REPORTS
IN
INFORMATICS

ISSN 0333-3590

The Bergen Ocean Model Benchmark 1.0

Kenth Engø

REPORT NO 183 **February 2000**

Department of Informatics

UNIVERSITY OF BERGEN

Bergen, Norway

This report has URL
http://www.ii.uib.no/publikasjoner/texrap/pdf/2000-183.pdf

Reports in Informatics from Department of Informatics, University of Bergen, Norway, is available at
http://www.ii.uib.no/publikasjoner/texrap/.

Requests for paper copies of this report can be sent to:

Department of Informatics, University of Bergen, Høyteknologisenteret,
P.O. Box 7800, N-5020 Bergen, Norway

The Bergen Ocean Model Benchmark 1.0

Kenth Engø
Email: Kenth.Engo@ii.uib.no
WWW: http://www.ii.uib.no/~kenth/

Department of Informatics
University of Bergen
N–5020 Bergen
Norway

7th February 2000

Abstract

The Bergen ocean model benchmark (BOMB) 1.0 is an Open MP benchmark based on the Bergen ocean model (BOM) pioneered and developed by Berntsen, Skogen and Espelid [2, 1]. In this note we briefly report on the work undertaken to implement the Open MP directives into the BOM code, and problems related to this task. We also show some results from running the BOM benchmark on a Cray Origin 2000.

AMS Subject Classification: None

Key Words: Bergen ocean model, Open MP, Benchmark

1 Introduction

The Bergen ocean model benchmark was developed as part of a national collection of benchmarks for high performance computing. BOMB 1.0 is the Open MP benchmark in this family of HPC codes.

This note is organized as follows: In the next section we discuss the actual implementation of the Open MP directives in the code, and problems to watch out for while doing this. Test results from running BOMB on a Cray Origin are reported on in Section 3, and in Section 4 we have included the Readme and the script file from BOMB 1.0.

2 Implementation of the Open MP directives

Using optimization offered by the compiler will in many respects produce the necessary optimal code. In the work of inserting the Open MP (OMP) directives into the Bergen ocean model code some of the automatic optimization done by the compiler has to be done by hand.

Consider the following example from the routine `laxwuv3.f90`:

```
DO I = 2,IM
  DO J = 1,JM
    DO K = 1,KB
      DELTAI(I,J,K) = FIELDD(I,J,K)-FIELDD(I-1,J,K)
    END DO
  END DO
END DO
```

The loop indices are in the order `I,J,K` and optimizing using a compiler would result in a reordering of the loops in the order `K,J,I`. When inserting a compiler directive in front of the do-loops, the compiler is told that the outer of the nested loops does not contain any data dependencies and can be run in parallel. The compiler is not told to reorder the loops according to it's own taste, and then run the outer loop in parallel. There is no guarantee in general that the permuted loops will not contain any data dependencies.

Hence, in order to make a code that will execute fast, we must manually reorder the indices according to speed and check for data dependencies, and then insert the parallel directive. The above example would then take the following form:

```
!$OMP PARALLEL DO PRIVATE(K,J,I)
DO K = 1,KB
  DO J = 1,JM
    DO I = 2,IM
      DELTAI(I,J,K) = FIELDD(I,J,K)-FIELDD(I-1,J,K)
    END DO
  END DO
END DO
```

On the Cray Origin doing the above will substantially improve performance through more effective memory access and less cache trashing. In the Bergen ocean model code this was the largest source to improvement in execution speed.

Another optimization issue is loop fusion. There were not many identifiable instances of this type. But again, such an optimization has to be done by hand. E.g. consider the following code from `fbcor2d.f90`:

```
UA = 0.
```

```
DO K = 1,KB-1
  DO J = 1,JM
    DO I = 1,IM
      UA(I,J) = UA(I,J) + DUM(I,J)*U(I,J,K)*DZ(K)
    END DO
  END DO
END DO
!
VA = 0.
DO K = 1,KB-1
  DO J = 1,JM
    DO I = 1,IM
      VA(I,J) = VA(I,J) + DVM(I,J)*V(I,J,K)*DZ(K)
    END DO
  END DO
END DO
```

Inserting one OMP-directive in front of each of the two above loops would not constitute the best solution. The reason is that you only guarantee data independencies in a parallel execution, if the two loops are run in two sequential steps. But, as we can see, the two loops can be joined into one – which the compiler will not – and the optimal code should look like this:

```
UA = 0.
VA = 0.
!$OMP PARALLEL DO PRIVATE(I,J,K)
DO K = 1,KB-1
  DO J = 1,JM
    DO I = 1,IM
      VA(I,J) = VA(I,J) + DVM(I,J)*V(I,J,K)*DZ(K)
      UA(I,J) = UA(I,J) + DUM(I,J)*U(I,J,K)*DZ(K)
    END DO
  END DO
END DO
```

The above two instances were the only hands-on type optimization that was done. We also tested the benchmark with various scheduling, but the conclusion from a few test runs was no noticeable improvement. We did not consider the effects of e.g. array padding.

3 Test runs on a Cray Origin 2000

Several test runs were performed on a Cray Origin 2000. The same model was run for two different sizes and two different time steps, and each of the four test cases were looping through

$1, 2, \ldots, 15, 16$ threads. Minor modifications where done to the script `bomb.csh`, see Section 4.3, to accomplish this. The size of the model is set in `state.f90`, and we ran the benchmark with problem size $42 \times 42 \times 29$ and $62 \times 62 \times 29$. In the file `Hoved.f90` you can change the time steps by changing the variable `NDIVIS`. The numerical experiments were done using `NDIVIS=24,32`, and the larger the value, the lesser the time step.

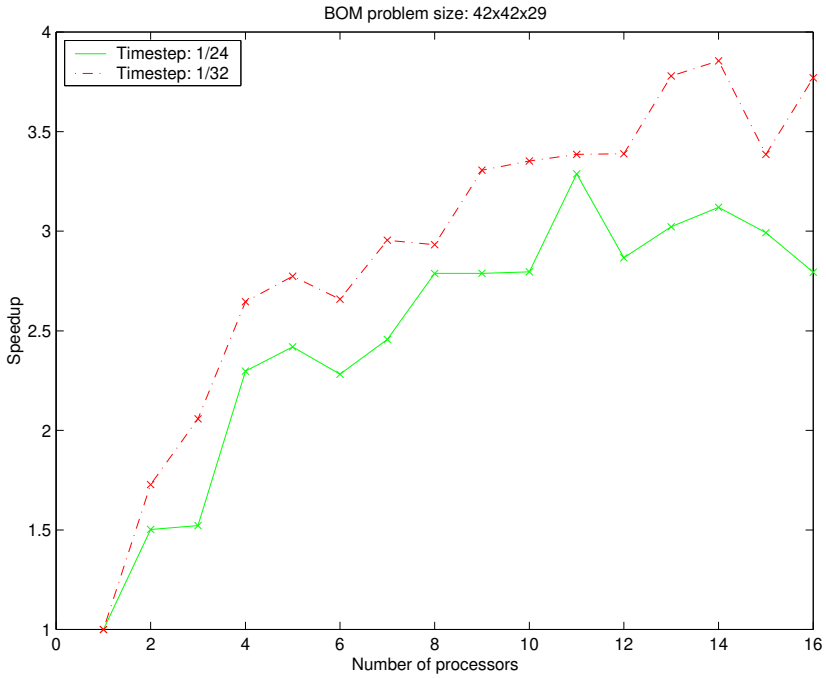

Figure 1: Speedup for model $42 \times 42 \times 29$ with two different time steps.

Figure 1 shows the resulting speedup from the two runs with the $42 \times 42 \times 29$ model. The obtained speedup is not at all of any high merits. There may be different reasons for this, e.g. memory access and overhead because of the creation and administration of the parallel threads. It should be mentioned that the tests were run in a multi user mode. Hence, the benchmark had to compete for the resources with other users of the computer, along with varying load and communication patterns. Thus, the plots presented in Figures 1 and 2 do not represent results obtained from an *ideal benchmark run*.

In Figure 2 we have run the larger $62 \times 62 \times 29$ model. Compared to the results in Figure 1 the speedup is better, as one could expect since the difference between the two cases is an increase in the horizontal space resolution.

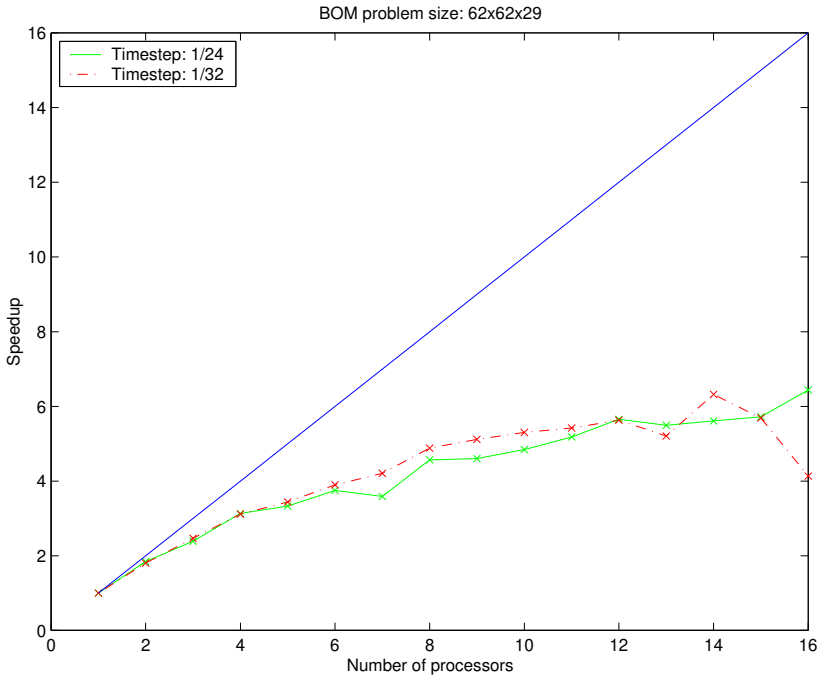

Figure 2: Speedup for model $62 \times 62 \times 29$ with two different time steps.

4 BOMB 1.0

4.1 Where to down-load BOMB 1.0?

The benchmark is freely available from the URL `http://www.ii.uib.no/bomb/`. From this site you can down-load a tar-file containing the benchmark along with instructions on what to do with the tar-file.

The initial configuration of the benchmark is the following: The stepsize is $1/16$, that is; `NDIVIS` $= 16$. The space resolution is set to $42 \times 42 \times 29$. Further, the benchmark script is initially set up to loop through $1, 2, 4$, and 8 threads. The running time on a Cray Origin with this configuration is effectively one hour, and recorded speedup is approximately $1, 1.3, 2.6$ and 2.4.

4.2 Readme file

```
The Bergen Ocean Model Benchmark (BOMB), version 1.0 (2000.02.03)
-----------------------------------------------------------------

                WWW  : http://www.ii.uib.no/bomb/
                ---------------------------------
```

How-to BOMB:

Follow these steps in order to prepare the script for execution on your system:

1) Check the first line in 'Bom/bomb.csh' for the correct csh-pathname. If editing is necessary, also edit the first line of 'Bom/sourcecode/make_bomb'. (Issue the command 'which csh' to find the correct path.)
2) Check the compiler variable 'F90'. Default value: 'f90'.
3) Check the remove variable 'RM'. Default value: '/sbin/rm'
4) Set the maximum number of threads by editing the variable 'MAXNUMTHREADS'. Default value is '8'.
5) The script 'Bom/bomb.csh' is written with the assumption that the environment variable controlling the number of threads is called 'MP_SET_NUMTHREADS'. If this is not the case on your system, change the line ' setenv MP_SET_NUMTHREADS $num' in the script accordingly.
6) The script variables 'F90FLAGS' and 'LFLAGS' are system specific (tailored for the Origin 2000 in Bergen), and should be changed to suit your platform.
7) Issue the command 'bomb.csh' to run the benchmark script. Each job is submitted to the queue system with a command looking like:
   bsub -S 2000000 -I -q normal -n $num eval 'time ./bom > bom.output '
   Remove/edit these two lines in the script according to your needs; i.e. jobs to the batch queue are submitted differently, or you do not want to run the jobs in the batch system.
8) When finished, check that output in the directory './executable' is similar to the 'reference solution' in the directory './refsol'.

With the original configuration of the bomb.csh-script, i.e. looping through 1,2,4, and 8 threads, model 42x42x29, and NDVIVS = 24; the running time on the Cray Origin in Bergen was approximately 1 hour.

Directory structure:

BOM_Benchmark_1_0/	executable/	Location of executable and output from executable.
	output/	Output from script and compilation
	output/origin	Files from a test run on Origin.
	refsol/	Location of reference solution.
	sourcecode/	Location of source code.

Output from script:

In the directory './output' you will find two log-files:
 - from running the script 'bomb.csh' : '.../output/bomb_script.log'
 - from the compilation : '.../output/bomb_compiling.log'

```
NB!NB!NB!NB!:
On origin the time function used in the script adds all execution
times for the individual threads and displays the total. Hence, to
find the true execution time divide the displayed time by the number
of threads. If you experience linear speed-up, the displayed time
should be the same as the time for running the code on only one
thread. This might be system specific, hence, it should be checked.

After the script has finished view the first of the above log-files in
order to check execution. In case the script aborts because of
non-existing executable 'bom', consult the second log-file for
compilation errors. The directory '.../output/origin/' contains two files
'bomb_compiling.log' and 'bomb_script.log', which are sample output
files from the compilation, and running the script on the Origin 2000.


More information on the Bergen Ocean Model:

The Bergen Ocean Model is a set of Fortran files originally developed
by Jarle Berntsen, Department of Mathematics, University of Bergen,
Norway, email: Jarle.Berntsen@mi.uib.no. The version of the code used
in the present benchmark was supplied by Helge Avlesen, Department of
Informatics, University of Bergen, Norway, email: Helge.Avlesen@ii.uib.no.

NB!! *ALL* QUESTIONS AND SPECIFICS RELATING TO THE CODE ITSELF SHOULD BE
DIRECTED TO EITHER JARLE BERNTSEN OR HELGE AVLESEN.

On the other hand, queries regarding the BOM benchmark 1.0 should be
directed to:

Kenth Engø
Department of Informatics
University of Bergen
P.B. 7800
N-5020 Bergen
Norway.

Email: Kenth.Engo@ii.uib.no
Phone: +47 55 58 41 95.
WWW  : http://www.ii.uib.no/~kenth/
```

4.3 Script file

```
#!/sbin/csh -f

echo " "
echo " Running the BOM Benchmark 1.0"
echo " "
echo " See: './output/bomb_script.log' for details on benchmark run."
```

```
echo " "

setenv F90            "f90"        # Fortran 90 compiler
setenv RM             "/sbin/rm"   # rm
setenv MAXNUMTHREADS  8            # Max number of threads.
setenv PROG           "../executable/bom" # Location of 'bom' (relative to source).
setenv F90FLAGS       "-apo -mips4 -OPT:roundoff=0 -O3"  # Compiler flags
setenv LFLAGS         "-apo -mips4 -OPT:roundoff=0 -O3"  # Linking flags

cd executable
$RM -f *

cd ../sourcecode
$RM -f -r rii_files *.o

set logfile = "../output/bomb_script.log"

echo "                      " >&! $logfile
echo "The BOM Benchmark 1.0 " >> $logfile
echo "                      " >> $logfile
eval 'date'                   >> $logfile
echo "                      " >> $logfile
echo "The following rm command is being used   :" $RM            >> $logfile
echo "The following F90 compiler is being used :" $F90           >> $logfile
echo "The max. number of parallel threads are  :" $MAXNUMTHREADS >> $logfile
echo "The executable is located (rel. 2 source):" $PROG          >> $logfile
echo "Complier flags                           :" $F90FLAGS      >> $logfile
echo "Linking flags                            :" $LFLAGS        >> $logfile
echo "                      " >> $logfile
echo "Start compilation:    " >> $logfile
eval 'time ./make_bomb '      >> $logfile    # Script running make
echo "... finished          " >> $logfile
echo "                      " >> $logfile
echo "See: 'Bom/output/bomb_compiling.log' for possible compilation errors. " >> $logfile
echo "                      " >> $logfile

cd ../executable

echo "Starting ...     "   >> $logfile

set num = 1
while ($num <= $MAXNUMTHREADS)
  setenv MP_SET_NUMTHREADS $num
  echo "                       "   >> $logfile
  echo "Number of threads : $MP_SET_NUMTHREADS" >> $logfile

  if ($num <= 4) then
    bsub -S 2000000 -I -q normal -n $num eval 'time ./bom > bom.output ' >> $logfile
  else
    bsub -S 2000000 -I -q parallell -n $num eval 'time ./bom > bom.output ' >> $logfile
  endif
```

```
  mv tab.dat tab$num.dat
  mv bom.output bom$num.output
  mv bom.dat bom$num.dat

  @ num = $num + $num
end

echo "Finished running the BOM benchmark. "        >> $logfile
echo "                                    "        >> $logfile

echo " Finished."
```


Acknowledgment

I want to thank Tor Sørevik for his great help during the work with BOMB.

References

[1] J. BERNTSEN, *A sensitivity study of a baroclinic model for the North Sea with focus on the Utsira-Orkneys transect*, Fisken og Havet 16, Institute of Marine Research, Bergen, Norway, 1998.

[2] J. BERNTSEN, M. D. SKOGEN, AND T. O. ESPELID, *Description of a σ-coordinate ocean model*, Fisken og Havet 12, Institute of Marine Research, Bergen, Norway, 1996.

[3] BOMB, *Bergen 0cean Model Benchmark 1.0*, February 2000. http://www.ii.uib.no/bomb/.

[4] OPENMP ARCHITECTURE REVIEW BOARD, *OpenMP Fortran Application Program Interface*. Version 1.0, http://www.openmp.org/, October 1997.