LA-UR - 73-565

TITLE: COMPUTER PROGRAM OPTIMIZATION USING SOFTWARE MONITORING TECHNIQUES

Conf-730501--1

AUTHOR(S): A. Frank McGirt
Lawrence E. Rudsinski
K. Jerry Melendez

SUBMITTED TO: AEC Scientific Computer Information Exchange (SCIE) Meeting May 3 and 4, 1973 at Sir Francis Drake Hotel, San Francisco, Calif.

los alamos
scientific laboratory
of the University of California
LOS ALAMOS, NEW MEXICO 87544

Form No. 836
St. No. 2629
1/73

UNITED STATES
ATOMIC ENERGY COMMISSION
CONTRACT W-7405-ENG. 36

COMPUTER PROGRAM OPTIMIZATION USING SOFTWARE MONITORING TECHNIQUES*

F. McGirt, L. Rudsinski, K. J. Melendez
Los Alamos Scientific Laboratory
University of California
Group C-4
P. O. Box 1663
Los Alamos, New Mexico 87544

ABSTRACT

Concern about the efficient use of computer resources at the Los Alamos Scientific Laboratory (LASL) has led to the development of software performance measurement tools and a formal computer program optimization project. In this paper the design, implementation, and use of such tools is discussed. A description of the optimization project execution procedures and typical gains in program efficiency versus manpower and machine time costs are given. Problems and recommendations which are of interest to others who plan development of computer performance measurement tools or establishment of formal optimization projects are discussed.

INTRODUCTION

Use of the computer at the Los Alamos Scientific Laboratory (LASL) has become one of the most valuable Laboratory resources. Controls of computer usage such as by allocation and direct charging of machine time have been implemented. This, in turn, has caused users to become more concerned about the execution efficiency of their codes. Were they making full use of the computer time for which they were paying? It was this growing concern about code efficiency that clearly pointed out the need for performance measurement tools and perhaps a formal code optimization effort.

COMPUTER SYSTEM CONFIGURATION AND JOB MIX

The LASL computer system configuration consists of three Control Data Corporation (CDC) 6600 computers and two CDC 7600 computers. However, the discussion will be restricted to the 7600 computers because most of the production work is done on these machines. The CDC 7600 computers each have $65,536_{10}$ words of Small Core Memory (SCM) and $512K_{10}$ words of Large Code Memory (LCM), three $40M_{10}$ word disk modules, and six tape drives. Four line printers, a card reader, and a card punch are available for each machine. The 7600's operate under the LASL developed CRØS operating system and are about four times faster than the CDC 6600's for production jobs.

The job mix at LASL is quite varied. Of the jobs run on the 7600's, 64% finish in less than 30 seconds and 92% finish in less than 5 minutes. Only 1.7% of the total jobs that execute on the 7600's run for longer than 30 minutes, but these jobs use more then 55% of the total available 7600 machine time. It is clear then that a few programs use most of the machine time and that an optimization effort should be initially directed toward those few programs.

PRELIMINARY OPTIMIZATION STUDY AND TOOL DEVELOPMENT

Before a formal optimization project could be tackled some tools and information had to be assembled. First we wanted to find out how efficiently the hardware was being used. LASL had done a very limited amount of hardware monitoring which consisted primarily of hooking counters to the 7600 functional units and measuring the MIP (millions of instructions per second) rate. These measurements showed that the object code generated by the FØRTRAN

*This work done under the auspices of the Atomic Energy Commission.

compiler executed at a rate between 9 and 12 MIP. The theoretical maximum execution rate for the 7600 is 36 MIP. This indicated that the 7600 hardware was executing only 25% of the instructions which were theoretically capable of being executed because the FORTRAN compiler was generating inefficient object code. Although work on the compiler would surely result in definite gains in execution efficiency, it was likely that some gains could also be made using optimization techniques with the current compiler.

We first wanted to know where time was being spent during program execution. For this we needed a tool which would monitor codes during execution and describe the areas being used most heavily. We decided on the following design goals for such a tool:

1. The tool must provide meaningful execution time statistics such as I/O wait time, and it should determine the relative frequency with which instructions are executed.

2. The tool should be easy to use.

3. The tool should use minimum overhead and interact with the executing program as little as possible.

4. Output should be displayed in an easily comprehensible form and quickly be accessible after each run.

With these goals in mind we decided that monitoring the program address register (P-register) was probably most feasible. By March of 1972, we had designed and implemented a routine called STAT[1] for execution on the 7600's. STAT makes use of a 7600 hardware interrupt feature to save the contents of the P-register every 3.5 milliseconds during program execution, with virtually no overhead to the executing program. At the end of the job, the results are displayed as a printer plot histogram which shows code address versus the number of times the P-register was found to be equal to the code address. Application of STAT to several user jobs showed that we had indeed satisfied our design goals and were able to isolate heavily-used areas of executing programs.

The next piece of information that we judged necessary to have before starting a formal optimization project was whether or not the laboratory programmers were using good techniques and whether they were taking advantage of certain features of the 7600's which were not available on the 6600 computer. To accomplish this we began a survey of randomly selected 7600 jobs from the input shelves to learn what percentage gains in efficiency could be expected. Efficiency gains were to be achieved in three rather simple ways which would serve to limit the effort expended on each program. The three methods used were:

1. Improve the FØRTRAN programming techniques,

2. Correct deficiencies in system routines, and

3. Improve numerical algorithms.

This survey ran for three months during the summer of 1972, and showed that significant gains (an average of 39% for 20 cases) in program efficiency could be realized with relatively small investments in manpower* and machine time. In addition to the significant improvement in program efficiency, the summer survey had other benefits. Programming techniques were discovwhich the FØRTRAN programmer could use at the source code level to improve object code execution efficiency.** Several tools were developed during the survey as the need became evident. One of these tools was a program called REGREF[2] which

---

* A full-time programmer and a summer student were responsible for the survey.

**These optimal programming techniques are being published for the benefit of all LASL programmers.

processes the print file from a FØRTRAN compiler or CØMPASS assembler and indicates delay cycles between instructions. Delays resulting from register conflicts, functional unit conflicts, and the instruction stack (which may have been introduced by poor programming techniques) are given. A register reference of the 24 CPU registers is also given.

OPTIMIZATION PROJECT

Based on the results of the summer survey and the fact that some necessary tools for software performance measurement had been developed, it was decided in December 1972 to begin a formal optimization project. This project has two primary goals:

1. To examine user programs which use appreciable amounts of machine time to determine if efficient use is being made of LASL computers and to improve programming efficiency where possible. Heavily used areas of code are isolated so that optimization efforts can be concentrated.

2. To document the optimization techniques developed during the project for future reference by LASL programmers.

The procedures for project execution are defined as follows:

1. The LASL Computer Division (C-Division) will provide a project leader whose duties are to accept jobs and assign them a priority, monitor the path of jobs, keep the cost and performance records for each job and provide the primary contact with users. In addition, the Computer Division will provide code optimizers, consulting, and assistance in optimization techniques.

2. A person will be appointed from each division to coordinate optimization activities with C-Division. This person will be responsible for insuring that appropriate test jobs and current production versions of the programs are made available to the C-Division optimizers and will monitor the results of the optimization. Since a good relationship between the code authors and the code optimizers is very important to the success of the project, the division coordinator will also provide this liaison.

3. After obtaining test jobs and pertinent documentation, the heavily-used areas of the codes to be optimized are isolated using the STAT routine previously described. These areas are analyzed by the optimizers and recommendations for improvement are made to the code author.

4. The code author can then make the necessary changes himself or take advantage of the laboratory programming services.

The above procedures imply close user-optimizer cooperation and communication. This is very important when large production codes are to be optimized because the optimization techniques will depend in large part on exactly what the author is trying to accomplish with his code. The optimizer is dependent on the code author to provide this information.

The project as presently begun will operate in stages. Stage 1 will concentrate on those codes which use the most 7600 time (> 20 hours per month). Stage 2 will include codes which use somewhat less time ($10 \leq t \leq 20$ hours per month). Stage 3 will accept the remainder of the production codes in the laboratory.

Stage 1 is presently staffed with five people, each spending 1/2 of their time on optimization related work. A sharing effort between several people is important because of the "cross fertilization" between project members who are expert in different areas of programming and mathematics. With several areas of expertise available, the entire program execution procedure can be analyzed for optimal execution. In fact, the analysis of each code to be optimized has been broken up into several areas which match the interests of individual project members. The areas which are analyzed for execution efficiency are:

1. Numerical algorithms.
2. FØRTRAN programming techniques.
3. Inner loops for possible reprogramming in machine language.
4. Existing machine language routines.
5. Program structure including data handling and I/O.

After the analysis is complete, the project members meet to discuss their findings and to prepare a report to the code author. Reprogramming of the code is then begun, either by the author or by a project member.

To date, work has been completed on three large production codes used for hydrodynamics and neutronics calculations. The use of each of these codes has varied between 100 to 150 hours/month of 7600 time for the past six months. Between 20% and 30% improvement in execution speed was realized for each code; however, the techniques used for achieving the gains were different in each case. Examples of some techniques which we used to achieve significant gains are described below.

Small FØRTRAN subroutines which are heavily used in inner loop calculations can be rewritten in machine language. The execution statistics which were obtained using the STAT program during a typical production run showed that 60% of the total run time for this particular code was being spent in a 500 word FØRTRAN subroutine. The FØRTRAN was analyzed and the decision made to rewrite the subroutine in machine language since little gain was evident from redoing the FØRTRAN. The reprogramming in machine language took about 1/2 man-month including debugging and testing and gained about 50% in execution speed for the subroutine which is a 30% improvement for the entire code. This is a savings of 50 hours/month of 7600 time.

If inefficient FØRTRAN subroutines are large (> 500 statements) with lots of branching and transfer statements so that program flow is relatively hard to define, reprogramming in machine language is usually not very effective. One can instead improve the FØRTRAN programming techniques. For example, on the 7600 a divide is very slow and should be changed to a multiply.* The code discussed above is an example of the case where large gains can be made with little investment. The next code to be optimized is a perfect counter example. The execution statistics for this code showed that there were few areas which were used much heavier than other areas. The three most heavily-used areas were picked for closer study. Two short FØRTRAN subroutines were rewritten (in FØRTRAN) and a portion of the FØRTRAN main program was replaced by a machine language subroutine. The analysis and reprogramming required about 2 man-months of effort. The gain in execution speed was about 24%, which translates into a savings of 24 hrs/month of 7600 time. In this case, the payoff was smaller for a much larger effort.

Yet another technique which can be used to achieve large savings in execution time, but may be expensive in manpower, is the understanding of algorithms and their subsequent

*See Reference 4 for other examples.

reprogramming for efficient execution. For example, one user was employing a least-squares spline package for smoothing experimental data. The data were equally spaced which meant that no sophisticated mathematical techniques were required for smoothing. A single weighted averaging technique would have sufficed.

One, of course, must sometimes employ a combination of these and other techniques. Execution statistics for a large production code showed two major problem areas. The first was a machine language subroutine which was being used about 30% of the total time for a typical production run. Close analysis revealed that the routine was essentially a translation of the object code produced by a FØRTRAN compiler. A study of the program flow pointed out many illogical and unnecessary branches. In fact, we discovered that one of the illogical branches was an error which had existed for several years. We redesigned the program flow for more efficient execution and a careful rewrite in FØRTRAN produced a version of the subroutine which executed as fast as the original machine language version. The FØRTRAN version was then rewritten in machine language and about a 15% improvement in execution speed was achieved for the entire code.

The second problem area for this code was the I/O wait time which varied from 7% to 20% of the total run time. Recommendations for changes which would improve this situation were made to the code user. He modified the data handling procedures and achieved up to a 20% reduction in execution speed for the entire program. A total savings of about 30 hours/month were achieved for this code.

SUMMARY

Given in the Table below is a summary of the costs and the gains for a three-month optimization effort on production codes. It is clear that the savings in computer time far outweigh the cost of the project, at least for the codes which have been optimized to date. However, there are several problems which were encountered which could add greatly to future project costs.

Some codes are much harder to optimize than others because of the detailed analysis required for understanding what the user is trying to accomplish with the code. This is particularly true when codes have been handed down from user to user as is the case with some of the older production codes at LASL. The initial logic is usually modified so that the documentation, if present, may be of little use. It

TABLE 1

OPTIMIZATION PROJECT COST EFFECTIVENESS SUMMARY

Total Usage of Production Codes before Optimization	370 hrs/month
Savings in Computer Time after Optimization	104 hrs/month
Manpower Expenditure for Optimization	3 man months
Computer Costs for Optimization	3 hours

is, therefore, probably not cost effective to try to optimize codes when little can be learned about the internal logic. The cost of analysis may outweigh the gains in execution speeds for such codes.

Codes which have no heavily-executed areas as compared with other areas are costly to optimize, because significant gains are usually accumulated from smaller gains. Several areas may yield gains of 3% to 5% in execution speed after much effort has been spent in analysis and reprogramming.

Subroutines in which a large fraction of time is spent in tight inner calculational loops and subroutines which are designed for a very specific (but time consuming) task are the most amenable to optimization. Usually, substantial gains in execution speed can be realized by using more efficient algorithms and either by improvement of the programming techniques or by reprogramming the inner loop in machine language. Machine language, however, should be used with care and good documentation must be provided because machine language routines are harder to maintain and may be difficult to move to new machines.

Debugging and testing efficient subroutines which have been written for inclusion in large codes may also prove quite expensive, especially if subprogram linkages are very complicated and difficult to isolate. For these cases using simple driver programs to test the subroutines is almost impossible. The subroutines must be tested as part of a regular production run or a tool must be developed which would allow monitoring the execution of the program and displaying intermediate results when appropriate. An interactive program debugging package is presently being developed by optimization-project people at LASL. Having or being able to develop program monitoring and debugging tools as needs become evident is a key part of any computer program optimization effort. Setting aside part of the optimization project resources for tool development not only helps to take care of technical project needs, but it also provides some variety for project members who are involved in quite complicated programming tasks. The freedom to design and implement software tools important to project requirements is a relaxing and productive alternative.

A final consideration which is extremely important to the success of an optimization project is the program author-program optimizer relationship. As stated earlier, close author-optimizer cooperation and communication are essential because the optimizer must know what the author is trying to accomplish with his program. The optimizer should recognize that often a high level of ego involvement exists between the computer program author and his program[3], and the optimizer should manage his relationship with the program author accordingly.

CONCLUSION

The optimization project presently underway at LASL has shown that appropriate specialists using appropriate tools and techniques can significantly increase the execution speed of production codes provided they have the cooperation of the authors and/or prime users of those codes. The work completed to date has shown a 3 to 1 profit/cost relationship. This ratio should increase as more optimization techniques and better tools are developed during the course of the project.

REFERENCES

1. "STAT: A Software Performance Measurement Tool for the CDC 7600", K. J. Melendez, F. McGirt, D. A. Plaisted, and L. E. Rudsinski, Rio Grande Chapter of the ACM, Santa Fe, New Mexico, October 13, 1972.

2. "Program Library, Abstracts and Usages", Programmers' Information Manual, Vol. 2, Los Alamos Scientific Laboratory, Los Alamos, New Mexico, 1972.

3. The Psychology of Computer Programming, Gerald M. Weinberg, Van Nostrand Reinhold Co., New York, 1971.

4. "CDC 7600 FØRTRAN Optimizing Techniques", F. McGirt, L. Rudsinski, and K. J. Melendez, LA-5219-MS, Los Alamos Scientific Laboratory, to be published.