OpenMP on MPPs and clusters of SMP nodes using Intel® Compilers with Cluster OpenMP

Rolf Rabenseifner, Bettina Krammer
rabenseifner@hlrs.de, krammer@hlrs.de
University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de
Sep 11, 2006
Used version of Intel® Compiler with Cluster OpenMP under Linux: Compiler 9.1

Goal

- To run OpenMP parallel applications on clusters
- Ease of OpenMP parallelization on cheap clusters
- Instead of
  - expensive MPI parallelization, or
  - expensive shared memory / ccNUMA hardware
Intel® Compilers with Cluster OpenMP – Consistency Protocol

Basic idea:
- Between OpenMP barriers, data exchange is not necessary, i.e., visibility of data modifications to other threads only after synchronization.
- When a page of sharable memory is not up-to-date, it becomes protected.
- Any access then faults (SIGSEGV) into Cluster OpenMP runtime library, which requests info from remote nodes and updates the page.
- Protection is removed from page.
- Instruction causing the fault is re-started, this time successfully accessing the data.

Consistency Protocol Detail of Intel® Cluster OpenMP

Node 0
- Pages: A B C
- Write A[1]
- Write C[1]
- OMP Barrier
- notices received and propagated by master thread
- WriteNotice(0A,2A,2B,0C)
- WriteNotice(0A,1B,0C)
- Calculate Diffs(A,TwinA)

Node 1
- Pages: A B C
- Write B[2]
- OMP Barrier
- WriteNotice(1B)
- node page
- Read A[1]
- Page Fault
- Diff Request(A)
- Re-Read A[1]
- Calculate Diffs(A,TwinA)

Node 2
- Pages: A B C
- Write A[2]
- Write B[1]
- OMP Barrier
- WriteNotice(2A,2B)
- page A starts read-only
- Page Fault
- allocate (TwinA)
- memcpy (TwinA := A)
- Re-Write A[2]
- by additional service thread
- Calculate Diffs(A,TwinA)
Real consistency protocol is more complicated

- Diffs are done only when requested
- Several diffs are locally stored and transferred later if a thread first reads a page after several barriers.
- Each write is internally handled as a read followed by a write.
- If too many diffs are stored, a node can force a "repossession" operation, i.e., the page is marked as invalid and fully re-sent if needed.
- Another key point:
  - After a page has been made read/write in a process, no more protocol traffic is generated by the process for that page until after the next synchronization (and similarly if only reads are done once the page is present for read).
  - This is key because it's how the large cost of the protocol is averaged over many accesses.
  - I.e., protocol overhead only "once" per barrier
- Examples in the Appendix

Comparison: MPI based parallelization ↔ DSM

- MPI based:
  - Potential of boundary exchange between two domains in one large message
    - Dominated by bandwidth of the network
- DSM based (e.g. Intel® Cluster OpenMP):
  - Additional latency based overhead in each barrier
    - May be marginal
  - Communication of updated data of pages
    - Not all of this data may be needed
    - i.e., too much data is transferred
    - Packages may be too small
    - Significant latency
  - Communication not oriented on boundaries of a domain decomposition
    - probably more data must be transferred than necessary

hybrid MPI+OpenMP → OpenMP only

by rule of thumb: Communication may be 10 times slower than with MPI
Comparing results with heat example

- Normal OpenMP on shared memory (ccNUMA) NEC TX-7

Heat example: Cluster OpenMP Efficiency

- Cluster OpenMP on a Dual-Xeon cluster

Efficiency only with small communication foot-print
Usage – key concept

- Variables that are used with "shared" data scope must be allocated as "sharable"
- If data declaration and "shared" data scope is within same lexicographic scope, "sharable" is done automatically
- Otherwise
  - C: #pragma intel omp sharable(var1, ...)
  - Fortran: !dir$ omp sharable(var1, ...)
- malloc() and Fortran Cray pointer must be substituted by
  - kmp_sharable_malloc()
  - kmp_sharable_realloc()
  - kmp_sharable_ralloc()
  - kmp_sharable_free()
- Fortran call by reference may imply that a temporary variable must be used

Acknowledgements

- This lecture is based on an Intel Cluster OpenMP Tutorial (with examples 1-3) by Larry Meadows and James Cownie
- The Intel® Cluster OpenMP (compiler 9.1) was installed by Dmitri Chubarov and Bettina Krammer
Summary

- Intel® Cluster OpenMP can be used for programs with small communication foot-print!
- Source code modification needed: shared variables must be allocated in sharable memory
- It works!
- But efficiency strongly depends on type of application!

For the appropriate application a suitable tool!

Intel® Cluster OpenMP – Practical (on cacau.www.de)

Initialization
- cd ~/OpenMP/#nr/clomp/
- qsub -I -V -l nodes=2:mem1gb,walltime=00:30:00 -d `pwd`
  (interactive batch environment on 2 nodes for 30 min.)
- module switch compiler/intel9.0 compiler/intel9.1
- . init_clomp
  (setup of licenses and kmp_cluster.ini in current dir.)

Compilation
- Compiling the application with Intel® Cluster OpenMP:
  icc -cluster-openmp -o my_prog my_prog.c
  or
  ifort -cluster-openmp -o my_prog my_prog.f

Execution
- export OMP_NUM_THREADS=4
- ./my_prog
**Intel® Cluster OpenMP – Practical (on cacau.www.de)**

- **init_clomp** must be started in a batch environment:

```bash
# synopsis:
#  . init_clomp
export INTEL_LICENSE_FILE=/cacau/HLRS/hlrs/hpcintel/clomp_license.lic
export LD_LIBRARY_PATH=/cacau/HLRS/hlrs/hpcintel/workspace/32e:$LD_LIBRARY_PATH
echo "--processes=2"| sed -e "s/$,/'" | sed -e "s/,'/,'" > kmp_cluster.ini
\nln -f kmp_cluster.ini .kmp_cluster
export KMP_CLUSTER_PATH=`pwd`
ulimit -s unlimited
```

- It produces **kmp_cluster.ini** in the local working directory, e.g.,

```bash
--processes=3 --process_threads=2 --hostlist=noco023.nec,noco024.nec,noco025.nec
```

- and **.kmp_cluster** for access from everywhere via `$KMP_CLUSTER_PATH`

---

**Examples**

- cd exa1
  - No modifications
  - `icc -cluster-openmp -o exa1 c_ex1.c`
  - `ifort -cluster-openmp -o exa1 f_ex1.f`
  - `./exa1`
  - Checking the number of threads should be 4 (on 2 nodes with each 2 threads)

- cd exa2
  - Declaration of sharable variables is necessary
  - `icc -cluster-openmp -clomp -sharable -propagation -ipo -o exa2 c_ex2.c`
  - `ifort -cluster-openmp -clomp -sharable -propagation -ipo -o exa2 f_ex2.f`
  - There are variables used as shared, but not declared as sharable:
    - C: `#pragma intel omp sharable(xdim, ydim, n)` in main
    - `r_tmp2 = ... kmp_sharable_malloc(...);`
    - Fortran: `!dir$ omp sharable(xdim, ydim)` in module dims
    - `!dir$ omp sharable(n)` in the main program
  - `icc -cluster-openmp -o exa2 c_ex2.c`
  - `ifort -cluster-openmp -o exa2 f_ex2.f`
  - `./exa2`
Intel® Cluster OpenMP – Practical (on cacau.www.de)

**Examples**
- cd ../../exa3
  - Declaration of sharable variables is necessary
  - icc -cluster-openmp -clomp -sharable -propagation -ipo -o exa3 c_ex3.c
  - ifort -cluster-openmp -clomp -sharable -propagation -ipo -o exa3 f_ex3.f
- There are variables used as shared, but not declared as sharable:
  - C:
    - #pragma intel omp sharable(xdim, ydim, n)
    - r_tmp2 = ... kmp_sharable_malloc(...);
  - Fortran:
    - !dir$ omp sharable(niter, ntemp)
    - !dir$ omp sharable(matrices)
    - ntemp = niter-1
    - call run(ntemp)
- icc -cluster-openmp -o exa3 c_ex3.c
- ifort -cluster-openmp -o exa3 f_ex3.f
- ./exa3
- Now it should work

Intel® Cluster OpenMP – Practical (on cacau.www.de)

**Examples**
- cd heat
  - No modifications
  - icc -cluster-openmp -Dmax=3000 -Dkmax=3000 -Dimax=10 -o heat heats2_x.c
  - ifort -cluster-openmp -Dmax=3000 -Dkmax=3000 -Dimax=10 -o heat heats2_x.f
  - ./heat
Appendix

- Intel® Compilers with Cluster OpenMP – Consistency Protocol – Examples

Notation
- ..=A[i] Start/End: Start/end a read on element i on page A
- A[i]=.. Start/End: Start/end a write on element i on page A, trap to library
- Twin(A): Create a twin copy of page A
- WriteNotice(A): Send write notice for page A to other processors
- DiffReq_A_n(s:f): Request diffs for page A from node n between s and f
- Diff_A_n(s:f): Generate a diff for page A in writer n between s and f, where s and f are barrier times. This also frees the twin for page A.
Exa. 1

Node 0
Barrier 0
A[1]=.. Start
Twin(A)
Twin(A)
Barrier 1
WriteNotice(A)
Diffreq_A_1(0:1)>
<~Diff_A_1(0:1)
Apply diffs
Barrier 2
WriteNotice(A)
Barrier 0

Node 1

Barrier 0

Node 2

Barrier 0

Exa. 2

Node 0
Barrier 0
A[1]=.. Start
Twin(A)
A[1]=.. End
Barrier 1
WriteNotice(A)
A[3]=.. (no trap to lib)
Barrier 2
(No WriteNotice(A) required)
A[3]=.. (no trap to lib)

Diff_A_0(0:2)>
~A[1] Start
<~Diffreq_A_0(0:2)
Apply diffs
~A[1] End
Barrier 3
(no WriteNotice(A) required because diffs were sent after the A[3]=..)
A[1]=.. Start
Twin(A)
Barrier 4
WriteNotice(A)
~A[1] Start
<~Diffreq_A_0(0:4)
Create Diff_A_0(2:4) send Diff_A_O(0:4)>
Apply diffs
~A[1] End

Courtesy of J. Cownie, Intel
These examples may give an impression of the overhead induced by the Cluster OpenMP consistency protocol.