Outline

- Authors:
  - Myungho Lee, Larry Meadows, Darryl Gove, Dominic Paulraj, Sanjay Goil, CPE
  - Brian Whitney, SAE
  - Nawal Copty, Yonghong Song, SCT
- OpenMP Support in Sun's Compilers
- Optimizing OpenMP Programs
- SPEC OMPL Benchmarks
- Benchmarking Systems
- Performance Results
- Summary and Future Directions
S1S8 Compiler Collection

- Sun's latest compilers and tools, released May 2003
- Includes:
  - Support for OpenMP Spec 2.0 in C, C++, and Fortran
  - Support for multiple user threads with OpenMP
  - Support for Thread Local Storage (TLS) in C, C++
  - Advanced compiler optimizations
  - Improved performance and scalability
C, C++, Fortran source with OpenMP directives

Front-End

- Recognize OpenMP directives

Optimizer

- Transform OpenMP constructs
- Insert calls to libmtsk
- Parallelize loops automatically

Code Generator

- Generate SPARC machine code

Code to run using multiple threads

Libmtsk
Compiler Transformations for OpenMP

- Analyze properties of variables in parallel region (SHARED, PRIVATE, FIRSTPRIVATE, REDUCTION, etc.)
- Put body of parallel region in a separate routine (outlined routine)
- Replace original parallel region by a call to an OpenMP runtime library routine (__mt_MasterFunction__, __mt_WorkSharing__) 
- Pass address of outlined routine as argument to runtime library routine
OpenMP Runtime Support Library (libmtsk)

- Implemented on top of Solaris threads library (libthread)
- Responsible for:
  - thread creation and management
  - assignment of work to threads
  - thread synchronization
- Threadsafe (supports multiple user threads)
OpenMP-Specific Optimizations

• Minimize synchronization costs
  fast locking mechanisms, tree-based barriers, etc.

• Minimize cost of accessing shared data
  make read-only shared data FIRSTPRIVATE, etc.

• Minimize threading overheads
  reuse threads, fast-path for sequential execution, etc.

• Minimize false sharing overheads
  pad lock variables, threadprivate variables, etc.
Other Compiler Optimizations

- Scalar optimizations
- Loop and cache optimizations
- Prefetching
- Optimizations for instruction-level parallelism
- Interprocedural analysis
- Automatic parallelization
SPEC OMPL Benchmarks (OMPL2001)

- Released in June 2002 by SPEC HPG
- 9 programs in C and Fortran, parallelized using OpenMP
- Representative of HPC and scientific applications
- Targeted to large scale servers
- Each benchmark requires up to 6.4GB memory when run on a single CPU machine
# SPEC OMPL Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Application</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>wupwise</td>
<td>Quantum chromodynamics</td>
<td>Fortran</td>
</tr>
<tr>
<td>swim</td>
<td>Shallow water modeling</td>
<td>Fortran</td>
</tr>
<tr>
<td>mgrid</td>
<td>Multi-grid solver</td>
<td>Fortran</td>
</tr>
<tr>
<td>applu</td>
<td>Partial diff equations</td>
<td>Fortran</td>
</tr>
<tr>
<td>equake</td>
<td>Earthquake modeling</td>
<td>C</td>
</tr>
<tr>
<td>apsi</td>
<td>Air pollutants</td>
<td>Fortran</td>
</tr>
<tr>
<td>gafort</td>
<td>Genetic algorithm</td>
<td>Fortran</td>
</tr>
<tr>
<td>fma3d</td>
<td>Crash simulation</td>
<td>Fortran</td>
</tr>
<tr>
<td>art</td>
<td>Neural network simulation</td>
<td>C</td>
</tr>
</tbody>
</table>
Benchmarking Systems
SF6800 & SF15K

Basic unit:
UniBoard (4 UltraSPARC III Cu processors with memory)

System Characteristics

<table>
<thead>
<tr>
<th></th>
<th>SF6800</th>
<th>SF15K</th>
</tr>
</thead>
<tbody>
<tr>
<td>No of CPUs</td>
<td>24 (6 UniBoards)</td>
<td>72 (18 UniBoards)</td>
</tr>
<tr>
<td>Clock Freq</td>
<td>1200Mhz</td>
<td>1200Mhz</td>
</tr>
<tr>
<td>Cache Coherence Protocol</td>
<td>Snoopy</td>
<td>Snoopy within UniBoard Directory between UniBoards</td>
</tr>
<tr>
<td>Aggregate System B/W</td>
<td>9.6 GB/sec</td>
<td>173 GB/sec</td>
</tr>
<tr>
<td>Latency</td>
<td>220 ns (local)</td>
<td>240 ns (local)</td>
</tr>
<tr>
<td></td>
<td>272 ns (remote)</td>
<td>455 ns (remote)</td>
</tr>
</tbody>
</table>
**Solaris 9U2: New Features**

**Memory Placement Optimization (MPO)**
- Latency between different UniBoards > latency within same UniBoard
- First-touch memory placement:
  - Default placement
  - Can significantly improve performance when data accesses are mostly to local UniBoard

**Multiple Page Size Support (MPSS)**
- Can specify different page sizes for different regions of virtual memory: 8K, 64K, 512K, 4M  (default = 8K)
- Large page size can significantly reduce no. of TLB entries needed => can reduce TLB misses for applications that use large memory
Performance Tuning

Identify performance bottlenecks

Tune using:

• **Compiler Features**
  - Compiler and runtime library (libmtsk) support
  - Advanced compiler optimizations

• **OS Features**
  - MPO: First-touch works the best
  - MPSS: 4M for base, 4M/512K/64K for peak

• **Other**
  - Processor binding: MT_BIND_PROCESSOR
  - Alternative src: art
Scalability of SPEC OMPL (Base)

SF6800

SF15K

No of Processors

Speedup over P=8

- 311.wupwise_l
- 313.swim_l
- 315.mgrid_l
- 317.applu_l
- 321.equake_l
- 325.apsi_l
- 327.gafort_l
- 329.fma3d_l
- 331.art_l
Scalability of OMPL Results
Comparing $P = 8$ vs. $P = 16$ Base Performance (SF6800) of swim, mgrid, applu

- Swim & mgrid: Scalability low due to memory B/W limit
- Applu: Super-linear speedup as more data gets cached with increased number of processors
Scalability of OMPL Results

Comparing P = 32 vs. P = 64 Base Performance (SF15K) of swim, mgrid, applu, equake

- Swim & mgrid: Scalability improves with higher memory B/W
- Applu: Speedup saturated on larger no. of processors as benefit of data caching diminishes
- Equake: Base performance on P = 64 is low because 4M pages are shared among different processors
Summary

- SPEC OMPL exhibits good performance and scalability on SF6800 and SF15K
- SPEC OMPL performance enhanced by:
  - Advanced optimizations in S1S8 compiler
  - MPO, MPSS in Solaris 9U2
  - Processor binding
Future Directions

- Enhanced static and runtime error checking of OpenMP programs
- Autoscopying of OpenMP programs
- Support for nested parallelism
- Support for multi-threaded architectures
- Enhanced compiler optimizations
- Improved performance