ANL/TM--484

DE91 002956

THE ADVANCED SOFTWARE DEVELOPMENT AND COMMERCIALIZATION PROJECT[1]
Progress Report PR-1

T. R. Canfield, M. Minkoff, C. Mueller, E. Plaskacz, D. P. Weber,
D. M. Anderson, I. U. Therios
Computing and Telecommunications Division
Argonne National Laboratory
9700 South Cass Ave., Argonne, IL 60439-4844

S. Aslam, R. Bramley, H.-C. Chen, G. Cybenko,
E. Gallopoulos, H. Gao, A. Malony and A. Sameh
Center for Supercomputing Research and Development
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801-2932

E. Gallopoulos, editor

September 1990

[1]Work supported by the State of Illinois Technology Challenge Grant, Grant No. 90-82144 with additional support from the National Science Foundation Grant No. CCR900000N for the use of the Cray X-MP/48 at the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign.

Contents

List of Tables

Chapter 1

Introduction

This is the first of a series of reports pertaining to progress in the *Advanced Software Development and Commercialization Project*, a joint collaborative effort between the Center for Supercomputing Research and Development of the University of Illinois and the Computing and Telecommunications Division of Argonne National Laboratory.

The purpose of this work is to apply techniques of parallel computing that were pioneered by University of Illinois researchers to mature computational fluid dynamics (CFD) and structural dynamics (SD) computer codes developed at Argonne.

The collaboration in this project will bring this unique combination of expertise to bear, for the first time, on industrially important problems. By so doing, it will expose the strengths and weaknesses of existing techniques for parallelizing programs and will identify those problems that need to be solved in order to enable wide spread production use of parallel computers. Secondly, the increased efficiency of the CFD and SD codes themselves will enable the simulation of larger, more accurate engineering models that involve fluid and structural dynamics. Such an enhanced capability is fundamental to industrial efficiency and competitiveness, and could serve as an exemplary model for similar future activities.

In order to realize the above two goals, we are considering two production codes that have been developed at ANL and are widely used by both industry and Universities. These are COMMIX and WHAMS-3D. The first is a computational fluid dynamics code that is used for both nuclear reactor design and safety and as a design tool for the casting industry. The second is a three-dimensional structural dynamics code used in nuclear reactor safety as well as crashworthiness studies. These codes are currently available for both sequential and vector computers only. Our main goal is to port and optimize these two codes on shared memory multiprocessors. In so doing, we shall establish a process that can be followed in optimizing other sequential or vector engineering codes for parallel processors.

1.1 Summary progress for first phase

We have completed the first phase of tasks and deliverables specified in the work plan shown in Appendix A, as copied from the Proposal of the Advanced Software Development and Commercialization Project funded by the State of Illinois Technology Challenge Grant, Grant No. 90-82144.

We summarize our accomplishments here.

1. The codes have been ported to the target multiprocessor machines (Cray X-MP/48 at NCSA,

Alliant FX/8(0) at CSRD and ANL).

2. Appropriate data sets have been selected to test the codes' performance. These fully exercise all aspects of the codes, in anticipation for the industrial data sets to be obtained in the context of close cooperation with Illinois industry (see below).

3. It was stated from the beginning that our main objective is to demonstrate the effectiveness of our techniques for problems of industrial application. We have established collaboration with Commonwealth Edison and the Thermal Hydraulics Section of their Nuclear Fuel Services Department has given to ANL an important industrial data set for one of our CFD codes (COMMIX 1AR/P). This problem requires a 12 hour simulation when performed on one vector processor of Commonwealth Edison's IBM 3090. Discussions are underway with General Motors, Inland Steel and other companies to secure inputs and collaboration for the structural dynamics code.

4. The codes have been profiled for the first time, their baseline (unoptimized) performance on each of the target machines was recorded, and the most time consuming subroutines have been identified.

5. We have already started applying automatic restructuring tools on some of the codes.

Overall the first phase has shown that significant improvements in the codes' performance result from vectorization. This is partly because of the effectiveness of vectorizing compilers, and partly because the principles of vectorization have been available to programmers for over 15 years. The first phase also shows that applying existing automatic restructurers for multiprocessing gives little improvement or even degradation in performance, but provides clues on how to achieve better results. This emphasizes the need to move the highly successful tools and techniques for multiprocessing from our research centers to the industrial base.

1.2 Acknowledgments

The authors would like to thank R. N. Blomquist, H. M. Domanus, E. M. Gelbard and J. M. Kennedy from Argonne National Laboratory, T. Belytschko from Northwestern Univ., and M. Berry from CSRD for their help during this project.

Chapter 2

Computational Environment

As described earlier, our work is oriented towards standard and mini supercomputers with multiprocessing capabilities. By all accounts these are the machines which are able to sustain the performance requirements for production runs with the complex codes we are dealing with.

For this phase of the experiments we have used the Cray and Alliant machines shown in Table 2.1. In some cases we also report results from runs performed on Sun Sparcs at ANL, due to the natural interest in understanding the performance on what is very likely to be on a scientist's desk, namely a fast workstation used for code development.

A planned addition to the machines of Table 2.1 is the CSRD Alliant FX/2800, whose 28 processors based upon the i860 RISC chip will be an interesting case study of parallelism and software emulated vector processing.

2.1 Methodology

Our overall strategy in these experiments has been the following: We first compile and link on the target machine. If baseline performance is needed we run the code to obtain baseline timing and validate runs. We then compile under different options described in Section 2.2 to test the effect of automatic parallelization and vectorization. The performance for each run was analyzed with the tools described in Section 2.3.

Table 2.1: Computational engines.

Machine	*O/S*	Memory	Location
CRAY X-MP/14	Unicos	4 MW	ANL
CRAY X-MP/48	Unicos	8 MW	NCSA
ALLIANT FX/80	Concentrix/Xylem	80 MB	CSRD
ALLIANT FX/8	Concentrix	64 MB	ANL

2.2 Compiler and restructurer switches

In this section we provide a summary of some of the compilation options used when performing the experiments. We note that since most of these options at some stage invoke source-to-source restructuring compilers, thus attempting automatic vectorization and parallelization, their use and our comments also belong to Phase 2 of this work.

No hand optimizations were performed.

2.2.1 Cray-X/MP48

The following optimization modes were used for our benchmarks.

Symbol	Command options	Optimizations
S	`cft77 -o novector`	scalar optimization
SV	`cf77 -c -Zc`	scalar and vector OR
	`cft77 -o`	
SV(Zv)	`cf77 -c -Zv`	enhanced scalar and vector
SVC	`cf77 -Zp`	scalar, vector, concurrent (autotasking)

With no options specified, the compiler attempts vectorization of the innermost loop. The `-Zv` option causes the compiler to invoke the dependency analyzer FPP, which attempts more complicated data dependence analyses and inserts directives for vectorization. Although it also inserts directives for microtasking, it does not interpret them (see below).

Both the `cft77` and `cf77` commands invoke the CFT77 compiler; `cf77` also includes the load step for creating an executable file. Otherwise, `cf77` uses most of the options available with the `cft77` command. The `-o` is an optimization option which, by default, implies both scalar optimization and vectorization. When `-o` is followed by the word `novector`, vectorization is suppressed and only scalar optimization is performed. The option `-Zp` causes the compiler to invoke FPP, which inserts directives for vectorization and microtasking. This automatic detection of and instrumentation for microtasking is called *autotasking*.

Although no hand optimizations were attempted, there were a few cases, documented in Section 3.2, where the code contained inline optimization directives for the Cray compiler.

2.2.2 Alliant FX/8[0]

The code was compiled with Alliant's Fortran compiler and VAST restructurer, with switches selected from the following:

Symbol	Command options	Optimizations
S	`fortran [-[D]AS] [-r8] -Og`	global scalar
SV	`fortran [-[D]AS] [-r8] -Ogv`	global scalar, vector
SC	`fortran [-[D]AS] [-r8] -Ogc`	global scalar, concurrent
SVC	`fortran [-[D]AS] [-r8] -O[gvc]`	global scalar, vector, concurrent

The option `-[D]AS` means that transformations for optimizing recurrences (based on the currently available number of processors, if the `D` option is also in effect) can use the associative law of addition. Unless stated otherwise, the use of this option will be assumed.

The option `-r8` forces the transformation of all `REAL` variables and intrinsics into `REAL*8`.

We note that Cray's FPP preprocessor is very similar to Alliant's VAST, as they are both products of Pacific Sierra, purchased by the respective computer companies to restructure code specifically for the corresponding machines. Although options and defaults differ, restructuring is done based on essentially the same data dependence information.

2.3 Performance evaluation tools

Unless mentioned otherwise, all times are in seconds.

2.3.1 Cray X-MP

Performance utilities such as FLOWTRACE, HPM, PERFTRACE on the subroutine level, and PERFTRACE on the loopnest level are available on the Cray X-MP [6]. FLOWTRACE, HPM, and PERFTRACE on the subroutine level were developed by Cray Research, Inc.; PERFTRACE on the loopnest level for the Cray X-MP/48 was developed by John Larson at NCSA.

FLOWTRACE generates printed information about all procedure calls in a program; its summary contains the following information:

- The time spent in each routines: amount, percentage of the total execution time, and average time per call.
- Number of calls to each procedure.
- Lists of routines that call and are called by each routine.
- A dynamic call tree of the main program and all called subprograms.

HPM reports performance of the entire program. It can issue any of four kinds of reports, named groups 0, 1, 2 and 3.

Group 0 Scalar activity: number of instructions, memory references, floating-point additions, multiplications, reciprocals, MFLOP.

Group 1 Conditions that delay instruction issue: Percent of time (in clock periods) waiting on resources such as semaphores, shared registers, scalar, vector and address registers functional units, scalar and block memory references.

Group 2 Information on central memory references and conflicts.

Group 3 Instruction types and vector operations.

PERFTRACE gives the same type of statistics about computer performance as those generated by the HPM, but with details for individual program units.

For the Cray X-MP we ran the codes through all four groups of the Hardware Performance Monitor (HPM). We also compiled and linked with each of FLOWTRACE, PERFTRACE and the profiler PROF for routine-by-routine performance monitoring.

As a start of Phase 2 advanced performance analysis, we have generated some preliminary traces of COMMIX-1AR/P and WHAMS-3D. The goal is to study performance behavior at a more refined level using trace data of routine entry and exit actions. We used tracing tools developed for the Cray X-MP and Cray 2 which are described in [9]. In summary these tools can capture detailed histories of routine invocation together with machine performance statistics.

2.3.2 Alliant FX/8[0]

The tools for performing performance analysis of the codes running on the Alliant FX/8[0] were the standard Alliant facility GPROF and calls to the `etime()` facilities.

The utility GPROF is used to obtain execution profiles of FX/Fortran codes. For each routine, GPROF counts the number of times it was called and determines the time elapsed in its execution.

Chapter 3

COMMIX code

COMMIX is one of the world's premier thermal-hydraulics codes, used at scores of government and industrial sites in support of a vast range of research and development projects. Argonne's development and refinement of COMMIX, which has continued for more than ten years, was originally supported by the U.S. Nuclear Regulatory Commission for application to a wide variety of reactor safety problems. COMMIX has been developed using a unique porous media approach to the solution of the Navier-Stokes equations in an arbitrary three-dimensional region. In its various versions, COMMIX can model separate single-phase fluids, multiphase flows, and free surface flows. The code uses differenced momentum/mass conservation equations which are combined to form a pressure equation. Once the pressures are known, the fluid velocities are updated to provide input to the energy simulation and the next iteration or time step. The hydraulic driving force may be flow or pressure boundary conditions at inlets and outlets, one of several pump models, or a fluid temperature/density distribution. The energy equations are differenced using the updated velocities, and the source terms are accumulated from the treatments of convection boundaries, conduction boundaries, thermal structures, or heat generation in the fluid itself.

One-dimensional shell structures superimposed on the fluid geometry model various fluid system thermal components such as vessels, pipes, baffles, tube-shell heat exchangers, and reactor fuel. Once the fluid temperature distribution is updated, the submerged thermal structures' internal temperature distributions are recomputed, assuming one-dimensional conduction through each thermal structure segment.

The momentum and fluid energy equation time differencing is implicit, which requires that the difference equation coefficients be constructed from end-of-timestep temperatures and velocities. Since these are not known when the coefficients are computed, a set of "outer" iterations is completed in which the momentum and energy equations' coefficients are computed from ever-better estimates of end-of-step values, a process repeated until the "outers" converge to the end-of-step solution. Each outer iteration consists of pressure matrix equation construction, pressure equation solution, velocity update, energy matrix equation construction, energy equation solution, and, finally, a structure- fluid heat flux update. Only when the time step converges are the radiation heat fluxes between thermal structures updated.

The COMMIX code exists in two versions, COMMIX-1C and COMMIX 1AR/P. The COMMIX family of codes were developed to analyze steady-state/transient, single-phase, three-dimensional compressible/incompressible flow with heat transfer in a reactor system. These codes are also applicable to a broad range of applications. Due to the wide range of applications fundamentally different codes have evolved out of a common software ancestor. We are using COMMIX-1AR/P

and COMMIX-1C, both of which are the latest derivatives of the COMMIX-1A code and its predecessors. The COMMIX-1A code was designed for thermal-hydraulic analysis of reactor components. It solves the conservation equations of mass, momentum, and energy as a boundary-value differential equation in space and an initial-value problem in time. Spatial discretization is accomplished by a staggered grid system to describe field variables at cell centers and flow variables at cell surfaces. The codes described below represent totally different extensions in both modeling capabilities and targeted computer architectures.

COMMIX-1AR/P is based on COMMIX-1A and contains new models and formulations which were added to the code over a period of about 5 years [5]. These include a pump model, radiation heat transfer, boundary conditions for inlet flow as a function of radiation surface temperature and to simulate expansion cells for constant mass calculations, multiple fluid capability, conjugate gradient solution technique for momentum/mass equations, implicit coupling of thermal structures to fluid, and change-based automatic time step control. It is run primarily on the Cray X-MP/14, and is substantially vectorized.

COMMIX-1C is based on COMMIX-1B [1, 2] (and ultimately on COMMIX-1A). It applies three new models to treat turbulence effects including the two-equation $k-\epsilon$ model which has been discretized to simulate subsonic compressible flow. A new porous-medium formation was developed which can be used to model anisotropic flow with stationary structures. The flow-modulated skew-upwind differencing scheme has been implemented to reduce numerically-induced diffusion of scalar transport. Other distinctive options include transient mass flow boundary conditions and application of direct solution of sparse matrix equations. It is run primarily on Sparc workstations at ANL.

3.1 Data sets

The data sets used for these experiments are routinely applied by ANL for testing the validity of any modifications to the code. For that reason they are designed to fully exercise the code and are very suitable for the goals of the first phase. We here summarize the results obtained from one data set for COMMIX 1C and COMMIX 1AR/P.

1AR/P data sets

Data set P1 Simulates two-fluid (sodium, air) flow and heat transfer in the 90 degree sector of a generic modular pool-type liquid metal (sodium) reactor. This transient simulates the reactor system's response to a postulate pumping failure in which the pump ramps to zero power linearly over five seconds, and the reactor power ramps to 7% (initial decay heat) during the first ten seconds. Cylindrical coordinates (r, θ, z) are used. The computational domain consists of 22 unique surface types, 330 (regular) surface elements and 205 computational cells for a maximum of 8, 3, and 12 cells in the r, θ and z directions respectively. The data set consists of four input decks corresponding to the following phases:

P1r0 Cold Start to Steady State (1000 time steps);

P1r1 Restart of Steady State (100 time steps);

P1r2 Transient Problem from Steady State (1100 time steps);

P1r3 Restart of Transient Problem (400 time steps).

Data set P2 This data set simulates the steady-state behavior of an experiment performed at Karlsruhe, Germany, to study the transient behavior of a seven-pin assembly during a flow transient. There are 48 axial meshes, and each pin and its adjacent coolant is represented by four x-y cells. The sodium flows vertically through the pin bundle, which is enclosed in a hexagon steel can. The sodium is heated by the pins as it flows upwards. The transient analyzed (not in this data set) is a linear flowrate decrease, and the event of interest is the time of odium boiling onset. Rectangular geometry is used for 6 unique surface types, 594 surface elements (some of which are irregular), 432 computational cells, for a maximum of 3, 3, and 48 cells in each coordinate direction. There are 4 force structures.

Data set P3 The object of the simulation is to determine in steady-state the degree of thermal relaxation in the Clinch river breeder reactor outlet plenum above the core. The sodium exits reactor subassemblies of three types (driver, blanket, or control), each with a specific design, power, and flow rate. The mitigation of thermal stress and shock (during transients) depends on the mixing of relatively hotter sodium with the cooler sodium flowing out of the neighboring subassemblies. This issue is also being addressed in the case of state-of-the-art liquid metal cooled reactor designs. The geometry is cylindrical, with 11 unique surfaces, 446 (regular) surface elements, and 346 computational cells, for a maximum of 6, 3, and 23 cells along each coordinate direction.

1C data set This data set was used as input for the 1C version of COMMIX. The TMLB' is one of the postulated reactor accidents that is currently being investigated by the U.S. Nuclear Regulatory Commission. In this accident, several different significant events and physical phenomena occur. During the progression of the TMLB' accident scenario, there is a time when the hot leg dries out and the core becomes uncovered. From that time on, multidimensional natural-circulation phenomena play an important role in heat transport and the heat-up of the various components in a reactor system. The multidimensional capabilities of the code make possible the simulation of the natural convection phenomena which are probable in the TMLB' scenario. The generated flow patterns, temperature distributions, and steam generator heat transfer rates provide useful guidance for simulation of one-dimensional systems. This data is needed to support the system analyses being performed at Los Alamos, Sandia and Idaho National Laboratories. In the transient that is simulated, the entire system at time $t = 0$ is isothermal, i.e., it contains saturated steam at $p = 1.61 \times 10^7$Pa. For time $t > 0$, decay heat was added to the core. While natural convection flow pattern was being established, the system was being perturbed by the opening and closing of the PORV value. The analysis was performed using a PORV model that is at the end of the surge line that is connected to the pressurizer. The geometry is a Cartesian box, with 1606 surface elements and 947 computational cells. The pressure equation is solved by means of the Yale Sparse Matrix Package (YSMP). For this test case, a constant turbulent viscosity model is used. There are three decks of interest:

C1r0 Run to steady state;

C1r1 137 time steps;

C1r2 45 time steps.

Consistent with the goals of this project, to demonstrate the benefits from using multiprocessor architectures on large important codes used by Illinois industry, a data set for a real world problem

has been obtained from the Thermal Hydraulics Section of Commonwealth Edison's Nuclear Fuel Services Department. This data set, which uses COMMIX to verify the conservatism in RETRAN licensing calculations which simulate the reactor's response to a steamline break accident, and takes up to twelve hours of simulation when performed on Commonwealth Edison's IBM 3090 vector processor, will be described in the context of our efforts in future phases of this work.

3.2 Results from COMMIX 1AR/P

As a reference point for the 1AR/P code we obtained the *baseline* performance for this data set, meaning that the code was compiled and ran without any automatic optimization options applied to it. The code was originally written so that it runs best on architectures with vector processing. The pressure equations of the mass-momentum loop are solved using a preconditioned conjugate gradient technique, which converges at an acceptable rate without requiring any sensitive iteration parameters from the user. Incomplete Cholesky factorization approximates the matrix inverse, and is mostly vectorized, but with vector lengths frequently far from optimum. No attempt has been made to exploit parallelization here. A large amount of the computing time is spent constructing matrix equations, a process which would require massive recoding to vectorize because of its large, logic-loaded loops. Such loops, however, are expected to lend themselves very well to the parallelization efforts which we plan to pursue in the context of this work.

From the 1AR/P data sets we described, we observed that the results obtained when using the decks P1r0 and P1r2 (begin steady state and begin transient) were very similar to those obtained from the restart decks 1 and 3. Moreover, although data sets P2 and P3 are useful for code development purposes, they exercised the code less than P1. Since we did not want to clutter the report with tables, we decided to provide only data from the most interesting and representative experiments. This means that, except for the baseline runs, we usually omit any information from P1r1, P1r3 and P2, P3.

Overall the code consists of six major groups of computations:

1. Momentum-related equations construction.
2. Momentum-related equations solution.
3. Energy-related equations construction.
4. Energy-related equations solution.
5. Thermal structure temperatures computation.
6. Thermal structure radiative heat flux computation.

We next list the function of the most important routines in the set:

ENERGI Construct coefficients in the energy equation.

LOWFCV Solve the upper triangular system as part of the conjugate gradient solution of the pressure equation.

PEQN Construct the coefficients in the pressure equation.

QSTRDS Calculates finite differences of solid/fluid heat transfer rate over the thermal structures.

QSTRUC Set solid-to-fluid source term for the fluid energy equation.

SOLVEV Solver of linear system for the energy equation using Gauss-Seidel relaxation on red-black ordering.

XMOMI, YMOMI, ZMOMI Sweep over all fluid cells to set-up the x, y and z direction momentum equations.

The original code is instrumented with calls to the Cray intrinsic function `second` to summarize run times for each of its major steps. Indeed, whenever presenting timing results for the above stages, these results were derived from the original code instrumentation.

As the code was written to take advantage of vector processing in some of its solver routines and in anticipation of the great costs involved otherwise, we decided *not to explicitly disable vectorization for the baseline runs.*

We note that the only subroutines of the original code containing inline Cray compiler directive lines `CDIR$` were DAXPXC, DAXPYC and LOWFCV. In those routines vectorization was helped using the `IVDEP` directive, which causes the compiler to ignore vector dependencies in its attempts to vectorize the corresponding `DO` loops. The effect of this is clearly seen in the performance results presented in Section 3.2.1.

For future reference, we first show Table 3.1, which summarizes the runtimes for each of the machines and compilation options for data sets P1r0 and P1r2.

Table 3.1: Execution times for SV, SV(Zv) and SVC COMMIX-1AR/P on the Cray X-MP/48 and Alliant FX/80.

Routine	P1r0	P1r2
Cray SV	93.96	920.94
Cray SV(Zv)	81.48	872.90
Cray SVC	225.25	2025.48
1 CPU	22.35	297.28
2 CPU	31.76	405.18
3 CPU	47.19	461.88
4 CPU	123.95	861.20
Alliant FX/80		
SV (1 CE)	1,322.4	12,331.2
SC (8 CE)	1,128.7	10,015.0
SVC (8 CE)	1,136.7	10,048.6

3.2.1 Cray X-MP

First we summarize the baseline performance obtained for each of the data sets. As mentioned earlier, the code was written to take take advantage of the vector processing capabilities of a single processor of the Cray X-MP. For the purpose of 1AR/P we thus consider as baseline the performance obtained from the code in SV compilation mode (cf. Section 2.2). When appropriate, in the tables

of this section, we mention the performance tool used to obtain the results (e.g. PERFTRACE, FLOWTRACE, etc.)

3.2.1.1 Summary of baseline runs

We first report results from the baseline runs (mode SV), as needed to satisfy our milestones for the first phase (cf. Appendix A).

The runtimes for each of the data sets are summarized in Table 3.2.

Table 3.2: Execution times and MFLOPS for COMMIX-1AR/P baseline SV performance on Cray X-MP/14 (from HPM).

Data set	P1r0	P1r1	P1r2	P1r3	P2	P3
Time	93.96	9.44	920.94	665.10	17.84	42.24
MFLOPS	9.12	7.93	7.68	7.14	11.43	10.02

We next order the most time-consuming subroutines for each of the data sets except P1r3, and report their MFLOPS, the number of times they were called, and the percentage of time (clock periods) spent on each. This data is obtained from PERFTRACE and shown in Tables 3.3–3.7.

3.2.1.2 Enhanced optimization options and further results

As mentioned earlier, by using special (automatic) restructuring options, the results in this section could also be considered as Phase 2 results. Since they were available however, we thought that it is appropriate to report them here.

Table 3.13 summarizes the execution times and MFLOPS for the code under SV(Zv) mode of compilation. We notice an improvement over the baseline times of Table 3.2, implying that the FPP optimizations were effective in certain cases. As will be seen, the greatest effect is seen in the SOLVEV routine.

We show in Table 3.14 the breakdown of times for each phase of the computation, when compiled with the SV(Zv) option.

COMMIX 1AR/P consists of approximately 150 subroutines. Table 3.15 shows that for those data sets examined, the nine listed subroutines consume over 60% of the time in all cases. Comparing with the baseline runs presented in Tables 3.3-3.5, we notice that the percentage of time spent in SOLVEV is almost halved by the additional vectorization that is achieved after preprocessing with the dependency analyzer FPP. This gives an indication of the advantages one can sometimes achieve when using more advanced automatic vectorization techniques.

Comparing Table 3.15 with Tables 3.3-3.5, we note that the small differences in the number of calls shown for some subroutines is due to the different methods of accounting used by PERFTRACE and FLOWTRACE.

Table 3.16 shows how the best MFLOPS rate is achieved for the SOLVEV subroutine. With the exception of LOWFCV – whose relative weight in the total runtime of the code is much smaller than SOLVEV's – the achieved rate is far superior than all other listed subroutines.

Regarding LOWFCV, we note that its superior performance for the simple SV compilation option, is due to the use of the IVDEP inline Cray compiler directives CDIR$.

Table 3.3: Characteristics of most time-consuming subroutines of COMMIX-1AR/P for data set P1r0 running in baseline SV mode on Cray X-MP (from PERFTRACE).

Rank	Program Unit	Times Called	% Execute	MFLOPS
1	SOLVEV	1000	14.76	10.29
2	QSTRDS	1001	11.52	6.20
3	ENERGI	1000	10.97	6.06
4	ZMOMI	1000	9.43	6.42
5	YMOMI	1000	6.18	6.97
6	QSTRUC	1001	4.68	6.67
7	PEQN	1000	4.53	3.05
8	XMOMI	1000	4.31	6.93
9	LOWFCV	26303	3.78	30.98
10	HSTRUC	1001	3.44	2.13

Table 3.4: Characteristics of most time-consuming subroutines of COMMIX-1AR/P for data set P1r1 running in baseline SV mode on Cray X-MP (from PERFTRACE).

Rank	Program Unit	Times Called	% Execute	MFLOPS
1	SOLVEV	100	18.31	10.26
2	QSTRDS	100	11.48	6.20
3	ENERGI	100	10.94	6.06
4	ZMOMI	100	9.40	6.42
5	YMOMI	100	6.16	6.97
6	QSTRUC	100	4.66	6.67
7	PEQN	100	4.52	3.05
8	XMOMI	100	4.30	6.93

Table 3.5: Characteristics of most time-consuming subroutines for data set P1r2 running in baseline SV mode on Cray X-MP (from PERFTRACE).

Rank	Program Unit	Times Called	% Execute	MFLOPS
1	QSTRDS	12998	15.18	6.19
2	ENERGI	12998	14.45	6.06
3	ZMOMI	12998	12.41	6.42
4	YMOMI	12998	8.13	6.97
5	QSTRUC	12998	6.17	6.66
6	PEQN	12998	5.96	3.06
7	XMOMI	12998	5.67	6.93
8	HSTRUC	12998	4.53	2.15

Table 3.6: Characteristics of most time-consuming subroutines for data set P2 running in baseline SV mode on Cray X-MP (from PERFTRACE).

Rank	Program Unit	Times Called	% Execute	MFLOPS
1	QSTRDS	65	10.32	7.71
2	ENERGI	64	8.27	2.84
3	HSTRUC	65	7.17	1.12
4	ZMOMI	64	7.15	5.33
5	DUCTWA	130	6.97	1.70
6	TLIQ	51321	6.85	1.73
7	QSTRUC	65	5.31	7.46
8	LOWFCV	3961	5.02	22.53

Table 3.7: Characteristics of most time-consuming subroutines for data set P3 running in baseline SV mode on Cray X-MP (from PERFTRACE).

Rank	Program Unit	Times Called	% Execute	MFLOPS
1	ZMOMI	459	16.12	6.69
2	ENERGI	459	15.18	3.92
3	XMOMI	459	12.37	7.31
4	YMOMI	459	10.56	7.25
5	PEQN	459	7.38	3.25
6	SOLVEV	459	4.94	35.87
7	SORTC	459	4.60	0.00
8	LOWFCV	9090	4.15	30.15

Table 3.8: HPM group 0 summary for baseline SV COMMIX-1AR/P code running on Cray X-MP/14 with data sets P1, P2 and P3.

	P1r0	P1r1	P1r2	P1r3	P2	P3
Million inst/sec (MIPS)	37.23	37.91	36.45	36.62	35.70	35.98
Avg. clock periods/inst	3.16	3.10	3.23	3.21	3.30	3.27
% CP holding issue	49.16	48.0	48.75	48.3(	50.67	49.45
Inst.buffer fetches/sec	0.40M	0.40M	0.47M	0.47M	0.47M	0.47M
Floating adds/sec	3.77M	3.14	3.20M	2.94M	4.46M	4.04M
Floating multiplies/sec	4.84M	4.28M	4.02M	3.74	6.31M	5.45M
Floating reciprocal/sec	0.51	0.51	0.46	0.45M	0.65M	0.53M
I/O mem. references/sec	0.57M	0.55M	0.55M	0.70M	0.11M	0.37M
CPU mem. references/sec	16.32M	13.93M	13.30M	12.15M	18.70M	19.04M
Floating ops/CPU second	9.12M	7.92M	7.68M	7.14M	11.43M	10.02

Table 3.9: HPM group 1 summary for baseline SV COMMIX-1AR/P code running on Cray X-MP/14 with data sets P1, P2 and P3.

	% of all CPs					
	P1r0	P1r1	P1r2	P1r3	P2	P3
Waiting on A-registers/funct. units	9.54	9.61	10.16	10.25	9.42	10.29
Waiting on S-registers/funct. units	27.91	28.35	3.40	31.15	28.74	26.55
Waiting on V-registers	5.39	4.65	2.71	2.81	6.25	5.63
Waiting on vector functional units	3.63	3.24	0.13	2.45	3.69	3.81
Waiting on scalar memory references	0.10	0.10	0.13	0.13	0.19	0.16
Waiting on block memory references	4.35	3.70	2.22	1.77	4.82	5.04

Table 3.10: HPM group 2 summary for baseline SV COMMIX-1AR/P code running on Cray X-MP/14 with data sets P1, P2 and P3.

	% of all CPs					
	P1r0	P1r1	P1r2	P1r3	P2	P3
Inst. buffer fetches/sec	0.40M	0.40M	0.47M	0.47M	0.47M	0.47M
Scalar memory refs/sec	5.79M	5.81M	6.82M	6.95M	5.68M	7.08M
% having conflicts	28.38	28.55	30.61	30.53	42.50	41.06
I/O memory refs/sec	0.48M	0.69M	0.64M	0.57M	0.16M	0.33M
% having conflicts	56.74	49.84	51.37	48.76	71.50	65.61
Block memory refs/sec	10.53M	8.12M	6.47M	5.21M	13.02M	11.96M
% having conflicts	62.43	68.61	75.91	84.24	60.63	73.37
Vector memory refs/sec	10.19M	7.79M	6.15M	4.90M	11.71M	11.70M

Table 3.11: HPM group 3 summary for baseline SV COMMIX-1AR/P code running on Cray X-MP/14 with data set P1.

	P1r0		P1r1		P1r2		P1r3	
type of instruction	inst. per CPU sec	% of all inst.	inst. per CPU sec	% of all inst.	inst. per CPU sec	% of all inst.	inst. per CPU sec	% of all inst.
jump/special	3.47M	9.32	3.64M	9.59	3.15M	8.64	3.18M	8.67
scalar	32.94M	88.50	33.59M	88.57	32.67M	89.64	32.89M	89.80
vector integer/logical	0.19M	0.51	0.21M	0.55	0.20M	0.55	0.20	0.55
vector floating point	0.20M	0.55	0.15M	0.4	0.13M	0.36	0.11M	0.29
vector memory	0.42M	1.12	0.33M	0.8	0.29M	0.81	0.25M	0.69
type of operation	ops per CPUsec	avg. VL	ops per CPUsec	avg. VL	ops per CPUsec	avg. VL	ops per CPUsec	avg. VL
Vector integer&logical	3.22M	16.91	4.23M	20.30	2.12M	10.61	2.04M	10.13
Vector floating point	5.11M	24.99	3.92M	25.57	2.94M	22.32	2.31M	21.53
Vector memory	10.19M	24.41	7.79M	23.41	6.15M	20.87	4.90M	19.35

Table 3.12: HPM group 3 summary for baseline SV COMMIX-1AR/P code running on Cray X-MP/14 with data sets P2 and P3.

type of instruction	P2		P3	
	inst. per CPU sec	% of all inst.	inst. per CPU sec	% of all inst.
jump/special	3.25M	9.10	2.57M	7.13
scalar	31.54M	88.37	32.60M	90.57
vector integer/logical	0.16M	0.46	0.21M	0.58
vector floating point	0.25M	0.70	0.20M	0.56
vector memory	0.49M	1.36	0.42M	1.17
type of operation	ops per CPUsec	avg. VL	ops per CPUsec	avg. VL
Vector integer&logical	3.73M	22.62	3.65M	17.62
Vector floating point	5.71M	22.93	5.88M	29.26
Vector memory	11.70M	24.03	11.70M	27.88

Table 3.13: Execution times and MFLOPS for SV(Zv) COMMIX-1AR/P code running on Cray X-MP/48 (from HPM).

	P1r0	P1r1	P1r2	P1r3
Time	82.35	8.17	880.64	628.17
MFLOPS	11.76	10.96	8.76	

Table 3.14: Timing profile for COMMIX-1AR/P running on Cray X-MP/48, compiled in SV(Zv) mode.

Data:	P1r0	P1r1	P1r2	P1r3
Total Execution Time	77.71	7.64	869.04	622.37
Momentum Eqns. Construction	21.93	2.12	297.70	204.49
Momentum Eqns. Solution	20.39	1.72	220.46	149.68
Energy Eqns. Construction	25.58	2.56	338.06	247.22
Energy Eqns. Solution	6.48	.836	18.12	13.13
Thermal Struct	2.81	.281	3.10	1.13
Thermal Struct Radiation	0.065	.0065	.072	0.026

Table 3.15: Profile of the most time consuming subroutines in SV(Zv) mode for COMMIX-1AR/P running on Cray X-MP/48 (from FLOWTRACE).

Routine	P1r0		P1r1		P1r2		P1r3	
	Calls	(percent)	Calls	(percent)	Calls	(percent)	Calls	(percent)
QSTRDS	1001	11.49	100	11.53	13156	13.99	9629	14.32
ZMOMI	1000	9.69	100	9.69	13156	11.77	9629	12.05
ENERGI	1000	9.52	100	9.64	13156	11.73	9629	12.00
SOLVEV	1000	7.45	100	9.95	13156	1.69	9629	1.72
YMOMI	1000	6.52	100	6.59	13156	8.0	9629	8.19
QSTRUC	1001	5.10	100	5.12	13156	6.22	9629	6.26
PEQN	1000	4.69	100	4.72	13156	5.73	9629	5.87
XMOMI	1000	4.55	100	4.59	13156	5.58	9629	5.71
LOWFCV	26373	4.28	1238	2.10	178939	2.86	89659	2.01

The linear system solver SOLVEV, although expensive in number of computations, has been written to take advantage of vectorization and thus performs at 35 MFLOPS on one CPU of the Cray X-MP. This is in contrast to a meager average of 6.5 MFLOPS for those sections of the code working on the task of matrix construction. This of course causes the overall performance to drop to almost 11 MFLOPS, even for the full vector optimization. It is thus clear that a first step in improving the performance of the code is going to be the restructuring of the matrix construction phase.

We remark here that Table 3.16 reveals a great deal about the nature of work to be done during this project:

- The difference in performance for SOLVEV under the different compilation options tells one about the ability of restructuring compilers to take advantage of the machine capabilities, if the code is written properly. It also shows that the use of more sophisticated restructurers can be very beneficial. We should keep in mind however that SOLVEV was coded for vector processing.

- The high performance of LOWFCV for both SV and SV(Zv) shows that even a less sophisticated restructurer can do well if it has some help from the user (in the form of inline directives).

- The low performance for both SV and SV(Zv) options for all other routines, shows that there is work to be done until these commercial compilers can handle satisfactorily dusty-decks ("dusty-decks" as the matrix assembly routines were not coded to take advantage of the architecture.) We will thus investigate the use of novel restructurers as part of Phase 2 of our work.

- The overall low performance for both compilation options, and the small difference amongst the two, shows that it is dangerous to concentrate only on those stages of the code which are related to well defined algebraic computations in need of new algorithms (e.g. linear system solvers), at least until the automatic transformation tools become more powerful.

Finally, Tables 3.1 and 3.17 show the performance of the code when run in the multiprocessing environments of the Alliant FX/80 and the Cray X-MP/48, with restructuring performed auto-

Table 3.16: MFLOPS profile of the most time consuming subroutines in baseline SV, and enhanced vectorization SV(Zv) modes for the COMMIX-1AR/P code running on Cray X-MP using data decks P1r0 and P1r2 (from PERFTRACE).

Data set	P1r0		P1r2	
Routine	SV	SV(Zv)	SV	SV(Zv)
QSTRDS	6.20	6.68	6.19	6.69
ZMOMI	6.42	7.19	6.42	7.15
ENERGI	6.06	6.61	6.06	6.64
SOLVEV	10.29	36.28	10.37	33.51
YMOMI	6.97	7.78	6.97	7.81
QSTRUC	6.67	7.28	6.66	7.29
PEQN	3.05	3.49	3.06	3.50
XMOMI	6.93	7.82	6.93	7.84
LOWFCV	30.98	34.12	30.99	34.57
Overall	9.12	11.76	7.68	8.76

Table 3.17: Execution time profile for the SVC version of COMMIX-1AR/P running on Cray X-MP/48 (from HPM).

Concurrency	P1r0	P1r2	Connect seconds	Connect×CPUs
1 CPU	22.25	297.28	412.6	412.6
2 CPU	31.76	405.18	303.3	606.5
3 CPU	47.19	461.88	231.9	695.9
4 CPU	123.95	861.20	364.5	1458.1
Totals	225.25	2025.48	1312.4	3173.2

matically by the respective compilers. It is clear that multiprocessing is of little benefit without manual intervention.

Table 3.17 gives the job accounting information for the vector-concurrent version. This information provides the connect time in each active CPU and the number of active CPUs. Let T_i, $i = 1, ..., 4$, be the connect time in each active CPU when i CPUs are active. The total execution time and total CPU time, as shown in the last two rows of this table, are defined as :

$$\begin{aligned} \text{total execution time} &= T_1 + T_2 + T_3 + T_4, \text{ and} \\ \text{total CPU time} &= T_1 + 2T_2 + 3T_3 + 4T_4. \end{aligned} \tag{3.1}$$

The average number of concurrent CPUs is thus the ratio of the total number of connect-seconds and the Connect×CPUs product. Thus 3173.2720/1312.4346 = 2.42.

The following observations can be drawn regarding the statistics reported in Table 3.18.

1. For the performance to be considered acceptable, the MFLOPS (last row in Table 3.18) should be between 20 and 200 MFLOPS; the average MFLOPS for vector SV(Zv) code is about 9

Table 3.18: HPM group 0 summary for SV(Zv) and SVC COMMIX-1AR/P code running on Cray X-MP/48 with data set P1.

Category	SV(Zv)				SVC			
	P1r0	P1r1	P1r2	P1r3	P1r0	P1r1	P1r2	P1r3
Million inst/sec (MIPS)	37.11	36.92	38.62		16.79	19.03	17.83	16.31
Avg. clock periods/inst	3.17	3.19	3.05		7.01	6.18	6.60	7.22
% CP holding issue	52.61	52.49	50.09		78.88	76.25	77.11	79.07
Inst.buffer fetches/sec	0.44M	0.45M	0.47M		0.17M	0.19M	0.21M	0.20M
Floating adds/sec	4.76M	4.17M	3.84M		1.71M	1.63M	1.66M	1.41M
Floating multiplies/sec	6.34M	6.08M	4.41M		2.03M	2.02M	1.85M	1.57M
Floating reciprocal/sec	0.67M	0.70M	0.50M		0.21M	0.24M	0.21M	0.19M
I/O mem. references/sec	0.18M	0.33M	0.49M		0.14M	0.22M	0.23M	0.11M
CPU mem. references/sec	21.12M	19.14M	16.25M		7.62M	7.49M	7.09M	5.98M
Floating ops/CPU second	11.76M	10.96M	8.76M		3.96M	3.88M	3.73M	3.17M

Table 3.19: Ratio of scalar and vector floating operations for COMMIX-1AR/P code running on Cray X-MP/48 with data set P1.

	SV(Zv)				SVC			
	P1r0	P1r1	P1r2	P1r3	P1r0	P1r1	P1r2	P1r3
Ratio	1.62	1.42	0.74		2.85	1.11	0.79	0.71

MFLOPS which is very low. The reported MFLOPS for vector-concurrent (-Zp) code (SVC in our notation) is even lower, an average 3.6. But one has to keep in mind that the code was running on up to four CPUs in multiuser mode. An average of 2.42 CPUs was used for the runs that produced the statistics in the table. The average MFLOPS is thus 8.9 (3.6×2.42) for the SVC version of the code.

2. The MIPS rate for the Cray X-MP should be between 20 and 80. For vector code the average MIPS is about 37. A low value accompanied with high MFLOPS indicates long vector instructions. This is not the case here indicating that code should be vectorized further.

3. The MFLOPS rate reported at the bottom of the table includes scalar and vector floating point operations. An important measure of extent of vectorization is the vector to scalar floating operations ratio. The HPM group 3 statistics report the vector floating point operations. Such data, reported later in Tables 3.22 and 3.23, can be used in conjunction with Table 3.18 to compute the vector/scalar ratio. Such ratios were computed and are presented in Table 3.19. These ratios are very low indicating that the code is spending most of its time in scalar operations.

The following observations can be drawn regarding the statistics reported in Table 3.20.

1. For multitasked (concurrent) code, a very large (55) percentage of the execution time was spent waiting (to synchronize) on semaphores.

2. The figure for "waiting on S-registers/funct. units", i.e., scalar registers and functional units, is also high compared to "waiting on V-registers" and " waiting on vector functional units". This shows that the code is spending more of its time in scalar mode.

Table 3.20: HPM group 1 summary for SV(Zv) and SVC COMMIX-1AR/P code running on Cray X-MP/48 with data set P1.

Category	SV(Zv) % of all CPs				SVC % of all CPs			
	P1r0	P1r1	P1r2	P1r3	P1r0	P1r1	P1r2	P1r3
Waiting on semaphores	0.00	0.00	0.00	0.00	55.63	51.61	58.98	62.47
Waiting on shared registers	0.00	0.00	0.00	0.00	0.18	0.30	0.15	0.15
Waiting on A-registers/funct. units	9.11	9.07	9.76	9.83	3.93	4.14	4.02	3.69
Waiting on S-registers/funct. units	26.46	26.69	29.13	29.62	11.37	12.49	11.89	11.05
Waiting on V-registers	6.75	6.62	3.87	3.32	2.32	2.30	1.44	1.13
Waiting on vector functional units	6.10	6.56	3.82	3.65	1.73	1.79	1.37	1.19
Waiting on scalar memory references	0.89	0.99	1.12	0.81	0.16	0.21	0.16	0.13
Waiting on block memory references	6.40	5.84	4.10	3.43	2.31	2.21	1.47	1.17

The following observations can be drawn regarding the statistics reported in Table 3.21.

1. A figure for "inst. buffer fetches/sec" larger than 1 million per second indicates "spaghetti" code. The numbers reported by group 2 statistics show this is not the case.

2. The comparison of scalar to vector memory references provides a measure for the extent of vectorization. The statistics show a ratio of vector to scalar references ranging from 0.93 to 2.23. This range is low.

3. A high "block memory ref./sec" rate with low scalar and vector memory reference rate would indicate that the code is spending too much time in entry and exit code of the subroutines and functions. This could happen if subroutines were called but did very little work. This is not the case. The block memory reference rate is comparable to vector and scalar reference rate.

The following observations can be drawn regarding the statistics reported in Tables 3.22 and 3.23.

1. The number of scalar instructions is overwhelmingly high, almost 88 percent of all instructions. But this can be misleading because it includes non-floating point instructions also. The ratio of the vector to scalar floating point operations reported in Table 3.19 is more indicative of the level of vectorization.

2. The average length per vector floating-point instruction, as reported in Table 3.22, is about 23, much lower than the length of the vector registers (64) and even lower than 53 and 45, the published measurements of $n_{1/2}$ for dyadic and triadic operations respectively, the vector length required to achieve half the maximum asymptotic performance (in this case taken to be 70 and 103 MFLOPS) [8].

3.2.2 Alliant FX/80

The following changes were needed to port the 1AR/P code.

Table 3.21: HPM group 2 summary for SV(Zv) and SVC COMMIX-1AR/P code running on 4-CPU Cray X-MP/48 with data set P1.

Category	SC(Zv)				SVC			
	P1r0	P1r1	P1r2	P1r3	P1r0	P1r1	P1r2	P1r3
Inst. buffer fetches/sec	0.45M	0.46M	0.48M	0.48M	0.17M	0.17M	0.20M	0.22M
Scalar memory refs/sec	6.50M	6.57M	7.39M	7.52M	2.56M	2.67M	3.05M	3.34M
% having conflicts	47.75	44.69	44.98	43.72	22.5	23.9	27.7	28.5
I/O memory refs/sec	0.18M	0.09M	0.11M	0.30M	0.07M	0.18M	0.16M	0.07M
% having conflicts	43.49	50.83	40.38	41.18	22.6	12.4	29.1	32.2
Block memory refs/sec	14.90M	12.91M	9.11M	7.85M	5.00M	4.20M	3.58M	3.29M
% having conflicts	33.30	32.63	25.42	24.43	20.8	21.1	19.6	18.8
Vector memory refs/sec	14.48M	12.49M	8.73M	7.49M	4.82M	4.02M	3.42M	3.12M

Table 3.22: HPM group 3 summary for SV(Zv) COMMIX-1AR/P code running on Cray X-MP/48 with data set P1.

type of instruction	P1r0		P1r1		P1r2		P1r3	
	inst. per CPU sec	% of all inst.	inst. per CPU sec	% of all inst.	inst. per CPU sec	% of all inst.	inst. per CPU sec	% of all inst.
jump/special	3.31M	8.77	3.34M	8.86	3.42M	8.69	3.44M	8.71
scalar	33.22M	88.02	33.21M	88.11	34.96M	89.00	35.16M	89.14
vector integer/logical	0.38M	1.01	0.43M	1.14	0.35M	0.88	0.35M	0.89
vector floating point	0.27M	0.73	0.23M	0.60	0.18M	0.45	0.15M	0.38
vector memory	0.56M	1.48	0.48M	1.29	0.39M	0.98	0.34M	0.87
type of operation	ops per CPUsec	avg. VL	ops per CPUsec	avg. VL	ops per CPUsec	avg. VL	ops per CPUsec	avg. VL
Vector integer&logical	8.82M	23.20	11.15M	25.92	5.56M	16.06	5.54M	15.82
Vector floating point	7.27M	26.43	6.44M	28.30	3.72M	21.11	3.08M	20.28
Vector memory	14.54M	26.01	12.56M	25.92	8.75M	22.70	7.50M	21.81

Table 3.23: HPM group 3 summary for SVC COMMIX-1AR/P code running on Cray X-MP/48 with data set P1.

type of instruction	P1r0		P1r1		P1r2		P1r3	
	inst. per CPU sec	% of all inst.	inst. per CPU sec	% of all inst.	inst. per CPU sec	% of all inst.	inst. per CPU sec	% of all inst.
jump/special	2.21M	10.27	2.00M	10.75	1.78M	9.17	1.77M	9.23
scalar	18.75M	87.29	16.20M	87.14	17.24M	88.69	16.98M	88.79
vector integer/logical	0.15M	0.69	0.13M	0.72	0.16M	0.81	0.15M	0.81
vector floating point	0.12M	0.58	0.08M	0.44	0.08M	0.42	0.07M	0.36
vector memory	0.25M	1.17	0.18M	0.95	0.18M	0.92	0.16M	0.81
type of operation	ops per CPUsec	avg. VL	ops per CPUsec	avg. VL	ops per CPUsec	avg. VL	ops per CPUsec	avg. VL
Vector integer&logical	2.80M	18.86	2.84M	21.26	2.35M	15.03	2.29M	14.78
Vector floating point	2.93M	23.67	1.96M	23.88	1.65M	20.26	1.32M	19.25
Vector memory	6.29M	24.99	4.27M	24.32	4.00M	22.39	3.33M	21.43

- Change real data type declarations from single to double precision. For all the compilations we used the `-r8` option but there were some explicit conversions necessary to handle explicitly typed double precision reals.
- Change machine dependent constants.
- Substitute calls to system level routines.
- Link with appropriate libraries. In particular we linked with the CSRD mathematical software library to use the functional equivalents of the LINPACK subroutines DGEFA, DGESL. From the CSRD library we used also the functional equivalent of the Cray **second** timing subroutine, which was written using the **etime** Alliant intrinsic.

Executable images were produced for each of the SV, SC, and SVC optimization options with the `-DAS -r8` options also in effect. The executable image was then run for each of the P1r0, P1r1 and P1r2 data sets.

In the case of COMMIX-1AR/P running on the Alliant FX/80 we were not able to obtain GPROF results whenever concurrency was one of the optimization options, as the code failed upon execution. We are currently investigating this problem. Instead we manually instrumented the code with calls to **etime** to accumulate the time spent in each of the most important subroutines. In a few cases we also used the system-level **time** command.

The vector-only code was executed on 1 CE.

We first show the summarized timing results for each mode of optimization. This is a yardstick for future performance improvements.

Table 3.24 presents the timing data for the SV, SC, and SVC optimization options. These data were obtained by hand instrumenting the code in its entrance and exit points with **etime**.

We notice immediately that the speedup obtained is very small, obtaining a maximum of 1.23 (cf. Table 3.25). Given our data from the Cray runs, this is hardly surprising.

An interesting observation which warrants further study is the slight increase of runtime across data sets when choosing an enhanced optimization option, namely SVC instead of SC. This is most

Table 3.24: Execution times for SV, SC and SVC COMMIX-1AR/P code running on an Alliant FX/80.

Version	P1r0	P1r1	P1r2
SV	1322.4	122.3	12331.2
SC	1128.7	106.4	10015.0
SVC	1136.7	106.6	10048.6

Table 3.25: Effect of number of CEs on execution times for SVC COMMIX-1AR/P code running on an Alliant FX/80.

	P1r0		P1r1		P1r2	
CEs	Time	Speedup	Time	Speedup	Time	Speedup
1	1385.9	1.0	130.9	1.0	12393.0	1.0
2	1230.7	1.12	114.6	1.14	11481.7	1.07
4	1173.5	1.18	110.7	1.18	10380.8	1.19
6	1142.4	1.21	107.0	1.22	10462.5	1.18
8	1136.7	1.21	106.6	1.22	10048.6	1.23

likely due to the short vector lengths for those loops which VAST transforms to vector-concurrent form. For a fixed number of CEs, and a loop processed in SVC (vector-concurrent) mode, the overhead involved in issuing and executing a vector instruction per CE will be greater than the time required to issue and execute the few required scalar instructions when operating in SC mode. One can also say that the "efficiency" of the vector instructions is very low. A witness to the difficulty is the information from HPM group 3 (cf. Table 3.23) for the Cray, which shows the average vector length for SV(Zv) mode to be only about 23 for each data set. Such a vector length would result in an average length of $\frac{23}{p}$ for the vector instructions issued when p CEs are executing. For $p = 8$, this means that the vector length is only about 3, which is below the length needed to achieve parity with scalar performance. One option we are currently investigating is the use of the `-alt` option of the Alliant Fortran compiler which produces alternate versions (scalar or vector) of `DO` loops, depending on the expected length of the vector statement to be assigned to each processor.

From Table 3.27 we notice that the SOLVEV routine consumes the maximum percentage of the runtime for P1r0 data. This is in agreement with the data for the SV baseline runs on the Cray (cf. Table 3.3) but in contrast to the SV(Zv) Cray runs presented in Table 3.15, and is due to the inability of the VAST preprocessor to do the necessary concurrent/vector transformations without the special directive corresponding to Cray's `IVDEP`. As similar remark can be made for LOWFCV.

We note that in Table 3.26 and the ensuing ones, the *Time step* entry refers to the total time taken by the subroutine TIMSTP.

Table 3.26: Execution times of various phases for COMMIX-1AR/P code running on an Alliant FX/80 for P1r0.

Phase	SV 1ce	SC 8ce	SVC 1ce	SVC 2ce	SVC 4ce	SVC 6ce	SVC 8ce
Time step	1289.57	1101.10	1352.57	1200.73	1145.35	1115.55	1108.89
Momentum constr.	296.03	271.94	291.85	281.32	290.46	284.13	283.14
Momentum solu.	440.76	331.55	456.68	385.51	344.50	332.94	330.08
Energy constr.	303.03	278.38	302.38	299.97	298.95	296.31	297.49
Energy solu.	209.51	180.28	261.52	194.75	173.75	165.31	160.88
Thermal Struct.	19.52	19.88	19.61	19.78	19.81	19.66	19.79
TS Radiation	1.71	1.67	1.69	1.70	1.74	1.68	1.73

Table 3.27: Execution times of CPU intensive routines in COMMIX-1AR/P code running on an Alliant FX/80 for P1r0.

Routine	SV 1ce	SC 8ce	SVC 1ce	SVC 2ce	SVC 4ce	SVC 6ce	SVC 8ce
SOLVEV	206.58	178.10	258.40	192.30	171.63	163.28	158.91
LOWFCV		135.14					136.76
QSTDRS	76.08	70.68	82.22	76.08	76.12	77.22	79.36
XMOMI	31.96	29.67	26.54	27.97	28.03	28.28	27.98
YMOMI	48.93	33.74	36.22	38.15	38.52	39.80	38.39
ZMOMI	49.08	48.97	70.18	60.74	65.95	61.31	62.61
ENERGI	48.94	52.04	50.20	58.29	59.04	57.17	56.54
QSTRUC	32.21	30.01	31.83	30.70	29.54	29.08	29.12

Table 3.28: Execution times of various phases for COMMIX-1AR/P code running on an Alliant FX/80 for P1r1.

Phase	SV 1ce	SC 8ce	SVC 1ce	SVC 2ce	SVC 4ce	SVC 6ce	SVC 8ce
Time step	118.26	102.93	126.18	110.30	106.53	102.91	102.49
Momentum constr.	28.73	27.03	27.50	28.40	28.44	27.36	27.61
Momentum solu.	29.15	21.05	30.58	24.12	22.11	21.11	20.66
Energy constr.	29.76	27.63	30.22	30.37	29.17	29.99	30.32
Energy solu.	26.63	23.37	33.86	23.54	23.01	20.77	20.15
Thermal Struct.	1.95	1.97	1.97	1.98	1.98	1.96	1.98
TS Radiation	0.17	0.17	0.17	0.17	0.17	0.16	0.18

Table 3.29: Execution times of CPU intensive routines in COMMIX-1AR/P code running on an Alliant FX/80 for P1r1

Routine	SV 1ce	SC 8ce	SVC 1ce	SVC 2ce	SVC 4ce	SVC 6ce	SVC 8ce
SOLVEV	26.27	23.09	33.50	23.24	22.74	20.52	19.89
LOWFCV		6.33					6.13
QSTDRS	7.29	6.70	7.86	8.61	7.35	7.55	8.12
XMOMI	3.10	3.20	2.51	2.60	2.65	2.88	2.48
YMOMI	4.59	3.57	3.34	3.57	3.63	4.01	3.39
ZMOMI	4.58	5.31	5.81	6.89	6.38	5.31	5.90
ENERGI	4.76	5.71	5.32	5.37	5.32	6.38	5.61
QSTRUC	3.20	2.99	3.19	3.05	2.94	2.90	2.89

Table 3.30: Execution times of various phases for COMMIX-1AR/P code running on an Alliant FX/80 for P1r2.

Phase	SV 1ce	SC 8ce	SVC 1ce	SVC 2ce	SVC 4ce	SVC 6ce	SVC 8ce
Time step	12297.20	9986.37	12358.54	11450.86	10351.53	10433.30	10019.23
Momentum constr.	3733.70	3110.72	3579.02	3554.11	3212.87	3299.41	3133.59
Momentum solu.	4071.72	2904.95	4233.12	3471.16	3074.20	3007.06	2967.60
Energy constr.	3888.48	3461.57	3831.89	3881.43	3578.85	3656.77	3458.13
Energy solu.	497.44	406.23	609.70	440.80	383.85	367.56	358.76
Thermal Struct.	21.59	21.79	21.79	21.98	22.02	22.00	22.02
TS Radiation	1.90	1.93	1.83	1.84	1.82	1.91	1.77

Table 3.31: Execution times of CPU intensive routines in COMMIX-1AR/P code running on an Alliant FX/80 for P1r2.

Routine	SV 1ce	SC 8ce	SVC 1ce	SVC 2ce	SVC 4ce	SVC 6ce	SVC 8ce
SOLVEV	458.38	375.29	570.36	410.00	357.43	341.73	334.15
LOWFCV		907.13					934.60
QSTDRS	999.55	886.96	959.58	982.58	952.28	960.68	906.94
XMOMI	435.81	359.01	393.95	373.60	315.14	328.59	311.66
YMOMI	587.24	399.98	519.84	503.95	432.57	442.40	428.66
ZMOMI	679.56	556.98	682.26	705.87	666.21	700.76	641.23
ENERGI	685.20	586.70	734.43	771.56	663.17	694.49	659.76
QSTRUC	421.05	386.95	419.25	395.88	382.98	379.64	377.24

Table 3.32: Routine events for COMMIX-1AR/P trace graph obtained from the Cray X-MP/48.

Event	Routine	Event	Routine	Event	Routine	Event	Routine
0	COMMIX	1	LOCF	2	CLEAR	3	TSCAN
4	MXPLNS	5	ALTER	6	AMAIN	7	GEOM3D
8	BOXES	9	FILLM	10	SHOME	11	TLEFTS
12	INITIAL	13	INITA3	14	INITA2	15	FITIT
16	SMOOTH	17	GETF	18	INHTX	19	INPUMP
20	INFORC	21	INPSTR	22	ICTEMP	23	BARIN
24	RSET2	25	DSET2	26	REDEF	27	BCTEMT
28	BCTEM0	29	BCTEMP	30	BCFLOW	31	BCPRES
32	GETMTS	33	GETIJK *	34	LODODD	35	HSTRUC
36	HEATCF *	37	TSTRUC	38	QSTRUC	39	OUTPUT
40	PSTRU1	41	RARRAY	42	PSTRUC	43	GDCONV
44	WATSTP	45	GETEKL	46	TIMSTP	47	MOLOOP
48	XMOMI	49	YMOMI	50	ZMOMI	51	FORCES *
52	PUMPQ	53	PEQN	54	CGLOOP	55	STORE
56	UPDATE	57	COMMAT	58	BOUND	59	ELIM
60	VOLCEL	61	WRPTST	62	SORTC	63	RESORT
64	CNGRIC	65	FACNCV	66	COFSRT	67	OPERXDF *
68	LOWFCV *	69	DDOTC *	70	DAXPYC *	71	DAXPXC *
72	MOMENI	73	ENLOOP	74	ESORCE	75	ENERGI
76	SOLVEV	77	QSRAD	78	WATTIM	79	RESTAR
80	PLTAPE						

3.2.3 Dynamic program execution tracing analysis

As mentioned in Section 2.3 we have started using tools developed at CSRD to capture the detailed histories of routine invocation together with machine performance statistics. In Figure 3.1, we show the routine trace graph for a COMMIX-1AR/P execution in vector mode on the Cray X-MP. Only fifty iterations were performed in this execution. Each routine has been given an event number as listed in Table 3.32.[1] The trace graph visually depicts where time is being spent in routines during the execution and the routine calling dynamics as the application proceeds. We hope to further apply this type of analysis in Phase 2 to identify performance limiting behavior.

3.3 Results from COMMIX-1C

We next list some results from the performance evaluation of COMMIX-1C on the ANL Cray X-MP/14 and the Alliant FX/8. For this particular code we also include performance results from running the code in its original environment, namely a Sun Sparc workstation. We note that although to be consistent with our previous runs we used the SV version of the code as baseline, COMMIX-1C was not written to profit from vector processing.

[1] Some of the routine events have been elided due to their high frequency; these events are identified by an "*" in the Table 3.32.

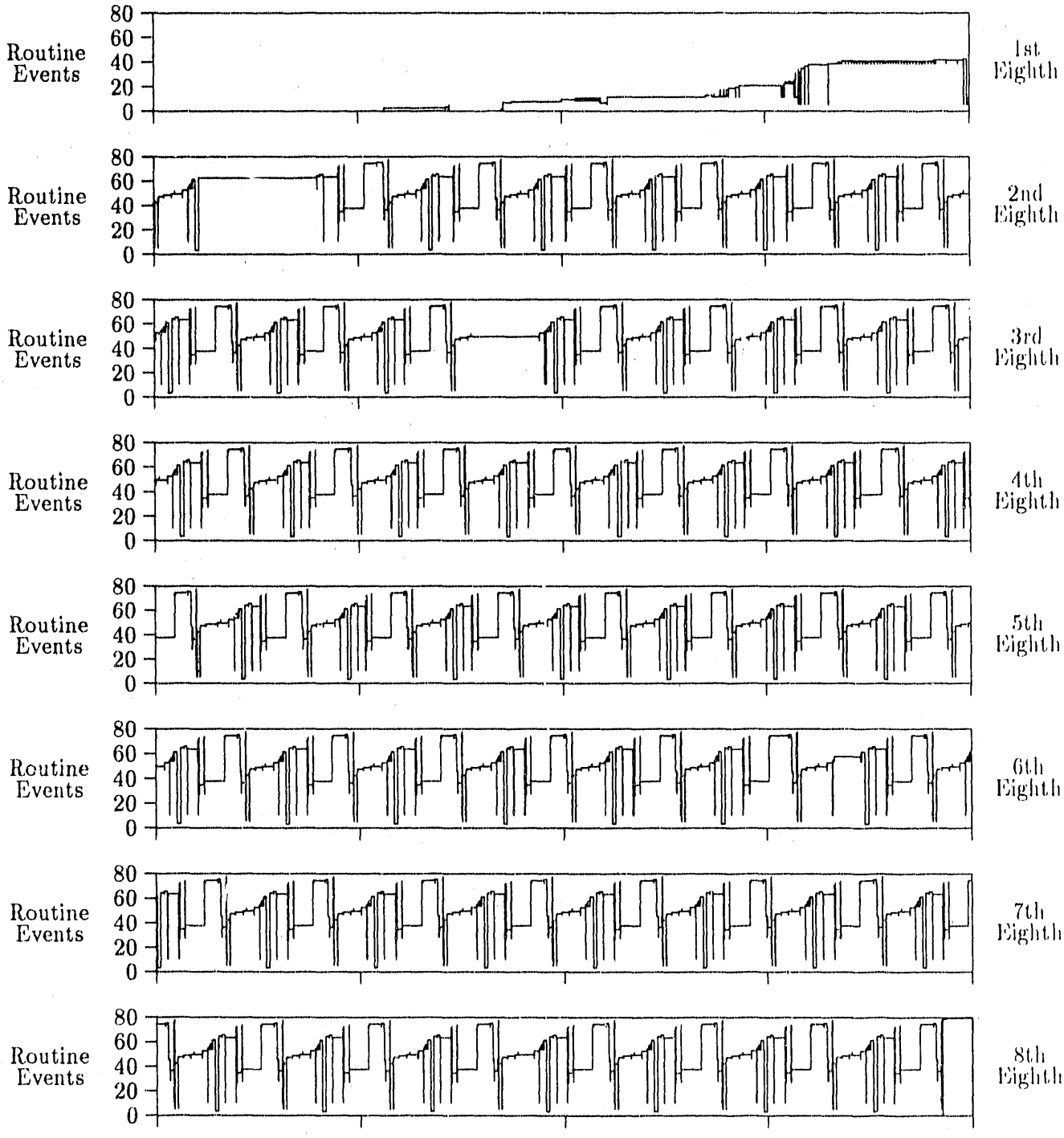

Figure 3.1: Routine graph for fifty iterations of COMMIX-1AR/P execution on the Cray X-MP/48.

Table 3.33: Timing results from COMMIX-1C runs on the Sparc, Cray X-MP/14 and Alliant FX/8.

	C1r0	C1r1	C1r2
Sparc	40.3	9,233.4	3,512.5
Cray (SV)	6.6	1,463.6	554.8
Alliant (SVC)	48.4	12,625.9	5,124.1

Table 3.34: HPM group 0 summary for baseline SV COMMIX-1C code running on Cray X-MP/14 with data set C1r2.

	C1r2
CPU seconds	554.86
Million inst/sec (MIPS)	36.28
Avg. clock periods/inst	3.24
% CP holding issue	45.67
Inst.buffer fetches/sec	0.61M
Floating adds/sec	2.30M
Floating multiplies/sec	2.93M
Floating reciprocal/sec	0.38M
I/O mem. references/sec	0.49M
CPU mem. references/sec	12.18M
Floating ops/CPU second	5.60M

To port the code from the Sparc to the Cray X-MP, we first had to change data type declarations to Cray single precision (64 bits), IEEE arithmetic traps specific to the Sun were removed, machine specific constants were changed, and dynamic memory allocation calls were converted to Cray specific. Several difficulties associated with a "Namelist" like input processor were also resolved. For the Alliant the above mentioned IEEE arithmetric traps were converted to Alliant traps. However, the principal problem was the lack of compatible dynamic memory allocation system routines (calls to the Alliant Fortran `allocate()` were not enough.) The problem was dealt with (for the present) by explicit allocation of large sections of memory, coded so as to check that the allocation is sufficiently large. In addition the use of a parallel/vector architecture led to concerns about the correctness of the computed results. This led us to perform additional experiments with various compiler options and run on a single processor. These experiments revealed that our transformations did not change the nature of the output data.

The results from Table 3.33 imply that the MFLOPS rates achieved for phase 2 of the computation were 0.88 on the Sparc, 5.6 on the Cray X-MP/14 and 0.61 on the Alliant FX/8.

We also list the HPM results from running the code in SV mode on the ANL Cray X-MP/14 and using data set C1r2. Tables 3.34-3.37 summarize the obtained results.

We see that the current version of the code performs very badly on the Alliant: Despite the automatic optimizations, the times are lower for the Alliant compared to the Sparc. The inability to automatically extract high performance for this code was also observed from the speedup values: Input deck C1r2 ran only 1.03 times faster on 8 than on a single CE of the FX/8.

Table 3.35: HPM group 1 summary for baseline SV COMMIX-1C code running on Cray X-MP/14 with data set C1r2.

	C1r2 % of all CPs
Waiting on A-registers/funct. units	13.28
Waiting on S-registers/funct. units	29.63
Waiting on V-registers	0.23
Waiting on vector functional units	0.36
Waiting on scalar memory references	0.18
Waiting on block memory references	1.37

Table 3.36: HPM group 2 summary for baseline SV COMMIX-1C code running on Cray X-MP/14 with data set C1r2.

	C1r2
Inst. buffer fetches/sec	0.61M
Scalar memory refs/sec	7.77M
% having conflicts	36.14
I/O memory refs/sec	0.64M
% having conflicts	51.72
Block memory refs/sec	4.41M
% having conflicts	72.96
Vector memory refs/sec	0.72M

Table 3.37: HPM group 3 summary for baseline SV COMMIX-1C code running on Cray X-MP/14 with data set C1r2.

	C1r2	
type of instruction	instr./CPU sec	% of all instr.
jump/special	3.30M	9.11
scalar	32.85M	90.54
vector integer/logical	0.05M	0.15
vector floating point	0.02M	0.04
vector memory	0.06M	0.16
type of operation	ops/CPU sec	avg. VL
Vector integer&logical	0.57M	10.60
Vector floating point	0.25M	16.45
Vector memory	0.72M	12.19

Looking at the HPM group 3 results in Table 3.37, we notice that the vector floating-point operations accounted for only 0.04% of all instructions. This compares with a typical 0.5% for the corresponding baseline SV runs of 1AR/P and 0.6% for the SV(Zv) runs. This low performance is not surprising: COMMIX-1C uses YSMP to solve the linear systems by direct methods which do not take advantage of vector or parallel processing capabilities. Finally, a comment similar to that made in the end of Section 3.2.2 regarding vector lengths and inefficiency of SVC processing could also be made here.

Chapter 4

WHAMS-3D description

The WHAMS-3D computer program employs explicit time integration to do nonlinear, transient analysis of frames, shells, plates and continua in three dimensions [4]. Both material nonlinearities due to elasto-plastic behavi ır and geometric nonlinearities due to large displacements can be treated. This program has been developed jointly at Northwestern University and Argonne National Laboratory and is internationally recognized as a state-of-the-art program for performing nonlinear transient analysis. WHAMS-3D has maintained its role as a leading edge program by performing extensive research in innovative methods (e.g., subcycling) for enhancing computational efficiency. Argonne has employed and developed this program to perform analysis of various reactor components and structures in a computationally efficient manner. Other organizations have employed the program to perform analysis of ice forces on Arctic structures, analysis of buried structures, military weapon analysis, aircraft engine structural analysis, analysis of electronic components, reactor safety fluid-structural analysis, impact and penetration analysis, and automobile crashworthiness simulations. The program has been extensively validated by comparisons with a wide range of experiments associated with large deformations, buckling, impact-penetration, etc.

The program employs a finite element format, so that it possesses considerable versatility in modeling complex shapes and boundary conditions. The element library consists of the following: quadrilateral and triangular plate-shell elements, a beam element, a spring element and a hexahedral continuum element. In addition, a rigid linkage is included which permits the efficient modeling of very stiff portions of a structure, such as the bottom ring of a core barrel. In a rigid linkage, the motion of a master node defines the motion of all slave nodes linked to the master node. This option is also useful for eccentrically connected elements where the midlines of the connected elements do not coincide, as for example, in stiffeners.

All of the elements in the program are three dimensional. The beam element is based on Euler Bernoulli theory, which assumes that planes normal to the midline remain planar and normal. The element has stiffness in torsion, bending about two axes, and in the axial direction.

If the material response is elasto-plastic, the cross-section of the beam is restricted to be thin-walled, and the cross-section is completely arbitrary and defined through input. Each element is assumed to be prismatic. For elastic beams, the cross-section may be defined directly through the section moduli and transverse shears may be included.

The quadrilateral plate element is based on Mindlin-Reissner theory. This is the recommended element for most simulations. It uses one-point quadrature in the surface of the shell to achieve computational efficiency. Spurious modes are suppressed through a consistent gamma-projection. A Mindlin-Reissner type triangular plate element is also available. The triangular element provides

versatility in modeling.

A three dimensional Lagrangian hexahedral element with eight nodes is included which can be used to model fluid and solid continua. The element uses only one quadrature point with a consistent control of hourglass modes so very large meshes can be handled effectively.

In all of the elements, a corotational element formulation is used. In this formulation a coordinate system is embedded in each element and all element computations are performed relative to this element coordinate system which rotates with the element. This introduces considerable simplifications into the formulation and adds substantially to the efficiency of the program.

Time integration is performed by the explicit, central difference method. Stable time steps can be automatically computed or input by the user or a driving program. Mixed time integration, a procedure which allows for different time steps in different parts of the mesh, may be employed. To provide a check on the stability of a calculation after it is completed, energy balances are computed. A lumped mass matrix is employed so that no matrix inversion is needed in the computations and core storage requirements are minimized.

Elasto-plastic material laws using Mises or Ilyushin criterion with isotropic strain hardening are included. Hardening is defined by a piecewise linear function. Elastic and hydrodynamic material laws are also available. The material laws are completely modularized, so other material laws can easily be added by the user.

An interaction algorithm for treating impact-penetration simulations with arbitrary erosion is included. This algorithm is based on interaction between slave nodes (projectile) and master elements (target) and no tracking of the sliding interfaces is needed. It employs an assembly of normals to identify the interface surface adaptively so that it can handle eroding elements in both the target and penetrator. In the interaction algorithm, momentum is exchanged between interacting nodes so that the total momentum is conserved.

The algorithm for explicit time integration primarily involves computing the internal forces of the elements which describe the simulation. Nodal accelerations, velocities and displacements are then made by central difference equations.

4.1 Data sets

Four data sets are used for the performance testing. These data sets all use quadrilateral plate elements. They, however, represent four different physical problems.

Cylindrical Panel (*cylpanel*). Since closed form solutions are not available for nonlinear transient programs, solutions obtained by finite elements are typically compared to experimental results. The cylindrical panel problem has been used as a benchmark for many nonlinear transient programs. Experimental results have been obtained for this shell by Balmer and Witner [3]. A 120 degree cylindrical panel subjected to an impulsive loading is modeled to test the elastic-plastic, large deformation capability and the ability to treat curved surfaces. The analytical model takes advantage of symmetry, so only half the panel is modeled; 1089 nodes and 1024 elements were used for the half panel. The impulsive loading is accomplished numerically by prescribing an initial velocity.

Column Buckling (*buckle*). In the Clinch River breeder reactor design, four columns were used to support the above core structures. The dynamic buckling analysis quantifies the energy absorption capability of these columns during core disruptive accidents. Symmetry conditions

Table 4.1: Execution times for WHAMS-3D on the Cray X-MP/48 (from HPM).

Version	Execution Time			
	buckle	cylpanel	frame	spcap
S	59.19	391.19	4048.66	88.02
SV	9.73	64.11	650.07	14.72
SVC	9.66	61.53	633.92	15.30

allow a half column model of 287 nodes and 240 elements. The column is loaded by prescribing an upward velocity of 500 in./sec. to the bottom nodes of the mesh with the top nodes fixed.

Spherical Cap (*spcap*). The pressure loaded spherical cap problem is a common benchmark for nonlinear finite element codes. Linear elastic and elastic-plastic materials are typically considered. A total of 332 nodes and 300 elements were used for the one-quarter model. A uniform pressure loading is applied over the cap.

Structural Frame (*frame*). The frame mesh is a common front-end automobile structural component subjected to crashworthiness testing. Hallquist, Benson, and Goudreau have documented the geometry definitions of this data set which was obtained from Suzuki Motor Company of Japan and used to benchmark the performance of other finite element formulations for shell analysis [7]. A total of 1122 nodes and 1100 elements were used for the mesh in our benchmark analyses. Impact was modeled by prescribing a uniform velocity of 800 in./sec. across the mesh with a clamped row of nodes at one end.

4.2 Results from WHAMS-3D

A detailed analysis of our timing results and porting activities can be found in the first progress report. We outline here the performance measurements we have obtained during the first phase of this project. In this initial benchmark we tried to get all the performance results based on original code. In particular, we did not attempt any hand optimizations.

As mentioned in previous sections, the original code WHAMS3D has been ported, compiled, and tested on both the Cray X-MP/48 of NCSA and the Alliant FX/80 of CSRD. On the Cray X-MP/48, we obtain three different (scalar, scalar-vector, and scalar-vector-concurrent optimized) versions of the object codes. On the Alliant FX/80, we have four different (scalar, scalar-vector, scalar-concurrent, and scalar-vector-concurrent optimized) versions. In this section, we present some of the performance results of these versions of the original code on both machines for all four data sets.

4.2.1 Cray X-MP/48

The performance results of the original code WHAMS3D on the Cray X-MP/48 are presented in Tables 4.1 through 4.5. Table 4.1 shows the execution time for all four data sets. As can be obtained from this table, the vector speedup of the SV version over the S version ranges from 5.98 for the data set *spcap* to 6.23 for *frame*. It is also observed that the SVC version does not yield good

Table 4.2: Execution times for SVC version of WHAMS-3D on the Cray X-MP/48 (from HPM).

No. of Concurrent CPUs (i)	Connect time (T_i) in each CPU			
	buckle	cylpanel	frame	spcap
1	3.15	19.66	162.35	5.99
2	0.59	9.92	131.54	1.88
3	2.45	12.30	147.89	4.34
4	3.47	19.74	211.34	3.08
Total exec. time	9.66	61.63	653.12	15.30
Total CPU time	25.54	155.37	1714.46	35.10

Table 4.3: Floating point operations on Cray X-MP/48 for the S version (from group 0 of HPM).

F.P. type	No. of F.P. operations			
	buckle	cylpanel	frame	spcap
adds	290.8M	2,021.7M	19,954.2M	430.8M
multiplies	345.6M	2,254.0M	23,588.5M	504.3M
reciprocals	36.0M	203.9M	2,473.2M	55.0M
Total	672.4M	4,479.6M	46,015.9M	990.1M

Table 4.4: MFLOPS for WHAMS-3D on the Cray X-MP/48 (from HPM).

Version	MFLOPS			
	buckle	cylpanel	frame	spcap
S	11.36	11.50	11.37	11.25
SV	71.00	71.65	72.65	69.05
SVC	71.38	74.49	76.97	66.56

Table 4.5: Performance data for WHAMS-3D on the Cray X-MP/48 using data set *cylpanel* (from FLOWTRACE).

Compiler options		S		SV		SVC	
name	calls	% time	time	% time	time	% time	time
QPLATE	25600	33.16	129.00	28.52	18.30	28.69	17.62
QMISES	12800	20.97	81.56	17.56	11.27	17.55	10.78
QRIGID	25600	9.08	35.33	7.04	4.52	7.00	4.30
QFORCE	25600	6.13	23.83	16.29	10.46	16.68	10.25
UPDATE	1600	5.48	21.33	3.69	2.37	3.69	2.27
SOLVE	1	5.33	20.73	6.36	4.08	6.29	3.87
BYIELV	63568	5.28	20.55	3.94	2.53	3.97	2.44
NEWSDV	63568	4.35	16.93	4.53	2.91	4.56	2.80
QVECTV	25600	3.53	13.75	2.24	1.44	2.29	1.41
PMODV	63568	2.83	11.01	2.58	1.66	2.58	1.59
CONSTR	1600	1.87	7.26	1.86	1.20	1.88	1.16
Total	– – –	98.01	381.28	94.61	60.74	95.18	58.49

performance although four CPUs are available. Table 4.3 shows the MFLOPS for the S version of the code on the Cray X-MP/48, as obtained from group 0 of HPM.

Table 4.2 gives the job accounting information for the SVC version. This information provides the connect time in each active CPU and the number of active CPUs (cf. the discussion Eq. 3.1). The performance of the scalar-vector version of this code reaches about 70 MFLOPS, 1/3 of the peak performance of this machine.

To better understand the behavior of this code, we also used the performance utility FLOWTRACE to gather information at the subroutine level. All the four data sets have been examined. In the following, we present only the performance data for the data set *cylpanel* since the behavior of this code on the other three data sets does not vary very much. The results are shown in Table 4.5 in which only heavily used subroutines are presented. A brief description of these most time-consuming subroutines is given below.

QPLATE Compute the internal forces for a quadrilateral plate element using one point integration by a velocity strain formulation.

QMISES Compute the stress given the strain for a plane stress biaxial elastic-plastic material.

QRIGID Compute deformed nodal coordinates for a quadrilateral plate element, calculate the values of the shape functions at integration point (0,0), and calculate the area of the element.

QFORCE Compute hourglass forces, transfer forces from element to global coordinate system, and update the global internal force vector.

UPDATE Update the nodal coordinate system.

SOLVE Integrate the equations of motion – $F = ma$.

BYIELDV Bring back stress to yield surface.

Table 4.6: Execution time for WHAMS-3D on the Alliant FX/80 (from GPROF).

Optimization	No. of CEs	Execution Time			
		buckle	cylpanel	frame	spcap
S	1	541.63	3802.19	37951.01	820.16
SV	1	231.62	1595.33	15811.41	328.20

Table 4.7: Execution time for the SC version of WHAMS-3D on the Alliant FX/80 (from GPROF).

No. of CEs	Execution Time			
	buckle	cylpanel	frame	spcap
1	568.63	3997.18	40759.16	815.39
2	339.89	2368.70	23685.83	478.80
4	215.77	1508.11	14643.84	302.38
8	155.82	1060.39	10346.51	228.20

NEWSDV Compute new stress by elastic-plastic material law.

QVECTV Compute the quadrilateral plate element coordinate system.

PMODV Look up the plastic modulus and yield stress.

CONSTR Enforce boundary conditions in the local coordinate system.

4.2.2 Alliant FX/80

We next present results from use of the Alliant FX/80. It should be noted that we did not use the compiler option -r8 to compile the source code. In other words, all versions on the Alliant are in single precision.

Table 4.6 shows the execution time using only one CE for the S and SV options. The vector speedup of the SV version over the S version ranges from 2.34 to 2.50, which is reasonably good. It should be noted that the timings for the S and SV versions presented were obtained without

Table 4.8: Execution times for the SVC version of WHAMS-3D on the Alliant FX/80 (from GPROF).

No. of CEs	Execution Time			
	buckle	cylpanel	frame	spcap
1	289.50	1896.92	19916.66	407.77
2	257.04	1763.51	17301.31	357.67
4	196.99	1347.01	12812.99	289.34
8	166.45	1094.76	11054.41	236.21

Table 4.9: MFLOPS for WHAMS-3D on the Alliant FX/80.

Optimization	No. of CEs	MFLOPS			
		buckle	cylpanel	frame	spcap
S	1	1.24	1.21	1.21	1.24
SV	1	2.90	2.88	2.91	3.10
SC	8	4.32	4.33	4.45	4.45
SVC	8	4.04	4.19	4.16	4.30

Table 4.10: Performance data for WHAMS-3D on the Alliant FX/80 using data set *cylpanel* (from GPROF).

Version		S		SV		SVC	
No. of CE's used		1		1		8	
name	calls	%time	time	%time	time	%time	time
QPLATE	25600	38.2	1452.83	25.2	401.33	15.7	171.75
QMISES	12800	14.3	544.90	19.8	316.36	26.2	287.13
QRIGID	25600	10.6	404.41	7.5	119.16	4.3	46.16
QFORCE	25600	6.6	251.00	9.1	145.91	10.2	112.18
UPDATE	1600	4.6	173.54	4.2	67.21	6.0	65.60
SOLVE	1	8.0	304.90	14.6	232.82	20.5	224.67
BYIELV	63568	3.2	123.40	2.7	43.54	2.2	24.21
NEWSDV	63568	5.2	197.44	4.7	75.31	2.5	27.67
QVECTV	25600	2.1	80.21	1.8	29.09	1.3	14.10
PMODV	63568	2.1	79.86	1.7	27.40	1.7	18.40
CONSTR	1600	2.6	98.80	3.5	55.52	3.3	35.59
Total	– – –	97.5	3711.29	94.8	1513.65	93.9	1027.91

explicitly specifying to use only one CE. The effect of the explicit specification of using one CE on performance is currently under investigation. Tables 4.7 and 4.8 show the execution time using 1, 2, 4, and 8 CEs for the SC and SVC versions, respectively. As can be seen from these two tables, the parallelism obtained through the use of compiler options results in speedups across processors for the SVC version that are rather poor.

Table 4.9 compares the performances in terms of MFLOPS for different versions of the object code, where we have used the floating point operation counts of the S version obtained from the Cray X-MP/48 because the Alliant does not have utility to gather this information.

Table 4.10 shows the performance data at the subroutine level using the Alliant FX/80 for data set *cylpanel* for the S, SV, and SVC versions of the object code. Note that the time is reduced by a factor of 2.45 in going from the S to the SV version on one processor. However, going to SVC decreases the time only by a factor of 1.47, giving a multiprocessor efficiency of only 18%. It should also be observed from 4.7 and 4.8 that the SVC version did not yield better performance than the SC version when all eight CE's were used. This is due to the fact that the vector length is not long

Table 4.11: Characteristics of the data sets used in WHAMS-3D.

Data Set	Problem Size	
	No. of nodes	No. of time steps
buckle	287	800
cylpanel	1089	1600
frame	1122	16000
spcap	332	1000
stcn	407	1136

Table 4.12: Execution times for WHAMS-3D on the ANL Alliant FX/8.

Data Set	Scalar Mode (`fortran -Og`)		Vector Mode (`fortran -Ogv`)		Vector-Conc. Mode (`fortran -Ogvc`)	
	user	system	user	system	user	system
buckle	691.1	1.1	336.4	1.1	181.1	1.1
cylpanel	4419.2	1.1	2286.4	1.7	1212.1	1.5
frame	44654.6	1.3	23030.0	1.6	12343.5	1.5
spcap	940.8	1.0	492.0	1.1	277.3	1.0
stcn	604.4	0.9	324.0	1.1	173.9	1.1

enough to take advantage of both vectorization and parallelization, as explained for the COMMIX code.

4.2.3 Additional Results

In addition to the Cray X-MP/48 of NCSA and the Alliant FX/80 of CSRD, we have also run the baseline code on the Argonne ARCF Alliant FX/8, Cray X-MP/14, and Sparcstation. Data sets employed in the test on these three machines include the four data sets mentioned previously and an additional one: a steel containment *(stcn)* subjected to a pressure loading. The problem sizes of these data sets are listed in Table 4.11. The performance data are presented in Tables 4.12 through 4.14. Table 4.12 shows the timings on the Alliant FX/8 using scalar optimization, vectorization, and parallelization. Table 4.13 presents the execution time for the Sparcstation and the Cray X-MP/14. A summary of the findings of this assessment on the Cray is provided in Table 4.14.

As indicated by the very high ratio of vector floating point operations to total floating point operations in Table 4.14, it is clear that this code has been vectorized to a large degree. The average vector lengths are close to the optimal vector length (64). A further indicator of good vectorization is the relatively low hold-issue condition percentage for the scalar registers and functional units as opposed to the vector registers and functional units. For maximum performance, the highest percentage of hold-issues should be on the vector registers and vector functional units as these are the fastest. The columns for hold issue conditions indicate the number of clock periods during which the issuance of an instruction was (held) delayed. Note that more than one hold issue condition

Table 4.13: Execution times for WHAMS-3D on the Sparc, Cray X-MP/14, and Alliant FX/8.

Machine	Execution Time				
	buckle	cylpanel	frame	spcap	stcn
scalar Sparcstation	896.9	6173.1	61647.9	1322.1	857.4
Cray X-MP/14 (vector)	10.3	67.1	689.7	8.4	11.5

Table 4.14: HPM performance data summary for WHAMS-3D running on the Cray X-MP/14.

Data Set	Floating Point Operations		Hold Issue Conditions (%)					MFLOPS	
	% Vectorized	Average VL	Scalar A-rg.	Scalar S-rg.	Vector rg.	Vector f.u.	Block	Total	Vector
buckle	99	58.78	7.1	8.3	27.3	25.4	27.5	67.2	66.7
cylpanel	99	63.19	6.0	9.0	27.1	25.7	31.8	68.4	67.7
frame	99	60.63	6.2	8.9	28.2	26.1	31.6	68.5	67.7
spcap	96	58.76	6.9	18.1	24.7	15.8	25.3	47.3	45.4
stcn	100	59.75	7.5	11.6	24.4	21.6	27.7	56.0	55.0

can occur during a given clock period so that the individual percentages cannot be added to get the overall percentage. Removing hold issue conditions promises to offer code speedups however the compiler attempts to catch bottlenecks and even the most optimized codes may have large percentages for hold issues. Only after further experimentation can it be determined whether the number of hold issues conditions can be reduced.

4.2.4 Dynamic program execution tracing analysis

In Figure 4.1, we show the routine trace graph for the first 2.954625 seconds of the WHAMS-3D execution in vector mode on the Cray X-MP; eight consecutive sub-sections of this interval are shown. Each routine has been given an event number as listed in Table 4.15.[1] The trace graph visually depicts where time is being spent in routines during the execution and the routine calling dynamics as the application proceeds. We hope to further apply this type of analysis in Phase 2 to identify performance limiting behavior.

[1]Some of the routine events have been elided due to their high frequency; these events are identified by an "*" in the Table 4.15.

Event	Routine	Event	Routine	Event	Routine	Event	Routine
0	MAIN	1	DRIVE	2	READKO	3	COREE
4	READMA	5	READNE	6	DECOD	7	CROSS
8	BLOCKS	9	ASSBLE	10	QASME	11	QVECTR
12	QDELT	13	READLD	14	READOU	15	SOLVE
16	OUTPUT	17	LOADPR *	18	FRCIN	19	QFRCIN *
20	QNODE *	21	QVECTV *	22	QRIGID *	23	QPLATE *
24	QMISES *	25	QFORCE *	26	PMODV *	27	BYIELV *
28	NEWSDV *	29	CONSTR	30	UPDATE	31	ETIME2
32	BTIME	33	ROUTPT	34	PLOTER		

Table 4.15: Routine events for WHAMS-3D trace graph

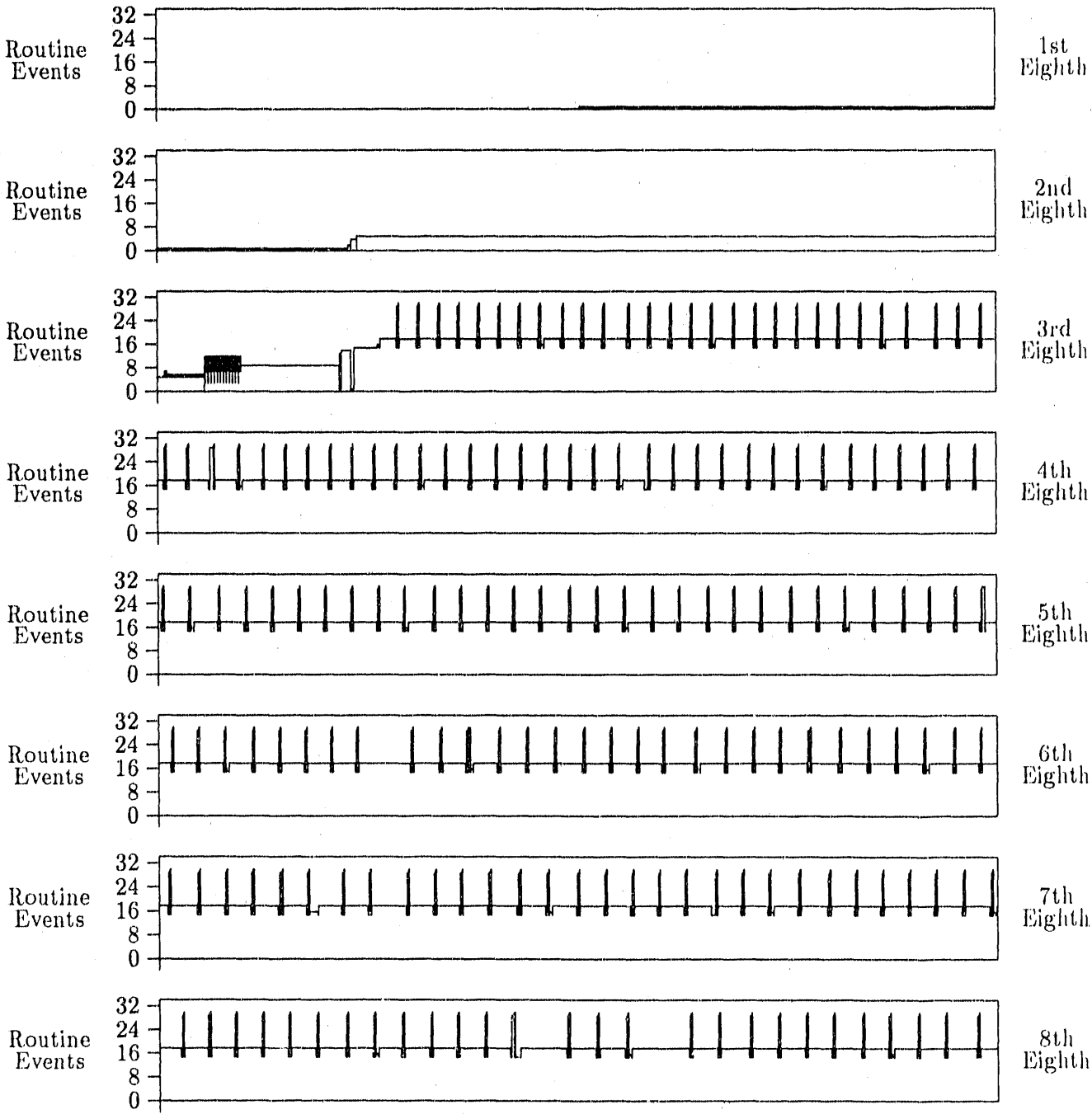

Figure 4.1: Routine graph for 0.0 to 2.954625 seconds of the WHAMS-3D execution

Chapter 5

Conclusion

We have completed the work prescribed for Phase 1 of our project. We have performed a large number of experiments without any hand tuning of the COMMIX-1AR/P, COMMIX-1C and WHAMS-3D codes. Our purpose was to have yardsticks by means of which future improvements to the code could be measured. Nevertheless, we were also able to obtain a lot of data for different compilation options, as part of our Phase 2 efforts.

Although, a full appreciation of the results requires time, even at this stage, several interesting observations can be made, which were not apparent before our experiments had taken place. These observations can be found in the commentary for the tables in the previous Sections.

We found that the performance of the COMMIX codes on the Cray and Alliant is much below the acceptable range, meaning that the codes have to be modified to take advantage of the parallel and vector processing aspects of these architectures. We noticed that even for COMMIX-1AR/P, whose parts dealing with the linear system solvers had been vectorized, the performance is very low and will remain such, until the parts of the code dealing with the matrix assembly are also modified. In that spirit we started testing the standard automatic vectorization/parallelization tools available on the target machines. Our experiments indicate the following: Without any help from the user, the tools only provide slight performance gains. Some help in the form of inline directives to the compiler can introduce very significant improvements in the performance of some subroutines. These experiences point to the next phase of our work, namely examining the individual subroutines to provide help to the automatic restructuring tools, and using more advanced data dependence tests to recover computations which can be processed in parallel.

For WHAMS-3D we have seen that the performance of the Cray under vector processing was quite reasonable. Nevertheless, the performance under the multiprocessing options was not satisfactory. In the next phase of this project we intend to investigate further the automatic transformation tools coupled with user directives to produce better optimized code.

Bibliography

[1] Analytical Thermal and Hydraulic Research Program, Components Technology Division, Argonne National Laboratory, *Commix-1B: A three-dimensional transient single-phase computer program for thermal hydraulic analysis of single and multicomponent systems. Volume I: Equations and Numerics*, Sep. 1985.

[2] ——, *Commix-1B: A three-dimensional transient single-phase computer program for thermal hydraulic analysis of single and multicomponent systems. Volume II: User's Manual*, Sep. 1985.

[3] H. A. Balmer and E. A. Witmer, *Theoretical-experimental correlation of large dynamic and permanent deformation of impulsively loaded simple structures*, Tech. Rep. FDP-TDR-64-108, Wright Patterson AFB, Ohio, 1964.

[4] T. Belytschko and C. S. Tsay, *WHAMSE: A program for three-dimensional nonlinear structural dynamics*, Tech. Rep. NP-2250, Dept. Civil Engin., Northwestern University, Evanston, Illinois, Feb. 1982. Research Project 1065-3.

[5] R. N. Blomquist, P. Garner, and E. M. Gelbard, *Code abstract for COMMIX-1AR/P*, July 1989.

[6] Cray Research, Inc., *UNICOS Performance Utilities Reference Manual*, May 1989.

[7] J. O. Hallquist, D. J. Benson, and G. L. Goudreau, *Implementation of a modified Hughes-Liu shell into a fully vectorized explicit finite element code*, in Finite Elements For Nonlinear Problems, P. G. Bergan, K. J. Bathe, and W. Wunderlich, eds., Springer-Verlag, Berlin, 1986, pp. 465–479.

[8] R. Hockney, *$(r_\infty, n_{1/2}, s_{1/2})$ measurements on the 2-CPU Cray X-MP*, Parallel Computing, 2 (1985), pp. 1–14.

[9] A. Malony, J. Larson, and D. Reed, *Tracing application program execution on the Cray X-MP and Cray 2*, Tech. Rep. 985, Center for Supercomputing Research and Development, Nov. 1990.

Appendix A

Appendix: Milestones for FY 1991

Copies of this Appendix can be obtained from the authors.

END

DATE FILMED

11 / 30 / 90