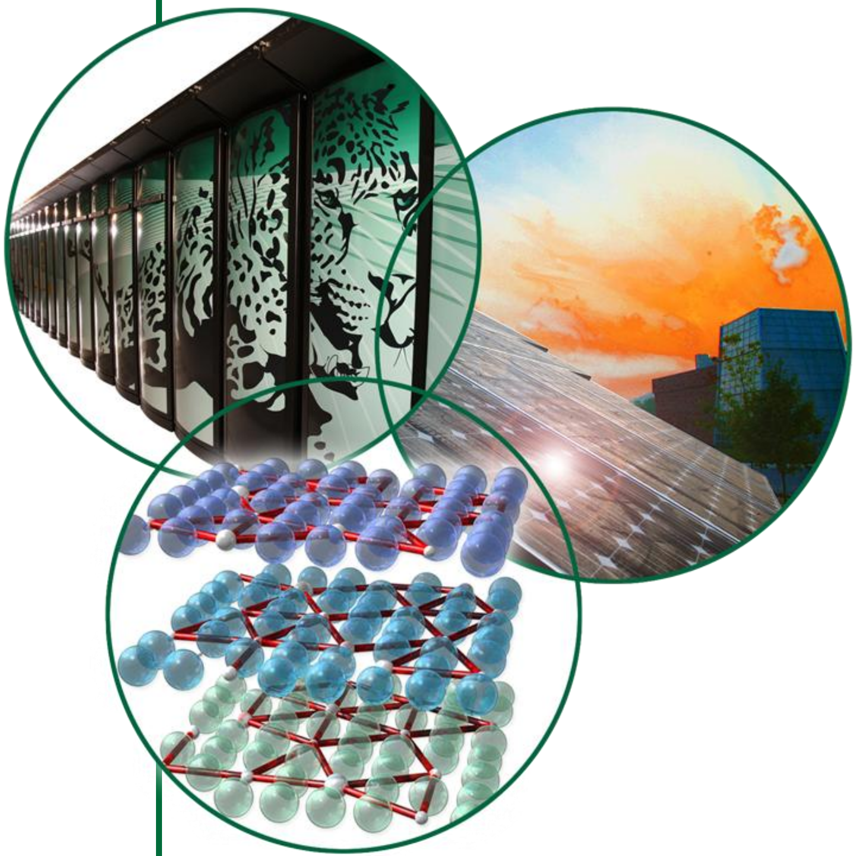
Memphis on an XT5

Pinpointing Memory Performance Problems on Cray Platforms



Collin McCurdy, **Jeffrey Vetter**, Patrick Worley and Don Maxwell

U.S. DEPARTMENT OF ENERGY

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY

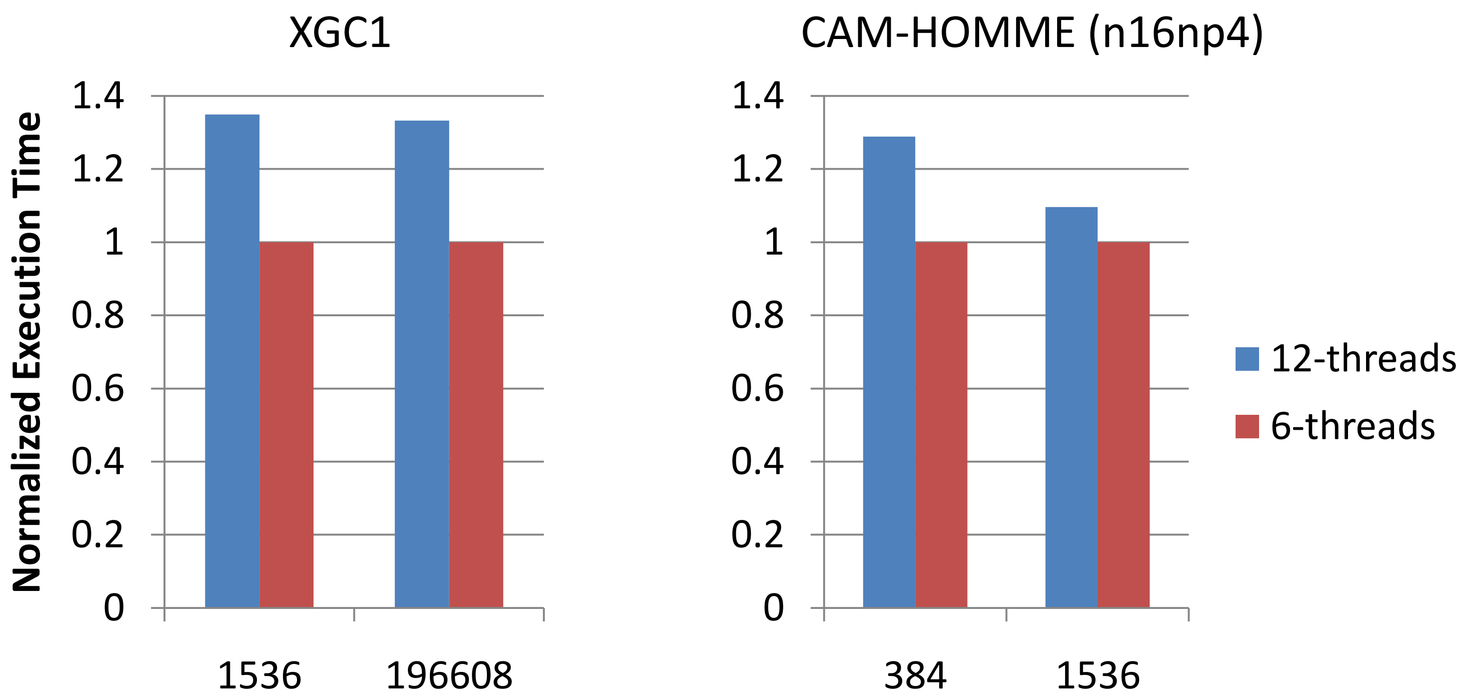
Overview

- Current projections: each chip in an Exascale system will contain 100s to 1000s of processing cores
  - Already (~10 cores/chip) memory limitations and performance considerations are forcing scientific application teams to consider multi-threading
  - At the same time, trends in micro-processor design are pushing memory performance problems associated with Non-Uniform Memory Access (NUMA) to ever-smaller scales
- This talk:
  - Describes *Memphis*, a toolset that uses sampling-based hardware performance monitoring extensions to pinpoint the sources of memory performance problems
  - Describes how we ported *Memphis* to an XT5, and runtime policies that make it available
  - Demonstrates the use of *Memphis* in an iterative process of finding problems and evaluating fixes in CICE

Case for Multi-threading

- **Claim**: As cores proliferate, scientific applications may *require* multi-threading support due to
  - Memory constraints (processes vs threads)
  - Performance considerations
- **Support**: Two large-scale, production codes that scale better with 6 threads per process than with 1
  - XGC1
    - Fusion code, models aspects of Tokamak reactor
    - Scales to 200,000+ cores
  - CAM-HOMME
    - CAM is the atmospheric model from CESM climate code
    - HOMME performs ‘dynamics’ computations, relatively new addition, better scaling properties than previous dynamics models
    - OpenMP pragmas only recently re-instated

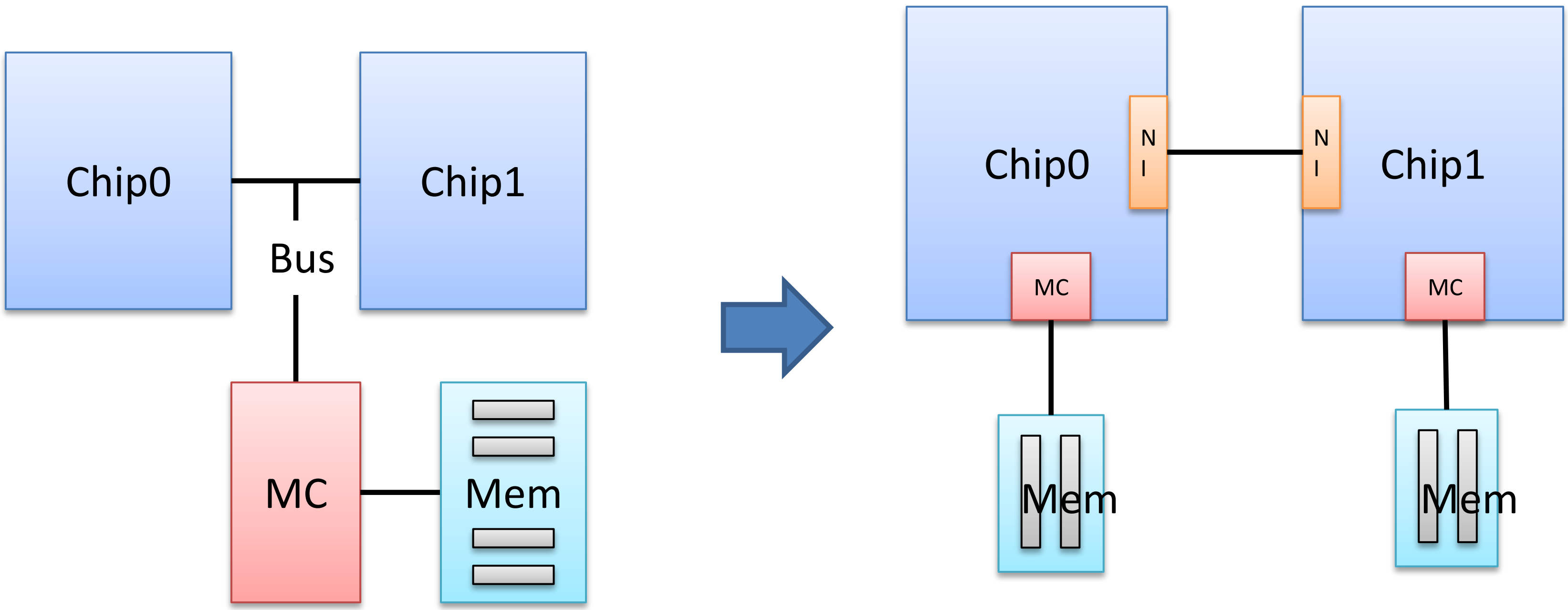
6 Threads Good, 12 Threads Better?



Not necessarily...on Jaguar, 12 threads me
sockets/NUMA-nodes. NUMA effects car

Two trends in microprocessor design are bringing NUMA to SMPs

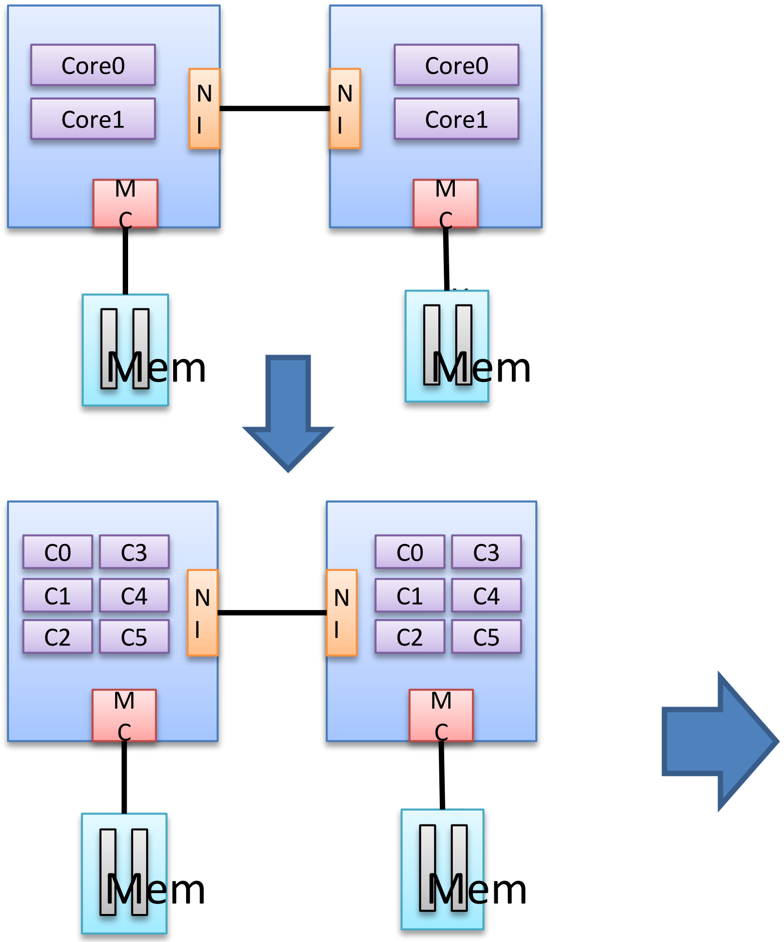
Trend 1: On-chip Memory Controller

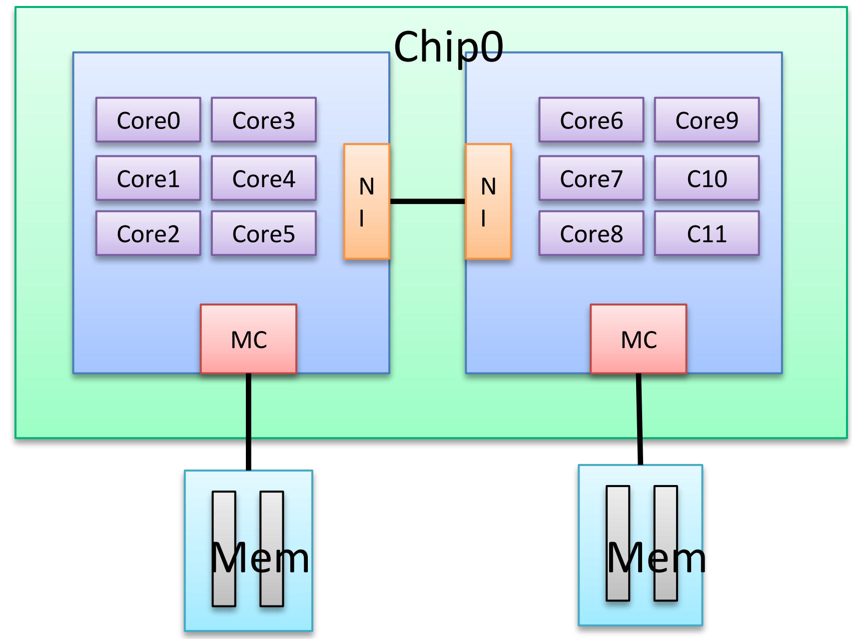


Multi-chip SMP systems used to be bus-based, limiting scalability.

On-chip memory controllers improve performance for *local* data, but non-local data requires communication.

Trend 2: Ever-Increasing Core Counts





NUMA *within* socket.

More and more pressure on shared resources until eventually...

Memory System Performance Problems

- Typical NUMA problems:
  - Hot-spotting
  - Computation/Data-partition mismatch
- NUMA can also amplify *potential* problems and turn them into significant *real* problems.
  - Example: contention for locks and other shared variables
    - NUMA can significantly increase latency (and thus waiting time), increasing possibility of further contention.

So, more for programmers to worry about, but there is Good News…

1. Mature infrastructure already exists for handling NUMA from software level
   - NUMA-aware operating systems, compilers and runtime
   - Based on years of experience with distributed shared memory platforms like SGI Origin/Altix
2. New access to performance counters that help identify problems and their sources
   - NUMA performance problems caused by references to remote data
   - Counters naturally located in Network Interface
     - On chip => easy access, accurate correlation

Instruction-Based Sampling

- AMD's hardware-based performance monitoring extensions
- Similar to ProfileMe hardware introduced in DEC Alpha 21264
- Like event-based sampling, interrupt driven; but not due to cntr overflow
  - HW periodically interrupts, follows the next instruction through pipeline
  - Keeps track of what happens to and because of the instruction
  - Calls handler upon instruction retirement
- Intel's PEBS-LoadLatency extensions are similar, but limited to memory (*lds*)
- Both provide the following data useful for finding NUMA problems:
  - Precise program counter of instruction
  - Virtual address of data referenced by instruction
  - Where the data came from: i.e., DRAM, another core's cache
  - Whether the agent was local or remote
- Post-pass looks for patterns in resulting data
- Instruction and data address enables precise attribution to code and *variables*

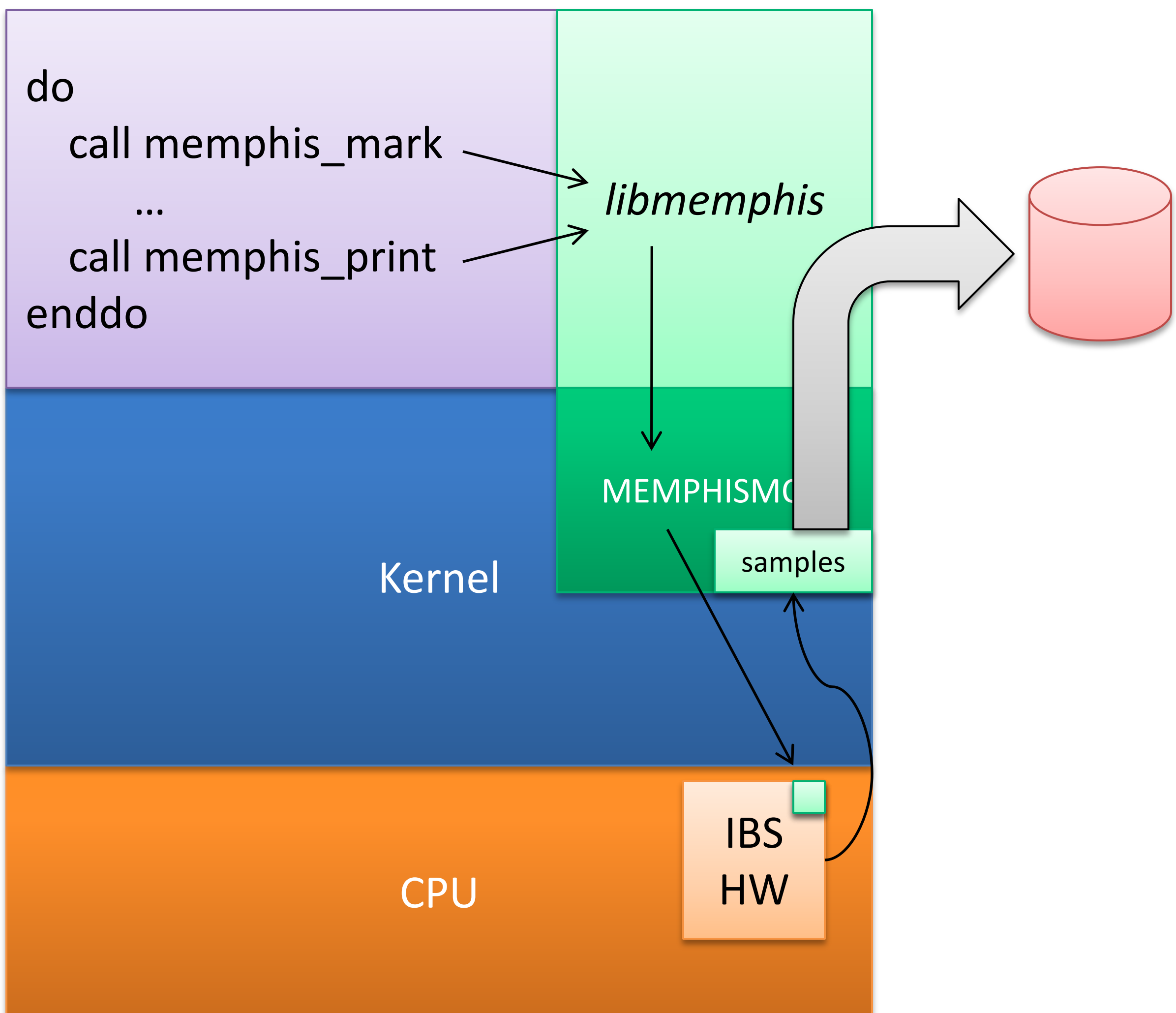
Memphis Introduction

- Toolset using IBS to pinpoint NUMA problems at source
- *Data*-centric approach
  - Other sampling-based tools associate info w/ instructions
  - Memphis associates info with *variables*

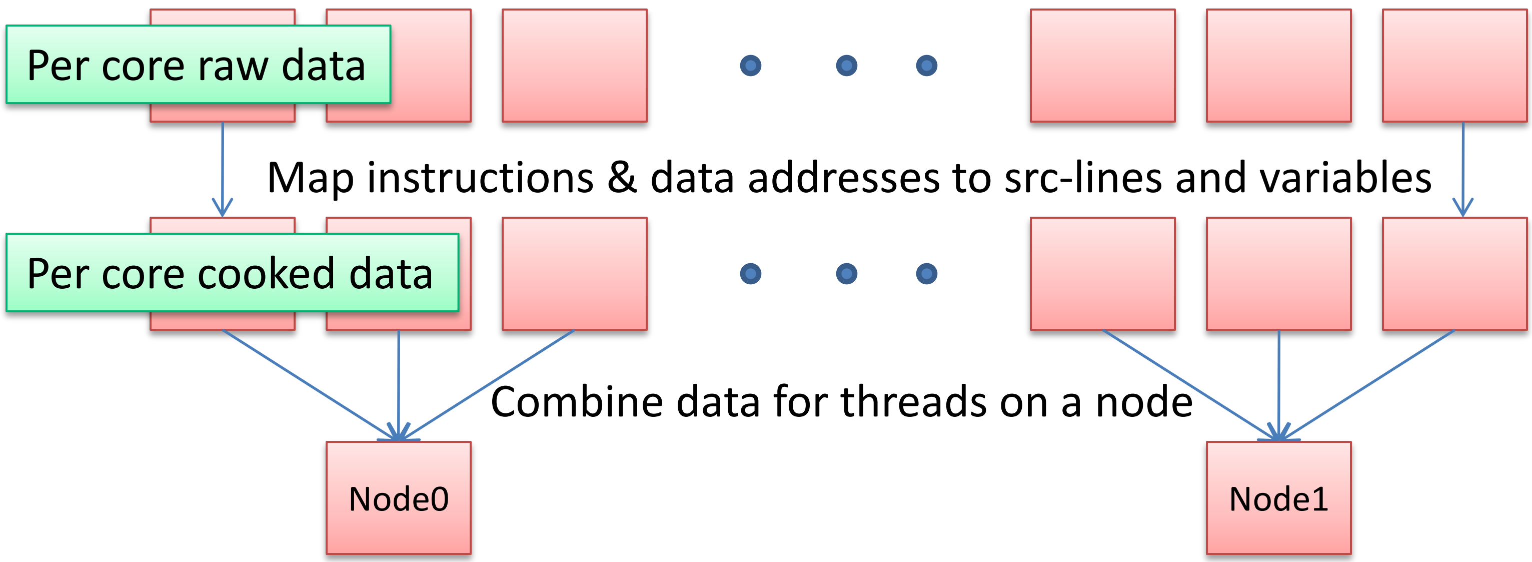
Key Insight: The source of a NUMA problem is not necessarily where it's evidenced

  - Example: Hot spot cause is variable init, problems evident at use
  - Programmers want to know
    - 1^{st} what variable is causing problems
    - 2^{nd} where (likely multiple sites)
- Consists of three components
  - Kernel module interface with IBS hardware
  - Library API to set 'calipers' and gather samples
  - Post-processing executable

Memphis Runtime Components



Memphis Post-processing Executable



Challenges:

1) Instructions -> src-line mapping depends on quality of debug info; more likely to find loop-nest than line
2) Address -> variable mapping for dynamic data (local vars in Fortran, global heap vars)

Memphis on Cray Platforms

- Compute Node Linux (CNL) is Linux-based
  - many components of *Memphis* work on Cray platforms without modification
- One exception: the kernel module
- Kernel module port complicated by the black-box nature of CNL (not open-source)
- Required the help of a patient Cray engineer (John Lewis) to perform first half of each iteration of the compile-install-test-modify loop
- Also required a mechanism for making *Memphis* available to jobs that want to use it

Kernel Module Modifications

- Initial port required two changes to the module
  1. Kernel used by CNL was older than the kernel for which we had originally developed the module; setting of interrupt-handler had changed between versions
     - Looking at other drivers we determined that kernel used by CNL required *set_nmi_callback* rather than *register_die_notifier*
  2. Several files defining functions and constants used to configure IBS registers were not contained in the CNL distribution
     - Hard-coded the values we required (found via *lspci* command) into calls that set configuration registers
- Current status:
  – After a recent system software upgrade
    - *Memphis* kernel module for the standard Linux kernel version used by the new system, worked without further modification

Runtime Policy and Configuration

- Goal:
  - Maximize the availability of *Memphis* for selected users, while minimizing impact of a bleeding-edge kernel module on others
- Policy:
  - Kernel module is always available on a single, dedicated node of the system
    - On system reboots the kernel module is installed on the dedicated node and a device entry created in /dev
  - Users that want to access *Memphis* have a ‘reservation’ on that node
    - Realized as a Moab *standing reservation*
- Only one node provides sample data
  - We have found that this is sufficient for our needs
  - Intra-node performance is typically uniform across nodes

A *Memphis* Queue?

- Can easily imagine an alternative, queue-based policy
  - Batch queue dedicated to jobs wishing to use *Memphis*
  - Some number of compute nodes would have the kernel module installed
  - One of those nodes required to be the initial node in allocation of any job submitted to the *Memphis* queue

Case Study: CICE

- CICE is sea ice modeling component of the Community Earth System Model (CESM) climate modeling code
- Recent large-scale CESM runs on the *Jaguarpf* system at ORNL, CICE was not scaling as well as other components
- While not a large fraction of overall runtime, CICE is on critical path, scalability is crucial to overall scalability
- Wished to use *Memphis* to investigate improvements in the memory system performance of the ice model that might improve scalability
- Having *Memphis* available on an *XT5* allowed measure performance in a realistic setting, with all components active and running a representative data set

CICE initial results

REMOTE DRAM References

NODE: 0 total: 6591
000) [heap]:tx [0x2a5b1588 - 0x2b017870] 1719
 ice_boundary.F90:4106:0x9d4834   [0x2a5c1468 - 0x2b017788] 1414
 ice_boundary.F90:4106:0x9d4830   [0x2a5b1588 - 0x2b017870] 279
 ...
001) [heap]:ty [0x2b022808 - 0x2ba83518] 1643
 ice_boundary.F90:4106:0x9d4834   [0x2b02d190 - 0x2ba83190] 1361
 ice_boundary.F90:4106:0x9d4830   [0x2b02d8b0 - 0x2ba83518] 251
 ...
002) [heap]:tc [0x29b4b158 - 0x2a5abee8] 1611
 ice_boundary.F90:4106:0x9d4834   [0x29b53d28 - 0x2a5abee8] 1377
 ice_boundary.F90:4106:0x9d4830   [0x29b4b158 - 0x2a5aae18] 205
 ...
003) [heap]:_ice_state_2_ [0x172a8dc0 - 0x180b0088] 1582
 ice_boundary.F90:4106:0x9d4834   [0x176bb2d8 - 0x17e35f48] 914
 ice_boundary.F90:2727:0x9cfa64   [0x174b1030 - 0x18044610] 482
 ice_boundary.F90:4106:0x9d4830   [0x176ba888 - 0x17e35930] 148
 ...

NODE: 1 total: 506
000) [heap]:<not-found> [0x24b94140 - 0x2c9cdb10] 69
 ice_history.F90:2564:0xa4585c   [0x29192040 - 0x29b40048] 66
 ...   . . .

13X more remote refs from Node 0, all from 4 arrays in 1 loopnest...

ice_boundary.F90:4106

```
do nmsg=1,halo%numLocalCopies
   iSrc     = halo%srcLocalAddr(1,nmsg)
   jSrc     = halo%srcLocalAddr(2,nmsg)
   srcBlock = halo%srcLocalAddr(3,nmsg)
   iDst     = halo%dstLocalAddr(1,nmsg)
   jDst     = halo%dstLocalAddr(2,nmsg)
   dstBlock = halo%dstLocalAddr(3,nmsg)

   if (srcBlock > 0) then
      if (dstBlock > 0) then
         do l=1,nt
            do k=1,nz
               array(iDst,jDst,k,l,dstBlock) = &
                   array(iSrc,jSrc,k,l,srcBlock)
            end do
         end do
   ...
end do
```

Timer	**Count**	**Value**
TimeLoop	240	40.687691
Bound	32410	24.978573
ice_halo4dr8	1700	12.600817
ice_halo4dr8_lclcpy	1700	7.242013

Responsible for fully 17% of CICE runtime, clear target for optimization.

Memphis-directed Modification 1

```
$OMP PARALLEL PRIVATE(myid,...)
myid = omp_get_thread_num()
do nmsg=1,halo%numLocalCopies
   iSrc     = halo%srcLocalAddr(1,nmsg)
   jSrc     = halo%srcLocalAddr(2,nmsg)
   srcBlock = halo%srcLocalAddr(3,nmsg)
   iDst     = halo%dstLocalAddr(1,nmsg)
   jDst     = halo%dstLocalAddr(2,nmsg)
   dstBlock = halo%dstLocalAddr(3,nmsg)
   if (srcBlock > 0) then
      if (dstBlock > 0 .and. &
          block_to_thr(dstBlock).eq.myid) then
         do l=1,nt
            do k=1,nz
               array(iDst,jDst,k,l,dstBlock) = &
                   array(iSrc,jSrc,k,l,srcBlock)
            end do
         end do
   ...
end do
```

Timer	Base	Mod1
TimeLoop	40.69	36.29
Bound	24.98	20.22
ice_halo4dr8	12.60	8.75
ice_halo4dr8_lclcpy	7.24	2.38

Improves loopnest performance by 3X, overall performance by 10%.

Memphis Results After Modification 1

REMOTE DRAM References

NODE: 0 total: 1156
000) [heap]:_ice_state_2_ [0x172d0e80 - 0x180b9018] 625
ice_boundary.F90:2779:0x9cfae4 [0x174cfae0 - 0x17fe41e0] 465
ice_boundary.F90:4245:0x9d48e0 [0x176ba7f0 - 0x17e35ef0] 105
...
001) [heap]:tc [0x29b45cf0 - 0x2a5abe08] 231
ice_boundary.F90:4245:0x9d48e0 [0x29b54848 - 0x2a5ab6a0] 216
...
002) [heap]:tx [0x2a5b14c0 - 0x2b017ad8] 135
ice_boundary.F90:4245:0x9d48e0 [0x2a5b1c50 - 0x2b017ad8] 93
ice_boundary.F90:4164:0x9d4460 [0x2a5b14c0 - 0x2b004730] 33
...
NODE: 1 total: 3305
000) [heap]:ty [0x2b01d348 - 0x2ba83890] 708
ice_boundary.F90:4245:0x9d48e0 [0x2b02be70 - 0x2ba837f0] 706
...
001) [heap]:tx [0x2a5b14c0 - 0x2b017ad8] 678
ice_boundary.F90:4245:0x9d48e0 [0x2a5b1c50 - 0x2b017ad8] 675
...
002) [heap]:_ice_state_2_ [0x172d0e80 - 0x180b9018] 562
ice_boundary.F90:4245:0x9d48e0 [0x176ba7f0 - 0x17e35ef0] 494
ice_boundary.F90:4245:0x9d48e4 [0x176c1b08 - 0x17e35fc8] 60
...

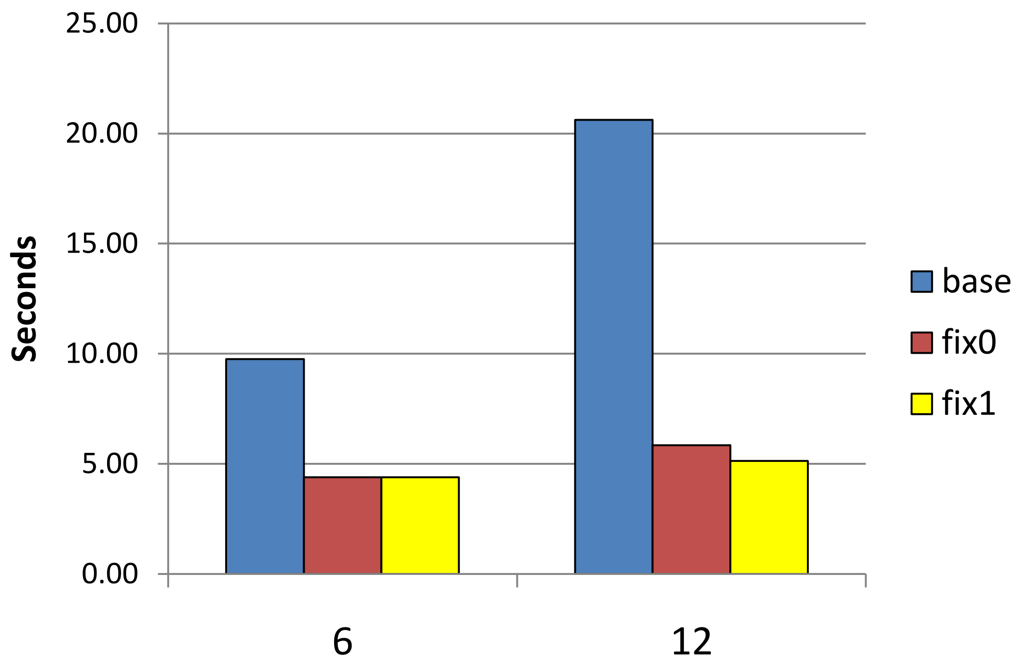
Remote misses more evenly distributed, but counts still high...see text!

Conclusion

- NUMA is already a problem, and it will only get worse...but there is hope.
  - *Memphis* is a toolset that uses sampling-based hardware performance monitoring extensions to pinpoint the sources of memory performance problems
  - *Memphis* is now available on Cray platforms
  - We have used *Memphis* to find and fix significant problems in several large-scale production applications
- Want us to look at your application? Let us know!
- Want *Memphis* on your system? Let us know!

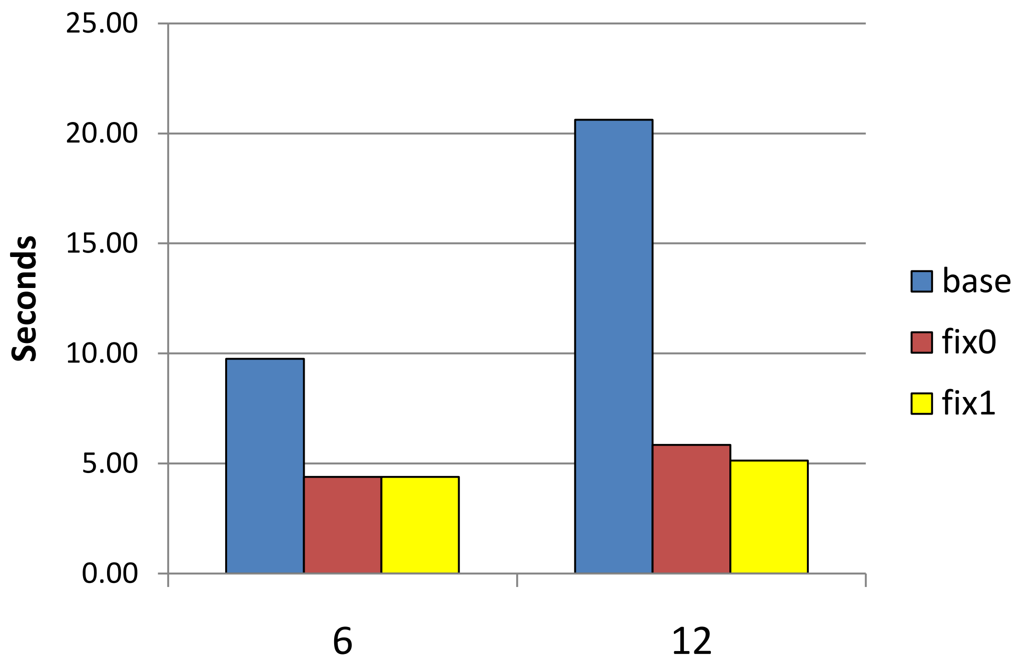
Bonus Slides…

App 1: XGC1



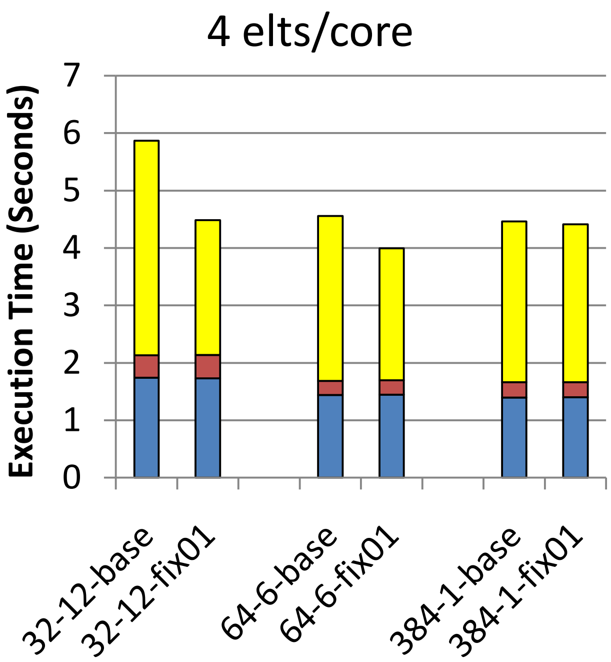
- Analysis (and shown results) on toy single-node input set
- Fix0 expands several F90 array statements, i.e.: a(:) = b(:)
  - Compiler was unable to analyze dependences; required locks
  - *Memphis* reported a large number of remote lock accesses
- Fix1 replicates fields of a table in multiple nodes

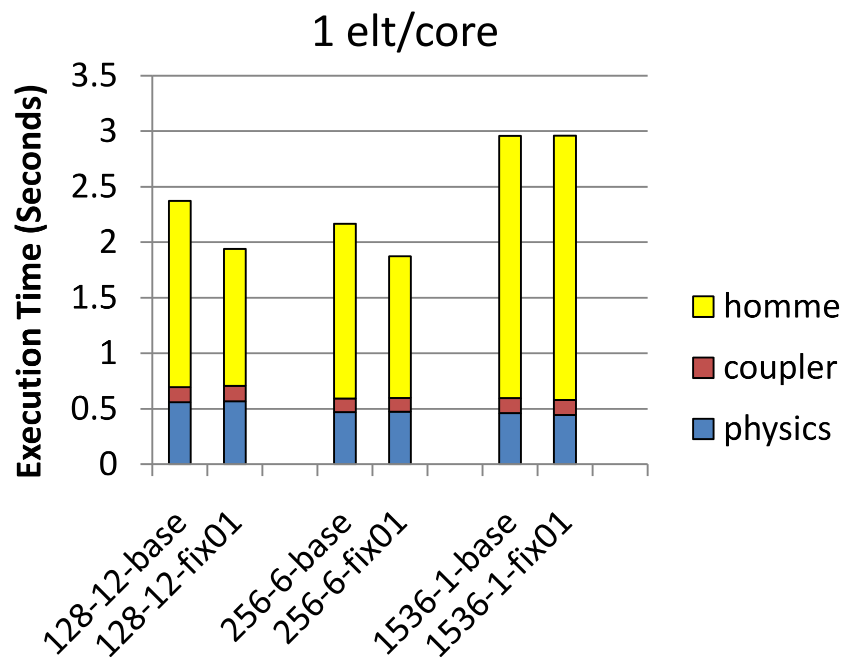
App 1: XGC1



- Fix0 is in XGC1 development tree.
- Results in 23% performance improvement for full-scale, dual-socket multi-threaded runs across ~200,000 cores.
- 12-thread performance *almost* equal to 6-thread...

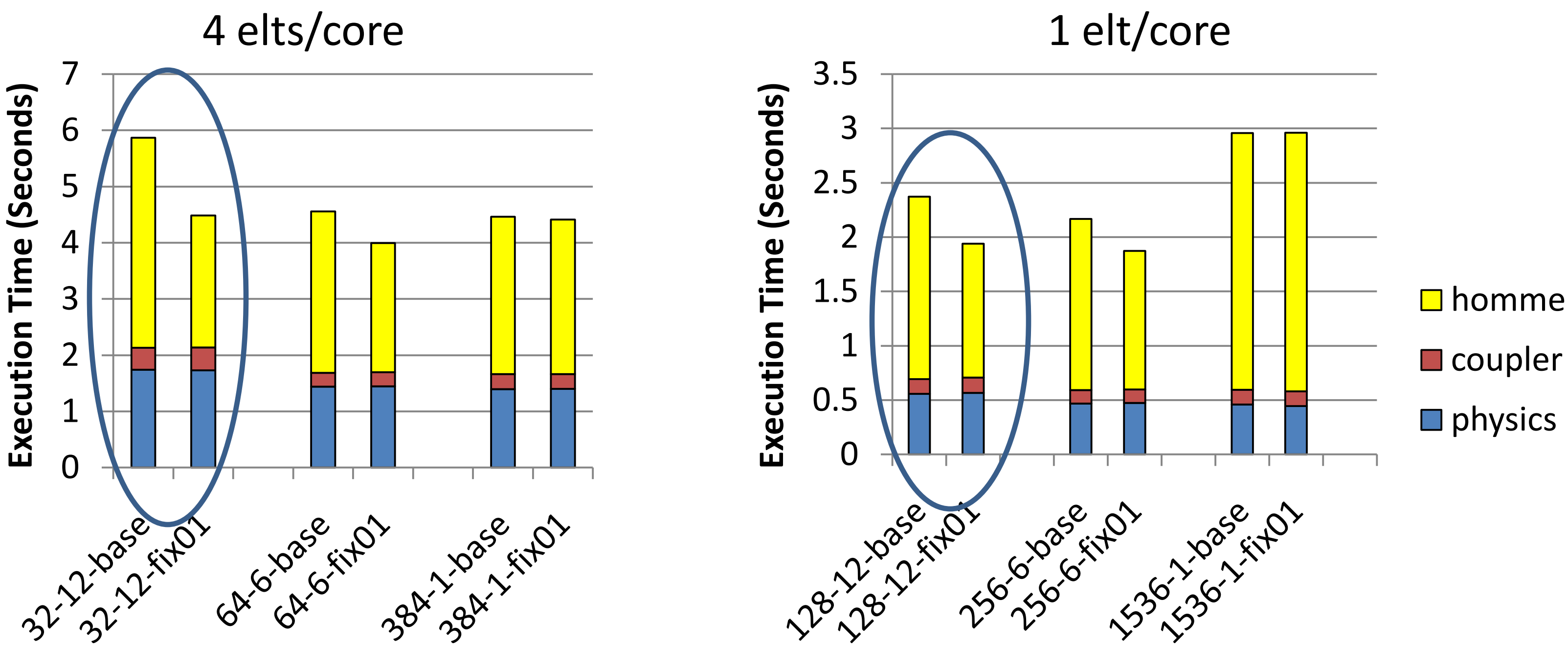
App 2: CAM-HOMME (ne16np4)





- Again, analysis done on toy input, but results here from real input.
- Fix0 again expands several F90 array statements.
- Fix1 replaces variable-sized arrays passed as arguments to several heavily used routines with (equivalent) constant-sized
  - Compiler repeatedly allocs/deallocs data, requiring fresh first-touches
  - *Memphis* pointed out a high-percentage of OS references

App 2: CAM-HOMME (ne16np4)



- Improves overall 12-thread CAM performance by 23% for 4 elts/core, 18% for 1.
- Also improves 6-thread performance.
- 12-thread HOMME performance roughly equals 6-thread performance.
- Still investigating larger inputs (BUG...)