

The Portland Group

Distributed OMP: Extensions to OpenMP for SMP Clusters

John Merlin, Douglas Miles, Vincent Schuster

jhm@pgroup.com

<http://www.pgroup.com>

EWOMP'2000, EPCC, Edinburgh

September 14 – 15, 2000

Introduction

- *OpenMP*: set of extensions to Fortran and C for shared-memory (SM) parallel programming.
 - Portable; easy to use; incremental parallelization.
 - *But* – only for SMPs. No data locality features.
- Increasing interest in hierarchical memory architectures, e.g. SMP clusters, CC-NUMA.
- Here we consider augmenting OpenMP for this purpose.
 - *'Distributed OMP'* (*'DMP'*) = OpenMP + data locality features.

DMP objectives:

- Portability across the memory hierarchy spectrum:
 - SMP clusters (1-sided or message-passing comms)
 - CC-NUMA machines (e.g. Origin 2000).
 - SMP and DM systems.

May need a variety of implementations.

- Compilable as OpenMP for SMP systems.
- Allow separate compilation and incremental parallelization.
- Straightforward basic implementation.
- Execution efficiency.

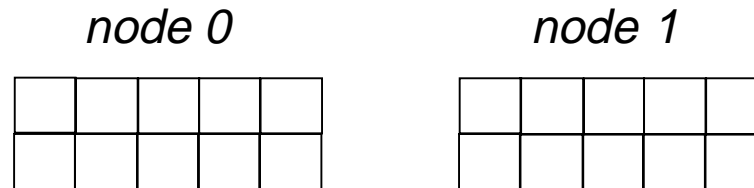
Overview

- Introduction
- DMP extensions
 - DISTRIBUTE directive
 - ON HOME directive
 - Library routines and environment variables
- Example execution models:
 - SMP cluster with one-sided communications
 - SMP cluster with message-passing
 - DSM or “pure” SMP system
- Conclusions

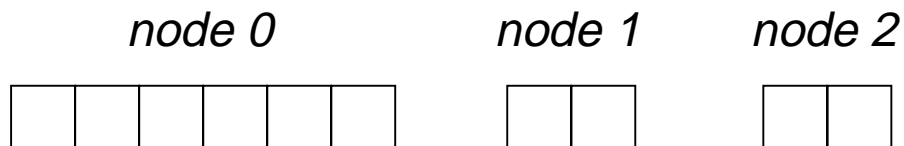
DMP extensions: (1) DISTRIBUTE directive

Partitions an array over *node* memories (or pages):

```
INTEGER d(2,10)  
!$DMP DISTRIBUTE d(*,BLOCK)
```



```
INTEGER d(10), dist_array(3)  
PARAMETER (dist_array = (/6,2,2/))  
!$DMP DISTRIBUTE d (GEN_BLOCK(dist_array))
```



- *Mapping*: storage layout in node memories (or pages):
 - *Distributed*: appearing in DISTRIBUTE directive
 - *Undistributed*: not appearing in DISTRIBUTE.
- *Any* array may be distributed (local, allocatable, dummy argument, or in a module or common block).
- *Constraint*: If a common block array is distributed, it must be distributed identically throughout program.

(This need not apply to ‘pure’ SMP / DSM targets.)

Mapping of procedure arguments

- Explicit interfaces not necessary.
- Actual arguments can have *any mapping*. *Dummy argument inherits mapping from actual*. E.g.:

```
REAL, DIMENSION (10) :: a, b, c
!$DMP DISTRIBUTE b (BLOCK)
...
CALL s (a)      ! undistributed
!$OMP PARALLEL, PRIVATE (c)
CALL s (b)      ! distributed
CALL s (c)      ! PRIVATE
```

- DISTRIBUTE directives for dummy args only apply if & when OMP PARALLEL is encountered in procedure.

DMP extensions: (2) ON HOME directive

- Can appear directly after an OMP [PARALLEL] DO, SECTION or SINGLE directive.
- Executes thread on the node that stores a particular scalar variable. E.g.:

```
        REAL a(n), b(n)
!$DMP DISTRIBUTE (BLOCK) :: a, b
        ...
!$OMP PARALLEL DO
!$DMP DMP ON HOME (a(i))
        DO i=2,n-1
            a(i) = (b(i+1) + b(i-1))/2.0
        ENDDO
```

E.g.2: !\$DMP DISTRIBUTE a(BLOCK)

 !\$OMP PARALLEL SECTIONS

 !\$OMP SECTION

 !\$DMP ON HOME (a(i))

 CALL s (a(i))

 !\$OMP SECTION

 !\$DMP ON HOME (a(j))

 CALL t (a(j))

 ...

- ON HOME variable needn't be distributed. It must not be PRIVATE.
- If ON HOME is incompatible with OMP SCHEDULE, ON HOME takes priority (in DoMP).

DMP extensions:

(3) Library routines & environment vars

E.g.:

```
INTEGER FUNCTION dmp_get_num_nodes ( )  
  
SUBROUTINE dmp_get_num_procs (np)  
  INTEGER, INTENT(OUT) :: np (num_nodes)  
  
SUBROUTINE dmp_set_num_threads (nt)  
  INTEGER, INTENT(IN) :: nt (num_nodes)  
  
SUBROUTINE dmp_get_num_threads (nt)  
  INTEGER, INTENT(OUT) :: nt (num_nodes)
```

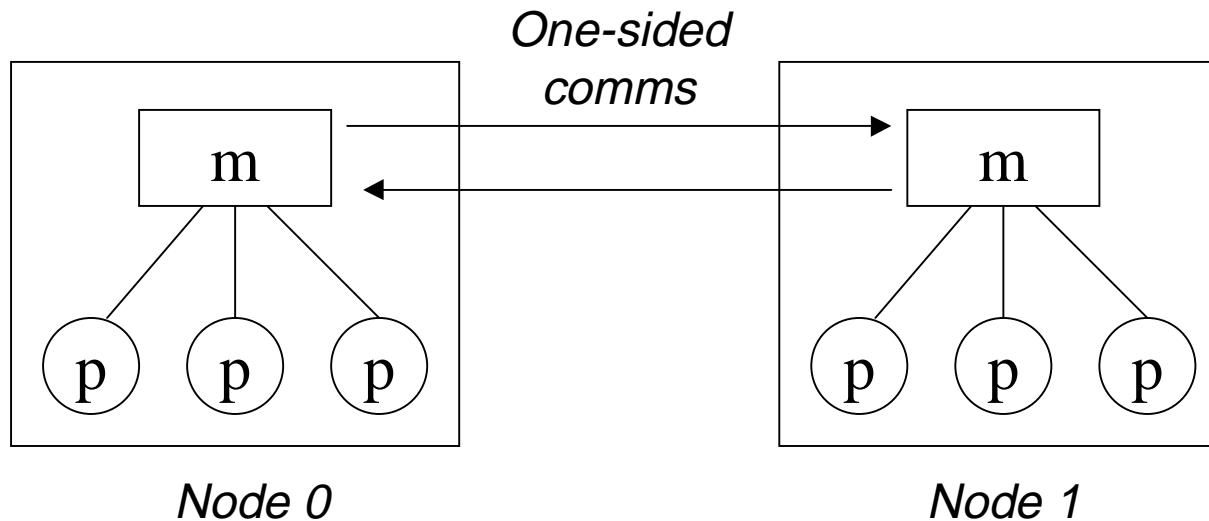
The OpenMP library routines `omp_get_num_threads()`
etc would relate to totals over all nodes.

Example execution models

Will consider example execution models for:

- SMP (or DM) cluster with one-sided communications (e.g. VIA or shmem).
- SMP (or DM) cluster with message-passing.
- 'Pure' SMP

Execution on SMP cluster with one-sided communications



- Execution model:
 - Each OpenMP thread executes on one CPU.
 - Each CPU can execute multiple threads.

Serial and parallel regions

Serial region: Code outside OMP PARALLEL – END PARALLEL. Single threaded (master thread).

Parallel region: Code within OMP PARALLEL – END PARALLEL. Multi-threaded.

Note: same code may be executed in serial or parallel:

```
CALL sub(x)           ! Serial region
!$OMP PARALLEL DO
DO i = 1,n
  CALL sub(a(i))      ! Parallel region
ENDDO
```

Data mapping in serial regions

Assume master thread runs on node 0.

Data mapping in serial region:

- *Undistributed variable*: stored on node 0.
- *Distributed variable*: possibilities:
 - stored on node 0 (i.e. ignoring DISTRIBUTE)
 - Pro*: only local accesses
 - Con*: re-distribution on entry to / exit from parallel region
 - stored distributed.
 - Pro*: no re-distribution
 - Con*: requires non-local accesses in serial region.
 - distributed storage *and* execution (...see later)

Data mapping in parallel regions

Data mapping in parallel regions:

- *OMP PRIVATE variables*: stored on executing node
- *Distributed Shared variables*: stored distributed
- *Undistributed Shared variables*: Can be:
 - Stored on one node:
 - All accesses may require comms.
 - Replicated, i.e. a copy stored on every node.
 - Reads are local; writes need broadcast.
 - Good for read-only objects, e.g. INTENT(IN) arguments.

Summary of mappings in parallel region

<i>Mapping</i>	<i>Meaning</i>	<i>Comms for</i>
distributed	One copy distributed over all nodes	R/W
localized on node n	One copy on node n (often 0)	R/W
replicated	One copy per node	W
private or node-private	One copy per thread or node, on executing node	no comms

Procedure calls

Proposal: generate 2 clones of every procedure:

- *Serial version*: called from serial regions;
- *Parallel version*: called from parallel regions.

Serial version

- Called on node 0 (by master thread)
- Dummy arguments & local variables localized on node 0. All accesses to them are local.
- DISTRIBUTE directives for dummy args & local vars take effect at start of parallel region.

E.g.:

```
      SUBROUTINE s (d)
      REAL, DIMENSION(100) :: d, v1, v2, v3
!$DMP DISTRIBUTE (BLOCK)    :: d, v1, v2

      ...      ! serial region:
      ...      !    d,v1,v2,c3 on node 0

!$OMP PARALLEL, PRIVATE (v2)
      ...      ! parallel region:
      ...      !    d,v1 distributed
      ...      !    v2 privatized
      ...      !    v3 on node 0
```

Parallel version of procedure

- *Dummy arguments* inherit mapping of actual args.
- *Local variables* are privatized on executing node. This follows from OpenMP rules.
- As an optimization, dummy args could have *descriptive* mapping directives, which *assert* mapping of actual:

```
!$DMP DISTRIBUTE a(BLOCK)
    . . .
!$OMP PARALLEL, PRIVATE (p)
    CALL s(a,p)
    . . .
    SUBROUTINE s(d1,d2)
!$DMP DISTRIBUTE* d1(BLOCK) ! assertion
!$DMP PRIVATE*   d2         ! assertion
```

Procedure calls from parallel region

- *INTENT(IN)* arguments can be *privatized*.
(E.g. if it's an expression, or INTENT known from explicit interface or IPA.)
- Otherwise, argument is passed-in with existing mapping (private, replicated, distributed, localized).
- Constraint: If actual argument is a *distributed* array, section or element, can't use sequence association.
 - Does not apply to “pure” SMP or DSM systems.
 - Can check this at runtime.

Execution on SMP cluster with message-passing

- Above execution model assumes one-sided comms between SMP nodes.
- If only message-passing between nodes, can use *distributed execution*:
 - execute code in each thread in distributed fashion on 1 CPU in *every* node (like HPF).
 - see: Leair, Merlin et al, *Proc. CPC'2000*, Aussois, France, Jan 2000.
- Could also use this in serial region in previous execution model, as an optimization (?).

Execution on DSM or “pure” SMP

Various possibilities:

- Ignore the DMP extensions! (...i.e. execute it as OpenMP).
- Use DISTRIBUTE directives to enhance localization

Conclusions

- *Distributed OpenMP (or DMP)*: a proposal to extend OpenMP with data locality features.

We have presented ideas for a small initial set of extensions, and an execution model for SMP clusters with one-sided comms.

- DMP properties:
 - Portable across the memory hierarchy spectrum:
 - SMP clusters (1-sided or message-passing comms)
 - CC-NUMA machines (e.g. Origin 2000).
 - “Pure” SM and DM systems.
 - Compilable as OpenMP for ‘pure’ SMP systems.
 - Allows separate compilation and incremental parallelization.
- Can be extended to C and C++.