The Future of OpenMP

Barbara Chapman
University of Houston

DOE PModels Workshop,
March 2015

http://www.cs.uh.edu/~hpctools
Agenda

- Evolution of OpenMP
- Who are the Users?
- Recent and On-going Change
- Potential Directions
Portable parallel programming across shared memory architectures since 1997:

- **Parallel Regions:**
  - OMP PARALLEL

- **Worksharing:**
  - OMP DO, OMP SECTIONS
  - MASTER, SINGLE

- **Data Environment**
  - SHARED, PRIVATE, FIRSTPRIVATE, THREADPRIVATE

- **Synchronization:**
  - ATOMIC, CRITICAL, BARRIER, ORDERED, FLUSH

- **Runtime functions/environment variables**
  - OMP_NUM_THREADS, OMP_SCHEDULE, etc.

```c
#pragma omp parallel
#pragma omp for schedule(dynamic)
for (i=0; i<N; i++)
    NEAT_STUFF(i);
/* implicit barrier here */
```
Early User Experience, 2000
Naval Research Lab

- NLOM, NCOM Ocean Models
- OpenMP significantly outperformed MPI on representative HALO benchmark
  - Use OpenMP code if possible, else MPI
- OpenMP and shmem versions *scale close to linearly* up to 112 nodes, MPI to 28 nodes, on Origin
Proposed OpenMP Extensions, 1999

- SGI page-based data distribution extensions
  - Allocates *pages* to memory across system nodes
  - Preserves illusion of true shared memory
- HPF-style data mappings
  - Poor performance on page-based system
  - SGI, Compaq

```
!$SGI  
!$OMP   
!$OMP&  
!$SGI+  
DISTRIBUTE array ( CYCLIC (1) )
PARALLEL DO PRIVATE ( i , active)
SHARED ( level )
AFFINITY (i) = DATA ( array ( i ) )
DO i = 1, max
  IF ( array ( i ) >= 1) then
    active = ....
    CALL solve ( active, level, ...)  
END IF
END DO
```

“first-touch” default mapping works pretty well (if developer is aware of it)
OpenMP is maintained by the OpenMP Architecture Review Board (the ARB), which

- Interprets OpenMP
- Writes new specifications - keeps OpenMP relevant
- Works to increase the impact of OpenMP

Members are organizations - not individuals

- Industry: Compaq, Fujitsu, HP, IBM, Intel, Intel KAI, NEC, SGI, Sun
- Other: ASCI, cOMPUnity

Researchers participate via cOMPUnity from 2002 on
Industrial Mixer Code, 2002

![Diagram of an industrial mixer with color-coded density distribution.](image-url)
Omni Compiler: Cluster-enabled OpenMP, 2002

- **OpenMP for a cluster (distributed memory system)**
  - message passing library (MPI, PVM) provides high performance, but difficult and cumbersome.

- **Use software distributed shared memory system SCASH as underlying runtime system on cluster**
  - Page-based DSM
  - Related Work: OpenMP compiler for TreadMarks by Rice (later clOMP)

- **OpenMP**
  - All variables are shared as defaults.
  - No explicit shared memory allocation

- **“shmemb” memory model**
  - All variables declared statically in global scope are private.
  - The shared address space must be allocated by a library function at runtime.
  - Example: SCASH, Unix “shmemb” system call
Task Translation to Reduce Synchronization

- Difficult to explicitly express computations as task graph.
- Compiler translates “standard” OpenMP into collection of work units (tasks) and task graph.
- Analyzes data usage per work unit to reduce synchronization.
- Trade-off between load balance, co-mapping of work units that use same data.
- What is “right” size of work unit?
  - Might need to adjust at run time.
OpenMP version uses same domain decomposition strategy as MPI for data locality, avoiding false sharing and fine-grained remote data access. OpenMP version slightly outperforms MPI version on SGI Altix 3700BX2, both close to linear scaling.
OpenMP Targets, 1997 - 2005

- Initial 1997 release for scientific applications
  - Tailored to array-based computations in Fortran
  - Main market is small SMP workstation or PC
  - C version increased range of SMP-parallel codes

- Provided by vendors on ccNUMA platforms
  - SGI and Compaq, with extensions

- Multicore systems, ca. 2005
  - General-purpose multicore programming
  - Tasks, C and C++ bindings support this
  - Growing compiler support (including ISVs)
OpenMP 3.0 Introduces Tasks, 2008

- Tasks explicitly created and processed

- Each encountering thread packages a new instance of a task (code and data)

- Some thread in the team executes the task

```c
#pragma omp parallel
{
    #pragma omp single
    {
        p = listhead ;
        while (p) {
            #pragma omp task
            process (p)
            p=next (p) ;
        }
    }
}
```
The OpenMP ARB 2009

- OpenMP is maintained by the OpenMP Architecture Review Board (the ARB), which
  - Interprets OpenMP
  - Writes new specifications - keeps OpenMP relevant
  - Works to increase the impact of OpenMP

- Members are organizations - not individuals
  - Current members
    - Permanent: AMD, Caps Entreprise, Cray, Fujitsu, HP, IBM, Intel, Microsoft, NEC, PGI, SGI, Sun, Texas Instruments
    - Auxiliary: ASCI, cOMPunity, EPCC, KSL, NASA, RWTH Aachen

www.compunity.org
The OpenMP ARB 2015

- OpenMP is maintained by the OpenMP Architecture Review Board (the ARB), which
  - Interprets OpenMP
  - Writes new specifications - keeps OpenMP relevant
  - Works to increase the impact of OpenMP

- Members are organizations - not individuals
  - Current members
    - Permanent: AMD, ARM, Convey, Cray, Fujitsu, HP, IBM, Intel, NEC, Nvidia, Oracle, Red Hat, PGI, Texas Instruments
    - Auxiliary: ANL, ASC/LLNL, cOMPunity, EPCC, LANL, LBNL, NASA, ORNL, RWTH Aachen, SNL, TACC, University of Houston

“High-level directive-based multi-language parallelism that is performant, productive and portable”
OpenMP: Who are the Users?

- Small to moderately large scientific and technical applications
  - Initially Fortran only, SMPs, SGI Origin (Altix)
- General-purpose multicore programming
  - Tasks, C and C++ bindings
- Entry-level parallel programmers
- Embedded systems
  - Tasks, kernel offloads
- Large-scale parallel computations
  - Usually in conjunction with MPI
Oct 1997 – 1.0 Fortran
Oct 1998 – 1.0 C/C++
Nov 1999 – 1.1 Fortran: interpretations added
Nov 2000 – 2.0 Fortran (F95, nested locks)
Mar 2002 – 2.0 C/C++
May 2005 – 2.5 Fortran/C/C++ (one API, multiple bindings, memory model, ICVs, terminology)
May 2008 – 3.0 (task execution model, explicit tasks, parallelization of multiple loop levels, nested parallelism; wait policy)
July 2011 - 3.1 (final, mergeable tasks, taskyield, atomic construct)
July 2013 – 4.0 (support for devices, target and data mapping; SIMD loops; thread affinity; task dependences; user defined reductions)

Runtime routines: 10 in 1.1; 19 in 3.0; 28 in 4.0
OpenMP Locality Research

Locations := Affinity Regions, Based on Locales, Places

- OpenMP 3.0: privatize data where possible, optimize cache usage
- “First touch” Implicit data layout
- Represent execution environment by collection of “locations” (Chapel/X10)
- Map data, threads to a location; distribute data across locations
- Align computations with data’s location, or map them explicitly
- Significant performance boost on mid-size SMP systems.

OpenMP 4.0 Affinity

- OpenMP Places and thread affinity policies
  - `OMP PLACES` to describe hardware regions
  - `affinity(spread|compact|true|false)`

- **SPREAD**: spread threads evenly among the places

  spread 8

  ![Spread 8 Diagram](image)

- **COMPACT**: collocate OpenMP thread with master thread

  compact 4

  ![Compact 4 Diagram](image)
OpenMP for Accelerators

```c
#pragma omp target data device (gpu0) map(to:n, m, omega, ax, ay, b, f[0:n][0:m]) map(tofrom:u[0:n][0:m]) map(alloc:uold[0:n][0:m])

while ((k<=mits)&&(error>tol))
{
    // a loop copying u[] to uold[] is omitted here
    ...
    #pragma omp target device(gpu0)

#pragma omp parallel for private(resid,j,i) reduction(+:error)
    for (i=1;i<(n-1);i++)
    for (j=1;j<(m-1);j++)
    {
        resid = (ax*(uold[i-1][j] + uold[i+1][j]) +
            ay*(uold[i][j-1] + uold[i][j+1])+ b * uold[i][j] - f[i][j])/b;
        u[i][j] = uold[i][j] - omega * resid;
        error = error + resid*resid ;
    } // rest of the code omitted ...
    
#pragma omp target data
```

Early Experiences With The OpenMP Accelerator Model; Chunhua Liao, Yonghong Yan, Bronis R. de Supinski, Daniel J. Quinlan and Barbara Chapman; International Workshop on OpenMP (IWOMP) 2013, September 2013
Looking Ahead: OpenMP 4.1

- **Device construct enhancements**
  - more control and flexibility in specifying data movement between host and devices
  - asynchronous, data flow execution support with addition of `nowait` and `depends`
  - multiple devices
  - “deep copy” for pointer-based structures/objects

- **Loop parallelism enhancements**
  - extended `ordered` clause to support `do-across` (e.g. wavefront) parallelism for loop nests
  - new `taskloop` construct for asynchronous loop parallelism with control over task grain size

- **Array reductions for C and C++**

- **Under consideration:**
  - memory affinity
  - task priorities (very likely)

On-going work also on interoperability:
Resource management, other threads, other APIs, multiple OpenMP computations
Feature Set: Future Directions

- Broad user base is a strength, but potential tension between general-purpose programming and HPC
  - Sometimes seen in choice of defaults
- Continued enhancement of expressivity of tasks
- Data locality? Now we have
  - Places, binding of threads to places; device data placement, data motion to/from devices, explicit data allocation
  - (Soon) asynchronous computation on host and device

- How can we build on top of this?
  - Affinity of data with places? Affinity of tasks to places?
  - (Page-based?) Mapping / migration of data to collection of places?
  - Modification of places? Virtualization?
  - Additional memory allocation / mapping information?
Subteams of Threads? GUI Threads?

Thread Subteam: subset of threads in a team
- Overlap computation and communication (MPI)
- Concurrent worksharing regions
- Additional control of locality of computations and data
- Handle loops with little work

for (j=0; j< ProcessingNum; j++)
    #pragma omp for schedule(dynamic)
    subteam(2:omp_get_num_threads()-1)
    for (k=0; k<M; k++) {
        ProcessData(); // data processing
    } // subteam-internal barrier

Increases expressivity of single-level parallelism
Dynamic Program Adaptation

- Reasonably amenable to dynamic adaptation
  - Adjustment of thread count, schedule
  - Adaptive barriers, reduction routines
  - Runtime decisions
  - Tasks, mergeable
- Use of performance interface to inform dynamic tools
  - Can help adjust data layout, find memory performance problems
- Need to develop more runtime techniques
XPRESS: OpenMP over HPX

- ParalleX execution model:
  dynamic adaptive resource management; message-driven computation; efficient synchronization; global name space; task scheduling
- OpenMP translation mostly works
  No direct interface to OS threads
    - No tied tasks; Threadprivate is tricky, slow
    - Doesn’t support Places, private memory
    - OpenMP task dependencies via Futures
    - HPX locks faster than OS locks
- Very interesting for “all-task” translation / optimization
Architecture-Aware Task Translation

- Restructure work units
  - Merging (or splitting) work units for better granularity
  - Guided by parameterized cost model
- Application structural representation
  - Work units and dependences
  - Data distribution among places
- Compile time approximation
  - Data mapping onto places
  - Data binding with work unit
  - Decision honored by runtime
    - But may be adapted and refined.

An “all task” approach
Program Development: Observations

- Scalability greatly influenced by programming style, code structuring and inherent suitability.

- OpenMP is a prescriptive model
  - Coarse grain approach to parallelization best, but might require significant rewriting of code
  - Data layout, access pattern (locality, affinity) has always mattered for performance
  - Overheads of features understood
  - Cache effects, especially false sharing, can distort performance

- Tool support for creation of OpenMP code with high locality needed
Questions?