Integrated Performance Views in Charm++: Projections meets TAU

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

Chee Wai Lee
Laxmikant V. Kale
Department Computer Science
University of Illinois
Urbana-Champaign
Outline

- Motivation for integrated performance views
- Charm++ background
  - Performance events
- Charm++ performance framework
  - Callback-based performance module and Projections
- Brief introduction to TAU performance system
- Development of TAU performance module
- NAMD performance case study
  - Demonstrate integrate performance views
- New results
- Conclusions and future work
Productivity and Performance

- High-level parallel paradigms improve productivity
  - Rich abstractions for application development
  - Hide low-level coding and computation complexities

- Natural tension between powerful development environments and ability to achieve high performance

- General dogma
  - Further the application is removed from raw machine the more susceptible to performance inefficiencies
  - Performance problems and their sources become harder to observe and to understand

- Dual goals of productivity and performance require performance tool integration and language knowledge
Challenges

- Provide performance tool access to execution events of interest from different levels of language and runtime
  - Used to trigger performance measurements to record metrics specific to event semantics
  - Event observation supported as part of execution model
- Enable different performance perspectives
  - Build measurement techniques and runtime support that can integrate multiple performance technologies
- Map low-level performance data to high-level parallel abstractions and language constructs
  - Incorporate event knowledge and computation model
  - Identify performance factors at meaningful level
- Open tools to enable integration and long-term support
Charm++ Background

- Parallel object-oriented programming based on C++
- Programs decomposed into set of parallel communicating objects (*chares*)
  - Runtime system maps *chares* onto parallel processes/threads

![Diagram showing programmer (logical) view and parallel system (implementation) view of Charm++](image-url)
Charm++ Computation Model

- Object entry method invocation triggers computation
  - Entry method message for remote process queued
  - Messages scheduled by Charm++ runtime scheduler
  - Entry methods executed to completion
  - May call new entry methods and other routines
Charm++ Performance Events

- Several points in runtime system to observe execution
  - Make performance measurements (*performance events*)
  - Obtain information on execution context

- Charm++ events
  - Start of an entry method
  - End of an entry method
  - Sending a message to another object
  - Change in scheduler state:
    - *active* to *idle*
    - *idle* to *active*

- Observation of multiple events at different levels of abstraction are needed to get full performance view
Charm++ Performance Framework

- How parallel language system operationalizes events is critical to building an effective performance framework

- Charm++ implements *performance callbacks*
  - Runtime system calls performance module(s) at events
  - Any registered performance module (*client*) is invoked
  - Event ID and default performance data forwarded
  - Clients can access to Charm++ internal runtime routines

- Performance framework exposes set of key runtime events as a base C++ class
  - Performance modules inherit and implement methods
  - Listen only to events of interest

- Framework calls performance client initialization
Charm++ Performance Framework Interface

// Base class of all tracing strategies.
class Trace {
  // creation of message(s)
  virtual void creation(envelope *, int epIdx, int num=1) {}
  virtual void creationMulticast(envelope *, int epIdx, int num=1,
                                  int *pelist=NULL) {}
  virtual void creationDone(int num=1) {}
  virtual void beginExecute(envelope *) {}
  virtual void beginExecute(CmiObjId *tid) {}
  virtual void beginExecute(
    int event,       // event type defined in trace-common.h
    int msgType,    // message type
    int ep,         // Charm++ entry point
    int srcPe       // Which PE originated the call
    int ml,         // message size
    CmiObjId* idx)  // index
  {}
  virtual void endExecute(void) {}
  virtual void beginIdle(double curWallTime) {}
  virtual void endIdle(double curWallTime) {}
  virtual void beginComputation(void) {}
  virtual void endComputation(void) {}
};
Charm++ Performance Framework and Modules

- Framework allows for separation of concerns
  - Event visibility
  - Event measurement
- Allows measurement extension and customization
- New modules may introduce new observation requirements
TAU Integration in Charm++

- **Goal**
  - Extend Projections performance measurement
    - Tracing and summary modules
  - Enable use of TAU Performance System® for Charm++
  - Demonstrate utility of alternate methods and integration
    - TAU profiling capability
    - address tracing overhead issues

- **Leverage Charm++ performance framework**
  - Merge TAU performance model with Projections

- **Apply to Charm++ applications**
  - NAMD
  - OpenAtom, ChaNGa
TAU Performance System®

- Integrated toolkit for performance problem solving
  - Instrumentation, measurement, analysis, visualization
  - Portable performance profiling and tracing facility
  - Performance data management and data mining
- Based on direct performance measurement approach
- Open source
- Available on all HPC platforms
TAU Performance Profiling

- Performance with respect to nested event regions
  - Program execution event stack (begin/end events)
- Profiling measures inclusive and exclusive data
- Exclusive measurements for region only performance
- Inclusive measurements includes nested “child” regions
- Support multiple profiling types
  - Flat, callpath, and phase profiling
TAU Module for Charm++ Performance Interface

- Events of interest
  - *Main*: scheduler is active and processing messages
  - *Idle*: scheduler wait state
  - Entry method events (identified by entry name)
  - User program events and MPI events
    - instrumented using TAU API

- Questions
  - What is the top-level event?
    - Scheduler regarded as top-level (*Main* is top-level event)

- Measurement
  - Execution time
  - Hardware counters
TAU Performance Overhead

- Measure module overhead with test program
  - Different instrumentation scenarios

- Overhead depends on several factors
  - Proportional to number of events collected

- Look at overhead per method event

<table>
<thead>
<tr>
<th></th>
<th>No measurement module</th>
<th>TAU module</th>
<th>Projections module</th>
<th>TAU and Projections modules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Charm++ fully optimizimized</td>
<td>0.09µs</td>
<td>0.49µs</td>
<td>2.52µs</td>
</tr>
<tr>
<td></td>
<td>Null trace module loaded</td>
<td>0.44µs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>with [NOTRACE] option</td>
<td>0.55µs</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>with selective instrumentation</td>
<td>0.74µs</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>with fastest available timers</td>
<td>1.03µs</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>with GET_TIME_OF_DAY() timers</td>
<td>1.21µs</td>
<td></td>
</tr>
</tbody>
</table>

ICPP 2009 Integrated Performance Views in Charm++
## TAU Module Validation

- Validate TAU performance measurement
  - Against Projections summary measurement
- See how performance profile information differs
- Test application
  - Charm++ 2D integration example

<table>
<thead>
<tr>
<th>TAU</th>
<th>Name</th>
<th>Projections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time percent</td>
<td>Ex. msec</td>
<td>In. msec</td>
</tr>
<tr>
<td>100.0</td>
<td>0.005</td>
<td>1043802</td>
</tr>
<tr>
<td>100.0</td>
<td>1947</td>
<td>1043802</td>
</tr>
<tr>
<td>88.2</td>
<td>919591</td>
<td>920479</td>
</tr>
<tr>
<td>7.7</td>
<td>80344</td>
<td>80344</td>
</tr>
<tr>
<td>2.1</td>
<td>21434</td>
<td>21877</td>
</tr>
</tbody>
</table>
NAMD Performance Study

- Demonstrate integrated analysis in real application
- NAMD parallel molecular dynamics code
  - Compute interactions between atoms
  - Group atoms in patches
  - Hybrid decomposition
    - distribute patches to processors
    - create compute objects to handle interactions between atoms of different patches
- Performance strategy
  - Distribute computational workload evenly
  - Keep communication to a minimum
  - Several factors: model complexity, size, balancing cost
NAMD ApoA1 Experiments

- Solvated lipid-protein complex in periodic cell
- Small 92K atom model
- Demonstrate performance of small computational grain
- Experiment on 256-processor Cray XT3 (BigBen)
NAMD STMV Experiments

- ApoA1 is a small problem
- Consider STMV virus benchmark
  - Ten times larger experiment
  - One million model
- Observe selected portion of the simulation
  - Remove startup
  - Look at 2000 timesteps
- Scaling studies
  - 256, 512, 1024, 2048, 4096
  - BigBen, Ranger, Intrepid
NAMD STMV Performance (Intrepid)

1024 processors

2048 processors

4096 processors

ICPP 2009

Integrated Performance Views in Charm++
NAMD STMV Scalability (Mean)

- **Intrepid**
  - Idle fraction increases due to lower utilization

- **Ranger**
  - Increase in Main reflects more time in communication
NAMD STMV – Comparative Profile Analysis

ICPP 2009

Integrated Performance Views in Charm++
NAMD STMV – Ranger versus Intrepid

**Metric:** LINUX_TIMERS
**Value:** Exclusive
**Units:** seconds

- **9.8**
  - namd-1024-lo-ranger – Mean: 2.798 (28.556%)
  - bgp-stmv-1024 – Mean: 2.798 (28.556%)
  - WorkDistrib::enqueueWorkA
    - (LocalWorkMsg* impl_msg)::129
- **7.275**
  - Main
    - 0.953 (13.1%)
- **6.856**
  - WorkDistrib::enqueueWorkB
    - (LocalWorkMsg* impl_msg)::130
    - 1.689 (24.643%)
- **5.918**
  - Idle
    - 1.21 (20.445%)
    - WorkDistrib::enqueueSelfA
      - (LocalWorkMsg* impl_msg)::127
      - 3.261
      - 1.338 (41.02%)
  - WorkDistrib::enqueueSelfB
    - (LocalWorkMsg* impl_msg)::128
    - 2.519
    - 0.921 (36.555%)
- **0.325**
  - CentralLB::LoadBalance(void)::178
  - 0.278 (85.471%)

*But Ranger has faster processors!*
NAMD STMV – Ranger versus Intrepid

We measured 10x the number of timesteps on Ranger than Intrepid!!!
TAU Profile Snapshots of NAMD

Snapshot Breakdown

Timeline (seconds)

Exclusive (microseconds)

- Idle
- WorkDistrib::enqueueWorkA(LocalWorkMsg* impl_msg)::133
- WorkDistrib::enqueueWorkB(LocalWorkMsg* impl_msg)::134
- Node::startup( void)::165
- Main
- WorkDistrib::enqueueSelfA(LocalWorkMsg* impl_msg)::131
- WorkDistrib::enqueueSelfB(LocalWorkMsg* impl_msg)::132
- CentralLB::LoadBalance( void)::182
- WorkDistrib::enqueueAngles(LocalWorkMsg* impl_msg)::126
- WorkDistrib::enqueueDihedrals(LocalWorkMsg* impl_msg)::127
- ComputePmeMngr::sendTrans( void)::262
- ProxyMngr::recvProxyData(ProxyDataMsg* impl_msg)::151
- WorkDistrib::enqueueBonds(LocalWorkMsg* impl_msg)::125
- ComputePmeMngr::ungridCalc( void)::270
- ComputePmeMngr::gridCalc2( void)::264
- ComputePmeMngr::initialize( CkQdMsg* impl_msg)::255
- CentralLB::ProcessAtSync( void)::177
- ComputePmeMngr::gridCalc1( void)::260
- WorkDistrib::enqueueCrossterms(LocalWorkMsg* impl_msg)::129
- ComputePmeMngr::gridCalc3( void)::267
- Other
NAMD Performance Data Mining

- Use TAU PerfExplorer data mining tool
  - Dimensionality reduction, clustering, correlation
  - Single profiles and across multiple experiments
NAMD STMV – Overhead Analysis

- Evaluate overhead as scale number of processors
- Overhead increases as granularity decreases
- Apply event selection and further overhead reduction
ChaNGa Performance Experiments

- Charm N-body GrAvity solver
  - Collisionless N-body simulations
- Interested in observing relationships between events
- Input TAU profiles to PerfExplorer

Correlation Results: $r = 0.8638213726236272$

128 processors
Conclusions

- TAU is now integrated with Charm++
  - Complements Projections performance capabilities
- Tested more advanced TAU features
  - User-level code events and communication events
  - Callpath and phase profiling
    - separate different aspects of the computation and runtime
- Charm++ has more sophisticated execution modes
  - Threading, process migration, dynamic adaption, …
  - Need to test TAU with these and make needed changes
- Apply to additional Charm++ applications
  - GPU-accelerated (see ParCo 2009 paper)
- Performance framework update and refinement
Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science

- NSF Software Development for Cyberinfrastructure (SDCI)
  - Grant No. NSFOCI-0722072