			SID-Y			BBID-X			SBID-X			SID-X	
									RECIRC_BLK-Y						RECIRC_BLK-X	
Board S/N									LTA Bin #								

Lag frame (masked) header words used as sorting key

<table>
<tr><td colspan="2">Start_BlockY</td><td colspan="2">NBlocks</td><td></td><td colspan="2">CChipID</td><td>CCC#</td><td>FType</td></tr>
<tr><td>BBID-Y</td><td colspan="2">SBID-Y</td><td colspan="2">SID-Y</td><td>BBID-X</td><td colspan="2">SBID-X</td><td colspan="2">SID-X</td></tr>
<tr><td colspan="5"></td><td colspan="2">RECIRC_BLK-Y</td><td colspan="2">RECIRC_BLK-X</td></tr>
<tr><td colspan="5">Board S/N</td><td colspan="4">LTA Bin #</td></tr>
</table>

Mapping of header words to (lag set sequence, frame offset) pair implemented using a hash table.

- Avoids wasted memory required by an array
- Unstructured (that is, no uniform relations required between number of, *e.g.*, subbands per baseband, or spectral channels per subband)
- Time used for lookup is not a significant factor
- Mapping can be done without decoding the header fields

Lag frame (masked) header words used as sorting key

<table>
<tr><td colspan="2">Start_BlockY</td><td colspan="2">NBlocks</td><td></td><td colspan="2">CChipID</td><td>CCC#</td><td>FType</td></tr>
<tr><td>BBID-Y</td><td colspan="2">SBID-Y</td><td colspan="2">SID-Y</td><td>BBID-X</td><td colspan="2">SBID-X</td><td colspan="2">SID-X</td></tr>
<tr><td colspan="5"></td><td colspan="2">RECIRC_BLK-Y</td><td colspan="2">RECIRC_BLK-X</td></tr>
<tr><td colspan="5">Board S/N</td><td colspan="4">LTA Bin #</td></tr>
</table>

Mapping of header words to (lag set sequence, frame offset) pair implemented using a hash table.

- Avoids wasted memory required by an array
- Unstructured (that is, no uniform relations required between number of, *e.g.*, subbands per baseband, or spectral channels per subband)
- Time used for lookup is not a significant factor
- Mapping can be done without decoding the header fields

Mapping of lag frame to lag set within a lag set sequence is done by simple timestamp-based indexing of the lag set sequence array.

`lagset_proc` process

Single-threaded, event-driven design

event	**action**
SIGINT, SIGTERM, SIGHUP	set flag for termination
low frequency timer	act on termination flag
lag set scan timer	scan lag set sequences for lag sets to process

Lag set processing

- scan sequence(s) for complete and late lag sets
- "lock out" complete and late lag sets
- process complete lag sets
  - normalization
  - windowing
  - Fourier transform
  - integration
    - ⋆ write output when integration is complete
- "unlock" near-term future lag sets

Role

Produce data stream in EVLA correlator output binary data format (MIME format with XML header and binary attachments).

Role

Produce data stream in EVLA correlator output binary data format (MIME format with XML header and binary attachments).

Requirements

- Create BDF data stream(s).
  - Create MIME data stream, and write XML header.
  - Assemble output data from `lagset_proc` processes in a single data stream.
- Send data stream to archive.

Role

Produce data stream in EVLA correlator output binary data format (MIME format with XML header and binary attachments).

Requirements

- Create BDF data stream(s).
  - Create MIME data stream, and write XML header.
  - Assemble output data from `lagset_proc` processes in a single data stream.
- Send data stream to archive.

Implementation

- Files are produced in the EVLA correlator output binary data format on a parallel file system.
- `lagset_proc` processes write visibilities, *etc.*
- Head node output process creates files, and writes the MIME and XML header data.
- Head node ships the completed files to archive.

Implementation

Input and `lagset_proc` processes on compute node.

Implementation

Input and `lagset_proc` processes on compute node.

Test environment

- Two, dual 3 GHz processor machines connected by gigabit Ethernet
- One (compute) node for CBE processes, one node for frame simulator, no CBE head node, no storage nodes
- Lag sets consisting of a single lag frame
- Data processing comprising normalization, (time-domain) windowing, Fourier transform, integration and ASCII file output

Implementation

Input and `lagset_proc` processes on compute node.

Test environment

- Two, dual 3 GHz processor machines connected by gigabit Ethernet
- One (compute) node for CBE processes, one node for frame simulator, no CBE head node, no storage nodes
- Lag sets consisting of a single lag frame
- Data processing comprising normalization, (time-domain) windowing, Fourier transform, integration and ASCII file output

Performance

- Achieved frames rates of up to approximately 106,000 frames per second (*i.e.*, $\sim 94\%$ of gigabit ethernet total bandwidth.)
- Processor utilization

process	**CPU utilization**
input	$\sim 95\%$
lagset_proc	$\sim 45\%$