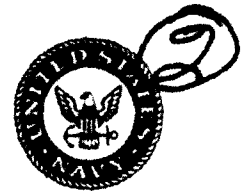


Naval Research Laboratory

Washington, DC 20375-5320

AD-A269 876



NRL/MR/5581--93-7400

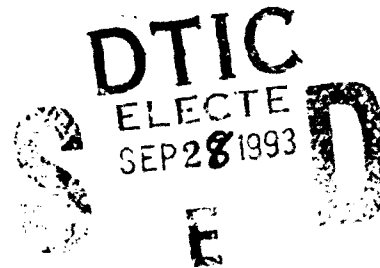
Connection Machine Software Conversion of a Navy Oceans Model

P. B. ANDERSON

M. A. YOUNG

*Advanced Information Technology Branch
Information Technology Division*

September 13, 1993



Approved for public release; distribution unlimited.

93-22369



REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1216 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 13, 1993	3. REPORT TYPE AND DATES COVERED		
4. TITLE AND SUBTITLE Connection Machine Software Conversion of a Navy Oceans Model		5. FUNDING NUMBERS PE - 622334N PR - RS34-J77		
6. AUTHOR(S) Paul B. Anderson and Michael A. Young				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Research Laboratory Washington, DC 20375-5320		8. PERFORMING ORGANIZATION REPORT NUMBER NRL/MR/5581-93-7400		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) The introduction of highly parallel machines with peak performance significantly exceeding the Cray machines has sparked interest in running scientific models on these new architectures. This report describes a software conversion, of the Navy model called OCEANS, starting with a Cray Y-MP/8 version in Fortran 77 and ending with a Fortran 90 version for the Connection Machine CM-200. Data mapping, conversion planning, and performance points of view are considered.				
14. SUBJECT TERMS Connection machine Fortran 90 Ocean modeling		15. NUMBER OF PAGES 16		
		16. PRICE CODE		
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

CONTENTS

1	INTRODUCTION	1
2	BACKGROUND	1
3	DATA LAYOUT	2
4	SOFTWARE STRUCTURE	2
5	CONVERSION PLAN AND RESULTS	8
6	PERFORMANCE MEASUREMENTS	9
7	CONCLUSIONS	11
8	ACKNOWLEDGMENTS	12
9	REFERENCES	12

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

DTIC QUALITY INSPECTED 3

List of Figures

1	Call Tree of Original Code—Part 1	4
2	Call Tree of Original Code—Part 2	5
3	Call Tree of Original Code—Part 3	6
4	Call Tree of Original Code—Part 4	7
5	Convex Timings—Original Model	9
6	Initial CM-200 Timings	10
7	Optimized CM-200 Timings	11

Connection Machine Software Conversion of a Navy Oceans Model

1 INTRODUCTION

Models which solve a set of partial differential equations form a large and important category of scientific applications. These applications are commonly structured to run well on vectorizing machines such as the Cray X-MP, Cray Y-MP, and the Convex C series.

The introduction of highly parallel machines [1, 2] with peak performance significantly exceeding the Cray machines has sparked interest in running scientific models on these new machine architectures. The demonstration over the past few years of many models restructured successfully for these machines has led to growing interest in code conversion. This is in part due to the widespread belief that economic factors, principally the leveraging of commodity microprocessor and memory technology, will make highly parallel machines more cost-effective than vector architectures.

This document describes one such conversion, of the Navy model called OCEANS, starting with a Cray Y-MP/8 version in Fortran 77 and ending with a Fortran 90 [4] version for the Connection Machine CM-200. Data mapping, conversion planning, and performance points of view are considered.

2 BACKGROUND

There are at least three basic solution methods for sets of partial differential equations. The method of finite differences uses the definition of a derivative to obtain approximations for rates of change of various quantities. Finite differencing is the most commonly employed method and has the most elementary theory.

An alternative method takes an approach using Fourier transforms. The primary advantage of a spectral method is that derivatives have simple forms and the Fast Fourier Transform algorithm makes conversion from spectral variables to grid variables relatively inexpensive.

Another approach in common use is the method of finite elements. This method seeks coefficients for functions which make up the desired solution. It can be thought of as a generalization of the spectral method.

Models may employ combinations of these methods. Spectral models in the horizontal may use finite differencing techniques in the vertical.

The model to be converted is a finite difference model in the horizontal and employs a specialized technique using vertical "layers" which is distantly related to both spectral and finite element approaches. The use of 6 layers provides adequate resolution and predictive power for models of basin areas up to global models.

3 DATA LAYOUT

A number of data layouts are possible but finite difference models of ocean or atmospheric variables tend to have simple mappings to the CM-200. Finite difference primitives are typically implemented using nearest neighbor communications, the most efficient communications primitives for the Connection Machine. The model itself will determine whether systems of simultaneous equations must be solved and this can affect the choice of memory layout.

A goal of the conversion was to produce a code suitable for use as a benchmark. Toward this end, it was decided to use the slicewise data model which is more flexible than the older PARIS model and is more consistent with the Connection Machine CM-5. In the slicewise model, array shapes should generally be a multiple of 4 in each axis. The precise rules are complex but slightly more lenient.

Most arrays of the model have a shape of (2163, 1041, 5, 2) where the first two axes have horizontal extent, the third axis is over the vertical layers, and the last axis indexes current and prior timestep values. Three of the positions in the first axis and one of the positions in the second axis contain copies of other elements so that shifted references can be made without conditional tests. This is an aid to vectorization but is a detriment to the CM-200 for two reasons. First, the communication operations necessary to maintain the extra columns are very expensive. Second, the resulting shape is not an efficient one for the CM-200.

Like the atmosphere, ocean models tend to be more computationally complex along vertical columns than in horizontal directions. It is therefore common that vertical columns be allocated completely within a single processor. This course of action was taken in the model conversion. For the other two axes, initial implementations were made with the axes unchanged. A later implementation with more CM-200 compatible dimensions was also undertaken.

4 SOFTWARE STRUCTURE

The software structure in the original model is well organized compared to many other existing Fortran 77 codes. The bulk of the computation is contained in a series of small, regular routines beginning with the leading characters sh, mh, and ch. Several sets of support routines for initialization, history output, land masking, and miscellaneous functions are also included. The land masking routines are the most significant among these non-computational routines since they involve complicated operations at each time step.

The main program is large and serves the function of hiding the layer structure from most routines. Low level computation routines see only two dimensional slices from truly 4 dimensional

arrays. The ordering of subscripts is important in taking the two dimensional array slices. In a non-distributed memory setting, the first two axes are contiguous for a given layer and timestep. This means that isolating a slice involves no data motion but simply a subscript calculation. The first element of a 2 dimensional slice for layer k and timestep 2 is at position $(1, 1, k, 2)$ in the array. Since Fortran 77 does not distinguish in parameter passing between an array and its first element, a contiguous subarray has effectively and dynamically been equivalenced. This practice is common in Fortran 77 codes. Unfortunately, in a distributed memory setting such as the CM-200 implicit equivalence techniques do not work or do not always work.

Current compiler restrictions have also been a factor in subscript order. Present CM Fortran compilers require that all serial axes precede the parallel one. This restriction is contrary to the normal, and most consistent usage patterns. To satisfy these restrictions, the subscripts of the main arrays in the model were reordered to place the last two axes, layer and timestep, first. In the present compiler, this results in efficient passing of two-dimensional horizontal sections to the low level computation routines. It is not known, however, whether future compilers will continue to generate the best code for this case.

The annotated indented call structure is shown in Figure 1. Note that a routine is listed more than once if calls appear in the program text in more than one place and that its entire subtree is reproduced at each call.

Call Tree	Function
oceans	driver
· aremh5	explicit hydrodynamic ocean model
· · xxinit	read in namelist
· · xhinit	calculates g-prime related constants - hydrodynamic models
· · xpxori	read in (via namelist) inflow values and outflow port friction and initialize inflow profile(s)
· · xxtopo	reads in topography (hd and htot) from data file and generates hb
· · · xxrang	fixes indexes to elements of a1 delimiting its range of values
· · xxlndi	setup data structures describing region
· · · zlidxi	setup index vectors pointing to 'boundary' nodes
· · · zllndi	set up line-based representations of land areas
· · · · xxmapp	outputs integer map of a1
· · · zlloop	initialize j-dependent 'safe' i-loop ranges, and the 'safe' envelope for history file fields
· · xxcset	matrix initialization by constant a1 = const
· · xxlndh	set unwanted nodes of h to 'flag' (i.e. all land nodes)
· · · zllndh	sets land points in h to 'flag'
· · xxmass	sum hnew correcting for 'spherical' gravity
· · xpxorh	tests position of ports and marks them in hnew
· · xxmapp	outputs integer map of a1
· · xxawnd	user supplied analytic wind forcing
· · · xxwprt	prints windstress fields
· · xxwndi	initialize winds at day wstart of first wind file
· · · xxwprt	prints windstress fields
· · xhhisi	initialize h,u,v at t = tstart from (input) history file

Fig. 1 — Call Tree of Original Code—Part 1

Call Tree	Function
· · xporj	complete the initialization of inflow ports, by calculating existing transport, velocity, and direction at these ports
· · xxtimi	initialize time series (work done by user supplied routines)
· · xhgrai	initialize output files for $t = t_{start}$
· · xxtimx	take a sample for one or more time series
· · xhgrat	outputs one layer of graphics at day t_{day}
· · xxsavh	saving and restoring hnd in predictor-corrector timestep
· · xxcopy	matrix copy $a1 = a2$
· · xxpors	save u,v,h outflow values from 1 and 2 time steps back
· · xxpore	save outflow values from what will be 3 time steps back
· · xxstrt	reset time step dependent constants to reflect the new dt - only needed when changing integration step type
· · xxstra	reset dependent constants to reflect the new a and dt
· · mhvel0	form velocity components from transports
· · xxlndv	set boundary conditions for velocity components uv and vv
· · · zlidxz	sets the zero boundary values for u or v
· · · zlidxn	sets the non-zero boundary values for u or v
· · xporv	set velocities at all ports
· · xzdiff	matrix subtraction $a1 = b1 - b2$
· · mhitx3	u momentum equation - interfacial stress term special cases

Fig. 2 — Call Tree of Original Code—Part 2

Call Tree	Function
. . mhity3	v momentum equation - interfacial stress term special cases
. . chmud0	u momentum equation - diffusion and dissipation terms
. . mhwtx0	u momentum equation - wind stress term
. . chmua0	momentum equation - advective terms
. . mhmuc0	u momentum equation - coriolis terms
. . shmud0	u momentum equation - diffusion and dissipation terms
. . shmua0	u momentum equation - advective terms
. . chmvd0	v momentum equation - diffusion and dissipation terms
. . mhwtv0	v momentum equation - wind stress term
. . chmva0	v momentum equation - advective terms
. . mhmvc0	v momentum equation - coriolis terms
. . shmvd0	v momentum equation - diffusion and dissipation terms
. . xxmult	1 layer of a0 = sum (k=1,kh) of sk * ak - used in calculating pressures and the lagrange multiplier transformation
. . chmup0	u momentum equation - pressure gradient term
. . mhmvp0	v momentum equation - pressure gradient term
. . shmup0	u momentum equation - pressure gradient term
. . chcnt0	continuity equation
. . shcnt0	continuity equation
. . xxadd0	matrix addition a1 = b1 + b2
. . mhity2	u momentum equation - interfacial stress term
. . mhity2	v momentum equation - interfacial stress term
. . xxlndt	boundary conditions for u and v and h if periodic
. . . zlidxz	sets the zero boundary values for u or v
. . . zlidxn	sets the non-zero boundary values for u or v
. . xxport	find transports at all ports
. . xxlndf	set unwanted nodes of u and v to 'flag' (i.e. all land nodes not next to a sea node)
. . . zllndt	sets land points in u and v to 'flag'

Fig. 3 — Call Tree of Original Code—Part 3

Call Tree	Function
· · xxavrg	elemental matrix average $a1 = (a1 + a2)/n$
· · xxwndr	read next wind input and update txf tyf
· · xxspot	saves one point of h and tx for printing every 12 timesteps
· · xxlndv	set boundary conditions for velocity components uv and vv
· · · zlidxz	sets the zero boundary values for u or v
· · · zlidxn	sets the non-zero boundary values for u or v
· · shmua0	u momentum equation - advective terms
· · shmva0	v momentum equation - advective terms
· · xhavew	writes out unlnd,vnld,hnld fields to scratch file
· · xhaver	average 'navrg' time levels reads unlnd,vnld,hnld from scratch file
· · xxwndm	roll back tx and ty 'kroll' timesteps
· · xhhisw	write one layer to history file at time tday
· · xxwndp	printout wind file statistics (only called after x*hisw)
· · xxwpvt	prints windstress fields
· · xhpout	prints out velocity and h fields (and mass balance) for layer
· · · mhvel0	form velocity components from transports
· · · xxlndv	set boundary conditions for velocity components uv and vv
· · · · zlidxz	sets the zero boundary values for u or v
· · · · zlidxn	sets the non-zero boundary values for u or v
· · · xxpov	set velocities at all ports
· · · xxlndf	set unwanted nodes of u and v to 'flag' (i.e. all land nodes not next to a sea node)
· · · · zllndt	sets land points in u and v to 'flag'
· · · · xxlndh	set unwanted nodes of h to 'flag' (i.e. all land nodes)
· · · · zllndh	sets land points in h to 'flag'
· · · xxfprt	prints the field a1 and mesh coordinates in degrees
· · · xxmass	sum hnew correcting for 'spherical' gravity

Fig. 4 — Call Tree of Original Code—Part 4

5 CONVERSION PLAN AND RESULTS

The memory layout of the basic history variables was systematically changed for the CM-200, moving the last two arguments to the beginning of the subscript list. These two axes were also declared to be serial, meaning that they were allocated within a processor. In this way 2 dimensional slices can be extracted without data motion as in the Fortran 77 case. The placement of serial axes at the beginning of the subscript list is a current CM-200 software limitation.

The regular software structure of the original model was used as the basis of the conversion. The module groups were converted and tested individually using a driver and simulated data for each group. The target routines have very little conditionalization but multiple tests were performed to the extent possible for alternative paths.

Each driver used a random number generator to provide simulated data inputs. This generator is a standard one that does not rely on knowledge of the binary representation of floating point values. It is therefore transportable and generates the same stream of numbers on a Convex C220, where the original Fortran 77 code was run, and on the CM-200. This allowed answers from the Convex to be compared directly with CM-200 results.

Since entire arrays of data were generated, a statistical approach was taken to verification. A set of 6 numbers, the minimum, maximum, L2 norm, sum, mean, and sample variance were generated for the inputs and outputs of each routine. Statistics from the Convex were automatically checked against CM-200 statistics using a program. Agreement to 10 significant figures was obtained although problems were encountered deciding when to employ relative error tests versus absolute error tests.

The above approach allowed the basic computation routines to be tested to a high degree of confidence. The same approach was also used with some of the support routines although their functions did not always fit the same pattern as the computational routines.

An exception to the pattern was a set of routines to handle topography and boundaries. This code also included managing the edges of the horizontal extents as mentioned previously. In this group of routines, an almost complete rewrite was necessary. The new method employs masks to identify the elements requiring specialized boundary processing. The input of topography data, which originally was done serially, became a serious performance problem in the large sizes. A one-time transfer to the DataVault was performed, allowing a fast read directly to the Connection Machine.

By far the most difficult conversion problems encountered came during integration of the routines. Since the basic functionality of the routines had already been established to a high confidence level, the problems were known to lie in the main routine or in the interface between the main and subordinate levels. Most problems were traced to layout differences between arrays declared in the main routine and passed to a subroutine and the subroutine formal parameter declarations. There were no available mechanisms at the time to detect these errors except direct visual checking.

One method, use of interface blocks, has since emerged as a way to detect these mismatches. A degree of checking is present in the current compiler version but the reliability and thoroughness is not clear.

Another possible method is a standalone tool to either generate interface blocks or produce a

listing of calls, actual arguments, subroutine declarations and formal arguments. Either of these approaches could result in a tool with high value for similar software conversions and even for new code development.

6 PERFORMANCE MEASUREMENTS

The original benchmark program has a very large memory space, amounting to 95 million words on a Cray Y-MP/8. This size is too large to run on the Convex and also too large to run on 8K CM-200 processors. A subsampled version with about 1% of the total grid points was used for software conversion and most testing.

Figure 5 shows a breakdown of timing into the more significant routines. The primary observation to make from this figure is that the bulk of the processing takes place in the *sh* and *mh* routines, together accounting for 86% of the total run time. Of these, the *sh* routines are the more important, containing 56% of the total run time.

The distribution of times would be expected to shift in a CM-200 version and this is indeed the effect seen. The primary reason for the shift is that the Convex (and the Cray) timings are dominated by the cost of the floating point operations whereas the CM-200 timings are dominated by the communications operations.

Percent	Cumulative	Routine(s)
12	12	Main
17	29	shmua0
16	45	shmva0
10	55	shmud0
10	65	shmvd0
3	68	shcnt0
4	72	shmup0
8	80	mhvel0
5	85	mhmvc0
5	90	mhmuc0
4	94	mhmvp0
2	96	mhitx3
2	98	mhity3
0	98	mhwtx0
0	98	mhwtly0
2	100	ln group
0	100	xx group
0	100	xh group

Fig. 5 — Convex Timings—Original Model

The performance of the initial CM-200 version is shown in Figure 6. The distribution of times is slightly different than for the Convex but *sh* and *mh* routines still account for 86% of the total run time and *sh* routines make up 63% of the total. The actual run time on 8K processors is approximately half that of the Convex, a rate that previously had represented about half of the speed of the Cray X-MP.

Percent	Cumulative	Routine(s)
4	4	Main
14	18	shmua0
15	33	shmva0
13	46	shmud0
13	59	shmvd0
8	67	shcnt0
3	70	shmup0
2	72	mhvel0
4	76	mhmvc0
4	80	mhmuc0
2	82	mhmvp0
0	82	mhitx3
0	82	mhity3
3	85	mhwtx0
3	88	mhwty0
8	96	ln group
4	100	xx group
0	100	xh group

Fig. 6 — Initial CM-200 Timings

During conversion, several Fortran 90 spread operations were introduced in the `sh` and `mh` routines. While it was known that these would be expensive, they were used so that a working version could be obtained as directly as possible to act as a starting point for future optimized versions.

The spread arrays are related to the discretization of the grid on the surface of the earth and are invariant once computed during initialization. As a second version, the spreads were moved to the main routine where they could be computed once and reused.

In addition to removing spreads, certain shifted quantities which were reused were also moved from the computational routines into main. These quantities change during the time stepping and hence the shifts must be periodically redone. Nevertheless, a net savings is generated since the cost of communications operations is high. The results of the first optimized version are shown in Figure 7. Now the distribution of times is quite different with 49% to `sh`, 13% to `mh`, and 34% of the run time attributed to remaining routines.

In the original Cray version, no calculations were performed over land, giving about a 20% speed increase on the Cray. The conditionalization required for the CM-200 does not result in a speed improvement because of the SIMD nature of the machine and in fact reduces performance. Allowing calculations over land yielded a code which was slightly faster and extremely uncluttered. A final phase of land masking at the end of each timestep was retained.

In absolute performance terms, this version is about twice as fast as before and about 4 times faster than the Convex version.

One final version was produced with assistance from the primary code maintainer, Alan Wallcraft of NRL Stennis. This version eliminates the overlapping rows and columns of the basic

Percent	Cumulative	Routine(s)
6	6	Main
10	16	shmua0
12	28	shmva0
11	39	shmud0
11	50	shmvd0
5	55	shcnt0
4	59	shmup0
4	63	mhvel0
2	65	mhmvc0
2	67	mhmuc0
3	70	mhmvp0
0	70	mhitx3
0	70	mhity3
1	71	mhwtx0
1	72	mhwty0
16	88	ln group
11	99	xx group
1	100	xh group

Fig. 7 — Optimised CM-200 Timings

2 dimensional shape. Fortran 90 cshift and eoshift operations were used to accomplish the same purpose as the extra rows and columns. Also, the (2163,1041) dimensions were altered to (2048,1152) to be more efficient on the Connection Machine. The first axis, representing the east/west direction, provides best performance for the cshift operation since it is a power of 2.

The resulting code is not only simpler but much more efficient. The run time from this version now appears to eclipse the Cray Y-MP/1 performance on 8K CM-200 processors. The performance ratio is approximately 2 to 1 so that a full 64K CM-200 would be faster than a Y-MP/8. This has not been tested so the claim must be taken as tentative. Performance of the new Cray Y-MP C90 is about three times that of previous Y-MPs per processor and the maximum number of processors has doubled to 16. A comparison between a 64K Connection Machine CM-200 and a Y-MP/16 C90 would probably favor the C90.

7 CONCLUSIONS

The code conversion of the model was straightforward. A basic conversion incorporating no substantive changes to the array shapes was produced. The performance of this version was approximately half that of the Cray, primarily because of inefficient array shapes.

An improved version with more efficient array shapes was then produced with the help of the code maintainer. The new version had greatly improved performance, significantly better than the Cray Y-MP.

It is worth noting that efficient array shapes on the Connection Machine are very unlikely to be found in existing Cray codes. This is due to the phenomenon of memory bank conflicts in past Cray architectures when array sizes are powers of two or contain powers of two. Cray codes have,

in fact, frequently been altered to produce just the odd-sized array dimensions encountered in the Fortran 77 version of the code to avoid memory bank conflicts.

The code was converted for the Connection Machine CM-200 under the *slicewise* model. Furthermore, strictly CM-200 optimizations were not incorporated since an easy port to the Connection Machine CM-5 was desired. The next step in the conversion process is to run the CM-200 code on a CM-5 equipped with vector units. Performance there is expected to be significantly better than on the CM-200, perhaps an order of magnitude better.

8 ACKNOWLEDGMENTS

The authors wish to thank Alan Wallcraft for his assistance with the conversion of OCEANS. We also thank Dr. Henry Dardy of the NRL Connection Machine Facility and Elizabeth Wald of the Office of Naval Technology (now part of Office of Naval Research) for their support.

9 REFERENCES

1. "Connection Machine Model CM-200 Technical Summary," Thinking Machines Corporation, June 1991.
2. "Connection Machine Model CM-5 Technical Summary," Thinking Machines Corporation, Jan. 1992.
3. "CMSSL for CM Fortran," Thinking Machines Corporation, Jan. 1992.
4. "American National Standards for Information Systems Programming Language Fortran," American National Standards Institute (in press).