Advanced Technical Skills (ATS) North America

Shared Memory Programming OpenMP

IBM High Performance Computing
February 2010

Y. Joanna Wong, Ph.D.
yjw@us.ibm.com

Terminology review

- Core/Processor , Node, Cluster
  - Multi-core architecture is the prevalent technology today
  - IBM started delivering dual-core Power4 technology in 2001
  - Intel multi-core x86_64 microprocessors
    - Quad-core Intel Nehalem processors
    - Quad-core / Six-core Intel Dunnington processors
  - A node will have multiple sockets
    - The x3850 M2 has 4 sockets. Each socket with the quad-core Intel Dunnington processors.
    - The x3950 M2 has 4 “nodes”, a total of 64 cores.
    - The x3950 M2 runs a single operating system
  - A cluster have many nodes connected via interconnect
    - Ethernet – Gigabit, 10G
    - InfiniBand
    - Other proprietary interconnect, e.g. myrinet from Myricom

Terminology review

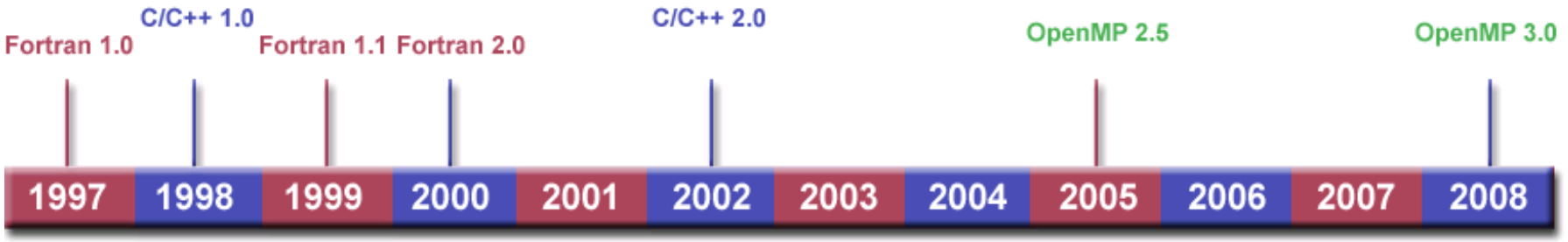
- Thread vs process
  - Thread
    - An independent flow of control, may operate within a process with other threads
    - An schedulable entity
    - Has its own stack, registers and thread-specific data
    - Set of pending and blocked signals
  - **Process**
    - Do not share memory or file descriptors between processes
    - A process can own multiple threads

Shared memory architecture

- Shared memory system
  - Single address space accessible by multiple processors
  - Each process has its own address space not accessible by other processes
- Non Uniform Memory Access (NUMA)
  - Shared address space with cache coherence for multiple threads owned by each process
- Shared memory programming enable an application to use multiple cores in a single node
  - An OpenMP job is a process, creating one or more SMP threads. All threads share the same PID.
  - Usually no more than 1 thread per core for parallel scalability in HPC applications

What is OpenMP?

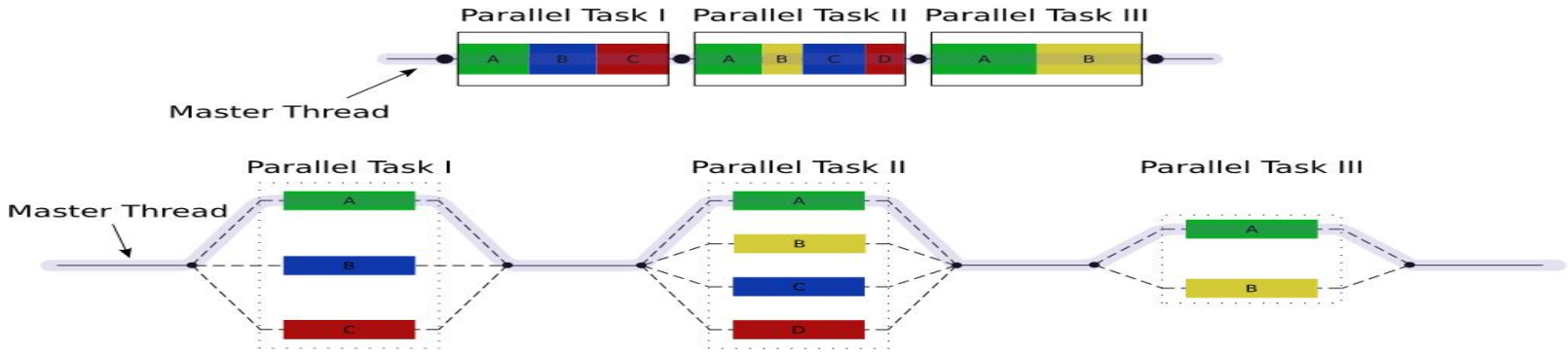
- References
  - OpenMP website: http:// openmp.org
  - OpenMP API 3.0 http://www.openmp.org/mp-documents/spec30.pdf
  - OpenMP Wiki: http://en.wikipedia.org/wiki/OpenMP
  - OpenMP tutorial: https://computing.llnl.gov/tutorials/OpenMP
- History
  - OpenMP stands for: Open Multi-processing or Open specifications for Multi-Processing via collaborative work between interested parties from hardware and software industry, government and academia.
  - First attempt at standard specification for shared-memory programming started as draft ANSI X3H5 in 1994. Waning interest because distributed memory programming with MPI was popular.
  - First OpenMP Fortran specification published by the OpenMP Architecture Review Board (ARB) in Oct 1997. The current version is OpenMP v3.0 released in May 2008.



source: https://computing.llnl.gov.tutorials.OpeMP/

What is OpenMP ?

- Standard among shared memory architectures that support multi-threading, with threads being allocated and running concurrently on different cores/processors a server.
- Explicit parallelism programming model
  - Compiler directives that marked sections of code to run in parallel. The master thread (serial execution on 1 core) forks a number of slave threads. The tasks are divided to run concurrently amongst the slave threads on multiple cores (parallel execution).
  - Runtime environment allocates threads to cores depending on usage, load, and other factors. Can be assigned by compiler directives, runtime library routines, and environment variables.
  - Both data and task parallelism can be achieved.



source: en.wikipedia.org/wiki/OpenMP

What is OpenMP?

- Advantages
  - Portable
    - API specification for Fortran (77, 90, and 95), C, and C++
    - Supported on Unix/Linux and Windows platforms
  - Easy to start
    - Data layout and decomposition are handled automatically by directives.
    - Can incrementally parallelize a serial program – one loop at a time
    - Can verify correctness and speedup at each step
    - Provide capacity for both coarse-grain and fine-grain parallelism
    - Significant parallelism could be achieved with a few directives
- Disadvantages
  - Parallel scalability is limited by memory architecture – may saturate at a finite number of threads (4, 8, or 16 cores). Performance degradation observed with increasing threads when the number of cores compete for the shared memory bandwidth.
  - Large SMP with more than 32 cores are expensive.
  - Synchronization between a subset of threads is not allowed.
  - Reliable error handling is missing.
  - Missing mechanisms to control thread-core/processor mapping.

OpenMP Fortran directives

- Fortran format
  - Must begin with a sentinel. Possible sentinels are:

    !$OMP

    C$OMP

    *$OMP
  - MUST have a valid directive (and only one directive name) appear after the sentinel and before any clause
  - OPTIONAL Clauses can be in any order, repeated as necessary
  - Example:

    !$OMP PARALLEL DEFAULT(SHARED) PRIVATE(BETA, PI)

    Sentinel OpenMP directive optional clauses
  - The “end” directives for several Fortran OpenMP directives that come in pairs is optional. Recommended for readability.

    !$OMP *directive*

    [structured block]

    !OMP end *directive* <= optional

OpenMP Fortran directives

- **Fortran Free form source**
  - !$OMP is only accepted sentinel
  - All Fortran free form rules (line length, white space, continuation, comment) apply for entire directive line
  - Initial directive must have space after senitel
  - Continuation lines must have an ampersand as the last non-blank character. The following lines must begin with a senitel and the continuation directives
  - Example

```
!$OMP PARALLEL  &
!$OMP       DEFAULT (SHARED)   PRIVATE (BETA, PI)
```

- **Fortran Fixed form source**
  - All possible sentinels (!$OMP C$OMP *$OMP) are accepted. Must start in column 1
  - All Fortran fixed form rules (line length, white space, continuation, comment) apply for entire directive line
  - Initial directive must have space/zero in column 6
  - Continuation lines must have a non-space/non-zero in column 6
  - Example

```
*$OMP PARALLEL
        c DEFAULT ( SHARED)
        c PRIVATE (BETA, PI)
```

OpenMP C/C++ directives

- C/C++ format
  - Required for all Open C/C++ directives

    #pragma omp
  - Must have a valid OpenMP directive (and only one directive name) appear after the pragma and before any clause
  - OPTIONAL Clauses can be in any order, repeated as necessary
  - A newline is required to preceed the structured block that is enclosed by the directive
  - Each directive applies only to one succeeding structured block, which could be a statement.
  - Example

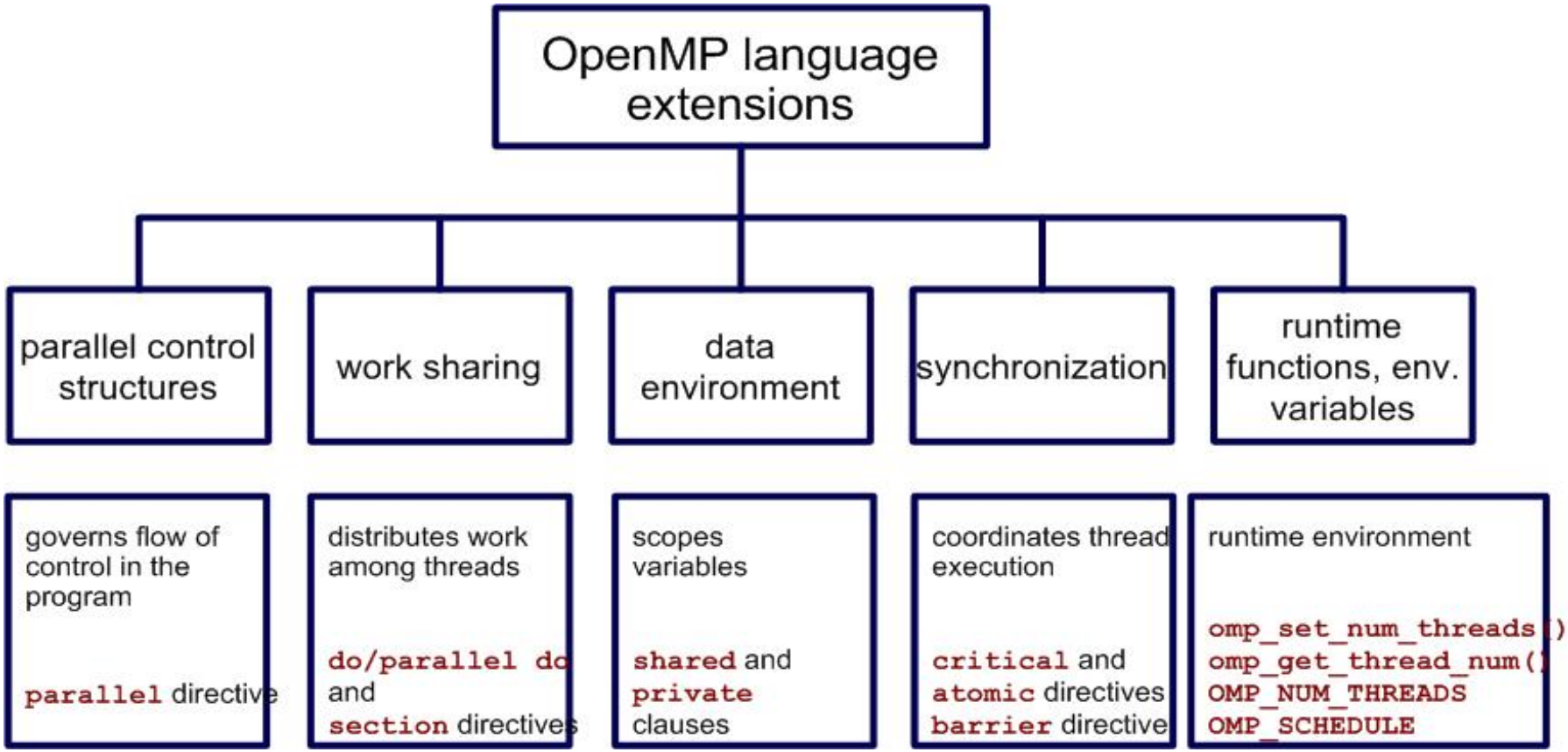
    #pragma omp parallel default(shared) private(beta, pi)

    required OpenMP directive optional clauses
- Follow conventions of C/C++ standard for compiler directives
- Case sensitive
- Can be continued by escaping newline character with a backslash "\" at the end of the line

  **#pragma omp parallel default(shared) **

  private(beta, pi)

OpenMP constructs



source: en.wikipedia.org/wiki/OpenMP

OpenMP constructs
!$OMP FORTRAN / #pragma omp c

- Thread creation construct

  PARALLEL / parallel

  – Original process (master thread) forked additional threads to run code enclosed in the parallel construct.

  – Thread ID for master thread is 0.

- Work-sharing constructs

  – DO / for
    - split up loop iterations among the threads in a parallel region

  – SECTIONS / sections
    - divide consecutive but independent section(s) of code block amongst the threads
    - barrier implied at the end unless the NOWAIT/nowait clause if used.

  – SINGLE / single
    - code block to be executed by 1 thread
    - barrier implied at the end unless the NOWAIT/nowait clause if used

  – MASTER / master : code block to be executed by master thread only (no barrier on other threads implied)

OpenMP constructs

- Work-sharing (cont'd)
  - WORKSHARE (Fortran only)
    - execution of structured block is divided into separate units of work, each to be executed once
    - Structured block must consist only
      - array or scalar assignment
      - FORALL and WHERE statements
      - FORALL and WHERE constructs
      - atomic, critical or parallel constructs
- Data scoping clauses
  - SHARED / shared **:** data are visible and accessible by all threads simultaneously. All variables in work-sharing region are shared by default, except the loop iteration.
  - PRIVATE / private **:** data is private to each thread. A private variable is not initialized. Loop iteration counter in work-sharing region is private.
  - DEFAULT / default **:** default data scoping in the work-sharing region (shared / private / none)

OpenMP constructs

- Synchronization clauses
  - CRITICAL / critical **:** enclosed code block to be executed by all threads, but only one thread at a time
  - ATOMIC / atomic : a mini-critical section specifying that a memory location must be updated atomically
  - ORDERED / ordered : iteration of enclosed code block is executed in same order as sequential
  - BARRIER / barrier : synchronizes all threads at the barrier. The barrier synchronization is implied at end of work-sharing construct.
  - NOWAIT / nowait : thread completing code block execution can proceed, at end of work-sharing construct where barrier is implied by default

- Scheduling clauses
  - SCHEDULE(*type, [chunk]*) / schedule(*type, [chunk]*)
    - STATIC / static : loop iterations are divided into *chunk* size and statically assigned to the threads. The iterations are evenly divided contiguously if no *chunk* is specified.
    - DYNAMIC / dynamic : loop iterations are divided into *chunk* size and dynamically scheduled amongst the threads. The default chunk size is 1.
    - GUIDED / guided : For a *chunk_size* of 1, the size of each chunk is proportional to the number of unassigned iterations divided by the number of threads in the team, decreasing to 1. For a *chunk_size* with value *k* (greater than 1), the size of each chunk is determined in the same way, with the restriction that the chunks do not contain fewer than *k* iterations (except for the last chunk to be assigned, which may have fewer than *k* iterations).
    - RUNTIME /runtime : scheduling decision deferred until runtime by environment variable OMP_SECHEULE
    - AUTO / auto : scheduling decision delegated to the compiler and/or runtime system.

- IF/ if control
  - If the condition is met, the code block will be parallelized. Otherwise, the code block executes serially.
- Initialization
  - FIRSTPRIVATE / firstprivate : the data is private to each thread but is initialized with value of the variable of same name from the master thread
  - LASTPRIVATE / lastprivate : the data is private to each thread. The value of the private data if the current iteration is the last iteration is copied to the global variable of same name outside the parallel region.
  - THREADPRIVATE / threadprivate :
    - The data is a global data but it is private in each parallel region during the runtime. The value of threadprivate data is preserved across parallel regions.

- Data copying
  - COPYIN / copyin**:** assign the same value to THREADPRIVATE variables for all threads
  - COPYPRIVATE / copyprivate: broadcast values of private variables acquired in a single thread directly to private variables in all other threads (associated with SINGLE directive)
  - REDUCTION / reduction: a reduction on the variables in the list

- Compiling OpenMP applications on Linux
  - GNU compiler since version 4.2

    -fopen
  - Intel compiler

    -openmp
  - PathScale compiler

    -mp
  - PGI compiler

    -mp
- A few compilers have early implementation for OpenMP 3.0
  - GCC 4.3.1
  - Intel C++ 11 compiler