Perspectives on Performance Tools for Exascale: Experiences with TAU

Allen D. Malony
Department of Computer and Information Science
University of Oregon
Outline

Part 1: Motivation (5 minutes)
- Performance engineering and productivity
- Role of performance knowledge
- “Extreme” performance engineering

Part 2: TAU Performance System (15 minutes)
- Overview
- Description of main components

Part 3: Experiences (20-25 minutes)
- Performance data mining (NWChem)
- Hybrid/heterogeneous performance analysis (MPAS-O, XGC)
- Communication/computation optimization (IRMHD)
- Understanding performance variability (CESM)
- Empirical autotuning

Part 4: Perspectives (5 minutes)
Parallel Performance Engineering

- Scalable, optimized applications deliver HPC promise
- Optimization through performance engineering process
  - Understand performance complexity and inefficiencies
  - Tune application to run optimally on high-end machines
- How to make the process more effective and productive?
  - What is the nature of the performance problem solving?
  - What is the performance technology to be applied?
- Performance tool efforts have been focused on performance observation, analysis, problem diagnosis
  - Application development and optimization productivity
  - Programmability, reusability, portability, robustness
  - Performance technology part of larger programming system
- Parallel systems evolution will change process, technology, use
Traditionally an empirically-based approach
observation ↔ experimentation ↔ diagnosis ↔ tuning

Performance technology developed for each level

- Performance Observation
- Performance Experimentation
- Performance Diagnosis
- Performance Tuning

Performance Technology
- Instrumentation
- Measurement
- Analysis
- Visualization
- Data mining
- Models
- Expert systems

Performance Technology
- Experimentation management
- Performance storage

Parallel Performance Engineering Process
Application-specific Performance

- What will enhance productive application development with a goal to improve performance optimization?
- Current performance engineering process decouples the application from the performance analysis
  - Little sharing of application knowledge with the tools
- Performance engineering process and tools must be more application-aware
- Support application-specific performance views
  - What are the important events and performance metrics?
  - How are these tied to the application structure and computational model?
  - How can knowledge about the application domain and algorithms be used to improve performance understanding?
Need for Whole Performance Evaluation

- Extreme scale performance is an optimized orchestration
  - Application, processor, memory, network, I/O
- Reductionist approaches to performance will be unable to support optimization and productivity objectives
- Application-level only performance view is myopic
  - Interplay of hardware, software, and system components
  - Ultimately determines how performance is delivered
- Performance should be evaluated *in toto*
  - Application and system components
  - Understand effects of performance interactions
  - Identify opportunities for optimization across levels
- Need *whole performance evaluation practice*
Role of Intelligence, Knowledge, and Automation

- Increased performance complexity forces the engineering process to be more intelligent and automated
  - Automate performance data analysis / mining / learning
  - Automated performance problem identification

- Performance engineering tools and practice must incorporate a performance knowledge discovery process

- Model-oriented knowledge
  - Computational semantics of the application
  - Symbolic models for algorithms
  - Performance models for system architectures / components

- Application developers can be more directly involved in the performance engineering process
Empirical performance data evaluated with respect to performance expectations at various levels of abstraction.
TAU Performance System® (http://tau.uoregon.edu)

- Tuning and Analysis Utilities (20+ year project)
- Performance problem solving *framework* for HPC
  - Integrated, scalable, flexible, portable
  - Target all parallel programming / execution paradigms
- Integrated performance *toolkit*
  - Multi-level performance instrumentation
  - Flexible and configurable performance measurement
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)
- Broad use in complex software, systems, applications
**General Target Computation Model in TAU**

- **Node**: physically distinct shared memory machine
  - Message passing node interconnection network
- **Context**: distinct virtual memory space within node
- **Thread**: execution threads (user/system) in context

![Diagram showing the general target computation model in TAU with nodes, context, threads, and interconnection network.]
TAU Observation Methodology and Workflow

- TAU’s methodology for parallel performance observation is based on the insertion of measurement *probes* into application, library, and runtime system code
  - Code is *instrumented* to make visible certain events
  - Performance measurements occur when events are triggered
  - Known as *probe-based (direct)* measurement

- Performance experimentation workflow
  - Instrument application and other code components
  - Link / load TAU measurement library
  - Execute program to gather performance data
  - Analysis performance data with respect to events
  - Analyze multiple performance experiments
TAU Performance System® Architecture

- Parallel performance framework and toolkit
- Software architecture provides separation of concerns
  - Instrumentation | Measurement | Analysis

** TAU Architecture **

** Instrumentation **
- **Source**
  - C, C++, Fortran
  - Python, UPC, Java
  - Robust parsers (PDT)

- **Wrapping**
  - Interposition (PMPI)
  - Wrapper generation

- **Linking**
  - Static, dynamic
  - Preloading

- **Executable**
  - Dynamic (Dyninst)
  - Binary (Dyninst, MAQAO)

** Measurement **
- **Events**
  - static/dynamic
  - routine, basic block, loop
  - threading, communication
  - heterogeneous

- **Profiling**
  - flat, callpath, phase, parameter, snapshot
  - probe, sampling, hybrid

- **Tracing**
  - TAU / Scalasca tracing
  - Open Trace Format (OTF)

- **Metadata**
  - system, user-defined

** Analysis **
- **Profiles**
  - ParaProf parallel profile analyzer / visualizer
  - TAUdb parallel profile database
  - PerfExplorer parallel profile data mining

- **Tracing**
  - TAU trace translation
    - OTF, SLOG-2
  - Trace analysis / visualizer
    - Vampir, Jumpshot

- **Online**
  - event unification
  - statistics calculation
TAU Components

- Instrumentation
  - Fortran, C, C++, Python, Java, UPC, Chapel
  - Source, compiler, library wrapping, binary rewriting
  - Automatic instrumentation

- Measurement
  - Internode: MPI, OpenSHMEM, ARMCI, PGAS, DMAPP
  - Intranode: Pthreads, OpenMP, hybrid, …
  - Heterogeneous: GPU, MIC, CUDA, OpenCL, OpenACC, …
  - Performance data (timing, counters) and metadata
  - Parallel profiling and tracing (with Score-P integration)

- Analysis
  - Parallel profile analysis and visualization (ParaProf)
  - Performance data mining / machine learning (PerfExplorer)
  - Performance database technology (TAUdb)
  - Empirical autotuning
TAU Instrumentation Approach

- Direct and indirect performance instrumentation
  - Direct instrumentation of program (system) code (probes)
  - Indirect support via sampling or interrupts
- Support for standard program code events
  - Routines, classes and templates
  - Statement-level blocks, loops
  - Interval events (start/stop)
- Support for user-defined events
  - Interval events specified by user
  - Atomic events (statistical measurement at a single point)
  - Context events (atomic events with calling path context)
- Provides static events and dynamic events
- Instrumentation optimization
TAU Instrumentation Mechanisms

- Source code
  - Manual (TAU API, TAU component API)
  - Automatic (robust)
    - C, C++, F77/90/95, OpenMP (POMP/OPARI), UPC
  - Compiler (GNU, IBM, NAG, Intel, PGI, Pathscale, Cray, …)
- Object code (library-level)
  - Statically- and dynamically-linked wrapper libraries
    - MPI, I/O, memory, …
  - Powerful library wrapping of external libraries without source
- Executable code / runtime
  - Runtime preloading and interception of library calls
  - Binary instrumentation (Dyninst, MAQAO, PEBIL)
  - Dynamic instrumentation (Dyninst)
  - OpenMP (runtime API, CollectorAPI, GOMP, OMPT)
- Virtual machine, interpreter, and OS instrumentation
Instrumentation for Wrapping External Libraries

- Preprocessor substitution
  - Header redefines a routine with macros (only C and C++)
  - Tool-defined header file with same name takes precedence
  - Original routine substituted by preprocessor callsite

- Preloading a library at runtime
  - Library preloaded in the address space of executing application intercepts calls from a given library
  - Tool wrapper library defines routine, gets address of global symbol (dlsym), internally calls measured routine

- Linker-based substitution
  - Wrapper library defines wrapper interface which calls real routine
  - Linker is passed option to substitute all references from applications object code with tool wrappers
Automatic Source-level and Wrapper Instrumentation

PDT source analyzer

Parsed program

Instrumentation specification file

BEGIN_EXCLUDE_LIST
Foo
Bar
D#EMM
END_EXCLUDE_LIST

BEGIN_FILE_EXCLUDE_LIST
f*.f90
Foo?.cpp
END_FILE_EXCLUDE_LIST

BEGIN_FILE_INCLUDE_LIST
main.cpp
foo.f90
END_FILE_INCLUDE_LIST

Application source

tau_instrumentor

tau_wrap

Instrumented source
TAU Measurement Approach

- Portable and scalable parallel profiling solution
  - Multiple profiling types and options
  - Event selection and control (enabling/disabling, throttling)
  - Online profile access and sampling
  - Online performance profile overhead compensation

- Portable and scalable parallel tracing solution
  - Trace translation to OTF, EPILOG, Paraver, and SLOG2
  - Trace streams (OTF) and hierarchical trace merging

- Robust timing and hardware performance support

- Multiple counters (hardware, user-defined, system)

- Metadata (hardware/system, application, …)
TAU Measurement Mechanisms

- Parallel profiling
  - Function-level, block-level, statement-level
  - Supports user-defined events and mapping events
  - Support for flat, callgraph/callpath, phase profiling
  - Support for parameter and context profiling
  - Support for tracking I/O and memory (library wrappers)
  - Parallel profile stored (dumped, snapshot) during execution

- Tracing
  - All profile-level events
  - Inter-process communication events
  - Inclusion of multiple counter data in traced events
Performance Analysis

- Analysis of parallel profile and trace measurement
- Parallel profile analysis (*ParaProf*)
  - Java-based analysis and visualization tool
  - Support for large-scale parallel profiles
- Performance data management (*TAUdb*)
- Performance data mining (*PerfExplorer*)
- Parallel trace analysis
  - Translation to VTF (V3.0), EPILOG, OTF formats
  - Integration with Vampir / Vampir Server (TU Dresden)
- Integration with CUBE browser (Scalasca, UTK / FZJ)
- Scalable runtime fault isolation with callstack debugging
- Efficient parallel runtime bounds checking
Profile Analysis Framework

Graphical representation of the Profile Analysis Framework, showing the flow of data from Performance Data to the Profile Analysis Framework (PerfDMF) and then to ParaProf, with various tools and features such as Call Graphs, Histograms, Call Trees, Bar Charts, Comparative Displays, Text Displays, and a 3D Displays module. The framework includes data collection tools like Supermon, MRNet, DBMS (PostgreSQL, MySQL, Oracle, DB2, Derby), and scripting interface tools like Jython.
Performance Data Management

- Provide an open, flexible framework to support common data management tasks
  - Foster multi-experiment performance evaluation
- Extensible toolkit to promote integration and reuse across available performance tools (PerfDMF)
  - Supported multiple profile formats: TAU, CUBE, gprof, mpiP, psrun, …
  - Supported DBMS: PostgreSQL, MySQL, Oracle, DB2, Derby, H2
- Re-engineer in TAUdb
TAUdb Database Schema

- Parallel performance profiles
- Timer and counter measurements with 5 dimensions
  - Physical location: process / thread
  - Static code location: function / loop / block / line
  - Dynamic location: current callpath and context (parameters)
  - Time context: iteration / snapshot / phase
  - Metric: time, HW counters, derived values
- Measurement metadata
  - Properties of the experiment
  - Anything from name:value pairs to nested, structured data
  - Single value for whole experiment or full context (tuple of thread, timer, iteration, timestamp)
Performance Data Mining / Analytics

- Conduct systematic and scalable analysis process
  - Multi-experiment performance analysis
  - Support automation, collaboration, and reuse
- Performance knowledge discovery framework
  - Data mining analysis applied to parallel performance data
    - parametric, comparative, clustering, correlation, …
  - Integrate available statistics and data mining packages
    - Weka, R, Matlab / Octave
  - Apply data mining operations in interactive environment
  - Meta-analysis based on metadata collection in TAU
    - hardware/system, application, user, …
PerfExplorer Performance Data Mining

- Programmable, extensible framework to support workflow automation
- Rule-based inference for expert system analysis
TAUdb Tool Support

- ParaProf
  - Parallel profile viewer / analyzer
  - 2, 3+D visualizations
  - Single experiment analysis

- PerfExplorer
  - Data mining framework
    - Clustering, correlation
  - Multi-experiment analysis
  - Scripting engine
  - Expert system
TAU Availability on New Systems

- Intel compilers with Intel MPI on Intel Xeon Phi™ (MIC)
- GPI with Intel Linux x86_64 InfiniBand clusters
- IBM BG/Q and Power 7 Linux with IBM XL compilers
- NVIDIA Kepler K20 with CUDA 5.5 with NVCC
- Fujitsu Fortran/C/C++ MPI compilers on the K computer
- PGI compilers with OpenACC support on NVIDIA systsems
- Cray CX30 Sandybridge Linux systems with Intel compilers
- AMD OpenCL libs with GNU on AMD Fusion cluster
- MPC compilers on TGCC Curie system (Bull, Linux x86_64)
- GNU on ARM Linux clusters (MontBlanc, Beacon, Stampede)
- Cray CCE compilers with OpenACC on Cray XK6, XK7
- Microsoft MPI w/ Mingw compilers on Windows Azure
- LLVM and GNU compilers under Mac OS X, IBM BG/Q
Common Infrastructure Integration – Score-P

- Community effort for a common tools infrastructure
  - Starting with TAU, Periscope, Scalasca, and Vampir
  - Open for other tools and groups

- Joint development of Score-P
  - Core performance measurement infrastructure
  - MPI, OpenMP, heterogeneous

- DOE-funded PRIMA project
  - University of Oregon
  - Forschungszentrum Jülich

- BMBF-funded SILC project (multiple partners)
Score-P with TAU Integration and Components

- Score-P software architecture

![Diagram showing the integration of Vampir, Scalasca, TAU, and Periscope with Score-P measurement infrastructure]

- Event traces (OTF2 format)
- Call-path profiles (CUBE4 and TAU formats)
- Online interface

- Score-P measurement infrastructure

- Hardware counters
- Memory management
- Other...

- Compiler
- TAU instrumentor
- OPARI 2
- User

- Instrumentation

- Target application (MPI, OpenMP, hybrid, serial)
NWChem Case Studies

- NWChem is a leading chemistry modeling code
- NWChem relies on Global Arrays (GA)
  - Provides a global view of a physically distributed array
  - One-sided access to arbitrary patches of data
  - Developed as a library (fully interoperable with MPI)
- Aggregate Remote Memory Copy Interface (ARMCI)
  - GA communication substrate for one-sided communication
  - Portable high-performance one-sided communication library
  - Rich set of remote memory access primitives
- Would like to better understand the performance of representative workloads for NWChem on different platforms
  - Help to create use cases for one-side programming models
NWChem One-sided Communication and Scaling

- Understand interplay between data-server and compute processes as a function of scaling
  - Data-server uses a separate thread
  - Large numerical computation per node at small scale can obscure the cost of maintaining passive-target progress
  - Larger scale decreases numerical work per node and increases the fragmentation of data, increasing messages
  - Vary #nodes, cores-per-node, and memory buffer pinning

- Understand trade-off of core allocation
  - All to computation versus some to communication

NWChem Instrumentation

- Source-base instrumentation of NWChem application
- Developed an ARMCI interposition library (PARMCI)
  - Defines weak symbols and name-shifted PARMCI interface
  - Similar to PMPI for MPI
- Developed a TAU PARMCI library
  - Intervals events around interface routines
  - Atomic events capture communication size and destination
- Wrapped external libraries
  - BLAS (DGEMM)
- Need portable instrumentation for cross-platform runs
- Systems
  - Fusion: Linux cluster, Pacific Northwest National Lab
  - Intrepid: IBM BG/P, Argonne National Lab

(Note: Runs on Hopper and Mira will scale greater, but will possibly show similar effects.)
FUSION Tests Comparing No Pinning vs. Pinning

- Scaling on 24, 32, 48, 64, 96 and 128 nodes
- Test on 8 cores (no separate data server thread)
- With no pinning ARMCI communication overhead increases dramatically and no scaling is observed
- Pinning communication buffers shows dramatic effects
- Relative communication overhead increases, but not dramatically
Intrepid Tests Comparing No Pinning vs. Pinning

- Scaling on 64, 128, 256 and 512 nodes
- Tests with interrupt or communication helper thread (CHT)
  - CHT requires a core to be allocated
- ARMCI calls are barely noticeable
- DAXPY calculation shows up more
- CHT performs better in both SMP and DUAL modes
Electronic Structure in Computational Chemistry

- Parallel performance is determined by:
  - How the application is design and developed
  - The nature and characteristics of the problem

- Computational chemistry applications can exhibit:
  - Highly symmetric diverse load (e.g., Benzene)
  - Asymmetric unpredictable load (e.g., water clusters)
  - QM/MM sheer large size (e.g., macro-molecules)

- Load balance is crucially important for performance
Focus on NXTVAL
- Atomic counter keeping track of which global tasks sent

Strong scaling experiment
- 14 water molecules
  - aug-cc-PVDZ dataset
- 124 nodes on ANL Fusion
  - 8 cores per node
- NXTVAL significant %
- Increasing per call time

When arrival rate exceeds processing rate, buffering and flow control must be used
Evaluation of Inspector-Executor Algorithm

- How to eliminate the overhead of centralized load balance algorithm based on NXTVAL
- Use an inspector-executor approach to assigning tasks
  - Assess task imbalance
  - Reassign
- Use TAU to evaluate performance improvement with respect to NXTVAL, overhead, task balance

Refinement from NXTVAL to Inspector/Executor

- Original NXTVAL measured
- Original NXTVAL reduced
- Inspector/Executor 1st iteration overhead
- Inspector/Executor subsequent iterations
MPAS (Model for Prediction Across Scales)

- MPAS is an unstructured grid approach to climate system modeling
  - Explore regional-scale climate change
  
- MPAS supports both quasi-uniform and variable resolution meshing of the sphere
  - Quadrilaterals, triangles, or Voronoi tessellations

- MPAS is a software framework for the rapid prototyping of single components of climate system models
  - Two SciDAC earth systems codes (dynamical cores)
    - MPAS-O (ocean model)
    - CAM-SE (atmosphere model)

http://mpas-dev.github.io
MPAS-Ocean (MPAS-O) Overview

- MPAS-O is designed for the simulation of the ocean system from time scales of months to millenia and spatial scales from sub 1 km to global circulations
- MPAS-O has demonstrated the ability to accurately reproduce mesoscale ocean activity with a local mesh refinement strategy
- In addition to facilitating the study of multiscale phenomena within ocean systems, MPAS-O is intended for the study of anthropogenic climate change as the ocean component of climate system models
Multiscale and MPAS-O Domain Decomposition

- Use multiscale methods for accurate, efficient, and scale-aware models of the earth system.
- MPAS-O uses a variable resolution irregular mesh of hexagonal grid cells.
- Cells assigned to MPI processes, grouped as “blocks”.
  - Each cell has 1-40 vertical layers, depending on ocean depth.
- MPAS-O has demonstrated scaling limits when using MPI alone.
- Look at increasing concurrency.
- Developers currently adding OpenMP.
  - Both split explicit and RK4 solvers.
MPAS-Ocean Performance Study

- Integrate TAU into the MPAS build system
  - Evaluate the original MPI-only approach
  - Study the new MPI+OpenMP approach
- Performance results
  - MPI block + OpenMP element decomposition
    - Reduces total instructions in computational regions
    - ~10% faster than MPI alone
  - Guided OpenMP thread schedule balances work across threads
    - ~6% faster than default
  - Weighted block decomposition using vertical elements (depth) could balancing work across processes (~5% faster in some tests)
  - Overlapping communication and computation could reduce synchronization delays when exchanging halo regions (underway)
- Evaluation is ongoing and includes porting to MIC platform
MPI Scaling Study (Hopper)

- Strong scaling
  - 192 to 16800

- Poor scaling over 6144 processes
  - Communication begins to dominate
  - Problem size might be too small

- Time profile by events
  - Only MPI events > 5% after 12K processes
Benefits of RK4 solver and OpenMP Scheduling

- Use modified RK4 solver versus original MPI solver
- Test cases
  - 64 cores
    - 64 processes (MPI-only)
    - 32 processes x 2 threads
    - 16 processes x 4 threads
  - 96 cores
    - 96 processes (MPI-only)
    - 48 processes x 2 threads
    - 32 processes x 3 threads
    - 16 processes x 6 threads
- OpenMP scheduling options
Benefits of Guided Scheduling with MPAS-O RK4

- MPI + OpenMP experiment
  - 96 cores (16 processes by 6 threads per process)
- Default scheduling generates a load imbalance
  - Threads arrive at boundary cell exchange at different times
  - Complete boundary cell exchange at different rates
    - only the master thread performs the exchange
- Guided scheduling provides better balance
MPAS-O on BG/Q (Vesta) on 16K (256x64)

MPI_Wait() master thread halo exchanges

OpenMP worker threads executing solver routines, including time spent waiting on halo exchanges by master threads
XGC for Studying Critical Edge Physics

- XGC is a set of codes to model gyrokinetic microturbulence in the simulation of fusion reactors
  - DOE SciDAC for Edge Physics Simulation (EPSi)
  - Particle-in-cell (PIC) codes:
    - XGC0, XGC1, XGC1a, XGC2, ...
- XGC1
  - ODE-based PIC approach on space grid using unstructured triangular grid
  - 5D gyrokinetic equations
    - ODE: time advance of marker particles
    - finite difference: partial integro-differential Fokker-Planck collision operator discretized on rectangular v-space grid
    - PDE (PETSc): Maxwell’s equations on unstructured triangular x-space grid
  - Usual interpolation issue exists
XGC1 Development and Performance

- Development of electromagnetic turbulence capability
  - Requires more accurate calculation of electrical current
  - More particles requires greater scaling

- Challenge to run XGC1 on heterogeneous systems (Titan)
  - Electron subcycling time-advance takes ~85% of total time
    - Designed to be without external communication in each cycle
    - Ideal for occupying GPUs while other routines use CPUs
    - Solver spend <5% of total computing time

- Weak scaling in number of particles
  - XGC1 has not reach the MPI communication limit
  - #particles per grid-node is fixed
  - #grid-nodes memory determined by compute-node memory
  - Need more compute nodes
EPSi with DOE SUPER institute is optimizing XGC1
- Computational kernel including port to GPU
- OpenMP parallelism
- MPI communication

Use TAU to investigate XGC1 performance questions:
- How to efficiently split work between the CPU and GPU
- How to improve the cache performance in the GPU
- Effects of memory copying with multiple GPUs per node

Experiments done on ACISS and Titan
- ACISS: 12-core 2.67GHz Xeon X5650 / Tesla M2070 (3x)
- Titan: 16-core 2.2GHz Opteron 6274 / Tesla K20X
Look at CPU+GPU worksharing execution scenario

Electron 'push' takes 85% of the computation time

Kernel (*pushe2*) was ported to the GPU

Assigning more work to the GPU can shorten the execution since it can compute the kernel faster
XGC Cache Optimization (ACISS)

- Shared memory capacity
  - L1 cache and device shared memory share physical storage
  - Default: 48KB shared memory / 16KB L1 cache
  - CUDA allows you to swap this allocation
  - Improved the performance of the *pushe2* kernel (~20%)
  - Improved the performance of XGC application (~17%)

![Graph showing performance metrics for different benchmarks using TAU](attachment:image.png)
Memory Copies with Multiple GPUs in XGC

- Given multiple GPUs per node in ACISS
  - XGC could be run with multiple MPI ranks per node
  - Each rank would be assigned 1 GPU
- How do CPU-GPU memory copies affect performance?
  - Compare
    - 1 MPI rank per node with 1 GPU
    - 3 MPI ranks per node with 3 GPUs
  - Slow memory copies result with multiple ranks+GPUs
- Slow memory copies can lead to MPI waiting
  - Leads to a slow down in MPI collectives
XGC Performance with Different # GPUs (ACISS)

- 3 MPI ranks
  - 3 GPUs per node (6 seconds)
  - Slow memory copies …
  - … cause faster MPI ranks to stall at MPI_Reduce

- 3 MPI ranks (different nodes)
  - 1 GPU per node (4 seconds)
  - Fast memory copies better aligns processes and reduces MPI stalls
XGC MPI Imbalance Due to PCI Bus Sharing

- 16 ACISS nodes
  - 3 GPUs per node (6 seconds)
- Slow memory copies …
- … cause faster MPI ranks to stall at MPI_Reduce
Heterogeneous Performance Measurement (Titan)

- CPU+GPU worksharing execution scenario

**Metric:** TAUGPU_TIME  
**Value:** Exclusive  
**Units:** seconds

82.498  
53.885

- TAU application  
- CPU_Work  
- cuStreamSynchronize  
- MPI_Allreduce()  

**Computation Kernels**

63.97  
32.11

- pushe_kernel_gpu  
- TAU application  
- Memory copy Host to Device  
- Memory copy Device to Host

2.2429E7  

- Bytes copied from Device to Host

**CPU Events**

**GPU Events**

**Memory Copy Events**
XGC Increasing Grid Size (Titan)

- Allocating a larger grid to improve GPU efficiency

Metric: TAUGPU_TIME
Value: Exclusive
Units: seconds

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>Mean Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>64x1x1</td>
<td>63.949</td>
</tr>
<tr>
<td>384x1x1</td>
<td>53.573 (83.775%)</td>
</tr>
</tbody>
</table>

Metric: CUDA.Tesla_K20X.domain_.d.active_cycles_.(averaged)_.(upper bound)
Value: Exclusive
Units: counts

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>64x1x1</td>
<td>1.9163E13</td>
</tr>
<tr>
<td>384x1x1</td>
<td>1.6329E13 (85.207%)</td>
</tr>
</tbody>
</table>

- Larger grid size increases the number of active cycles by 15% and leads to a 5% increase in total SM efficiency
XGC1 Performance Scaling on Titan

- EPSi-SUPER collaboration
- Weak scaling on DIII-D grid
  - 10 ion timesteps
  - 3.2 million ions
  - 3.2 million electrons per node
- Ran 512 node experiment
IRMHD Performance on Argonne Intrepid and Mira

.incite magnetohydrodynamics simulation to understand solar winds and coronal heating
   ✗ First direct numerical simulations of Alfvén wave (AW) turbulence in extended solar atmosphere accounting for inhomogeneities
   ✗ Team
     ➢ University of New Hampshire (Jean Perez and Benjamin Chandran)
     ➢ ALCF (Tim Williams)
     ➢ University of Oregon (Sameer Shende)

 irmhd (inhomogeneous reduced magnetohydrodynamics)
   ✗ Fortran 90 and MPI
   ✗ Excellent weak and strong scaling properties
   ✗ Tested and benchmarked on Intrepid and Mira

 hpc source article and ALCF news
Communication Analysis (MPI_Send, MPI_Bcast)

- IRMHD demonstrated performance behavior consistent with common synchronous communication bottlenecks
  - Significant time spent in MPI routines
- Identify problems on a 2,048-core execution on BG/P
  - MPI_Send and MPI_Bcast took significant time
  - Suggest possible opportunities for overlapping computation and communication
  - Identified possible targets for computation improvements
Effects of IRMHD Optimizations

- Developed a non-blocking communication substrate
- Deployed a more efficient implementation of the underlying FFT library
- Overall execution time reduced from 528.18 core hours to 70.85 core hours (>7x improvement) for a 2,048-processor execution on Intrepid
- Further improvement on Mira …
Mira (BG/Q) Performance

- Test with 32K MPI ranks
- Load imbalance apparent
  - See imbalance reflected in MPI_Alltoall() performance
Optimizations on Mira

- Remove MPI_Barrier in regions where not needed

- Next, oversubscribe nodes ...
Oversubscribing Mira Nodes

- Vary #MPI ranks on nodes (1024 nodes)
  - 16 ranks per node (16K) versus 32 ranks per node (32K)

- Overall time improvement
  - 71.23% of original

- More efficient barriers within node leads to performance improvement
Performance Variability in CESM

- Community Earth System Model (CESM)
- Observed performance variability on ORNL Jaguar
  - Significant increase in execution time led to failed runs
- End-to-end analysis methodology
  - Collect information from all production jobs
    - modify jobs scripts
    - problem/code provenance, system topology, system workload, process node/core mapping, job progress, time spent in queue, pre/post, total
  - Load in TAUdb, qualify nature of variability and impact
Example Experiment Case Analysis

- 4096 processor cores
  - 6 hour request
  - Target: 150 simulation days
- 35 jobs
  - May 15 – Jun 29, 2012
  - Two of which failed

Minimum execution time  Maximum execution time

High variability intra-run
TAUdb and CCSM Data – Mismatched Nodes

Value Chart: TIME

Slowest – 272 unmatched

Fastest – 0 unmatched
Extents of Gemini Network Matter
MPI Rank Placement Matters Too

Fastest case: 4:14:08 (hh:mm:ss)

Slowest case: 5:35:50 (hh:mm:ss)
Autotuning is a Performance Engineering Process

- Autotuning methodology incorporates aspects common to “traditional” application performance engineering
  - Empirical performance observation
  - Experiment-oriented
- Autotuning embodies progressive engineering techniques
  - Automated experimentation and performance testing
  - Guided optimization by (intelligent) search space exploration
  - Model-based (domain-specific) computational semantics
- However, autotuning is based on optimization and search, not performance diagnosis
- There are shared objectives for performance technology and opportunities for tool integration
TAU Integration with Empirical Autotuning

- Goal is to integrate TAU with existing autotuning frameworks
  - Use TAU to gather performance data for autotuning/specialization
  - Store performance data with metadata for each experiment variant and store in performance database (TAUdb)
  - Use machine learning and data mining to increase the level of automation of autotuning and specialization

- Autotuning components
  - Active Harmony autotuning system (Hollingsworth, UMD)
    - software architecture for optimization and adaptation
  - CHiLL compiler framework (Hall, Utah)
    - CPU and GPU code transformations for optimization
  - Orio annotation-based autotuning (Norris, ANL)
    - code transformation (C, Fortran, CUDA) with optimization
Autotuning Integration (CHiLL + AH)

Measurement
- Parameter profiling
- TAUdb storage

Process
- search
- transformation
- measurement
- storage
- feedback

Source Code
- ROSE outliner
  - Outlined Function
    - Selective Instrumentation File (specifying parameters to capture)
      - tau_instrumentor
        - Instrumented Function
          - execute
            - Parameterized Performance Profile
              - TAUdb

tau_instrumentor

CHiLL Recipes
- parameters from TAUdb
  - Outlined Function
    - Search Driver (brute force or Active Harmony)
      - CHiLL
        - code variant
          - Selective Instrumentation File (specifying parameters to capture)
            - tau_instrumentor
              - Instrumented Variant
                - execute
                  - Parameterized Performance Profile
                    - TAUdb
Autotuning with TauDB Methodology

- Each time the program executes a code variant, we store metadata in the performance database indicating by what process the variant was produced:
  - Source function
  - Name of CHiLL recipe
  - Parameters to CHiLL recipe
- The database also contains metadata on what parameters were called and also on the execution environment:
  - OS name, version, release, native architecture
  - CPU vendor, ID, clock speed, cache sizes, # cores
  - Memory size
  - Any metadata specified by the end user
Learning Performance Specialization

- Apply machine learning to data stored in TAUdb
  - Generate decision trees based upon code features

```c
void matmult(float **c, float **a, float **b, int L, int M, int N)
parameterize using L, M, N
```
Decision Code Generation to Effect Specialization

- Use a ROSE-based tool to generate a wrapper function
  - Carries out the decisions in the decision tree and executes the best code variant
- Decision tree code generation tool takes Weka-generated decision tree and a set of decision functions
  - If using custom metadata, user needs to provide a custom decision function
  - Decision functions for metadata automatically collected by TAU are provided
Orio

- Orio is an annotation-based empirical performance tuning framework
- Source code annotations allow Orio to generate a set of low-level performance optimizations
  - After each optimization (or transformation) is applied the kernel is run
  - Set of optimizations is searched for the best transformations to be applied for a given kernel
- First effort to integrate Orio with TAU was to collect performance data about each experiment that Orio runs
  - Move performance data from Orio into TAUdb
  - Orio reads from TAUdb
TAU's GPU Measurement Library

- Focused on Orio’s CUDA kernel transformations
- TAU uses NVIDIA’s CUPTI interface to gather information about the GPU execution
  - Memory transfers
  - Kernels
    - runtime performance
    - performance attributes
  - GPU counters
- Using the CUPTI interface does not require any recompiling or re-linking of the application
ORIO data in TAUdb / PerfExplorer

- Orio tuning of a simple 3D vector multiplication
- 2,048 experiments fed into TAUdb
  - PerfExplorer with Weka to do component analysis
The first person approach is problematic for exascale use
- Highly concurrent and dynamic execution model
  - post-mortem analysis of low-level data prohibitive
- Interactions with scarce and shared resources
  - introduces bottlenecks and queues on chip/node and between nodes
- Multiple objectives (performance, energy, resilience, …)
- Runtime adaptation to address dynamic variability

Third person measurement model (in addition) required
- Focus is on system activity characterization at different levels
  - system resource usage is a primary concern
- Measurements are analyzed relative to contributors
- Online analysis and availability of performance allows introspective adaptation for objective evaluation and tuning
Exascale “Performance” Observability

- Exascale requires a new, fundamentally different “performance” observability paradigm
  - Designed specifically to support introspective adaptation
  - Reflective of computation model mapped to execution model
  - Aware of multiple objectives (“performance”)
    - system-level resource utilization data and analysis, energy consumption, and health information available online

- Key parallel “performance” abstraction
  - Inherent state of exascale execution is dynamic
  - Embodies non-stationarity of “performance”
  - Constantly shaped by the adaptation of resources to meet computational needs and optimize execution objectives
Idea – Integration in Exascale Software Stack

- Exascale observability framework can be specialized through top-down and bottom-up programming specific to application
- Enables top-down application transformations to be optimized for runtime and system layers by feeding back dynamic information about HW/SW software resources from bottom-up
- Exascale programming methodology can be opened up to include observability awareness and adaptability
  - Programming system exposes alternatives to the exascale system
    - parameters, algorithms, parallelism control, …
  - Runtime state awareness can be coupled with application knowledge for self-adaptive, closed-loop runtime tuning
    - richer contextualization and attribution of performance state
- Performance portability through dynamic adaptivity