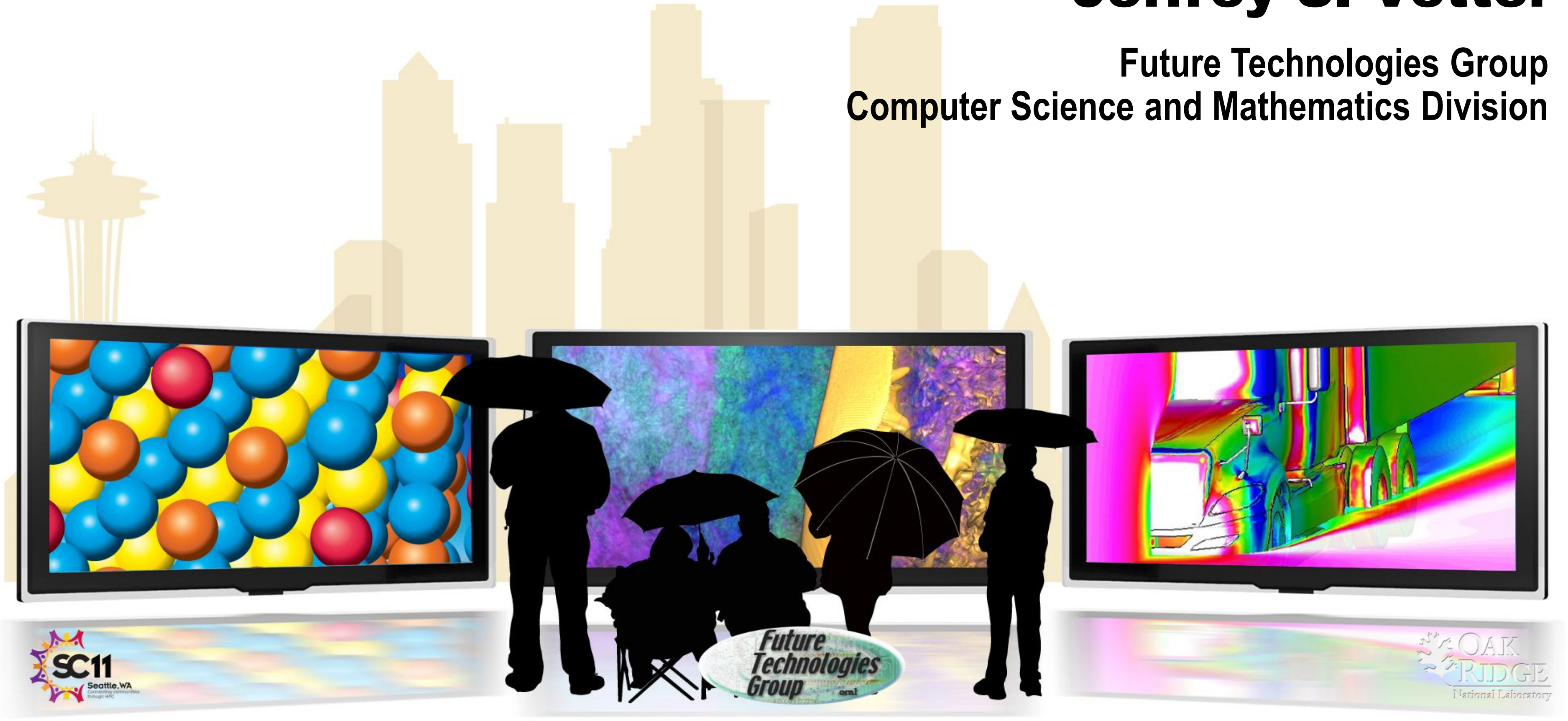
Memphis: Finding and Fixing NUMA-Related Performance Problems on Multi-core Platforms

Presented by

Collin McCurdy
Jeffrey S. Vetter

Future Technologies Group
Computer Science and Mathematics Division



Overview

- **Current projections call for each chip in an Exascale system to contain 100s to 1000s of processing cores**
  - **Already (~10 cores/chip) memory limitations and performance considerations are forcing scientific application teams to consider alternatives to "MPI-everywhere"**
  - **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**
- ***Memphis** uses sampling-based hardware performance monitoring extensions to pinpoint the sources of memory system performance problems due to, or exacerbated by, NUMA**

*C. McCurdy and J. S. Vetter, "Memphis: Finding and Fixing NUMA-Related Performance Problems on Multi-core Platforms," In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, March 2010.

NUMA Performance Problems

- Typical performance problems associated w/ NUMA:
  - Hot-spotting
    - Due to poor initialization, memory not distributed across nodes
  - Computation/Data-partition mismatch
    - Memory distributed, but not appropriately
- NUMA can also amplify small performance bugs, turning them into significant 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 for handling NUMA from software level already exists
   - 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 with 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. What variable is causing problems
    2. 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

Recent Extensions

- Mapping addresses to dynamically allocated variables
- Port to Cray CNL*
- Eclipse-based GUI

*C. McCurdy, J.S. Vetter, P. Worley, and D. Maxwell, "Memphis on an XT5: Pinpointing Memory Performance Problems on Cray Platforms," in *Proc. Cray Users Group Conference (CUG 2011)*, May 2011.

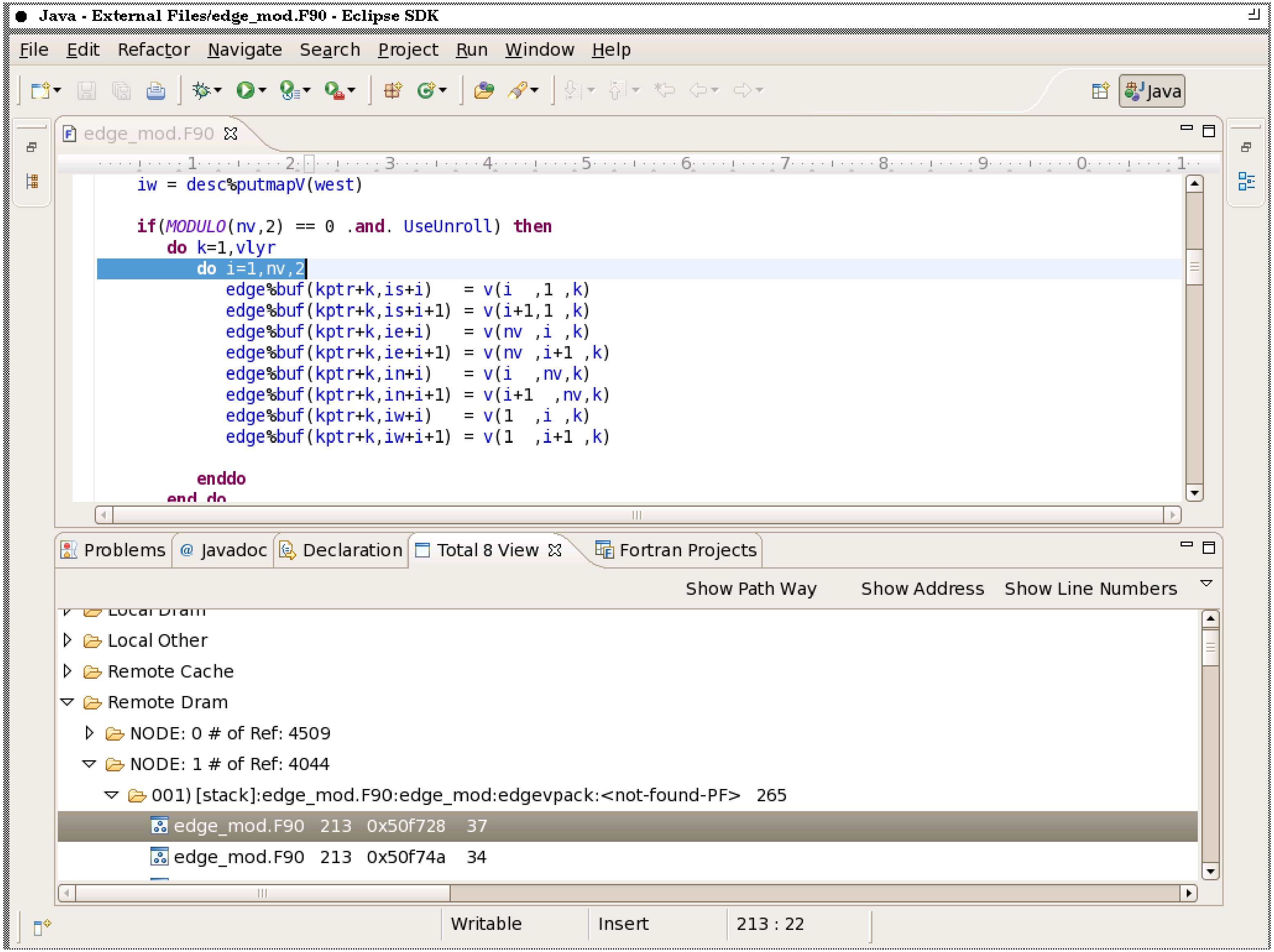
Allocation Instrumentation Tool

- Adds capability to map addresses to dynamically allocated variables
- Based on a Tau tool, built on top of Program Database Toolkit from University of Oregon
- Easily integrated into build process
  - Extra step in the rule to compile F90 files in Makefile
- At runtime, each dynamic allocation dumps variable-to-address-range mapping for use by post-processing tool
- Potential drawbacks
  - Adds overhead to each dynamic allocation
  - Requires access to source (i.e., cannot instrument libraries)
- In practice, benefits significantly outweigh drawbacks

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
  - Several predefined kernel constants and functions not contained in the CNL distribution
  - Required finding and hard-coding values into calls that set configuration registers
- Kernel module port complicated by the black-box nature of CNL (not open-source)
  - Required the help of a patient Cray engineer to perform first half of each iteration of the compile-install-test-modify loop
- Also implemented: mechanism for making *Memphis* available to jobs that want to use it

Eclipse GUI



```
    iw = desc%putmapV(west)

    if(MODULO(nv,2) == 0 .and. UseUnroll) then
       do k=1,vlyr
          do i=1,nv,2
             edge%buf(kptr+k,is+i)   = v(i  ,1 ,k)
             edge%buf(kptr+k,is+i+1) = v(i+1,1 ,k)
             edge%buf(kptr+k,ie+i)   = v(nv ,i ,k)
             edge%buf(kptr+k,ie+i+1) = v(nv ,i+1 ,k)
             edge%buf(kptr+k,in+i)   = v(i  ,nv,k)
             edge%buf(kptr+k,in+i+1) = v(i+1  ,nv,k)
             edge%buf(kptr+k,iw+i)   = v(1  ,i ,k)
             edge%buf(kptr+k,iw+i+1) = v(1  ,i+1 ,k)

          enddo
       end do
```

```
Local Dram
Local Other
Remote Cache
Remote Dram
  NODE: 0 # of Ref: 4509
  NODE: 1 # of Ref: 4044
    001) [stack]:edge_mod.F90:edge_mod:edgevpack:<not-found-PF>  265
      edge_mod.F90  213  0x50f728   37
      edge_mod.F90  213  0x50f74a   34
```

Writable | Insert | 213 : 22

Memphis Evaluation

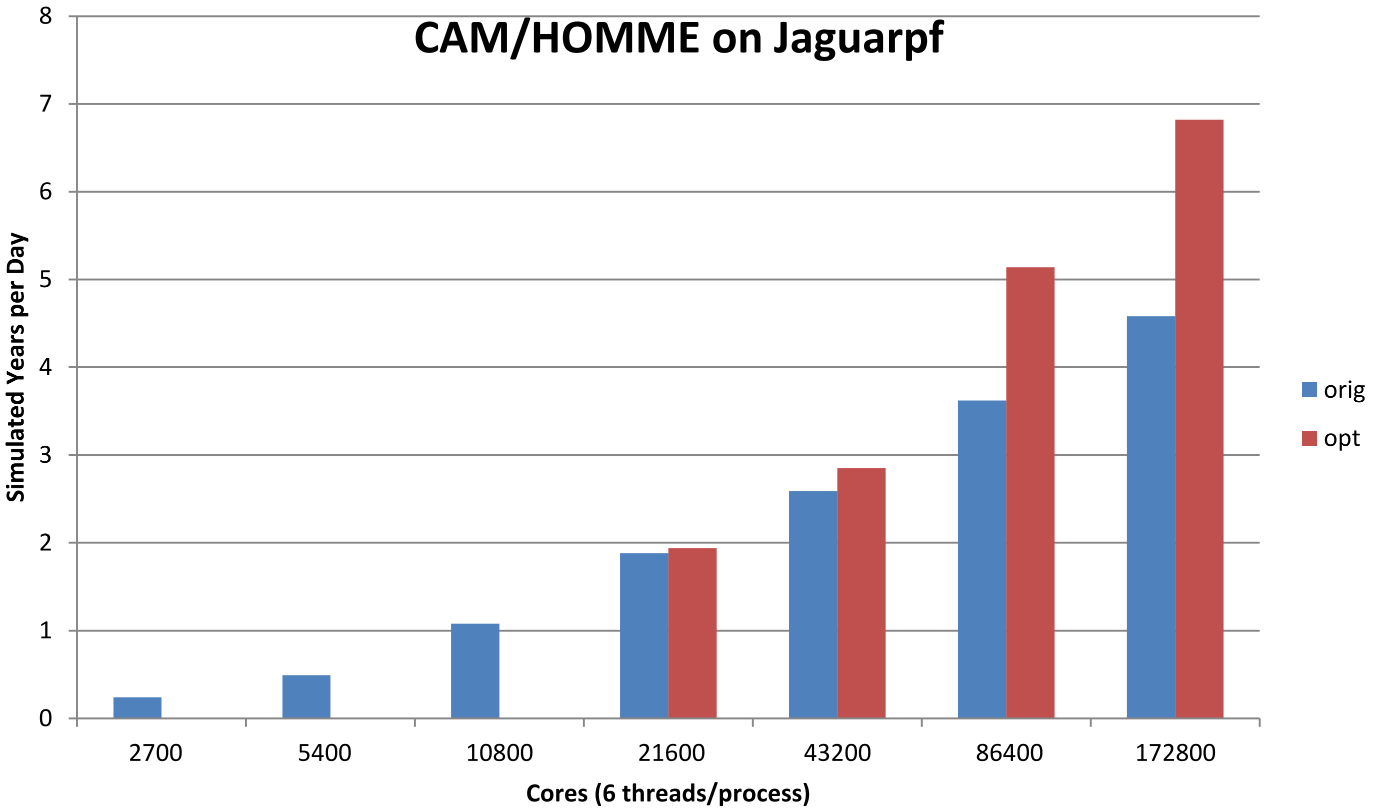
- Quick demonstration of two aspects of ‘performance’
  - Runtime overhead
  - Usefulness: *application* performance improvements

Runtime Overhead

	IBS Off, No Instrumentation	IBS On, Instrumented
Base	40.69	41.18
Mod1	36.29	36.63
Mod2	35.90	36.31

- Even with allocation statements instrumented, overhead is ~1%.

Performance Improvements: CESM



- *Memphis*-directed changes to one file (of *many*).
- Performance of 12 threads (two NUMA nodes) is comparable.

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 an application? Let us know!
- Want *Memphis* on your system? Let us know!

Contacts

Collin McCurdy

(865) 241-6433
cmccurdy@ornl.gov

Jeffery S. Vetter

Future Technologies Group
Computer Science and Mathematics Division
(865) 356-1649
vetter@ornl.gov