When Is a Cray-2 No Better Than a Workstation?

by

Clark D. Thomborson (a.k.a Thompson)†
James L. Fenno†

September 15, 1989

Technical Report 89–10

When Is a Cray-2 No Better Than a Workstation?

by

Clark D. Thomborson (a.k.a Thompson)†
James L. Fenno†

September 15, 1989

Technical Report 89–10

Department of Computer Science
University of Minnesota
Duluth, Minnesota 55812
U.S.A.

†Supported in part by the National Science Foundation, through its Design, Tools and Test Program under grant number MIP 8706139, and by the Minnesota Supercomputer Institute.

Abstract

We describe a framework for deciding whether an application will benefit from execution on a supercomputer. Our framework identifies, and in most cases gives quantitative measures for, five important parameters: runtime, difficulty of optimization, frequency of use, floating point precision, and working-set size. Using our framework, we find that our time-intensive FORTRAN program for wire-routing in gate arrays is better suited to a workstation than to the Cray-2 vector supercomputer.

When working with a program that is at, or reaching, the acceptable limits of runtime on a desktop, the following question arises naturally: Will the performance of the application increase when run on a supercomputer? We pose five questions as a conceptual framework for determining whether or not it is worth porting code to a vector supercomputer:

1. What is an acceptable runtime for solving my problem?

2. Can my code be easily vectorized?

3. Will my program be used enough to warrant hand-optimization?

4. Is precision a factor in my computation?

5. How large is my program's working-set?

This article describes our experience in answering these questions for a wire routing program based upon linear programming [7]. Our program generates a fairly large (832 constraints) linear program for a small (12x15) gate array. The cpu-intensive kernel of our code is Roy E. Marsten's XMP package for solving linear programs. His library of FORTRAN routines is running on many different computers, ranging from PCs to supercomputers [6, page 3].

Since a Sun-3 desktop takes 100 seconds to solve our 832-constraint problem using the XMP library, we fear that our system will not be feasible for large gate arrays. To give the reader a sense of scale, a state-of-the-art gate array such as the 1 μm CMOS TGC118 has 18,620 sites in a 133x140 array [4]. Routing such an array with our method might result in a linear program approximately 100 times larger than our 12x15 example. Since linear programming by the simplex method (as in XMP) is generally assumed to run in average time $O(n^3)$, a rough estimate of Sun-3 runtime on a 133x140 array might take $(100)^3$ times as long as our 12x15 example: 10^8 seconds, or about two years.

Since the Cray-2 typically runs at tens of MFLOPS, and our Sun-3 with a 68881 co-processor typically runs at tens of KFLOPS, we naively hoped to see our runtime decrease by a factor of 1000 when we ported our program to a Cray-2. With the expected 1000-fold speedup, we hoped to route a full TGC118 array in a day of Cray-2 runtime. (While this would not be an economical use of a day of Cray-2 runtime, our wire-routing system is still in the developmental stage. Its runtime should improve greatly with algorithmic adjustment.)

The purpose of this report is to warn its readership against a naive expectation of a 1000-fold speedup of a Cray-2 over a Sun-3. On our code, we observe only a 6-fold speedup! We also generalize from our experience, providing a conceptual framework for similar problems in porting from workstations to vector supercomputers. Our framework is organized as a series of five questions, treated in turn below.

1 What is an acceptable runtime for solving this problem?

When working with large programs, an initial problem is to determine the longest acceptable runtime for the application. In our application, we believe an end-user would be willing to invest hours or days of desktop time, or up to an hour of supercomputing time, to obtain a solution for a large problem of 10,000 to 100,000 constraints. Such a problem would arise in the global routing of wires for a state-of-the-art gate array with perhaps 20,000 gates.

We have timed our 832-constraint linear program on four different machines: a Sun-3/160, a Sun-4/110, a DECstation 3100, and a Cray-2 supercomputer. Figure 1 is a table of our results.

Using the highest level of compiler optimization, the Sun-3/160 ran the program in 101 seconds, the Sun-4/110 in 42.1 seconds, the DECstation 3100 produced an error message, and the Cray-2 ran the program in 20.9 seconds. After trying all combinations of optimization and vectorization, we found the best runtime for the Cray, 18.0 seconds, was obtained with no vectorization.

— BUT NOT PARALLELIZATION (AT THE TIME, THE PRERELEASE FRONT-END TO THE COMPILER CRASHED WHEN ASKED TO PARALLELIZE XMP)

The -O3 compiler option for the DECstation apparently produces incorrect code. Accordingly, we use the -O2 result of 19.9 seconds, as the best result for the DECstation 3100. Note that the Cray-2 supercomputer is only marginally faster than this workstation.

Under the assumption that simplex codes, such as XMP, generally run in $O(n^3)$ time, we plotted a graph of the number of constraints in a problem versus the number of seconds needed to solve the problem. See Figure 2. In view of the time limits developed above, we plotted the execution time of the Cray-2 to one hour and all of the workstations to seven days. We also included an extra line for the Cray-2 that assumes a factor of 10 increase in speed for a hand-optimized program. This factor of 10 speedup is the limit for typical Cray XMP code [5, pages 6–7]. We presume that similar speedups can be achieved on the Cray-2.

CRAY BUZZWORDS "MICROTASKING + MACROTASKING"

With a factor of 10 speedup for hand-optimized code, Figure 2 implies the Cray-2 could solve a problem of approximately 30,000 constraints in one hour. The DECstation could solve the same size problem in 10 hours and a problem with 90,000 constraints in 7 days.

Readers familiar with numerical codes know it is dangerous to extrapolate runtimes of floating point programs as we do in Figure 2. For one thing, our analysis is based on a naive $O(n^3)$ model of runtime. More devastatingly,

Computer	Compiler	Compiler Options	Page Faults	Real Time	User Time	System Time
Sun-3/160 • Sun UNIX 4.2 release 3.5 • CPU: MC68020 • FPC: MC68881 • 16.7 MHz clock • 8 MB memory • 2 MIPS	f77 • ver 2.10	-f68881	0	127.0	124.2	0.8
		-f68881 -fstore	0	127.0	124.2	0.8
		-f68881 -O	0	103.0	101.0	0.6
		-f68881 -O -fstore	0	113.0	111.3	0.7
Sun-4/110 ME-8 • SunOS UNIX 4.0 • CPU: MB86900 32 bit SPARC • FPU: Weitek 1164/1165 • 14.28 MHz clock • 8 MB memory • 7 MIPS	f77 • ver 2.10	-O1	0	59.0	58.8	0.3
		-O2	0	42.0	42.1	0.1
DECstation 3100 • ULTRIX-32 ver 2.0 rev 7.0 • CPU: MIPS R2000 • FPU: MIPS R2010 • 16.67 MHz clock • 16 MB memory • 64 KB instruction cache • 64 KB write through data cache • 14 MIPS	f77 • ver 1.31	-O0	3	31.0	27.7	0.3
		-O1	3	27.0	24.0	0.3
		-O2	3	23.0	19.9	0.3
		-O3	The compiler produced incorrect code.			
Cray-2 • UNICOS 4.0 • 4 CPU's • 512 Mword memory	cft77 • ver 3.1c	full nozeroinc	--	63.0	20.9	0.16
		full zeroinc	--	27.0	20.9	0.16
		novector nozeroinc	--	46.0	18.1	0.12
		novector zeroinc	--	43.0	18.0	0.24
		off nozeroinc	--	56.0	21.0	0.14
		off zeroinc	--	38.0	21.0	0.18
		*	--	33.0	19.0	0.15
		**	--	37.0	19.1	0.12

* la05bd.f -e novector,nozeroinc
all others -e full,nozeroinc

** laos.f -e novector,nozeroinc
all others -e full,nozeroinc

(5.0 SEC UNDER UNICOS 5.0.7 IN DOUBLE PREC

(19 SEC IN EXTENDED)

Figure 1: Timing results for a linear program with 832 constraints

CRAY X.MP 3.4 SEC

$\log_{10} 2 = .301$

~~$\log_{10} 2 = .6$~~

$\log 3.6 = .556$

$\log 6.048 = .782$

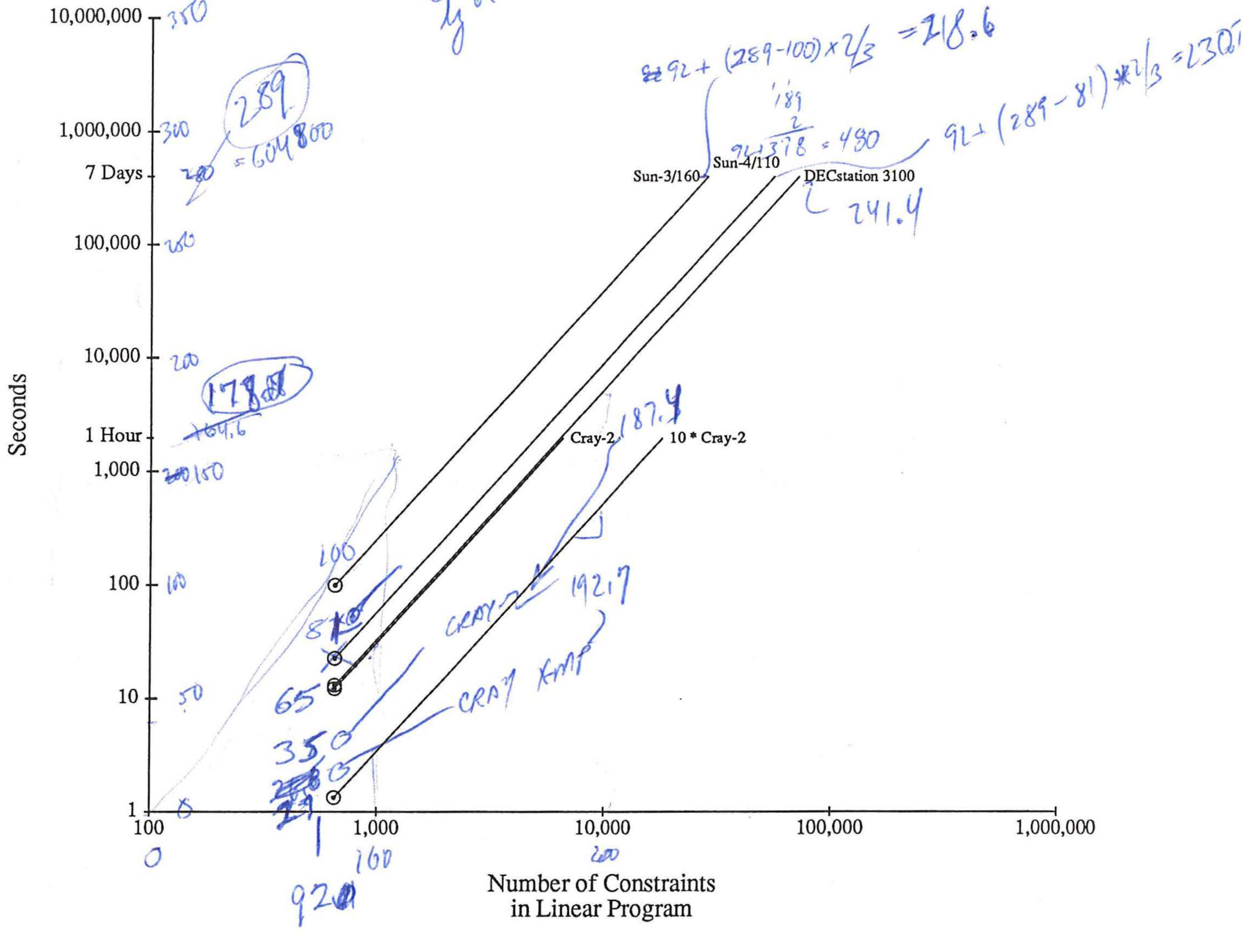

Figure 2: Size of a linear program solvable in x seconds

TERMINATES NORMALLY,

such an extrapolation ignores the problem of round-off error. With larger problems, we cannot be sure that the results will be accurate or that the program will even complete. The extrapolation of Figure 2 is thus suggestive but certainly not definitive.

OCCASIONALLY OBSERVED ON SUPERCOMPUTERS,

2 Can the code be easily vectorized?

In general, code that was written for use on scalar machines will not take advantage of the specialized functional units of a supercomputer. To gain ~~the full speed advantage of a vector machine~~ THE 1000-FOLD SPEEDUPS, programs must be vectorized [5, page 81]. This can be accomplished in two ways.

The simplest way to vectorize a program is merely to use a vectorizing compiler without making any modifications to the source code. Unfortunately, most programs contain constructs that inhibit a vectorizing compiler. In particular, any of the following constructs hamper the vectorization of loops: recursion, subroutine calls, I/O statements, assigned GOTO's, certain nested if statements, GOTO's that exit the loop, backward transfers within the loop, and references to non-vectorized external functions [5, page 90]. Even if the compiler is successful at vectorizing an inner loop, loops that are iterated only a small number of times will run more slowly when executed in vector mode.

The second way to vectorize code is to have a programmer restructure the program. If the programmer is successful, the compiler will be able to recognize code fragments which are vectorizable, resulting in a significant speedup over unmodified code. ~~Even if successfully~~

WHETHER A SPEEDUP IS OBTAINED OR NOT, ~~[illegible]~~ RESTRUCTURING CODE IS AN EXPENSIVE AND TIME-CONSUMING PROCESS. IN COMPANY PERSONNEL MUST BE HIRED, TRAINED, FAMILIARIZED WITH THE CODE AND THE VECTORIZING COMPILER; ~~THE [illegible] RESTRUCTURED CODE [illegible]~~ ~~MUST BE~~ THOROUGHLY TESTED AND DOCUMENTED.

In our experiments, we found that the standard Cray FORTRAN compiler, cft77, is unable to produce any vector code that will speed up the execution of the unmodified XMP solution routines. Upon investigation, we found that the inner loop of the XMP solver iterates just 1.8 times. This finding is sufficient explanation for the failure of cft77 to obtain a speedup by vectorizing the code. We believe the only possibility for obtaining a vector speedup in the XMP solution of our linear programs is for a skilled programmer to make extensive revisions to the code, perhaps restructuring its sparse matrix calculation from a row-oriented to a block-oriented method.

LACKING SOURCE CODE FOR THE CPLEX SOLVER, WE ARE UNABLE ~~[illegible]~~ TO EVALUATE ITS VECTORIZATION BY INSPECTION. ~~PROBABLY IT VECTORIZES ONLY SLIGHTLY~~ HOWEVER, SINCE ITS MANUFACTURER CLAIMS ONLY A ?-FOLD SPEEDUP OF A CRAY X OVER A DECSTATION 3100, WE CONCLUDE IT IS NOT EFFECTIVELY VECTORIZED.

3 Will the program be used enough to warrant hand-optimization?

As we noted in the preceding section, global compiler directives are not always sufficient to give much (or any) speedup. Hand-optimization is necessary to exploit the functional units of a supercomputer and gain its full

ONLY 3 OR NEW,

~~[illegible]~~ of

UNLESS ONE'S CODE IS INFEASIBLE ON A HIGH-PERFORMANCE WORKSTATION OR AN IBM RS6000, IT IS NOT WORTH THE TROUBLE TO PORT IT TO A SUPERCOMPUTER. ONCE IT IS PORTED, IT IS NOT WORTH OPTIMIZING UNLESS THE PROJECTED TOTAL SUPERCOMPUTER RUNTIME EXCEEDS TEN HOURS. OTHERWISE, ONE WILL SURELY SPEND MORE

speed advantage over a scalar processor. The time required to hand-optimize a program will depend upon the size of the utilized code [5, pages 6–7], the style in which it was written, and the programmer's familiarity with it. By style we mean, "Does the program contain good comments and descriptive variables?" Of course, the expense of programmer-mediated optimization must be offset with the savings gained from the decrease in execution time.

Whenever possible, of course, one looks for similar code that has already been vectorized. For example, we are aware of the existence of packages for solving large linear programs efficiently on the Cray supercomputers; we are now investigating their suitability in our application.

A final, and relatively inexpensive, alternative to the use of a vector supercomputer is to run unmodified, unvectorized code on a high performance scalar processor such as a DECstation 3100.

IN OUR EXPERIENCE PORTING WORKSTATION CODE TO A CRAY ~~HAS~~ A POTENTIAL FOR DISASTER. ~~[illegible]~~ A LARGER LINEAR PROGRAM (FOR A CRAY XMP) WAS SOLVED ON ALL WORKSTATIONS BY THE XMP SOLVER, BUT WAS NOT SOLVABLE ON A CRAY-2 USING THE SAME FORTRAN CODE. WE SUSPECT THAT THE CODE WAS TUNED, IN ITS TERMINATING CONDITIONS, FOR ~~IEEE~~ IEEE-STANDARD ARITHMETIC.

4 Is precision a factor in the computation?

Our fourth question is pointed at the relative ~~imprecision~~ "SINGLE-PRECISION" of ~~the~~ floating point arithmetic on the Cray-2. ~~The Sun workstations and the DECstation~~ ALL MODERN WORKSTATIONS follow the IEEE 754 floating point standard. The Cray-2 does not.

The Cray floating point storage format has 1 sign bit, 15 exponent bits, and 48 mantissa bits. Floating point numbers have the most significant bit in the leftmost mantissa bit, giving 48 bits of precision [1]. The IEEE-standard double-precision storage format has 1 sign bit, 11 exponent bits, and 52 mantissa bits. Here, the most significant bit is implied, giving the user 53 bits of precision [2, page 1-9], fully 5 bits more storage precision than is available on the Cray-2.

The Cray multiplication algorithm uses 56 bits of precision [1]. This is significantly less than the intermediate format of the IEEE standard, which has 64 bits of precision [2, page 1-9]. The 56-bit Cray result is not an intermediate format in the IEEE sense, since it is not available after the multiplication is complete. Thus, for example, inner-product accumulations are done with at most 48 bits of precision on the Cray, whereas the IEEE STANDARD standard provides 64 bits.

We conclude that all of the workstations we tested are superior to the Cray-2, in respect to the precision of their floating point arithmetic. In our case, floating point precision is important: our preliminary investigation shows the Cray-2 is incapable of solving a 2,000-constraint problem using the untuned XMP code.

WE NOTE, IN PASSING, THAT HIGHER-PRECISION ARITHMETIC IS AVAILABLE ON SOME WORKSTATIONS AND ON THE CRAY, FOR USERS WILLING TO PAY A SUBSTANTIAL TIME PENALTY (~~[illegible]~~ A FACTOR OF FOUR TO TEN) FOR TWICE AS MUCH PRECISION.

~~WITH THE RELATIVELY SMALL INVESTMENT OF A WEEK OF~~

~~I BELIEVE THAT SMALL~~

~~[illegible]~~ TO ESTIMATE TOTAL SUPERCOMPUTER RUNTIME,

~~I SUGGEST~~ ~~[illegible] THE [illegible]~~

IN SUMMARY, TO ANSWER ~~OUR~~ THE THIRD QUESTION, I SUGGEST YOU ESTIMATE YOUR TOTAL WORKSTATION RUNTIME, MULTIPLY BY TEN TO THIRTY, THEN COMPARE THE RESULT TO TEN HOURS.

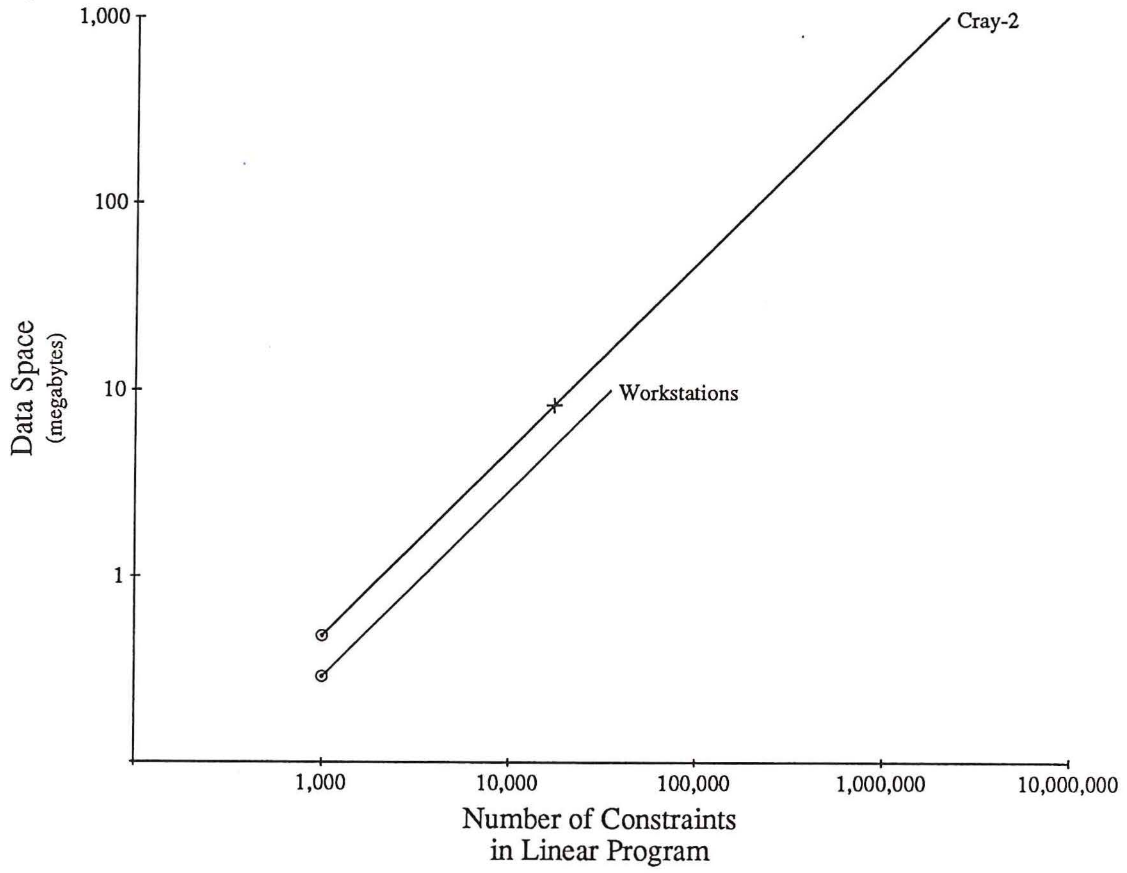

Figure 3: Data space required for a problem with x constraints

5 How large is the program's working-set?

A significant advantage of the Cray-2 is that it can be configured with hundreds of megabytes of fast RAM. By contrast, it is a rare workstation that has more than a few tens of megabytes. Thus, if your program has a working set in excess of, say, 20 megabytes, but less than 500 megabytes, it will execute much more rapidly on a vector supercomputer — even if your code is not vectorizable.

As indicated in Figure 1, our workstations' memory ranges from 8 to 16 MBytes. Of this, at most 12 MBytes is available for a program's workspace. In contrast, the memory limit for the Cray-2 is 64 MWords interactively and 450 MWords when using the batch queuing system [3, page 59]. A Cray-2 word is 8 bytes, giving the user a 1000-fold advantage in size over a workstation.

In our application, a linear program with 1,000 constraints uses 0.5

MBytes of memory for data storage on the workstations, and 0.7 MBytes of memory on the Cray-2. The difference results from the use of 4-byte integers on the workstations, as opposed to the 8-byte Cray-2 integers; we have no need of the extra integer precision.

The workstations thus provide enough memory for problems up to approximately 60,000 constraints, according to the linear extrapolation of Figure 3. The Cray-2 memory line extends far to the right, up to more than a million constraints. The "+" on the Cray-2 line indicates the point (30,000 constraints) at which our Cray-2 runtime reaches one hour. Due to runtime limitations, then, we will probably be unable to take advantage of the Cray-2's vast memory capacity.

6 Conclusions

To determine if it is worthwhile to run a program on a supercomputer, we suggest you answer five questions.

First, you should determine an acceptable runtime limit for solving your problem. In our case, we believe that up to one hour of supercomputing, or up to seven days of desktop execution, is acceptable for large linear programs.

Second, determine if your code can be easily vectorized. Sometimes, the compiler on a supercomputer will produce a significant decrease in execution time without extensive code modification. In all other cases, a programmer must restructure the code. In our application, restructuring for effective vectorization would take a considerable amount of effort and a high level of expertise.

Third, if the vectorization achieved with compiler directives will not be sufficient, you should decide if the program will be used enough to warrant hand-optimization.

Next, consider the issue of floating point precision. The Sun-3, Sun-4, and DECstation follow the IEEE 754 floating point standard, while the Cray-2 does not. This gives the workstations five more bits of precision in stored floating point numbers and sixteen more bits of precision in inner-product accumulations. This lower level of precision for the Cray-2 may limit the size of problems that it can solve.

Finally, consider the size of your program's working-set. Will it fit on a workstation without excessive paging activity? In our case, we expect to encounter runtime limits before we have trouble with paging.

In view of our answers to the questions above, we conclude it is not worthwhile to port our XMP-based wire routing software to the Cray-2. We are now investigating other packages of linear program solution routines to see which, if any, are suitable for our application.

Acknowledgements

We would like to thank Roy Marsten, John Gregory, and the staff of the Minnesota Supercomputer Institute for their assistance in technical questions arising in the course of this research.

References

[1] *Cray-2 Computer System Functional Description.* Cray Research, Inc. Manual HR-2000.

[2] *MC68881/MC68882 Floating-Point Coprocessor User's Manual.* Motorola, Inc., first edition, 1987.

[3] *MSC User Guide – MSI Special Edition.* Minnesota Supercomputer Center, Inc., 1.1 edition, May 1989.

[4] *TGC100 Series 1mum CMOS Gate Arrays Data Manual SRGS007.* Texas Instruments, Inc., 1989.

[5] John M. Levesque and Joel W. Williamson. *A Guidebook to FORTRAN on Supercomputers.* Academic Press, Inc., 1989.

[6] Roy E. Marsten. *XLP Technical Reference Manual.* XMP Software, Inc., July 1987.

[7] Antony P-C Ng, Prabhakar Raghavan, and Clark D. Thompson. Experimental results for a linear program global router. *Computers and Artificial Intelligence (Czechoslovakia)*, 6(3):229–242, 1987.