The Cray Compiling Environment: Bridging the User to High Performance

Luiz DeRose
Programming Environment Director
Cray Inc.

CSCS
July 13-15, 2009

Cray Programming Environment Focus

- It is the role of the Programming Environment to **close the gap** between observed performance and peak performance
  - Help users achieve ***highest possible performance*** from the hardware
- The Cray Programming Environment addresses issues of scale and complexity of high end HPC systems.
  - The Cray Programming Environment helps users to be more **productive**
  - It is the place at which the **complexity** of a system **is hidden** from the user
- User **productivity** is **enhanced** with
  - Increased ***automation***
  - ***Ease of use***
  - Extended ***functionality*** and improved ***reliability***
  - Close ***interaction with users*** for feedback targeting functionality enhancements

The Cray Compiling Environment

- Ability and motivation to provide high-quality support for custom Cray network hardware
- Cray technology focused on scientific applications
  - Takes advantage of Cray's extensive knowledge of **automatic vectorization**
  - Takes advantage of Cray's extensive knowledge of **automatic shared memory parallelization**
  - Supplements, rather than replaces, the available compiler choices
- Standard conforming languages and programming models
  - Fortran 2003 Compliant – Working on Fortran 2008
  - OpenMP
    - Fully integrated with other compiler optimizations, including automatic shared memory parallelization
  - UPC & CoArray Fortran
    - **Fully optimized** and integrated into the compiler
    - No preprocessor involved
    - Target the network appropriately:
      - GASNet with Portals
      - DMAPP with Gemini

CCE Main Features

- Fortran 2003 standard compliant
  - Selected F2008 features
- C99 and C++ support
- UPC 1.2 and Fortran 2008 CAF functional support
- OpenMP 3.0 support (with limitations)
- Vectorization
- Automatic cache blocking
- Automatic multithreading
- Prefetching, Interchange, Fusion
- Cray performance tools and debugger support

OpenMP

- CCE 7.1 supports the OpenMP 3.0 specification, with minor limitations:
  - C++ random access iterator loops marked for work sharing may not get work shared
  - Task switching is not implemented
  - Limitations to be removed in future releases
- OpenMP and automatic multithreading are fully integrated with the compiler
  - Share the same runtime and resource pool
  - Aggressive loop restructuring and scalar optimization is done in the presence of OpenMP
  - Consistent interface for managing OpenMP and automatic multithreading
- Nested parallelism and OpenMP tasks can be used to take advantage of increasing numbers of cores within a node

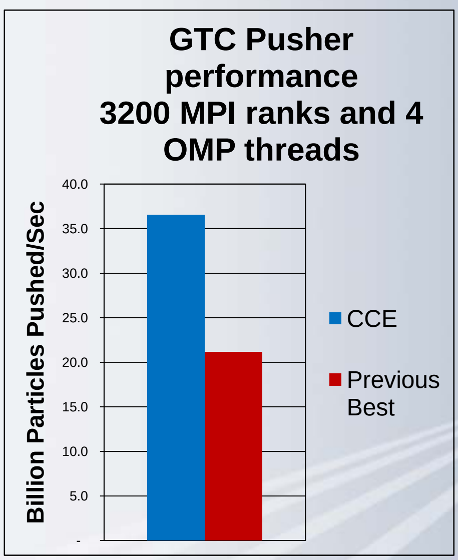
CCE case studies

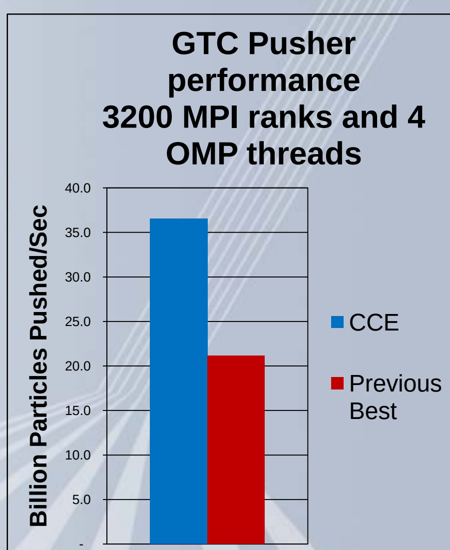
- Cray benchmark team studies
- Examples of multiple levels of parallelism
  - MPI
  - OpenMP (including nesting and tasks)
  - Vectorization

Case Study: The Gyrokinetic Toroidal Code (GTC)

- Plasma Fusion Simulation
- 3D Particle-in-cell code (PIC) in toroidal geometry
- Developed by Prof. Zhihong Lin (now at UC Irvine)
- Code has several different characteristics
  - Stride-1 copies
  - Strided memory operations
  - Computationally intensive
  - Gather/Scatter
  - Sorting and Packing
- Main routine is known as the “pusher”

Case Study: GTC

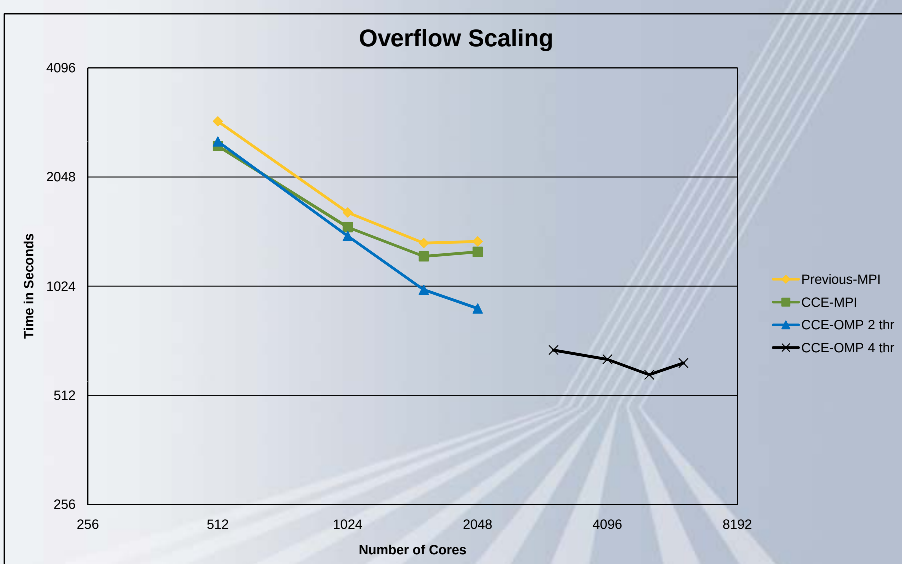




Case Study: Overflow

- Overflow is a NASA developed Navier-Stokes flow solver for unstructured grids
- Subroutines consist of two or three simply-nested loops
- Inner loops tend to be highly vectorized and have 20-50 Fortran statements
- MPI is used for parallel processing
  - Solver automatically splits grid blocks for load balancing
  - Scaling is limited due to load balancing at > 1024
- Code is threaded at a high-level via OpenMP

Overflow Scaling using only MPI vs MPI & OMP



Case Study: PARQUET

- Materials Science code
- Scales to 1000s of MPI ranks before it runs out of parallelism
- Want to use shared memory parallelism across entire node

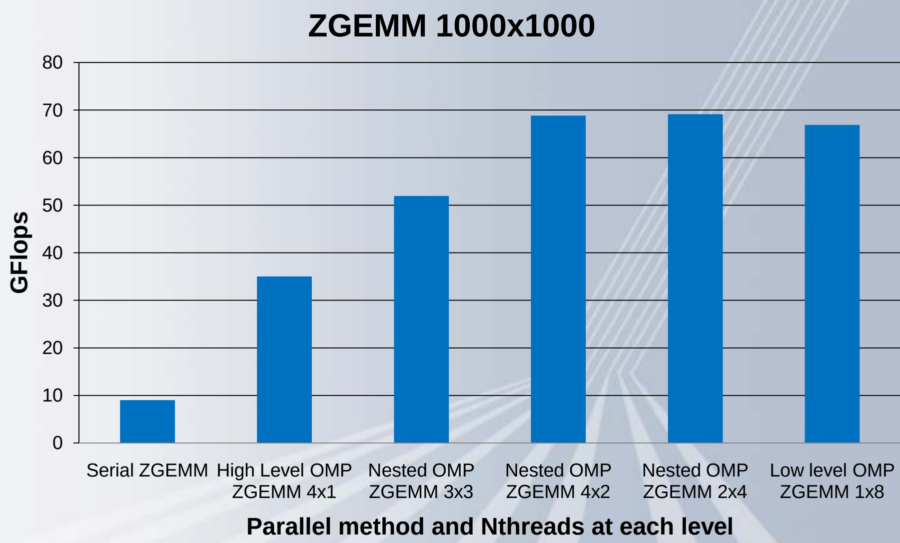
- Main kernel consists of 4 independent zgemms
- Want to use multi-level OMP to scale across the node

Case Study: PARQUET

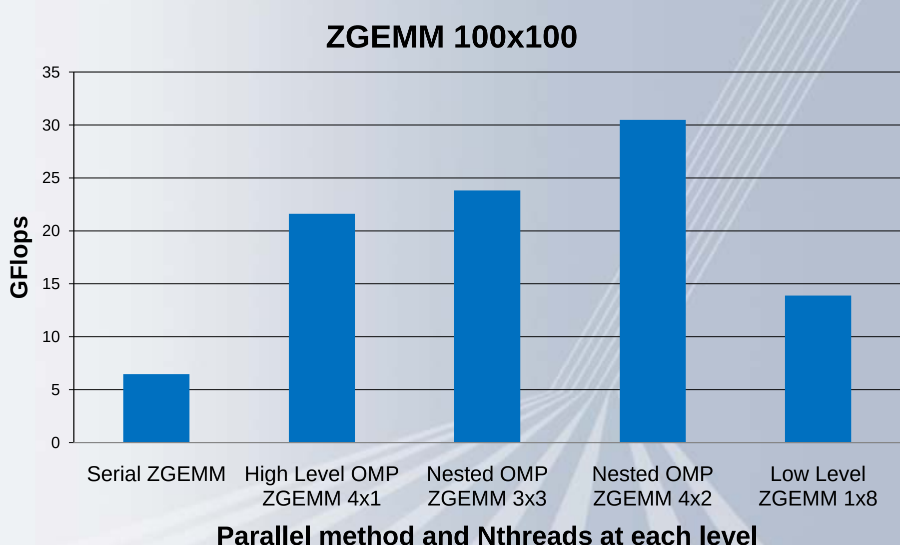
```
!$omp parallel do …
do i=1,4
    call complex_matmul(…)
enddo

Subroutine complex_matmul(…)
!$omp      parallel do private(j,jend,jsize)! num_threads(p2)
    do j=1,n,nb
      jend = min(n, j+nb-1)
      jsize = jend - j + 1
      call zgemm( transA,transB, m,jsize,k,                    &
           alpha,A,ldA,B(j,1),ldb, beta,C(1,j),ldC)
    enddo
```

Case Study: PARQUET



Case Study: PARQUET



CrayPat OpenMP Performance Metrics

- Per-thread timings
- Overhead incurred at enter/exit of
  - Parallel regions
  - Worksharing constructs within parallel regions
- Load balance information across threads
- Separate metrics for OpenMP runtime and OpenMP API calls
- Default view (no options needed to pat_report)
  - Focus on where program is spending its time
  - Shows imbalance across all threads
  - Assumes all requested resources should be used
  - Highlights non-uniform imbalance across threads

UPC Overview

- Unified Parallel C (UPC)
  - Extensions to the standard C language
- Programmer can statically or dynamically allocate data in shared storage that is accessible from all PEs
- Different parts of an allocation can have affinity to different Pes
  - Referred to as Partitioned Global Address Space (PGAS)
- All PEs execute the whole program
  - UPC PEs are called *threads*
    - these are not pthreads
- Conditional code used to limit certain program regions to smaller groups of PEs

UPC 1.2 extensions

- Shared type qualifier can be applied to scalars, arrays, and pointers
- Array elements distributed across PEs in a "round-robin" fashion
- Optional block size value can change the distribution granularity from one element to many

UPC for Cray XT systems

- Command line option –hupc
- UPC is fully integrated into the compiler
  - No preprocessor involved
  - Compiler optimizer is UPC-aware
- Runtime provided by Cray libpgas
  - Libpgas supports both UPC and Fortran with Coarrays (CAF)
- Remote accesses turn into networking layer library calls
  - GASNet
    - Berkeley library for supporting PGAS programming models
    - Used for Cray XT SeaStar systems (Portals conduit)
  - DMAPP
    - Cray Distributed Memory Application communication library
    - Optimized for Cray XT Gemini systems

UPC compiler implementation

- Recognize global memory (PGAS) accesses
- Linearize PGAS addresses such that the PE number is in the upper bits
- Mask off the PE number for accesses that are really local
- Turn UPC operations into libpgas calls
  - Use direct calls to GASNet or DMAPP when possible
- Optimize stride-1 data accesses
  - Compiler pattern matching feature
  - Recognize stride-1 data access patterns and substitute calls to optimized library routines

UPC Gemini enhancements

- Compiler takes advantage of new functionality available with DMAPP on Gemini
  - Remote AMO functions
  - PE-strided remote memory access functions
  - Remote memory scatter and gather functions

Summary

- The Cray Compiling Environment is focused on enhancing user productivity
  - Deliver high performance automatically
    - Aggressive optimization with default command line options
    - Support for custom Cray network hadware
  - Support standard programming models
    - Fully integrated OpenMP
    - Fully integrated UPC and Fortran Coarrays

Thank You!

Q&A

CSCS
July 13-15, 2009