U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

NISTIR 4331

National Computer Systems Laboratory

Emulation Through Time Dilation

John K. Antonishek
Robert D. Snelick

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
National Computer Systems Laboratory
Advanced Systems Division
Gaithersburg, MD 20899

CMRF

COMPUTER MEASUREMENT
RESEARCH FACILITY
FOR HIGH PERFORMANCE
PARALLEL COMPUTATION

May 1990

Partially sponsored by
- Defense Advanced Research Projects Agency
- Department of Energy

NISTIR 4331

EMULATION THROUGH TIME DILATION

John K. Antonishek
Robert D. Snelick

Advanced Systems Division
National Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

Partially sponsored by the
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

To appear in the proceedings of the
Fifth Distributed Memory Computing Conference,
Charleston, South Carolina, April 1990

May 1990

U.S. DEPARTMENT OF COMMERCE, Robert A. Mosbacher, Secretary

National Institute of Standards and Technology, John W. Lyons, Director

TABLE OF CONTENTS

Page

EMULATION THROUGH TIME DILATION

John K. Antonishek
Robert D. Snelick

Computation and communication are the primary dichotomy of loosely-coupled multiprocessor resources. Modeling and simulation are the techniques used to estimate machine performance based on the speed of these resources. The accuracy of the resulting performance estimates is often questionable, since such techniques cannot usually take into account all of the system detail: They become intractable in effort. Any real, hardware implementation of a multicomputer immediately fixes the speed of its resources, and can only yield a single point on the performance curve. We have implemented a third technique, called time dilation [2], to evaluate the performance of loosely-coupled multiprocessors by varying the ratio of computation speed to communication speed. This technique requires a high-speed clock and a test multicomputer system. Time dilation provides a way to measure accurately the performance of a given program on a variation of the physical transport system of a real machine.

Key words: communication; emulation; hypercube; measurements; models; performance; time dilation.

1. Introduction

MIMD (Multiple-Instruction stream, Multiple-Data stream) machines are commonly subdivided into two categories: shared memory machines and distributed memory machines. In shared memory machines, processors communicate through access to a common memory via an interconnection network (usually a bus). In distributed memory machines, every processor has its own private memory, and all communication and synchronization between processors is done via messages. The examples to follow are implemented

---

 Partially sponsored by the Defense Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, Virginia 22209 under ARPA Order No. 7223, April 15, 1987.

on an architecture of the latter type. The particular topology is a hypercube with 16 processors. The hypercube is a coarse grain-size machine; it possesses a limited number of powerful nodes. Communication between processors on such a machine is often *expensive*, and must be considered in the evaluation of a hypercube application. Time dilation aids in this evaluation by emulating various transport speeds of the communication system.

Algorithm performance on loosely-coupled machines is sometimes hindered by communication speeds, and at other times by poor algorithm design. Current methods of investigating the effects of communication speeds are modeling and simulation. We have implemented a third method called "time dilation" [2], which can be used to accurately emulate the performance of loosely-coupled multiprocessors by varying the ratio of two parameters: communication time and computation time.

Time dilation provides a way to investigate how increased communication speeds affect the performance of a particular application and architecture. This is achieved by allowing the user to increase, or dilate, the computational portion of the code by a specified factor, thereby creating the appearance that the communication speed has increased in relationship to the rate of computation. Time dilation will allow the user to readily and accurately determine to what extent a modified architecture (faster communications) of the particular machine would benefit that particular application. The advantage of this type of emulation is that all effects of the system are still present, whereas in simulation and modeling, not all system effects can be accurately accounted for. However, time dilation will only provide insight as to how a particular application will benefit from increased communications speed for the specific architecture upon which the time dilation experiments are performed. Time dilation provides a way to accurately predict the behavior of an application without taking the very expensive (and single-point) alternative of redesigning the communication hardware. A discussion on our implementation of time dilation and the results obtained from our preliminary experiments follows.

2. Implementation

To perform time dilation experiments, the operating system of the machine under investigation must be instrumented with code to perform the dilation function. This additional operating system code relies on a high-speed clock to accurately and quickly measure the computation and communication intervals. The high speed clock is provided by our Loosely-coupled TRAce Measurement System (LTRAMS). LTRAMS is a hybrid performance measurement tool that, among other things, provides a readable clock accurate to one microsecond [1],[7],[8].

An important distinction must be made between the computation and message portions of the code. The computational portion, at least for our preliminary experiments, includes the user computational work, the operating system routines called by the user, operating system message handling routines (*such as packet formation*), and miscellaneous operating system tasks (*such as interrupt handling and context switching*). The handling of a message by the operating system, including packet and header formulation, is considered part of the computation time because we are investigating just the effect of increased transmission speed of communication, not the actual method of message passing. A message passing implementation may include hardware formulation of headers, which does not require use of the CPU. This distinction can be changed in later experiments.

The user specifies the dilation factor (D) to the operating system through an operating system call. The user only has to add the statement specifying the dilation factor to the existing code to perform a time dilation experiment. To determine whether an application would benefit from faster communications, the user runs the application with a dilation factor of one (no change), and then runs the application with other dilation factors, all greater than one. The overall time for the application to complete for each run is "normalized" by dividing the time obtained from the run by the dilation factor for that particular run. If the normalized time from the run is nearly the same as the time from the run with a dilation factor of one, the application will not benefit much from an increase in communication speed. However, if the normalized time from the run is smaller than the run with a dilation factor of one, then the application would benefit. The larger the difference, the more the application can be expected to benefit from an increase in communication speed.

The implementation of time dilation for a loosely-coupled machine requires considering the period, t, between two user-initiated communication events (*i.e.*, the time between a send-and-receive, send-and-send, etc.) as containing four distinct parts: t_1, t_2, t_3, and t_4, where:

t_1 = user execution time
t_2 = system time in support of this process
t_3 = system time in support of other users (i.e., message store and forward)
t_4 = system time waiting for a communication to complete

The time that is to be dilated is $t_1 + t_2$ (the computation time), thus $t_{new} = t + (D - 1) * (t_1 + t_2)$. However, t_1 and t_2 are not easily measurable times, so the quantity $t_1 + t_2$ is computed from known times $t - (t_3 + t_4)$. If the time for t_4 exceeds the computed dilation time (*i.e.*, $t_4 \geq t_{new}$), then no dilation will take effect for this interval because the logical delay has exceeded the dilation time.

3. Interpretation

3.1 Approach

A program's service demand on a hypercube can be analyzed by investigating two modes of execution: computation and communication. A computational step is an arithmetic or logic operation performed on a datum within a processor. The computational portions of the code includes system interrupts and other operating system services that steal computational cycles from the program. For these preliminary experiments, the evaluation of the computational partition of a program is divided into two components: *user time* (composed of an application's computation, operating system services, and

system interrupts excluding interrupts for communication) and *communication interrupt time*.

In a communication step, a datum travels from one processor to another through the communication network. This communication service demand of a program is also defined as having two components: *receive time* and *send time*. Receive time is the measurement of time from an initiation of a receive request until completion of that request; this includes delays due to synchronization. Send time is composed of the measured time for a process to dispatch a message.

The partitioning of components to analyze a homogeneous hypercube program can be represented in a *use tree* (*UT*)[6]. A UT describes the system's dominant features. Figure 1 illustrates a UT skeleton for the evaluation of a general application on a hypercube. The measured execution time of the components sum to the overall service demand of the program. This provides a basis for analysis of an application on the hypercube. The examples that follow use these component times and show how they are affected by a faster physical transport network.

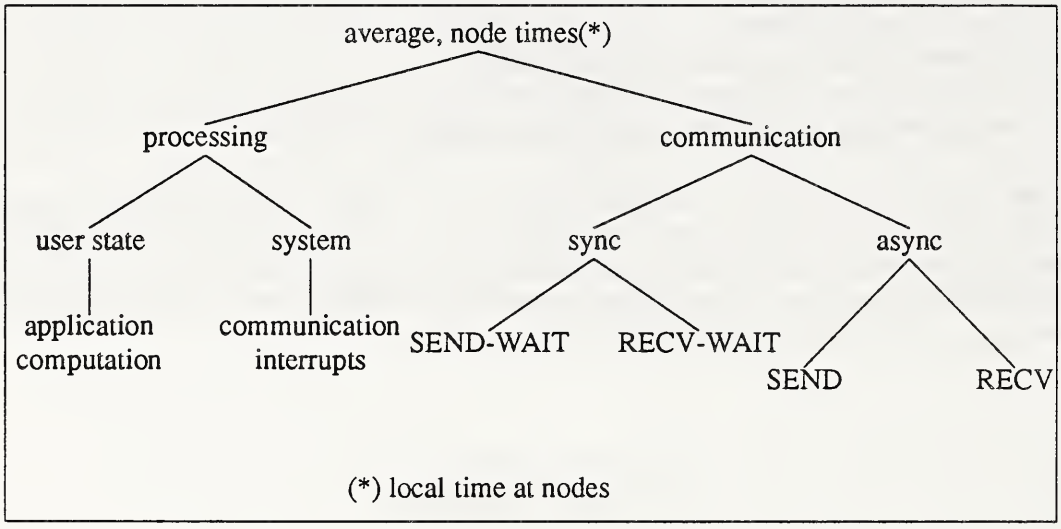

Figure 1. UT Skeleton for Hypercube.

3.2 Example Application: A Ring Model

The test algorithm is a synthetic ring benchmark that models applications which exemplify global process dependencies[3],[4],[5]. Molecular dynamics codes have computations of this character. The ring application is an implementation of a specific case of the UT diagram (Figure 1). It uses synchronous receive and asynchronous send protocols.

The program works as follows: A synthetic ring with nearest neighbor connections is created with N logical nodes. Each node (process) will originate a given number of messages, and additionally, process all other messages passing by. Each message makes a complete circuit around the ring, while being processed synthetically by each node. Once a message returns to its originator, it is removed from the ring network. Another message will be sent until that node has sent all of its messages. When all nodes have processed all of their messages the program is complete.

The ring provides a set of parameters that allows the user to simulate a wide range of application settings. The list of parameters includes the number of nodes in the ring, message length, the number of messages, and computation per datum. Communication in the ring is semi-synchronous, with messages being acknowledged within the program on a one-to-one basis. This flow control limits the number of messages piling up on a given node thus preventing buffer overflows. For this algorithm all communications (*i.e., messages*) are between neighboring physical nodes on the hypercube.

3.3 Analysis

To evaluate results, a normalized time (T_D) is defined as the average node service time (T) for a component (*e.g., receive service time equals the average receive time for N nodes*) divided by the dilation factor (D). The dilation factor (D) signifies the emulated speedup of the physical transport system. E is the overall elapsed execution time of the dilated program and E_D is a normalized elapsed time.

$$T_D = \frac{T}{D} \quad \text{and} \quad E_D = \frac{E}{D}$$

Normalized time gives a measure of what a component's time, such as receive time, or the overall execution time of the program will be on a machine with a faster physical transport system.

Another measure that is useful is the speedup of the application for a given D. This is obtained by dividing E_1 (*i.e.,* E_1 is the overall execution time of an undilated program) by E_D with a D greater than one.

$$Speedup = \frac{E_1}{E_D} \quad (where\ D > 1)$$

Speedup gives an indication of what the faster communication transport system yields in terms of overall execution time (*i.e., wall clock time*) performance of a program.

3.4 Results

Time dilation provides a way to investigate how a faster communication transport system affects performance of a given application and architecture. To illustrate this, two example problems are demonstrated from the ring benchmark. One investigates an application that is mostly computationally bound. Another shows an example at the other extreme: the application is burdened with frequent communications. Table 1 shows the application signatures[6] (*i.e., percent of time spent in a given mode*) of the examples to follow.

APPLICATION	condensed SIGNATURE COMPONENTS			
	User	Comm. Intr.	Receive-Wait	Async. Send
EX1: Computation	93.7%	0.5%	5.5%	0.3%
EX2: Communication	19.9%	5.1%	73.5%	1.5%

Table 1. E_1 Signatures of Two Ring Settings.

3.4.1 Example 1: Computationally Bound Application. As expected, for a mostly computationally bound program (*See Table 1*), the normalized components show little change with dilation. Figure 2 shows results. Normalized times are plotted for the four service demand components and overall execution time. User time makes up the majority of the overall execution time. Since the communication components demand little processing in such an application, clearly only slight benefit is derived from a faster transport system.

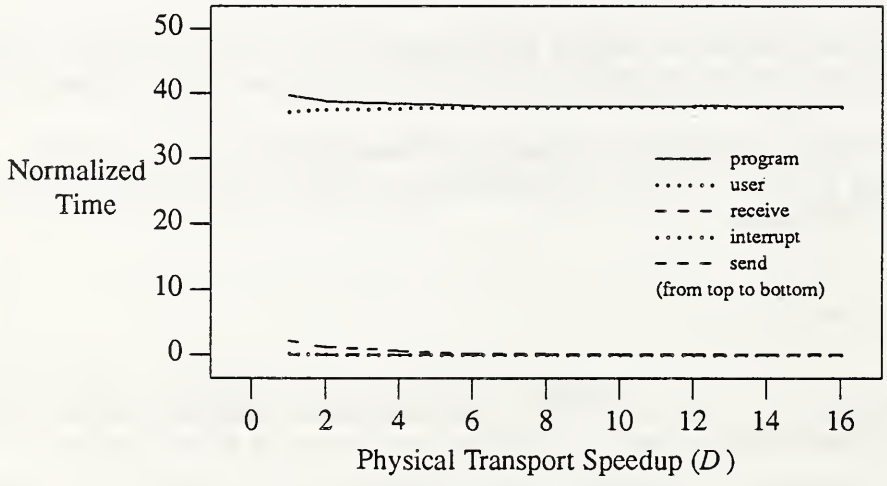

Figure 2. Normalized Time for Program and Service Demand Components for a Computational Bound Application.

The speedup for this example is negligible. No matter how much the communication system is enhanced, this application stands little chance for improvement. The user should investigate other areas to increase performance for this application. Since the computation load is high, the speed of the processor is the logical choice for investigation.

3.4.2 Example 2: Communication Bound Application. In this example the parameter settings for the ring are adjusted to simulate an application with frequent communications (*See Table 1*). Figure 3 illustrates results for dilation factors from one to sixteen.

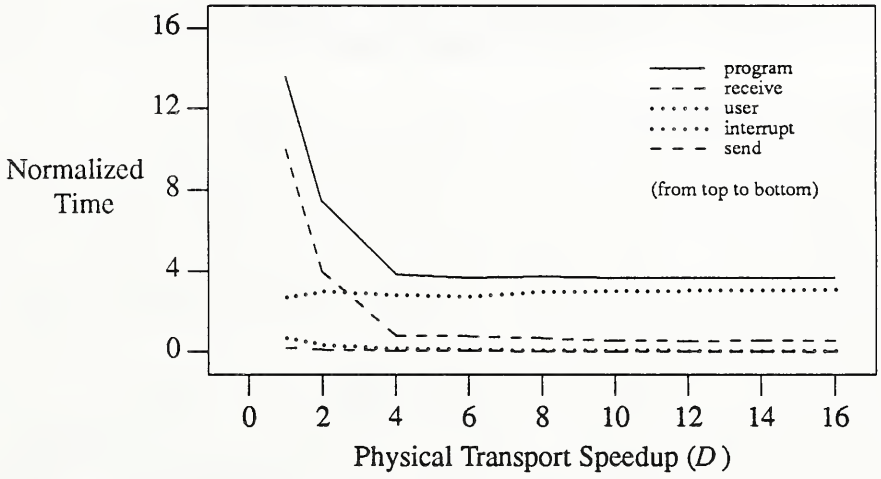

Figure 3. Normalized Time for Program and Service Demand Components for a Communication Bound Application.

As shown in the graph, the time spent in receive mode is significantly reduced as the speed of the physical transport system is increased. The reduction in this component directly accelerates overall execution of the program. This is clearly indicated by the shapes of the normalized program time and normalized receive time curves. With a transport system twice as fast, normalized receive time is reduced from 10.0 seconds to 3.85 seconds. Normalized receive time continues to decrease (*down to 0.8 seconds*) with a transport system speedup of four. At this point the computation-communication balance of the program shifts towards the computation end. In fact, the communication portion of the code now only makes up 22.1% (*previous 73.5%*) of the overall execution time of the program.

The speedup curve for example 2 is plotted in Figure 4. It is evident for this application that a significant increase in performance can be obtained from a faster communication system.

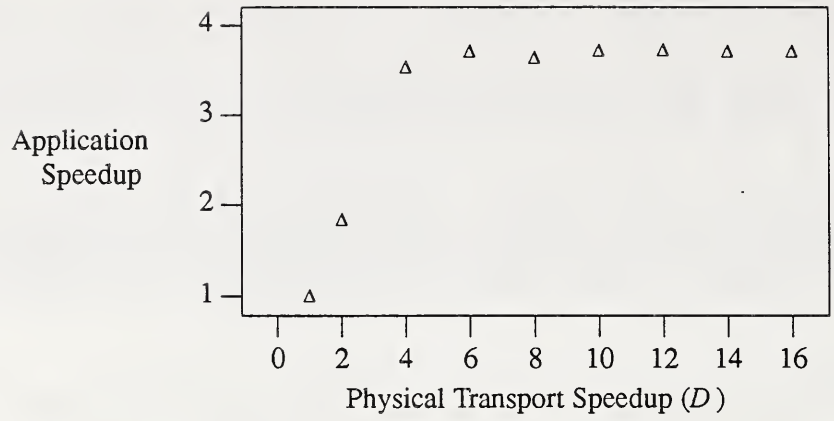

Figure 4. Speedup for a Communication Bound Application.

If the modified architecture has a communication system twice as fast as the original, this program exhibits a speedup of 1.8. This figure climbs to 3.5 when the transport system is four times as fast. The speedup of the program peaks to 3.7 with a dilation factor of 6. Beyond this point the curve asymptotically flattens off. The user would no longer benefit from any further enhancements to the communication system.

In the previous example, the application exhibits increased performance and thus will benefit from enhanced communication capabilities. However, if an application with a similar signature (*i.e., communication bound*) shows little speedup, it may be an indication that the algorithm is logically poor. In this case, logical delays are a problem, and revision of the algorithm should be considered. This is a topic that will be given a thorough treatment in the future.

4. Conclusion

Time dilation provides an accurate method to investigate various communication speeds of a loosely-coupled multiprocessor; this aids in the evaluation of a hypercube application. An application that exhibits increased performance with dilation will benefit from an enhanced physical transport system. However, an application that shows little change with dilation may indicate (i) a good match between algorithm and architecture or (ii) a logically poor algorithm. Time dilation allows the user to perform necessary winnowing before exploring other architectural alteratives.

4.1 Acknowledgments

The original design sketch for time dilation was proposed by Gordon Lyon during

development of the hardware instrumentation; he also proposed the interpretation of the results seen in the experiments. Gordon Lyon and Alan Mink read earlier versions of the text and suggested numerous details for improvement.

5. References

[1] Carpenter, R., *Performance Measurement Instrumentation for Multiprocessor Computers*, in High Performance Computer Systems, E. Gelenbe, ed., pp 81-92, Elsevier Science Publishing Co., New York, NY, 1988.

[2] Lyon, G. Personal communication: Concept of Time Dilation for Performance Measurement of Loosely-Coupled Multiprocessors, circa February 1988; A Simple Emulator for Variable Hardware Balance on a Hypercube, May, 1988; Time Dilation (Logical vs. Physical Delay), September, 1989.

[3] Lyon, G. On Parallel Processing Benchmarks. NBSIR 87-3580, June, 1987, 35pp.

[4] Lyon, G. Design Factors for Parallel Processing Benchmarks. *Jour. of Theoretical Computer Science,* April, 1989, 175-189.

[5] Lyon, G. and Snelick, R. Architecturally-Focused Benchmarks with a Communication Example. NISTIR 89-4053, March, 1989, 38pp.

[6] Lyon, G. Hybrid Structures for Simple Computer Performance Estimates. NISTIR 89-4063, March, 1989, 24pp.

[7] Mink, A., Draper, J., Roberts, J. and Carpenter, R., *Hardware Assisted Multiprocessor Performance Measurements*, Proc. of The 12th IFIP WG 7.3 International Symposium on Computer Performance: Performance 87, Brussels, Belgium, Dec. 1987, pp 151-168.

[8] Roberts, J., Antonishek, J. and Mink, A., *Hybrid Performance Measurement Instrumentation for Loosely-Coupled MIMD Architectures*, The Fourth Conference on Hypercubes, Concurrent Computers, and Applications, Monterey, CA, March 1989

NIST-114A
(REV. 3-89)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 4331

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
May 1990

4. TITLE AND SUBTITLE

Emulation Through Time Dilation

5. AUTHOR(S)

John K. Antonishek and Robert D. Snelick

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

10. SUPPLEMENTARY NOTES

[ ] DOCUMENT DESCRIBES A COMPUTER PROGRAM; SF-185, FIPS SOFTWARE SUMMARY, IS ATTACHED.

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

Computation and communication are the primary dichotomy of loosely-coupled multiprocessor resources. Modeling and simulation are the techniques used to estimate machine performance based on the speed of these resources. The accuracy of the resulting performance estimates is often questionable, since such techniques cannot usually take into account all of the system detail: They become intractable in effort. Any real, hardware implementation of a multicomputer immediately fixes the speed of its resources, and can only yield a single point on the performance curve. We have implemented a third technique, called time dilation [2], to evaluate the performance of loosely-coupled multiprocessors by varying the ratio of computation speed to communication speed. This technique requires a high-speed clock and a test multicomputer system. Time dilation provides a way to measure accurately the performance of a given program on a variation of the physical transport system of a real machine.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

communication; emulation; hypercube; measurements; models; performance; time dilation

13. AVAILABILITY

[X] UNLIMITED

[ ] FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).

[ ] ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.

[X] ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
13

15. PRICE
A02

ELECTRONIC FORM