Center for Scalable Application Development Software

John Mellor-Crummey, Keith Cooper (Rice)
Peter Beckman, Ewing Lusk (ANL)
Jack Dongarra (UTK)
Bart Miller (Wisconsin)
Katherine Yelick (UCB/LBNL)
Goals

• Provide open source software systems, tools, and components that address a spectrum of needs
  – directly usable by application experts
  – provided to the CS community to enable development of other tools
• Engage directly with DOE application teams
• Target architectures of critical interest to DOE
  – Cray XT
  – Blue Gene/P
  – multicore processors in general
• Outreach
Outline

Community engagement

- Research and open source software development
  - system software and language runtime systems
  - communication for partitioned global address space languages
  - math libraries for multicore
  - performance tools
  - compilers
  - applications

- FY09 plans
Community Engagement

CScADS Summer Workshop Series

• Goals
  – identify challenges and open problems for leadership computing
  – brainstorm on promising approaches
  – foster collaborations between computer and application scientists
  – engage the broader community of enabling technology researchers

• Workshops to engage SciDAC and INCITE application teams
  – Leadership class machines, petascale applications, and performance
  – Scientific data analysis and visualization for petascale computing

• Workshops to foster development of enabling technologies
  – Autotuning for petascale systems
  – Performance tools for petascale computing
  – Libraries and algorithms for petascale applications

2009 Workshops at Granlibakken
Some Workshop Outcomes

• Leadership class machines, applications, and performance
  – introduced developers to changes in the leadership platforms
  – introduced developers to OpenMP, advanced MPI, parallel I/O, & tools
  – hands-on: ET researchers assisted developers with platforms and tools

• Performance tools
  – create an international community to share ideas and software
  – foster development of tool components rather than monolithic tools
  – “performance tool dating”
    • unstripping tool looking for symbol table info & an x86 instruction cracker

• Autotuning
  – bring together experts on architecture, libraries, and compilers
  – brainstorm on how to broaden reach of autotuning approaches
  – share experiences and ideas; explore opportunities for collaboration
  – identify common tools and benchmarks
Outline

• Community engagement
• Research and open source software development
  – system software and language runtime systems
  – communication for partitioned global address space languages
  – math libraries for multicore
  – performance tools
  – compilers
  – applications
• FY09 plans
CScADS ZeptoOS Research

• Exploring performance improvements for system software on leadership-class multicore platforms

• Focus
  – memory management
  – I/O forwarding and job control
  – communication software stack

• Benefits
  – foster software research on leadership computing platforms
  – extend the usage of leadership computing platforms
Memory Management on BG/P

- **General purpose OS loses memory performance**
  - worst case: standard Linux on ppc450 achieves only 25% of the theoretical memory bandwidth due to high cost of TLB misses

- **Solution**
  - introduced flat memory management to Linux
  - enables a compute task to access memory without TLB misses
I/O Forwarding and Job Control

- ZOID (ZeptoOS I/O Daemon) provides
  - complete job management
  - file I/O and IP forwarding for Zepto Compute Node Linux
- Extensible through plugins
  - custom I/O forwarding APIs
    - e.g. file system client, communication layer
- Open, full source code available
  - enables independent computer science research
- Optimized performance
  - multithreading to hide latency
  - reduced context switching
ZeptoOS I/O Daemon (ZOID)

Architecture

compute nodes
- libc
- FUSE
- ADIO
- IP fwd
- Job mgmt
- UNIX
- ZOIDFS
- ZOID daemon
- tree
- libzoid_cn

I/O nodes
- UNIX
- ZOIDFS
- IP fwd
- Job mgmt
- UNIX
- PVFS

Performance

Read from /dev/zero on ION

(raw link bandwidth is 6.8 Gb/s)
ZeptoOS Results

Blue Gene/P Compute Node OS and I/O layer operational

- Supports High Performance Computing (HPC) on BG/P
  - BG/P compute node software stack has been ported
  - MPICH is ready to use

- Supports High Throughput Computing (HTC) on BG/P
  - Falkon task execution framework has been ported
Outline

• Community engagement

• Research and open source software development
  – system software and language runtime systems
  – communication for partitioned global address space languages
  – math libraries for multicore
  – performance tools
  – compilers
  – applications

• FY09 plans
**CScADS PGAS Communication**

- **Planned SC08 release of GASNet and Berkeley UPC**
  - updated Portals conduit for Cray XT3/4/5 platforms with “firehose”
    - avoids pin/unpin costs with caching of registration table entries
  - new BG/P conduit based on low level DCMF layer
  - updated Infiniband conduit using new OpenIB/OpenFabrics verbs API
  - LAPI conduit for IBM Power uses RDMA
  - jointly funded by PModels and others

- **Implementation goals**
  - low latency for small to medium transfers
  - high bandwidth transfers
  - efficient collective communication

GASNet is being used in the upcoming Chapel release
GASNet vs MPI Latency on BG/P

- Recent work on DCMF Conduit for GASNet
- Jointly funded by PModels and CScADS

(lower is better)
GASNet outperforms MPI on small to medium messages, especially when multiple links are used.

(higher is better)
3D FFT Performance on BG/P

Strong scaling: good performance up to 16K cores

Upper bound is based on performance model of torus and bandwidth (higher is better)
UPC Collectives on Multicore

- Collective communication is important for many algorithms
- Technology trends encourage sharing on multicore chips
- Many factors affect performance
  - tree structure: balanced vs. binomial
  - collective routine
  - data size
- Exploring use of autotuning on collectives for multicore nodes
- Study different architectures
  - Intel Cloverton
  - Sun Niagara2

Autotuning collectives for Niagara2
(lower is better)
Outline

- Community engagement
- Research and open source software development
  - system software and language runtime systems
  - communication for partitioned global address space languages
  - math libraries for multicore
    - performance tools
    - compilers
    - applications
- FY09 plans
• Multicore is a disruptive technology for software
• Must rethink and rewrite applications, algorithms and software
  – as before with cluster computing and message passing
• Numerical libraries, e.g. LAPACK and ScLAPACK, need to change

• CScADS research
  – pOSKI: extend OSKI to autotune sparse matrix kernels for multicore
  – event-driven DAG scheduled computations
    • direct solvers (LU) on distributed memory using UPC
    • PLASMA software framework dense linear algebra for multicore
  – mixed precision
PLASMA: Parallel Linear Algebra s/w for Multicore

- **Objectives**
  - parallel performance
    - high utilization of each core
    - scaling to large numbers of cores
  - any memory model
    - shared memory: symmetric or non-symmetric
    - distributed memory
    - GPUs

- **Solution properties**
  - asychronicity: avoid fork-join (bulk synchronous design)
  - dynamic scheduling: out-of-order execution
  - fine granularity: independent block operations
  - locality of reference: store data using block data layout

A community effort led by Tennessee and Berkeley (similar to LAPACK/ScaLAPACK)
Computations as DAGs

Reorganize algorithms and software to work on tiles that are scheduled based on the directed acyclic graph of the computation.

PLASMA Methodology

Cholesky 4 x 4

QR 4 x 4
Cholesky using PLASMA

- PLASMA
- Arbitrary DAG
- Fully dynamic scheduling
PLASMA Provides Highest Performance
Leveraging Mixed Precision

• Why use single precision as part of the computation? Speed!
  – higher parallelism within vector units
    • 4 ops/cycle (usually) instead of 2 ops/cycle
  – reduced data motion
    • 32-bit vs. 64-bit data
  – higher locality in cache
    • more data items in cache

• Approach
  – compute a 32-bit result
  – calculate a correction for 32-bit results using 64-bit operations
  – update of 32-bit results with the correction using high precision
Iterative refinement for dense systems, $Ax = b$, can work this way:

\[
L \ U = lu(A) \\
x = L \backslash (U \backslash b) \\
r = b - Ax
\]

WHILE $||r||$ not small enough

\[
\begin{align*}
z &= L \backslash (U \backslash r) \\
x &= x + z \\
r &= b - Ax
\end{align*}
\]

END

Wilkinson, Moler, Stewart, & Higham provide error bound for SP floating point results when using DP floating point.

Using this, we can compute the result to 64-bit precision.
Results for Mixed Precision

Iterative Refinement for Dense $Ax = b$

<table>
<thead>
<tr>
<th>Architecture (BLAS)</th>
<th># proc</th>
<th>$n$</th>
<th>DP Solve /SP Solve</th>
<th>DP Solve /Iter Ref</th>
<th># iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Opteron (Goto – OpenMPI MX)</td>
<td>32</td>
<td>22627</td>
<td>1.85</td>
<td>1.79</td>
<td>6</td>
</tr>
<tr>
<td>AMD Opteron (Goto – OpenMPI MX)</td>
<td>64</td>
<td>32000</td>
<td>1.90</td>
<td>1.83</td>
<td>6</td>
</tr>
</tbody>
</table>
Outline

• Community engagement
• Research and open source software development
  – system software and language runtime systems
  – communication for partitioned global address space languages
  – math libraries for multicore
    ☞ performance tools
      – compilers
      – applications
• FY09 plans
Infrastructure for performance tools

- **Abstract interfaces**
  - provide portability and multiplatform support

- **Component-based approach**
  - enable sharing, rapid prototyping, co-development, development of best-of-breed algorithms and representations

- **Extensible data representations**
  - support layered library development

- **Open source**
  - support the above goals and allow broader adoption
Component-based Approach

Benefits

- Increases sharing and reuse
  - reduces redundant development
- Large research tool groups can focus on their priority missions without having to develop all parts of an end-to-end solution
- Small research groups (young investigators!) can explore focused research topics with a software code base comparable to that of the larger groups

Collaborations with internal (Rice, Wisconsin) and external (LLNL, Cray, Intel, Berkeley, Oregon, Jülich) groups on various APIs
workshop discussions are a critical part of the design process
The Deconstruction of DynInst

Realizing our push towards tool components

- **InstructionAPI**
  - abstract representation of instruction decode and address modes.

- **SymtabAPI**
  - abstraction of symbols, debug and dynamic linkage information
  - updating to support binary rewriting

- **StackwalkerAPI**
  - walk stacks: first or third party, standard vs. optimized frames, custom frames (from instrumentation or exceptions)
  - uses a variety of techniques from full symbols and libunwind to stripped binaries requiring control-flow analysis

- **ControlFlowAPI**
  - platform independent representation of CFG, associated query routines, and extensible data structures

CScADS funded components are underlined
libmonitor

An interface between OS and first-party tools

• Processes
  – parent: pre_fork, post_fork
  – child: init_process, fini_process

• Threads
  – parent: init_thread_support, thread_pre_create, thread_post_create
  – child: init_thread, fini_thread

• Signals
  – selectively catch signals before or instead of delivering to application

• Intercept functions to maintain control
  – e.g. dlopen, sigmask, pthread_sigmask, exit, signal

• Stack unwinding support
  – stack_bottom; identification of PC for bottommost frame
A Few Component Consumers

- **Rice**: using SymtabAPI and libmonitor in HPCToolkit
- **Krell Institute (Open|SpeedShop)**: Using SymtabAPI to get symbols for their offline collectors. Using libmonitor to manage first-party tools
- **UNC and LLNL**: using SymtabAPI and StackwalkerAPI for PnMPI project.
- **LLNL (STAT project)**: using SymtabAPI and StackwalkerAPI.
- **SiCortex**: porting SymtabAPI to Linux/MIPs; uses libmonitor underneath HPCToolkit
- **Cray**: started work using StackwalkerAPI and SymtabAPI for new APT (Abnormal Process Termination) system
- **Univ. of Oregon**: using binary rewriter as part of TAU instrumentation
- **Forschungszentrum Jülich**: using SymtabAPI for Scalasca
- **Berkeley (BitBlaze)**: APIs for binary processing (security tools)
Pinpointing Scalability Bottlenecks

![Graph showing efficiency vs. CPUs](image)

- **Ideal efficiency**
- **Actual efficiency**

Note: higher is better
Bottleneck Analysis Challenges

- **Parallel applications**
  - modern software uses layers of libraries
  - performance is often context dependent

- **Monitoring**
  - bottleneck nature: computation, data movement, synchronization?
  - size of petascale platforms demands acceptable data volume
  - low perturbation for use in production runs

Example climate code skeleton
HPCToolkit Performance Tools

- Measurement, analysis, attribution, and presentation of application performance
- Pinpoint and quantify performance bottlenecks
  — across scalable parallel systems
  — within multicore nodes
  — independent of programming model or cause

- Measurement, analysis, attribution, and presentation of application performance
- Pinpoint and quantify performance bottlenecks
  — across scalable parallel systems
  — within multicore nodes
  — independent of programming model or cause
Call Path Profiling

Measure and attribute costs in context

- Sample timer or hardware counter overflows
- Gather calling context using stack unwinding

Call path sample

- return address
- return address
- return address
- instruction pointer

Calling Context Tree (CCT)

Overhead proportional to sampling frequency... 
...not call frequency
Weak Scaling: 1K to 10K processors
S3D Multicore Losses at the Loop Level

Highlighted loop is 2.84x slower on 8 cores in a weak scaling study

Multicore Loss (Multicore time - single core time)
Moab: Integrated Static and Dynamic Info
Outline

• Community engagement
• Research and open source software development
  – system software and language runtime systems
  – communication for partitioned global address space languages
  – math libraries for multicore
  – performance tools
  ◁ compilers
    – applications
• FY09 plans
A source of inefficiency in large-scale applications is the “glue” that holds together code from different sources:
- library code, code cribbed from other applications
- often different languages with different programming models
  - e.g., call object-oriented code from inside imperative C program

Classic compilers cannot improve this kind of code:
- compiler never sees all the pieces
  - limits the scope of analysis and transformation
  - could build link-time optimizer, but would miss dynamically linked code
- good application for runtime re-optimization
  - monitor execution, notice inefficiency, rewrite code to avoid it
Runtime Re-optimization

Opportunities for improvement

• Classic answers
  – straighten hot paths to avoid jumps
  – fold constants from input data
  – reschedule long-latency ops to reflect actual behavior

• Opportunities in large-scale applications
  – improve procedure calls & chains of calls (libraries, CCA)
    • Runtime inlining and specialization of calls
  – fold constants from distribution of work and data
  – runtime selection of library components
    • e.g., choose communication routines based on actual distribution
Runtime Re-optimization

Current work

- Experimental
  - we are running a series of small-scale experiments to look at opportunities for improvement and magnitude of potential benefits
  - we have worked with both PIN (x86-specific tool) and LLVM

- Analytical
  - compile-time analysis to predict how much improvement we might find, given some set of runtime-knowable facts
  - compile-time analysis in support of the actual transformations and techniques to encode the results in the executable image
Outline

• Community engagement
• Research and open source software development
  – system software and language runtime systems
  – communication for partitioned global address space languages
  – math libraries for multicore
  – performance tools
  – compilers
  🏅 applications
• FY09 plans
Application Engagement: GTC

- GTC: simulates turbulent plasma in tokamak reactors
  - 3D particle-in-cell code; 1D decomposition along toroidal direction
    - charge: deposit charge from particles to grid points
    - solve: compute the electrostatic potential and field on grid points
    - push: compute the force on each particle from nearby grid points
- Extend performance analysis work of PERI Tiger team
- Used measurement and modeling tools developed at Rice with CScADS support to pinpoint performance losses
  - poor spatial locality due to vector of structures representation for ions
  - unrealized opportunities for temporal reuse between loops over ions
- Code optimization
  - manually transform to structure of vectors
  - manually apply fusion and blocking to improve temporal reuse
  - transmit improvements back to GTC code team
GTC: Node Performance Improvements

- Metrics normalized to measurements of original code
- Lower is better
GTC: Locality Degrades as Ions Swirl

- Locality is best when particles are sorted in cell order
  — potential computation uses cell data only
  — charge deposition and particle pushing involve interactions between particles and cells
- Initially particles are uniformly distributed in cell order
- Locality is best when particles are sorted in cell order
  — potential computation uses cell data only
  — charge deposition and particle pushing involve interactions between particles and cells
- **Over time, the particle distribution diverges from cell order**

Time step 20
GTC: Potential Improvement from Reordering

- Locality degrades gradually at run-time
- Assumptions:
  - periodic particle reordering restores locality and performance
  - performance degrades at similar rate after each sorting step

![Graph showing execution cost over time steps with and without sorting](image)
GTC: Compute Optimal Sorting Interval

- **Notations**
  - \( f(x) \) = time step cost function
  - \( C \) = cost of sorting
  - \( G(t) \) = gain from sorting every \( t \) time steps

- **Find** \( t \) that maximizes \( G(t) \) over \( N \) steps

\[
G(t) = \sum_{k=1}^{N-1} \left( \int_{k\cdot t}^{(k+1)\cdot t} f(x) \, dx - \int_{0}^{t} f(x) \, dx - C \right)
\]

\[
G(t) = \left[ \int_{0}^{N} f(x) \, dx - \frac{N}{t} \int_{0}^{t} f(x) \, dx \right] - \frac{N}{t} C + C
\]

- **Find** \( t \) that minimizes \( h(t) \)
  - \( h(t) = \frac{1}{t} \left( \int_{0}^{t} f(x) \, dx + C \right) \)
  - \( h(t) \) = average time step cost with sorting
Adaptive Particle Sorting Algorithm

Step 0: *Compute C*. Measure cost of sorting after program executed a fixed number of time steps. Start initialization at step 1.

Step 1: *Evaluate* $h(t)$. At each time step compute the value of the integral incrementally, evaluate $h(t)$, and update $h_{\text{min}}$ as needed.

Step 2: *Compute local optimum*. If last $h(t) > h_{\text{min}}$, local optimum $\tau_{\text{local}} = t - 1$

Step 3: *Compute global optimum*. Apply reduction with MAX operator across all processors. Global optimum $\tau = \max(\tau_{\text{local}})$

Step 4: *Provide confidence in the global optimum*. Continue initialization (steps 1 to 3) while $t < 2\tau$. 
Parallel Performance Results

- Combined optimizations reduce GTS execution time by
  - 37% on Itanium2 cluster
  - 21% on Cray XT and Cray XD1
GTC: Electron Sub-Cycle Loop

- GTC simulates ions + trapped electrons
- GTS transformations still apply to ion simulation
- Electrons move much faster than ions
  - execute multiple sub-cycle steps in each time step
  - electron sub-cycle loop dominates simulation cost
  - electron data reused in each sub-step; reuse distance large
- Locality improvements
  - each electron simulated for multiple sub-steps at a time
    - electron data reused with a short distance
    - electron migration accomplished with fewer, larger messages
    - better locality to grid data when electrons become disordered
GTC: Electron Sub-Cycle Loop Results

- Evaluate code with restructured electron sub-cycle loop
  - GTS transformations not applied
  - $128p = 32$ poloidal planes $\times$ 4 particle domains
  - $256p = 32$ poloidal planes $\times$ 8 particle domains
  - $4x = four$ times more grid points, # particles in tokamak unchanged
Application Engagement: S3D

- Direct numerical simulation (DNS) of turbulent combustion
  - state-of-the-art code developed at CRF/Sandia
    - PI: Jaqueline H. Chen, SNL
  - 2007/2008 INCITE awards at NCCS
  - pioneering application for 250TF system

- Extend performance analysis work of PERI Tiger team
  - use HPCToolkit to locate single-core performance bottlenecks
    - compiler inserted array copies
    - streaming calculations with low data reuse
    - loop nests with recurrences
  - identified opportunities for compiler-based improvement
  - enhanced LoopTool for addressing S3D’s needs
  - improved loop nests with LoopTool’s semi-automatic transforms
    - transformed code is now part of S3D’s source base
  - used HPCToolkit to assess multicore scaling issues
S3D: What Opportunities Exist?

5D loop nest:
2D explicit loops
3D F90 vector syntax

initialize

update

reuse

performance problem
data streams
in/out of memory

reuse

reuse

reuse
LoopTool: Loop Optimization of Fortran

Rice University’s tool for source-to-source transformation of Fortran (transformation subset shown)

Loop Unswitching

if P then
  S1
else
  S2
  S3

else
  S2
  S3

Controlled Loop Fusion

do k = 1,n
  do j = ...
    S1(j,k)
  enddo
enddo

Unroll and Jam

do k = 1,n-1,2
  do j = ...
    S1(j,k)
    S1(j,k+1)
  enddo
  enddo
Add LoopTool directives to source program
Optimization of S3D Diffusive Flux Loop

Transformation Log:
- scalarization (4 stmts)
- loop unswitching (2 conditions)
- fusion (loops within 4 outer nests)
- unroll-and-jam (2 loops)
- peeling excess iterations (4 nests)

2.94x faster than original (6.7% total savings)
Engagement: Other

• Enabling technologies engagement
  – APDEC: Chombo (structured AMR)
  – ITAPS: Moab/iMESH (meshing)

• Application engagement using HPCToolkit
  – UNEDF: MFDn (many Fermion dynamics - nuclear)
  – USQCD: Chroma (quantum chromodynamics)
  – Center for Turbulence Research: Hybrid (shock + turbulence)
  – NETL: MFiX (multiphase flow with interface exchanges)
  – Iowa State: CAM-EULAG (atmospheric modeling)
  – Gromacs (cellulosic ethanol)

• Working with Fortran 2008 J3 standards committee on parallelism via coarrays
Outline

• Community engagement
• Research and open source software development
  – system software and language runtime systems
  – communication for partitioned global address space languages
  – math libraries for multicore
  – performance tools
  – applications

☞ FY09 plans
FY09 Plans

• ANL
  – continue to replace components in BG/P s/w stack with open source

• Berkeley
  – release UPC and GASNet with improved support for BG/P, XT, and IB
  – optimize sparse linear algebra libraries for multicore (with UTK)

• Rice
  – performance tools
    • deploy HPCToolkit on the leadership computing platforms
    • devise support for working with data from a huge # of cores
  – compilers
    • continue work on dynamic optimization, ROSE, scripting languages
    • release a version of LoopTool for use by application teams

• Tennessee
  – explore dynamic and adaptive out-of-order execution patterns for
    linear algebra on multicore and heterogeneous nodes

• Wisconsin
  – continue development of InstructionAPI and ControlFlowAPI