High Productivity Computing Systems

Goals:
- Provide a new generation of economically viable high productivity computing systems for the national security and industrial user community (2007 – 2010)

Impact:
- **Performance** (efficiency): critical national security applications by a factor of 10X to 40X
- **Productivity** (time-to-solution)
- **Portability** (transparency): insulate research and operational application software from system
- **Robustness** (reliability): apply all known techniques to protect against outside attacks, hardware faults, & programming errors

Applications:
- Intelligence/surveillance, reconnaissance, cryptanalysis, weapons analysis, airborne contaminant modeling and biotechnology

Fill the Critical Technology and Capability Gap
Today (late 80’s HPC technology)…..to…..Future (Quantum/Bio Computing)
Motivation for Cascade

Why are HPC machines unproductive?

• Difficult to write parallel code (e.g.: MPI)
  • Major burden for computational scientists
• Lack of programming tools to understand program behavior
  • Conventional models break with scale and complexity
• Time spent trying to modify code to fit machine’s characteristics
  • For example, cluster machines have relatively low bandwidth between processors, and can’t directly access global memory…
  • As a result, programmers try hard to reduce communication, and have to bundle communication up in messages instead of simply accessing shared memory

*If the machine doesn’t match your code’s attributes, it makes the programming job much more difficult.*

*And code’s vary significantly in their requirements…*
The Cray Roadmap

Realizing Our Adaptive Supercomputing Vision

Phase I: Rainier Program
Multiple Processor Types with Integrated Infrastructure and User Environment

Phase II: Cascade Program
Adaptive Hybrid System

2006
- Cray X1E
- Cray XD1
- Cray X1E

2007
- Cray XT3
- Cray XT4
- Cray XMT

2008
- BlackWidow

2009
- Cray XT4 Upgrade

2010
- DARPA

Phase 0: Individually Architected Machines
Unique Products Serving Individual Market Needs
Diverse Application Needs

• To scale an application, it must have some form of parallelism.
• Many HPC apps have rich, SIMD-style *data-level parallelism*
  • They perform similar operations on arrays of data
  • Can significantly accelerate via *vectorization*
• Those that don’t generally have rich *thread-level parallelism*
  • Allow many independent threads performing independent work
  • This parallelism may be found at multiple levels in the code
  • Can significantly accelerate via *multithreading*
• Some parts of applications are not parallel at all
  • Need fast *serial scalar* execution speed
  • Slow serial performance will drag down performance (*Amdahl’s Law*)
• Applications also vary in their *memory and network bandwidth* needs
  • Low vs. high
  • Dense vs. sparse
Cascade Approach to Higher Productivity

• Ease the development of parallel codes
  • Legacy programming models: MPI, OpenMP
  • Improved variants: SHMEM, UPC and CAF
  • New alternative: Global View (Chapel)

• Provide programming tools to ease debugging and tuning at scale
  • Automatic performance analysis, relative debugging

• Design an **adaptive, configurable** machine that can match the attributes of a wide variety of applications:
  • Serial (single thread, latency-driven) performance
  • SIMD data level parallelism (vectorizable)
  • Fine grained MIMD parallelism (threadable)
  • Regular and sparse bandwidth of varying intensities

⇒ Increases performance
⇒ Significantly eases programming
⇒ Makes the machine much more broadly applicable
Cascade Processing Technology

- Partnering with AMD on HPCS Phase III Proposal
- Start with best of class microprocessor: AMD Opteron™
  - Industry standard x86/64 architecture
  - Integrated memory controller
    - very low memory latency (~50ns)
  - Open standard, high speed interface (HyperTransport)
  - Dual core today with strong roadmap
- Cray communications acceleration
  - Globally addressable memory
  - Scalable addressing, translation and synchronization
  - Unlimited concurrency for latency tolerance
  - Support for low latency, low overhead message passing too
- Cray computational acceleration
  - MVP (multi-threaded/vector processing) architecture
  - Exploits compiler-detected parallelism within a node
  - Extremely high single-processor performance
Cascade System Architecture

- Globally addressable memory with unified addressing architecture
- Configurable network, memory, processing and I/O
- Heterogeneous processing across node types, and within MVP nodes
- Can adapt at configuration time, compile time, run time
Integrated Multi-Architecture System

HPC Application Programs

Programming Models

Library Based
- pthreads
- OpenMP
- SHMEM
- MPI
- Other

Language Based
- MPI
- Other

COTS Enhanced Libraries

ALPS (Application Level Placement Scheduler) Infrastructure

COTS Enhanced Runtime
- CNL

Granite Runtime
- CNL

FPGA Runtime
- CNL

FPGA Libraries

COTS Tools
- Fortran, gcc, gdb
- PBSpro, LSF, Moab

Programmers
- Chapel (future)
- Cascade Compiler
- Debugger suite
- CrayPat/Apprentice2
- ALPS
- Cray PE Tools

COTS Enhanced Runtime
- CNL

Granite Runtime
- CNL

FPGA Runtime
- CNL

COTS Tools
- Linux

Global Shared Memory Segments

Memory

Opteron

Core

Opteron

Core

Opteron

Core

Opteron

Core

Opteron

Core

Opteron

Core

PCle2 I/O
Cascade from the User’s Viewpoint

• First Step
  • Run application across Opteron cores and profile to identify computational hotspots

• Second Step
  • Can we incorporate the hotspot into a high level parallel loop?
    • Are the inner loops vector?
      ▪ Direct the compiler to parallelize the outer parallel loop using OpenMP and/or compiler directives to generate tens of threads
      ▪ Compiler will vectorize inner most loop nests
    • Are the inner loops scalar
      ▪ Direct the compiler to use the outer loops to generate hundreds of threads.
    • No parallel outer loops
      ▪ Run in scalar on the Opteron

• Third Step
  • Can we use global addressing to improve scalability

• Fourth Step
  • Can we use heterogeneity to divide application into parallel tasks each of which may use scalar and/or parallel. Parallel scalar threading and parallel vector threading can co-exist in the same routine.
Cascade from the User’s Viewpoint

- Granite has a fully functional compiler that is built upon the best vectorizing/parallelizing compiler in the industry
  - Analyzing and generating vector threads is identical to techniques the multi-streaming, vectorizing compiler used for X1E and BW
  - Analyzing and generating scalar threads is identical to the techniques used by the multi-streaming compiler for the X1E and the threading compiler for the MTA and XTM

- The user can use Fortran, C, C++
  - Directives can be used to aid the compiler in generating the most efficient parallelization – just like always

- Or the user can employ Chapel to write a new application
So the user does not have to code in this:

twist : process (clk, reset_n) is
    variable v_y : std_logic_vector(31 downto 0);
begin  -- process twist
    if reset_n = '0' then               -- asynchronous reset (active low)
        s_mt_kk      <= (others => (others => '0'));
        s_mt_kk_m    <= (others => (others => '0'));
        s_mt_new     <= (others => (others => '0'));
        s_mt_wr_addr <= (others => (others => '0'));
    elsif clk'event and clk = '1' then  -- rising clock edge
        if (enable = '1') then
            s_mt_wr_addr <= s_mt_wr_addr(2 downto 0) & s_kk;
            s_mt_kk(3 downto 2)   <= s_dob(0)(63 downto 32) &
                s_dob(0)(31 downto 0);
            s_mt_kk_m(3 downto 2) <= s_dob(1)(63 downto 32) &
                s_dob(1)(31 downto 0);
            s_mt_kk(1 downto 0)   <= s_mt_kk(3 downto 2);
            s_mt_kk_m(1 downto 0) <= s_mt_kk_m(3 downto 2);
            v_y := s_mt_kk(word)(31) & s_mt_kk(word+1)(30 downto 0);
            if (v_y(0) = '0') then        -- even
                s_mt_new(word) <= s_mt_kk_m(word+1) xor ('0' & v_y(31 downto 1));
            else                          -- odd
                s_mt_new(word) <= s_mt_kk_m(word+1) xor ('0' & v_y(31 downto 1))
                xor c_magic;
            end if;
        end if;
    end if;
end loop;  -- word
end if;
But something they are used to:

```plaintext
!$OMP PARALLEL DO PRIVATE(iblock,this_block)
  
do iblock=1,nblocks_tropic
    this_block = get_block(blocks_tropic(iblock),iblock)
    
    !---- calculate (PC)r store in Z
    if (lprecond) then
      call preconditioner(Z,R,this_block,iblock)
    else   ! use diagonal preconditioner
      Z(:,:,iblock) = R(:,:,iblock)*A0R(:,:,iblock)
    endif

    !---- Compute intermediate result for dot product
    WORKN(:,:,1,iblock) = R(:,:,iblock)*Z(:,:,iblock)

    !---- update conjugate direction vector S
    S(:,:,iblock) =  Z(:,:,iblock)

    !---- compute Q = A * S
    call btrop_operator(Q,S,this_block,iblock)

    !---- compute intermediate result for dot product
    WORKN(:,:,2,iblock) = S(:,:,iblock)*Q(:,:,iblock)
  end do

!$OMP END PARALLEL DO
```
Chapel
A new parallel language developed by Cray for HPCS

• Themes:
  • raise level of abstraction, generality compared to SPMD approaches
  • support prototyping of parallel codes + evolution to production-grade
  • narrow gap between parallel and mainstream languages

• Chapel’s Productivity Goals:
  • vastly improve **programmability** over current languages/models
  • support **performance** that matches or beats MPI
  • improve **portability** over current languages/models (similar to MPI)
  • improve code **robustness** via improved abstractions and semantics

• Status:
  • draft language specification available
  • portable prototype implementation underway
  • performing application kernel studies to exercise Chapel
  • working with HPLS team to evaluate Chapel
  • initial release made to HPLS team in December 2006
Chapel Code Size Comparison

<table>
<thead>
<tr>
<th></th>
<th>STREAM Triad</th>
<th>Random Access</th>
<th>FFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Chapel</td>
<td>433</td>
<td>86</td>
<td>124</td>
</tr>
<tr>
<td>Reference</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Chapel</td>
<td>1668</td>
<td>1406</td>
<td>156</td>
</tr>
</tbody>
</table>

Legend:
- **Framework**: Blue
- **Computation**: Dark blue
- **Prob. Size (common)**: Light blue
- **Results and output**: Green
- **Verification**: Turquoise
- **Initialization**: Red
- **Kernel declarations**: Dark green
- **Kernel computation**: Yellow
So what are we asking the user to do?

- Continue to think about using 30,000-40,000 MPI tasks
  - At least some of the users are doing this today
- Continue to investigate cache optimization to avoid the memory wall
  - Blocking
  - Alignment
- Think about extending the size of your problem on the node – a second level of parallelism.
- Think about extending your MPI to utilize the using global addressability.
- Think about heterogeneity
Cascade from the User’s Viewpoint

- Craypat will be extended to address the multi-processing modes of the Granite processor
  - Overhead of generating Granite parallel region versus benefit of running on the Granite processor.
  - Comparative performance of running in various modes
    - Scalar on Opteron
    - Fully threaded
    - Vector

- Comparative debugger
  - Shows differences in running in various modes
    - Scalar on Opteron
    - Fully threaded
    - Vector
  - Quickly identify if loop should not have been parallelized/vectorized
    - Perhaps private variables should be shared, etc.
Cray Peta-Scale Software Themes

- Cray Software Focus is on Productivity
  - Delivering performance – absolute & time to completion
  - Ease of use
  - Compatibility with previous generations and HPC community

- Building from Baker
  - Extending the evolution of software from XT to Baker to Peta-Scale
  - Utilizing technology from other Cray products

- Heterogeneity Management
  - Transparently managed for the user - OR -
  - User can access directly the specific components that are required

- Extending with Granite
  - New, finer grained, techniques for vector and threaded parts of applications
  - More productive user interaction – compiler provides more assistance
Application Needs Drive Software Development

Key Application Requirements –
- Support for existing and future programming models
  - MPI and Fortran are dominant programming models currently.
  - Future programming models such as CAF, UPC, SHMEM, ARMC1 require Globally Addressable Memory

Application libraries
Walkthroughs of many applications show which math, IO, and communication libraries we need to optimize.

<table>
<thead>
<tr>
<th>Programming Model</th>
<th>Scaling Aids</th>
<th>Performance Libraries</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran 77</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fortran 90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fortran 95</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C/C++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pthreads</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAF/UPC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHMEM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MP12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assembler</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLAS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFTW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAPACK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PBLAS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PetSC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalapack</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cray Scilib</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HDF5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI-IO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>netCDF</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Library</th>
<th>Now</th>
<th>Rewrite</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>MILC</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>NAMD</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>WRF</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>HOMME</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CICE</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>RMG</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>PARSEC</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Espresso</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>LSMS</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SPECFEM3D</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Chimera</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>GTC</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>GAMESS</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Now and Rewrite columns indicate which libraries are being used in the current and future versions of the applications.
Programming Environment (outstanding features)

Provides for all of these common HPC programming models.
- MPI, Cray SHMEM, UPC, CAF, OpenMP, pthreads, ARMCI, and Global Arrays

Provides math libraries that support **ALL** the programming models
- Vector and matrix algebra, sparse matrix, and FFT

Performance Analysis System
- CrayPAT – instrumentation and data collection
- Cray Apprentice² – interactive analysis of performance data
- Automatic Performance Analysis – automated identification of bottlenecks

Scalable Data Centric Debugger
- Familiar interfaces such as TotalView and gdb
- Provides a Scalable Debug Manager
- Comparative debugger – simultaneously compares a properly working version of an application to a version that is not working
Peta-scale Debugging Challenges

• Peta-scale debugging is mind boggling
• User’s brain and debugger have trouble scaling
• Future architecture adds complexity
  • Hundreds of thousands of threads make direct control untenable
  • Decoupled instruction streams
  • Multiple ISAs
  • Imprecise traps
Debug Suite

- Support for industry accepted Opteron debugging tools which are popular for initial small scale debugging of an application
  - gdb
  - TotalView
- Innovative techniques for usability and scalability
  - Scalable debugger manager
    - Tailored to MPI application debug
  - Comparative debugging
    - Data-centric, comparative debugging
  - Dual code debugging
    - Run optimized until area of concern, then instrumented
Scalable Debug Manager

- Component of Parallel Tools Project (PTP)
- Developed at LANL for debugging MPI applications at scale
- Deployed as an MPI “co-application” with ALPS
- Uses gdb as the debugger demon on each node
- SDM provides control and debug output filtering for the thousands of gdb processes
Comparative Debugging

- Focus more on data – not as much on program flow
- “Control” and “experiment” program run in large lock steps
- Program data is compared at each step
- Effective for comparing different implementations
- Narrow down problem without massive thread study
Comparative Debugging

• Scenarios
  • Porting between architectures
  • Serial converted to parallel
  • One optimization level versus another
  • Small scaling versus large scaling
  • One programming language converted to another
  • COTS only (a la cluster) versus MVP
  • MVP threaded versus MVP vector

• Requirements
  • Simultaneous run of two applications
  • Ability to compare data from the different applications
  • Ability to assert the match of data at given points in execution
Data Comparison and Reports

- Data comparison
  - Tolerance control – nobody expect it to be perfect
  - Array subsets – correlate serial to parallel bits
  - Array index permutation – loops rearranged
  - Automated asserts – let it run until a problem is found
  - Forcing correct values – continue on with correct data

- Discrepancy reporting
  - Print – for really simple stuff
  - Rectangular bitmap – black or white pixels
  - Visualization packages
Comparative Debugging Example

- MM5 Weather model
- Serial versus parallel
- Difference >0.1%
- Narrowed to missing term in equation
- … term found and added in

Courtesy of David Abramson from GuardSoft
Performance Analysis Tools for Petascale

Cray’s Apprentice² tool is already a world leader in large scale performance analysis. Known for being easy to use.

- **Call Graph Profile**
- **Function Overview**
- **Pair-wise Communication View**
- **Load balance views**
- **Time Line & I/O Views**
- **Communication & I/O Activity View**
Optimization with Traditional Performance Tools

Huge amount of Measurement data → Little Simple analysis → Even more Derived analysis data
Cray Performance Analysis 2011

- **Performance measurement tools** provide the data needed for users to tune and optimize applications
  - Users still need to understand details of system software and architecture to correlate observations from performance measurement with the system in use to understand the performance behavior of the application

- **Automatic performance analysis tools** using expert systems guided by performance models to analyze the performance data in order to identify and expose performance anomalies:
  - Load imbalance
  - Communication / synchronization problems
  - Saturation of network links
  - Automatic performance analysis
    - Post-mortem
    - During runtime
    (to manage data scalability)
Cascade Summary

• Performance
  • Configurable, very high bandwidth memory and interconnect
  • Globally addressable memory with fine-grain synchronization
  • Heterogeneous processing to match application (serial, TLP, DLP)

• Programmability
  • Shmem(), UPC, CAF, and Chapel high-level parallel language
  • Automatic performance analysis and scalable debugging tools
  • Globally addressable memory with fine-grain synchronization
  • Heterogeneous processing supports wide range of programming idioms

• Portability
  • Linux-based OS supports standard POSIX API & Linux services
  • Support for mixed legacy languages and programming models
  • Chapel provides an architecturally-neutral path forward for code

• Robustness
  • Central administration and management
  • Hardware Supervisory System
  • Transactional system state, virtualized failover
Q&A