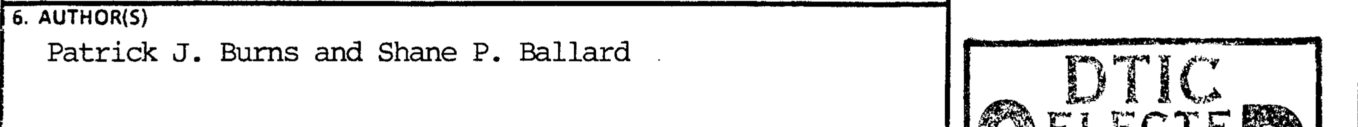
MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

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 this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 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
3/20/95

3. REPORT TYPE AND DATES COVERED
Final Report 4/13/92 - 11/30/94

4. TITLE AND SUBTITLE
Performance Enhancements in CEM VII, A Large Scale Discrete Event Simulation

5. FUNDING NUMBERS
DAAL03-92-G-0176

6. AUTHOR(S)
Patrick J. Burns and Shane P. Ballard

DTIC ELECTE JUL 05 1995 S B D

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Colorado State University
Department of Mechanical Engineering
Fort Collins, CO 80523

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U. S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ARO 29530.2-MA

11. SUPPLEMENTARY NOTES
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

The Concepts Evaluation Model VII (CEM VII) is the latest combat battle simulation in a long line of US army theater-level battle simulations. It has been developed under the auspices of the US army Concepts Analysis Agency (CAA) in Bethesda, Maryland. CEM VII is used by the CAA to access and optimize combat force capabilities. It can simulate months of theater land and air combat in a few hours on a super-computer.

CEM VII can take several hours of supercomputer CPU time to run, which can be very costly. The goal for this project was to increase the performance of CEM VII in order to reduce the run time. This goal was achieved through two steps. First, a higher degree of vectorization was incorporated into the kernel of CEM VII. Second, a process called "Data Packing" was eliminated from the computer code. The combined changes resulted in a 14.9 percent reduction in run time.

14. SUBJECT TERMS

DTIC QUALITY INSPECTED 5

15. NUMBER OF PAGES
79

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

19950630 142

A Structured Approach to Large-Scale Battlefield Simulation

Final Report

Shane P. Ballard and Patrick J. Burns

18 March 1995

U. S. Army Research Office

Contract DAAL03-92-G-0176

Department of Mechanical Engineering

Colorado State University

Fort Collins, CO 80523

Approved for Public Release:

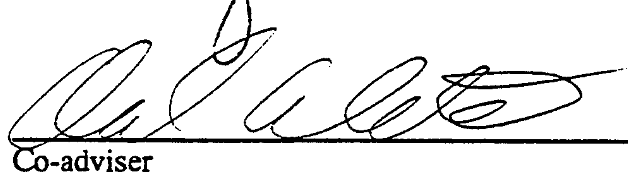
Distribution Unlimited

COLORADO STATE UNIVERSITY

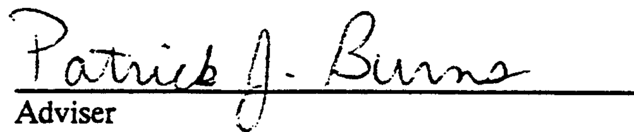
December 15, 1994

WE HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER OUR SUPERVISION BY SHANE PATRICK BALLARD ENTITLED, "PERFORMANCE ENHANCEMENTS IN CEM VII, A LARGE SCALE DISCRETE EVENT SIMULATION," BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE.

Committee on Graduate Work



Co-adviser



Adviser

Department Head

ABSTRACT OF THESIS

PERFORMANCE ENHANCEMENTS IN CEM VII,
A LARGE SCALE DISCRETE EVENT SIMULATION

The Concepts Evaluation Model VII (CEM VII) is the latest combat battle simulation in a long line of US Army theater-level battle simulations. It has been developed under the auspices of the US Army Concepts Analysis Agency (CAA) in Bethesda, Maryland. CEM VII is used by the CAA to assess and optimize combat force capabilities. It can simulate months of theater land and air combat in a few hours on a supercomputer.

CEM VII can take several hours of supercomputer CPU time to run, which can be very costly. The goal for this project was to increase the performance of CEM VII in order to reduce the run time. This goal was achieved through two steps. First, a higher degree of vectorization was incorporated into the kernel of CEM VII. Second, a process called "Data Packing" was eliminated from the computer code. The combined changes resulted in a 14.9 percent reduction in run time.

Shane Ballard
Department of Mechanical Engineering
Colorado State University
Fort Collins, CO 80523
Spring 1995

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

ACKNOWLEDGEMENTS

I would like to give special thanks to my Mother, Father, Aunt, and Grand Parents for their moral and financial support during my entire college carrier.

Additionally I would like to thank Dr. Patrick J. Burns for his guidance and allowing me the opportunity to work on this project. I feel that I have gained invaluable knowledge about high-performance computing, and large scale simulations.

I would like to thank the U.S. Armies Concepts Analysis Agency for the funding of this project, and specifically Bill Allison for his seemingly endless patience for my inquiries.

Finally I would like to thank my office mates Matt, Sukhi, John, and Charles for making the office a fun place to work. I give a special thanks to John Dolaghan for his answers to my seemingly endless questions about the trivialities of writing a thesis.

Table of Contents

Section Page

1. Introduction 1

1.1 Background 1

1.2 Discrete Event Simulation 3

1.3 Battlefield Schematic 4

1.4 Input Cases 7

1.5 Performance Issues in CEM VII 7

1.5.1 Vectorization 7

1.5.2 Data Motion 8

1.5.3 Input/Output (I/O) 8

1.5.4 Debugging 8

1.6 Overview of Remainder of Thesis 9

2. Monitoring Tools 10

2.1 Hardware Performance Monitor (hpm) 10

2.2 Flowview 10

2.3 CEMPlot 11

2.4 Total Kills 14

3. Analysis of Original CEM VII 18

3.1 Flowview 18

3.2 hpm 19

3.3 Approaches Taken 20

3.3.1 Vectorization of the CEM VII Kernel (ATCALC) 20

3.3.2 Elimination of Packing / Unpacking 21

3.3.3 Input/Output (I/O) 21

4. Vectorization of the ATCALC Subroutine 22

4.1 Concept of Vectorization 22

4.2 Description of ATCALC 25

4.3 Description of Direct Fire and Indirect Fire Calculations 27

4.3.1 Direct Fire 27

4.3.2 Indirect Fire Calculations 29

4.3.3 Sorting Subroutine (QKSRTI) 32

4.4 Approach 33

4.5 Vectorization of Direct Fire Subroutines 38

4.6 Vectorization of Indirect Fire Subroutines 40

4.7 New Sort Routine 41

4.8 Results 42

4.8.1 Total Kills of Vehicles 42

4.8.2 CEMPlot 43

4.9 Performance 44

4.9.1 hpm 44
4.9.2 Flowview 44
4.10 Vector Lengths in CEM VII 46
5. Elimination of Packing / Unpacking 49
5.1 Background 49
5.2 Approach 50
5.3 Results 55
5.3.1 Total Kills of Vehicles 55
5.3.2 CEMPlot 56
5.4 Performance 56
5.4.1 hpm 56
5.4.2 Flowview 57
6. Combination of "Vectorization" and Elimination of Data Packing 58
6.1 Results 58
6.1.1 Total Kills of Vehicles 58
6.1.2 CEMPlot 58
6.2 Performance 59
6.2.1 hpm 59
6.2.2 Flowview 59
7. Conclusions 62
8. Recommendations 64
References 66

List of Figures

Figure — Page

Figure 1.1 Typical Branching Structure in CEM VII 3

Figure 1.2 Battlefield and Combat Unit Deployment Schematic 4

Figure 1.3 Two-Dimensional View of Terrain 6

Figure 2.1 Initial Flowtrace Screen 11

Figure 2.2 Location of CEMPlot in the CEM Hierarchy 13

Figure 2.3 Loss Graphs Generated by CEMPlot 14

Figure 2.4 FEBA Overlaid on Terrain for Theater Cycle 0 15

Figure 2.5 FEBA Overlaid on Terrain for Theater Cycle 5 15

Figure 2.6 FEBA Overlaid on Terrain for Theater Cycle 10 16

Figure 2.7 FEBA Overlaid on Terrain for Theater Cycle 15 16

Figure 3.1 hpm Statistics for Original CEM VII 20

Figure 4.1 Vector Hardware Showing the Addition of Arrays A and B with Storage into Array C 24

Figure 4.2 Flow Chart of Original Firing Order in ATCALC 26

Figure 4.3 Depiction of "Gather-Scatter" in Vector Hardware 35

Figure 4.4 Flow Chart of Modified Firing Order in ATCALC 36

Figure 4.5 Flow Chart of Final Firing Order in ATCALC 37

Figure 4.6 Percent Difference in Total Kills of Vehicles Between Original CEM VII and Vectorized Version 42

Figure 4.7 Loss Graphs of Original and Vectorized Codes After 16 Theater Cycles 43

Figure 4.8 FEBA Location of Original and Vectorized Codes After 16 Theater Cycles 43

Figure 4.9 hpm Statistics for Vectorized Version of CEM VII 44

Figure 4.10 Performance of Indexed Gathers on Cray YMP 47

Figure 4.11 Floating Point Performance on Cray YMP 48

Figure 5.1 Typical Binary Storage of 750 and 1,234 49

Figure 5.2 Data Packing Storage of 750 and 1,234 50

Figure 5.3 hpm Statistics for "Unpacked" Version of CEM VII 56

Figure 6.1 Percent Difference in Total Kills of Vehicles Between Original CEM VII and Combined Vectorization and No Data Packing 59

Figure 6.2 hpm Statistics for Combined Version of CEM VII 60

List of Tables

Table — Page

Table 3.1 CPU Usage in Original CEM VII 18

Table 4.1 Sorted Indices from a Deterministic/Non-Deterministic Sort 33

Table 4.2 CPU Usage in Vectorized CEM Code 45

Table 5.1 Packed Arrays and Their Replacements 53

Table 5.2 CPU Usage in "Unpacked" Version of CEM 57

Table 6.1 CPU Usage in Combined Version of CEM 60

Table 7.1 Increase in Performance Resulting from Changes 62

Nomenclature

A_{ik}	Target availability factor for type i shooters and type k targets.
$(DEMAND)_i$	Total demand for fire from type i shooters.
E_{ij}	Expenditure of rounds from j^{th} weapons on i^{th} shooters.
F_{ik}	The total number of rounds fired at type k targets by type i shooters.
$FRAC_{ijk}$	Fraction of type ij rounds which are associated with type k targets.
$(IM)_k$	The importance of the type k targets.
$(KL_k)_i$	Kills of type k targets by type i shooters.
L_{ijk}	Lethality parameter for shooter/target pair.
$(MUNPR)_{ij}$	Munition priority for type j weapons on type i shooters.
$(MUNPRS)_i$	Sum of biased munition priorities for type i shooter.
N_i	Total number of type i shooters.
N_k	Total number of type k targets.
P_{ijk}	Kills-per-round figure for shooter/target pair.
$(RATE)_i$	Number of rounds a type i shooter is able to fire during an engagement.
$(ROUNDS)_i$	Number of rounds fired by all type i shooters.
$(RSPNS)_i$	Response factor for type i shooters.
$(TGTPR)_{ijk}$	Target priority for type j weapons on type i shooters, shooting at type k targets.
$(TGTPRS)_{ij}$	Sum of target priorities of type j weapons on type i shooters.

Subscripts

i	Denotes type i vehicle.
j	Denotes type j weapon.
k	Denotes type k target.
k'	Denotes target types with priority greater than type k

Theme of the Work

"I shall be accused, I suppose, of saying that no event in war can ever occur which may not be foreseen and provided for. To provide the falsity of this accusation, it is sufficient for me to cite the surprises of Cremona, Bergop-zoom, and Hochkirch. I am still of the opinion, however, that such events even as these might always have been anticipated, entirely or in part, at least within the limits of probability or possibility."

Baron de Jomini, General and Aid-de-Camp of the Emperor of Russia, *The Art of War*, 1862 (trans. by Capt. G. H. Mendell and Lieut. W. P. Craighill).

1. Introduction

1.1 Background

The Concepts Evaluation Model VII (CEM VII) is a theater level, discrete event, battle simulation that exists under the auspices of the US Army Concepts Analysis Agency (CAA), located in Bethesda, Maryland. "CEM VII is a fully automated, deterministic combat simulation that can simulate months of theater land and air combat in a few hours on a computer" [Allison, et. al., 1985]. CEM VII is used by the CAA to assess and optimize combat force capabilities. CEM VII originated in 1968 as the Theater Combat Force Requirement Model (TCM) developed by the Research Analysis Corporation. TCM was designed to provide combat capabilities and requirements which would be sensitive to the state of units on both sides of a battle. TCM was then modified to include force evaluation and to satisfy the needs of an army project called "Conceptual Design for the

Army in the Field (CONAF)," creating the CONAF Evaluation Model I (CEM I). Over the next six years the model was improved several times. In 1974 the project was turned over to the Army and renamed Concepts Evaluation Model IV. With the emergence of a radically different theater defense concept for Europe, CEM IV was improved again and renamed CEM V, which was studied by the CAA from 1979 to 1983. In 1983 CEM V evolved to CEM VI with the onset of new methodologies for calculating combat losses, ATCAL (An Attrition Model Using Calibrated Parameters) [Jones, H.W., 1991]. The final evolution of the CEM program encompassed two major improvements. First, the process of assigning supplies to each side was changed. Second, rather than the use of lookup tables to determine how far a unit could advance, a "defenders advantage" was created through the analysis of historical data, thus creating the Concepts Evaluation Model VII (CEM VII).

The CEM program is actually comprised of a pre-processor, a main program, and a post-processor. The pre-processor initializes the terrain information, troop positions, and force capabilities. After the main program completes, the post-processor is used to create output files which are easier to interpret. All in all, the program is comprised of approximately 44,500 lines of FORTRAN programing code. Because the main program can take several hours of CPU time, and computer time can be very costly, this project was undertaken in the Mechanical Engineering Department at Colorado State University in an attempt to increase the performance of the CEM VII program. With improved performance, CPU time will be reduced. A reduction in run-time will allow the CAA to perform additional simulations to achieve a greater degree of optimization. Additionally, the convergence tolerances in the code can be reduced to obtain optimal results, but still allow a reduced

run-time. The remainder of this thesis is devoted to reporting the processes undertaken and the results obtained in an attempt to increase performance.

1.2 Discrete Event Simulation

The evolution of a CEM VII solution is very complex in that the structure of the solution evolves with the simulation. The progression of the simulation is dependent upon the input at the beginning of the simulation, with intermediate results influencing the remainder of the simulation. Even though there are multiple branching levels, each containing multiple constraints, the simulation is deterministic in that, with the same starting input, the same results are obtained. Still, the CEM VII model does have branching *a priori* unknown, in that the evolution of the structure of the simulation can be different from one simulation to another, depending on the input data, i.e. the structure of the simulation is input driven. Figure 1.1 depicts this evolutionary structure.

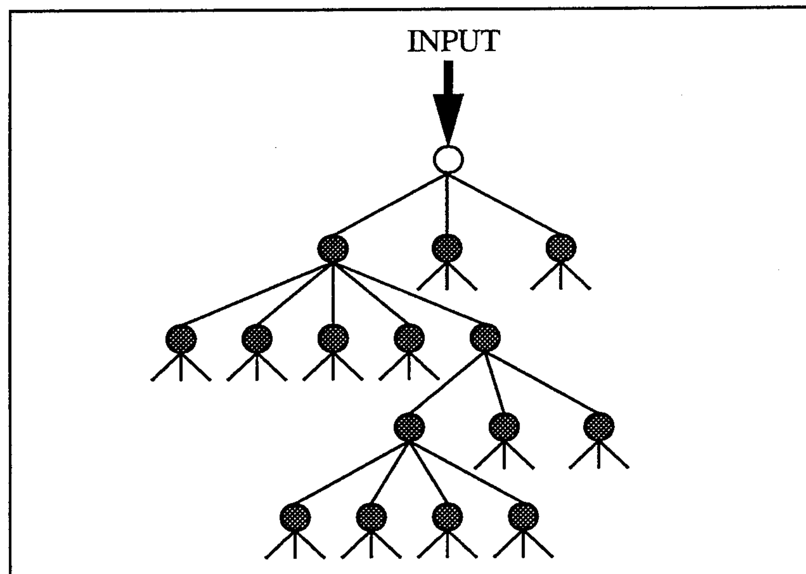


Figure 1.1 Typical Branching Structure in CEM VII

Each of the nodes in the Figure represents a point at which the structure and/or path of the simulation can change, depending on the state at that node.

1.3 Battlefield Schematic

Figure 1.2 shows how the battlefield is laid out. Distances in the direction of force movement (vertical) are measured in hectometers[1], while the smallest transverse (horizontal) distance is a minisector, with minisectors numbered successively from left to right, where the number of minisectors is dependent on the input case, and have no real dimension.

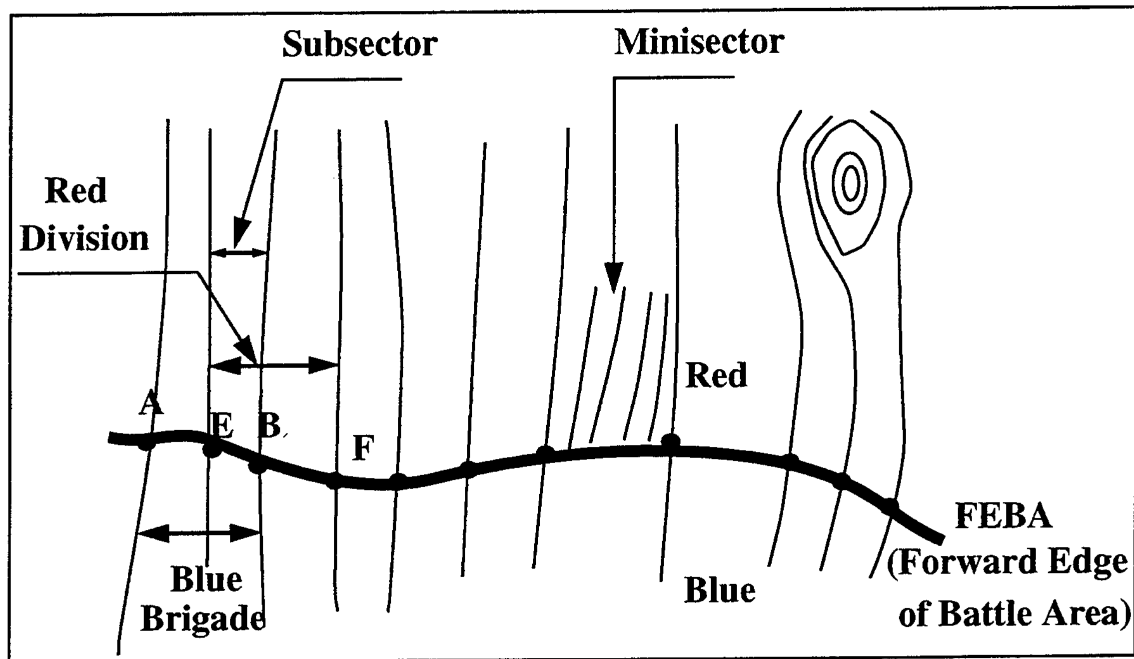


Figure 1.2 Battlefield and Combat Unit Deployment Schematic

The entire area that the battle spans is called the theater. Forces in the theater are defined as distinct units of blue brigades and red divisions, which are separated by the Forward

1. 1 hectometer = 0.1 kilometer

Edge of Battle Area (FEBA). Brigades and divisions are positioned along the FEBA such that the boundaries between themselves and their neighbors are on minisector boundaries. In Figure 1.2 a blue brigade exists between points A and B along the FEBA, while a red division exists between points E and F. Engagements in CEM VII are performed across subsectors. Subsectors are defined roughly as the smallest distance in which two opposing forces overlap. In Figure 1.2 , this is depicted as the distance along the FEBA from point E to point B. This causes subsector boundaries to coincide with minisector boundaries, and a subsector may contain many minisectors. Thus in Figure 1.2 , a portion of the Blue Brigade is fighting an engagement with a portion of the Red Division from points E to B. The remainder of the Blue Brigade (from A to E) is fighting in another engagement. For this reason any given combat unit in the theater may be divided between multiple engagements.

This whole structure is overlaid on a diverse terrain which is partitioned into four types. The four terrain types, A, B, C, and D, represent the general nature of the terrain and influence the mobility of the combat units. Using a package called CEMPlot, developed by Shivaswamy [1994], Figure 1.3 depicts the terrain on which a typical battle may be fought. CEMPlot will be discussed in more detail subsequently. However each colored rectangle represents a terrain type. Terrain type A (green) represents mainly flat land. Type B (blue) represents rolling hills. Type C (yellow) is indicative of mountainous terrain, while type D (black) represents some major obstacle, whether it be a river, lake, marsh, or man-made barrier.

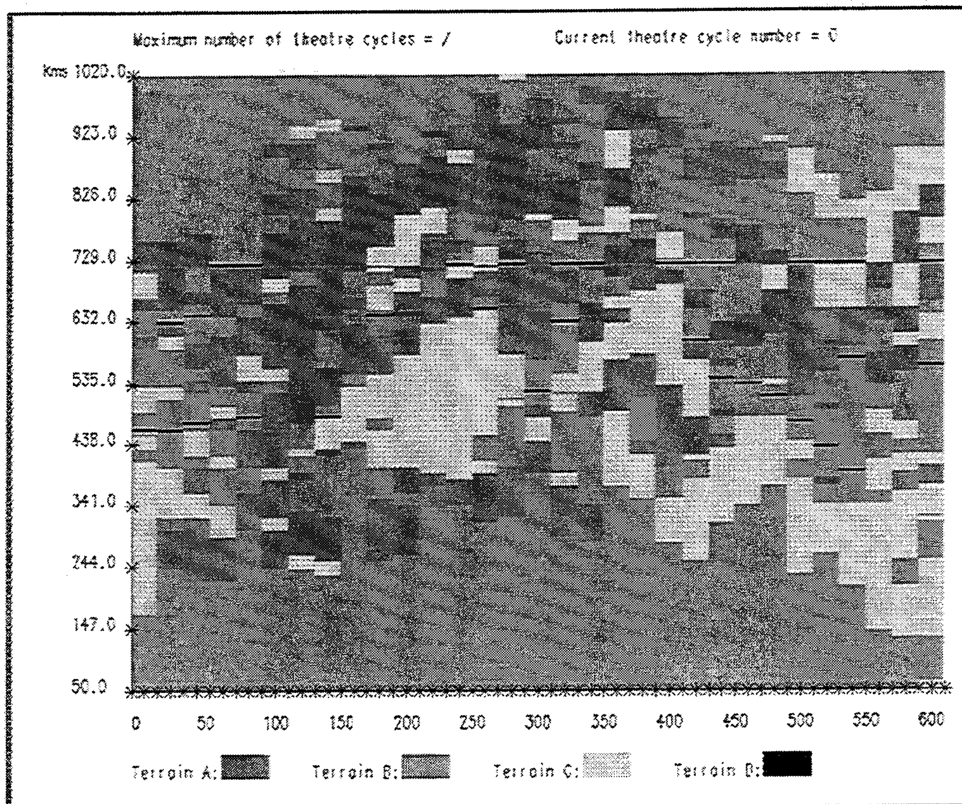


Figure 1.3 Two-Dimensional View of Terrain

In the CEM program there are several hierarchical levels at which calculations are performed. First, at the top of the hierarchy, there are theater cycles, which represent the largest time step. Within each theater cycle there are army cycles, where decisions involving units at the granularity of an army are made. Each army is made up of subordinate corps. As a result there are corps cycles within army cycles, where calculations are made about corps. Each corps has subordinate divisions beneath it, creating division cycles within a corps cycle, where calculations are made about individual divisions. While no further subdivisions of a Red Division is allowed, each Blue Division can have up to three subordinate brigades.

1.4 Input Cases

Individual battle scenarios are simulated with the CEM VII code with different input data. A single file, read by the pre-processor, contains all of the data required to start a simulation. Typical data found in this file are terrain information, number of weapons in each battle unit, and the hierarchical structure of the battle units at the beginning of the war. Additionally, there is a set of Killer-Victim (K/V) scoreboards for each battle scenario. The "K/V scoreboards" are used in attrition calculations, and are developed through a separate program, which takes into account many parameters, to create statistical data about a battle. The K/V scoreboards are given in matrix form: for example, for each weapon on each side there is a probability of a kill for each vehicle on the other side.

1.5 Performance Issues in CEM VII

There are four main issues of importance associated with the performance of CEM VII. These are vectorization, data motion, input/output (I/O), and debugging.

1.5.1 Vectorization

With the advent of advanced computer architectures, namely machines using vector hardware, vectorization of the CEM code may play a significant role in the performance. At the moment, many CEM simulations are run on Cray supercomputers. The advantage of the Cray architecture is the use of the vector hardware. If the same calculation will be performed many times, but on different data, the Cray architecture allows the data, if contiguous in memory, to "stream" through the machine's hardware. By doing this, the

instructions required to perform a calculation need be loaded only once, rather than every time the calculation is to be performed. This concept will be discussed further in Chapter 4.

1.5.2 Data Motion

Data motion involves the movement, and possibly unnecessary calculations on data. A concept that will be discussed in chapter 4, called gather-scatter, is a prime example of data motion. This process involves extra movement of data in order to use vector hardware. Often, vector hardware will perform calculations on all data in an array when only a subset of the data are required. This is done in order to keep the data contiguous in memory. Another concept that can be considered data motion is called "data packing." This concept will be discussed further in chapter 5, but it involves a specific type of data storage that requires additional operations to access and store data.

1.5.3 Input/Output (I/O)

CEM VII produces many files that contain information relevant to the battle. This output takes extra CPU time to complete. This could be streamlined by using unformatted output, but the files created would not be directly readable. Because it is desirable to access intermediate results about the simulation in ASCII (readable) form, it is not advantageous to consider changing this portion of the code at this time. However, the I/O buffer size can be increased to amortize "writes" over more data.

1.5.4 Debugging

Debugging also plays an important role in the process of performance enhancement, as altering computer code does little good if correct results are not obtained. Debugging

allows the programmer to find where a problem is occurring in the event that the code is not functioning correctly. While the ability to debug a code should not affect performance, it is beneficial in assessing when changes have been made correctly.

1.6 Overview of Remainder of Thesis

Chapter 2 presents the tools that are used to assess the results of a CEM VII run, followed by the analysis of the original CEM VII program to determine where efforts should be focused. The following chapters discuss the approaches taken and the results that were obtained. Finally, the conclusions that were found are reported followed by recommendations for further work.

2. Monitoring Tools

In order to assess the performance and accuracy of the CEM code, some tools are required to aid in the analysis of a battle simulation. The following sections describe the tools used in this investigation.

2.1 Hardware Performance Monitor (hpm)

The Hardware Performance Monitor (hpm) is a feature existing in the UNICOS operating system found on most Cray supercomputers. It is used, as a program runs, to obtain information about floating-point operations, instruction issue and hold issue, instruction buffer fetches and I/O and CPU memory references, as well as other statistics. For purposes in this study hpm is used to obtain total CPU time, and millions of floating point operations per second (MFLOPS). The MFLOPS rating is the best interpretation about the use of the vector hardware on the Cray. A higher MFLOP rating will generally indicate that the vector hardware is being used more efficiently. Consequentially, a higher MFLOP rating can cause the program to run in less CPU time.

2.2 Flowview

Flowview is a program found on Cray computers that is used to display a file generated from a "flowtraced" program. Flowtracing is a compiler option that builds information about the CPU time spent in individual subprograms of a program. Flowtracing monitors

time spent in each subprogram, counts the number of times a routine is called, and constructs a dynamic calling tree for the program. The primary use of Flowview in this work is to discern where the CEM program is spending most of its CPU time. Figure 2.1 shows the screen that results from a Flowview run. From this initial window many options are available to view other statistics about the program.

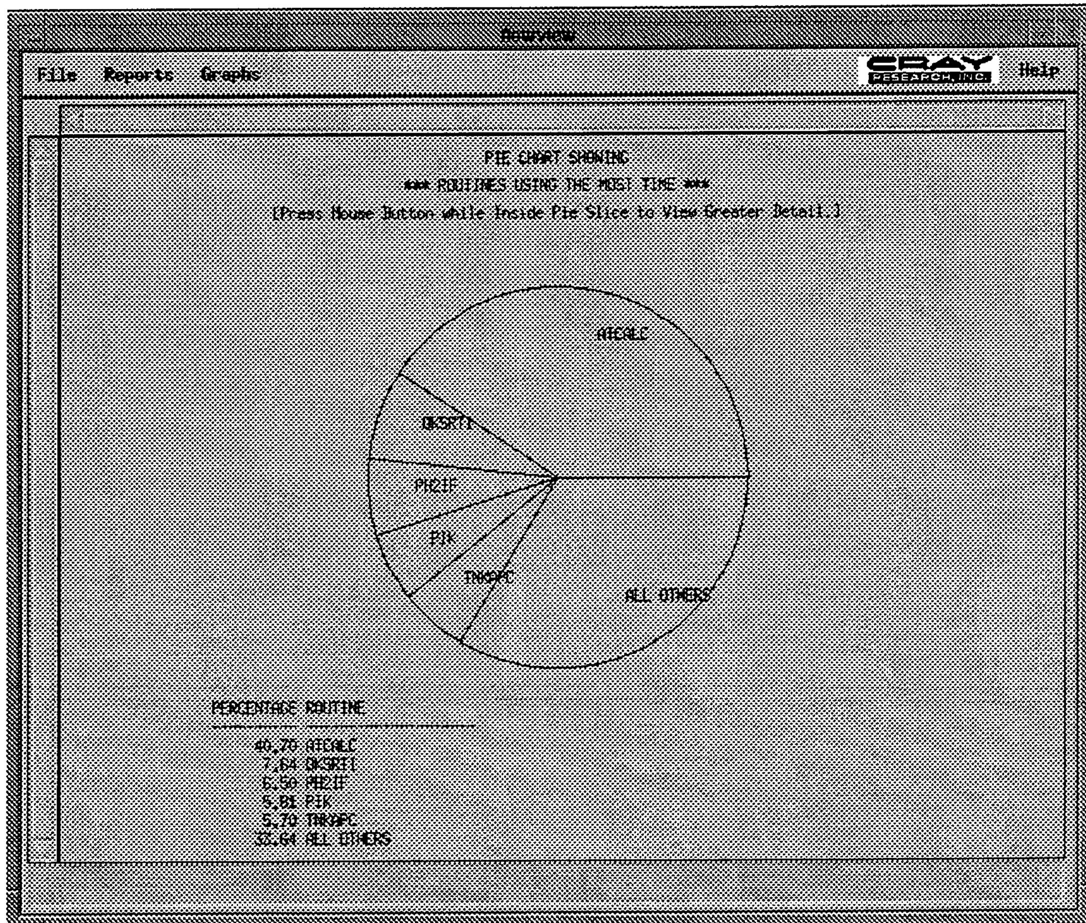


Figure 2.1 Initial Flowtrace Screen

2.3 CEMPlot

In order to have a visual means to assess simulation outcomes, the visualization program CEMPlot was developed by Shivaswamy [1994]:

> "CEMPLOT runs in the X Windows and OpenWindows environments and uses the XGL graphics library extensively (XGL 2.0). It uses a graphics toolkit XView to provide user-interface objects such as menu buttons. The program has the capability to display CEM output data in color and accepts information and commands from the user, interactively. The user-interface consists of a base frame or window which contains the graphics canvas (e.g., where the terrain and FEBA are plotted) and the input panel (control buttons). The button items include panel buttons which perform a function when clicked on, and menu buttons which allow the user to select a particular option from a menu."

CEMPlot is written in the C programing language, and has the following capabilities:

1. Display the battle terrain in two dimensions, or as a three dimensional fractal image.
2. Display the FEBA, overlaid on the two-dimensional terrain.
3. Display data from the tactical report. This includes number of personnel, artillery, helicopters, etc., for each unit assembled in each theater cycle.
4. Create a "Loss Graph," showing the use of ammunition, and loss of personnel for each theater cycle.
5. Present a roll-call showing all the units assembled along the FEBA, for both sides, at each theater cycle.
6. Perform geometric transformations, such as scaling and rotation, on the terrain/ FEBA.

7. Step through all theater cycles, in succession, creating an animation of the FEBA movement during the battle.

Figure 2.2 shows where CEMPlot resides in the CEM hierarchy, and the files that are used as input to CEMPlot.

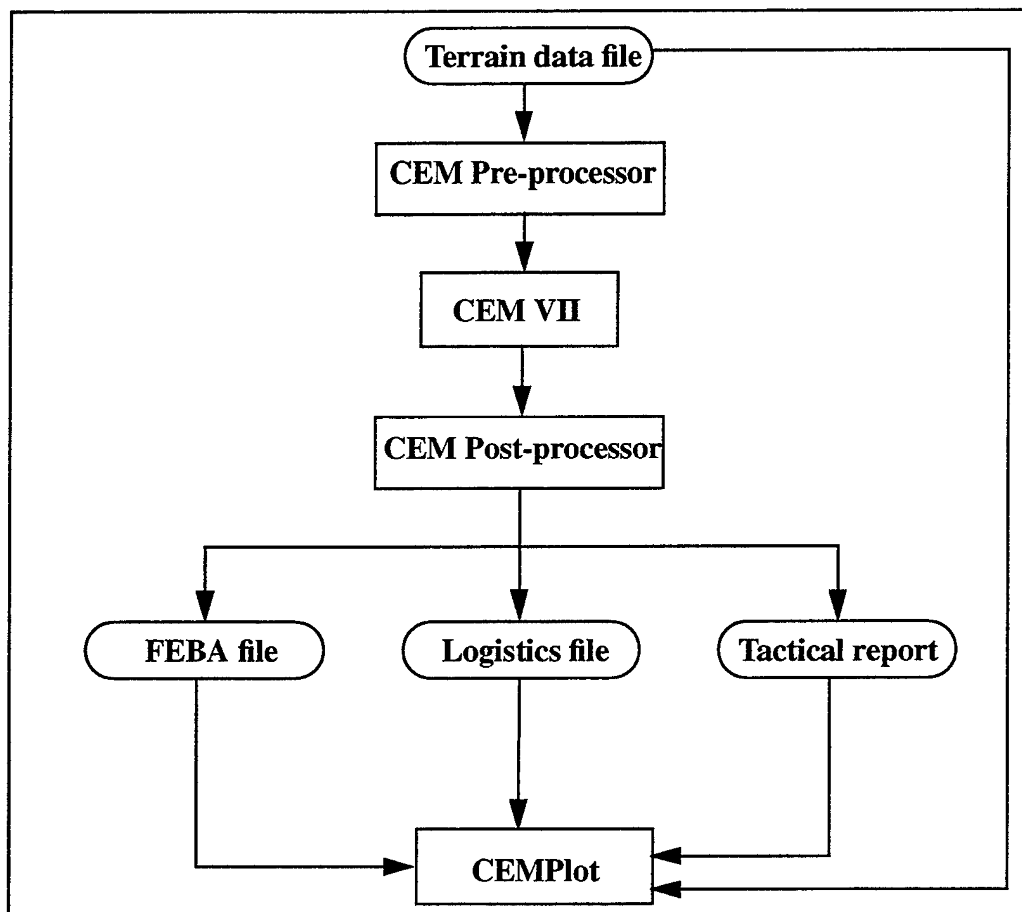


Figure 2.2 Location of CEMPlot in the CEM Hierarchy

The CEMPlot package is used primarily to determine whether or not the same results are being obtained after changes are made to the CEM code. The following figures show the "loss graphs" associated with the entire simulation, and the FEBA overlaid on the terrain, for different theater cycles of a battle simulation. Figure 2.3 depicts: 1) The number of

personnel lost by each side vs. Theater Cycle, and 2) The amount of ammunition used by each side vs. Theater Cycle. Figures 2.4 through 2.7 depict the FEBA at theater cycles 0, 5, 10, and 15 respectively.

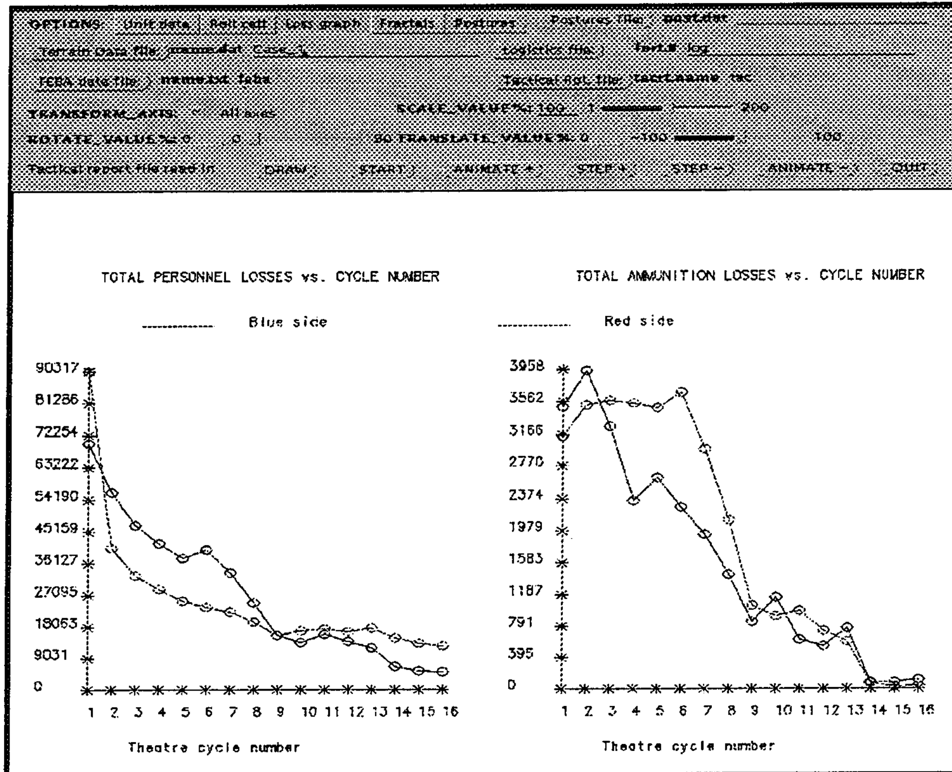


Figure 2.3 Loss Graphs Generated by CEMPlot

2.4 Total Kills

One last metric used to assess the validity of changes made in CEM VII is the total number of kills of equipment throughout the entire simulation. For comparison purposes the kills of each equipment type are accumulated throughout all engagements, over all theater cycles, and written to a file at the end of the simulation. If changes to the code are not done correctly, the error is likely to appear as differences in total kills of one or more of the equipment types. It is fairly safe to infer that identical kills of all equipment types suggests that performance changes to the code were done correctly.

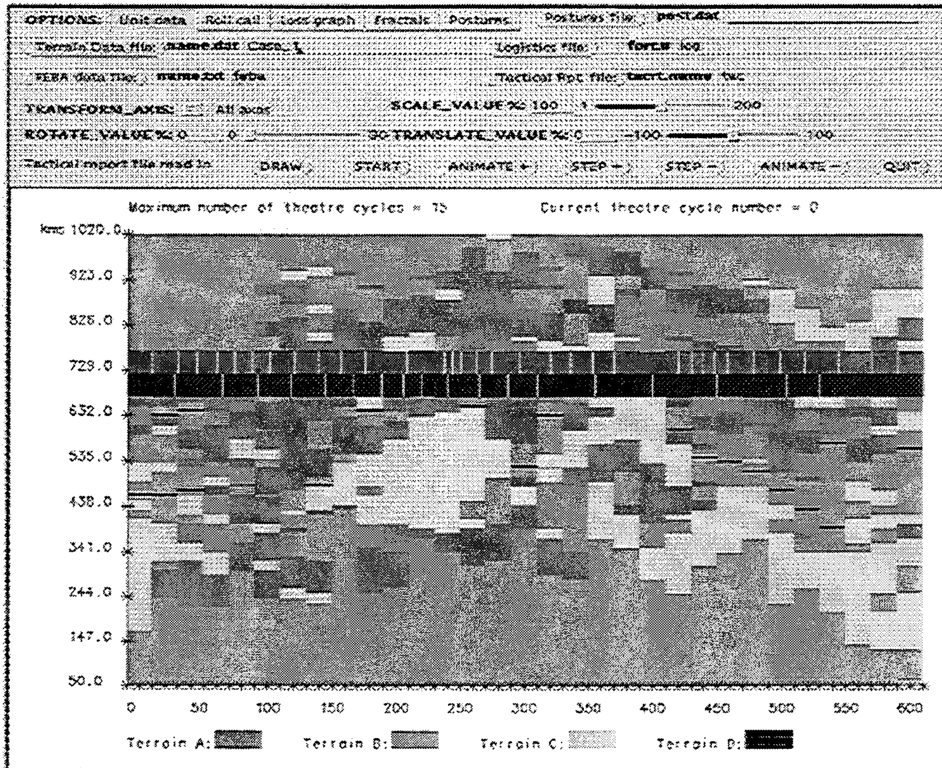


Figure 2.4 FEBA Overlaid on Terrain for Theater Cycle 0

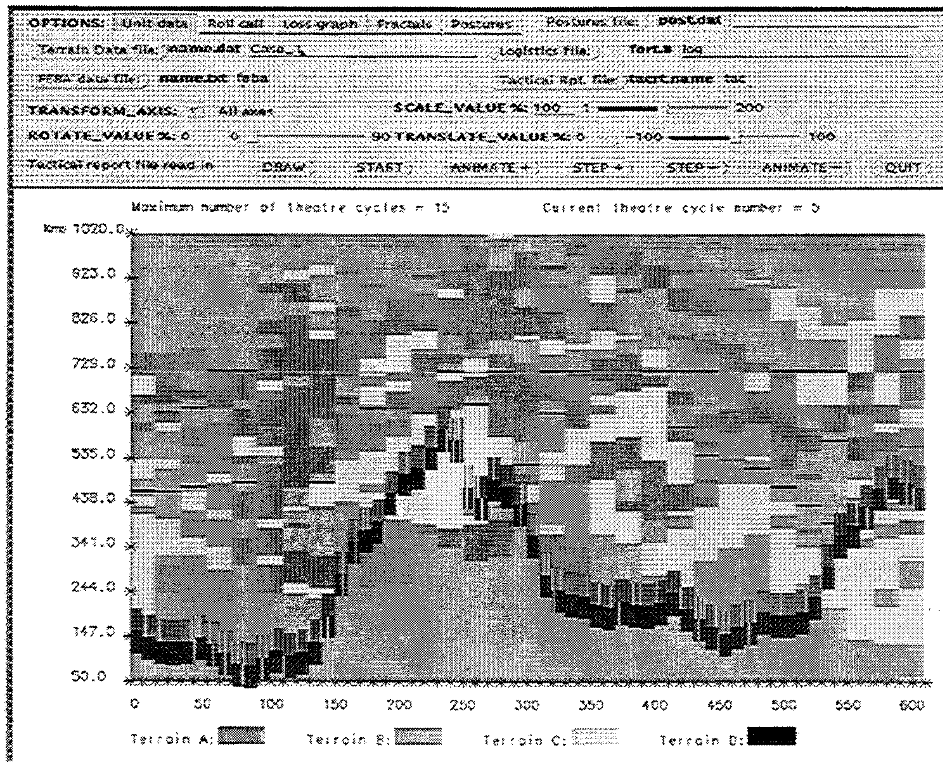


Figure 2.5 FEBA Overlaid on Terrain for Theater Cycle 5

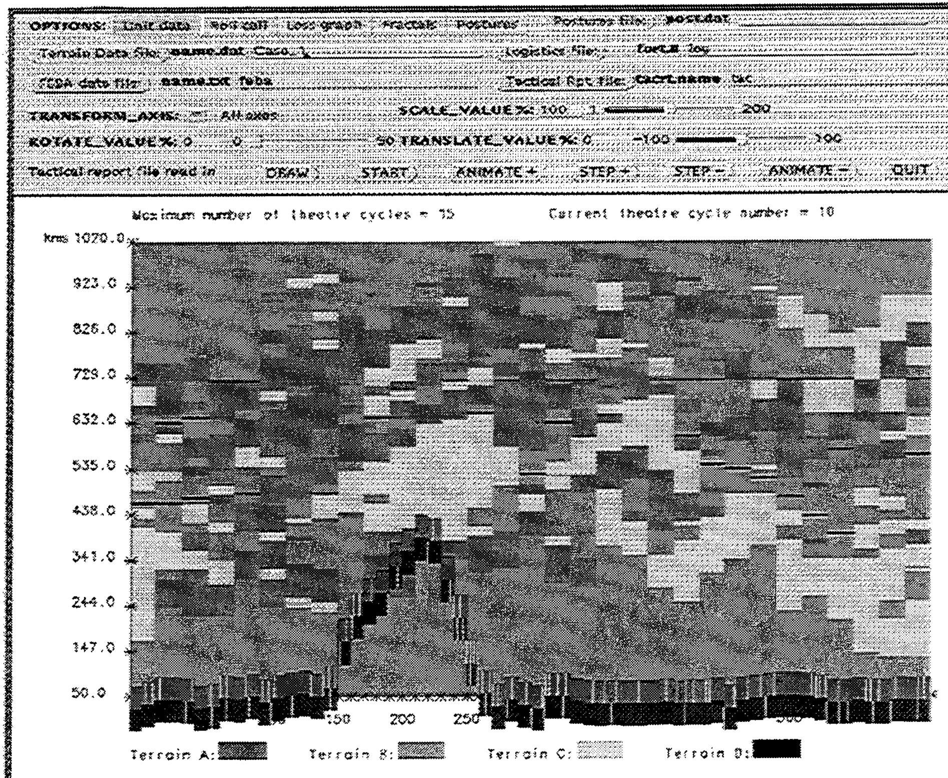


Figure 2.6 FEBA Overlaid on Terrain for Theater Cycle 10

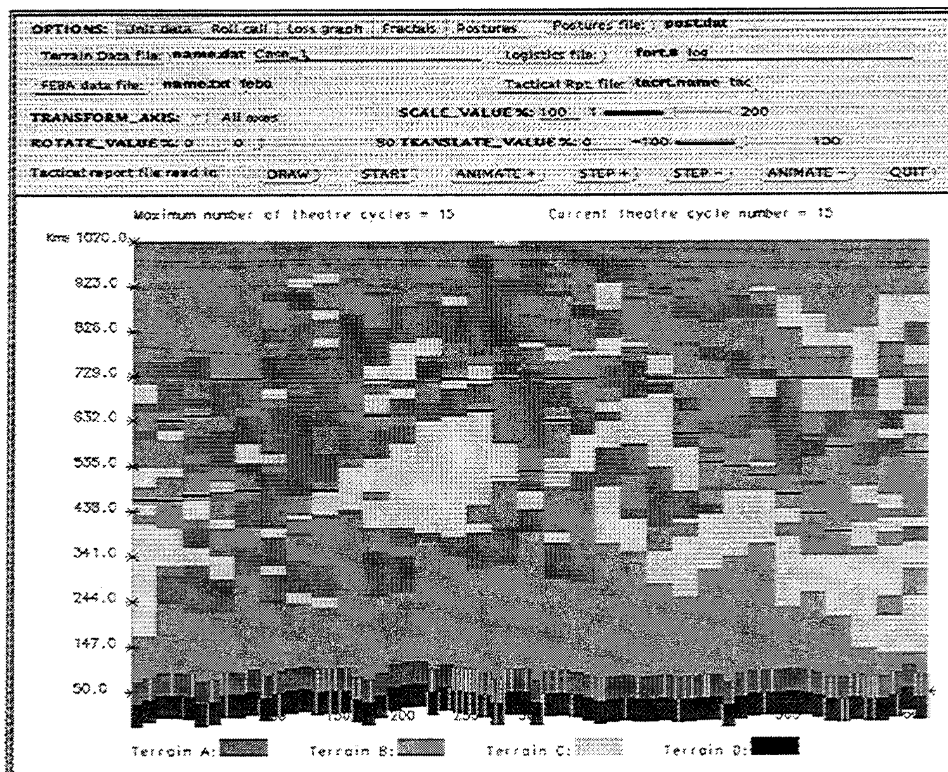


Figure 2.7 FEBA Overlaid on Terrain for Theater Cycle 15

The term equipment is rather broad. There are seven distinct categories of equipment in the CEM program. These are (1) tanks or air defense, (2) light armored personnel, (3) helicopters, (4) armored personnel carriers, (5) dismantled infantry, (6) artillery pieces, and (7) close air support. Each equipment type can have up to 2 weapons on it. Hereinafter, equipment types will be referred to as vehicles. Altogether there are up to 51 vehicle types available for each side, for a total of 102 vehicle types in the entire theater. Because there can be 2 weapons on each vehicle, there are 102 Blue weapon types and 102 Red weapon types, totaling 204 weapon types in the entire theater.

3. Analysis of Original CEM VII

In order to determine where effort would be best spent, the original CEM VII code was studied extensively to determine which areas of the code were consuming the largest portions of CPU time. This was done by compiling the original CEM VII code with flowtracing on, and running it with a typical input file for 16 theater cycles.

3.1 Flowview

Flowview was used to view the flowtrace statistics about the CEM VII program. Table 3.1 shows the results obtained from a 16 theater cycle run of the original CEM VII code. The Table contains information on the CPU usage of the top twenty subroutines. It is arranged

Table 3.1 CPU Usage in Original CEM VII

Flowtrace Statistics Report Showing Routines Sorted by CPU Time (Descending) (CPU Times are Shown in Seconds)					
Routine Name	**Total Time**	**Number of Calls**	**Average Time / Call**	**Individual Percentage**	**Cumulative Percentage**
ATCALC	2.46E+02	17281	1.43E-02	40.70	40.70
QKSRTI	4.63E+01	417656	1.11E-04	7.64	48.35
PH2IF	3.93E+01	645782	6.09E-05	6.50	54.84
PIK	3.52E+01	7412552	4.75E-06	5.81	60.65
TNKAPC	3.45E+01	68905	5.01E-04	5.70	66.36
PH2DF	3.11E+01	2827544	1.10E-05	5.13	71.49
ARTCAS	3.09E+01	17281	1.79E-03	5.10	76.59
RNDFRD	2.02E+01	17281	1.17E-03	3.33	79.92
DIVRPT	8.45E+00	128	6.60E-02	1.40	81.32
ASSESS	7.42E+00	128	5.80E-02	1.23	82.55
CASL	7.09E+00	17281	4.10E-04	1.17	83.72
ESTOUT	6.39E+00	85573	7.47E-05	1.06	84.77
PQMOD	6.29E+00	95730	6.57E-05	1.04	85.81

Table 3.1 CPU Usage in Original CEM VII

Flowtrace Statistics Report Showing Routines Sorted by CPU Time (Descending) (CPU Times are Shown in Seconds)					
Routine Name	**Total Time**	**Number of Calls**	**Average Time / Call**	**Individual Percentage**	**Cumulative Percentage**
UNITS	6.14E+00	37359	1.64E-04	1.01	86.82
PAK	5.73E+00	1056240	5.43E-06	0.95	87.77
STAMAT	4.83E+00	95730	5.05E-05	0.80	88.57
CINDEX	4.79E+00	1658274	2.89E-06	0.79	89.36
ITER	4.77E+00	102756	4.64E-05	0.79	90.15
DDEND	4.28E+00	128	3.35E-02	0.71	90.86
DBSTAT	3.92E+00	14613	2.68E-04	0.65	91.50

so that the first subroutine reported is using the most CPU time, while the last is using the least CPU time. The first column indicates the subroutine being reported, while the second column reports the total time that was spent in this subroutine for the entire simulation. The next column reports the number of times that the subroutine was called, followed by the average time spent in that subroutine per call. The last two columns report the individual percentage of time used by this subroutine and the cumulative percentage of time used in this and all previous subroutines.

3.2 hpm

For future comparison purposes, Figure 3.1 shows the hpm statistics reported for the original CEM VII. The numbers are for a sixteen theater cycle run. Note that the original code runs in 541.5 seconds, at an execution rate of 5.78 Mflops.

```
clark 174 % hpm -g0 cem2.2 <UNIT9> out1
WAR      HALT     ED A     FTER      THE      ATER      CYC      LE       16
CEM      EXIT     ING      NORM     ALLY
 STOP executed at line 12683 in Fortran routine 'EOW'
 CP: 541.495s,  Wallclock: 674.656s,  40.1% of 2-CPU Machine
 HWM mem: 1428439, HWM stack: 2048, Stack overflows: 0
Group 0:  CPU seconds    : 541.49     CP executing     :  90249146909

Million inst/sec (MIPS) : 43.56      Instructions     :  23586742175
Avg. clock periods/inst : 3.83
% CP holding issue      : 55.77      CP holding issue :  50333560050
Inst.buffer fetches/sec : 0.36M      Inst.buf. fetches:    193589501
Floating adds/sec       : 3.95M      F.P. adds        :   2137915043
Floating multiplies/sec : 1.65M      F.P. multiplies  :    892496208
Floating reciprocal/sec : 0.18M      F.P. reciprocals :     98213963
I/O mem. references/sec : 0.65M      I/O references   :    353147685
CPU mem. references/sec : 18.15M     CPU references   :   9825736819

Floating ops/CPU second : 5.78M
```

Figure 3.1 hpm Statistics for Original CEM VII

3.3 Approaches Taken

3.3.1 Vectorization of the CEM VII Kernel (ATCALC)

Using the Flowtrace statistics found in Table 3.1, it is apparent that subroutine ATCALC consumes the largest amount of CPU time. ATCALC is the kernel of the program, and is responsible for performing individual engagements over subsectors. Also, in the top twenty time-consuming subroutines are QKSRTI, PH2DF, and PH2IF. All three of these subroutines are called in ATCALC. The combined use of CPU time for these four subroutines is roughly 60% of the total CPU time used in this simulation. Since this is a majority of the time spent, beginning work was focused on these subroutines. Specifically, attempts were made to introduce a higher degree of vectorization in ATCALC. Chapter 4 details the work that was done in this regard.

3.3.2 Elimination of Packing / Unpacking

Viewing Table 3.1 once again, the reader will notice the subroutines PIK, PAK, and CINDEX are also within the top twenty time-consuming subroutines. These three subroutines are used to perform "Data Packing." "Data Packing" is a process that uses significant CPU time, but is no longer warranted on modern computers with large memories. Additionally, elimination of this process will make it easier to port the code to different computer architectures. As a result this was also addressed. Chapter 5 will discuss this further.

3.3.3 Input/Output (I/O)

Initially is was believed that the large amounts of output were causing the CEM program to run more slowly. For this reason I/O was studied. ATCALC, being the kernel of the program, is responsible for most of the output. To determine if I/O was a prime reason for the poor performance in CEM, all output was eliminated from ATCALC. Running this new version resulted in a run time that was not significantly different from the original. This indicated that I/O is not responsible for greatly reducing the performance of the CEM VII program.

4. Vectorization of the ATCALC Subroutine

4.1 Concept of Vectorization

To obtain a full understanding of vectorization would require a complete paper in itself, or a class on vectorization. This section will give the reader a fundamental overview of the concept. The driving force behind vectorization is to perform all like computations using a single instruction. This requires that the commands to perform a calculation need be loaded only once, instead of loading them each time the calculation is performed. Vectorization on a Cray requires that calculations be performed inside a loop. Rather than stepping thought the calculations in a loop sequentially, where any line in the loop is executed the number of times that the loop is executed, each line in the loop is executed once for all the data associated with that line. In this regard, the loop is actually executed only a single time. To illustrate this, consider the following FORTRAN code.

```
Do i = 1 , 50
    A(i) = B(i) + C(i)
    D(i) = A(i) * E(i)
    F(i) = A(i) + D(i)
End Do
```

If run on a machine that does not have vector hardware, this loop will be performed 50 times, and has three computations to be calculated each time through the loop. If run on a CRAY supercomputer, the loop will be "vectorized," and calculations will be done in a

different manner. The loop will actually be executed once, and all calculations associated with each line will be done when that line is executed. When the first line is executed arrays "B" and "C" are loaded into vector registers. These registers are then added in the vector hardware, and the results are then stored in a third vector register if they are to be used subsequently, else they are stored in the memory locations reserved for vector "A." This allows the first element of the B array to be added to the first element of the C array, and the resultant is stored in the first element of the A array. Then the second element of the B array is added to the second element of the C array, and the result is placed in the second element of the A array. This is repeated 50 times; once for each addition that is requested. Actually, the second add starts before the first add is finished. The number of adds that are being performed at once depends upon the number of instructions required to perform an add. This "streaming" of data through the vector hardware is called "pipelining."

Machine language is more complex than high level programing languages like FORTRAN. A simple command like "+" may actually require several operations in machine language. When the first line of the example loop is executed, the commands are loaded. Imagine that an add requires four "ticks" to complete the instruction. The vector registers are loaded and begin to "stream" through. Initially, the first element of each array is positioned in the first operation. After the first tick, these first elements move onto the second operation, and the second elements of each array move into the first operation. This sequence is continued until the first elements are completed by the fourth operation, and the fourth elements have been processed by the first operation. Figure 4.1 depicts this process.

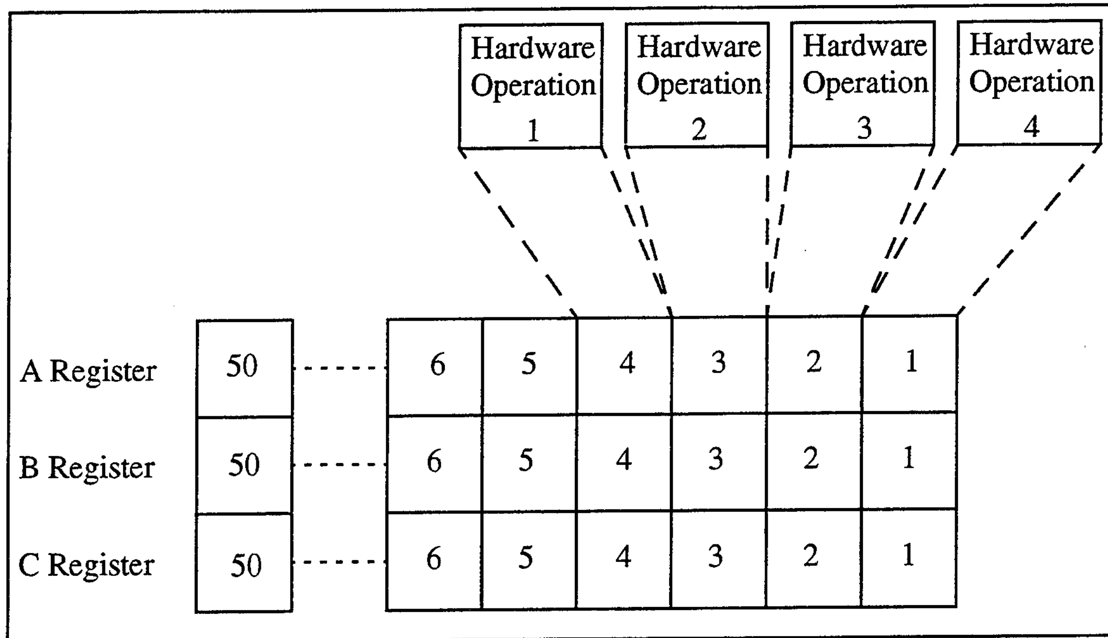


Figure 4.1 Vector Hardware Showing the Addition of Arrays A and B with Storage into Array C

The elements continue to "stream" through until all array elements have been processed by all commands. The next line of the loop is then executed. Here, the "A" array is already in a vector register, and the "E" array must be loaded into the vector registers. All A elements are multiplied by all E elements and the results D, are stored in a vector register. This process is repeated for each line inside the loop. The real performance is gained through the multiple calculations being performed at once.

There are several situations that will keep a loop from vectorizing. A prime example is data dependency. This occurs when a piece of information is required before it is computed. If a computation needs a value in A, but the A values are computed in a line farther down the loop, the loop will not vectorize. The following FORTRAN code illustrates such a data dependency.

```
Do i = 5, 50
```

```
    C(i) = A(i-1) + B(i-1)
    A(i) = C(i) + D(i)
End Do
```

If all C(i)'s are calculated before the A(i)'s, then value for A(i-1) will not be available for use in the calculation of the C(i)'s. If the programer knows that it is acceptable to use the data that are in the A array before A is updated, a command can be given to ignore the data dependency and the loop will be vectorized.

In order for a compiler to create an executable loop that will vectorize, the compiler must have a "template" that it can follow. This, in effect, is a set of instructions that the compiler is to follow when it encounters a certain structure. Therefore the compiler can only vectorize pieces of code for which it has such "templates" defined.

4.2 Description of ATCALC

ATCALC is the kernel of CEM VII, and performs the individual engagements occurring across subsectors. For each engagement in the original code, there are 102 Blue weapon types (each type of which is dealt with by number), which fire individually at 51 Red vehicle types. Attrition of the Red vehicles is then calculated. Afterward, the remaining 102 Red weapon types fire individually at the 51 Blue vehicle types, and attrition of the Blue vehicles is calculated. Since the algorithm is non-linear, this process is repeated, with updated vehicle numbers, until the number of all vehicles at the end of one iteration is within a certain convergence tolerance of the vehicles at the end of the last iteration. Figure 4.2 shows this structure graphically.

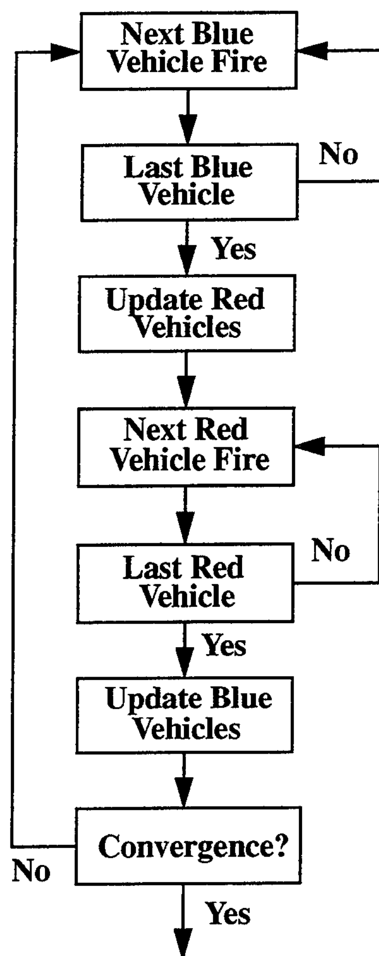


Figure 4.2 Flow Chart of Original Firing Order in ATCALC

The fact that the Red side is at a reduced state before it fires at the Blue side is commonly referred to as the “Blue Bias.” The CAA found, however, that the “Blue Bias” caused an engagement to converge in fewer interations, using less CPU time, while not affecting the results significantly. A study was conducted into the results that are obtained when the convergence tolerance is tightened. As was expected, tightening the tolerances required the program to perform more iterations which resulted in longer run times. Additionally, the results would change significantly, depending on the input case. Not only did the total kills of vehicles change, but the FEBA movement was affected as well.

Each vehicle can have two weapons types. Weapons may be either of a direct fire type or an indirect fire type. A two-dimensional array, BIAS(I,J), contains the criterion of whether a firing weapon is a direct fire or an indirect fire type. The variable I represents the vehicle number (1-102), while J represents the weapon on the vehicle (1 or 2). If the value of BIAS(I,J) is less than zero, the weapon is a direct fire type. If BIAS(I,J) is greater than or equal to zero, the weapon is indirect fire. The terms direct fire and indirect fire may be misleading. The difference between the two types stems from the type of attrition logic occurring at the target, not whether a line of sight exists to the target. For instance, a rifle might be considered a direct fire weapon, while a mortar might be considered an indirect fire weapon, even though a mortar operator may view the target. Depending on the weapon type, either a direct fire subroutine or an indirect fire subroutine is called once for each weapon.

4.3 Description of Direct Fire and Indirect Fire Calculations

4.3.1 Direct Fire

Subroutine PH2DF is used to perform calculations for direct fire weapons in the original CEM VII code. Subroutine ATCALC calls PH2DF once for each direct fire weapon. Inside PH2DF, individual weapons fire at vehicles on the opposing side. The equations used for direct fire calculations have been translated from a CAA technical description of CEM [Johnson, R.E., 1985]. The purpose of the direct fire subroutines is to calculate the kills of each target type. This is done by analyzing the importance of each target in conjunction with the availability of each target, and the amount of ammunition that each

weapon is able to fire. To study the equations involved, A_{ik} is defined as the target availability matrix, represented as the fraction of time a particular vehicle type is available as a target for a particular weapon type. The subscript "i" represents the ith weapon as the shooter, while the subscript "k" represents the kth vehicle as the target. $1-A_{ik}$ is then the target non-availability. If there are N_k targets of type "k," the fraction of time at least one target of type "k" is available to a type "i" shooter is:

$$1-(1-A_{ik})^{N_k} \tag{4.1}$$

For all N_i shooters, each with the ability to fire $(RATE)_i$ rounds per engagement, the amount of ammunition fired at type k targets in one time step will be:

$$F_{ik} = N_i \times (Rate)_i \times \left[1-(1-A_{ik})^{N_k}\right] \tag{4.2}$$

In CEM VII, the average number of weapons and vehicles are used. In addition, something must be done about competing target types. These two changes result in the following equation:

$$F_{ik} = \overline{N}_i \times (Rate)_i \times \left[1-(1-A_{ik})^{\overline{N_k}}\right] \times (1-A_{ik'})^{\overline{N_{k'}}} \tag{4.3}$$

This represents the fire from all type i shooters at type k targets where the k' target type has a higher priority than type k. In general there is a string of non-availability factors multiplied by this expression, one for each target type that has a priority greater than that of type k's. To calculate attrition on the kth vehicle type, F_{ik} need only be multiplied by the

kills-per-round figure for this combination of shooter and target producing this final equation:

$$(KL_k)_i = \overline{N}_i \times (Rate)_i \times P_{ik} \times \left[1 - (1 - A_{ik})^{\overline{N}_k}\right] \times \prod_{k'} \left((1 - A_{ik'})^{\overline{N}_{k'}} \right) \quad (4.4)$$

which represents the kills of type k vehicles by type i weapons. The symbol $\prod$ represents the product of non-availability factors for all target types (k') with priority greater than target k. The quantities A_{ik} and P_{ik} are values generated through off-line simulation and are provided as input to the main program.

4.3.2 Indirect Fire Calculations

Like subroutine PH2DF, subroutine PH2IF is called by subroutine ATCALC to perform indirect fire calculations. If the first weapon on a vehicle is an indirect fire weapon then the second weapon will also be an indirect fire weapon, and PH2IF will perform calculations for both weapons on this vehicle. If the first weapon on a vehicle is a direct fire weapon and the second is an indirect fire weapon, PH2DF is called for the first weapon, then PH2IF is called for the second weapon only. However it is possible for both weapons on a vehicle to be direct fire weapons. The following equations are also taken from the technical description of CEM [Johnson, R.E., 1985].

The process starts by computing the target priorities and their sum (for normalization in future calculations) for each weapon type (j) on each vehicle type (i) shooting at each target type (k) as follows:

$$(TGTPR)_{ijk} = P_{ijk} \times (IM)_k \tag{4.5}$$

$$(TGTPRS)_{ij} = \sum_k (TGTPR)_{ijk} \tag{4.6}$$

Where P_{ijk} is defined as the Kills-per-round figure for the jth weapon type on the ith shooter type firing at the kth target type, and $(IM)_k$ is the importance of the kth target type.

The next step is to calculate munition priorities and the demand for fire. Once the demand for fire has been computed, the munition priorities are replaced with biased priorities.

$$(MUNPR)_{ij} = \sum_k (TGTPR)_{ijk} \times \overline{N_k} \tag{4.7}$$

$$(DEMAND)_i = \sum_j (MUNPR)_{ij} \tag{4.8}$$

$$(MUNPR)_{ij} \rightarrow (BIAS)_{ij} \times (MUNPR)_{ij} \tag{4.9}$$

Next, the biased munition priorities are summed and the number of rounds fired by all weapons of type i are calculated:

$$(MUNPRS)_i = \sum_j (MUNPR)_{ij} \tag{4.10}$$

$$(ROUNDS)_i = (RSPNS)_i \times (DEMAND)_i \tag{4.11}$$

The values for P, IM, BIAS, and RSPNS are either computed previously, or are included in the K/V scoreboards. $(RSPNS)_i$ is defined as the response factor for the ith weapon type. The value for $(ROUNDS)_i$ is then checked against a stored maximum value, which

ROUNDS is not allowed to exceed. The next step is to calculate the expenditure of rounds from type j tubes on type i vehicles. This value is designated by E_{ij}.

$$E_{ij} = (ROUNDS)_i \times (MUNPR)_{ij} / (MUNPRS)_i \tag{4.12}$$

The actual computer code is more complicated than this, as it makes provisions to apply a stockpile constraint, if desired. Next, the kills-per-round figure is calculated for each munition/target combination as follows:

$$P_{ijk} = \left(\overline{N_k}/N_k\right) \times L_{ijk} \tag{4.13}$$

Where L represents the lethality parameter which is also included in the K/V scoreboards. The average number of targets $\overline{N}_k$ is the quantity computed in the previous iteration of ATCALC, while N_k is the actual number of targets. (For the first iteration, $\overline{N}_k = N_k$, and P = L.) Next, the fraction of type ij rounds associated with target type k is computed:

$$(FRAC)_{ijk} = \frac{(TGPRS)_{ij}}{(TGPRS)_{ij} + \dfrac{(MUNPR)_{ij}}{\overline{N}_k \times (BIAS)_{ij}} - (P_{ijk} \times (IM)_k)} \tag{4.14}$$

Finally, the kills of type k targets by type ij rounds is computed:

$$(KL_k)_{ij} = E_{ij} \times P_{ijk} \times (FRAC)_{ijk} \tag{4.15}$$

4.3.3 Sorting Subroutine (QKSRTI)

The direct fire subroutine, PH2DF, calls the subroutine QKSRTI. QKSRTI is a sorting algorithm, which sorts a one-dimensional array in descending order. QKSRTI is used to sort vehicle importances. The sorted vehicle importances are used to determine which vehicles are important targets for a shooter. Additionally, in the event that ammunition consumption is constrained, this information is used to reduce the amount of ammunition used. Due to the recursion inherent in sorting routines, QKSRTI will not vectorize fully.

For future discussions, it is noteworthy to point out that this is a non-deterministic sort algorithm. To illustrate determinism in a sorting routine, consider the following list of numbers to be sorted.

1) 5.0
2) 4.3
3) 3.7
4) 5.0
5) 7.3
6) 2.9
7) 5.0
8) 3.7
9) 4.5

Current "smart" sorting routines actually return an ordered list of the indices. In a deterministic sort, the order of multiple occurrences of the same number are preserved in the original order. That is, a deterministic sort routine and non-deterministic sort routine may return the following lists.

Table 4.1 Sorted Indices from a Deterministic/Non-Deterministic Sort

	Indices of Sorted Items	
Ordered List Number	**Deterministic Sort**	**Non-deterministic Sort**
1)	6	6
2)	3	8
3)	8	3
4)	2	2
5)	9	9
6)	1	4
7)	4	7
8)	7	1
9)	5	5

In the list to be sorted, note that the number five is found in the first, forth, and seventh places. In the deterministic sort, the sorted list contains the indices 1, 4, and 7 in original order. A different order (*i.e.* 4, 7, 1), may be returned by the non-deterministic sort.

4.4 Approach

In order to achieve complete vectorization over shooter target pairs, it was necessary to invoke a concept called "Gather-Scatter." The concept of "Gather-Scatter" is implemented in a computer's hardware, so "Gather-Scatter" is significant only for machines with such vector hardware. As stated before, vector hardware requires that data be contiguous in memory. Since some weapons are direct fire and some are indirect fire, and all vehicle types may not be used in an engagement, the information required to perform all direct fire, or indirect fire calculations is not contiguous in memory. Thus, the concept of "Gather-Scatter" comes into play. By knowing beforehand which weapons belong to which group, the "Gather" portion accesses discontinuous information in memory and loads it into a contiguous vector register. Then the vector hardware can be used. When the

calculations are finished, the "Scatter" portion stores the results into the correct discontiguous place in memory. To illustrate this process consider the following Fortran code:

```
Do I= 1,100,2
    C(I) = A(I) + B(I)
End Do
```

This loop is used to add all odd numbered elements in the A array to all odd numbered elements in the B array, and to store the values in all odd numbered elements of the C array. Because calculations are being performed on every other element, the data are not contiguous in memory. The "Gather" portion of the "Gather-Scatter" loads every other element of the A and B arrays into contiguous storage in the vector registers. Now that the values are contiguous in the vector registers, the vector hardware can be used. As the resultant numbers emerge from the vector hardware, the "Scatter" unit stores the results in every other element of the C array. Figure 4.3 depicts this sequence. Note that the gray boxes in Figure 4.3 represent data used in the computation, while the other boxes represent data that are not used in the calculations.

In order to implement "gather-scatter," it is necessary to pre-determine the indices of all direct fire weapons and all indirect fire weapons. This allows the direct fire and indirect fire subroutines to be called once for each engagement, and all weapons of each category can be "ganged" together. This provides an opportunity for vectorization inside the direct fire and indirect fire subroutines. Figure 4.4 shows the new firing structure that was created in order to attempt vectorization in the direct fire and indirect fire subroutines.

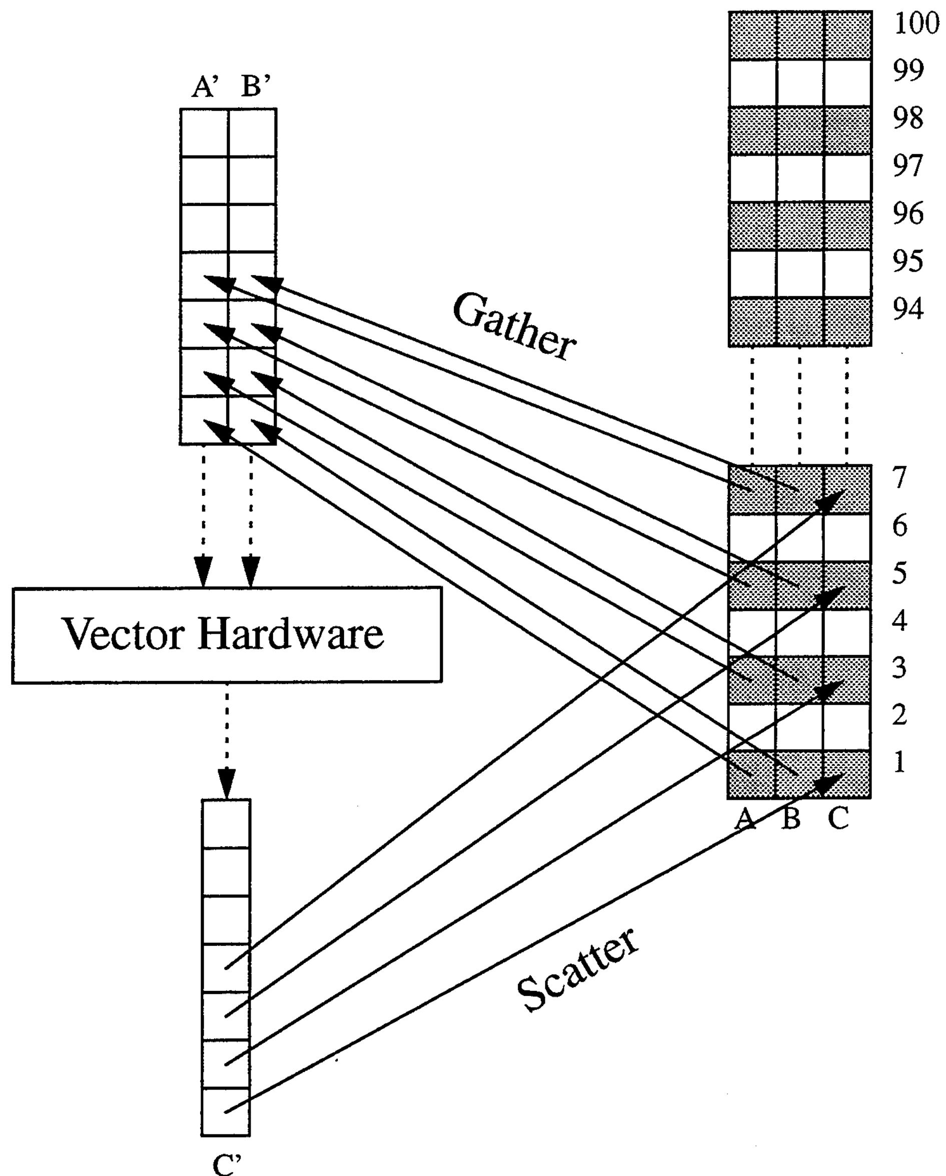


Figure 4.3 Depiction of "Gather-Scatter" in Vector Hardware

This is computationally attractive as it allows an increase in vector (array) size, since all weapons on a side fire together, and allows for enhanced performance of the CRAY vector hardware. This new structure effectively eliminates the "Blue Bias," since the Red side is able to fire before sustaining casualties. Initial results indicated that the battle progressed in a different manner, due to the elimination of the "Blue Bias," and the overall outcome

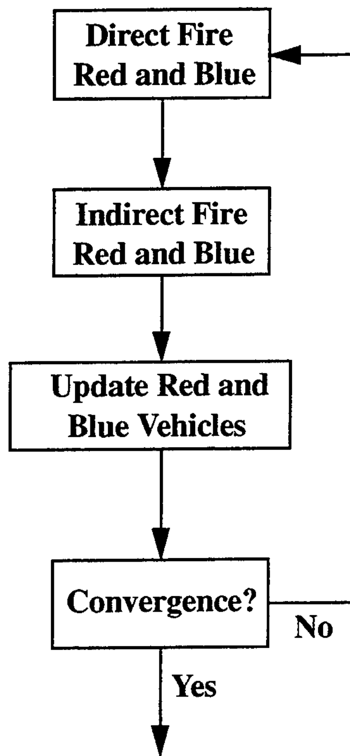


Figure 4.4 Flow Chart of Modified Firing Order in ATCALC

of the battle differed slightly from the original code. This lead to continued work with an attempt to re-introduce the "Blue Bias," to preserve consistency.

By sorting the direct fire and indirect fire vehicles into the Blue and Red subdivisions, re-introducing the "Blue Bias" was accomplished. This was implemented through the creation of four arrays: 1) information regarding all Blue direct fire weapons, 2) information regarding all Blue indirect fire weapons, 3) information regarding all Red direct fire weapons, and 4) information regarding all Red indirect fire weapons. Four subroutines are used to perform the firing calculations, one for each of the above arrays. A call to each of these performs all calculations for that weapon category. The new ATCALC subroutine re-introduces the "Blue Bias" by processing the firing order as shown in

Figure 4.5. Here all Blue direct fire calculations, and all Blue indirect fire calculations are

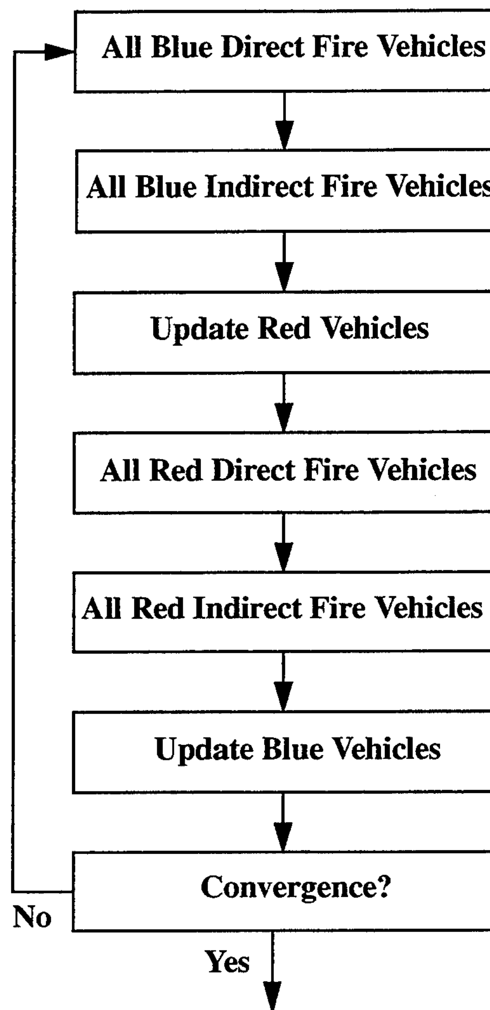


Figure 4.5 Flow Chart of Final Firing Order in ATCALC

performed, followed by attrition calculations on the Red side. Then all Red direct fire calculations are performed, followed by all Red indirect fire calculations, and finally Blue attrition calculations are done. This results in answers identical to those produced by the original code, with increased performance.

4.5 Vectorization of Direct Fire Subroutines

Since information regarding Blue and Red direct fire weapons has been pre-determined, the gather-scatter hardware can be used to vectorize the direct fire subroutines. The Blue and Red direct fire subroutines are almost identical, so changes to one subroutine can be implemented in the other. The difference stems mostly in the initialization of variables describing which vehicles are the shooters and which vehicles are the targets.

The direct fire subroutine performs many tasks before performing the actual attrition calculations. The most relevant of these is the calculation of the target priorities for each shooter, and a subsequent sort, in order to determine the most important targets. Due to the complex logic involved in the assigning of target importances, this portion of the code will not vectorize, so these calculations are isolated in a loop of their own. This loop preforms the calculations and sortings of all targets importances for all Blue/Red direct fire weapons that are shooting.

Following the calculation and sorting of importances, attrition calculations are performed. One way to perform calculations over all weapons firing at all targets is to create a double loop. The outer loop indexes over shooters, and the inner loop indexes over targets. Many of the indices that are used in attrition calculations are in the form of two-dimensional arrays. Using these two-dimensional arrays in conjunction with the gather-scatter hardware creates a pattern for which the compiler does not have a vector "template." This problem was overcome by storing all the required information in one-dimensional arrays, and creating single loops in which calculations for all shooter-target pairs are performed.

Information regarding Blue direct fire weapons is stored in array dfbvn(N,M), where N is the number of Blue direct fire weapons. The M subscript can be either 1 or 2. The value in dfbvn(N,1) is the index of the vehicle type for the Nth Blue direct fire weapon. The value in dfbvn(N,2) relays the index of which weapon type is being used. It is this array that is passed to the Blue direct fire subroutine.

When the target importances are sorted, the information is placed in array iqksrt(K,N), where the N subscript again denotes the number of Blue direct fire weapons. The K subscript refers to the target. For instance, iqksrt(4,5) is the index of the fourth most important vehicle for the fifth Blue direct fire weapon to shoot at. Suppose the Nth Blue direct fire weapon is shooting at its Kth most important target. The target availability would be accessed follows:

availability = a(dfbvn(N,2),iqksrt(K,N))

To simplify this and permit vectorization, the information in dfbvn and iqksrt is stored in one-dimensional arrays. Suppose the number of Blue direct fire weapons is denoted by the variable nbdf, and there are k targets for each weapon. One-dimensional arrays" l" and "sq" are created to hold the information in the two-dimensional arrays dfbvn and iqksrti. The code to transfer the information between the arrays could look as follows:

```
nk = 0
Do n = 1,nbdf
    Do ik = 1,k
        nk = nk + 1
        l(nk) = dfbvn(n,2)
        sq(nk) = iqksrt(ik,n)
    End Do ;On ik loop
End Do ;On n loop
```

The variable nk now holds the total number of shooter-target pairs. Only one loop is required to loop over all nk shooter-target pairs. Accessing the target availability would now appear as follows:

```
Do nik = 1,nk
    availability = a(l(nik),sq(nik))
        .
        .
End Do ;On nik loop
```

It is, however, necessary to create 3 of these single loops. This is necessary due to a data dependency in the calculations. Referring to Section 4.3.1, note that the kills of vehicles are dependent upon the product of non-availability factors for all targets with priority greater than the present target. For this reason it is necessary to compute the product of non-availability factors for the present vehicle before calculations on the next vehicle can be started. If all calculations were done in one loop, this data dependency would prohibit the entire loop from vectorizing. This was overcome by creating three loops that perform the calculations. The first loop performs all calculations up to the product of non-availability factors. The second loop is responsible for the calculation of product non-availability factors for all shooter target pairs. The third loop performs all remaining calculations. The first and third loops successfully vectorized. However, due to the data dependency, vectorization of the middle loop is not possible.

4.6 Vectorization of Indirect Fire Subroutines

Due to the complex logic in the indirect fire subroutines, full vectorization of these subroutines was not successful. This was mostly due to the fact that calculations are

performed on a vehicle basis, rather than a weapon basis, as in the direct fire subroutines. As stated before, if the first weapon on a vehicle is an indirect fire weapon, then the second weapon will also be an indirect fire weapon. But if the first weapon is a direct fire weapon, then the second can be either a direct fire or an indirect fire weapon. This creates complex logic in the indirect fire subroutines that is not amenable to complete vectorization. However, it was possible to vectorize two of the minor loops in the subroutines. For instance, the calculations involved with Equations 4.5 through 4.7 were isolated in a loop of their own, which was vectorized. Additionally, the calculations involved with Equations 4.13 through 4.15 are in their own loop which also vectorizes.

4.7 New Sort Routine

Subroutine QKSRTI was replaced by a modified Batchers Sort [Knuth, 1973], written by a previous CSU Masters candidate, Michael Brewer. While this new sort program does not vectorize completely, Mr. Brewer found it achieves the highest degree of vectorization of all sort routines. It was noted earlier that the original sort routine found in CEM VII is not "deterministic." In the same sense, the new sort created by Mr. Brewer is also non-deterministic. However, the new sort is non-deterministic in a different manner. As a result, the outcome of a battle can be slightly different. When there are vehicles of the same importance, the two sort algorithms will return the indices of the sorted array in different order. This does yield slightly different results, as will be illustrated in the next section.

4.8 Results

4.8.1 Total Kills of Vehicles

The "old" and "new" CEM versions were run, and the total kills were computed by vehicle type. Figure 4.6 shows the percent difference in total kills of vehicles between the "old" code and the "new" code. The differences in kills was isolated to the difference

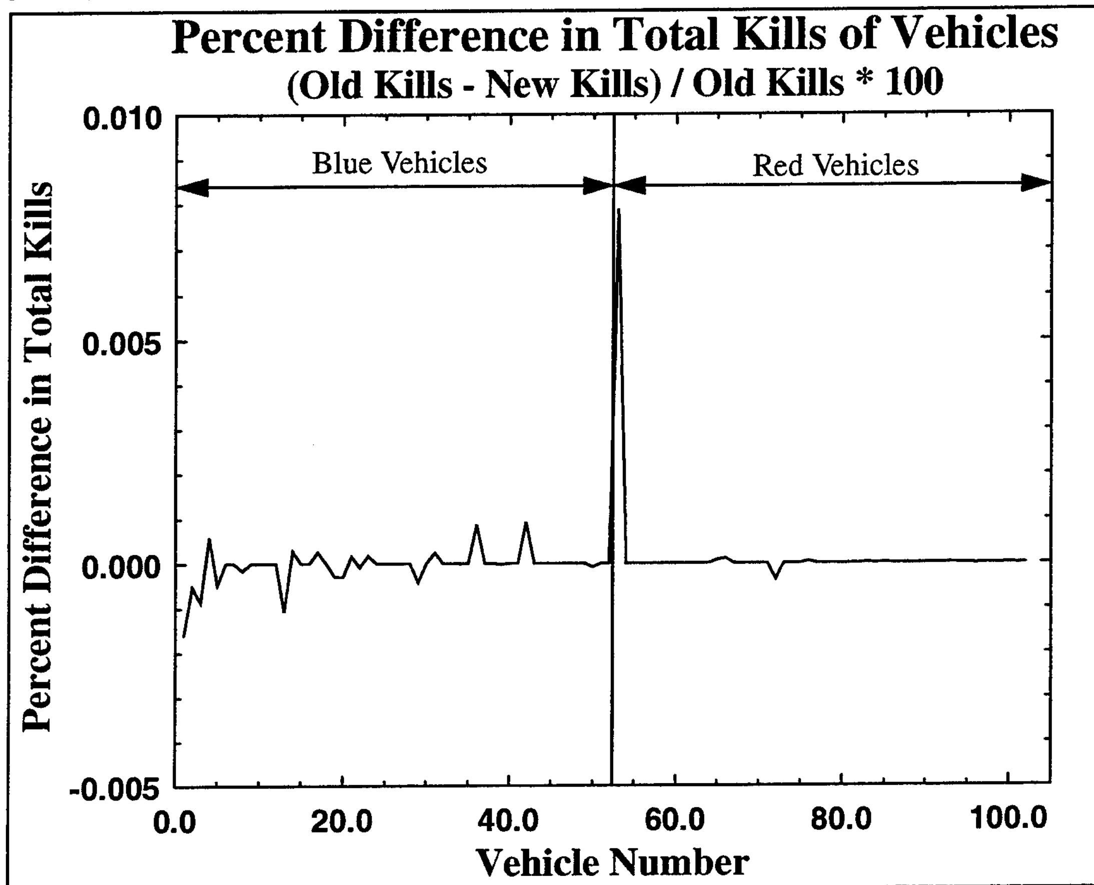
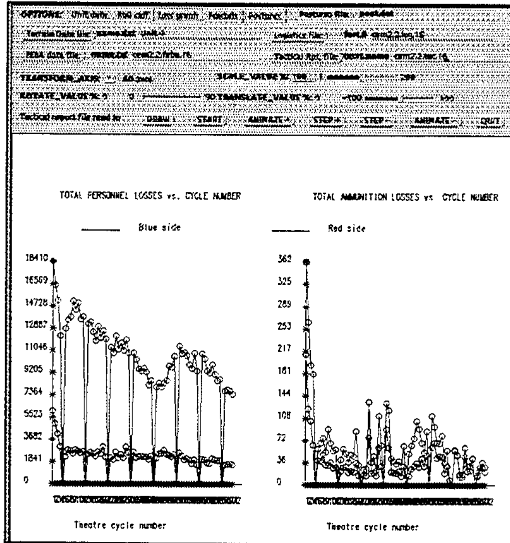


Figure 4.6 Percent Difference in Total Kills of Vehicles Between Original CEM VII and Vectorized Version

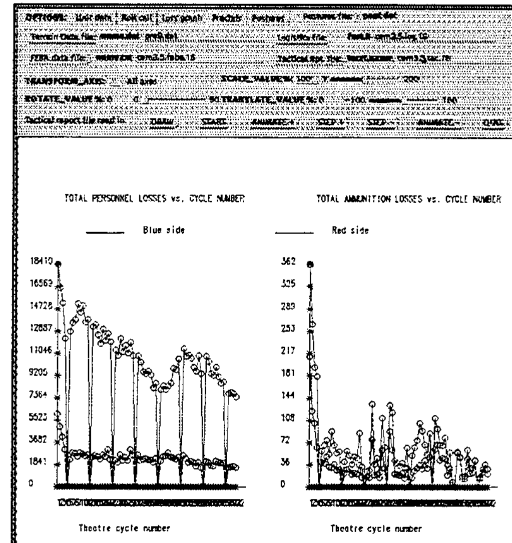
between the non-deterministic sorts of the two codes. That is, when the "new" code is run using the "old" sort routine, identical results are obtained. Note that, with the "new" sort, the maximum difference in kills is very small - only about 0.008 percent.

3.8.2 CEMPlot

Because the two codes produce almost identical results, the FEBA movement and loss graphs are also almost identical. Figure 4.7 shows the loss graphs for the two simulations, while Figure 4.8 shows the positions of the FEBA at the end of 16 theater cycles.

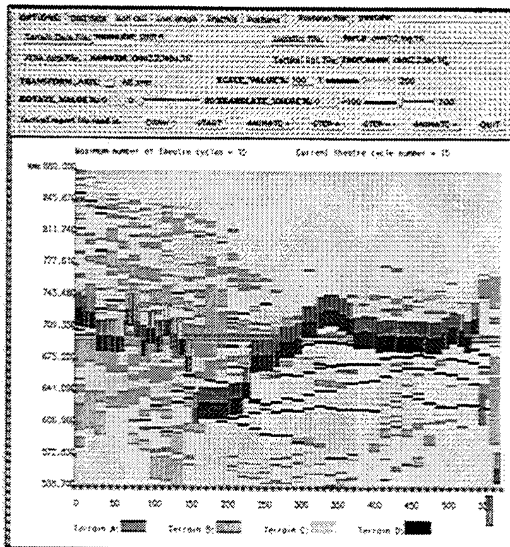


(a) Original Code

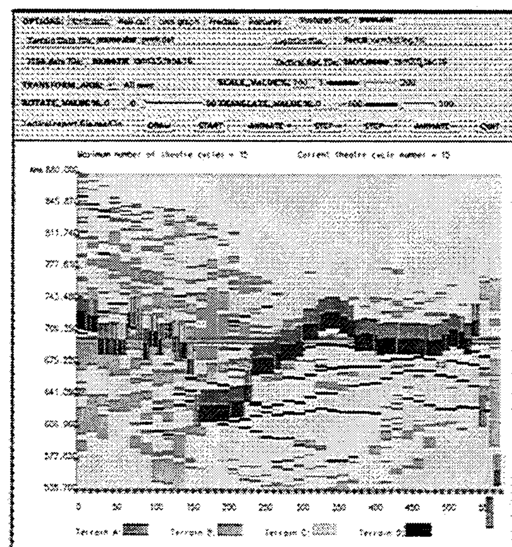


(b) Vectorized Code

Figure 4.7 Loss Graphs of Original and Vectorized Codes After 16 Theater Cycles



(a) Original Code



(b) Vectorized Code

Figure 4.8 FEBA Location of Original and Vectorized Codes After 16 Theater Cycles

4.9 Performance

4.9.1 hpm

Figure 4.9 shows hpm statistics for the final vectorized version of CEM. The run time of

```
clark 100 % hpm -g0 cem3.5.3 <UNIT9> cem3.5.3.out
WAR      HALT     ED A     FTER      THE      ATER      CYC      LE          16
CEM      EXIT     ING      NORM      ALLY
 STOP executed at line 12764 in Fortran routine 'EOW'
 CP: 499.623s,  Wallclock: 693.308s,  36.0% of 2-CPU Machine
 HWM mem: 1607209, HWM stack: 2048, Stack overflows: 0
Group 0:  CPU seconds    : 499.62      CP executing       :    83270601958

Million inst/sec (MIPS) : 44.49      Instructions       :    22225829519
Avg. clock periods/inst : 3.75
% CP holding issue      : 56.71      CP holding issue :      47223674058
Inst.buffer fetches/sec : 0.34M      Inst.buf. fetches:        168674060
Floating adds/sec       : 4.65M      F.P. adds          :     2323053594
Floating multiplies/sec : 2.37M      F.P. multiplies    :     1183545969
Floating reciprocal/sec : 0.35M      F.P. reciprocals :        174932134
I/O mem. references/sec : 0.14M      I/O references     :       69011803
CPU mem. references/sec : 19.96M     CPU references     :     9974898360

Floating ops/CPU second :       7.37M
```

Figure 4.9 hpm Statistics for Vectorized Version of CEM VII

499.623 seconds represents a 7.73% speedup from the original run time of 541.495. This disappointingly small increase in performance is attributed to short vector lengths, as will be discussed in Section 4.10.

4.9.2 Flowview

Table 4.2 shows the CPU usage for the top 20 subroutines in the final vectorized version of CEM.

Table 4.2 CPU Usage in Vectorized CEM Code

Flowtrace Statistics Report Showing Routines Sorted by CPU Time (Descending) (CPU Times are Shown in Seconds)					
Routine Name	**Total Time**	**Number of Calls**	**Average Time / Call**	**Individual Percentage**	**Cumulative Percentage**
ATCALC	2.30E+02	17281	1.33E-02	41.78	41.78
SORT	3.75E+01	417656	8.97E-05	6.81	48.59
PIK	3.53E+01	7412552	4.76E-06	6.41	55.00
TNKAPC	3.48E+01	68905	5.05E-04	6.32	61.32
ARTCAS	3.09E+01	17281	1.79E-03	5.61	66.93
RNDFRD	2.05E+01	17281	1.19E-03	3.73	70.67
PH2IFR	1.32E+01	69226	1.91E-04	2.40	73.07
PH2DFB	9.43E+00	69226	1.36E-04	1.71	74.78
PH2DFR	8.57E+00	69226	1.24E-04	1.56	76.34
DIVRPT	8.48E+00	128	6.63E-02	1.54	77.88
PH2IFB	8.02E+00	69226	1.16E-04	1.46	79.34
ASSESS	7.42E+00	128	5.80E-02	1.35	80.69
CASL	7.16E+00	17281	4.14E-04	1.30	81.99
ESTOUT	6.42E+00	85573	7.51E-05	1.17	83.16
PQMOD	6.25E+00	95730	6.53E-05	1.14	84.30
UNITS	6.20E+00	37359	1.66E-04	1.13	85.42
PAK	5.74E+00	1056240	5.43E-06	1.04	86.47
CINDEX	4.83E+00	1658274	2.91E-06	0.88	87.34
ITER	4.82E+00	102756	4.69E-05	0.88	88.22
STAMAT	4.78E+00	95730	5.00E-05	0.87	89.09

Comparing the total time spent in ATCALC, QKSRTI, PH2DF, PH2IF between Table 3.1 for the “old” code, and Table 4.2 for the “new” code, indicates a speed up of 15.43% in these subroutines. This yields only a 7.73% increase in the run time of whole program, since these subroutines account only for a portion of the CPU time.

4.10 Vector Lengths in CEM VII

In order to determine why the increase in performance was smaller than anticipated, a study was done on how the gather-scatter and vector hardware perform. Specifically, two issues were addressed. First, since the new code spends significant time in determining the indices of the weapon types, and creating arrays that contain only the indices of vehicles that will actually be firing, data on the rate at which the hardware sorts the information is required. Secondly, information is required on how fast the hardware can perform indexed vector computations. This was done with two parameters in mind. Namely the input vector length and the "density" of the operation. Density refers to the fraction of operands which yield a resultant, *i.e.* the fraction of operands for which the logical test is "true." Sometimes, density is referred to alternately as "truth ratio." For example, after the vehicles are filtered into the correct firing type, they are again filtered to check that vehicles of each type still exist. If, on average, 8 of 10 vehicles still remain, the density would be 0.8. Viewing Figure 4.10 will help to understand the study. On the x-axis is the input vector length. The y-axis is the number of indices that are generated per second. The individual curves represent different densities. For example, suppose there are 50 Blue direct fire weapon types that need to be processed to determine which remain to engage in battle. Further, suppose that 6 out of every 10 have vehicles left (*i.e.* a density of 0.6.) Figure 4.10 indicates that the Cray YMP should be able to generate roughly 19 million indices per second (Minds). This may seem like good performance unless one observes that an input vector length of 200 and a density of 1.0 results in about 43 million indices generated per second. No matter what the density is, there is diminished performance at smaller input vector lengths.

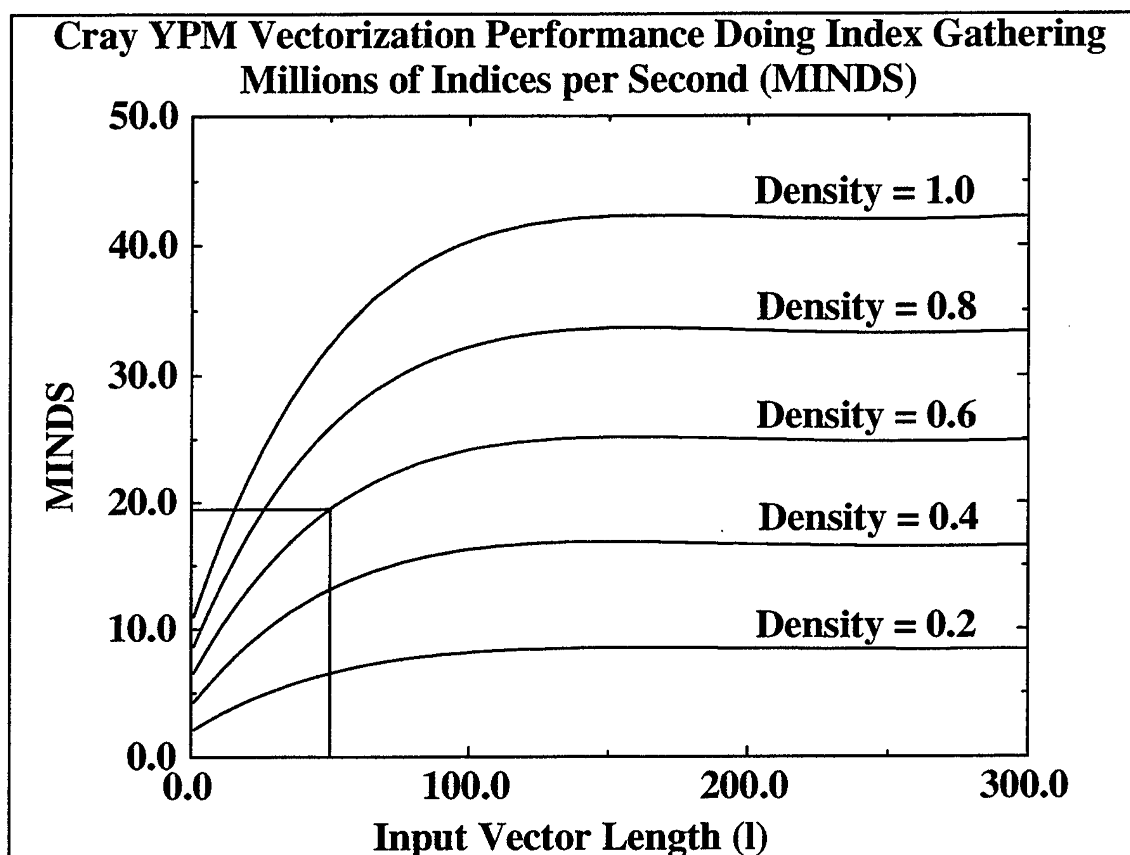


Figure 4.10 Performance of Indexed Gathers on Cray YMP

The same sort of trend exists when performing floating point operations. Figure 4.11 shows information similar to Figure 4.10, only rather than gathering indices, the computer is performing indexed floating point multiplies. Again, no matter what the density, the performance is reduced at smaller input vector lengths. It was found, for the typical input case that was being run, that the input vector lengths averaged 18 with densities that were relatively small, around 0.17. Using Figure 4.11, an estimated Mflops rating would be around 5 for this combination of vector length and density. It is for this reason that the increase in performance was modest. It is noteworthy to state that non-indexed floating point operations can result in performance ratings greater that 200 Mflops.

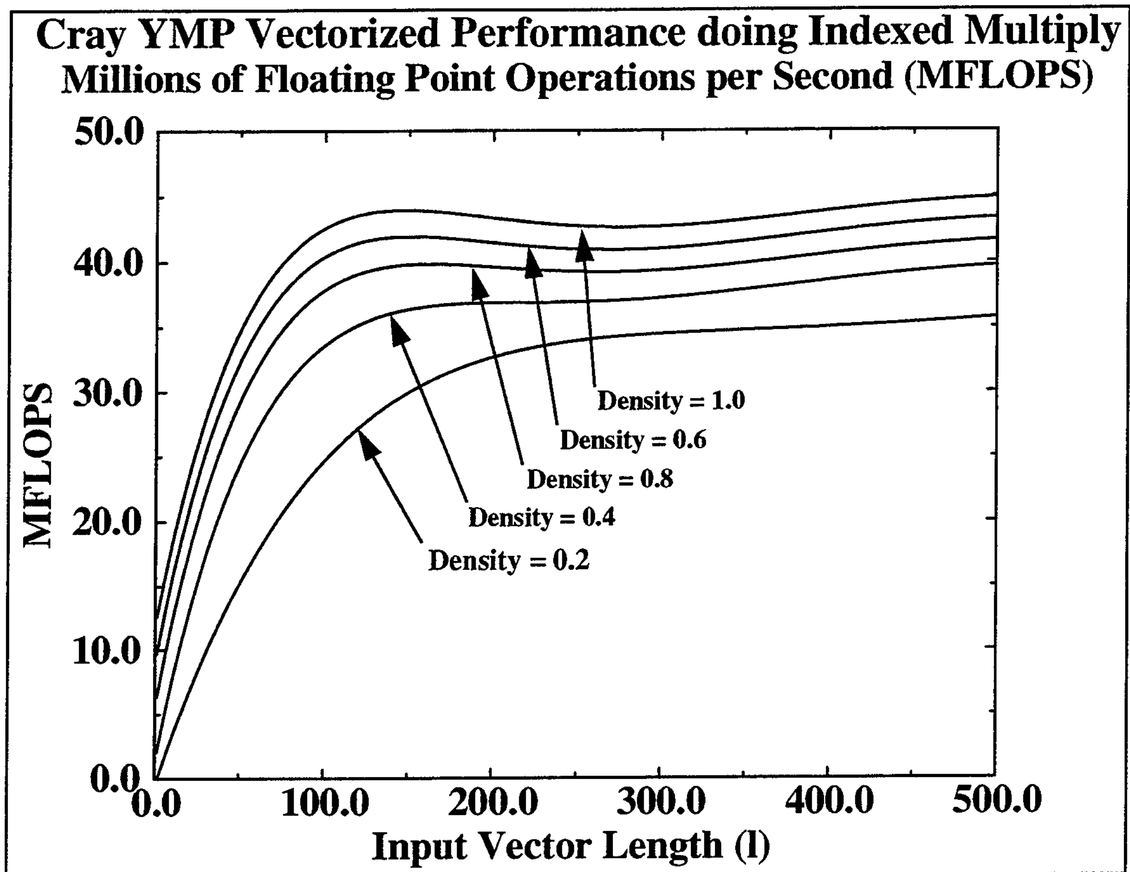


Figure 4.11 Floating Point Performance on Cray YMP

5. Elimination of Packing / Unpacking

5.1 Background

When CEM VII was first developed computers had little internal memory. As a result, the CEM code was implemented using a concept called Data Packing, where multiple values are "packed" or stored in a single word. Typically, when a number is stored in memory, a whole memory word is devoted to that piece of information. For example the number 1,234 requires only 11 bits to be stored in binary form. In Cray supercomputers, a memory word comprises 64 bits. This means that if stored in typical fashion, the other 53 bits in the word are unused if 1,234 is stored. When Data Packing is used, 11 bits are devoted to storing 1,234, and other information can then be stored in the remaining 53 bits. The following figures depict the process. Figure 5.1 represents how the numbers 750 and 1,234 would normally be stored in contiguous memory.

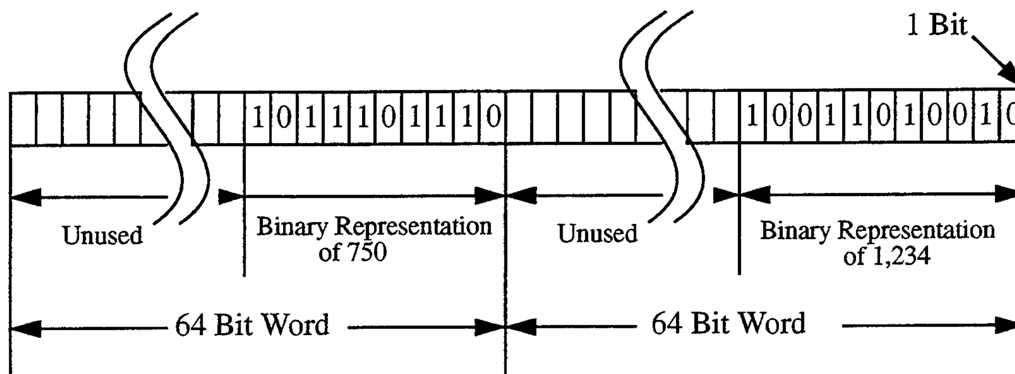


Figure 5.1 Typical Binary Storage of 750 and 1,234

Figure 5.2 shows how the two numbers would be stored using Data Packing. Note that in both Figures 64 bits are specified in each word.

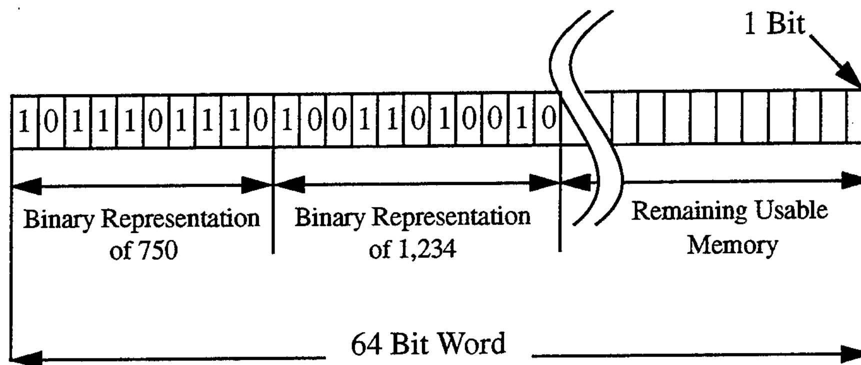


Figure 5.2 Data Packing Storage of 750 and 1,234

While the reduction of required internal memory is a benefit, there are drawbacks. First, in order to retrieve (unpack) the information, additional data on how the information is stored must be kept. Second, and the more significant of the drawbacks, CPU time is required to "pack" and "unpack" the information, rather than simply addressing a word in memory. Additionally, due to the different word size in different computer architectures, "Data Packing" makes it difficult to port the code to other computers.

5.2 Approach

Another approach to increase performance and portability of CEM VII was to eliminate the "Data Packing and Unpacking." This is an entirely feasible concept, as current computers have large internal memories. In the original CEM VII code there are eighteen "packed" arrays. Some of them are created in the pre-processor and read in when the main program is initiated, while others are created and used in the main program. The main

program retrieves, from the "packed" arrays, the information as needed, and stores information, in "packed" form, as it is produced.

Data that is stored in the "packed" array requires two pieces of information for retrieval; (1) the number of bits into the array at which the information starts, and (2) the number of bits required to hold the information. At first it would seem that storing all of this additional information would make data packing redundant, since two pieces of information are required for each piece of packed information. This is true unless there are many items of information with an identical data structure. This is so for many arrays in the CEM program. An example would be the ARMY array. This array keeps track of all information pertaining to all armies present in the theater. The ARMY array contains 15 pieces of information, and there can be up to 24 armies in the theater- 12 Blue and 12 Red. Some of the information stored in this array follows:

1. Index of current army mission -- 0=delay, 1=defend, 2=attack.
2. Lowest minisector controlled by army ("low minisector bound").
3. Highest minisector controlled by army ("high minisector bound").
4. Number of non-divisional general support artillery battalion equivalents.
5. Number of close air support squadrons assigned to army.
6. Number of corps cycles remaining until army's reserve subordinate corps will be committed.
7. Number of subordinate corps attached to army.

Since the data storage for one army is the same for all armies, the structure of the information has to be kept for only one army, but can be used for all armies. For this to work, only one additional piece of information is needed; the number of bits required to store all of the information for a complete army. With all this known, it is now possible to move into the packed array the correct distance to locate the beginning of the army of interest.

In the CEM program all bit manipulation for the data packing / unpacking is performed in three subroutines: CINDEX, PIK, and PAK. CINDEX is used to determine the word in which the data begins, and the number of bits into that word where the information starts. For example, one of the pieces of information stored for an army is the highest minisector controlled by the army. The information is the third item of data, starts twelve bits into the army's data, and is ten bits long. In order to perform the computations it is necessary to know that the data for a complete army requires eighty-six bits. To retrieve the highest minisector for the third army, the subroutine CINDEX is called as follows:

```
CALL CINDEX (3 , 86 , INDEX , LOVER)
```

This returns INDEX and LOVER. INDEX is the word index, of the "packed" army array, where the third army data starts. LOVER holds how many bits into that word the information begins. Knowing INDEX and LOVER, subroutine PIK is called. PIK is used to access the information in the array. This involves bit manipulation that is beyond the scope of this paper. The call to PIK would look as follows:

```
CALL PIK (ARMY(INDEX) , LOVER+BSARHM , BLARHM , MINIH)
```

BSARHM represents the number of bits into the army's information that the data starts. BLARHM holds how many bits long the data is. MINIH is the variable passed back which

holds the information that was requested, in this case the highest minisector controlled by the third army.

To pack information the same process is followed, with the exception that subroutine PAK, rather than PIK, is used.

For obvious reasons it is more efficient simply to have a two-dimensional array, ARMY(army# , info), and retrieve the information as follows:

MINIH = ARMY(3 , 3)

There were eighteen packed arrays in CEM VII that were eliminated. For each packed array that was eliminated, an unpacked array was created to replace it. To begin with, the work focused solely on the main program. For those packed arrays brought in from the pre-processor, the array was first unpacked and the information stored in the unpacked array. From there it was just a matter of replacing the lines that involved packing/ unpacking with lines that simply stored or retrieved the information from the normal arrays. The following table shows the packed arrays that were replaced, the information in the array, and the name of the replacement array.

Table 5.1 Packed Arrays and Their Replacements

Packed Arrays	Information Held in Array	Unpacked Arrays
ACHIST	History data for all corps on line for the present and previous two army cycles for both sides.	onlncorp
BARMY	Current status information for blue armies.	armyb
BCORPS	Current status information for blue corps.	corpb
BDIV	Current status information for blue divisions, and their subordinate brigades.	divb
RARMY	Current status information for red armies.	armyr
CDHIST	History data for all divisions on line for the present and previous two corps cycles for both sides.	onlndiv

Table 5.1 Packed Arrays and Their Replacements

Packed Arrays	Information Held in Array	Unpacked Arrays
RCORPS	Current status information for red corps.	corpr
RDIV	Current status information for red divisions.	divr
RFDVB & RFDVR	Contains information on when division will be entering the battle.	nrnfdv
DBHIST	History data for all blue brigades/red divisions on line for the present and previous two division cycles.	onlnbrig
FEBA	Feba location by minisector.	febaloc
FEBM	Feba movement coefficients for all minisectors.	febacoef
FLEXIN	Denotes which red divisions have been withdrawn for rebuild.	flexin2
IWPOOL	Logistics arrival data.	iwpool2
MINIA	History data concerning corps locations by minisector for the present and previous two army cycles for both sides.	corpmini
MINIC	History data concerning division locations by minisector for the present and previous two corps cycles for both sides.	divmini
MINID	History data concerning blue brigade/red division locations by minisector for the present and previous two division cycles.	brigmini
TERN	Description of CEM terrain.	tern2

Once this was working correctly, the data packing and unpacking were eliminated from the pre-processor and post-processor. This involved altering the read and write statements in all three parts of CEM in order to read and write the "unpacked" arrays. Additionally, the lines used to perform "Data Packing / Unpacking" were replaced with code to use standard arrays. This eliminated the need to unpack the arrays that were brought in from the pre-processor at the beginning of the main program.

Elimination of the packed arrays will facilitate future debugging attempts. It is now possible to simply print the arrays, rather than having to unpack them prior to printing.

Additionally, the removal of "Data Packing" will assist in future vectorization attempts in subroutines which use "Data Packing." This is due to the fact that subroutines containing "Data Packing" are unvectorizable. Finally, the elimination of "Data Packing" makes it easier to port the code to other computer architectures. Different computer architectures have different word sizes. In order to run the original CEM code on other machines, the code requires alteration to allow for the different word size. With "Data Packing eliminated, this is no longer an issue.

5.3 Results

The following sections report the results obtained with the "Data Packing and Unpacking" eliminated. Note that these changes were made to the original CEM VII code, and not to the vectorized version. This was done to determine the increase in performance that was attributed to the elimination of "data packing" alone. Chapter 6 will detail the results obtained by elimination of "data packing" in conjunction with vectorization.

5.3.1 Total Kills of Vehicles

Because the changes were made only to the type of array that was being used, there should be no difference in the results the two codes produce. Both codes produced exactly the same number of kills, for all vehicle types, over the entire simulation.

5.3.2 CEMPlot

As would also be expected, CEMPlot showed the same FEBA movement and loss graphs throughout the entire simulation. Therefore, it is unnecessary to show the figures.

5.4 Performance

5.4.1 hpm

Figure 5.3 shows the hpm statistics generated through a run with the data packing and unpacking eliminated from the original CEM VII code. As was hoped, the run time for the

```
clark 56 % hpm -g0 cem3.4.b1 <UNIT9> out2 &

WAR      HALT     ED A     FTER     THE      ATER      CYC      LE        16
CEM      EXIT     ING      NORM     ALLY
 STOP executed at line 13137 in Fortran routine 'EOW'
 CP: 504.053s,  Wallclock: 7555.184s,  3.3% of 2-CPU Machine
 HWM mem: 1530662, HWM stack: 2048, Stack overflows: 0
Group 0:  CPU seconds    : 504.05    CP executing      :  84008933362

Million inst/sec (MIPS)  : 42.88     Instructions      :   2161229866
Avg. clock periods/inst  : 3.89
% CP holding issue       : 57.00     CP holding issue  :  47886469708
Inst.buffer fetches/sec  : 0.27M     Inst.buf. fetches:     134108424
Floating adds/sec        : 4.21M     F.P. adds         :   2123989533
Floating multiplies/sec  : 1.75M     F.P. multiplies   :    883005368
Floating reciprocal/sec  : 0.19M     F.P. reciprocals  :     96042866
I/O mem. references/sec  : 0.65M     I/O references    :    330014897
CPU mem. references/sec  : 17.84M    CPU references    :   8991010512

Floating ops/CPU second  : 6.16M
```

Figure 5.3 hpm Statistics for "Unpacked" Version of CEM VII

new code was reduced. The old code ran in 541.2 s for a sixteen theater cycle run, while the new code ran in 504.1 s. This is a 6.9% reduction in run time, or a 6.9% increase in performance.

5.4.2 Flowview

Table 5.2 shows the top 20 subroutines used in the new code. Note that the three subroutines that were used for packing and unpacking are absent, as the use of these subroutines has been completely eliminated.

Table 5.2 CPU Usage in "Unpacked" Version of CEM

Flowtrace Statistics Report Showing Routines Sorted by CPU Time (Descending) (CPU Times are Shown in Seconds)					
Routine Name	**Total Time**	**Number of Calls**	**Average Time / Call**	**Individual Percentage**	**Cumulative Percentage**
ATCALC	2.45E+02	17281	1.42E-02	46.57	46.57
QKSRTI	4.67E+01	417656	1.12E-04	8.86	55.43
PH2IF	3.90E+01	645782	6.04E-05	7.40	62.83
TNKAPC	3.44E+01	68905	4.99E-04	6.53	69.36
PH2DF	3.12E+01	2827544	1.10E-05	5.92	75.28
ARTCAS	3.06E+01	17281	1.77E-03	5.80	81.08
RNDFRD	2.04E+01	17281	1.18E-03	3.88	84.95
DIVRPT	8.46E+00	128	6.61E-02	1.60	86.56
ASSESS	7.25E+00	128	5.67E-02	1.38	87.93
CASL	7.13E+00	17281	4.12E-04	1.35	89.29
ESTOUT	6.39E+00	85573	7.46E-05	1.21	90.50
PQMOD	6.24E+00	95730	6.52E-05	1.18	91.68
STAMAT	4.84E+00	95730	5.05E-05	0.92	92.60
DDEND	4.29E+00	128	3.35E-02	0.81	93.41
DBSTAT	3.96E+00	14613	2.71E-04	0.75	94.16
CRQMNT	3.50E+00	53192	6.57E-05	0.66	94.83
CB	2.17E+00	17281	1.25E-04	0.41	95.24
UNITS	1.73E+00	37359	4.63E-05	0.33	95.56
EXCHG	1.55E+00	53192	2.92E-05	0.29	95.86
PRTBDV	1.40E+00	1654	8.45E-04	0.27	96.12

6. Combination of "Vectorization" and Elimination of Data Packing

Having successfully accomplished the "vectorization" and elimination of "Data Packing," the last step was to combine the two approaches into one code. This was achieved by taking the appropriate subroutines out of the "vectorized" version and placing them into the "unpacked" version.

6.1 Results

6.1.1 Total Kills of Vehicles

The resulting kills of each vehicle are, as should be expected, identical to the original vectorized code. Figure 6.1 again shows the percent difference in total kills, of all vehicles, between the original code and the combined "vectorized" and "unpacked" code. Again, this difference is due to the difference in non-deterministic sorts in the two codes.

6.1.2 CEMPlot

Additionally, the results, as viewed with CEMPlot, are identical to the results from "vectorization" alone. For this reason, no figures are shown here (refer to section 4.3.2).

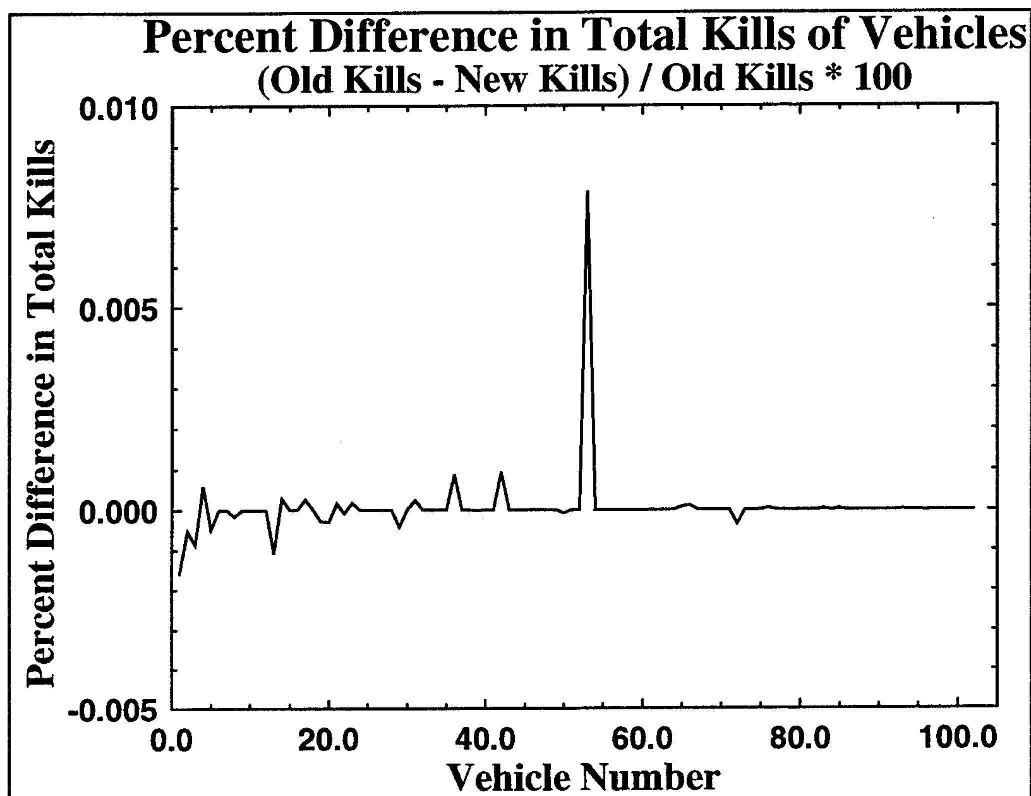


Figure 6.1 Percent Difference in Total Kills of Vehicles Between Original CEM VII and Combined Vectorization and No Data Packing

6.2 Performance

6.2.1 hpm

Figure 6.2 shows the data obtained from hpm on the run of the combination of "vectorization" and elimination of "Data Packing."

The original code ran in 541.5 seconds. The 460.6 second run time for the new code results in a 14.93% decrease in run time.

6.2.2 Flowview

Table 6.1 shows flowview information on the combined version of the CEM VII code

```
clark 43 % hpm -g0 cem3.7.3 <UNIT9> cem3.7.3.out
WAR      HALT     ED A     FTER      THE      ATER      CYC      LE          16
CEM      EXIT     ING      NORM     ALLY
 STOP executed at line 13217 in Fortran routine 'EOW'
 CP: 460.646s,  Wallclock: 944.645s,  24.4% of 2-CPU Machine
 HWM mem: 1709469, HWM stack: 2048, Stack overflows: 0
Group 0:  CPU seconds    : 460.65      CP executing     :    76774455345

Million inst/sec (MIPS) :  43.94      Instructions     :    20241943607
Avg. clock periods/inst :  3.79
% CP holding issue      :  58.01      CP holding issue :    44534594215
Inst.buffer fetches/sec :  0.23M      Inst.buf. fetches:      107595142
Floating adds/sec       :  5.01M      F.P. adds        :     2309133214
Floating multiplies/sec :  2.55M      F.P. multiplies  :     1174055914
Floating reciprocal/sec :  0.38M      F.P. reciprocals :      172761101
I/O mem. references/sec :  0.03M      I/O references   :       13946799
CPU mem. references/sec : 19.84M      CPU references   :     9139607573

Floating ops/CPU second :  7.94M
```

Figure 6.2 hpm Statistics for Combined Version of CEM VII

Table 6.1 CPU Usage in Combined Version of CEM

Flowtrace Statistics Report Showing Routines Sorted by CPU Time (Descending) (CPU Times are Shown in Seconds)					
Routine Name	**Total Time**	**Number of Calls**	**Average Time / Call**	**Individual Percentage**	**Cumulative Percentage**
ATCALC	2.30E+02	17281	1.33E-02	48.74	48.74
SORT	3.75E+01	417656	8.98E-05	7.94	56.68
TNKAPC	3.48E+01	68905	5.05E-04	7.37	64.05
ARTCAS	3.06E+01	17281	1.77E-03	6.48	70.53
RNDFRD	2.05E+01	17281	1.18E-03	4.34	74.86
PH2IFR	1.29E+01	69226	1.87E-04	2.74	77.61
PH2DFB	9.47E+00	69226	1.37E-04	2.01	79.61
PH2DFR	8.53E+00	69226	1.23E-04	1.81	81.42
DIVRPT	8.48E+00	128	6.63E-02	1.80	83.22
PH2IFB	8.12E+00	69226	1.17E-04	1.72	84.94
ASSESS	7.25E+00	128	5.67E-02	1.54	86.47
CASL	7.12E+00	17281	4.12E-04	1.51	87.98
ESTOUT	6.41E+00	85573	7.49E-05	1.36	89.34
PQMOD	6.35E+00	95730	6.64E-05	1.35	90.69
STAMAT	4.80E+00	95730	5.01E-05	1.02	91.70
DDEND	4.34E+00	128	3.39E-02	0.92	92.62

Table 6.1 CPU Usage in Combined Version of CEM

Flowtrace Statistics Report Showing Routines Sorted by CPU Time (Descending) (CPU Times are Shown in Seconds)					
Routine Name	**Total Time**	**Number of Calls**	**Average Time / Call**	**Individual Percentage**	**Cumulative Percentage**
DBSTAT	4.05E+00	14613	2.77E-04	0.86	93.48
CRQMNT	3.44E+00	53192	6.47E-05	0.73	94.21
CB	2.15E+00	17281	1.24E-04	0.45	94.67
UNITS	1.70E+00	37359	4.56E-05	0.36	95.03

Note that subroutine ATCALC, the subroutines for sorting (SORT,) direct fire (PH2DFB & PH2DFR,) and indirect fire (PH2IFB & PH2IFR) are still within the top 20. Because these subroutines comprise the kernel of the program, they will always constitute a large portion of the run time. But it is important to note that the total time which these subroutines use has been reduced. The main goal of this project was to reduce the run-time of the program. While this goal has been meet so far, additional work could possibly increase the performance even more.

7. Conclusions

Table 7.1 Increase in Performance Resulting from Changes

Modification	Resulting Increase in Performance
Vectorization	7.7%
Elimination of "data packing"	6.9%
Combination	**14.9%**

The CEM VII program, while highly computation intensive, is limited to relatively low performance due to the short vector lengths that exist in CEM VII. However, by introducing a higher degree of vectorization in to the kernel of the program, and removing an obsolete process called "Data Packing," it was possible to increase the performance of the code.

By implementing a concept called "gather-scatter" it was possible to attain a high degree of vectorization in the direct fire subroutines. Due to the complex nature of the indirect fire subroutine, the degree of vectorization that was attained in the direct fire subroutines was not attained. The original sort routine was replaced with a modified Batchers sort. Due to the difference in non-determinism of the two sort routines, the results obtained from the new code are slightly different than the original code, but of little consequence. All in all, the vectorization of CEM VII resulted in a 7.7% increase in performance.

Because early computers had relatively small internal memories, a process called "Data Packing" was implemented in the CEM VII code. Through the elimination of "Data Packing," the performance of the original CEM VII code was increased by 6.9%.

Combining the vectorization of CEM VII with the elimination of "Data Packing" resulted in a new CEM VII that performs 14.9% better than the original CEM VII. A larger increase in performance was anticipated, but due to the relatively small vector lengths that exist in CEM VII, the full capabilities of a Cray YMP's vector hardware could not be utilized.

8. Recommendations

As far as continued work toward vectorization goes, a restructuring of how the indirect fire calculations are performed could possibly allow the indirect fire subroutines to attain a higher degree of vectorization. Specifically, the calculations need to be done on a weapon basis rather that a vehicle basis.

Additionally, it may be possible to vectorize over engagements, in the attempt to increase vector lengths. This would involve gathering information for all engagements occurring across the entire FEBA at a given time step. For example, if there are 20 engagements occurring during a time step and the average vector length for an engagement is 18, vectorizing over engagements could produce vector lengths, on average, of 360. Referring to Figure 4.11, an estimated Mflops ratings around 35 is found, if the density remains around 0.2. This would be a considerable improvement over the 7 Mflops that is observed for the present vectorized version of CEM VII.

While extreme performance enhancements are unlikely in other parts of the code, it would be possible to rewrite many of the loops in other parts of the code to obtain vectorization and attain a slightly higher performance.

Due to the short vector lengths existing in CEM VII, it may be more beneficial to run the CEM VII program on a fast scalar machine. This may be advantageous since scalar machines are much cheaper than vector machines. However, it may be difficult to obtain

the same results on a scalar machine since the precision on a scalar machines is usually different than that on vector machines.

References

1. Allison, W.T., Devlin, H., and Johnson, R.E., 1985. *Concepts Evaluation Model VII (CEM VII), Volume II - User's Handbook*, CAA-D-85-1, US Army Concepts Analysis Agency, Bethesda, MD, rev. December 1991.

2. Johnson, R.E., 1985. *Concepts Evaluation Model VI (CEM VI), Volume I - Technical Description*, CAA-D-85-1, US Army Concepts Analysis Agency, Bethesda, MD, rev. October 1987.

3. Jones, H.W., 1991. *Attrition Calibration (ATCAL) Evaluation Phase I - Direct Fire (ATVAL PHASE I)*, CAA-SR-91-10, US Army Concepts Analysis Agency, Bethesda, MD.

4. Knuth, D., 1973. *The Art of Computer Programming, Volume 3 - Sorting and Searching*, Addison-Wesley Publishing Company.

5. Shivaswamy, K., Thesis, 1994. *Advanced Techniques For Interpreting Terrain and Force Composition in Concepts Evaluation Model Seven (CEM VII)*, MS Thesis, Department of Mechanical Engineering, Colorado State University.

List of All Participating Scientific Personnel and Degrees Awarded

ARO Contract DAAL03-92-G-0176

1. Graduate Students

Mr. Michael Brewer
MS Candidate in Mechanical Engineering (no degree awarded)

Mr. Kiran Shivaswamy
MS in Mechanical Engineering, Spring 1994

Mr. Shane Ballard
MS in Mechanical Engineering, Spring 1995

Mr. Suki Nagesh
MS in Mechanical Engineering, Expected Summer 1995

2. Faculty

Dr. David Aliciatore
Assistant Professor of Mechanical Engineering

Dr. Patrick J. Burns
Professor of Mechanical Engineering

All work was conducted and all degrees were awarded at Colorado State University.