Porting FCRASH to the Cray X1 Architecture

Alexander Akkerman, *Ford Motor Company*
Dave Strenski, *Cray Inc.*

ABSTRACT: *FCRASH is an explicit, nonlinear dynamics, finite element code for analyzing transient dynamic response of three-dimensional solids and structures. The code is used as a safety research tool to simulate vehicle impact at both full vehicle and component levels. This paper describes our experience porting FCRASH to the Cray X1 architecture and presents performance results relative to the Cray T90 platform.*

1. Introduction

FCRASH is an explicit finite element analysis (FEA) code written in Fortran and C programming languages. The code consists of about 1,400 subroutines and functions. FCRASH is one of the many FEA solvers used at Ford Motor Company that aid in the design of vehicles and components for crash-worthiness. The code is used as a research tool and is currently running on many different hardware platforms ranging from desktop workstations to a Cray T90 supercomputer in Dearborn, Michigan.

FCRASH was chosen to evaluate the performance of the latest Cray supercomputer, the Cray X1, to gage the performance of other FEA tools used at Ford when they become available on the Cray X1 architecture. The high vector content of FCRASH and shared memory programming model make this code a good tool to provide insight into the potential capability of the Cray X1 system. The scope of this evaluation will be limited to single processor performance and parallelism within a single node as FCRASH is available in shared memory parallel environment only.

The strong motivation for evaluating the Cray X1 is the result of lack of advancement in high-end computing since the mid 1990's. The demand for the highest possible performance continues to increase as models continue to grow in size and complexity along with new problems that require longer simulations. Model sizes grew from 50,000 to 100,000 elements in the early 1990's to 500,000 and up to a million elements today. As commercial FEA codes migrated to the distributed memory programming model, turnaround times remained manageable on microprocessor based systems. However, lower time steps resulting from smaller elements in the models and increased simulation times for new simulations can not be adequately addressed by simply adding more processors to jobs. Cray X1 architecture is in a better position to address these increased requirements as it promises excellent single-processor performance, as well as scaling to a large number of processors.

Before describing the work to optimize FCRASH to the Cray X1 architecture, it is useful to review the fundamentals of an explicit time integration based FEA program. The purpose of this review is to analyze the structure of the program and focus on optimization.

Central difference formulas for velocity and acceleration:

$$\dot{u}^{n+1/2} = \frac{u^{n+1} - u^n}{\Delta t}$$

$$\ddot{u}^n = \frac{\dot{u}^{n+1/2} - \dot{u}^{n-1/2}}{\Delta t}$$

Basic equation of motion:

$$M\ddot{u} = f = f_{ext} - f_{int}$$

Velocities and positions can be updated as follows:

$$\dot{u}^{n+1/2} = \dot{u}^{n-1/2} + \Delta t^n M^{-1}\left(f_{ext}^n - f_{int}^n\right)$$

$$u^{n+1} = u^n + \Delta t^{n+1/2}\dot{u}^{n+1/2}$$

Incremental time step at *n* is calculated as:

$$\Delta t^n = \left(\Delta t^{n-1/2} + \Delta t^{n+1/2}\right)/\,2$$

Internal element forces are defined as:

$$f_{int} = \int_{V_c} B^{\mathrm{T}} \sigma dV$$

Where: $u =$ nodal displacement

$f_{int}, f_{ext} =$ internal and external forces

$\sigma =$ element stress

$B =$ shear displacement matrix

$M =$ mass matrix

The main integration loop for a typical explicit simulation program is summarized in the flow chart in Figure 1.

It has been well known for some time that FEA codes are rich in potential parallel constructs[1]. Parallelism within a simulation code exists at several levels, from the job or program level to the instruction level[2]. For a single simulation on a parallel vector processor, parallelization across the task or procedure (i.e. MPI, OpenMP) and instruction level parallelization (i.e. streaming, vectorization) provide the biggest opportunity to performance improvement.

Calculation of the element forces provides the greatest opportunity for streaming and parallelization. In a typical impact simulation, element calculations represent 60 to 80% of computational effort. Element calculations are entirely independent and thus inherently parallel. Element force calculations are performed on elements of the same type and material and can easily be vectorized.

The I/O needs of an explicit FEA solution are minimal. The entire analysis is typically performed in memory, I/O is only required for storing the state of simulations at regular intervals for post-processing purposes. I/O typically represents 1 to 3% of the total simulation time.

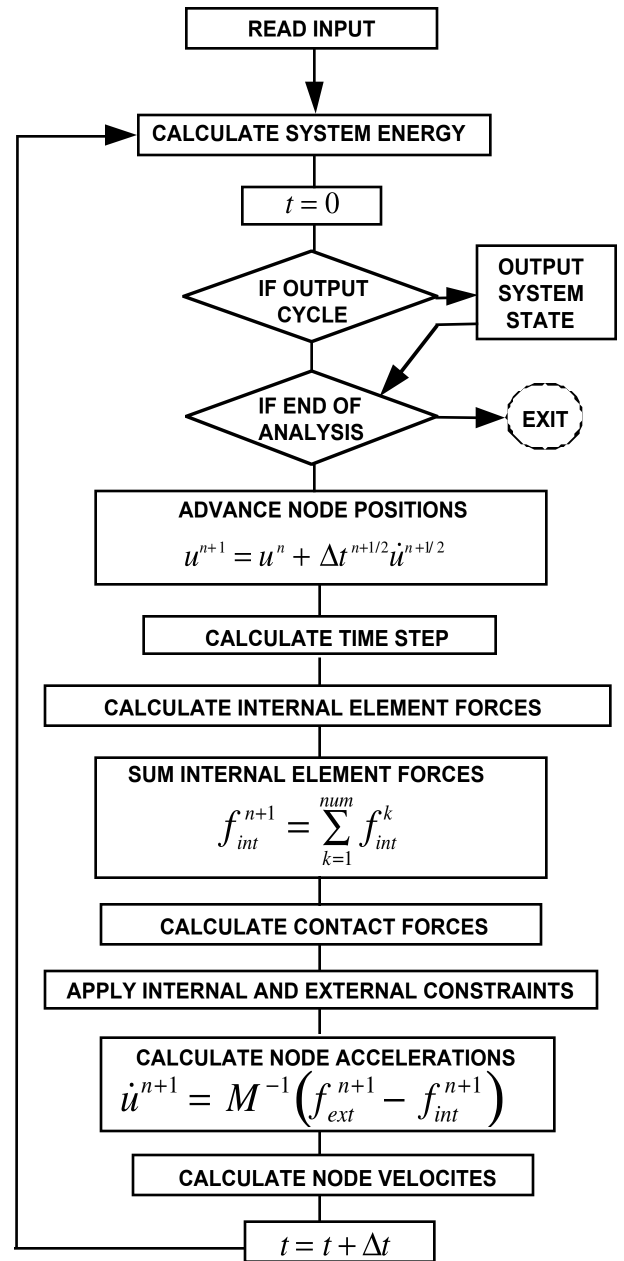


Figure 1. Explicit Formulation Flowchart

2. Cray X1 overview

Before describing the porting of FCRASH to the Cray X1, it is useful to explain how the architecture differs from the Cray T90. Comparison between the two systems is shown in Table 1.

The biggest difference in processor architecture is the multi-streaming capability of the Cray X1. In a way, one CrayX1 processor could be described as four Cray T90 cpu's tightly coupled together.

The Multi-Streaming Processor (MSP) consists of four Single-Streaming Processors (SSP), each having two vector pipes and capable of completing four floating point operations per clock. The compiler treats these four SSPs as a single unit and automatically schedules work for all four. In a simple example of double nested loops, the Cray T90 compiler vectorizes the inner loop. The next iteration of the outer loop will not start until the vector pipes are free.

	Cray T90	Cray X1
Maximum Processors	32	4096
Shared Memory	Yes	upto 16 SSP
Distributed Memory	No	Yes
Max Memory Size	256GB	64TB
Address Space	Real Memory	Virtual Memory
Vector Length	128	64
Scalar Registers	32	128
Vector Registers	8	128
Vector Mask Regs	1	64
Clock Rate (Mhz)	450	400/800
Instruction Cache	None	16KB
Scalar/Vector Cache	1KB/none	Shared 2MB
Vector Pipes	2	8
Max FLOPS per clk	4	16
Peak GFLOPS	1.8	12.8
BMM/popcnt/leadz	Yes	Yes
Vector compress/iota	No	Yes
64-bit arithmetic	Yes	Yes
32-bit arithmetic	No	Yes
Multi-streaming	No	Yes

Table 1. Comparison between Cray T90 and Cray X1.

On the other hand, the Cray X1 MSP can stream on the outer loop and vectorize the inner loop, thus completing four times as much work per clock. The streaming also works on simple loops by dividing the loop trip count into four chunks and working on them at the same time. A similar effect could be accomplished on the Cray T90 by inserting auto tasking directives before the loop (i.e.: CMIC DO ALL), however, these directives introduce a significant overhead of starting a parallel loop as compared to streaming.

The memory architecture on the Cray X1 is another major consideration, especially when porting a code designed for shared memory systems. FCRASH on the Cray T90 is able to utilize the entire system (up to 32 cpus), but limited to four MSP on the Cray X1. Scaling beyond four MSPs could be accomplished by utilizing the distributed memory programming model, however, outside the scope of this evaluation.

3. Porting Issues

As we started this project, two issues have emerged that slowed down our progress: the availability of Cray X1 resources and the rapidly changing compiler environment. The Cray X1 system utilized for this work, located in Chippewa Falls, Wisconsin, had limited availability, often busy or restricted to dedicated users. Compilers and various libraries were improving rapidly. However, in the course of our work, we encountered numerous problems dealing with issues that seemed to come and go as new versions were introduced. As we utilized 3 to 4 models in this project at any point in time, we had difficulties with some models but not others, and with executables compiled with varying optimization flags. With changes in the environment, it was difficult to keep track of the current state of the code and multiple tests of identical jobs had to be repeated.

On the positive side, we filed about a dozen Software Problem Reports, and contributed to the improvement of the development environment. Availability of the trigger environment and cross compilers made this project possible, allowing us to work off line while the Cray X1 was not available.4. Arabidopsis Data File.

4. Test Model

The model we chose for tracking our performance optimization is shown in Figure 2. This is a component level model, consisting of 88,000 shell elements, about 75% of them four-node quadrilaterals and the remaining 25% three-node triangular elements.

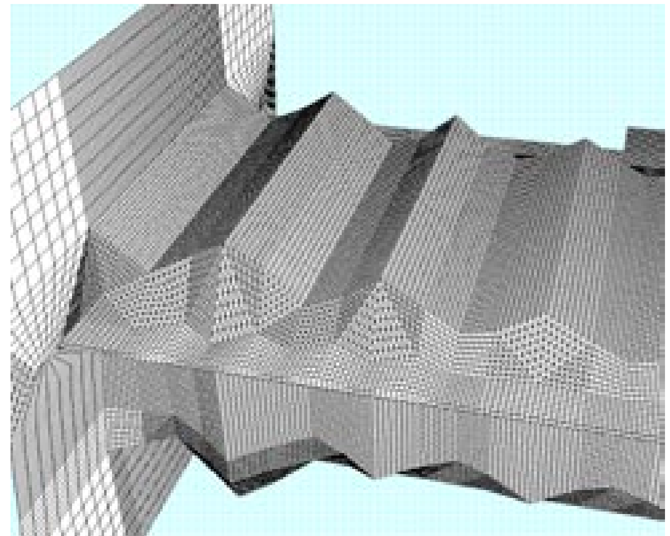


Figure 2. Close-up of the test model.

We ran the simulation for 5ms at a time step of 0.15 microseconds, resulting in 33,334 cycles. Figure 3 shows a deformed shape of this model at the end of the 5ms simulation.

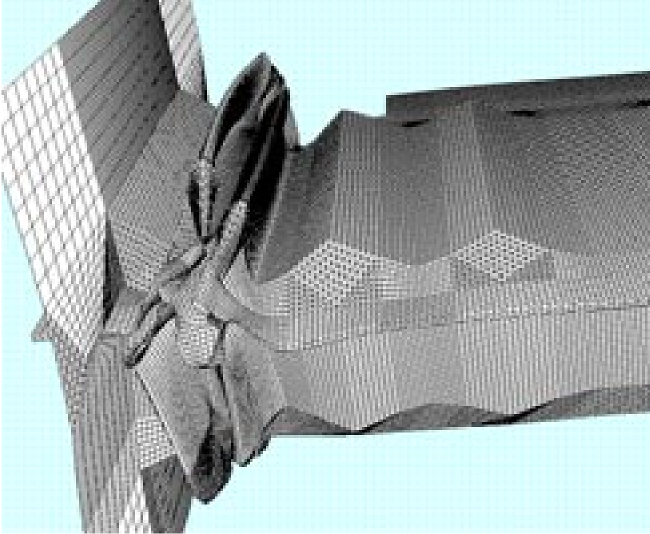


Figure 3. Close-up of the deformed test model.

5. Results

Our first successful run produced about 750 MFLOPS, a good first step for the early compilers relative to the 510 MFLOPS we achieved with the same model and mature compilers on the Cray T90.

COMPILER FLAGS

As compilers improved, we settled on the following compiler flags:

-V -rm -dp -s default64

-O inlinefrom = inlinelib.F

-O inline3, scalar3, vector3, stream3, task0

The -dp -s default64 options were chosen to provide compatibility with the Cray T90 version. This combination, along with FCRASH being written in double precision, compiles the code using 64 bit floating point numbers and 64 bit integers. In the future we intend to switch to 64 bit floating point and 32 bit integer implementation with the hope of improving performance from reduced memory traffic and higher cache utilization.

The -V -rm are standard Cray compiler options for printing out the compiler version and generating a listing file that contains loop marks that help in the understanding of the compilation process.

We are using six arguments for the -O option which controls the optimization level. It is convenient to specify each level of the compiler options. Currently we are using two options for inlining. The option "inlinefrom" pulls the inline code from a file that contains small, often used routines as determined from our development effort on the Cray T90, and inline3 specifies the level of automatic inlining. We are also using the highest level of optimization for the scalar, vector and streaming units of the compiler. Since we are focused on single MSP performance, task0 turns off all shared memory parallel directives, present in our code.

Applying these compiler flags improved our performance to 901 MFLOPS. This performance was achieved with the vector blocking factor of 256 and page size of 16M. The effects of the vector blocking factor and page size on performance are described in the following sections.

VECTOR BLOCKING FACTOR

The next step was to optimize the blocking factor that is typically used as an inner loop trip count in the element force calculations and elsewhere in the code. Higher blocking factors generally improve single processor performance, although potentially at the expense of lower parallel performance. Parallelism is achieved by distributing blocks of elements to different processors and smaller blocking factors result in more blocks and thus better load balancing. We used 256 in the Cray T90 version. This number seemed too low on the Cray X1, given its 8 vector pipes. As we varied the blocking factor from 128 to 1024, our performance changed, as described in Figure 4a. In the figure, the horizontal axis is the blocking factor and the vertical axis is the performance in MFLOPS for our test model run on a single MSP.

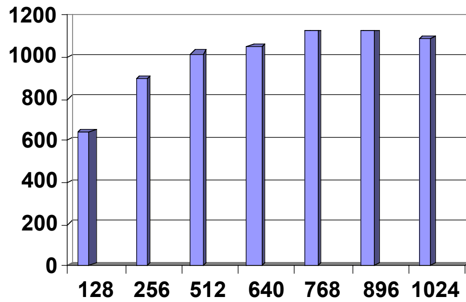
Even though the best performance was accomplished with the blocking factor of 768, we chose to keep it at 512, to provide a better opportunity for parallel performance, the ultimate goal of this project.

PAGE SIZE

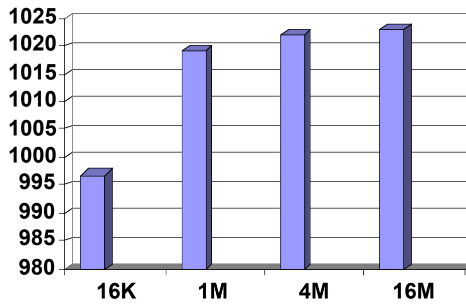
The Cray X1 system allocates memory in units of pages. Pages that are allocated to a user program for the executable itself are referred to as text pages. All other pages allocated to user tasks are called “other” pages. The size of a memory page is not fixed and can be set separately by the user for text and data pages. The size of a page can range from 16KB to 4GB[5].

The hardware mechanism that translates virtual addresses to physical has limited space to retain page-by-page translation tables. This suggests that the user should specify the appropriate size of both text and data pages as part of the aprun command.

The option used for the command is –p <text_size>:<user_size>. In Figure 4b we show the program performance, in MFLOPS, as a function of the page size. For simplicity we chose the same page size for both text and user pages.



4a) Performance for different vector blocking factors (page size = 16M)



4b) Performance for different page sizes (vector blocking factor = 512)

Figure 4. Changes in performance by modifying vector blocking factors and page size argument.

In Figure 5 we show the actual number of memory references made per translation table miss. Higher number of memory reference between misses translates into higher performance.

As can be seen from figures 4b and 5, larger page sizes result in better code performances due to reduced calculations for the page translation tables. They also show the diminishing returns for larger page sizes, big enough to hold most of the addresses.

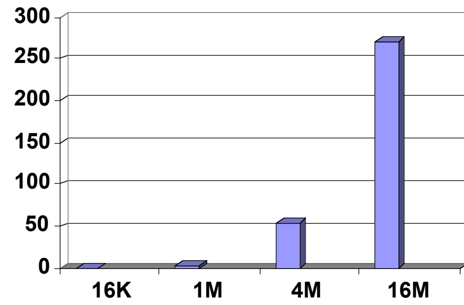


Figure 5. Number of memory references (million) per translation table miss for different page sizes.

PROFILE TOOL PAT

The performance tool used on the Cray X1 is called PAT (Performance Analysis Tool). The tool works like many traditional profiling tools and monitors the code to generate a profile of how much time is spent in each part of the code. A very useful feature of the PAT program is its ability to provide streaming information. The profile not only shows how much time is spent in each routine, but also the amount of time used in each steam. For example, profiles pointed out several problems in the early compilers as quite a few subroutines were using only one stream. By analyzing those parts of the code, and adding streaming directives (i.e. !CSD$ PREFERSTREAM) the problems were corrected and performance improved.

MSP vs. SPP

As mentioned above, while profiling the performance of our code, we noticed several routines showing a great deal of imbalance in utilization of four SSPs within an MSP processor. In some cases we were able to improve the balance by inserting directives in front of the loops in the code, however, we feel that this work should be done by compilers and hope that this issue diminishes as compilers mature.

Another approach is to utilize SSP as independent processor units. By adding –Ossp as the compiler option, the MSP is effectively split into four separate processors. To evaluate this alternative, we decided to measure streaming performance of MSP by running the code in MSP and then again in SSP modes, comparing the difference. We measured 1123 MFLOPS in MSP mode and 685 in SSP, a ratio of 1.6. This translates into effective use of 1.6 out of a possible four SSP streams. Ideally, MSP performance should equal four times the SSP.

The effective speedup from steaming of 1.6 is lower than we had hoped; however, in addition to multi-streaming, Cray X1 architecture provides an alternative to utilize the four SSPs by moving parallelization from the streaming level to the tasking level. Even though the parallelizaton at the tasking level is burdened with additional overhead, a parallel scalable code has the potential of higher overall performance by utilizing this option. We have not yet performed this test on FCRASH. However, given the current state of the compilers and FCRASH scaling performance on other platforms, we may achieve better performance running in parallel on four SSPs relative to a single processor MSP.

PARALLELIZATION

The Cray X1's implementation of OpenMP has not yet been released as of the time of this report. We had access to an early beta version; however, we chose to focus on a single processor performance before considering parallelism. Considering Cray's move toward industry standard by adopting OpenMP for shared memory parallelism, we don't anticipate major issues in building a parallel executable.

6. Conclusions and next steps

This paper is a report on a work in progress. Our effort is far from over. As access to Cray X1 hardware becomes easier and development environment (mainly compilers and libraries) stabilize and mature, we expect to continue to make improvements in our code's performance. Current performance of little over two times the Cray T90 is a good start but well below our goal of matching Cray T90's performance as measured by the percent of peak. Our current 8% of MSP peak certainly leaves much room for improvement. However, 21% of SSP peak is very encouraging.

Finally, we have not had an opportunity to evaluate Cray X1's parallel performance. As single processor (SSP and MSP) performance improves, parallelism within a four-MSP (or 16 SSP) node will be the next topic of our attention.

7. References

[1] C. H. Farhat and L. Crivelli, A General approach to Nonlinear FE Computations on Shared Memory Multi-processors, Rep. No. CU-CSSC-87-09, University of Colorado, Boulder, CO, 1987.

[2] K. Hwang, F.A. Briggs, Computer Architecture and Parallel Processing. McGraw-Hill, 1984.

[3] Cray Research Inc., UNICOS Performance Utilities Reference Manual, SR-2040 7.0, May 1992.

[4] C. H. Farhat, E. Wilson and G. Powell, Solutions of Finite Element Systems on Concurrent Processing Computers, Engineering Computing, Vol 2, pp-157-165, 1987.

[5] Cray Inc., Cray X1 Application Programming and Optimization Student Guide, TR-X1PO-B, 2003.

[6] J. G. Malone, Parallel Nonlinear Dynamic Finite Element Analysis of Three-Dimensional Shell Structures, Computers & Structures, Vol. 35, No. 5, pp. 523-539, 1990.

8. About the authors

Alexander Akkerman is a senior technical specialist in the Numerically Intensive Computing (NIC) department of Ford Motor Company. He can be reached at 313-337-1634 or aakkerma@ford.com. Dave Strenski is an application analyst for Cray Inc., located onsite at Ford Motor Company. He can be reached at 313-317-4438 or stren@cray.com.