SAND2013-8085P

STK Transition Team

4.29.8

Sprint Review

Outline

- This sprint's goal:
  Correct issues uncovered with early adoption of STK Mesh
- Sprint 4.29-8 backlog
  - consistent connectivity API
  - excessive memory use at scale
  - optional upward connectivities
  - user support
- Results
- Next sprint…

STK Mesh Connectivity API

Background:

STK mesh upgrade activity was recently completed, however changes to API were recommended by SM team (as part of their removal of backward compatibility layer)

Story:

Document and socialize a proposal for a consistent connectivity API.

Story:

Implement & refactor apps to use new API

STK Mesh Connectivity API

- API for connectivity access socialized and documented
- All methods to access "others" removed or made private to STK mesh classes
- On **BulkData**:

  ```
  Entity const* begin_XXXs(Entity entity) const;
  ```

  where XXX is [node, edge, face, element]

- On **Bucket**:

  ```
  Entity const* begin_YYYs(size_type bucket_ordinal)
      const;
  ```

  where YYY is [node, edge, face, element]

- Other connectivity types (e.g. constraints) may be accessed through equivalent `begin(…, EntityRank rank)` methods
- Feedback from teams indicates work to be done on constraint support, additional methods for other relation types, and more examples

STK Mesh Connectivity API Implementation

- STK Mesh connectivity API implemented in all applications
- Minimal changes required to remove “others” methods from public API
- Removing from **BulkData** required changes only to Agio_PeriodBC_Base.C
- Removing from **Bucket** required only a change from public to private methods (internal macro-defined methods still call these)
- Runtime of performance tests assessed before and after changes (about 80 remained after culling short, failing, and inconsistent tests):

Performance Tests	% Change
Average	-1.72
Worst	0.92
Best	-11.97

This is a result of fragile Redsky testing.

Excessive memory use at scale

Background:

FY13 ASC L2 Milestone Nalu scaling w/ Trilinos uncovered issues with excessive memory use at large scale.

Team surmised that distributed index should be investigated.

Story:

Decisively instrument & collect data on excessive memory use.

Identify root cause & propose solutions.

Nalu memory profiling – weak/strong scaling

Instrument & collect data on excessive memory use on the Nalu problem that identified scaling issue.

Strong scaling:

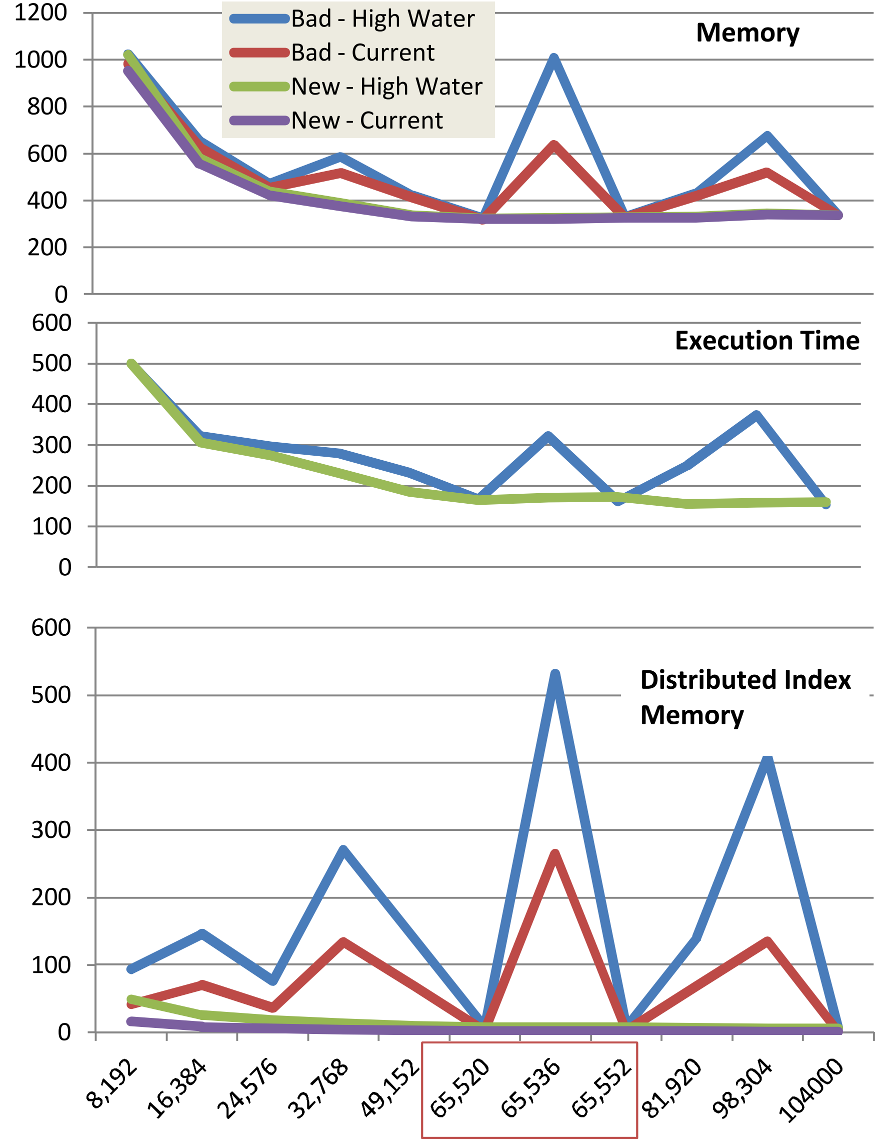
EdgeOpenJet Nalu test on cielo. 1.1 Billion Elements

- From: 8,192 processors – 137,000 elem/proc
- Up to: 104,000 processors – 10,780 elem/proc

- Use "Tracking Allocator" to gather detailed data
- Identified memory/communication issue if #proc = 2^N
  - Edge ID numbering algorithm was overloading Distributed Index containers on lower numbered processors (see next slide)
  - Added workaround which significantly reduced memory use
  - Need to "fix" underlying issue with Distributed Index.

Nalu memory profiling – strong scaling

NOTE: All memory numbers are “max processor”
“Bad” uses unmodified edge id formula
“New” uses edge id formula with workaround



Simplified Illustration of Edge ID/DI issue

- Processor responsible for Entity Key (~ID):
  - Each processor has several bins of size 256 to store keys.
  - Responsible Processor = (key/256) % #proc
- When creating edges, need unique key/id. Each processor was using this formula:
  - EdgeID = (rank+1) << 32 + 1..2..3..
  - If #proc = 2^N, then first 256 ids/edges on **each** processor assigned to processor 0
- New formula:
  - EdgeID = (rank+1) << 32 + (256 * rank) + 1..2..3..
  - Each processor more likely to be responsible for local edges.
  - More evenly distributes edges in worst case
  - Only slightly less even distribution in “best” case
  - Much less communication in most cases.
  - CON: EdgeID formula knows about internals of Distributed Index algorithm.

Distributed Index	max	min	median	
65536 - old	532.3	4.865	5.035	high water
	265.2	1.084	1.149	current
65536 - **new**	8.058	5.708	7.243	high water
	2.272	1.900	2.035	current
65552 - old	7.930	5.712	7.253	high water
	2.213	1.895	2.036	current
65552 - **new**	8.019	5.756	7.264	high water
	2.235	1.933	2.036	current

Nalu memory profiling – strong scaling

Ratio Peak/Current	Ratio of **peak to current** processor.										
total	1.01	1.00	1.00	1.00	1.01	1.01	1.01	1.01	1.02	1.03	1.03
distributed index	3.12	3.15	3.17	3.23	3.35	3.59	3.55	3.59	3.77	4.10	4.29
fields	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
entity comm	1.00	1.00	1.00	1.06	1.00	1.00	1.00	1.00	1.00	1.03	1.00
fixed connectivity	1.41	1.37	1.34	1.33	1.29	1.30	1.30	1.31	1.28	1.29	1.30
dynamic connectivity	2.00	1.97	1.99	2.02	2.10	2.08	2.10	2.10	2.14	2.18	2.18
%Total (Peak)	Percentage of each allocation type to the total (peak)										
distributed index	0.16	0.14	0.13	0.13	0.12	0.13	0.13	0.13	0.12	0.13	0.14
fields	0.33	0.33	0.32	0.32	0.31	0.31	0.31	0.31	0.31	0.30	0.31
entity comm	0.20	0.23	0.25	0.28	0.27	0.27	0.28	0.28	0.29	0.31	0.28
fixed connectivity	0.09	0.09	0.08	0.08	0.08	0.08	0.08	0.08	0.07	0.07	0.07
dynamic connectivity	0.67	0.65	0.63	0.62	0.62	0.61	0.61	0.62	0.60	0.60	0.61
%Total (Current)	Percentage of each allocation type to the total (current)										
distributed index		0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.03	0.03	0.03
fields	0.34	0.33	0.32	0.32	0.32	0.32	0.32	0.32	0.31	0.31	0.32
entity comm	0.20	0.23	0.25	0.26	0.28	0.28	0.28	0.28	0.30	0.30	0.29
fixed connectivity	0.07	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
dynamic connectivity	0.34	0.33	0.32	0.31	0.30	0.30	0.30	0.30	0.29	0.28	0.29
	8k	16k	24k	32k	48k	64k-16	64k	64k+16	80k	96k	101k

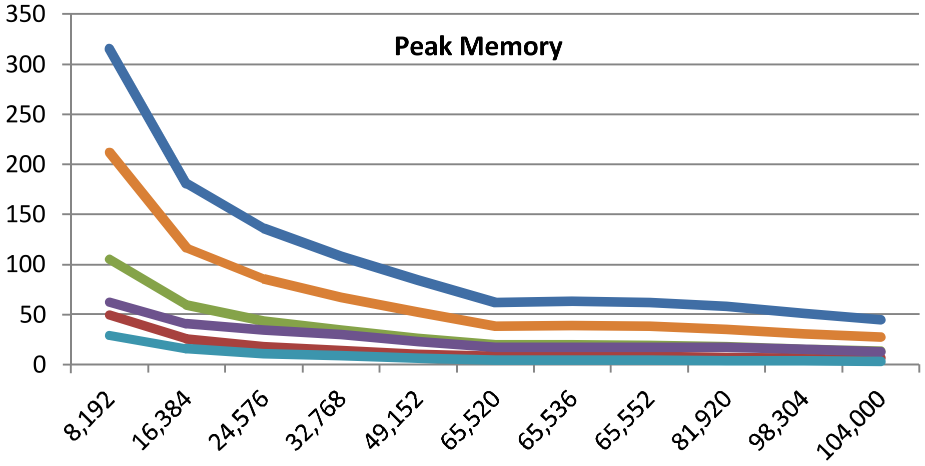
Good peak/current

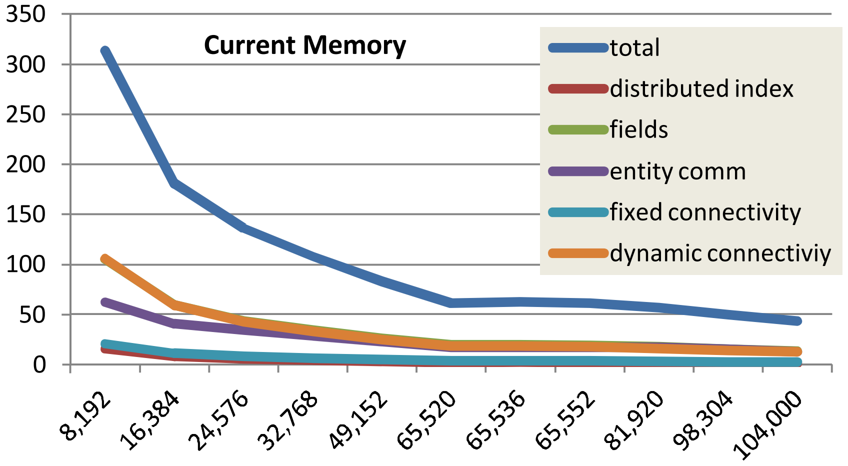
Bad? Scaling

High/Medium peak/current ratio

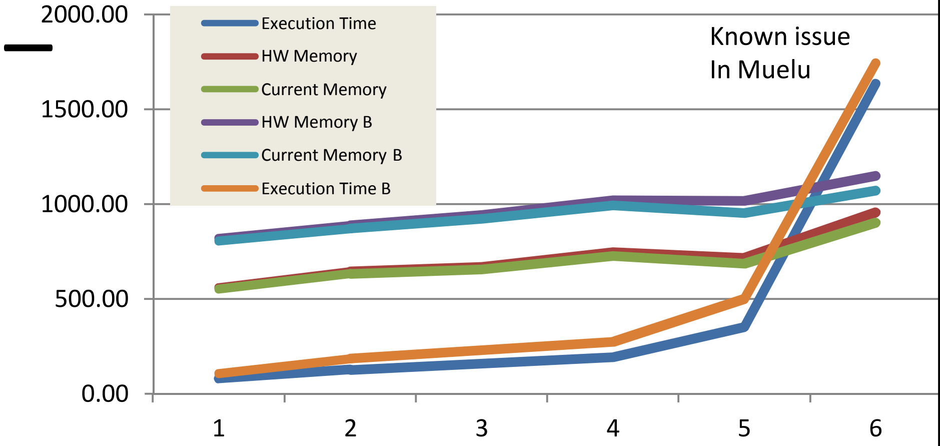
High peak/total ratio

Reasonable?





Nalu memory profiling – weak scaling



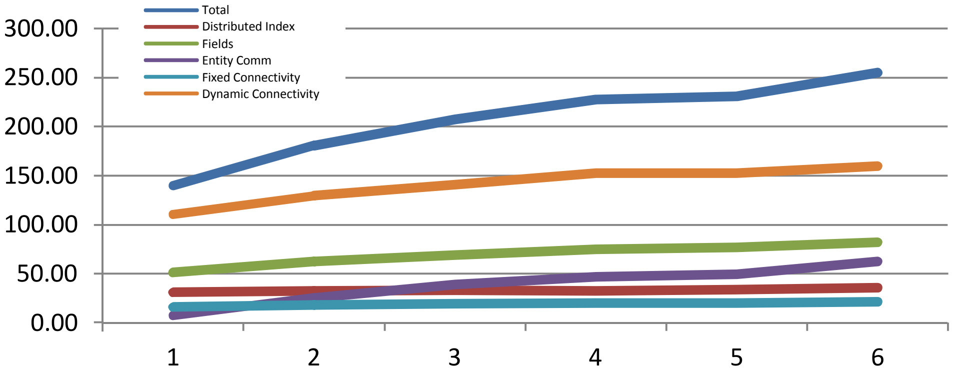
Known issue
In Muelu

Weak Scaling – 2 sets of runs

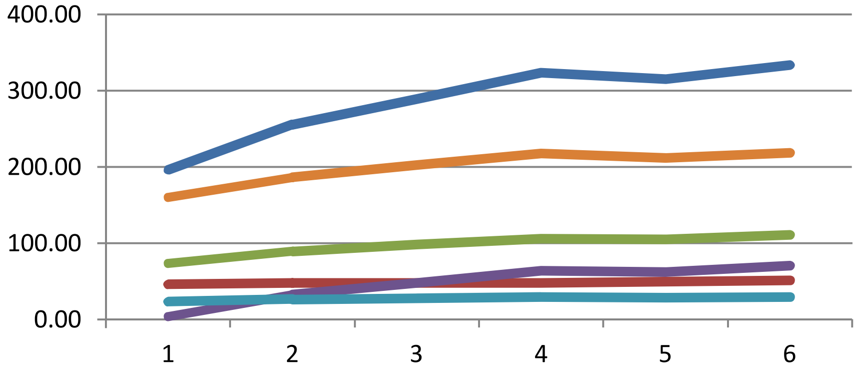
A: 3, 24, 192, 1536, 12288, 98304 (91,240 el/p)

B: 2, 16, 128, 1024, 8192, 65536 (136,860 el/p)

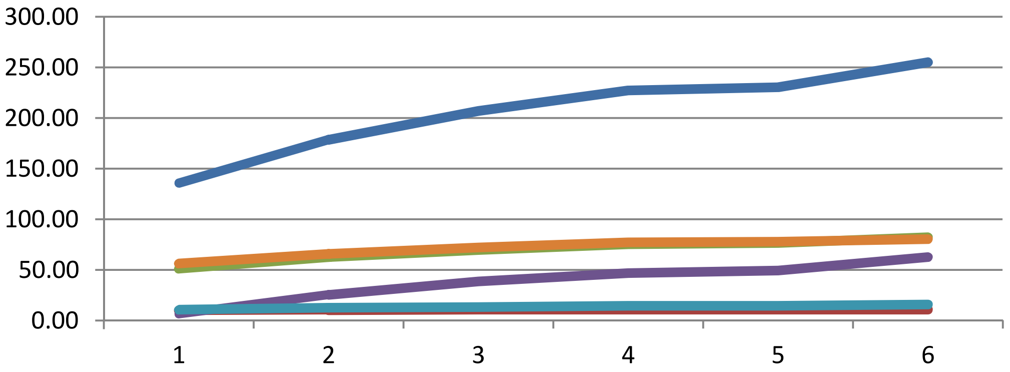
High Water Memory A



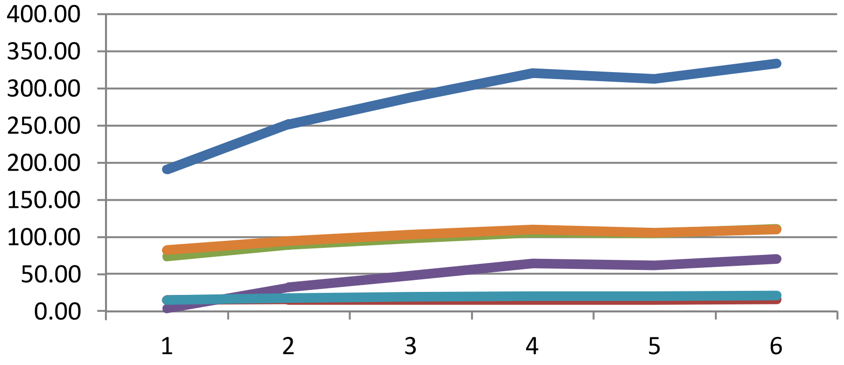
High Water Memory B



Current Memory A



Current Memory B



Excessive memory use at scale: Nalu memory profiling – weak/strong scaling

- Identified and “fixed” issue with edge id/Distributed Index interaction
  - Can now run larger jobs more reliably
- Identified algorithms with large peak/current memory use
  - story added to investigate Dynamic Connectivity
- Identified algorithms with unexpected memory usage and/or growth.
  - Known issue in Muelu
- Tracking allocator very useful in identifying memory usage patterns.

Optional upward connectivities

Background:

Part of the recent STK mesh upgrade work proposed (but didn’t implement) a memory saving option to conditionally store upward connectivities.

Story:

Implement option to store upward connectivity & determine impact on performance (run-time & memory use).

Optional Upward Connectivity

- High level
  - STK used to require that every “downward” (higher to lower topology) connectivity be matched with an “upward” connectivity
  - Upward connectivity can cost significant memory to maintain
  - Some problems have no need for certain types of upward connectivity
    - E.g. edge-based nalu problems do not need Edge->Element connectivity
  - Node->Element connectivity is mandatory, allows any type of upward connectivity to be computed if it is not stored

Optional Upward Connectivity

- What we did
  - Refactor STK products to break the assumption that upward connectivity is always stored
    - Encapsulate connectivity retrieval into a new function “get_connectivity”; this func works regardless of whether connectivity is stored or needs to be computed
      - App code should not, in general, need to use this. If an app uses certain connectivity types, it’s best to store it.
    - Identify which algorithms required upward connectivity
      - E.g. induced parts, aura maintenance
    - Have these algorithms use get_connectivity
  - Run certain tests to quantify memory savings when some upward connectivity is not stored
  - Run all tests to quantify overhead of get_connectivity with default settings (all upward conn stored)

Optional Upward Connectivity

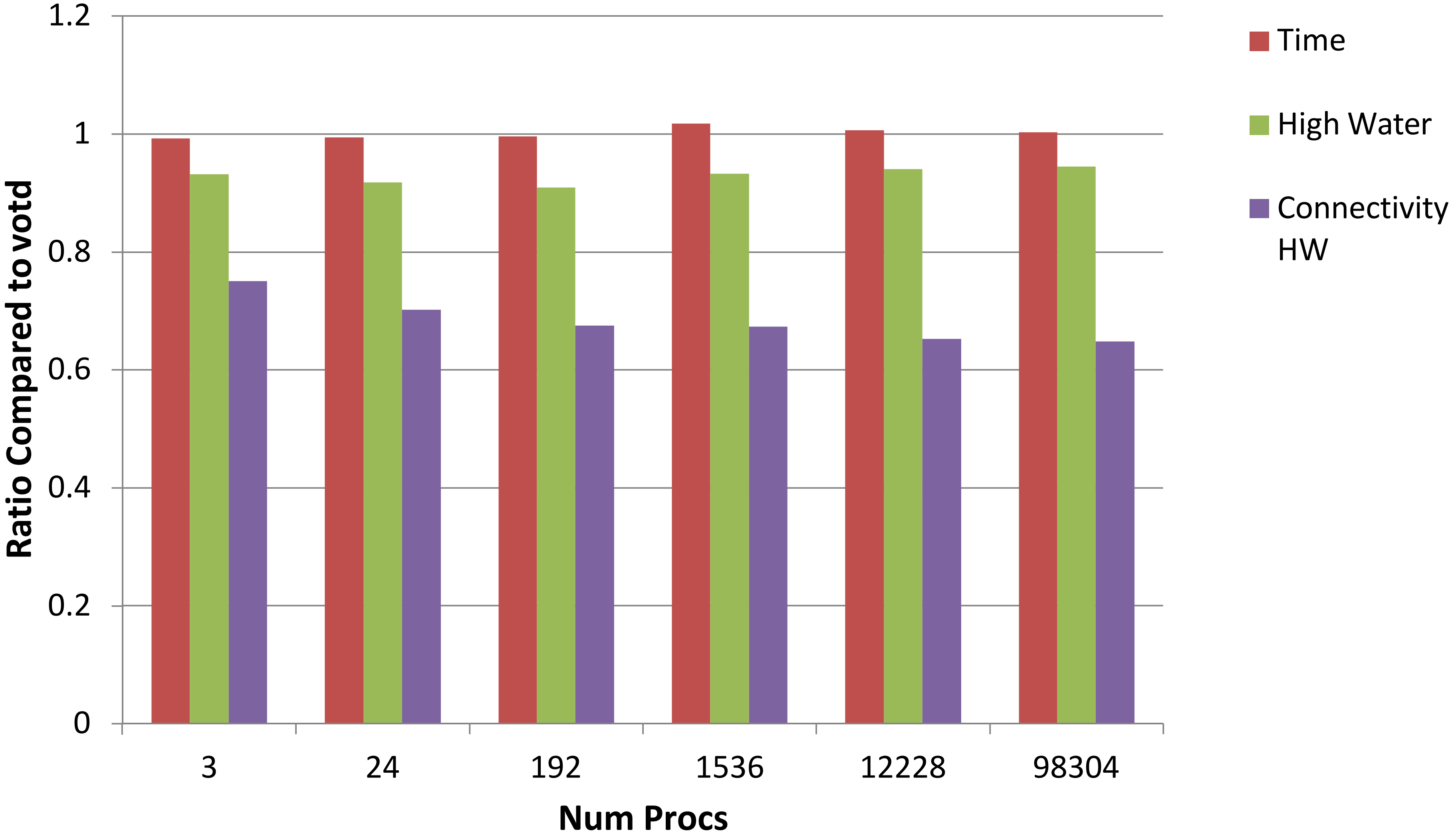
- The Test
  - Nalu edgeOpenJet weak scaling
  - 91,240 elements per proc
  - Connectivity choices: [on: regular, irregular
    off: invalid]

		To			
		Node	Edge	Face	Element
From	Node	-	off	off	on (irregular)
	Edge	on (regular)	-	off	off
	Face	on (regular)	on (irregular)	-	on (irregular)
	Element	on (regular)	on (irregular)	on (irregular)	-

Always required

Required for Nalu

Optional Upward Connectivity



Optional Upward Connectivity

- Future work
  - Try to quantify overhead of get_connectivty in default case (all connectivity left on, traditional behavior)
    - Redsky runs revealed average 1% slowdown, with 5+% slowdown on a handful of aplication performance tests
    - We could not duplicate this slowdown on our blades
    - We did see a small ~3% slowdown for our internal stk performance tests
    - As a result, this work has not been pushed
  - Disabling upward connectivity is not available to Fmwk-based apps due to widespread assumptions in Fmwk (and other) layers
    - If there is demand for this feature, it will require additional stories

Performance diagnostic tools

Background:

STKT team is committed to measuring impact of any significant work on Apps performance (run-time & memory use).

Also, we want to better inform FY14 performance milestone improvements with data (hot spots, etc…)

Story:

Evaluate Valgrind Dynamic Heap Analysis Tool

Story:

Develop a tool to ‘automatically’ gather App teams’ Performance Test Suite (PTS) data.

Memory Profiling Tool: DHAT

- DHAT – Valgrind **D**ynamic **H**eap **A**nalysis **T**ool

```
max-live:    4,834,680 in 1 blocks
 tot-alloc:  11,347,360 in 4 blocks (avg size 2836840.00)
 deaths:     4, at avg age 44 (0.00% of prog lifetime)
 acc-ratios:  0.00 rd, 0.00 wr  (0 b-read, 0 b-written)
    at 0x4A054E8: malloc (vg_replace_malloc.c:270)
    by 0xE55629: ML_matmat_mult (ml_matmat_mult.c:1156)
======================================================
   Bvals = NULL;
   while (Bvals == NULL) {
     lots_of_space++;
     Bvals     = (double *) ML_allocate( B_allocated * sizeof(double));
     B_allocated /= 2;
   }
   ML_free(Bvals);
```

Probably OK, but…

- Find alignment holes in classes.
- Find unused memory
- Find uninitialized memory
- Find quick new/delete loops
- Find unevenly accessed class data
- Cache use…

```
Aggregated access counts by offset:
[    0]  701396 701396 701396 701396  59916  59916  59916  59916
[    8]     7834     7834     7834     7834        0         0         0        0
[  16]  111998 111998 111998 111998 111998 111998 111998 111998
[  24]  23498 23498 23498 23498 23498 23498 23498 23498
[  32]  0 0 0 0 0 0 0 0

typedef struct ML_SuperNode_Struct
{
   int    length;             // Size 4
   int    maxlength;          // Size 4
   int    index;              // Size 4     + 4 byte alignment hole
   int    *list;              // Size 8
   struct ML_SuperNode_Struct *next;   // Size 8
   Where are the last 8 bytes?
} ML_SuperNode;

#define ML_allocate(i)    malloc((i + sizeof(double) ))
```

Should "*list" be moved up since accessed frequently…

```
------------------- 160 of 200 -------------------
max-live:    284,256 in 1 blocks
tot-alloc:   284,256 in 1 blocks (avg size 284256.00)
deaths:      1, at avg age 11,878,118,517 (75.02% of prog lifetime)
acc-ratios:  0.00 rd, 2.00 wr  (0 b-read, 568,512 b-written)
   at 0x4A0631F: operator new(unsigned long) (vg_replace_malloc.c:298)
   by 0xAC1F73: std::vector<long,..) (vector.tcc:481)
   by 0x173CE59: Ioss::Map::build_reorder_map(long, long) (stl_vector.h:1004)
   by 0x145C081: Ioex::DatabaseIO::handle_node_ids(void*, long) const
```

Code rewritten to eliminate when not needed

Performance Testing Script

- Performance testing on Redsky is difficult and labor intensive
- Typically the following steps are performed:

  1) Build on sierra0XX for Intel 11 MPI 1.4.2
  2) Create empty directory on Redsky
  3) Copy bin directory and underlying structure to empty directory on Redsky
  4) Assign performance tests to assigned tests file in Redsky directory
  5) Run tests from Redsky directory multiple times

  ```
  do for i in $(seq 5); do ./redsky_run.sh ; done
  ```

  where `redsky_run.sh` contains the appropriate `testrun` command

- Repeat for every commit to be compared: ~10 hrs each
- `compare_test_result_timings.py` used to compare to multiple sets of results
- A script to automate steps 1 – 5 above for two different commits has been created and placed in the contrib scripts directory:

```
performanceDiff.py --mod XXXXXX --ref YYYYYY --run_directory
    /path_to_directory_on_redsky/ --tests_directory
  /path_to_tests_on_redsky/ --redsky_account FYZZZZZZ
```

User-Support Tickets

Defect Tickets:

- 24 tickets closed
- 28 touched and still open
- 9 new tickets opened
- 40 defect tickets total open (reduced from 56 previous end of sprint)
- 10 pending verification
- 2 Navy, 4 Goodyear, 1 JPL, 2 KCP, 43 Sandia
- 6 application code related, 3 framework, 26 toolkit, 12 Seacas, 5 “other”

Enhancement Tickets:

- 11 enhancement/backlog tickets total open
- 1 new enhancement ticket (feature request)
- 3 tickets touched (none closed)

Support evaluation of Basic STK Restart Capability

```
mesh_data.add_restart_field(fields[i], name);   // Optional db
                                                  field name

mesh_data.create_restart_output(restart_filename);

mesh_data.define_restart_fields();
… in loop: mesh_data.process_restart_output(time);  // Can also
                                                  specify
'step'

mesh_data.process_restart_input(step);  // Can also specify
                                          'time'
```

Stefan implemented in Nalu; after some debugging (stk_io problem), we verified that it is working

Basic STK Restart Capability

However, additional capability is (potentially) needed:

- Verify that field state support is working correctly.
- Handle multii-state fields correctly.
- Non-field data. (Parameters, Resources)
- Investigate alternative ways to declare field for restart. (Persistent)
- What can change during a "restart"
- Better "auto restart" options for process_restart_input()
- Robustness:
  - Consistency over all processors
  - Was field found on restart database
  - Others…
- Functionality:
  - Examine framework restart capability and see what is needed
  - Scheduling, Callbacks, Mangling, Overwrite, Concatenation, Topology Change,
  - Parsing / Interface to user

Next Sprint, 4.29-9

- prepare for 4.30 Release
  - Clean dashboard
  - User support as needed by Apps (SM 2 tickets, TF 1 ticket)
  - Documentation
- Kokkos tutorial
- Optional upward connectivity capability ??
- Begin SD/SM consolidation of preload followed by modal transient capability for mid October demonstration
- Create conchas2, prepare for early FY14 transition to STK