MANAGEMENT CONSIDERATIONS IN COMPUTER EVALUATION

Jeffrey Pulcini
Perkin-Elmer Corporation
Binghamton, NY 13901

ABSTRACT

Quality and programmer productivity are more than the latest buzzwords. Quality is demanded by the customer and productivity is essential in competitive procurements. To that extent, the selection of the computer system should be a management issue as well as a technical issue.

This paper is directed towards managers who have previously limited their role as the decision maker in the world of computer selection and now see the need to take control of a business area that is a significant cost driver.

There are two major areas of interest in evaluating a computational system. These major areas are:

1. Evaluation of computer performance
2. Program development environments.

Within each area of discussion, the results of research and management guidelines will be presented as applicable.

INTRODUCTION

In the early 1970's, the computer hardware accounted for as much as 25% of the cost of a simulator. In the late 70's, the cost was approximately 10% of the value of a simulator. Today, this cost is reduced to 5% of the total procurement value. This has led many managers to believe that the computer selection process is no longer as important since the computer is regarded as a "commodity".

In reality, the selection of a computer system is more important than ever. While the hardware cost is approaching zero, the cost to develop, debug and install software is rising geometrically. Further, life cycle costs, supportability, vendor investment in the industry, and the support the vendor gives the contractor can dramatically affect the cost of implementing the simulation software.

COMPUTER EVALUATION

A Short History

In the 1960's, we made the step from analog simulators to digital simulator technology with our own machines. Rediffusions R-2000 and The Singer Company's GP series computers were designed for simulation and there was no question as to the computational performance of the machines or caveats in the computers design of which to beware. After all, they were our babies, and we knew them well.

In the early 70's, commercially available minicomputers were being built and sold in large quantities. These machines proved more cost effective than those manufactured by the simulation community. One major problem with the commercially available machines was determining the performance level available for simulation.

Performance measurement was attacked using empirical and theoretical analysis based on our "simulation instruction mix". We felt confident. After all, we coded in assembly language, wrote our own executives, and had a good knowledge of the simulation tasks to be performed. In essence, we did a pretty good job of evaluation.

In the mid 70's, things began to get out of control. The simulator using community began to require simulator code in high level languages and the use of the vendor operating system for the control of the simulation programs. Further, the simulation requirements were growing in terms of both fidelity and the number of auxillary training systems such as record-playback, malfunction insertion and instructor displays. These factors challanged our ideas of instruction mix and overheads. We tried to adapt.

The final blow came in the late 70's, early 80's. This time the culprit was technology. The building blocks of computers, chips, were becoming more dense and less costly. In turn, the designers were able to architect systems using techniques that were previously cost prohibitive.

These techniques included instruction pipelining, caching, and complex instruction sets. The latter of these would be exploited by the new compiler technology which was also coming into its own.

The bottom line of all of this was that traditional methods of rating computers became inadequate and worse, misleading.

Pitfalls in Computer Evaluation

The major pitfalls in computer evaluation can be divided into the following areas:

1. Inadequate system testing.
2. Skewed results.
3. Vendor software evaluation.

System Testing. Inadequate system testing can be attributed to several factors. The first factor is historical. In the past, the one portion of the computer system tested was the CPU. This was not a problem since little I/O was done and our problem was primarily compute intensive. Today's environment is more complex. With record/playback, visual displays within the cockpit as well as outside the cockpit, and a wider range of instructor support CRTs, I/O plays a very important role. Yet, very few benchmarks exercise the I/O system during computation or the CPU during I/O.

The second factor that contributes to inadequate testing is cost. Simply, it takes time and resources to do a proper job. Management has been either unwilling or unable to devote the resources necessary.

Finally, in most companies, true computer expertise is as rare a commodity as the other engineering disciplines. Therefore, finding that individual with the skill, knowledge, and experience, who knows the production environment and can then translate that into valid testing is extremely difficult. Invariably, this individual is involved in "more important" (or at least more urgent) assignments. This lack of proper, experienced analysis, coupled with increased complexity of the production environment and the computational hardware available leads to skewed results.

Skewed Results. There are numerous techniques and devices used by computer scientists to improve the performance of a processing system. Two of these devices are cache and optimizing compilers. If the evaluator is not prepared to cope with these devices, the results will be misleading.

Caching is a technique by which a high speed buffer is placed between main memory and the CPU. By providing data and instructions to the processor faster than main memory could, cache will boost the machines performance. The traditional small benchmarks, such as the Whetstone, may fit in and execute from cache and thus will give artificially high results.

In a recent paper, Drs. Cho, Lin, and Jen demonstrated that programs whose execution range exceeded cache size could experience a degradation of as much as 78% over the cache resident version of the program [Cho84]. Figure One is reproduced from the aformentioned report.

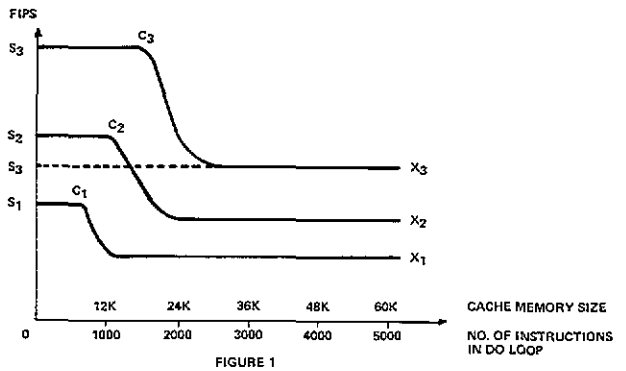

FIGURE 1

CACHE CAUSED PERFORMANCE DEGRADATION

SOURCE: ANNUAL SIMULATION SYMPOSIUM, MAY 1984

In short the machine could be drastically overrated by small, unrepresentative benchmarks.

Optimizing compilers can cause similar problems. High quality optimizing compilers develop a great understanding of a program's structure and variable usage. Benchmarks that contain loops that do nothing may have that code segment entirely removed. After all, why generate code (work for the CPU) that has no meaningful result? So, while the compiler has made an intelligent decision, the results of the benchmark will show a higher than normal performance.

As mentioned earlier, I/O plays a significant part in the simulation task. The need for I/O is significant in that it can affect the performance of a processor. The processor gets its data and instructions ultimately from main memory. If the memory is busy with I/O, the processor waits. How long and how often the processor waits is a function of the bandwidth of the memory system and the processor speed.

Figure Two shows memory bandwidth requirement vs. processor speed. Cache can reduce this, but its effects are variable primarily as a function of memory architecture and program flow.

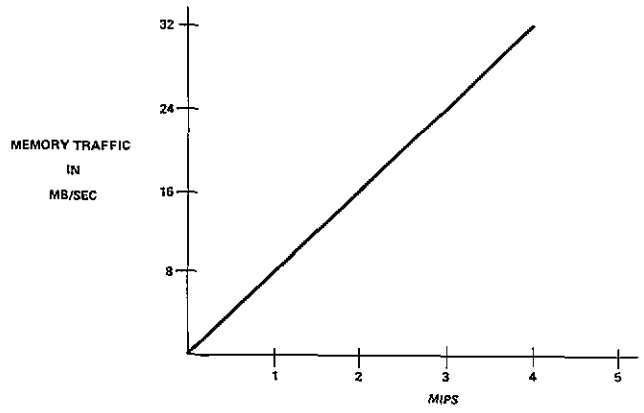

FIGURE TWO

MEMORY BANDWIDTH NEEDED FOR SIMULATION TASKS

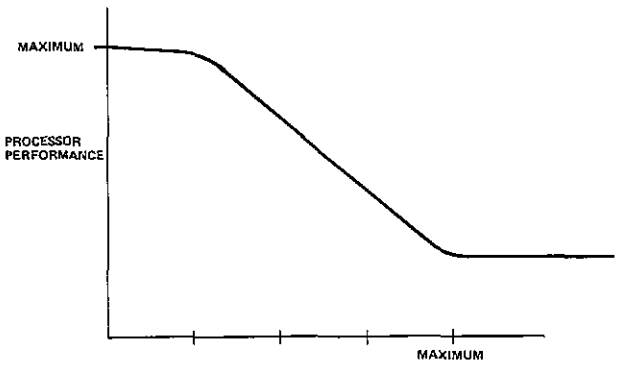

FIGURE THREE

PROCESSOR PERFORMANCE UNDER I/O LOAD

Figure Three shows how a non-cached processor's performance degraded as I/O load was increased. On a system of limited memory bandwidth, processor performance can degrade from the "benchmarked" level.

Vendor Software Evaluation. If there is one area that receives little attention, it is the software system provided with the computer. The lack of analysis in this area is not surprising. Historically, we have used little vendor software. We wrote our own executives, did our own I/O, and worked in assembly. In the last ten years, however, we have been raquired to use high level languages and vendor supplied operating systems. This software can have a significant effect on our ability to "get the job done". For the moment, the area in which we are most interested is the real time capabilities.

Often, if analysis of software capabilities is attempted, it is based on one or two parameters. For operating systems, context switch speed and I/O overhead are the most common parameters requested. These parameters are so ill-defined, that misleading conclusions are often drawn. In fact, a broad range of issues are left unaddressed. Some of these issues are:

1. Are there hardware/software architecture limitations requiring modifications to our current real time design philosophy or limiting program growth or portability within a computer product line?

2. Can we use the operating services as they exist or must we work around them?

3. If we must work around the OS, will we still be customer specification compliant (i.e. no modification of the vendor OS)?

4. Will the OS help us do our job?

The final point in the evaluation of real time software requirments has to do with structured code. With MIL-STD-1644A and Ada* now a reality, the demands of structured programming must be faced.

In brief, the more structured the language and implementation, the more computer resources are required to execute the code. In a previous work, this author cited testing done on highly structured code [Pulc83]. When increasingly sophisticated optimizations were used, the physical instruction execution count decreased from 39.6 million to 15.2 million instructions. For the trainer manufacturer and the ultimate customer, the implications are simply this: buy more hardware and increase acquisition and life cycle costs, or spend more computer time optimizing.

Guidelines for Evaluation

How can an organization prevent improper evaluation? The following guidelines are submitted as a basis from which to begin.

1. Create a computer selection board. This board would consist of senior people with experience in program development, various vendors' systems, marketing, sales, and program management.

2. Be certain that benchmarks and other evaluation criteria are representative of the production environment and capable of realistic use of caches, compilers, and I/O.

---

* Ada is a trademark of the United States Department of Defense (Ada Joint Program Office).

3. Be prepared to revise and revisit benchmarks and evaluations as the production environment changes and vendor product lines evolve.

4. A good benchmarking system will not be completely portable. This is an advantage. It will allow time to be spent by the technical members of the evaluation board using the potential vendors' development system. The next section focuses on the importance of the development environment.

5. Work with the vendor rather than trying to isolate the vendor from your production environment. This will serve two purposes. First, the amount of support a vendor gives during a sales situation is a small measure of his ability to support you after the sale. Second, by attempting a preliminary design of a real time system on the vendors' equipment, the board may uncover the strengths and weakness of the proposed hardware and software.

Section Summary

The importance of proper evaluation is found, not in the right or wrong of the evaluation, but in the costs incurred based on the evaluation. These costs fall into two areas. First is the cost to purchase additional computer hardware to bring the performance up to the level needed for simulation. This cost may be as significant as buying a complete CPU and shared memory. This cost is quantifiable and much weeping and gnashing of teeth usually accompanies this decision. Though potentially great, this cost is small compared to the hidden cost.

The hidden cost is the cost to redesign and rework the simulation load. This cost is more significant since it occurs during the hardware/software integration phase, when time is at a premium, and design documentation is complete.

PROGRAM DEVELOPMENT ENVIRONMENTS

While accurately gauging hardware and software performance will reduce risk and rework costs, it is the program development cycle that contributes most heavily to the costs of a training device. This being so, management's attention has turned to finding a development environment that reduces costs by improving productivity.

Up to a few years ago, the only development environment for a computer system was the proprietary system supplied by the computer vendor. This locked the training device manufacturer into that vendors' compilers, linkers, and editors. Further, if tools were developed, those tools were most likely not portable to other vendors' processors. The UNIX** and Ada language environments were developed to address these problems.

UNIX and Ada have captured the attention of both the technical staff and management. Both environments promise a measure of portability, user friendliness and increased productivity. Because of this, many view these environments and their attendant languages as a panacea. Nothing could be further from the truth. While each environment will go further than *any previous environment in* achieving the above objectives, it remains imperative to approach each with a "heads up" attitude. The following subsections on UNIX and Ada are an attempt to put the capabilities and limitations of each environment into perspective.

The UNIX Environment.

UNIX is the hottest word in the computer market place today. This collection of software has been touted as the be all and end all of operating systems. Increased productivity, program portability, machine independence, user friendliness and standardization are arguments used to encourage migration to the system. It is important to examine these arguments carefully.

Portability and Independence. When a person speaks of portablilty, most people envision a no effort transportation of software from one machine to another. While such mobility is possible it is seldom achieved even in user level software. A common language such as C or Ada helps, but other factors such as arithmetic formats, memory organization and word lengths may cause execution problems even if compilation proceeds smoothly [Wall82].

These problems are exacerbated when portable system software (operating systems, etc) is attempted. Operating system software must address different peripherials, different hardware memory management schemes, and even differing numbers of registers within the CPU. In essence, it is a "porting" rather than a "transporting" of software that occurs.

In support of this statement, it should be noted that AT&T and only AT&T has UNIX. Everyone else has systems based on AT&T UNIX. It is in the porting of the UNIX system from AT&T hardware to other vendors' hardware that incompatabilities may occur.

** UNIX is a trademark of AT&T Bell Laboratories.

Numerous case studies have been published showing the trials and tribulations of porting UNIX [Jali83, Tils83]. A good rule of thumb is a high quality port of a UNIX system requires approximatly six to twelve months. The most significant detriment to transporting UNIX is the collection of physical differences in the base hardware. Because of hardware differences, commands and utilities may vary between systems.

Of equal concern are the "enhancements" to the system each vendor is free to make. If a computer vendor is to remain competitive, he must have something to separate him from "the other guy". System programmers may innocently use commands and utilities they believe to be common to all UNIX based systems only to find later these commands and utilities were unique to a certain vendor. Just as vendor enhancments made FORTRAN code non-transportable, so will UNIX environments and tools be non-transportable if great care is not exercised.

One other factor must be considered when portability is discussed. AT&T is now a competitor in the computer market place. No one is certain how this will affect the distribution and compatability of subsequent releases of UNIX [Whit84].

User Friendly? It is said that beauty is in the eye of the beholder. So it is with computer systems and software. The friendliness of a system is a function of who you are and what you're trying to accomplish. To the computer professional, the UNIX environment provides tools and facilities second to none when developing system software. In fact, the UNIX environment is not only user friendly, but user helpful. To the non-professional, however, the UNIX environment has been described as user hostile. This disparity leads to many a heated debate [Bese84, Gall84, Hold84, Litt84, Metz84].

An environment can be created that will be "friendly" to the most naive user of a UNIX system. This will require resources for the development of a shield product. For the manager committed to investing in such a productivity tool, careful control must be exercised over its development to prevent portability problems when moving from one vendor's system to another.

Increased Productivity. By and large, the single most important factor in turning to a UNIX environment is the promise of increased productivity. Indeed, by all accounts, the productivity gain is there. One merely has to keep in mind that there is no free lunch. There is a large learning curve for those coming from traditional systems to a UNIX system. By comparison, documentation is poor and error messages are generally cryptic or non-existant. With proper planning and training the transition to a UNIX system can be accomplished with satisfactory results [Kola84].

Ada and the Development Environment

Ada is upon us with a promise of, not only a language, but a programming environment that is standard. Problems solved? I think not. While Ada will go further than any system has gone to date, by necessity it will still contain many pitfalls discussed in the previous section.

One of the great powers of Ada rests in the fact that it is a closed language in an open ended system. The Ada language is "closed" because, unlike FORTRAN, the specification gives maximum as well as minimum syntax requirements. This means, with the exception of hardware limitations, any program written in Ada will run on any machine after recompilation. To take advantage of this portability and encourage tool kit building, Ada also supports packages.

A package can be thought of as a library of common, logically related routines. These routines are self contained entities of both instructions and data. The ability to have a tool kit full of packages that may be joined together to perform different functions is what gives Ada its open endedness. It is in this open endedness that portability problems may occur.

Just as in the UNIX world vendors had to develop a competitive edge, so an edge must be found in the Ada world. This edge may be in vendor supplied packages. Portions of a vendor supplied package may contain vendor hardware or software specific routines. Thus, as with the UNIX environment, these vendor packages may be innocently used to build a useful tool which now becomes non-portable.

Other Factors Affecting Productivity

System Support. The linkers, compilers, and packages discussed in the previous section form what is known as the APSE -- Ada Programming Support Environment. If you haven't realised how big this system will be, let me assure you large amounts of resources will be needed to support the Ada system. In a tongue-in-cheek article, W. E. Drissel of Cyberscribe Associates observed "Every IBM 370 APSE will require the usual complement of OS system programmers plus a small army of APSE specialists to support applications programmers" [Dris82]. He may not be far from the truth.

The current trend is to build Ada on top of a UNIX system, and UNIX is well known for its need of resident gurus. Whether this trend continues or not, the software development environment needs support not only for problem resolution, but to enhance the system with tools for the engineering user. Despite what appears to be an increase in overhead, a coherent tool kit, and timely problem resolution will yield consistant long term benefits.

Physical Facilities. While not germane to rating computers, it is important to mention that the most neglected area for improving productivity is the physical environment in which the programmer works. An engineer who produces software and documentation needs a large, uncluttered work area, quiet and pleasant surroundings, sufficient storage, and easy access to terminals and small meeting areas [McCu78].

Guidelines for Evaluating Environments

Today's programming staff is being asked to provide more software to meet the demands of higher fidelity training and increased functionality. Further, increased emphasis is being placed on structured design, rigorous design reviews and detailed documentation. There is a great need to track the design and provide configuration control of both software and documentation.

Before evaluating any vendors development environment, the wise manager asks:

1. What are the tasks to be performed?

2. What are the skills of the people who perform these tasks?

3. What tools have been produced in the past? Which are still being used and why?

4. What other tools are commercially available and what is their cost in dollars, computer resources, and training?

5. Will the current or proposed environment accept these tools?

6. What support staff is required to integrate, maintain, and assist in utilization of these tools?

7. What future requirements will be placed on the corporation as a result of new languages and environments such as Ada?

With these questions in mind, the manager can now begin to examine the various proposed solutions in the quest to improve productivity and reduce cost.

Section Summary

The need to improve productivity has been recognized for many years. UNIX and Ada are broad attempts at an answer to this need. Yet, they are not without problems. UNIX has the potential for being unstable and unfriendly. Ada is still in its infancy with what currently appears to be little DOD standardization on APSEs and only recent attempts to standardize Ada's interface to any vendor supplied operating system.

But, we must still produce trainers, and to that end, the manager should have a vendor's program development environment evaluated as closely as the real time environment. The computer selection board should choose a system that is tailored to the development staff and their skills, rather than trying to tailor the staff to the environment.

Finally, if this section has seemed negative towards any particular solution, it was not meant to be so. Rather, it is hoped that it has brought to the surface that no supplied environment has ever been or ever will be the total solution to a particular development problem. Therefore, management must be willing to devote resources to develop tools and techniques to solve current and long term problems. The selected vendor should have market stability and a track record for supplying high quality software that adheres to known standards thus helping to assure long term solutions.

CONCLUSIONS

The evaluation of a computer system should not be left in the hands of one or two randomly selected individuals. Rather, the selection should be accomplished by a committee of knowledgable individuals from R&D, marketing, program office, engineering, and purchasing.

This committee should be a permanently funded effort with the resourses to develop and maintain the tools necessary to evaluate computer vendor's offerings either as new hardware is marketed or as internal development and production requirements change.

1. The technical members of the committee should evaluate the proposed computer system in light of existing production and development needs.

2. R&D should evaluate the vendor and the proposed system in light of future production and development goals.

3. Program Office should look for competitive advantages the system

would create when incorporated in the training system.

4. Marketing should be involved to make known to the committee the customer's desires and to be prepared to redirect the customer if his desires are not consistent with the best technical solution.

5. Purchasing needs to be involved to evaluate the business relationships that accompany a computer selection.

Finally, a perfect solution will never be found. If given clear corporate direction, the committee should be able to choose acceptable systems and create a stable environment that will enhance programmer productivity, improve the overall quality of a system, and allow for highly competitive end product offerings.

LIST OF REFERENCES

Bese84 Besemer, Jim, "UNIX: What's good about it?," EDN, May 17, 1984, Volume 29, Issue 10, Page 106.

Cho84 Cho, C.K, Lin, E.K., Jen, C.L., "On Performance Evaluation of Multiprocessor Systems for Real-time Simulation," Annual Simulation Symposium, IEEE Computer Society, May, 1984.

Dris82 Drissel, W.E., "Some Observations, Predictions, Prejudices, and Impressions About Ada," Computer, IEEE Computer Society, January 1982, Volume 15, Issue 1, Page 120.

Gall84 Gallant, John, "Big Systems Users Discovering the Strengths of UNIX," Computerworld, May 2, 1984, Volume 18, Issue 22, Page 1.

Hold84 Holden, Steve, "More Opinions?," The DEC Professional, January 1984, Volume 3, Number 1, Page 24.

Jali83 Jalics, Paul J. and Heines, Thomas S., "Transporting A Portable Operating System," Communications of the ACM, December 1983, Volume 26, Issue 12, Page 1066-1072.

Kola84 Kolansky, Michael S., "UNIX in Research: A Plus But No Panacea," Micro Manager, Fairchild Publications, New York, May 1984, Page 12-15.

Litt84 Little, John, "UNIX: What's wrong with it?," EDN, May 17, 1984, Volume 29, Issue 10, Page 109.

McCu78 McCue, Gerald M., "IBM's Santa Teresa Laboratory - Architectural Design for Program Development," IBM Systems Journal, 1978, Volume 17, Number 1.

Metz84 Metz, Arthur, "A Rebuttal - UNIX Realities," The DEC Professional, January 1984, Volume 3, Number 1, Page 30.

Pulc83 Pulcini, Jeffrey, "Computer Performance Evaluation for Real-Time Simulation," Proceedings of the IEEE 1983 National Aerospace and Electronics Conference NAECON, May 1983.

Tils83 Tilson, Michael, "Moving UNIX to New Machines," AYIE Publications, October 1983, Volume 8, Issue 10, Page 266-276.

Wall82 Wallis, Peter J. L., "Ada and Portable Programming," Portable Programming, Halsted Press, John Wiley & Sons 1982, 11:89-93.

Whit84 White, C. E., "Battle of the UNIX Units," Telecommunications, May 1984, Volume 18, Issue 5, Page 113.

ABOUT THE AUTHOR

Mr. Jeffrey Pulcini is a National Systems Specialist for the Perkin-Elmer Corporation in Binghamton, NY. He holds a Bachelor of Science in Aeronautical Science, has authored several papers on computer performance and evaluation and has been involved in the simulation community for ten years.