Shared Memory Parallel Programming

Shared Memory Systems
Introduction to OpenMP
Parallel Architectures

- Distributed Memory Machine (DMP)
- Shared Memory Machine (SMP)
Sun E450 SMP

4 UltraSPARC processors (400-480MHz)

6KB D-cache, 16kB I-cache per processor

4GB main memory in 4 banks

4-8MB E-cache per processor
Sun Fire 6800 Server

Up to 24 UltraSPARC III processors in SMP architecture
192GB memory
9.6GB bandwidth
4-way associative on-chip 64KB data and 32KB instruction cache
8MB E-cache
Shared Memory Architectures

Built during the 1980’s, now popular again

Multiple processors that access *same main memory*

- local cache, connected to main memory via bus
- When a memory location is updated, any copies in the local cache of other processors must be invalidated
- Bus is bottleneck; traditional wisdom is that only small systems are possible

But plenty of R&D helped overcome this
Shared Memory

- All processes/threads access same shared memory data space
More Realistic View of Shared Memory Architecture

For small numbers of processors

Diagram showing:
- Shared memory
- Multiple caches (cache1, cache2, cache3, cacheN)
- Multiple processors (proc1, proc2, proc3, procN)
Distributed Shared Memory Systems

- Architecture designed to overcome size limits of shared memory systems
- Distributed memory system but memory is globally addressed
- Hardware supports cache coherency
- So looks like shared memory system

Examples: HP Exemplar, SGI Origin, Altix
Distributed Shared Memory

For larger numbers of processors

mem1  mem2  mem3  memN

cache1  cache2  cache3  cache4

proc1  proc2  proc3  procN
From Clusters of SMPs to Distributed Global Address Space

Global addressing and many threads
Future HPC Hardware

• Feature set?
  – Massive parallelism
  – Global address space
  – High bandwidth to memory
  – Thread-based components
  – Efficient synchronization, context switching

• Execution model?
  – Multi-level parallelism, multithreading
Multicores are coming!

- **AMD Opteron**: Dual Core
- **Intel Montecito**: 1.7 Billion transistors, Dual Core IA/64
- **Intel Tanglewood**: Dual Core IA/64
- **Intel Pentium D (Smithfield)**
- **Intel Dempsey**: Dual Core Xeon
- **Intel Pentium Extreme**: 3.2GHz Dual Core
- **Intel Yonah**: Dual Core Mobile
- **AMD Opteron**: Dual Core
- **Intel Tejas & Jayhawk Unicore (4GHz P4)**
- **Cancelled**
- **AMD Athlon 64**: Dual Core
- **Intel Power4 & 5**: Dual Cores Since 2001
- **IBM Power 4 and 5**: Dual Cores Since 2001
- **IBM Power 6**: Dual Core
- **MIT Raw**: 16 Cores Since 2002
- **IBM Cell**: Scalable Multicore
- **Sun Olympus and Niagara**: 8 Processor Cores

Timeline:
- 2H 2004
- 1H 2005
- 2H 2005
- 1H 2006
- 2H 2006
Multicore Complexity

- Resources (L2 cache, memory bandwidth): shared or separated
- Each core: single thread or multithreaded, complex or simplified
- Among cores: symmetric or asymmetric (Heterogeneous)
Challenges Posed By New Architectures

- Hierarchical and hybrid parallelism
  - Clusters, SMPs, CMP (multicores), SMT (simultaneous multithreading), ...
- Diversity in kind and extent of resource sharing, potential for thread contention
  - ALU/FP units, cache, MCU, data-path, memory bandwidth
- Homogeneous or heterogeneous
- Deeper memory hierarchy
- Size and scale

Compiler cannot extract parallelism for multithreading
Challenges Posed By New Architectures

- Hierarchical and hybrid parallelism
  - Clusters, SMPs, CMP (multicores), SMT (simultaneous multithreading), …

- Diversity in kind and extent of resource sharing, potential for thread contention
  - ALU/FP units, cache, MCU, data-path, memory bandwidth

- Homogeneous or heterogeneous
- Deeper memory hierarchy
- Size and scale

Compiler cannot extract parallelism for multithreading
What is OpenMP?

• New standard for shared memory parallel programming for scientific applications
• Focus is on scientific applications
• Rapidly gaining acceptance among vendors and application writers
• For Fortran, C and C++
• http://www.openmp.org
The OpenMP ARB

• OpenMP is maintained by the OpenMP Architecture Review Board (the ARB).

• The ARB:
  • Interprets OpenMP
  • Writes new specifications - keeps OpenMP relevant
  • Works to increase the impact of OpenMP

• Members are organizations - not individuals
  – Current members
    • Permanent: Fujitsu, HP, IBM, Intel, NEC, PGI, SGI, Sun
    • Auxiliary: ASCI, cOMPunity, EPCC, KSL, NASA, RWTH Aachen
cOMPunity

- Researchers also participate in work of the ARB
- cOMPunity set up to organize this
  - Non-profit organization dedicated to furthering OpenMP
  - Participates in on-going work of the ARB
  - Also organizes series of international workshops

- [www.compunity.org](http://www.compunity.org)
OpenMP Release History

- **1997**
  - OpenMP Fortran 1.0

- **1998**
  - OpenMP C/C++ 1.0

- **1999**
  - OpenMP Fortran 1.1

- **2000**
  - OpenMP Fortran 2.0

- **2002**
  - OpenMP C/C++ 2.0

- **2005**
  - A single specification for Fortran, C and C++
OpenMP Overview

• A set of compiler directives inserted in the source program
• Also some library functions
• Ideally, compiler directives do not affect sequential code
  – pragmas in C / C++
  – (specially written) comments in Fortran code
Idea of OpenMP

Sequential code:

statement1;
statement2;
statement3;

Assume we want to execute statement 2 in parallel, and statement 1 and 3 sequentially
Idea of OpenMP

statement 1;
#pragma <specific OpenMP directive>
statement 2;
statement 3;

Statement 2 (may be) executed in parallel
Statement 1 and 3 are executed sequentially
Idea of OpenMP

statement 1
!$OMP <specific OpenMP directive>
statement2
!$OMP END <specific OpenMP directive>
statement3

Statement 2 (may be) executed in parallel
Statement 1 and 3 are executed sequentially
Basic Idea of OpenMP

• Program has sequential part and parallel parts
• Master thread executes sequential part
• Master and slaves execute parallel parts
  – Master creates team of slave threads
  – Very similar to fork-join semantics of Pthreads create/join primitives
OpenMP Programming Model

- Master thread spawns a team of threads as needed.
- Parallelism is added incrementally until desired performance is achieved: i.e. the sequential program evolves into a parallel program.
Basic Idea of OpenMP

• Each thread performs part of the work
• Rule of thumb: One thread per processor
  – But for latest machines one thread per core
  – If SMT (hyperthreading) is enabled, then there can be even more
• Sequential parts executed by single thread
• Dependences in parallel parts require synchronization between threads
Basic Idea of OpenMP

• User states how work in program is to be executed in parallel by threads

• Work parallelism, not data parallelism
  – Data is shared, we don’t want to distribute it
  – But related: in data parallelism, data mapping determines work parallelism
Role of User

• User inserts directives telling compiler how statements are to be executed
  – what parts of the program are parallel
  – how to assign code in parallel regions to threads
  – what data is private (local) to threads

• Compiler generates explicit threaded code
Role of User

- User must remove any dependences in parallel parts
- Or introduce appropriate synchronization
- OpenMP compiler does not check for existence of dependences in parallel code
- Most common error in OpenMP code is not taking a dependence into account
  - When two threads update an object and their updates are not synchronized, we have a race condition
  - Some tools exist to help look for them
OpenMP Implementation

- OpenMP compiler translates code and user directives into multithreaded application
- Calls to run time library routines
- Works on true shared memory machines (SMPs) and DSM architectures
OpenMP Usage

Diagram:
- Annotated Source
- Fortran/C/C++ compiler
- Sequential Program
- Parallel Program
- OpenMP compiler

Sequential compiler

Flow:
- Annotated Source → Fortran/C/C++ compiler
- Fortran/C/C++ compiler → Sequential Program, Parallel Program, OpenMP compiler
OpenMP Usage

• If program is compiled sequentially
  – OpenMP comments and pragmas are ignored
• If code is compiled for parallel execution
  – comments and/or pragmas are read, and
  – drive translation into parallel program
• Ideally, one source for both sequential and parallel program (big maintenance plus)
What is OpenMP?

- Compiler directives and library routines for shared memory programming
  - First version for Fortran 77 published late 1997
- Memory shared, threads have private data
- Work distribution, synchronization and mutual exclusion
Data in OpenMP

• Data assigned to a thread is **private** or **local** to that thread and is not accessible to other threads
• Threads share other data
• Performance problems are likely if two threads write data on same cache line
  – The cache line might “ping-pong” between different caches
  – Called **false sharing**
• Principle: use private data where possible to avoid false sharing
Parallel Loop With Synchronization

The iterations of the loop are shared among the threads.

A single copy of RES is shared between threads. Each thread has its own copy of B.

```c
C$OMP PARALLEL DO PRIVATE(B)
    C$OMP& SHARED(RES)
    DO 100 I=1,NITERS
        B = DOIT(I)
    C$OMP CRITICAL
        CALL CONSUME (B, RES)
    C$OMP END CRITICAL
100    CONTINUE
```

- Only one thread at a time can enter a critical region
The OpenMP* API

Same Stuff in C

- Only one thread at a time can enter a critical region

```c
float res;
#pragma omp parallel
{
    float B;  int i;
    #pragma omp for
    for(i=0;i<niters;i++){
        B = big_job(i);
    #pragma omp critical
        consume (B, RES);
    }
}
```

* The mark “OpenMP” is the property of the OpenMP Architecture Review Board.
Matrix Multiply

for( i=0; i<n; i++ )
    for( j=0; j<n; j++ )
        c[i][j] = 0.0;
for( i=0; i<n; i++ )
    for( j=0; j<n; j++ )
        for( k=0; k<n; k++ )
            c[i][j] += a[i][k]*b[k][j];
Parallel Matrix Multiply

- No loop-carried dependences in i- or j-loop
- Loop-carried dependence on k-loop
- All i- and j-iterations can be run in parallel
Parallel Matrix Multiply (contd.)

- OpenMP permits parallelization of only one loop in loop nest (but this will change)
- Multiple threads can read data in local caches without performance penalty
- So long as no-one writes data
Problem Statement
Matrix Multiply

#pragma omp parallel for
for( i=0; i<n; i++ )
    for( j=0; j<n; j++ )
        c[i][j] = 0.0;
#pragma omp parallel for
for( i=0; i<n; i++ )
    for( j=0; j<n; j++ )
        for( k=0; k<n; k++ )
            c[i][j] += a[i][k]*b[k][j];
for some number of timesteps/iterations {
    for (i=1; i<n; i++)
        for( j=1, j<n, j++)
            temp[i][j] = 0.25 *
                ( grid[i-1][j] + grid[i+1][j]
                 grid[i][j-1] + grid[i][j+1] );

    for( i=1; i<n; i++)
        for( j=1; j<n; j++)
            grid[i][j] = temp[i][j];
}

Jacobi Method
Parallel Jacobi Method

- No dependences between iterations of first (i,j) loop nest
- No dependences between iterations of second (i,j) loop nest
- True and anti-dependence between first and second loop nest in the same timestep
- True dependence between second loop nest and first loop nest of next timestep
Parallel Jacobi (continued)

• First (i,j) loop nest can be parallelized
• Second (i,j) loop nest can be parallelized
• But keep order of loops and timesteps
  – need a barrier at the end of each (i,j) loop nest
• OpenMP automatically inserts a barrier at the end of each parallel loop
Parallel Jacobi Method

for some number of timesteps/iterations {
    for (i=1; i<n; i++) ↩ distribute iterations
        for (j=1, j<n, j++)
            temp[i][j] = 0.25 *
                ( grid[i-1][j] + grid[i+1][j]
                  grid[i][j-1] + grid[i][j+1] );
    … synchronization point …
    for (i=1; i<n; i++) ↩ distribute iterations
        for (j=1; j<n; j++)
            grid[i][j] = temp[i][j];
    … synchronization point …
}
Data Usage in Parallel Jacobi

Although updated by thread on another processor, this is in shared memory. No false sharing, since the update occurs in a different loop nest.
OpenMP Jacobi Method

for some number of timesteps/iterations {
#pragma omp parallel for
for (i=1; i<n; i++)
   for( j=1, j<n, j++ )
      temp[i][j] = 0.25 *
         ( grid[i-1][j] + grid[i+1][j]
            grid[i][j-1] + grid[i][j+1] );
#pragma omp parallel for
for( i=1; i<n; i++)
   for( j=1; j<n; j++ )
      grid[i][j] = temp[i][j];
}
for some number of timesteps {
    for all molecules i
        for all nearby molecules j
            force[i] += f( loc[i], loc[j] );
        for all molecules i
            loc[i] = g( loc[i], force[i] );
}
Molecular Dynamics (continued)

• A typical “data parallel” strategy distributes region to processor

• Processor is assigned molecules in region
Molecular Dynamics (continued)

for some number of timesteps {
    for( i=0; i<num_mol; i++ )
        for( j=0; j<count[i]; j++ )
            force[i] += f(loc[i], loc[index[j]]);
    for( i=0; i<num_mol; i++ )
        loc[i] = g( loc[i], force[i] );
    /* compute new neighbors, count[i] for each i */
}
Molecular Dynamics (continued)

- In first loop nest
  - No loop-carried dependence in outer loop
  - Loop-carried dependence (reduction) in j-loop
- No loop-carried dependence in second loop nest
- True dependence between first and second loop nests
Molecular Dynamics (continued)

- Outer loop in first loop nest can be parallelized
- Second loop nest can be parallelized
- OpenMP performs synchronization between loops
- Memory is shared, so
  - share molecules between threads
- Problems with solution
  - if large differences in number of neighbors, can get load balancing problem
  - cache interferences between threads possible
for some number of timesteps {
#pragma omp parallel for
for( i= ... ; i< ... ; i++ )
    for( j=0; j<count[i]; j++ )
        force[i] += f(loc[i],loc[index[j]]);
#pragma omp parallel for
for( i= ...; i< ... ; i++ )
    loc[i] = g( loc[i], force[i] );
    exchange loc[i] values with neighbors
}
Irregular Codes in OpenMP

• Easy to parallelize irregular computations using OpenMP
• Don’t need to figure out which data elements are neighbors
• But hidden costs as multiple threads may need to update data in the same cache line