An Overview of OpenMP

SoHPC, 2021

Outline

- What is OpenMP
- Terminology
- Timeline
- Code Design
- Directives and compiling

OpenMP *Open specifications for Multi Processing*

- is an API for shared-memory parallel computing;
- is an open standard for portable and scalable parallel programming, multi-platform;
- is flexible and easy to implement;
- is a specification for a set of compiler directives, library routines, and environment variables;
- is designed for C, C++ and Fortran.

ICHEC
Irish Centre for High-End Computing

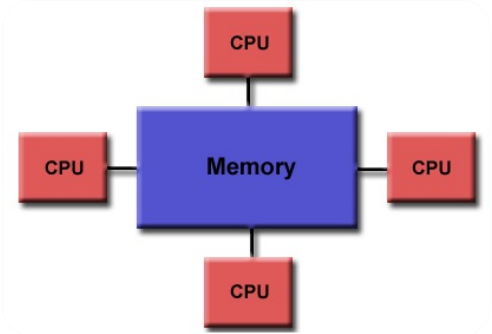
PRACE

OpenMP

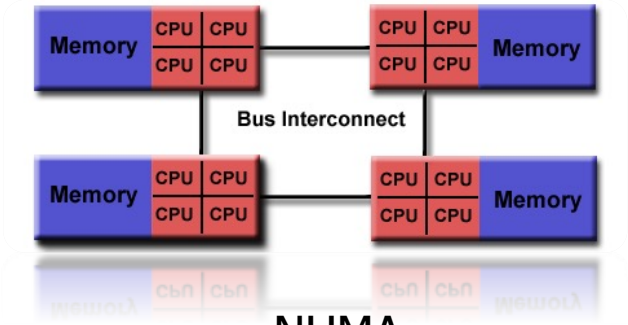
The OpenMP API specification for parallel programming

OpenMP Programming Model

OpenMP is designed for multi-processor/core UMA or NUMA shared memory systems.



UMA

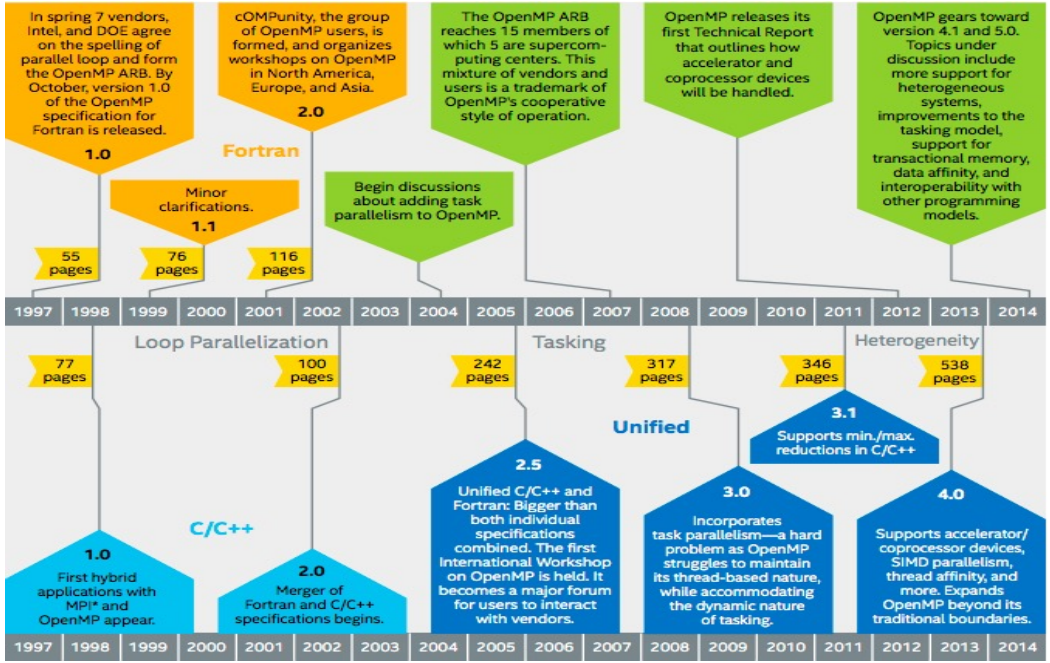


NUMA

Terminology

- OpenMP thread: a is an instance of a program + its data.
- thread team: a set of threads which co-operate on a task
- master thread: the thread which co-ordinates the team
- thread-safe: correctly executed by multiple threads
- OpenMP directive: line of code with meaning only to certain compilers
- construct: an OpenMP executable directive
- clause: controls the scoping of variables during the execution

Timeline


In spring 7 vendors, Intel, and DOE agree on the spelling of parallel loop and form the OpenMP ARB. By October, version 1.0 of the OpenMP specification for Fortran is released.
1.0
cOMPunity, the group of OpenMP users, is formed, and organizes workshops on OpenMP in North America, Europe, and Asia.
2.0
The OpenMP ARB reaches 15 members of which 5 are supercomputing centers. This mixture of vendors and users is a trademark of OpenMP's cooperative style of operation.
OpenMP releases its first Technical Report that outlines how accelerator and coprocessor devices will be handled.
OpenMP gears toward version 4.1 and 5.0. Topics under discussion include more support for heterogeneous systems, improvements to the tasking model, support for transactional memory, data affinity, and interoperability with other programming models.
Fortran
Minor clarifications.
1.1
Begin discussions about adding task parallelism to OpenMP.
55 pages
76 pages
116 pages
1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
Loop Parallelization
Tasking
Heterogeneity
77 pages
100 pages
242 pages
317 pages
346 pages
538 pages
Unified
3.1
Supports min./max. reductions in C/C++
2.5
Unified C/C++ and Fortran: Bigger than both individual specifications combined. The first International Workshop on OpenMP is held. It becomes a major forum for users to interact with vendors.
3.0
Incorporates task parallelism—a hard problem as OpenMP struggles to maintain its thread-based nature, while accommodating the dynamic nature of tasking.
4.0
Supports accelerator/coprocessor devices, SIMD parallelism, thread affinity, and more. Expands OpenMP beyond its traditional boundaries.
C/C++
1.0
First hybrid applications with MPI* and OpenMP appear.
2.0
Merger of Fortran and C/C++ specifications begins.
1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014

Timeline

- 2
  - Loop parallelization.
- 3
  - Task parallelization + extension to loop construct
- 3.1
  - Thread affinity + more loop functionality
- 4.0
  - Support for vectorization and accelerators. More features for affinity.
  - Task dependency
- 4.5
  - Taskloop construct.
- 5
  - Task reduction

Writing and OpenMP App

- Identify compute intensive tasks in serial code
  - Ideally, these tasks can be worked on independently of the others.
- Map tasks onto "threads of execution" (processors/cores)
- Decide data type. Threads have *shared* and *private* data
  - Shared: used by more than one thread
  - Private: local to each thread
- Write source code using compiler directives and RTL functions.
- Choices may depend on (among many things)
  - The nature of the problem
  - The level of performance needed
- Decide the necessary environment variables.

Main components

- Compiler Directives and Clauses: appear as pragmas in C/C++ and comments in Fortran, executed when the appropriate OpenMP flag is specified
  - Control
    - how is parallelism created?
    - what ordering is there between operations?
  - Synchronization
    - What operations are used to coordinate parallelism ?
    - What operations are atomic (indivisible)?
  - Data
    - What data is private or shared?
    - How is logically shared data accessed or communicated?

Main components

C/C++:

#pragma omp *directive-name [clause[clause]...]*
{...}

Fortran free form:

!$omp *directive-name [clause[clause]...]*
...
!$omp end *directive-name*

Fortran fixed form:

!$omp | c$omp | *$omp *directive-name [clause[clause]...]*
...
!$omp | c$omp | *$omp end *directive-name*

Compiling:

	Compiler	Flag
Intel	icc (C) icpc (C++) ifort (Fortran)	-qopenmp
GNU	gcc (C) g++ (C++) g77/gfortran (Fortran)	-fopenmp

See: http://openmp.org/wp/openmp-compilers/ for the full list.

Thank You

Any questions?