High Performance Computing

GPGPU efforts at Los Alamos, Status and Tool needs

- plus, CBTF for monitoring efforts

David Montoya

Input from: Laura Monroe, Phil Romero, Bob Robey, Scott Pakin, Pat McCormick, Mike Mason

CScADS 2012

6/26/12

Los Alamos NATIONAL LABORATORY

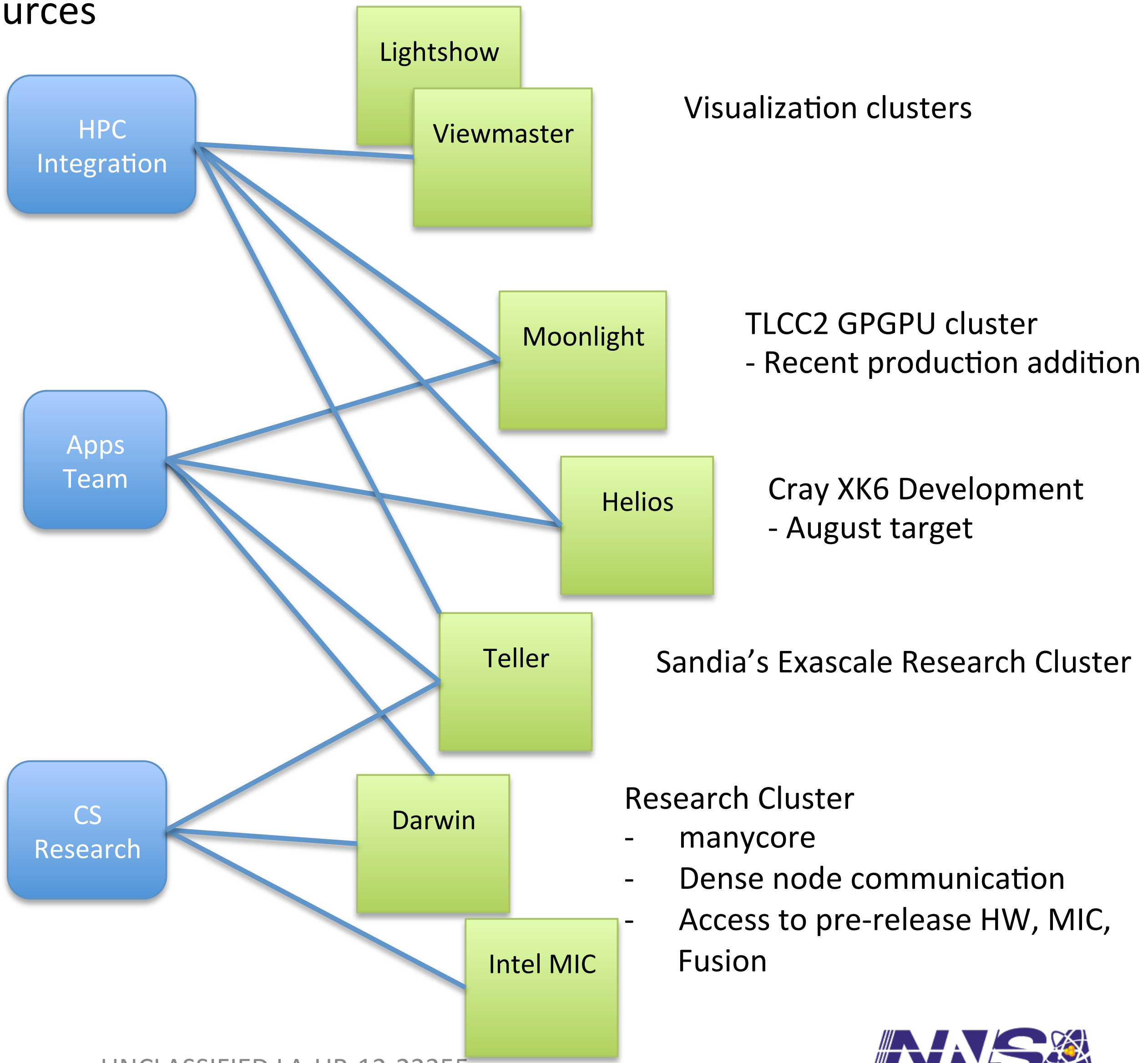
National Nuclear Security Administration

Resources

- Testing TLCC2 machines with GPGPUs
- Working out libraries and drivers to support initial production implementation into TOSS
- Working on visualization capability

- DoE "mini-app" developed to facilitate collaboration between hardware and software development

- Tools to identify code transition to GPGPUs
- Compiler Infrastructure research



Visualization clusters

TLCC2 GPGPU cluster
- Recent production addition

Cray XK6 Development
- August target

Sandia's Exascale Research Cluster

Research Cluster
- manycore
- Dense node communication
- Access to pre-release HW, MIC, Fusion

High Performance Computing

Production/Development Machines

Moonlight (TLCC2 -first Production Machine)

- Intel Sandy Bridge processor + Nvidia M2090 GPU
- Each CPU has dedicated PCIe link to Tesla M2090
- Peak Compute Performance: 488 TF/s
- Compute nodes: 308
- Compute cores: 4,928 CPU + 315,392 GPU

Viewmaster (Visualization cluster)

- 92 Back-end Rendering Nodes
  - 2 NVIDIA Quadro 6000s (6 GB GDDR5 on-board memory)
  - 96 GB DDR3-1333 memory
- 12 Advanced User Nodes
  - 1 NVIDIA Quadro 6000
  - 96 GB DDR3-1333 memory
- 56 Standard User Nodes
  - 1 NVIDIA Quadro 5000 (2 GB GDDR5 on-board memory)
  - 24 GB DDR3-1333 memory
- 39 Facilities Nodes to support CAVE, PowerWall, etc
  - 1 NVIDIA Quadro 6000, with g-sync daughterboard for framelock
  - 96 GB DDR3-1333 memory

Helios – August (Cray XK6 Development machine)

- 16 blades, 4 nodes per blade = 64 nodes total.
  - 3 blades are I/O blades
  - 13 blades are compute blades
- Compute blade:
  - 4 amd G34 8 core CPU, 32 gb memory per node.
  - 4 NVIDIA 512 core GPUs (Tesla X2090) 6gb built in memory per GPU node.
- Each GPU estimated 1331 Gigaflops peak single precision floating point performance.

CS Testbeds

- Darwin
  - It has 48 AMD Opteron 6168 cores on each node,
  - with some of the nodes having 2 NVIDIA Tesla C2050 GPUs and others with 2 ATI
  - FirePro 7800 GPUs.
- Intel MIC – 7 node
- Access to Sandia's Exascale Research Cluster

Software Stack and Testing - Moonlight

GPGPU Software Stack

- Integrated into tri-lab standard TOSS stack
  - Both Moonlight and VM2 are running TOSS
  - Standard release includes CUDA toolkit 4.1, OpenCL
- Investigating further packages
  - OpenACC
  - DSLs like Liszt
- Debuggers: Allinea, TotalView
- User-specified tools: in “experimental” space
- Libraries as needed: Thrust, CUDAMat, CUSP library
- Testing: ORNL’s Scalable Heterogeneous benchmark suite, future development for monitoring and diagnostics

Testing

- Ran Nbody, SHOC test and HPL
- HPL
  - 488 TFLOPS peak
  - 1041 GFLOPS realized on a single node
  - Peak power 937 watts
- Saw several double-bit errors
  - HW replacement on those GPUs

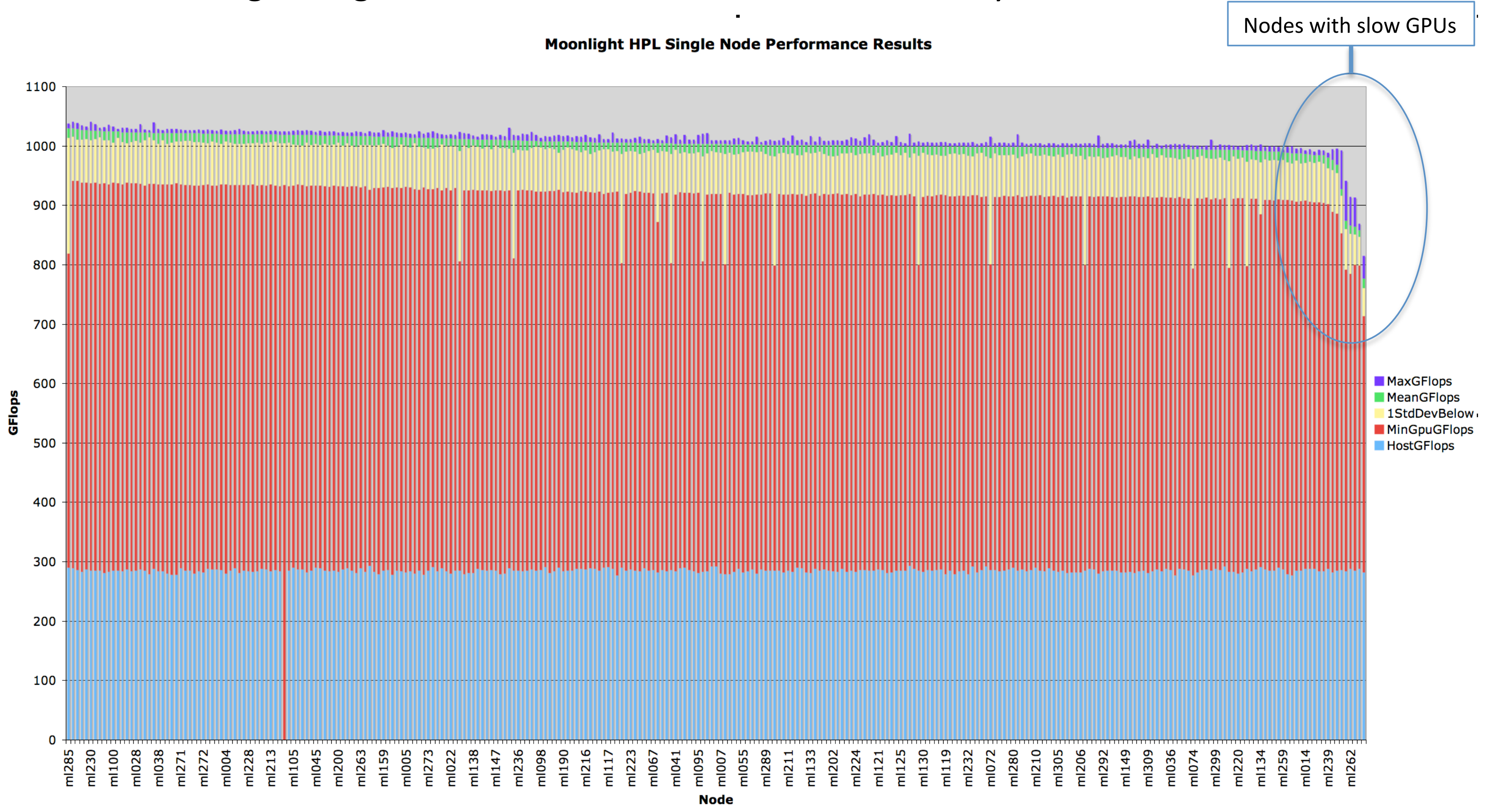
Drivers

- LANL goal: one driver to handle
  - Compute (on Moonlight)
  - GPU rendering, display framelocking (on Viewmaster 2)
- Not there yet
  - Framelocking an ongoing issue
    - There’s a fix in a very recent driver, haven’t tested
  - Earlier drivers showed some rendering performance issues on certain specific use cases
    - Later drivers show better performance

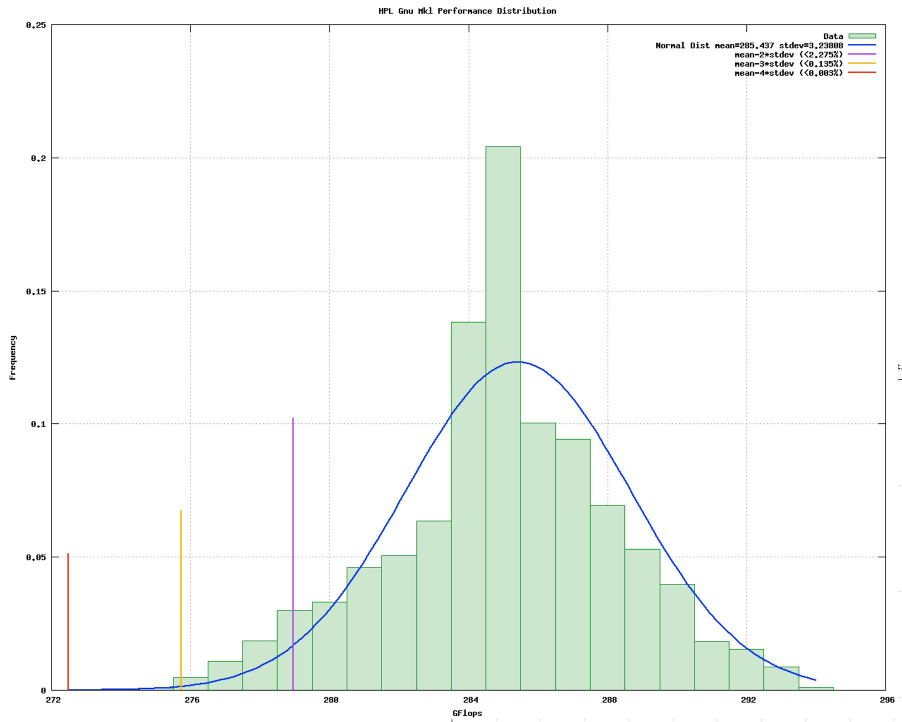
Testing Results – Phil Romero

- GPUs have different performance characteristics than CPUs
- Performance in transferring data from GPUs to CPUs varies widely even when correct affinities are utilized. Not true when transferring data from CPUs to GPUs.
- GPUs much more sensitive to memory usage on the host than CPUs, sometimes even producing non-linear results on performance.
- CPUs perform in a tight performance range approximating a normal distribution, GPUs do not.
- Three tools that monitor GPU based clusters are now under development.

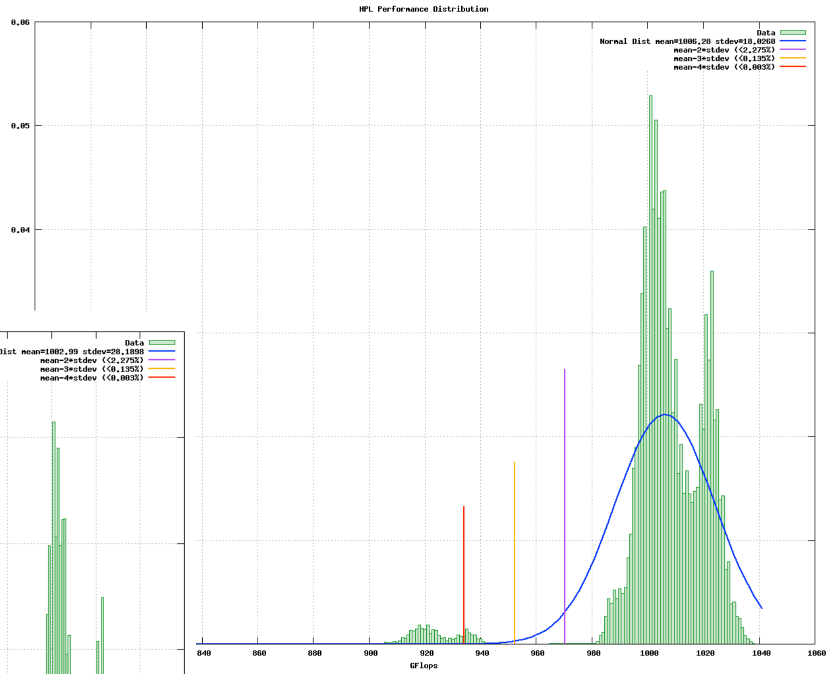
Moonlight Single Node HPL Performance Distribution by Node



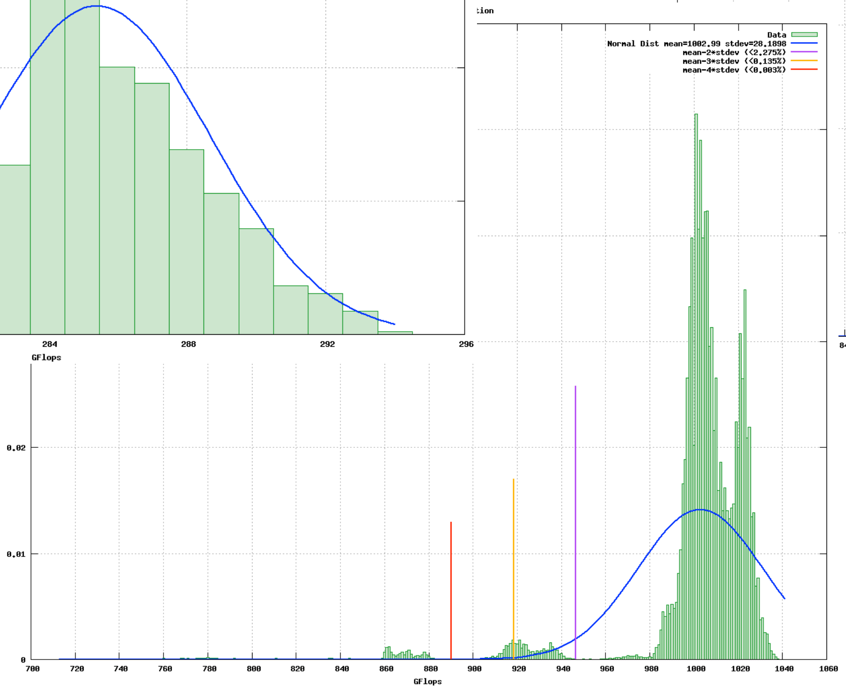
Moonlight Single Node HPL Performance Distribution Utilizing Only CPUs



Utilizing GPUs and CPUs w/o slow node GPUs



Utilizing GPUs and CPUs – w/ slow node GPUs



Monitoring GPUs with CBTF – Phil Romero

Three tools that utilize CBTF (Component Based Tool Framework) and NVML (NVIDIA Management Libraries) are currently under development:

checkGpuMemory - finds GPUs that have unrecoverable double-bit ECC errors, and monitors single-bit ECC errors. GPUs with double-bit ECC errors are defective parts that need to be exchanged. The serial number and bus slot are also identified to ensure the correct GPU is identified. Sample output is shown below:

```
FAILED Node ml048 Gpu0 2bitMemoryErrors 48 BusId 0000:03:00.0 SerialNumber 0324811024641
WARNING Node ml159 Gpu1 1bitMemoryErrors 5286 BusId 0000:81:00.0 SerialNumber 0324811023640
OK Node ml040   [ will be omitted from final output ]
```

checkGpuConfigs - finds GPU configuration outliers. Sample output is shown below:

```
283 Configurations with Vbios 70.10
2 Configurations with Vbios 70.09 ml013,ml301
284 Configurations with PersistenceMode 1
1 Configuration with PersistenceMode 0 ml273
```

checkGpuUtilization - find GPUs whose current utilization state is significantly underperforming with regard to its counterparts. This tool is designed to be run while jobs are in progress, unlike the other tools that run on idle nodes. Several different forms of outputs are possible, for example…

Identification of statistical outliers:

```
31 Nodes with GpuPowerUsage mean 895.3 kW
1 Node with GpuPowerUsage mean 413.7 kW ml083
```

- Continuously polled Graphical output of various GPU, CPU and memory utilizations by node
- Graphical output summarizing various GPU, CPU and memory utilizations by node aggregated over a single job.
- Mixing data from IPMI sources with data from NVIDIA Management Library

LANL Programming Models Research

- Improved compiler infrastructure and supporting toolchain
- Compiler-based Application Analysis – Scott Pakin

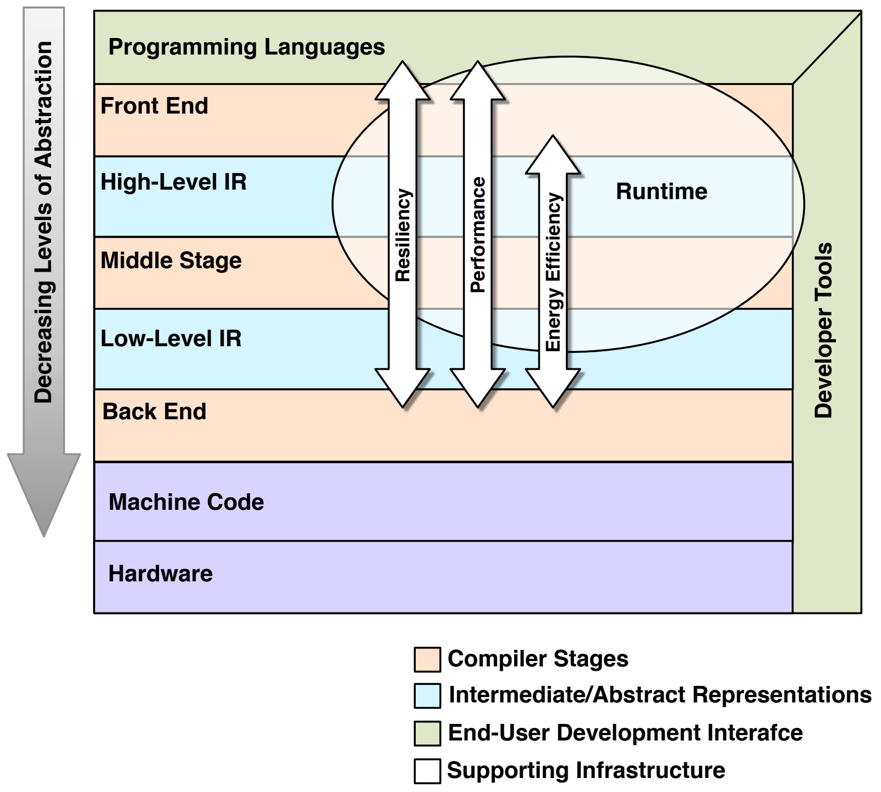
Compiler Infrastructure – Pat McCormick

The focus is a set of common intermediate representations (IR) to coordinate feedback across the entire toolchain (including runtime components). Overall design matches that of a modern compiler with front-end, middle, and back-end stages.

• **Common Representations**: Front-ends map the source language into a language-independent intermediate representation (IR) that is semantically richer (a higher level of abstraction) than is typical practice (e.g., capture concurrency, parallelism, memory locality, scheduling, data layout, etc.)

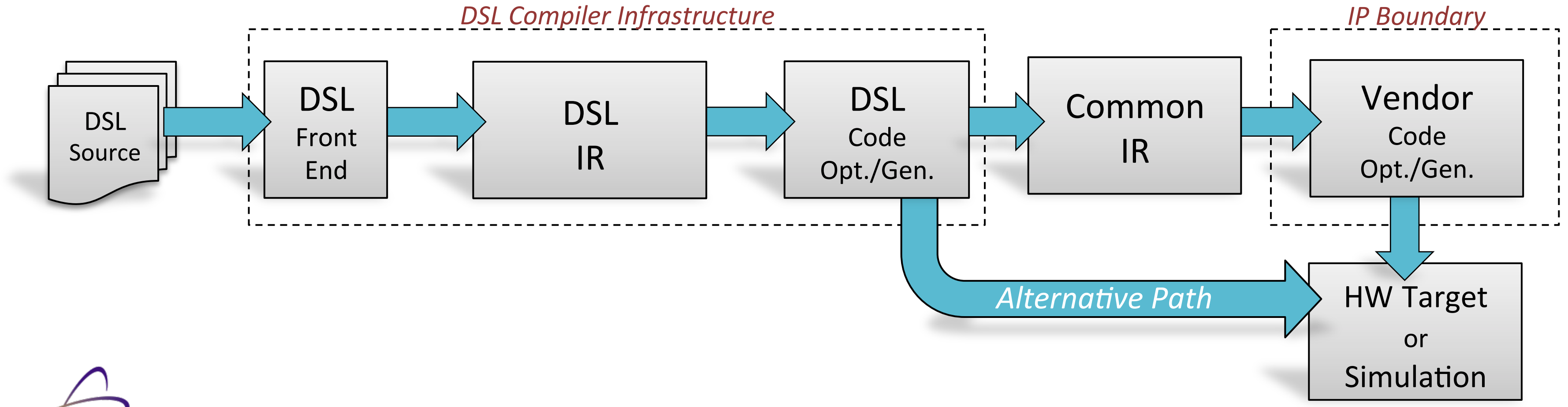
• **Middle**: The middle stage takes the HLIR as input and runs a series of analysis and optimization passes and also provides a natural access point for both static and dynamic performance analysis and debugging tools. HLIR is then lowered to LLIR (low-level IR – using LLVM IR as a building block).

• **Back-end**: The back-end includes traditional analysis steps (e.g., data and control flow) as well as lower-level generic optimizations such as dead code elimination and loop transformations. Current work matches closely with LLVM (http://llvm.org).



Example: Supporting Domain Specific Languages

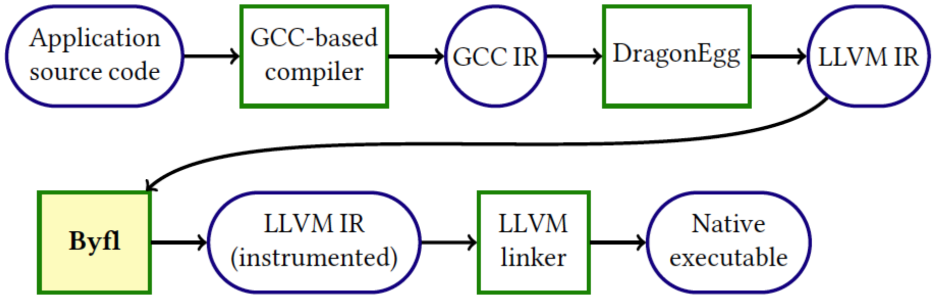
- Co-Design activities with vendors for common LLIR (low-level IR) based on LLVM IR
  - Effort underway with AMD, Intel, NVIDIA)
  - NVIDIA CUDA layers 4.x/5.0 already supported via open-source LLVM release and compiler SDK (beta)
  - AMD and Intel work in progress
- Leverage vendors' toolchain (reduced effort)
  - Also provides an alternative path for tying into future architectural simulation



Compiler-based Application Analysis – Scott Pakin

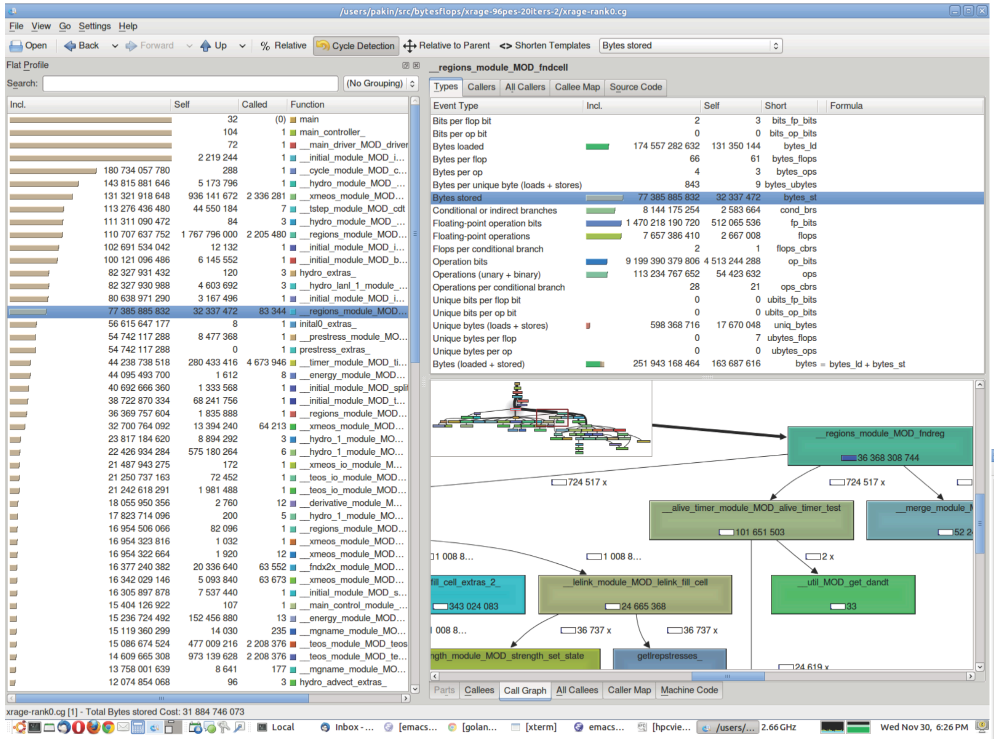
Goal: Develop tool to identify functions (or whole applications) that are likely to run well at exascale - and those that need immediate attention.

- Implement a form of software performance counter that measures application characteristic not available at the hardware level. Instrument the code at compile time, adding counters of various operations of interest.
- Byfl – initial ability to measure byte and flop counts – integrates with the LLVM compiler toolchain as a compiler pass.



```
BYFL_SUMMARY: -----------------------------------------------------------------
BYFL_SUMMARY:          7021934541 basic blocks
BYFL_SUMMARY:          5273766649 conditional or indirect branches
BYFL_SUMMARY: -----------------------------------------------------------------
BYFL_SUMMARY:        140154955078 bytes (106609718886 loads + 33545236192 stores)
BYFL_SUMMARY:           568213078 unique bytes
BYFL_SUMMARY:          5056852452 flops
BYFL_SUMMARY:         82872028639 ops
BYFL_SUMMARY: -----------------------------------------------------------------
BYFL_SUMMARY:       1121239640624 bits (852877751088 loads + 268361889536 stores)
BYFL_SUMMARY:          4545704624 unique bits
BYFL_SUMMARY:        970915667328 flop bits
BYFL_SUMMARY:       7468902667646 op bits
BYFL_SUMMARY: -----------------------------------------------------------------
BYFL_SUMMARY:              3.1781 loads per store
BYFL_SUMMARY:              0.9589 flops per conditional/indirect branch
BYFL_SUMMARY:             15.7140 ops per conditional/indirect branch
BYFL_SUMMARY: -----------------------------------------------------------------
BYFL_SUMMARY:             27.7158 bytes per flop
BYFL_SUMMARY:              1.1548 bits per flop bit
BYFL_SUMMARY:              1.6912 bytes per op
BYFL_SUMMARY:              0.1501 bits per op bit
BYFL_SUMMARY: -----------------------------------------------------------------
BYFL_SUMMARY:              0.1124 unique bytes per flop
BYFL_SUMMARY:              0.0047 unique bits per flop bit
BYFL_SUMMARY:              0.0069 unique bytes per op
BYFL_SUMMARY:              0.0006 unique bits per op bit
BYFL_SUMMARY:            246.6592 bytes per unique byte
BYFL_SUMMARY: -----------------------------------------------------------------
```

Modify the compiler to inject instrumentation code. Data are output as easy-to-parse "BYFL"-prefixed lines.

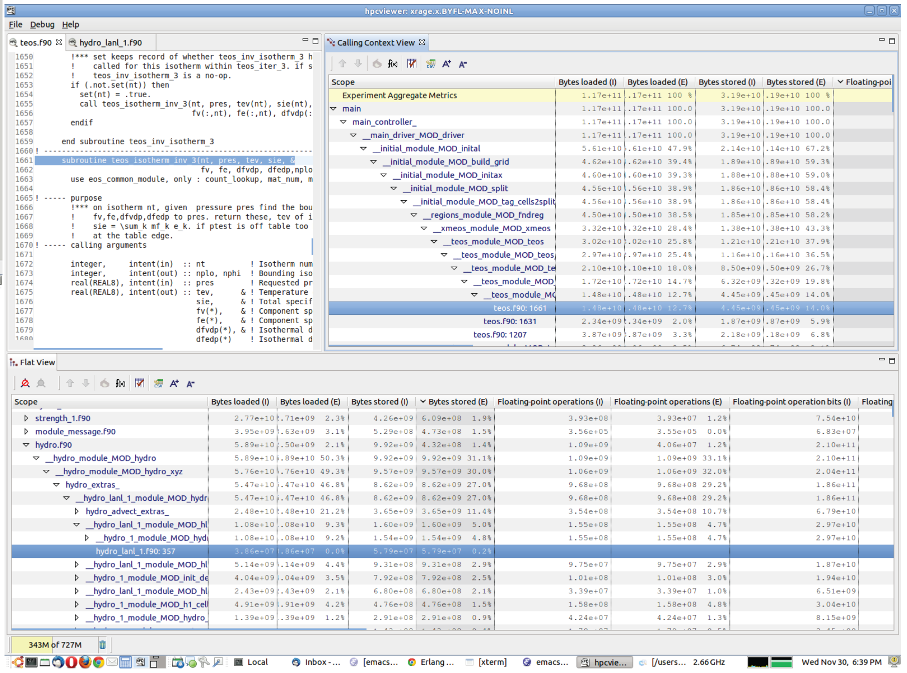

/users/pakin/src/bytesflops/xrage-96pes-20iters-2/xrage-rank0.cg
File View Go Settings Help
Open Back Forward Up Relative Cycle Detection Relative to Parent Shorten Templates Bytes stored
Flat Profile
Search: (No Grouping)
__regions_module_MOD_fndcell
Types Callers All Callers Callee Map Source Code
__regions_module_MOD_fndreg
36 368 308 744
__alive_timer_module_MOD_alive_timer_test
101 651 503
__lelink_module_MOD_lelink_fill_cell
24 665 368
__util_MOD_get_dandt
33
getirepstresses
Parts Callees Call Graph All Callees Caller Map Machine Code

Incl.	Self	Called	Function
	32	(0)	main
	104	1	main_controller_
	72	1	__main_driver_MOD_driver
	2 219 244	1	__initial_module_MOD_i…
180 734 057 780	288	1	__cycle_module_MOD_c…
143 815 881 646	5 173 796	1	__hydro_module_MOD_…
131 321 918 648	936 141 672	2 336 281	__xmeos_module_MOD…
113 276 436 480	44 550 184	7	__tstep_module_MOD_cdt
111 311 090 472	84	3	__hydro_module_MOD_…
110 707 637 752	1 767 796 000	2 205 480	__regions_module_MOD…
102 691 534 042	12 132	1	__initial_module_MOD_i…
100 121 096 486	6 145 552	1	__initial_module_MOD_b…
82 327 931 432	120	3	hydro_extras_
82 327 930 988	4 603 692	3	__hydro_lanl_1_module_…
80 638 971 290	3 167 496	1	__initial_module_MOD_i…
77 385 885 832	32 337 472	83 344	__regions_module_MOD…
56 615 647 177	8	1	inital0_extras_
54 742 117 288	8 477 368	1	__prestress_module_MO…
54 742 117 288	0	1	prestress_extras_
44 238 738 518	280 433 416	4 673 946	__timer_module_MOD_ti…
44 095 493 700	1 612	8	__energy_module_MOD…
40 692 666 360	1 333 568	1	__initial_module_MOD_split
38 722 870 334	68 241 756	1	__initial_module_MOD_t…
36 369 757 604	1 835 888	1	__regions_module_MOD…
32 700 764 092	13 394 240	64 213	__xmeos_module_MOD…
23 817 184 620	8 894 292	3	__hydro_1_module_MO…
22 426 934 284	575 180 264	6	__hydro_1_module_MO…
21 487 943 275	172	1	__xmeos_io_module_M…
21 250 737 163	72 452	1	__teos_io_module_MOD…
21 242 618 291	1 981 488	1	__teos_io_module_MOD…
18 055 950 356	2 760	12	__derivative_module_M…
17 823 714 096	200	5	__hydro_1_module_MO…
16 954 506 066	82 096	1	__regions_module_MOD…
16 954 323 816	1 032	1	__xmeos_module_MOD…
16 954 322 664	1 920	12	__xmeos_module_MOD…
16 377 240 382	20 336 640	63 552	__fndx2x_module_MOD…
16 342 029 146	5 093 840	63 673	__xmeos_module_MOD…
16 305 897 878	7 537 440	1	__initial_module_MOD_s…
15 404 126 922	107	1	__main_control_module_…
15 236 724 492	152 456 880	13	__energy_module_MOD…
15 119 360 299	14 030	235	__mgname_module_MO…
15 086 674 524	477 009 216	2 208 376	__teos_module_MOD_teos
14 609 665 308	973 139 628	2 208 376	__teos_module_MOD_te…
13 758 001 639	8 641	177	__mgname_module_MO…
12 074 854 068	96	3	hydro_advect_extras_

Event Type	Incl.	Self	Short	Formula
Bits per flop bit	2	3	bits_fp_bits	
Bits per op bit	0	0	bits_op_bits	
Bytes loaded	174 557 282 632	131 350 144	bytes_ld	
Bytes per flop	66	61	bytes_flops	
Bytes per op	4	3	bytes_ops	
Bytes per unique byte (loads + stores)	843	9	bytes_ubytes	
Bytes stored	77 385 885 832	32 337 472	bytes_st	
Conditional or indirect branches	8 144 175 254	2 583 664	cond_brs	
Floating-point operation bits	1 470 218 190 720	512 065 536	fp_bits	
Floating-point operations	7 657 386 410	2 667 008	flops	
Flops per conditional branch	2	1	flops_cbrs	
Operation bits	9 199 390 379 806	4 513 244 288	op_bits	
Operations (unary + binary)	113 234 767 652	54 423 632	ops	
Operations per conditional branch	28	21	ops_cbrs	
Unique bits per flop bit	0	0	ubits_fp_bits	
Unique bits per op bit	0	0	ubits_op_bits	
Unique bytes (loads + stores)	598 368 716	17 670 048	uniq_bytes	
Unique bytes per flop	0	7	ubytes_flops	
Unique bytes per op	0	0	ubytes_ops	
Bytes (loaded + stored)	251 943 168 464	163 687 616	bytes	= bytes_ld + bytes_st

xrage-rank0.cg [1] - Total Bytes stored Cost: 31 884 746 073

HPCToolkit view of Byfl measurements



```
      !*** set keeps record of whether teos_inv_isotherm_3 h
      !    called for this isotherm within teos_iter_3. if s
      !    teos_inv_isotherm_3 is a no-op.
      if (.not.set(nt)) then
        set(nt) = .true.
        call teos_isotherm_inv_3(nt, pres, tev(nt), sie(nt),
                                 fv(:,nt), fe(:,nt), dfvdp(:
      endif

    end subroutine teos_inv_isotherm_3
! -----------------------------------------------------------
    subroutine teos_isotherm_inv_3(nt, pres, tev, sie, &
                                   fv, fe, dfvdp, dfedp,nplo
      use eos_common_module, only : count_lookup, mat_num, m

! ----- purpose
      !*** on isotherm nt, given  pressure pres find the bou
      !    fv,fe,dfvdp,dfedp to pres. return these, tev of i
      !    sie = \sum_k mf_k e_k. if ptest is off table too
      !    at the table edge.
! ----- calling arguments

      integer,      intent(in)  :: nt           ! Isotherm num
      integer,      intent(out) :: nplo, nphi   ! Bounding iso
      real(REAL8), intent(in)  :: pres          ! Requested pr
      real(REAL8), intent(out) :: tev,       & ! Temperature
                                  sie,       & ! Total specif
                                  fv(*),     & ! Component sp
                                  fe(*),     & ! Component sp
                                  dfvdp(*), & ! Isothermal d
                                  dfedp(*)    ! Isothermal d
```

Scope	Bytes loaded (I)	Bytes loaded (E)	Bytes stored (I)	Bytes stored (E)	Floating-poi
Experiment Aggregate Metrics	1.17e+11	.17e+11 100 %	3.19e+10	.19e+10 100 %	
main	1.17e+11	.17e+11 100.0	3.19e+10	.19e+10 100.0	
main_controller_	1.17e+11	.17e+11 100.0	3.19e+10	.19e+10 100.0	
__main_driver_MOD_driver	1.17e+11	.17e+11 100.0	3.19e+10	.19e+10 100.0	
__initial_module_MOD_inital	5.61e+10	.61e+10 47.9%	2.14e+10	.14e+10 67.2%	
__initial_module_MOD_build_grid	4.62e+10	.62e+10 39.4%	1.89e+10	.89e+10 59.3%	
__initial_module_MOD_initax	4.60e+10	.60e+10 39.3%	1.88e+10	.88e+10 59.0%	
__initial_module_MOD_split	4.56e+10	.56e+10 38.9%	1.86e+10	.86e+10 58.4%	
__initial_module_MOD_tag_cells2split	4.56e+10	.56e+10 38.9%	1.86e+10	.86e+10 58.4%	
__regions_module_MOD_fndreg	4.50e+10	.50e+10 38.5%	1.85e+10	.85e+10 58.2%	
__xmeos_module_MOD_xmeos	3.32e+10	.32e+10 28.4%	1.38e+10	.38e+10 43.3%	
__teos_module_MOD_teos	3.02e+10	.02e+10 25.8%	1.21e+10	.21e+10 37.9%	
__teos_module_MOD_teos	2.97e+10	.97e+10 25.4%	1.16e+10	.16e+10 36.5%	
__teos_module_MOD_te	2.10e+10	.10e+10 18.0%	8.50e+09	.50e+09 26.7%	
__teos_module_MOD	1.72e+10	.72e+10 14.7%	6.32e+09	.32e+09 19.8%	
__teos_module_MC	1.48e+10	.48e+10 12.7%	4.45e+09	.45e+09 14.0%	
teos.f90: 1661	1.48e+10	.48e+10 12.7%	4.45e+09	.45e+09 14.0%	
teos.f90: 1631	2.34e+09	.34e+09 2.0%	1.87e+09	.87e+09 5.9%	
teos.f90: 1207	3.87e+09	.87e+09 3.3%	2.18e+09	.18e+09 6.8%	

Scope	Bytes loaded (I)	Bytes loaded (E)	Bytes stored (I)	Bytes stored (E)	Floating-point operations (I)	Floating-point operations (E)	Floating-point operation bits (I)	Floating
strength_1.f90	2.77e+10	.71e+09 2.3%	4.26e+09	6.09e+08 1.9%	3.93e+08	3.93e+07 1.2%	7.54e+10	
module_message.f90	3.95e+09	.63e+09 3.1%	5.29e+08	4.73e+08 1.5%	3.56e+05	3.55e+05 0.0%	6.83e+07	
hydro.f90	5.89e+10	.50e+09 2.1%	9.92e+09	4.32e+08 1.4%	1.09e+09	4.06e+07 1.2%	2.10e+11	
__hydro_module_MOD_hydro	5.89e+10	.89e+10 50.3%	9.92e+09	9.92e+09 31.1%	1.09e+09	1.09e+09 33.1%	2.10e+11	
__hydro_module_MOD_hydro_xyz	5.76e+10	.76e+10 49.3%	9.57e+09	9.57e+09 30.0%	1.06e+09	1.06e+09 32.0%	2.04e+11	
hydro_extras_	5.47e+10	.47e+10 46.8%	8.62e+09	8.62e+09 27.0%	9.68e+08	9.68e+08 29.2%	1.86e+11	
__hydro_lanl_1_module_MOD_hydr	5.47e+10	.47e+10 46.8%	8.62e+09	8.62e+09 27.0%	9.68e+08	9.68e+08 29.2%	1.86e+11	
hydro_advect_extras_	2.48e+10	.48e+10 21.2%	3.65e+09	3.65e+09 11.4%	3.54e+08	3.54e+08 10.7%	6.79e+10	
__hydro_lanl_1_module_MOD_hl	1.08e+10	.08e+10 9.3%	1.60e+09	1.60e+09 5.0%	1.55e+08	1.55e+08 4.7%	2.97e+10	
__hydro_1_module_MOD_hyd	1.08e+10	.08e+10 9.2%	1.54e+09	1.54e+09 4.8%	1.55e+08	1.55e+08 4.7%	2.97e+10	
hydro_lanl_1.f90: 357	3.86e+07	.86e+07 0.0%	5.79e+07	5.79e+07 0.2%				
__hydro_lanl_1_module_MOD_hl	5.14e+09	.14e+09 4.4%	9.31e+08	9.31e+08 2.9%	9.75e+07	9.75e+07 2.9%	1.87e+10	
__hydro_1_module_MOD_init_de	4.04e+09	.04e+09 3.5%	7.92e+08	7.92e+08 2.5%	1.01e+08	1.01e+08 3.0%	1.94e+10	
__hydro_lanl_1_module_MOD_hl	2.43e+09	.43e+09 2.1%	6.80e+08	6.80e+08 2.1%	3.39e+07	3.39e+07 1.0%	6.51e+09	
__hydro_1_module_MOD_h1_cell	4.91e+09	.91e+09 4.2%	4.76e+08	4.76e+08 1.5%	1.58e+08	1.58e+08 4.8%	3.04e+10	
__hydro_1_module_MOD_hydro_	1.39e+09	.39e+09 1.2%	2.91e+08	2.91e+08 0.9%	4.24e+07	4.24e+07 1.3%	8.15e+09	

KCachegrind view of Byfl measurements

Los Alamos NATIONAL LABORATORY

UNCLASSIFIED LA-UR-12-22355

National Nuclear Security Administration

Application Team Efforts – Bob Robey

- The Application Team have an shallow-water AMR mini-app with thousands of lines of OpenCL code and dozens of GPU kernels.
  - It also has an MPI layer over the OpenCL to give a hybrid parallel implementation.
  - It is a DoE "mini-app" developed to facilitate collaboration between hardware and software development. It is available at http://www.github.com/losalamos/CLAMR
- The experience developing this mini-app is the basis for the much of the thoughts here.
  - Just starting to look at the performance issues at the MPI/GPU interface.
  - Expect significant performance issues at the interface and will be working on how to improve it.
- Robustness and crash behavior still needs work.
  - Development is in OpenCL so that portability and performance can be assessed on different hardware platforms.
  - This is made difficult when one system doesn't like a few lines of kernel code and we have to hang nodes repeatedly to work out how to recode it. On some hardware, we even have to power cycle the node or computer to fix it. Perhaps an emulator or "safe" operating mode that prevents crashes or hangs and gives better reports on the location of the problem could be developed.

High Performance Computing

Application Team Efforts – tool needs

- Memory tools for GPU -- to develop an application they need memory tools similar to those developed for the CPU. Tools/functionality that are needed include memory leaks, memory use, out-of-bounds. GPU memory and objects are allocated in a linked-list structure so they can "walk the heap" (ezcl package in CLAMR at github site). But haven't yet implemented the reporting calls. A general package for this functionality would be useful.

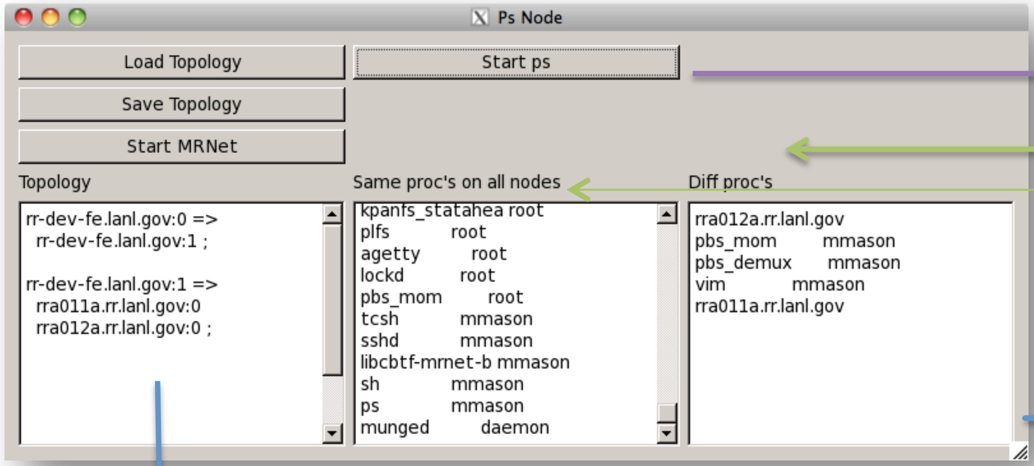
- Debugging -- a "mailbox" technique for debugging GPUs is used. (the idea for this technique came from a Stanford graduate student). The process is:
  – Send in an extra array
  – Write debug information to it in the kernel
  – Read back on CPU
  – Print.

  It is very effective, but time-consuming to setup. (an aside -- could a similar process be used to get partial timings for kernels).

Update on CBTF deployment – Mike Mason/TJ Machado

- At LANL – assessing use as an administrative tool at large scales and as a component in the monitoring infrastructure
- Component Based Tool Framework (CBTF - a tri-lab effort with Krell Institute) is flexible and general enough to be used for any tool that needs to “do something” on a large number of nodes and filter or collect the results.
- Sysadmin Tools
  - Poll information on a large number of nodes.
  - Run commands or manipulate files on the Backends.
  - Make decisions at the Filter level to reduce output or interaction
- Debugging Tools
  - Use cluster analysis to reduce thousands (or more) processes into a small number of groups
- Monitoring Tool
  - Memory
  - GPGPU

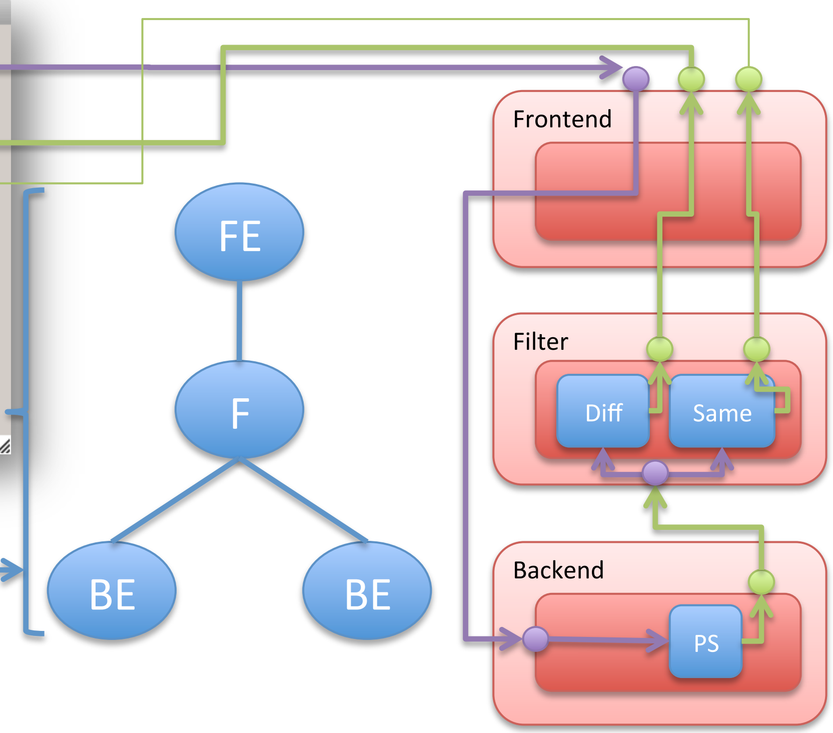
- PS Tool (tools/contrib/psTool)
  - Run ps on all Backends, use filters to identify the procs running on all nodes and what procs are unique to each node.



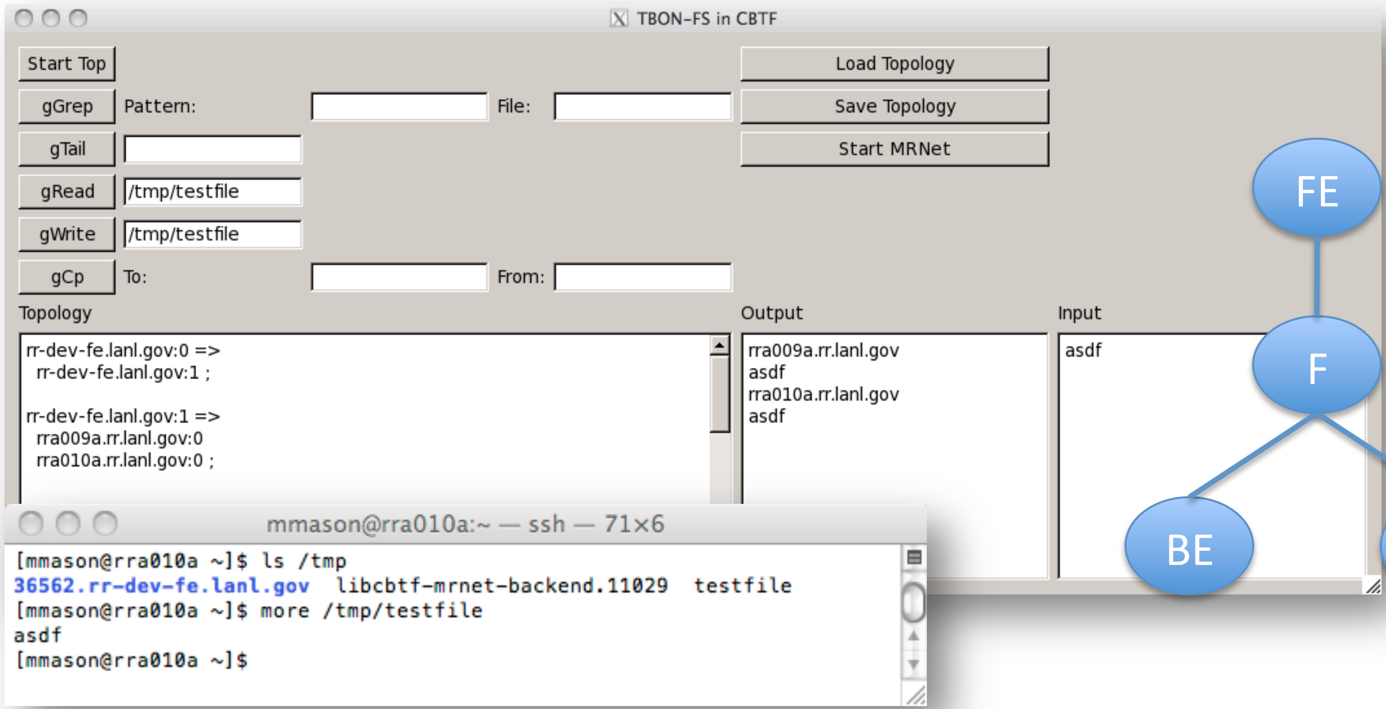
```
rr-dev-fe.lanl.gov:0 =>
  rr-dev-fe.lanl.gov:1 ;

rr-dev-fe.lanl.gov:1 =>
  rra011a.rr.lanl.gov:0
  rra012a.rr.lanl.gov:0 ;
```

```
kpanfs_statahea root
plfs          root
agetty          root
lockd         root
pbs_mom          root
tcsh          mmason
sshd          mmason
libcbtf-mrnet-b mmason
sh            mmason
ps            mmason
munged          daemon
```

```
rra012a.rr.lanl.gov
pbs_mom        mmason
pbs_demux        mmason
vim           mmason
rra011a.rr.lanl.gov
```



- TBON-FS implementation (tools/contrib/tbonFS)
  - Perform group file operations on the Backend local file system.
  - Run top on all Backends.
  - Run grep, tail, read or write files on all Backends.
  - Copy a file from NFS to the local /tmp on all Backends.



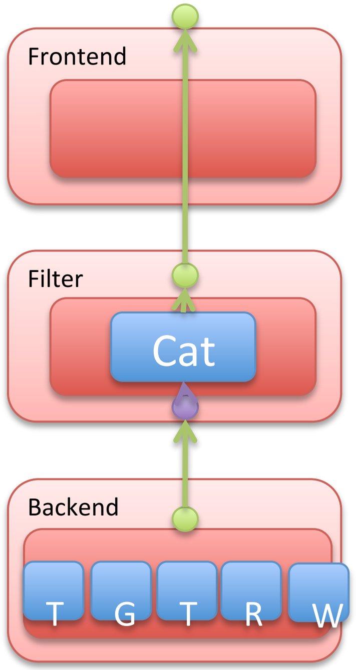
gRead: /tmp/testfile
gWrite: /tmp/testfile

Topology:
```
rr-dev-fe.lanl.gov:0 =>
  rr-dev-fe.lanl.gov:1 ;

rr-dev-fe.lanl.gov:1 =>
  rra009a.rr.lanl.gov:0
  rra010a.rr.lanl.gov:0 ;
```

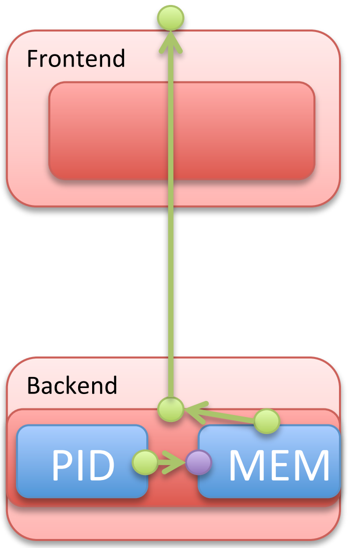
Output:
```
rra009a.rr.lanl.gov
asdf
rra010a.rr.lanl.gov
asdf
```

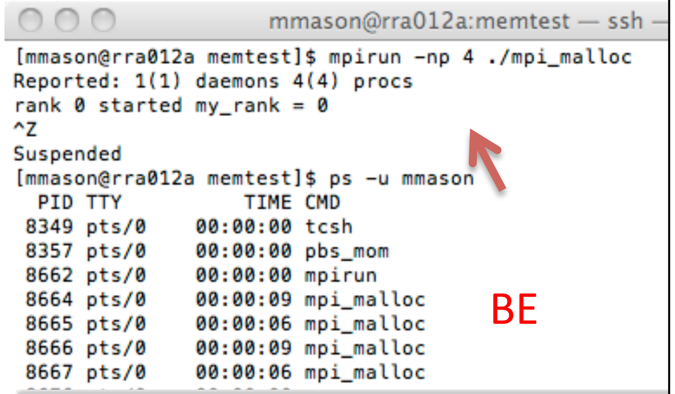
Input:
```
asdf
```

```
[mmason@rra010a ~]$ ls /tmp
36562.rr-dev-fe.lanl.gov  libcbtf-mrnet-backend.11029  testfile
[mmason@rra010a ~]$ more /tmp/testfile
asdf
[mmason@rra010a ~]$
```

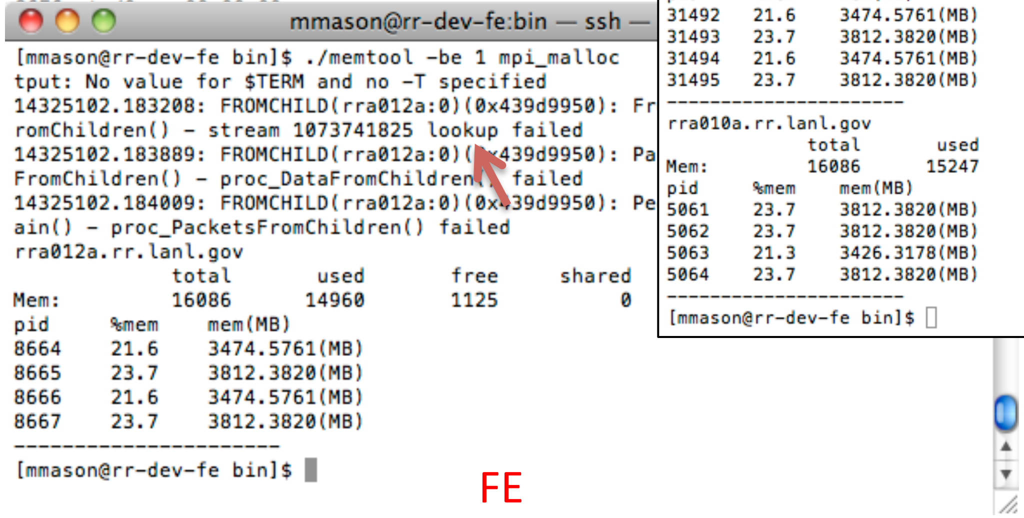


- Simple memory tool - memTool (tools/contrib/memTool)
  - Displays memory info for each node and total memory used for each MPI process.

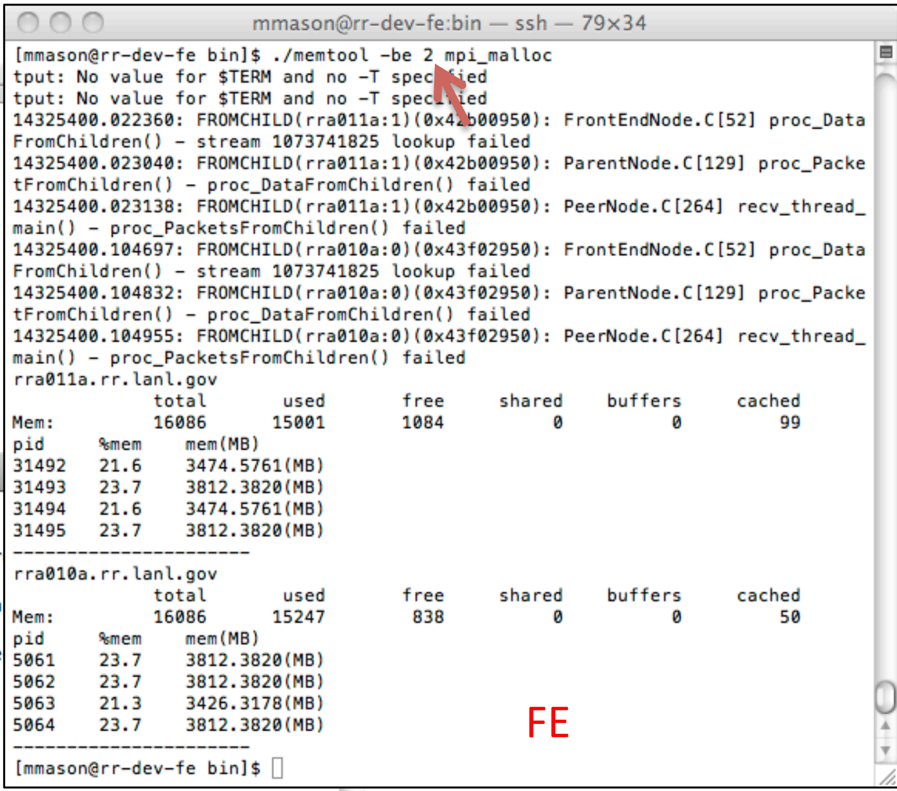




```
[mmason@rra012a memtest]$ mpirun -np 4 ./mpi_malloc
Reported: 1(1) daemons 4(4) procs
rank 0 started my_rank = 0
^Z
Suspended
[mmason@rra012a memtest]$ ps -u mmason
  PID TTY          TIME CMD
 8349 pts/0    00:00:00 tcsh
 8357 pts/0    00:00:00 pbs_mom
 8662 pts/0    00:00:00 mpirun
 8664 pts/0    00:00:09 mpi_malloc
 8665 pts/0    00:00:06 mpi_malloc
 8666 pts/0    00:00:09 mpi_malloc
 8667 pts/0    00:00:06 mpi_malloc
```



```
[mmason@rr-dev-fe bin]$ ./memtool -be 1 mpi_malloc
tput: No value for $TERM and no -T specified
14325102.183208: FROMCHILD(rra012a:0)(0x439d9950): Fr
romChildren() - stream 1073741825 lookup failed
14325102.183889: FROMCHILD(rra012a:0)(0x439d9950): Pa
FromChildren() - proc_DataFromChildren() failed
14325102.184009: FROMCHILD(rra012a:0)(0x439d9950): Pe
ain() - proc_PacketsFromChildren() failed
rra012a.rr.lanl.gov
                 total        used        free       shared
Mem:             16086       14960        1125            0
pid      %mem      mem(MB)
8664     21.6      3474.5761(MB)
8665     23.7      3812.3820(MB)
8666     21.6      3474.5761(MB)
8667     23.7      3812.3820(MB)
----------------------
[mmason@rr-dev-fe bin]$ 
```



```
[mmason@rr-dev-fe bin]$ ./memtool -be 2 mpi_malloc
tput: No value for $TERM and no -T specified
tput: No value for $TERM and no -T specified
14325400.022360: FROMCHILD(rra011a:1)(0x42b00950): FrontEndNode.C[52] proc_Data
FromChildren() - stream 1073741825 lookup failed
14325400.023040: FROMCHILD(rra011a:1)(0x42b00950): ParentNode.C[129] proc_Packe
tFromChildren() - proc_DataFromChildren() failed
14325400.023138: FROMCHILD(rra011a:1)(0x42b00950): PeerNode.C[264] recv_thread_
main() - proc_PacketsFromChildren() failed
14325400.104697: FROMCHILD(rra010a:0)(0x43f02950): FrontEndNode.C[52] proc_Data
FromChildren() - stream 1073741825 lookup failed
14325400.104832: FROMCHILD(rra010a:0)(0x43f02950): ParentNode.C[129] proc_Packe
tFromChildren() - proc_DataFromChildren() failed
14325400.104955: FROMCHILD(rra010a:0)(0x43f02950): PeerNode.C[264] recv_thread_
main() - proc_PacketsFromChildren() failed
rra011a.rr.lanl.gov
              total       used        free      shared     buffers      cached
Mem:          16086      15001        1084           0           0          99
pid      %mem      mem(MB)
31492    21.6      3474.5761(MB)
31493    23.7      3812.3820(MB)
31494    21.6      3474.5761(MB)
31495    23.7      3812.3820(MB)
----------------------
rra010a.rr.lanl.gov
              total       used        free      shared     buffers      cached
Mem:          16086      15247         838           0           0          50
pid      %mem      mem(MB)
5061     23.7      3812.3820(MB)
5062     23.7      3812.3820(MB)
5063     21.3      3426.3178(MB)
5064     23.7      3812.3820(MB)
----------------------
[mmason@rr-dev-fe bin]$ 
```