Naval Research Laboratory
Washington, DC 20375-5320

NRL/MR/5593--95-7755

CM-5 Kernel Optimization of a Global Weather Model

P.B. ANDERSON
D.W. NORTON
M.A. YOUNG

Center for Computational Science
Information Technology Division

July 7, 1995

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 7, 1995	

4. TITLE AND SUBTITLE

CM-5 Kernel Optimization of a Global Weather Model

5. FUNDING NUMBERS

PE - 62435N
PR - R035E35

6. AUTHOR(S)

P.B. Anderson, D.W. Norton, and M.A. Young

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/5593--95-7755

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5660

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*)

This report documents the results of optimizing the computational kernel of the National Meteorological Center (NMC) weather forecasting model running on the Thinking Machines Corporation CM-5. The baseline for this optimization work is a basic version of the kernel which runs on both the CM-5 and CM-200. An optimized version for the CM-5 is produced and its performance is compared with the basic version on both CM-5 and CM-200 as well as the best-performing CM-200 version.

14. SUBJECT TERMS

Connection machine
Fortran 90
Weather Modeling

15. NUMBER OF PAGES

19

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

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

CONTENTS

List of Figures

CM-5 KERNEL OPTIMIZATION OF A GLOBAL WEATHER MODEL

1 INTRODUCTION

This report documents the results of optimizing the computational kernel of the National Meteorological Center (NMC) weather forecast model running on the Thinking Machines Corporation (TMC) CM-5. A main issue to be investigated is whether the data layout and computational approach used for the TMC CM-200 remains a good one for the CM-5, a machine with a significantly different architecture. For descriptions of the CM-200 and CM-5 architectures, see [1, 2].

The baseline for this optimization work is a basic version of the kernel which runs on both the CM-5 and CM-200. An optimized version for the CM-5 is produced and its performance is compared with the basic version on both CM-5 and CM-200 as well as the best-performing CM-200 version.

The optimizations described below took place during a staged upgrade of the NRL CM-5 hardware to CM-5E level. These changes included replacing the Cypress SPARC processing nodes with Viking SuperSPARC chips, upgrading the network interface (NI) chips, quadrupling the memory to 128 MBytes per node, and increasing the clock speed from 32 MHz to 40 MHz. Some effort has been made to identify and separate the improvement derived from the hardware upgrade from those improvements made through kernel software changes. We hope mature software able to take greater advantage of the improved network hardware will yield a significant performance increase. The remainder of the code is affected by some of the changes made to the kernel so additional work is needed before these improvements can be applied to the full forecast model.

The next section summarizes the performance that has been achieved. Section 3 provides some background information about the kernel. The changes that were retained in the optimized version are documented in Section 4 and the resulting performance is given in Section 5. Changes which were considered but not implemented in the optimized version are described in Section 6. The report concludes with future plans and some final remarks in Sections 7 and 8.

2 SUMMARY OF RESULTS

Figure 1 shows the relative performance on the CM-200 and CM-5E of a common baseline version and machine dependent optimized versions. The numbers are ratios of execution time—time for the row version divided by the time for the column version. Only the upper triangle is

Manuscript approved May 18, 1995.

	CM-200		CM-5E	
	Baseline	Optimized	Baseline	Optimized
CM-200 Baseline	1.00	2.17	3.83	8.91
CM-200 Optimized		1.00	1.76	4.09
CM-5E Baseline			1.00	2.33
CM-5E Optimized				1.00

Fig. 1 — Relative Performance Comparisons (Row Time/Column Time)

filled in since elements of the lower triangle are simply the inverses of the corresponding upper triangle elements. In our experience with upgrading a CM-5 to CM-5E, performance of the CM-5E is almost exactly 25% greater than the CM-5. Absolute performance figures are given in Section 5.

CM-200 relative performances shown in figure 1 were obtained from measurements on the NRL machine using 512 Floating Point Units (FPUs) in slicewise mode. CM-5E performance figures were derived from measurements on all 256 processing nodes (PNs) of the NRL machine. Each processing node is equipped with four vector units (VUs) per PN.

Comparisons between the NRL CM-200 and the NRL CM-5E are complicated by differences in architecture, particularly with regard to floating point functional units. Considering just the clock rates and hardware floating point unit counts of both CM-5 and CM-200 configurations, a relative performance level of about

$$\frac{16\ \text{MHz}}{10\ \text{MHz}} \cdot \frac{1024\ \text{VUs}}{512\ \text{FPUs}} = 3.2$$

would be expected. For the CM-5E compared to the CM-200, relative performance would be expected to be about

$$\frac{20\ \text{MHz}}{10\ \text{MHz}} \cdot \frac{1024\ \text{VUs}}{512\ \text{FPUs}} = 4.0$$

These simple metrics yield a potentially useful summary characterization but they ignore the significant differences in the network architectures of the CM-200, CM-5, and CM-5E. Counterbalancing this oversight is the fact that the CM-5 architecture is intended as a successor to the CM-200 and, having been developed by the same company, probably embodies similar design assumptions.

The changes made to improve CM-5 performance are in the fine details of the code. Since no significant changes were made to the baseline processing approach, it is clear that the CM-200 data layout has continued to be efficient for the CM-5.

3 BACKGROUND

The conversion of model variables between spectral space and a grid space representation occurs during every forecast model timestep. Efficient mappings between grid and spectral spaces are therefore critical to performance. Much of the remaining model timestep computation is done element-by-element in either spectral or grid space. Since element-by-element computation presents few problems on a distributed memory parallel processor, primary focus for optimization is on efficient transforms between spectral and grid spaces.

The process of converting from spectral space to grid space consists of evaluating the double sum

$$F(\lambda,\theta) = \sum_{l=0}^{J}\sum_{n=l}^{J} f_n^l p_n^l(\theta) e^{il\lambda} \tag{1}$$

where λ and θ are the longitude and co-latitude, $F(\lambda,\theta)$ is the grid space quantity, the f_n^l are the corresponding spectral expansion coefficients in zonal wavenumber l and ordinal wavenumber n. The sums range to the truncation limit J. The functions $p_n^l(\theta)$ are the associated Legendre polynomials, evaluated at co-latitude θ. Co-latitudes are chosen such that an exact Gaussian quadrature can be performed during the grid to spectral computation. The inner sum is sometimes referred to as a Legendre transform and the outer sum is a discrete Fourier transform.

The quantities f_n^l form a triangular array as shown in figure 2. This array has been linearized as discussed in [6].

	$l=0$	$l=1$	$l=2$	$\cdots$	$l=J-1$	$l=J$
$n=J$	f_J^0	f_J^1	f_J^2	$\cdots$	f_J^{J-1}	f_J^J
$n=J-1$	f_{J-1}^0	f_{J-1}^1	f_{J-1}^2	$\cdots$	f_{J-1}^{J-1}	
$n=J-2$	f_{J-2}^0	f_{J-2}^1	f_{J-2}^2	$\cdots$		
$\vdots$	$\vdots$	$\vdots$				
$n=1$	f_1^0	f_1^1				
$n=0$	f_0^0					

Fig. 2 — Triangular Structure of f_n^l

1. Spread f_n^l, adding a latitude dimension.
2. Multiply the spread f_n^l by associated Legendre polynomials evaluated at gaussian grid latitudes.
3. Perform inner sums of Eq. (1).
4. Zero unused $F(\lambda,\theta)$ array.
5. Copy (Send) inner sums to $F(\lambda,\theta)$ array.
6. Perform complex-to-complex FFTs along longitude axis.
7. Copy real part of result to output array.

Fig. 3 — Spectral-to-Grid Conversion Algorithm

4 IMPLEMENTED OPTIMIZATIONS

Four separate groupings of optimizations were implemented. The most fundamental change involved reordering array axes to be more consistent with standard Fortran order. This is described in Section 4.1. Section 4.2 discusses changes made to improve communication efficiency through

use of multiple word packing. Changes relating to FFT performance are briefly described in Section 4.3. Finally, the replacement of two short code segments by CMSSL calls is explained in Section 4.4.

4.1 Axis Reordering and Ensembles

The order of axes in the baseline version was determined largely by compiler restrictions, specifically CM Fortran version 1.2 and earlier. In the new CM Fortran version 2.1 compiler, most of these restrictions have been relaxed (under appropriate compiler options) so that memory locality considerations common to conventional architectures are now of highest priority in selecting axis ordering.

Figures 4 through 6 show the old and new axis orders for the major shapes. There were two basic goals of reordering. First, to maximize memory reference locality (and to provide large contiguous blocks for important operations), an axis ordering was selected which generated the longest strings of whole axis array descriptors before the appearance of a scalar axis index. That is, long sequences of leading colons in a variable reference such as $x(:,:,:,:,k)$ were preferred to $x(:,:,k,:,:)$. The new axis ordering is also a good one for shared memory systems.

Version	Shape	Layout
Baseline	(2, levs, 2, cmn2)	(serial, serial, serial, parallel)
Reordered	(2, 2, cmn2, ens, levs)	(serial, serial, parallel, parallel, serial)

Fig. 4 — fln Axis Orders

Version	Shape	Layout
Baseline	(2, levs, 2, lats, cmn2)	(serial, serial, serial, serial, parallel)
Reordered	(2, 2, cmn2, ens, lats, levs)	(serial, serial, parallel, parallel, serial, serial)

Fig. 5 — flns Axis Orders

Version	Shape	Layout
Baseline	(2, levs, 2, lats, lonf)	(serial, serial, parallel, parallel)
Reordered	(2, lonf, lats, ens, 2, levs)	(serial, parallel, parallel, parallel, serial, serial)

Fig. 6 — cgr Axis Orders

The second factor affecting axis ordering was to provide a basis for the use of array aliasing (described in the next section). Array aliasing is a rough equivalent of the equivalence feature of standard Fortran and provides a way to improve performance through double and quadruple word memory and network accesses.

A new feature, an ensemble axis, was added to most arrays. The ensemble axis indexes a set of independent forecast variables to be used in computing forecast ensembles. The axis can be set at size one to give an equivalent single forecast model as before. If an ensemble forecast model is desired, the ensemble axis length can be set to the desired size, on the order of 8 to 16. The ensemble axis is spread across processors and offers a new source of parallelism, thereby improving the efficiency of the code, especially on large processor configurations.

Reordering the axes was not expected to give a significant performance improvement. Timings of an intermediate version incorporating just reordered axes confirmed that there was no measurable change in performance.

4.2 Communication Packing

A large fraction of the overall performance improvement was obtained through use of more efficient interprocessor communication. In Section 4.2.1 a justification is given for packing single words into double or quadruple word groups and a simple performance analysis is made. Packing not only increases interprocessor communication performance, memory access (especially write access) is also improved. Section 4.2.2 discusses how and where packing was employed.

4.2.1 Packing Rationale

The interprocessor communication system of the CM-5 is packet based with a two word header and up to four words of payload. When communication is needed, current TMC software for send and scan operations uses packets having a payload the same size as the operation element size. That is, a single precision send operation would send data in single word packets. Figure 7 shows how large an effect this can be.

Element Size (Words)	Packet Size (Words)	Overhead (Percent)
1	3	67
2	4	50
4	6	33

Fig. 7 — Network Packet Overhead

If the data to be sent can be packed into larger units, greater network efficiency can be obtained. In addition, packing reduces the number of packets sent, thereby diminishing any fixed startup cost associated with communication. Empirical trials have shown that packing single words into double words can speed up a typical send operation by a factor of two. Quadruple word packing can give a speedup of about three.

To evaluate the fixed overhead and effective transfer rate for the NMC kernel, assume that communication time t consists of a fixed component u and a time per word w:

$$t = u + kw.$$

Since fewer packets are needed for larger packet sizes, measured performance times t_1, t_2, and t_3 corresponding to single, double, and quadruple word transfers should satisfy the equations

$$\begin{aligned} 4(u+3w) &= t_1/q \quad \text{(Single word transfers)}, \\ 2(u+4w) &= t_2/q \quad \text{(Double word transfers), and} \\ (u+6w) &= t_3/q \quad \text{(Quadruple word transfers)}, \end{aligned}$$

where q is a factor that accounts for the total number of packets. For the two sends in the NMC kernel, $q = 1400832/256$ and $q = 700416/256$.

Given three measurements t_1, t_2, and t_3, a solution for the two unknowns u and w can be obtained by minimizing the sum of the squared residuals of the system

$$\begin{pmatrix} 4 & 12 \\ 2 & 8 \\ 1 & 6 \end{pmatrix} \begin{pmatrix} u \\ w \end{pmatrix} = \begin{pmatrix} t_1/q \\ t_2/q \\ t_3/q \end{pmatrix}$$

This system is of the form $Ax = b$ and the least squares solution is the value of x which minimizes $||Ax - b||_2^2$. The well known solution of the above expression is

$$x = \left(A^T A\right)^{-1} A^T b.$$

Substituting numbers from the A matrix above, the solution is

$$\frac{1}{224} \begin{pmatrix} 136 & -72 & -176 \\ -28 & 28 & 56 \end{pmatrix} \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = \begin{pmatrix} u \\ w \end{pmatrix}.$$

Filling in measured values for the first send

$$\begin{aligned} t_1/q &= 5.83 \times 10^{-5} \\ t_2/q &= 2.78 \times 10^{-5} \\ t_3/q &= 1.80 \times 10^{-5} \end{aligned}$$

gives the solution

$$\begin{aligned} u &= 12.3\mu\text{s and} \\ w &= 0.69\mu\text{s/word}. \end{aligned}$$

Measurements for the second send,

$$\begin{aligned} t_1/q &= 1.94 \times 10^{-5} \\ t_2/q &= 1.25 \times 10^{-5} \\ t_3/q &= 1.12 \times 10^{-5} \end{aligned}$$

give the solution

$$\begin{aligned} u &= -1.04\mu\text{s and} \\ w &= 1.94\mu\text{s/word}. \end{aligned}$$

These results are somewhat inconsistent with a standard TMC network performance test which produces $u = 3.3 \pm 40$ μs and $w = 0.38 \pm 0.008$ μs/word. This test uses low level message passing calls rather than global mode operations. The raw transfer rates implied by the two experimentally derived values of w are about 5.8 and 2.0 MBytes/second/PN compared with 10.7 MBytes/second/PN in the TMC measurement. In passing, negative startup times are simply the result of the least-squares fitting procedure and have no physical interpretation.

It is not surprising that there is a significant difference between the TMC value and the kernel values since it is known that the low level message passing library is more efficient than typical communication in the global mode. In addition, the communication load measured through the weather kernel reflects a degree of imbalance whereas the TMC test has near-perfect balance. The overall performance of the quadruple word versions of the weather kernel based only on the payload gives 0.85 and 1.36 MBytes/second/PN.

4.2.2 Packing Applications

Packing can be done implicitly or explicitly. Implicit packing relies on the aliasing features of the compiler and run time system to associate an array of double or quadruple word elements with the memory space of an array of single words. The single word array typically has a shape with more axes or the same number of axes but larger axis lengths. Aliasing accomplishes its objective through an alternative array descriptor which maps an existing memory area into a new shape and element size.

Explicit packing creates new arrays from old arrays using standard language features. In particular, the construction of a complex array composed of separate real and imaginary parts using the Fortran `cmplx` intrinsic function explicitly packs data into double word form.

Packing can be applied to the send operations and to the scan operations. In the case of the two send operations, packing to either double words (implicitly or explicitly) or quadruple words (implicitly) can be done. The reordered shapes discussed in Section 4.1 were selected, in part, so that two leading length two axes would be available for aliasing. Figure 8 demonstrates the aliasing technique.

For the send operation, both double and quadruple word aliasing versions were developed and tested. In the double word version, an array of type complex is aliased to an array with a leading length two axis. This version is faithful to the meaning of the data since the first axis indexes real and imaginary components. A performance improvement of 50% is achieved.

Quadruple word aliasing involves absorbing another length two axis indexing even and odd components, handling the result as if it were of type double complex. Communication performance of send operations increased an additional 44% through this change. However, the double complex type is actually improper and any operation other than copying (such as complex addition) would produce erroneous results.

For the scan operations, double word aliasing was employed. Quadruple word aliasing will not work for the add scan because, as mentioned above, the data is improper. The copy scan used in the grid-to-spectral conversion is a candidate for quadruple word aliasing because arithmetic is not performed on the quadruple word elements. In this version, however, the copy scan has been

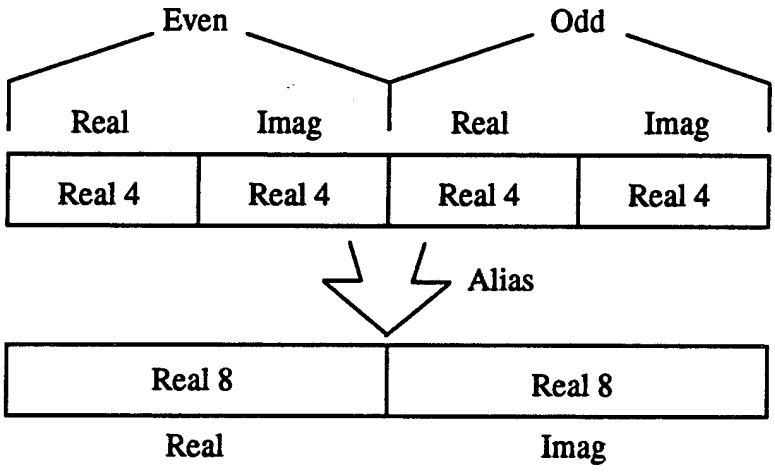

Fig. 8 — Aliasing Relationships

temporarily replaced by an add scan due to the copy scan's poor performance. Copy scan performance, currently limited by an especially inefficient implementation, is expected to increase in the near term. When performance improves, a quadruple word copy scan may then be advantageous.

4.3 FFT Shape Changes

Early experimentation with the FFT routines from the CM-5 CMSSL [3] revealed that the 18 vertical level FFTs which were done efficiently on the CM-200 were inefficient on the CM-5. This has been attributed to the algorithm used in the CM-5 FFT which essentially involves transposing the FFT axis so that it is serial. When the vertical axis length was increased from 18 to 20, a large performance improvement was observed, on the order of 200 times faster. Since the current need is to provide even greater numbers of levels, FFT performance is no longer problematic. Once the relationships between FFT axis length and other axis layouts are known, it is relatively easy to find a size which is efficient. In particular, multiples of 8 perform well for both T84 and T170 model resolutions.

4.4 Use of CMSSL BLAS Routines

The conversions from spectral to grid representation and back include expressions which can be cast as Basic Linear Algebra Subroutine (BLAS) level 2 operations (i.e., maxtrix-times-vector operations). Since CMSSL provides optimized routines for some of these BLAS routines, a performance gain is possible. Applications of level 2 BLAS routines were made in the forward and inverse Legendre transforms.

The basic CMSSL outer product operation is defined as $A = xy^T$ where A is a matrix result and x and y are column vectors. The CMSSL outer product can handle multiple instances of the outer product in a general way and hence can be applied in situations which appear quite different than the simple definition. The fragment of code shown in figure 9 which evaluates the summation

terms in the Legendre transform

$$F^l(k,\theta) = \sum_n f_n^l(k) p_n^l(\theta)$$

is such a case. In terms of shapes, the outer product is producing

$$(2, 2, \texttt{cmn2}, \texttt{ens}, \texttt{levs}) \otimes (2, \texttt{cmn2}, \texttt{ens}, \texttt{lats}) \Longrightarrow (2, 2, \texttt{cmn2}, \texttt{ens}, \texttt{lats}, \texttt{levs}).$$

The outer product is revealed in the last axes of the two operands, `fln` and `pln`, and the last two axes of the result, `flns`. Figure 10 shows the replacement code. The spread operation duplicates the $p_n^l(\theta)$ values to achieve a compatible shape for the CMSSL operation.

```
plntemp = spread(pln(:,:,:,:,section),dim=1,ncopies=2)
do k=1,levs
  do lts=1,lats
    flns(:,:,:,:,lts,k) = fln(:,:,:,:,k)*plntemp(:,:,:,:,lts)
  end do
end do
```

Fig. 9 — Baseline Code for Outer Product

```
plntemp = spread(pln(:,:,:,:,section),dim=1,ncopies=2)
call gen_outer_product_noadd(flns,fln,plntemp,6,5,5,5,ier)
```

Fig. 10 — Code for CMSSL Outer Product

A similar application of a CMSSL BLAS routine is possible in the inverse Legendre transform

$$f_n^l(k) = \sum_\theta F^l(k,\theta) p_n^l(\theta) w(\theta).$$

The BLAS matrix-times-vector routine is applied to perform the complete transform. In terms of simple vectors and matrices, the computation is $y = y + Ax$. The code fragment replaced is shown in figure 11. The new code fragment is shown in figure 12. In the code fragment, the role of the matrix is played by the `flns` array. The `k` subscript marks the axis acting as the matrix row and the `lts` subscript appears in the axis indexing columns. It is easy to verify correctness since the `lts` axis is summed out and does not appear in the result whereas the column axis `k` axis does appear in the result.

5 DETAILED RESULTS

This section gives more detailed performance results. Some of these results are calculated rather than measured. Values for the CM-200 were obtained by scaling performance results from an 18 level model to 32 levels. Experience on the CM-200 has confirmed that scaling to a larger number of levels is an accurate representation of performance, assuming sufficient memory is available.

```
plnwttemp = spread(plnwt(:,:,:,:,section),dim=1,ncopies=2)
do k=1, #levs
  do lts=1, #lats
    gln(ri,:,:,:,k) = gln(ri,:,:,:,k) +
&                     flns(ri,:,:,:,lts,k)*
&                     plnwttemp(:,:,:,:,lts)
  end do
end do
```

Fig. 11 — Baseline Code for Matrix-Vector Product

```
plnwttemp = spread(plnwt(:,:,:,:,section),dim=1,ncopies=2)
call gen_matrix_vector_mult(gln,flns,plnwttemp,5,6,5,5,ier)
```

Fig. 12 — Code for CMSSL Matrix-Vector Product

The number of floating point operations for a Legendre transform with truncation J with K levels and M latitude pairs is obtained as follows. For each element $F^l(k,\theta)$ computed from the equation

$$F^l(k,\theta) = \sum_{n=l}^{J} f_n^l(k) p_n^l(\theta)$$

there are four flops in each term of the summation because the quantity f_n^l is complex and p_n^l is real. The product $f_n^l p_n^l$ involves two operations and each term of the summation requires two operations. The sum is along the n axis of the triangle of figure 2 which, for a given value of l, has height $J-(l-1)$. Combining these facts, computation of $F^l(k,\theta)$ requires $4(J-l+1)$ flops. There are $KM(J+1)$ such elements giving a total Legendre transform requirement of

$$\begin{aligned} KM\sum_{l=0}^{J}\{4(J-l+1)\} &= 4KM(J+1)^2 - 4KM\sum_{l=1}^{J} l \\ &= 4KM(J+1)^2 - 2KMJ(J+1) \\ &= 2KM(J+1)(J+2) \end{aligned}$$

flops.

The inverse transform is similar to the forward Legendre except that a weighted p_n^l is used. Weights are applied to a separate copy of the p_n^ls so the same number of flops are required. The equation is

$$f_n^l(k) = \sum_{\theta} F^l(k,\theta)\hat{p}_n^l(\theta)$$

where $\hat{p}_n^l(\theta)$ is the weighted p_n^l.

The FFTs are assumed to require the well known figure of $5N\log_2(N)$ flops each. There are $2KM$ N-point FFT instances.

For $K = 32$, $M = 128$, $J = 170$, and $N = 512$, the Legendre transforms require about 2.41×10^8 flops (224 MFlops) and the FFTs require 1.89×10^8 flops or 176 MFlops. These values are used along with the times from figure 14 to obtain overall floating point rates shown in figure 15.

	256 FPUs		512 FPUs		1024 VUs
	Send	CC	Send	CC	CM-5E
Overall	2.08	0.90	1.49	.65	0.17
Forward Legendre	.80	.42	.50	.37	0.07
Form fln×pln	.06	.06	.03	.05	0.01
Scan	.23	.24	.22	.22	0.04
Send to grid	.51	.13	.25	.10	0.02
FFT to grid	.10	.10	.05	.05	0.01
FFT from grid	.10	.10	.05	.05	0.01
Inverse Legendre	1.08	.27	.88	.17	0.07
Send to spectral	.16	.13	.10	.09	0.01
Scan	.78	N/A	.70	N/A	0.06
Form fl×pln×wt	.14	.13	.09	.08	0.00

Fig. 13 — Timing Breakdown for T84, 32 Levels (secs)

	256 FPUs		512 FPUs		1024 VUs
	Send	CC	Send	CC	CM-5E
Overall	8.92	3.93	4.99	2.29	0.56
Forward Legendre	2.80	1.44	1.44	.96	0.23
Form fln×pln	.29	.27	.11	.16	0.03
Scan	.61	.62	.48	.48	0.10
Send to grid	1.91	.55	.85	.33	0.10
FFT to grid	.41	.40	.21	.20	0.05
FFT from grid	.42	.36	.21	.19	0.05
Inverse Legendre	5.23	1.54	3.12	.87	0.23
Send to spectral	.58	.60	.36	.40	0.03
Scan	3.56	N/A	2.25	N/A	0.19
Form fl×pln×wt	1.10	.94	.51	.47	0.01

Fig. 14 — Timing Breakdown for T170, 32 Levels (secs)

6 UNIMPLEMENTED ALTERNATIVES

Several optimizations were considered or prototyped but not implemented in the final version. The concepts behind these unimplemented changes are valid so some or all of them may be implemented at a later time when circumstances, primarily system software performance, improve.

	CM-200		1024 VUs
	256 FPUs	512 FPUs	CM-5E
Forward Legendre	155	233	974
Inverse Legendre	145	257	974
FFT	440	880	3520
Round Trip	243	349	1429

Fig. 15 — Performance Levels of Computational Elements (MFlops/s) at T170/32

6.1 Parallel Latitudes

An earlier version of the kernel for the CM-200 used a parallel latitude axis in the `flns` shape used during step 1 of the spectral to grid process shown in figure 3. This was found to be useful at the limits of scalability when the spectral truncation (or resolution) is small relative to machine size. Since the 256 PN CM-5 operates as if it has 1024 processors (the VUs), the CM-5 is actually a more parallel machine than the CM-200 owned by NRL. This means that the situation where parallel latitudes are beneficial is more common for the CM-5 than for the CM-200.

Parallel latitudes are implemented on the CM-200 using a send and copy scan approach. Parallel latitudes may also be implemented using the CM-5 CMSSL vector outer product routine. Due to the relatively poor CM-5 performance of both the scan and outer product, the cost of establishing the parallel latitudes axis is not competitive with the current serial latitude axis. Improved CMSSL and scan software may change this tradeoff in the future however.

6.2 Segmented Scan

The performance of segmented scan operations on the CM-200 was found to be about three times slower than unsegmented scan operations. In the process of improving performance for the CM-200, the segmented scan used for the Legendre transform was replaced by an unsegmented scan and a more expensive send. These operations correspond to steps 3-5 of figure 3. The send involved moving twice as many elements as for a segmented scan, thereby increasing the send time. However, the tradeoff favors the more expensive send operation and faster scan combination.

The tradeoff continues to be favorable on the CM-5E but there is a potential that the revised scan routines will change this. If the segmented scan becomes more efficient, either through a more efficient implementation or via the idea expressed in Section 6.4, it may be possible to return to a segmented scan and a less expensive send operation.

6.3 Gather/Scatter Operations

The best-performing version of the kernel on the CM-200 used the TMC Communication Compiler to optimize the routing of communication operations. While this was highly successful on the CM-200, the CM-5E communication system is autonomous and no such routing optimization is expected to be possible. The highest performance communication packages available

on the CM-5E are `sparse_util_gather`, `sparse_util_scatter`, `sparse_util_vec_gather`, and `sparse_util_vec_scatter`. Because of intrinsic shape requirements, these would be needed in pairs to replace a single send. TMC analysts have speculated that a performance gain may be possible but have also indicated that additional routines with more compatible shape requirements are forthcoming.

6.4 New Segmented Scan Concept

The relative cost of segmented scans compared to unsegmented ones has been an important factor in kernel optimizations both on the CM-200 and CM-5. On the CM-5 at least, the explanation for lower performance of segmented scans stems from the extra memory accesses to load segmentation information and from extra VU instructions to apply the segmentation masks to intermediate scan values. Of these two components the memory access is presumed to be larger.

The current segmentation capability is general in that each scan instance may have a separate segmentation structure (i.e., sequence of segment start bits). However, in the authors' experiences using scans, dating back to the original Connection Machine CM-1, most segmented scans have used the same segmentation structure across all instances. If the memory access of the segmentation information is a large component of the cost of doing segmentation, it may be possible to improve substantially segmented scan performance by using the fact that segmentation is often the same across all instances. The memory access for segmentation information would then be amortized across the instances.

Software changes to implement the additional scan capability should be rather low. A change in the application interface would also be required. This could be done by adding an additional parameter to the existing scan interface indicating that the segmentation structure is the same across instances. Alternatively, a new interface could be provided for the specialized version. This latter approach has the advantage that no existing user program would be affected.

6.5 Level-Combined Legendre and Fourier Transforms

It is possible to reduce the number of instances in the FFTs by combining pairs of them. This is a consequence of the fact than complex-to-complex FFTs are used when only conjugate-symmetric-to-real FFTs are required. Furthermore, the same approach can sometimes be used in the Legendre transforms.

Deferring implementation of these ideas is partly due to time and machine availability constraints. Additionally, since the relative speeds of FFT, scan, and send operations are critical implementation tradeoff parameters, a decision was made to defer these changes until the improved scan software is available.

Forward and inverse Fourier transforms may be defined as

$$F(f) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i f t} dt \text{ and}$$

$$f(t) = \int_{-\infty}^{\infty} F(f) e^{2\pi i f t} df.$$

Since the Fourier transform is linear, the transform of

$$F(f) = G(f) + iH(f)$$

is just

$$\begin{aligned} f(t) &= \int_{-\infty}^{\infty} F(f) e^{2\pi i f t} df \\ &= \int_{-\infty}^{\infty} (G(f) + iH(f)) e^{2\pi i f t} df \\ &= \int_{-\infty}^{\infty} G(f) e^{2\pi i f t} df + i \int_{-\infty}^{\infty} H(f) e^{2\pi i f t} df \\ &= g(t) + ih(t). \end{aligned}$$

When $g(t)$ and $h(t)$ are known to be purely real functions, the two transforms $g(t)$ and $h(t)$ can be separated from $f(t)$. This situation applies in the case of the spectral quantities of the forecast model but a complication arises in that half of the spectrum is implicit in the conjugate-symmetry stemming from the fact that $g(t)$ and $h(t)$ are real. To use packing, the full spectrum must be used, potentially doubling the amount of information sent.

The Legendre transform, like the Fourier transform,

$$F^l(\theta) = \sum_n f_n^l p_n^l(\theta)$$

is linear and hence amenable to packing of two separate transforms. Define

$$f_n^l = g_n^l + ih_n^l.$$

Then

$$\begin{aligned} F^l(\theta) &= \sum_n f_n^l p_n^l(\theta) \\ &= \sum_n \left(g_n^l + ih_n^l\right) p_n^l(\theta) \\ &= \sum_n g_n^l p_n^l(\theta) + i \sum_n h_n^l p_n^l(\theta) \\ &= G^l(\theta) + iH^l(\theta). \end{aligned}$$

The two components $G^l(\theta)$ and $H^l(\theta)$ can be recovered at this point through

$$\begin{aligned} \frac{1}{2}\left\{F^l(\theta) + \left[F^{-l}(\theta)\right]^*\right\} &= \frac{1}{2}\left\{G^l(\theta) + iH^l(\theta) + \left[G^{-l}(\theta) + iH^{-l}(\theta)\right]^*\right\} \\ &= \frac{1}{2}\left\{G^l(\theta) + iH^l(\theta) + \left[\left(G^l(\theta)\right)^* + i\left(H^l(\theta)\right)^*\right]^*\right\} \\ &= \frac{1}{2}\left\{G^l(\theta) + iH^l(\theta) + \left[G^l(\theta) - iH^l(\theta)\right]\right\} \\ &= G^l(\theta) \end{aligned}$$

and

$$\frac{1}{2i}\left\{F^l(\theta) - \left[F^{-l}(\theta)\right]^*\right\} = \frac{1}{2i}\left\{G^l(\theta) + iH^l(\theta) - \left[G^{-l}(\theta) + iH^{-l}(\theta)\right]^*\right\}$$

$$
\begin{aligned}
&= \frac{1}{2i}\left\{G^l(\theta) + iH^l(\theta) - \left[\left(G^l(\theta)\right)^* + i\left(H^l(\theta)\right)^*\right]^*\right\} \\
&= \frac{1}{2i}\left\{G^l(\theta) + iH^l(\theta) - \left[G^l(\theta) - iH^l(\theta)\right]\right\} \\
&= H^l(\theta)
\end{aligned}
$$

but there is a communication cost in combining the l- and $-l$-indexed elements.

The above techniques can be applied in the spectral-to-grid and grid-to-spectral transforms to reduce the amount of work needed. If two adjacent levels of a spectral field f_n^l are combined as above, the amount of work done in the scan and FFT can be cut in half. If the individual $F^l(\theta)$s are needed, however, extra communication is needed as shown above to separate the two Fourier components. There is an implementation tradeoff between increasing the scan speed by combining adjacent levels and the cost of the extra communication to separate levels before the FFT. In the grid to spectral direction, level-combining can be done on FFTs but not on the Legendre transforms.

In terms of the send operations, the changes include transmitting both non-negative and negative frequencies into the grid shape. This is no more work than the current approach since there are half as many levels. Furthermore, The current send operations differ in that the send from spectral to grid, which must transmit both beginning and ending values, involves twice as many total elements as during the inverse process. Current timings differ, however, by a ratio of about three to one rather than two to one. Thus, it is possible that, assuming performance differences are due to network congestion, sending half as many elements may be more than twice as fast. Separating the Fourier components after the forward Legendre transform but before an FFT to grid space involves a send from the grid shape into itself.

Figure 16 shows where level-combining can be used. The possibility of separating Fourier components from a combined Legendre transform is not shown.

To Grid	To Spectral
Spectral Coefficients	Spectral Coefficients
↙ ↘	↑
Packed Legendre / Unpacked Legendre	Unpacked Legendre
↘ ↙	↑
Packed FFT	Packed FFT
↓	↑
Grid Values	Grid Values

Fig. 16 — Opportunities for Level-Combined Legendres and FFTs

7 FUTURE PLANS

Optimizing the kernel routines is the first step in the continued development of the forecast model on the CM-5. Moving to the CM-5 platform from the CM-200 means more processing power and more memory. Also, the CM-5 will be the platform for continued hardware and software evolution for Thinking Machines Corporation whereas the CM-200 has been relegated to maintenance only.

We expect to make use of the increased memory size and computational power of the CM-5 in two ways. First, we will be able to run a version of the code which executes an ensemble of forecasts in parallel. Ensemble forecasting is an active research area which promises to improve the estimation of forecast uncertainty.

The second development thrust is to increase model resolution. Simply increasing resolution, however, is expensive because the timestep must also be decreased to maintain stability. Semi-lagrangian timestepping allows significantly larger timesteps and we have begun the large effort involved in implementing a semi-lagrangian capability in the model.

Thinking Machines Corporation is planning an improved implementation of the scan library used in the kernel. Scan operations are two of the largest components of the kernel run time so a reduction in their run times will mean a significant reduction in overall kernel execution time. Additionally, improved run time software is expected which will take advantage of the recent upgrade to CM-5E configuration. The net impact of system software upgrades is hard to quantify. We expect at least a 25% to as much as a 100% overall improvement in kernel performance.

One final area of future planning involves translation of the existing CM Fortran code to High Performance Fortran [5]. This promises to give a degree of portability to the model, allowing comparison of machine performance based on a single software benchmark.

8 CONCLUSIONS

The experience of optimizing the NMC weather kernel for the CM-5 has shown that the data layout chosen for the CM-200 continues to be a good one for the CM-5. This is reassuring for the long term viability of our code, especially given the significant architectural differences between the SIMD CM-200 and the inherently MIMD CM-5.

The largest change in the kernel, reordering axes, was made to bring the code into closer accord with the Fortran 90 standard. There is no inherent logical change in the code derived from the reordering. The new axis order may make it more feasible to run an efficient version of the kernel on a uniprocessor.

We await an opportunity to port and run the kernel on an MPP architecture from another vendor. Good performance on such a machine would lend support to efforts in parallelizing the code on Connection Machines.

9 ACKNOWLEDGMENTS

The authors wish to thank Dr. Henry Dardy of the NRL Connection Machine Facility, Dr. Joseph G. Sela of the National Meteorological Center, 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, **21**, April 1994, 140-149.