



Shared Memory Parallelism with OpenMP: A Brief Introduction

Martin Plummer

Computational Science and Engineering Department,
CLRC Daresbury Laboratory, Warrington, Cheshire, WA4 4AD.

Revision: 1.0

Abstract

OpenMP is a method of parallelizing Fortran and C/C++ codes on shared memory computers. Shared memory computer architectures allow several processors to access memory throughout the system efficiently. This is in contrast to distributed memory systems in which each processor has a localized memory and communication between processors must be imposed explicitly using a language such as MPI.

This is a Technical Note of the UKHEC Collaboration.

Report available from <http://www.ukhec.ac.uk/publications/notes/openmp.pdf>

© UKHEC 2002.

Neither the UKHEC Collaboration nor its members separately accept any responsibility for loss or damage arising from the use of information contained in any of their reports or in any communication about their tests or investigations.

In practice, OMP directives are 'easy' to add to a code, and essentially consist of pseudo-comment lines wrapped around local parallelizable sections of code. The compiler ignores these directives unless OpenMP is loaded. Shared memory parallelism uses the concept of 'threads' and the 'fork-join' model. The program begins as a single (master) thread and continues as such until a parallel construct is encountered. The required number of (slave) threads are then created and program statements within the construct are executed in parallel by each thread. At the end of the parallel construct the threads synchronise and only the master thread continues execution. This is in contrast to distributed memory systems in which serial parts of the code are executed independently by each processor.

The main use of OpenMP is to take advantage of the shared memory hardware architecture: implicit data movement, globally addressable 'cache-coherent' systems. From a programming point of view, serial codes do not have to be rewritten top-to-bottom to be usable. Compilers may provide some automatic parallelization of straightforwardly written code together with the use of efficiently parallelized library routines: the IBM -qsmp option and esslsmplib subroutine library are examples. Manually, sections of code may be parallelized incrementally with minimal alteration of structure. As the user becomes more confident in applying OpenMP directives the parallelization may be increased.

The disadvantage of this approach is that efficiency gains are generally lower than can be obtained from a properly written distributed memory code. Most obviously the unaltered serial 'grand structure' may not be appropriate for a parallel realisation. Each parallel construct requires setting up and shutting down of the parallel environment, plus definitions of which variables remain shared and which variables become temporarily private to each thread. The compiler may not recognise parts of the code as parallelizable. Many 'clever' intrinsic functions may not be recognised as parallelizable by the compiler and need to be expanded explicitly into parallel constructs. The piecemeal approach to parallelism may introduce errors and inconsistencies. For example, nested parallelism is allowed in OpenMP with strict rules, however some computers will not support it and revert to serial mode when it is encountered. The localised nature of parallel constructs in sections of code may inadvertently lead to nested parallelism, such as calling a parallelized library routine from within a parallel do-loop, or parallelizing an outer control loop which contains calls to routines which already contain parallel constructs. This may or may not be permissible.

In conclusion, production of a shared memory OpenMP code which is as efficient as a properly written distributed memory code will require at least as much work in rewriting the code as is required for the distributed memory code. However, proper use of parallelizing compiler options and shared memory specific subroutine libraries combined with judicious introduction of parallel constructs will allow the code to take reasonable advantage of the computer architecture for very little effort.

A Fortran Example

This calc_pi routine is used in the CSAR/MCC OpenMP training course.

```
program calc_pi
implicit none
integer, parameter :: wp = selected_real_kind (12)
real (wp) :: f, w, sum, pi, a, x
integer :: i, n

f(a) = 4.0_wp / (1.0_wp + a * a)

read (5,*) n
w = 1.0_wp / real(n, wp)
sum = 0.0_wp

!$OMP PARALLEL PRIVATE(x,i), SHARED(w,n) &
!$OMP REDUCTION(+:sum)
!$OMP DO
do i = 1, n
    x = w * (real(i, wp) - 0.5_wp)
    sum = sum + f(x)
end do
!$OMP END DO
!$OMP END PARALLEL

pi = w * sum
write (6,*) 'pi =', pi

stop
end program calc_pi
```

Using the Manchester CSAR SGI Origin 2000 computer 'fermat,' the MIPSpro 7 compiler command `f90 -mp -O3` and setting `n = 1,000,000,000`, execution time was 53s, 27s, 13s, 7s and 12s (each +/-1s) for 1,2,4,8 and 16 processors respectively. The `-mp` flag allows OpenMP commands to be interpreted. If the OpenMP commands are removed from the program and the flag is replaced by `-ap` (auto-parallelization) the same effect is achieved, however the `-ap` flag only works properly for `-O3`.

Useful Addresses:

<http://www.openmp.org/>

This site is the home of OpenMP, which provides the standard specification for OpenMP compiler directives, library routines and environment variables for shared memory parallelism. These have been agreed via collaborative work with interested parties from the hardware and software industries, and from government and academia. OpenMP specification is an informal agreement between vendors and users. The IBM SMP specification, for example, contains customised directives that do not conform to OpenMP in addition to standard directives. These are often less cumbersome than OpenMP directives but are not portable.

http://www.cs.man.ac.uk/cnc/staff/michael/ORIGIN/problems/OpenMP_compiler_problems.html

Michael Bane's site about issues with the SGI MIPSpro Fortran compiler when compiling OpenMP programs.