(1)

aval Research Laboratory
ıshington, DC 20375-5320

NRL/MR/5590--94-7494

Connection Machine Software Conversion of the Navy TOPS Model

P. B. Anderson
Michael A. Young

Center for Computational Science
Information Technology Division

July 5, 1994

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE July 5, 1994	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE Connection Machine Software Conversion of the Navy TOPS Model	5. FUNDING NUMBERS PE-62234N 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/5590--94-7494
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research 800 North Quincy Street Arlington, VA 222-5660	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 TOPS, starting with a Cray Y-MP/8 version in Fortran 77 and ending with a Fortran 90 version for the Connection Machine CM-5. Data mapping, conversion planning, and performance points of view are considered.

14. SUBJECT TERMS Connection Machine Fortran 90 Ocean Modeling	15. Number of Pages 23
	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

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std 239-18
298-102

CONTENTS

List of Figures

CONNECTION MACHINE SOFTWARE CONVERSION OF THE NAVY TOPS 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 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.

Many existing codes reflect decades of optimization for sequential processing. Achieving maximum parallelism from such a starting point can be difficult. An attractive alternative to code conversion is development of a new model from the basic mathematical formulation. This is appealing as a way to build a model taking maximum advantage of the machine potential, both from a coding and an algorithmic point of view. It also affords an opportunity to revise the formulation to incorporate newer ideas or to rectify deficiencies.

Code conversion was chosen rather than re-development for two reasons. Firstly, the cost of conversion is lower than the cost of re-development. Previous experiences with similar models has clearly shown that the conversion cost is not typically high. The mapping of a finite difference grid point model to an MPP is not especially challenging. Secondly, an existing model often has a long history of operational use or has been comprehensively validated at some point. The operational history or validation gives operational confidence in the existing model predictions which a new model would lack.

This document describes the conversion of the Navy model called TOPS, the Thermodynamic Oceanographic Prediction System. The conversion started with a serial version in Fortran 77 and ended with a Fortran 90 [4] version for the Connection Machine CM-5. Data mapping, conversion planning, and performance points of view are considered.

Manuscript approved April 21, 1994.

2 BACKGROUND

This conversion effort is part of a series of conversions for ocean and atmospheric models[6, 7]. The approach taken for TOPS is similar to that taken for OCEANS and differs from the approach used in [6]. The TOPS model structure required substantially greater effort to convert than OCEANS due to software structural features which needed to be changed.

The starting Fortran 77 version of TOPS is a generic code with a generic test case. There are many provisions in the code for alternative input or output files and hooks for the addition of additional history variables beyond TOPS' basic set of two current components, temperature, and salinity.

3 DATA LAYOUT

Finite difference models of ocean or atmospheric variables tend to have simple mappings to the CM-5. Often a set of simultaneous equations must be solved and this can affect the choice of memory layout.

In the case of TOPS, tridiagonal systems are solved in routine TRID, called within the routine UPDAT2. There are many tridiagonal systems, one for each vertical column and for each variable. All of the systems for a given variable may be solved in parallel.

3.1 Layout Principles

The layout of data in an MPP is often the single most important factor in achieving good performance. Layout is a global decision in the sense that the best layout of data to optimize the performance of a routine or group of routines may not be the best to optimize any one operation in the code. For a compiler to make a reasonable choice, it would have to consider more than a single statement or loop nest and it would need knowledge of the iterative structures within and among routines. It is primarily for these reasons that MPP vendors have chosen to avoid the data layout problem and forced the programmer to choose.

Like atmospheric models, 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.

The basic shape of data in TOPS is three dimensional, two horizontal dimensions, x and y, and a vertical dimension z. The indices of z increase from the surface downwards. There are two sets of values corresponding to two time steps of the four basic variables.

The serial TOPS code approached data layout from the point of view of minimizing memory usage. A "slab" processing approach was used. Figure 1 shows how the method was implemented for TOPS. The main time step loop contains an inner slab loop which sequences through lower-dimensional sections of the history variables. In the case of TOPS, sections corresponding to x-z planes are taken. The slab loop must preserve the context needed to process the individual slabs in the y direction. This introduces an asymmetry in processing along the y direction which is not present in the mathematical formulation.

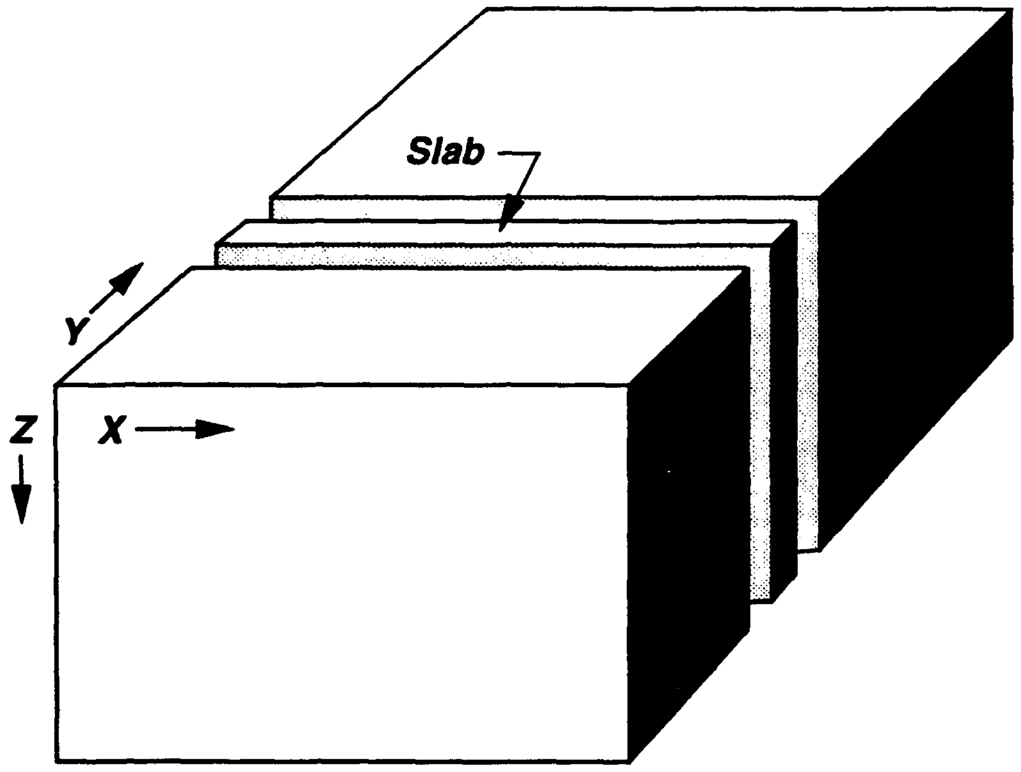

Fig. 1 — Slab Processing

The parallel version of TOPS does not use the slab approach because its low level of parallelism is insufficient for (and inefficient on) the Connection Machine. This fundamental change in the software caused most arrays shapes in lower level routines to change.

Current Thinking Machines compilers require that all serial axes precede the parallel ones. 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 three axes, level, time step, and variable first. In the present compiler, this does not result in efficient passing of three-dimensional sections to the low level computation routines.

Since the completion of the original TOPS port, Thinking Machines Corporation compilers have been changed to store arrays in a more standard form. Some of the current problems with array handling can be removed at the cost of again revising the code in a systematic way.

3.2 Major Variables

Figures 2 through 5 show the shapes of major variables originally and after conversion. Note that these definitions assume that serial axes last are acceptable. Since the conversion took place before the compiler restriction was lifted, all serial axes of parallel arrays have been moved to the left side, in the same order. Axes dimensioned `(n,m)` represent the x and y axes. All arrays with the combination of axes `(n,m)` are `NEWS` parallel. The `l` axis always represents the vertical direction and such axes are always allocated serially, either as completely serial arrays or serial within processor if the array is a parallel one (i.e., has the n and m axes). The axis nt ranges over time steps (2), and the nr axis ranges over variables (the 4 basic u, v, t, and s plus the potential for others). History variables are stored in a single array, `r(n,m,l,nt,nr)`.

Original Variable	New Variable	Notes
dxb(m)	dxb(n,m)	Spread
dxm(m)	dxm(n,m)	Spread
dxbr(m)	dxbr(n,m)	Calculated from dxb
dxmr(m)	dxmr(n,m)	Calculated from dxm
da(m)	da(n,m)	Spread
dxbda1(m)	dxbda1(n,m)	Calculated
dxbda2(m)	dxbda2(n,m)	Calculated
dydar(m)	dydar(n,m)	Spread
zb(l)	zb(l)	(Serial)
zm(l)	zm(l)	(Serial)
dzb(l)	dzb(l)	(Serial)
dzm(l)	dzm(l)	(Serial)
dzbr(l)	dzbr(l)	(Serial)
dzmr(l)	dzmr(l)	(Serial)
elong(n)	elong(n,m)	Spread
alat(m)	alat(n,m)	Spread

Fig. 2 — TOPS Arrays—Grid Parameters

Original Variable	New Variable	Notes
mask(n,m)	mask(n,m)	
xmask(n,m)	xmask(n,m)	
ymask(n,m)	ymask(n,m)	

Fig. 3 — TOPS Arrays—Mask Parameters

The notes in figures 2 through 5 indicate some arrays were *spread*. This means that a new axis has been introduced and the values along the new axis are copies of the original array values. A similar situation exists for arrays with the notation *calculated* but in these cases, the redundant values were computed rather than copied. In all of these cases, the redundant values were introduced to eliminate the need to communicate the non-redundant set. These variables all relate to grid parameters and do not have a vertical l axis. The space taken up by the redundant values is not significant. The other notation used in the figures is *collected*. This means that the m axis corresponding to the slabs has been added and the values along this axis correspond to the succession of values taken by the original array inside the slab loop.

The original code used a set of temporaries, again to conserve space. These temporaries were passed to lower level routines as work arrays. This practice is not effective on the Connection Machine because of the implicit equivalencing that is sometimes involved. The parallel code dispenses

Original Variable	New Variable	Notes
f(m)	f(n,m)	Spread
istype(ntyp)	istype(ntyp)	(Serial)
iptype(n,m)	iptype(n,m)	
qrf(l,ntyp)	qrf(l,ntyp)	(Serial)
xk(2)	xk(2)	(Serial)
yk(2)	yk(2)	(Serial)
zk(2)	zk(2)	(Serial)
xkdxr(n,m,2)	xkdxr(n,m,2)	
ykdyr(n,m,2)	ykdyr(n,m,2)	
zkdzr(l,2)	zkdzr(l,2)	(Serial)
r(n,m,l,nt,nr)	r(n,m,l,nt,nr)	
tlm(n,m)	tlm(n,m)	
kt(n,m)	kt(n,m)	

Fig. 4 — TOPS Arrays—Problem Parameters (Part 1)

Original Variable	New Variable	Notes
rgb(n,m,nr)	rgb(n,m,nr)	
rgs(n,nr)	rgs(n,m,nr)	Collected
qr(n)	qr(n,m)	
rbgt(n,m,3,nr-2)	rbgt(n,m,3,nr-2)	
ug(n,l)	ug(n,m,l)	Collected
vg(n,2,l)	vg(n,m,l)	Collected
u(n,l)	u(n,m,l)	
v(n,2,l)	v(n,m,2,l)	
w(n,l)	w(n,m,l)	
p1(n,2,l)	p1(n,m,l)	Collected
p2(n,2,l)	p2(n,m,l)	Collected
fly(n,2,l,nr)	fly(n,m,l,nr)	Collected
sum(n,l,nr)	sum(n,m,l,nr)	Collected
r2(n,l,nr)	r2(n,m,l,nr)	Collected

Fig. 5 — TOPS Arrays—Problem Parameters (Part 2)

with many of these temporaries in the new calling sequences but many instances not causing equivalence problems still remain. Needed temporaries are often dynamically allocated within the lower level routines, resulting in cleaner interfaces and higher quality code. The practice of allocating temporaries in the lower level routines is not possible in Fortran 77 since dimensional information is passed down to the lower level routines. Passing dimensions is often considered good programming practice but is the cause of some awkward programming idioms. A more desirable alternative is an include file containing all of the problem dimensions but this necessitates complete recompilation for each set of dimensions.

4 SOFTWARE STRUCTURE

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.

Table 1 — Call Tree of Original Code

Call Tree	Function
main	Entry point routine
· **dimens**	Define dynamic dimensions for run
· · **getint**	Read integer from standard input
· · · **strlen**	Find trimmed length of string
· **tops2**	True main routine
· · **define**	Define data set, model, and run parameters
· · · **vgrid**	Calculate vertical grid
· · · **sphgd2**	Calculate parameters for horizontal lat-lon spherical grid
· · · **prntpar**	Print out data set parameters
· · **lsmask**	Define the land-sea mask (boundary)
· · **tin**	Initialize temperature field
· · · **strcc3**	Concatenate three strings together
· · · · **strlen**	Find trimmed length of string
· · · **bgsrd**	Read 3D data file
· · · **tinitl**	Define initial 3D temperature field
· · · **getenv**	Get environment variable value
· · · **rdhdf**	Dummy routine for reading hdf files
· · **sin**	Initialize salinity field
· · · **strcc3**	Concatenate three strings together
· · · · **strlen**	Find trimmed length of string
· · · **bgsrd**	Read 3D data file
· · · **sinitl**	Dummy routine for initial salinity field

Call Tree	Function
· · · tsrel	Define initial 3D salinity field via T-S relation
· · · getenv	Get environment variable value
· · · rdhdf	Dummy routine for reading hdf files
· · · saladm	Adjust salinity profile
· · · · saladj	Adjust salinity profile
· · · · · interp	Perform linear interpolation
· · · · · pdenav	Calculate density anomaly for seawater
· · · · · coefex	Calculate coefficients of expansion of seawater
· · dats	Write initial temperature and salinity fields to DA file
· · · darite	Write 3D field or section to DA file
· · rin	Initialize auxiliary fields
· · setup	Perform setup calculations for TOPS
· · · extpro	Calculate solar extinction profile for mixed layer
· · · · extjerl	Interpolate an extinction profile for solar radiation
· · · · · interp	Perform linear interpolation
· · · · extmuel	Calculate solar flux extinction (Mueller and Lange)
· · · · · extir	Compute extinction with depth of IR spectrum
· · · · extmorl	Compute solar flux attenuance by pigment concentration
· · · · · extir	Compute extinction with depth of IR spectrum
· · · · · intrpb	Perform linear interpolation
· · prntgrd	Print parameters associated with horizontal or vertical grids
· · · prntd	Print all or part of 2D integer/real array
· · ein	Define initial Ekman velocities and mixed-layer depth
· · · surfbc	Calculate surface boundary conditions
· · · · sbctst	Calculate forcing for test case 12
· · · · sbcshp	Dummy routine for atmospheric forcing
· · · · sbcnog	Dummy routine for atmospheric forcing
· · · · sbcecm	Dummy routine for atmospheric forcing
· · · · darite	Write 3D field or section to DA file
· · · iekman	Initialize Ekman velocity field
· · · · buoy	Calculate buoyancy from termperature and salinity
· · output	Output model fields
· · · timepr	Print time in hours, days, etc
· · · print	Print out model fields interactively
· · · · prntsi	Print section of 3D field
· · · · · secdef	Interactively get 3D section
· · · · · prntsj	Print section of 3D field
· · · · · · prnts	Print section of 3D array in integer format

Call Tree	Function
· · · · · **secdef**	Interactively get 3D section
· · · · · **daread**	Read sections of fields from DA file
· · · · · **prntsj**	Print section of 3D field
· · · · · · **prnts**	Print section of 3D array in integer format
· · · · · **getreal**	Read real number from standard input
· · · · · · **strlen**	Find trimmed length of string
· · · · · **mld3am**	Estimate MLD field from 3D termperature field
· · · · · **prntc**	Print all or part of 2D integer array
· · · · · **profil**	Print vertical profiles at a single point
· · · · · · **pdenav**	Calculate density anomaly for seawater
· · · · · **getint**	Read integer from standard input
· · · · · · **strlen**	Find trimmed length of string
· · · · · **switch**	Swap two integers
· · · · **prbgt**	Print regionally averaged heat and salt budgets
· · · · · **avellm**	Find area average for a data field
· · · · · · **sphgdw**	Calculate weights for statistics on lat-lon grid
· · · · · **daread**	Read sections of fields from DA file
· · · **surfbc**	Calculate surface boundary conditions
· · · · **sbctst**	Calculate forcing for test case 12
· · · · **sbcshp**	Dummy routine for atmospheric forcing
· · · · **sbcnog**	Dummy routine for atmospheric forcing
· · · · **sbcecm**	Dummy routine for atmospheric forcing
· · · · **darite**	Write 3D field or section to DA file
· · · **geovel**	Calculate geostrophic currents
· · · · **geovelb**	Calculate pressure
· · · **cirvel**	Calculate circulation currents
· · · · **getcmhis**	Read history files from OCEANS
· · · · **extendhad**	Extend/extrapolate 2D field to fill in unknown values
· · · · **cirvelc**	Calculate pressure
· · · **advvel**	Calculate total advection velocity
· · · **davel**	Write geostrophic circulation and total velocities to file
· · · · **darite**	Write 3D field or section to DA file
· · · **cfl**	Calculate CFL parameters for advection and diffusion
· · · **updat2**	Update fields
· · · · **advdif**	Calculate rate of change of field due to several causes
· · · · **budget**	Calculate cumulative budget for temperature and salinity
· · · · **mod2**	Calculate vertical eddy coefficients
· · · · · **buoy**	Calculate buoyancy from termperature and salinity

Call Tree	Function
· · · · · hann3	Apply Hanning 3-point smoother to an array
· · · · trid	Solve tridiagonal system of linear equations
· · switch	Swap two integers
· · conad	Perform convective adjustment of density field
· · · buoy	Calculate buoyancy from termperature and salinity

5 CONVERSION

This section discusses the process used to convert the Fortran 77 serial version of TOPS to a Fortran 90 (or CM Fortran) parallel TOPS. Section 5.1 discusses the basics of the code conversion. In Section 5.2, a practice called *parallel extension* is described and an example is shown. The most significant changes to TOPS related to the tridiagonal solver. Section 5.3 discusses the underlying theory used and shows the changes in tridiagonal setup in order to solve all tridiagonal systems in parallel. The section closes with a discussion of verification methodology.

5.1 Basic Conversion

In the conversion of the OCEANS model (see [7]), the most difficult conversion problems 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. A degree of run-time checking is present in the current compiler version but the reliability and thoroughness is not clear.

One metnod, use of interface blocks, has since emerged as a way to detect these mismatches. In the TOPS conversion interface blocks were used to check interfaces at compile time. In order to make interface blocks easy to maintain, a single include file was constructed which defined the interfaces for all routines. This proved to be very convenient and helpful in that interface problems never became a major source of run-time bugs. I few minor anomalies were noted in the use of interface blocks, particularly in practices which involved implicitly passing array sections. Although these practices are common in sequential Fortran 77 usage, they are somewhat questionable in Fortran 90. The use of an include file minimizes software maintenance effort and the entire interface block file can be replaced by commented lines if the interface block concept is not supported or not desired.

As in the OCEANS conversion, common blocks were moved to include files. TOPS, however, does not rely on common blocks in a significant way and uses the technique of passing arguments through the formal parameter list. As mentioned in Section 3, the TOPS practice of passing dimensions was occasionally awkward.

In general the conversion of TOPS retained the same module structure as in the Fortran 77.

The primary deviations came from the process of parallel extension (see Section 5.2) were where it was sometimes necessary to retain a serial version as well as the new parallel version.

Much of the code operates on the interior of the horizontal extent with horizontal subscripts running from 2 through n-1 and 2 through m-1. This creates visually cluttered code because section designators must be specified for each array usage. There is also a small run-time cost to calculate the masks. A better approach is to use a mask which is .TRUE. in the interior and .FALSE. on the boundary. When used in the WHERE statement, cleaner code is produced with the potential for executing slightly faster on the CM-5. An easy method of handling these masks is to place them in common blocks and have the routines use them as needed. The problem with this in TOPS is the previously mentioned practice of passing dimensional information through the parameter list. The masks in common must have static dimensions but the compiler cannot know or assume that the passed dimensions of array arguments are the same as the common mask arrays. An alternate is to explicitly pass the mask arrays to the lower level routines but this requires changes to almost all parameter lists and gives a somewhat cluttered style. It should be noted that in the current code masks are frequently generated as local temporaries. This can be considered a compromise solution although many routines still contain explicit sectioning.

5.2 Parallel Extension

Many routines in the Fortran 77 TOPS operated on a single point or a single column, an approach lacking parallelism. In most cases, and in all cases occurring within the time step loop, the routines were extended to operate over all points in a vertical plane or on all columns. The approach chosen in the serial code often reflects the fact that the operations were applied only to a subset of points or columns. In the parallel versions, a mask is supplied which selects which elements are to be operated on by the routine.

Figure 6 shows a typical routine from the Fortran 77 version. In this case the routine operated on a single point at a time, applying a pair of formulas to the scalar values t and s, water temperature and salinity.

The parallel routine is shown in figure 7. Dimensional information has been added (parameters ni and mj) as well as a controlling mask array. The t and s parameters are now entire horizontal extents. The expressions in lines 24 and 25 of the figure are now array operations and the result parameters alphag and betag are arrays.

This example also illustrates the practice of setting inactive elements (according to the mask) to zero so that they at least have valid values, occasionally a helpful situation during debugging. The Connection Machine layout directives are shown in lines 8–10. The interface blocks are included in line 12.

5.3 Tridiagonal Solver TRID

This routine originally solved a single tridiagonal system of order *k* where the first *k* levels of the history variables represent the mixed layer. Since the depth of the mixed layer potentially differs in each column, a method was needed to solve a large number of tridiagonal systems of varying order in parallel.

```
      subroutine buoy(t,s,alphag,betag)
c  subroutine to calc buoyancy parameters from temperature
c  and salinity at atmospheric pressure.  calculation is
c  based on polynomial equation of state of fredrich and
c  levitus (jpo, oct 72).
c       t = temperature in deg c.
c       s = salinity in ppt.
c       alphag = -g/rozero * d(rho)/d(t).
c       betag  =  g/rozero * d(rho)/d(s)
c  note that alphag and betag are defined to be positive.
c  value of g is taken to be 980 cm/s**2.
c  created 7-27-82.  paul j martin.   norda code 322.
      dimension ae(5),be(3)
      save ae,be
      data ae/-4.7577e-2,  1.4516e-2, -1.0093e-4,
     &         2.8743e-3, -7.1320e-5/
      data be/ 7.7023e-1, -2.8743e-3, 3.5659e-5/
c
      alphag=ae(1)+t*(ae(2)+t*ae(3))+s*(ae(4)+t*ae(5))
      betag =be(1)+t*(be(2)+t*be(3))
c
      return
      end
```

Fig. 6 — Example of Parallel Extension—Before

```
      subroutine buoy(ni,mj,t,s,alphag,betag,mask)
c
c  Arguments:
c
      integer ni,mj
      real t(ni,mj),s(ni,mj),alphag(ni,mj),betag(ni,mj)
      logical mask(ni,mj)
cmf$  layout t(:news,:news),s(:news,:news)
cmf$  layout alphag(:news,:news),betag(:news,:news)
cmf$  layout mask(:news,:news)
c
      include 'interface.blocks'
c
c  Local Variables:
c
      real ae(5),be(3)
cmf$  layout ae(:serial),be(:serial)
      save ae,be
      data ae/-4.7577e-2,  1.4516e-2, -1.0093e-4,
     &         2.8743e-3, -7.1320e-5/
      data be/ 7.7023e-1, -2.8743e-3, 3.5659e-5/
c
      where(mask)
        alphag = ae(1)+t*(ae(2)+t*ae(3))+s*(ae(4)+t*ae(5))
        betag  = be(1)+t*(be(2)+t*be(3))
      elsewhere
        alphag = 0.0
        betag = 0.0
      end where
c
      return
      end
```

Fig. 7 — Example of Parallel Extension—After

The method chosen was to imbed the individual kth-order systems

$$Ax = b$$

within larger systems, all of a constant fixed order in such a way that the result is essentially the same. This allows solving multiple instances in parallel using a simple extension of the original single system solver. The system being solved is, of course,

$$\begin{pmatrix} A & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b \\ y \end{pmatrix} \tag{1}$$

where y is a vector containing the values of the history variable (temperature or salinity xample) below the mixed layer.

There are two problems with this approach. First, the amount of work is greater since a larger system is solved in each column than before. In past experiences, it was advantageous to trade additional floating point operations to reduce communications or to maintain a high level of parallelism. New capabilities in the Connection Machine CM-5 system software now permit an alternative approach. See section 7 for more details.

The second problem relates to the condition numbers of the modified tridiagonal systems which potentially differ from the condition numbers of the original systems. The condition number of a linear system of equations $Ax = b$ is related to the singular value decomposition of the matrix A. The condition number reflects how sensitive the solution of $Ax = b$ is to errors in the elements of A. A system with a high condition number may produce inaccurate results. Some algorithms misbehave for ill-conditioned systems.

The singular value decomposition is a factorization of the matrix into a product of three matrices

$$A = U\Sigma V^T \tag{2}$$

where U and V are orthogonal and Σ is diagonal. T Alternatively, using the orthogonality of U and V,

$$U^T AV = \Sigma. \tag{3}$$

The matrix Σ has n positive singular values σ_1 to σ_n of non-increasing magnitude where the rank of the matrix is n. If the order of the matrix is greater than n, Σ contains trailing zeros to give Σ the same order as A. The factorization always exists for any rectangular matrix.

The condition number of a matrix A is defined in terms of the singular values of A to be

$$\kappa_2(A) = \frac{\sigma_1(A)}{\sigma_n(A)}. \tag{4}$$

That is, the condition number is the ratio of the largest and smallest singular values.

The condition number of an embedded system

$$\hat{A} = \begin{pmatrix} I & 0 & 0 \\ 0 & A & 0 \\ 0 & 0 & I \end{pmatrix}$$

σ_1	σ_n	κ_2
< 1	< 1	Increased
< 1	> 1	Impossible
> 1	< 1	Unchanged
> 1	> 1	Increased

Fig. 8 — Relationship of $\kappa_2(\hat{A})$ to $\kappa_2(A)$

is not the same as the condition number of A. To reveal this relationship observe that

$$\begin{pmatrix} I & 0 & 0 \\ 0 & U^T & 0 \\ 0 & 0 & I \end{pmatrix} \cdot \begin{pmatrix} I & 0 & 0 \\ 0 & A & 0 \\ 0 & 0 & I \end{pmatrix} \cdot \begin{pmatrix} I & 0 & 0 \\ 0 & V & 0 \\ 0 & 0 & I \end{pmatrix} = \begin{pmatrix} I & 0 & 0 \\ 0 & U^T AV & 0 \\ 0 & 0 & I \end{pmatrix} \quad (5)$$

$$= \begin{pmatrix} I & 0 & 0 \\ 0 & \Sigma & 0 \\ 0 & 0 & I \end{pmatrix}. \quad (6)$$

The above demonstrates that the set of singular values of $\hat{A}$ is the same as the set of singular values of A with the addition of the singular value 1. Thus, the condition number of $\hat{A}$ is

$$\kappa_2(\hat{A}) = \frac{\sigma_1(\hat{A})}{\sigma_n(\hat{A})} \quad (7)$$

$$= \frac{\max(1, \sigma_1(A))}{\min(1, \sigma_n(A))}. \quad (8)$$

There are several possibilities for $\kappa_2(\hat{A})$, summarized in the following table. The singular values of typical A matrices used in the TOPS tridiagonal routine have not been determined so that it is unknown which case in figure 8 applies.

In terms of TOPS arrays, the augmented tridiagonal system is

$$\begin{pmatrix} T_{kt-1} & 0 \\ 0 & I_{51-(kt-1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \quad (9)$$

where kt is the index of the first vertical level past the mixed layer.

The setup code from the Fortran 77 serial TOPS version is shown in figure 9. The variable kt is the index of the first point below the mixed layer and the variable ktm1 is just one less than kt. The subscript k indexes the vertical dimension of a column and am, bm, and cm are vectors for the lower, main, and upper diagonals of a single tridiagonal system.

Figure 10 shows the parallel code. The complexity of the new code appears high at first but is really fairly simple once the programming technique, used often in the parallel version, is understood. The loop in lines 1 through 7 simply sets the three diagonals of all active columns to the identity. The mask updmask selects the columns of the interior of the region. The initial values of the main and upper diagonals are set in lines 8 through 11. Note that only columns with at least two elements in the mixed layer are affected. The loop from line 12 to line 18 is perhaps the most

```
      ...
      cm(1)=-dtzmb2(1)*zk2(2,id)
      bm(1)=1.-cm(1)
      if (ktm1 .gt. 2) then
        do k=2,ktm2
          am(k)=-dtzmb1(k)*zk2(k,id)
          cm(k)=-dtzmb2(k)*zk2(k+1,id)
          bm(k)=1.-am(k)-cm(k)
        enddo
      endif
      am(ktm1)=-dtzmb1(ktm1)*zk2(ktm1,id)
      bm(ktm1)=1.-am(ktm1)
      ...
      call trid(ktm1,am,bm,cm,r2(1,ir))
```

Fig. 9 — Tridiagonal Setup—Before

interesting. The vertical index k sweeps throuhout its total possible range (all possible interiors of the tridiagonal systems) and the WHERE statement in line 13 insures that only columns that should be updated are modified. As k increases toward its maximum value, fewer and fewer columns will be selected. Lines 19 through 24 have a similar structure to the preceding loop and here simply set the final values in the lower and main diagonals. Note that the WHERE statement sweeps through all possible vertical indices and selects no more than one index in any column. Finally, all of the tridiagonal systems are solved together in line 26.

5.4 Verification

The software structure of the original model was used as the basis for the verification of the conversion. The module groups were converted and tested in groups. The groups were based on order of execution including two initialization groups and the time step group. Multiple tests were performed to the extent possible for alternative paths. At a minimum, each converted routine and all major branches within were executed at least once.

In contrast to the approach used in OCEANS the test data and verification were carried out using the serial code as a generator. The serial code was changed by the addition of code to write out data generated by the serial code. This was used both as input and as output to be verified. This method was chosen over the random inputs used for OCEANS because it was believed that the inputs would be better test cases for the converted code.

The arrays of data generated were verified element by element. This again differs from the OCEANS conversion process where a statistical verification approach was used. It has been found that the statistical approach often misses errors and is not helpful in identifying where errors occur even when detected in the statistics. Agreement to 5 significant figures was generally obtainable although problems were encountered deciding when to employ relative error tests versus absolute

```
do k=1,l
  where (updmask)
    am(k,:,:) = 0.0
    bm(k,:,:) = 1.0
    cm(k,:,:) = 0.0
  end where
end do
where (updmask .and. kt .gt. 2)
  cm(1,:,:) = -dtzmb2(1)*zkmh(2,:,:)
  bm(1,:,:) = 1.-cm(1,:,:)
end where
do k=2,l-2
  where (k .le. kt-2 .and. updmask .and. kt .gt. 2)
    am(k,:,:) = -dtzmb1(k)*zkmh(k,:,:)
    cm(k,:,:) = -dtzmb2(k)*zkmh(k+1,:,:)
    bm(k,:,:) = 1.-am(k,:,:)-cm(k,:,:)
  end where
end do
do k=2,l-1
  where (k .eq. kt-1 .and. updmask .and. kt .gt. 2)
    am(k,:,:) = -dtzmb1(k)*zkmh(k,:,:)
    bm(k,:,:) = 1.-am(k,:,:)
  end where
end do
...
call trid(n,m,l,am,bm,cm,u2,updmask .and. kt .gt. 2)
```

Fig. 10 — Tridiagonal Setup—After

error tests. In one portion of the code, the routine UPDAT2 and the called routine MOD2, it was impossible to duplicate the Sun Fortran 77 results on the CM-5. In further testing, it was found that Convex single precision results also differed from the Sun and the CM-5. Eventually, double precision Convex results were verified with double precision CM-5.

6 PERFORMANCE

The conversion of TOPS was carried out only to the point where correct answers were obtained through a number of time steps. As the result of changing priorities, further work on the parallel version of TOPS has been delayed and comparative timings in particular remain undone.

The projected performance of TOPS on the Connection Machine is expected to be good. No unusual problems were encountered that required general communication. No attempt has been made, however, to collect together repeated communications operations as was done with the OCEANS code.

One final performance comment is appropriate. The relative performance of TOPS will increase with larger grid sizes (higher resolutions). The largest resolution which seems appropriate for TOPS is about 768 by 384 by 51 points. Most parallelism is in 768 by 384 layers. This grid size is only moderate for the Connection Machine, and is significantly smaller than the approximately 2000 by 1000 grid size used in OCEANS. Performance is thus expected to be somewhat less than that achieved by OCEANS running on the CM-5. For details on the performance of OCEANS, see [7].

7 FUTURE WORK

The TOPS conversion was overtaken by changing priorities and was not completely finished although the model appears to be producing correct results. A number of generally small changes could be, and perhaps should be, undertaken to move the model to a more usable state.

First, the Thinking Machines compiler restrictions on serial axes have been removed and hence the axes could be reordered into a more standard Fortran ordering. This change is relatively straightforward to carry out but involves changing almost every line of code.

Once the axes have been reordered, a new run time feature called array aliasing can be employed. This is most useful as a means of improving the efficiency of communication operations, exclusively CSHIFT operations in TOPS. This may yield a significant improvement but the long term improvement may disappear as the compiler and run time system are improved. Additionally, this feature is definitely non-portable.

A minor change is removal of dead code. Most of the existing TOPS Fortran 77 code is still present, commented out, in the source code. This is useful during debugging but it does detract from the readability of the finished code. This change is pervasive but cosmetic only. Despite the possibility of introducing errors by accidentally deleting live code, the final product is worthwhile.

The tridiagonal solver is also an area where improvements can be made. First, it is possible to reduce the order of the tridiagonal systems solved. Currently the size is equal to the total vertical dimension. This is probably highly conservative and it could be reduced with little effort to the maximum size needed over all active columns. This is likely to yield a useful improvement but it typical magnitude is unknown.

Also in the tridiagonal solver, a new compiler capability called local/global mode could be exploited. Fortran 90 code normally runs in what is called global mode where operations are synchronized at a statement level. In local mode a subroutine executes independently on its own local data, essentially in MIMD mode. Applying this to the tridiagonal solver, each CM-5 processing node can solve all of its local tridiagonal systems using whatever method desired and the total running time is simply the maximum time needed by any processing node. Local mode is not part of Fortran 90 or High Performance Fortran and so this change would not be portable.

Some of the code in TOPS was not converted because it was not used in the test case. Perhaps the most important of these was routine CONAD. This routine has been preliminarily converted but not debugged. Code conversion was the most difficult in all of TOPS. Discussion with code author Paul Martin has led to the conclusion that a revised formulation could be the best longer term solution since performance of CONAD as currently converted would not appear to be especially good.

The final code change is a relatively minor one. TOPS has a direct access file in which history variables and key intermediaries are stored for output and debugging purposes. The time to write this file is a significant part of the current parallel run time and improvements in this area would be beneficial to the overall run time. The improvement can come from using the CM-5 Scalable Disk Array, a parallel file system, in one of several ways. On the other hand, the frequency with which the file is written is user controlled and need not occur often in practice.

Finally, TOPS needs to be timed on larger test cases and performance problems fixed. This will allow direct and meaningful performance comparisons between the parallel version and the serial version. There is much external Navy interest in these comparisons.

8 CONCLUSIONS

The code conversion of the model was relatively straightforward although not as simple as in the OCEANS model. The conversion process introduced generally systematic changes in the data structures because the slab technique used in the Fortran 77 version was extended to fully parallel for the Connection Machine.

The performance of the parallel version was not tested although no evident performance problems were encountered.

Verification of the code was problematical. It was finally discovered that a portion of the code was sensitive to round off error in single precision and hence verification was not possible until both serial and parallel codes were run using 64 bit arithmetic.

Changing priorities have left the TOPS conversion at a preliminary operational level. There are a number of relatively small activities which need to be done for TOPS, particularly timings on high resolution grids.

9 ACKNOWLEDGMENTS

The authors wish to thank Dr. Paul J. Martin for his assistance with the conversion of TOPS. We also thank Dr. Henry Dardy of the NRL Connection Machine Facility, Dr. Steve Piacsek of NRL-Stennis, and Dr. Robert A. Peloquin and Elizabeth Wald of the Office of Naval Research for their support.

10 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).
5. "High Performance Fortran," Available from C. H. Koelbel, Rice University Computer Science Department, Houston, TX.
6. Sela, J. G., Anderson, P. B., Norton, D. W., Young, M. A., Massive Parallelization of NMC's Spectral Model, Journal of Parallel and Distributed Computing, to appear.
7. Anderson, P. B., Young, M. A., Connection Machine Software Conversion of a Navy Oceans Model, NRL memorandum report 7400, 1993.