Software Performance Modeling in PC Clusters

by

Wolfgang Baer
Associate Research Professor of Computer Science
Naval Postgraduate School, Monterey CA 93943
Tel. 408-656-2209 EMail baer@cs.nps.navy.mil

Steve Decato
Maj. US Army
Naval Postgraduate School, Monterey CA 93943

ABSTARCT:

Execution of course grain parallel programs in PC clusters promises super-computer performance in low cost hardware environments. However the overhead associated with data distribution, synchronization, and peripheral access can easily eliminate any performance gain promised by the individual cluster capacity. Application specific system performance analysis is required both to engineer PC cluster hardware and evaluate the cost effectiveness of parallelizing software components.

This paper presents a distributed system performance model and software analysis methodology suitable for estimating the execution times of large grain parallel application programs in clusters of PC hardware. The performance model emphasizes the use of application hardware performance results readily available in most systems. These are combined with single thread application software resource requirements in order to estimate the achievable execution rates in target clusters.

A case study of the analysis of a video realistic battlefield simulator implementation in a PC cluster running under Linux is presented. Benchmark results and performance estimates for specific candidate hardware configurations are calculated and compared with actual results.

KEY WORDS: Distributed Programming, PC Cluster, Performance Prediction

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 SEP 2000	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Software Performance Modeling in PC Clusters	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School, Monterey CA 93943**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**19**	

INTRODUCTION:

The study of computer architecture[1,2,3] has long been a subject which includes the estimation of software execution speeds and resource requirements. In the past execution estimates have largely been studied with the hope of improving computer hardware designs of single machines. With the emergence of networked computers and computer clusters4 the problem of designing software which executes in a networked environment now requires the software designer to become aware of networked architecture issues.

Coding standards such as PVM[5] , LAM/MPI6, and DCE7 are now available to allow writing of portable code for an abstract distributed machine. Deciding when to utilize such available standards and actually write distributed code is not clear.

*How to divide code into modules which can execute on separate machines?

*How to estimate the performance improvement of such code divisions in a specific networked architecture?

*When does it make sense to spend software development budgets on networked parallelization rather than faster single node hardware?

These are questions with which the software engineer and project manager must now deal.

This paper addresses these issues by providing a description of an analysis methodology applied to the problem of porting a perspective view scene generation program[8,9] from a single processor SGI workstation to a PC cluster environment. PC hardware has become powerful enough to compete with single workstations in recent years. The question of how to evaluate performance of networked PC clusters and whether such clusters can can compete with parallel workstations such as available in ONYX and Challenge SGI machines is an opewn one which will be addressed in this paper.

1) Serial Program Resource Description

In order to analyze the issue of porting workstation code to PC clusters it is necessary to provide a quantitative description of the programs to be implemented. Figure 1 shows a diagram of a generic sequential program. The program is divided into N calculations. Each is named by a numerical label J. Each of the J calculations used part or all of the global data generated by the previous calculations as input data and in turn generates global data as output.

There are four resource measurements available to estimate the resource requirements of such a computation:

*Its real time execution speed measured in wall clock seconds (RTEX[J]) *The average percent of the CPU utilized during this execution (CPU%[J]), *The data volume output DVOL[J,OUT] or input DVOL[IN,J]

* The average message size used for in and output of the data MSGZ[J,OUT] and MSGZ[IN,J].

It is convenient to write the resource characteristics of the program in the form of a matrix shown in figure 2. Here the rows have been labeled by the calculation names J while the columns have been labeled by the sequencial execution order, S, with which the calculations are carried out. Each element in the matrix is then characterized by the four resourcecharacteristics defined above. The diagonal elements represent the J'th sequential calculation in the original program defined in fig 1. The upper off-diagonal elements represent the send functions (SND)while the lower off-diagonal elements characterize the receive functions(RCV). Hence the send function SND[0,N] is responsible for sending DVOL[0,N] bytes of data from calculation C [0,0] to the communicating media while function RCV [N,0] receives the same number of bytes from the media and provides them to calculation C[N,N]. Solid arrows show the flow of data from a calculation through a SND/RCV pair and into the program space of a receiving calculation. The communication functions are necessary in order to explicitly analyze the program in a distributed environment. For a sequential implementation in a single processor only he diagonal elements are required and the matrix reduces to the linear form shown in figure 1.

In this notation the general formula for the run time of the program is simply the sum of all the calculations.

eq. - 1

The average percentage of the CPU utilized during the execution of the program is then:

eq. - 2

The number of CPU seconds (CPUS) utilized by the execution of the program can be consistently calculated from the formula:

eq. -3

The three equations presented above are sufficient to calculate the two main performance characteristics of a program. How fast it runs is given by RTEX. How much of the computer is being utilized to execute the run is given by CPU% or alternatively by the number of CPU seconds (CPUS) used by the program.

The purpose of this paper is to answer performance question when an already existing program is ported from one, possibly distributed, environment to another. It is therefore assumed that an executing copy of the code is available in some environment and that performance measurements are available.

Translating performance estimates then divides into two main tasks. These are,

1) estimate the resource requirements of a calculation module when changing machines.
2) estimate the resource requirements of the SND and RCV functions when changing the communication media between calculation modules.

1.1 - Battlefield Visualization Program Test Case

The cluster analysis presented in this paper uses an existing video realistic battlefield simulator[8,9] as a test case. The bare bones version of the program is divided into N calculations. There are three main calculation types. These are 1) a terrain data server which retrieves multi resolution (64 km to 1m) terrain data posts required for the calculation from disk, 2) a set of 1 to N-1 ray tracers each of which calculate the pixel color index for 1/(N-2)'th portion of the screen, and 3) a display calculation which interprets the index and scales pixel intensities to a display window.

A sequential implementation of the test case program is available and executes in a conventional single processor machine. Each of the calculating modules C[J,J] represented by diagonal elements passes a volume of data DVOL[J,S] to other modules through global memory areas using simple assignment statements such as,

```
/* RCV is implemented as*/   LOCAL_VARIABLE    = GLOBAL_VARIABLE;
/*SND is implemented as*/    GLOBAL_VARIABLE = LOCAL_VARIABLE;
```

These statements are part of the calculating module C[J,J] code. Their effect on run-time performance is included in the measurement of the module. Hence all off-diagonal execution times (RTEX[J,S]) in the programs resource characteristic matrix are zero and the double sum in equations 1,2 and 3 above reduces to single sums.

The sequential program characteristics of these calculation types for a single frame calculation in a 100Mhz Pentium 64MB using a Diamond 64 2MB display card are listed below.

Calculation Name	**Characteristic**	**Comment**
Data Server [0,0]	RTEX[0,0] =.2 sec.	one fifth execution per frame
	CPU%[0,0] =.8	
	DVOL[0,N] =.25MB	data broadcast to raytracers
	MSGZ[0,N] = 1024	average data block in bytes
Raytrace [1,1]	RTEX[1,1] =.6/(N-2)	run time per frame
to	CPU%[1,1] = 1	saturates CPU when running
Ratrace[N-1,N-1]	DVOL[1,N] = $2^{16/(N-2)}$	screen portion sent to display
	MSGZ[1,N] =256	column size
	DVOL[0,N] =.25MB	received from server
	MSGZ[0,N] = 1024	average message size
Display [N,N]	RTEX[N,N] =.04	display rate DIAMOND 64 2MB
	CPU%[N,N] =1	saturates CPU
	DVOL[N,1] =2^{16}	total 256x256 screen size(bytes)
	MSGZ[N,1] =256	minimum message received

For sequential execution when the communication media is global main memory only the diagonal elements of the run time equation 1 are non zero and the value is given by,

.

This agrees with the actual run time for a typical view calculated when the viewpoint is situated 300 meters above the ground with a 30 degree look down angle moving at 60 miles per hour requiring the terrain data server to execute a full update cycle once every five frames.

This video realistic battlefield visualization program has been coded for parallel execution in a 21 processor Transputer and PowerPC 601 based parallel computer costing $250K in 1994. It operates at approximately 16 FPS on this machine and at 1.2 FPS on a $1500 PC. Our immediate problem is whether or not it makes sense to port this program to a PC cluster using low cost communication links and how fast we can expect it to operate as we add PC computing nodes?

2) Conversion of Resource Parameters from One System To Another

The simplest option for increasing software performance is simply to buy faster compatible hardware. In order to calculate the performance parameters of a calculation module from one machine to another it is necessary to introduce two new parameters. These are the CPU speed of machine M (CPUSPEED[M]) and the peripheral or I/O speed of machine M (PERSPEED[M]). We also note that the difference between the CPU seconds used in a calculation and the run time of the calculation is usually attributed to the time the calculation has to wait on a peripheral to complete its task. The relation,

eq. 4

shows this relationship using WAITSEC to specify the time the calculation is waiting.

If a CPU M2 is faster than a CPU M1 we would expect the corresponding CPU seconds used for a calculation to be decreased as follows,

eq. 5

Similarly if the peripheral speed of machine M2 is faster than M1 we would expect the wait time for peripheral response to decrease as follows,

eq. 6

We can now calculate the run-time of a calculating module in the second computer from the resource measurements in the first computer by substituting eq. 5 and eq. 6 into eq. 4

and using the definition of CPUS from eq. 3. The resulting conversion equations simplify to the following equation for the run time

eq. 7

and the CPU utilization,

eq. 8 .

Although absolute measures of CPU and peripheral speeds are often given in terms of standard benchmarks for our purposes it should be noted that only the ratio of speed appear in the resource conversion equations. If we double the CPU speed without changing the peripherals in our system the run time of a calculating module which utilizes half the CPU is given by eq. 7 as,

Hence only a 25% speed up is achieved because the sample calculation spends the same amount of time waiting in both machines.

For our application the CPU is utilized almost 100%. The SpecInt95 benchmark[10] of the Pentium Pro 200Mhz machine is 8.2 compared with 3.33 for the Pentium 100 we are using here. If these numbers can be taken as relative measures of computer speed the RTEX for a frame on the Pentium Pro 200 should be 0.34 seconds or 3 FPS. A dual Pentium Pro would be expected to run between 4-5 FPS. This simple hardware upgrade is the practical criteria against which performance improvement achieved by cluster implementation must be compared.

3) Estimation of the SND and RCV Functions Resource Requirements

In the previous section we showed how resource parameters are converted between machines assuming no code changes and no architectural changes in the topology of the communication media in which the program executes. In order to estimate the performance of code ported from one network topology to another it is necessary to take the effect of the message passing overhead into account. We have formally included the message passing functions in figure 2 as the off-diagonal SND and RCV functions. In the simple case of a sequential program executing on one CPU these could be ignored. In a distributed environment however explicit functions must be included which in themselves require CPU and wall clock resources. We must therefore estimate the overhead associated with these functions.

We will assume that data is transmitted as messages of size MSGZ bytes. The average message size transmitted from calculation J to calculation S is then given by,

eq. 9 .

We will further assume that the communication system utilizes blocks of size BLKSZ in bytes so that the message is brocken into a sequence of full block transmissions (#BLKS) plus the number of bytes in the residual (#LEFT) which does not fit into a full block. The message size is related to these parameters as follows:

eq 10

A typical communication system will take some time (MSGSETUP) to process a message transmission request. It will break the message into blocks and take some additional time (BLKSETUP) to set up the transmission of each block. The actual data transfer then happens on a byte level interrupt basis and takes the inverse of the low level communication bandwidth (1/LLCBW) per byte to transmit. Using these parameters to characterize the message transmission the real time required for a message to go from calculation J to S is give by,

eq 11

To estimate the cpu utilization we note that both MSGSETUP and BLKSETUP usually involve CPU calculations without input output wait time. All the waiting occurs in the low level bandwidth term. If the communications peripheral were fast enough to keep up with the IO bus of the system then the CPU would be kept busy all the time and the run time and CPU seconds would be the same. The CPU utilization is then calculate by utilizing eq. 3 as follows:

eq 12

Here we have used the IO bus band width (IOBBW) in the numerator as a limiting factor. If the communicating peripheral is faster than the IO capability of the machine then the CPU%[J,S] = 1, hence the formula only holds for communication systems which are slower than the CPU's to which they are attached (i. e. LLCBW < IOBBW).

We have developed formula for communications functions resource parameters which can be substituted into eq. 1 and eq. 2 in order to estimate the overall performance of the program in a cluster. These formula are given in terms of new variables such as setup time and band width. In order for these expressions to be useful we must find ways of determining the value of these new parameters for the candidate clusters topologies under consideration.

3.2 Simplified Bandwidth Expressions

The formula developed in the last section define resource requirements for communication functions in terms of parameters which are application software dependent (DVOL, #MSG) and parameters such as setup time and bandwidth which depend only on the characteristics of the communication system itself. Much like the CPU speed and peripheral speeds parameters introduced in section 2 the application independent parameters can be measured once for a given configuration and applied to performance estimation of any program one wishes to port.

In order to facilitate measurements it is useful to introduce the application to application program bandwidth (A2ABW). This is the bandwidth of data sent from one calculation to another including all communication overhead from all the layers of data handling subsystems involved. This is after all the number most software designers wish to know. How fast can I get data from the address space of one calculation to another? A2ABW is simply the total data volume one program wishes to send divided by the real time it takes to send it. Utilizing the formula derived above the value of the application to application bandwidth is given by:

eq. 13

Interestingly enough this expression is independent of the data volume and depend only upon the size of the messages an application wishes to send and the characteristics of the communication subsystem.

Figure 3 shows a generic plot of the application to application bandwidth against message size. Communication parameters introduced in the can be extracted from features of this plot as follows. The locations of the bandwidth drop spikes are one BLKSZ apart and show the effect of the additional block setup time required whenever a new block is needed.

The shape of the curve to the first block size is independent of the number of blocks (#BLKS) and, if extended would reach the application level communication bandwidth

(ALCBW) asymptotically. Clearly as the message size increases to infinity the effect of the message setup time can be ignored and equation 13 reduces to:

eq. 14 .

The formula for application level bandwidth suggests that we might be able to rewrite eq. 13 in terms by rewriting the terms involving BLKSETUP and LLCBW as follows,

eq. 15 .

The is strictly true only if we treat the number of blocks (#BLKS) parameter as a floating point value and ignore the bandwidth drop spikes. The term in the bracket will be recognizes as the reciprocal of the application level bandwidth and by substituting this definition into eq. 13 we get the simplified bandwidth formula,

eq. 16 .

which is shown as the dashed curve in figure 3. The simplified formula of eq. 16 acts like an upper level envelope to the more accurate eq. 13 since it ignores the effect of lower level blocking. Similarly eq. 13 is an upper level envelope to a still more accurate formula which would include the effects of still lower level setup times. The cumulative effect of these approximations at all levels is to over estimate the communication band with relative to low level hardware capability. For our purposes the recognition that such over estimation is taking place is largely eliminated by using actual communication band with measurements hence the simplified formula are usually adequate.

Assuming we have made a measurement of ALCBW[J,S] by measuring the bandwidth for a large message we can now calculate the MSGSETUP parameter by making a second measurement at some small message size. If we designate the application to application bandwidth of message size N bytes as A2ABW@N then the message setup time can be calculated from this measurement from eq. 16 as follows:

eq. 17 .

Hence by making two measurements, one for a small message of perhaps 100 bytes and one for a very large message on the order of megabytes we have sufficient information to characterize the upper bound bandwidth of the communication channel represented by the dashed curve in figure 3.

In terms of these measurements and the simplified band with expressions of eq. 16 and eq. 17 we can rewrite the communication resource parameters as follows.

The run time for the SND and RCV functions from eq. 11 is now,

eq 18 .

Since run time for a communication is the same whether one is at the receiving or transmitting end this equation is symmetric in J and S meaning that the run time for the SND and RCV function is identical.

The CPU utilization from eq. 12 tentatively reduce to,

eq 19 .

We say tentative because CPU utilization is generally not symmetric between send and receive functions. Often the receive side must do more error checking and takes a higher fraction of the CPU than the send side. We have introduced the notion of an application bus bandwidth in the parameter APBBW to formally account for the fact that the CPU utilization is essentially a measure of the time the CPU waits on the network. The APBBW is the bandwidth measured from the application to peripheral without actually sending the data. It is asymmetric and can be measured by taking CPU% measurements on the send and receive side of the communication system. For approximately symmetric communication protocols this parameter can be equated to the large message application to application bandwidth when the computer is in a loopback configuration in which all the communication software is exercised but data is actually transferred within a single machine and not communicated to the outside.

3.1 Communications Parameters Measurement[11]

. Communications bandwidths and CPU utilization were measured as a function of message size using networked 100Mhz Pentium PC running under the Linux operating system. Communication hardware utilized included 3COM 100Mbit per second and 3COM 10Mbit per second ethernet cards.

Table 1: Communications Parameters Measurement Results

	Socket TCP/IP		Socket UDP		LAM/MPI Measured			LAM/MPI Theoretical	
Message size in bytes	BW MBS	CPU %	BW MBS	CPU %	BW MBS	CPU %		BW MBS	% ERROR

200	.5	23	.58	21	.31	43		.31	0%
600	1.01	23	1.08	21	.73	39		.73	0%
1,000	1.4	23	1.57	21	1.0	42		.99	1%
1,400	1.6	23	1.78	21	1.2	42		1.17	3%
2,000					1.3	36		1.35	4%
6,000					1.9	36		1.79	6%
10,000					1.8	36		1.91	6%
20,000					2.1	39		2.01	4%
80,000					1.8	27*		2.09	14%
640,000					2.0	67*		2.1	5%
*NOTE: Setup times of 1to 2 sec and disk activity indicated disk swapping									

Communication software included socket test code[12] utilizing UDP and TCP/IP protocols, the LAM implementation of the Message Passing Interface using MPI_Send() and MPI_Recv() functions. Though various network topologies were tested the primary results required for cluster performance analysis are direct point to point communications between otherwise idle machines with no additional network traffic. The results are shown in table 1 above. All measurements were made by sending messages of size shown in column one back and forth between two machines a large number of times while the resource measurement function TOP was running. The total number of bytes sent was divided by half the total time in order to get the one way bandwidth. The last message received was compared with the first sent for error detection. TCP/IP and LAM/MPI measured values shown on table 1 had no transmission errors. UDP had errors and was unreliable as expected. We noticed two important features of TCP/IP transmission which were surprising:

1) Errors occur when message sizes exceed the ~1500 ethernet packet size.
2) The socket write commands do NOT block, hence stacking up sequential writes in a tight loop still causes errors.

We found these effects to be true on SUN and SGI workstations using TCP/IP as well as under FDDI communications links when packet sizes were above 4500 bytes. Hence socket transmission though faster could not be considered a reliable application to application protocol without adding error checking software. The LAM/MPI implementation, though slower than direct socket code, is therefore the indicative of reliable application to application performance.previous section were estimated. The parameter values for the TCP/IP and LAM/MPI measurements are shown in Table 2 below.

Parameters from Table 2 were used in eq. 16 to calculate the application to application bandwidth for both TCP/IP and LAM/MPI. The last two columns of table 1 shows the theoretical calculations and the% error from measured result for the LAM/MPI communication links. Below 1000 byte size messages errors are negligible while above this size errors of 4 to 6% indicate the setup and handling time of internal block structures is not modeled

accurately in the approximation represented by eq. 16 as expected. For TCP/IP eq. 16 and measured results matched exactly for the ranges shown in table 1.

.

Table 2: Communication Measurement Parameters for T100 links between Pentium 100Mhz PC's

Parameter	Description	TCP/IP Value	LAM/MPI Value
APLCBW	Application Level Communication Bandwidth in Mega Bytes	2.5	2.1
MSGSETUP	Message Setup Time in Milli Seconds	.4	.47
APBBW	Application Bus Bandwidth in Mega Bytes	11	5.3
CPU%	CPU utilization for message sizes above 100 bytes	23%	39%

Using the measured values from table 1 communications parameters presented in the CPU utilization and application bus bandwidth were calculated from eq. 19. For messages sizes above 100 bytes setup time is negligible and the CPU% is given by a the ratio of application level communication to bus band widths independent of message size. An independent measure of socket transfer bandwidth using two tasks on a single machine verified the 11 Mbyte per second transfer rate between main memory address spaces of two executing processes using the socket read/write calls.

4) Cluster Execution Analysis

The formula developed in section 3 allow us to estimate the communication overhead introduced by message passing SND/RCV calls as functions of application data volumes, average message sizes, and application independent communication parameters. We are therefore in a position to estimate the run time of an application in a PC Cluster given only the sequential run time (RTEX[J,J], the sequential CPU utilization (CPU%[J,J]), and the inter calculation data transfers characterized by DVOL[J,S] and #MSG[J,S] as defined in Figure 2. To do so, however, requires a further examination of the data dependencies between calculations in order to determine which, if any, calculations can be run in parallel.

In general calculations can only be run in parallel if they are data independent. If in figure 2 the DVOL[J,S] = 0 between two calculations J and S they do not transfer data and each calculation can proceed independently. This case applies to the ray trace calculations j=1 to N-1 in our test application. A second category if data independence occurs when data is transferred between two calculations however the arrival time is not critical. This requires data transfer resources to be accounted for, but the exact order of such transfers are not critical. The terrain data server J=0 in out test application falls in this category. Terrain data is required by the ray trace calculation but, since these calculations are written to utilize low resolution background data when more recent high resolution data is not available, the only penalty for late data arrival is a degradation of picture quality. Since this happens only when the eye point is moving at high velocity the degradation is interpreted by the operator as an image blur which is expected from high speed motion and thus considered acceptable in the simulation. From an analytic point of view if we do not care when the data arrives we can rearrange the location of the SND and RCV functions for such calculations since the strict time order of execution is not required.

Data dependency considerations allows the sequential program matrix from figure 2 to be rearranged into a parallel execution schedule as shown in figure 5. Note the jump from a sequential matrix to a parallel schedule is application data dependent and requires the ingenuity of a program analyst on a case by case basis. What we show in figure 5 are three calculation types. The terrain data server, J=0, broadcasts its data to the ray tracers but otherwise runs continuously and in parallel with the other processes. The ray tracers can all run in parallel but their output is required input for the display process, J = N, which must therefore be run sequentially to them.

In calculating the resource parameters shown in figure 5 the RTEX and CPU% for the raytrace and display calculations were taken directly from the sequential characteristics given in section 1.1. The terrain data server executes in whatever time is available since it its high resolution output is required on an as available basis explained above. The CPU utilization for the SND and RCV functions are all identical at 39% assuming a LAM/MPI environment. While there are three RTEX values calculated from the bandwidth measurements (see Table 1) and eq. 18. The results for a 256x256 pixel image size are listed in tabel 3 below.

Note the communication load represented on the output is dependent on the frame rate per second (FPS) while the communication load from the terrain data server depends on the motion of the eye point and is independent of the frame rate. Figure 5 below summarizes the program resource characteristics when our test program is divided to execute in N processors.

The numerical value along the bottom row shows the run-time in each time slice which can not be executed in parallel. These numbers are no longer the sum of all the run time values of each sequential calculation as given by eq. 1 but rather the sum of the maximum run time in each sequential time slice. For the four time slices shown above the formula is,
eq. 20 .

The 0.25 seconds for terrain data transfer is true each second of run-time. Hence at RTEX =1 the formula gives the relation between FPS and the number of processors. The maximum frame rate for an infinite number of processors is 3.75 frames per second. This is not great. The main culprit is the communication load represented by the 0.16*FPS term. The higher the frame rate the more data must be sent to the display processor and the longer it will take. The communications load therefore severely limits the effectiveness of the cluster approach.

In the Transputer architecture multiple hardware links are utilized to eliminate the bottle neck on the display processor. To test this approach in PC hardware we installed several communications cards and ran multiple TCP/IP communication treads. Typical A2ABW(MB/sec)results are shown in table 4.

At a message size of 256 bytes this represents a 20% improvement in run time execution at the cost of 100% increases in CPU cycles. Hence a custom socket coded 8 processor system using 6 communications receive threads would change the 0.16 *FPS term in eq. 20 to.1*FPS and generate 3.1 FPS while a infinite number of processors would give only 5.3 FPS.

Given the level of code characterization presented in section 1.1 for our battlefield visualization test case the Cluster analysis would end here. We cannot improve upon the schedule shown in figure 5 or the performance predictions of eq. 20 within the confines of the data provided in section 2. Only by appealing to additional detailed knowledge of the code

internals, and breaking the code down into finer grained calculations can further run time improvements be made. Using such knowledge is a little like pulling rabbits out of a hat since the reader has no means to judge the truth of our code specific assertions. We feel however that such code specific improvements will inevitable become part of a Cluster port analysis and include a few words here as an example.

4.1 Test Case Specific Improvements

Examination of the test case code shows that the SND, RCV for the raytrace processors can be run in parallel rather than sequentially as shown in figure 5. This is because the terrain data dependency has been programmed to contain a low resolution fall-back algorithm and the output can be sent more frequently than at the end of the calculation indicated by the sequential schedule. A simple formula for combining calculations which can be run in parallel (S_P) and those which must be run sequentially (Ss) in one machine can be derived by assuming the parallel programs will fully utilized the machine. RTEX is then equal to,

eq 21 .

This formula holds in our case since the raytrace calculation is always available to take up spare cycles relinquished by other processes in the sum and the context switching overhead can be neglected. If the three raytrace functions in processors 1 to N-1 are run as parallel threads The formula for the run time using FPs as independent variable is,

eq. 22a .

Here N is the number of processors in the parallel configuration. There are at least three processors. FPS is the frame rate per second. The first term is the CPUS for receiving terrain data. Here the 0.39 is the CPU% for the receive function. The second is the ray trace calculation. The third term is the communication load to the display processor. All three are run in parallel.

We can also run two communication cards with multiple threads of the display RCV function speeding up this component by 20% according to measurements reported in table 4. By applying eq. 21, the display processes RTEX estimate is,

eq. 22b .

The 0.04*FPS is the X-window display time. While the second term denotes the receive communications load. Note the formula is written explicitly using A2ABW(MSGZ) as a parameter in order to account for internal buffering into messages larger than the 256 bytes originally specified. This requires code changes and restricts the number of parallel processors to (2^{16} = MSGZ*(Nmax -2)) - a code specific effect. The expected frames per second is calculated as the minimum of eq. 22 a and b. The result is plotted in figure 6a for the candidate Cluster configuration shown in figure 6b.

The Cluster configuration diagram shows three networks. One to broadcast data to the ray trace processors and two to carry the resultant image date to the display processor.

The shape of the performance estimates in figure 6a is quite typical. Initially the performance dips due to the increased overhead of explicit send and receive functions added to the code. After adding 4 processors the performance improves indicating a good parallelization of calculations has been found. Lastly the effect of the communication bottleneck in the display processor limits the otherwise linear speed up in the ray trace processors.

5) Conclusion

We have described a general method for predicting the performance of software in a PC cluster. The technique assumes the run time, cpu utilization, and data communications volumes are known and provides formula for calculating program performance in a distributed message passing environment. The method concentrates on application to application communication performance. It provides simple expressions for message passing overhead in terms of a few easily measured communications parameters. Comparison between theory and measured communication characteristics appears quite good in the test case measured.

We also applied this technique to analyze the benefit of porting an existing battle field visualization application to a networked cluster. This exercise serves both as a test case for the analysis technique and provides guidance to program development questions for our project.

The conclusion for our test case, summarized in figure 6, shows that after an initial performance drop to accommodate the message passing overhead a linear performance increase is expected by increasing the number of processors until a display bottleneck is reached. We estimate that by utilizing eight processors, modifying code to utilize message passing subroutines, and optimizing message structures for a networked environment we would achieve 8.7 FPS or approximately 7 times speed up over a single node sequential implementation.

The question of whether or not to actually build a cluster implementation now becomes a price, packaging, and maintenance issue. We believe better performance can also be achieved by buying a faster single machine rather than expand into a cluster. We have therefore purchased a dual Pentium 200MHZ Pro machine to explore whether the predicted 4 to 5 fold speed up estimated by eq. 7 can be achieved with less cost, maintenance, and storage overhead.

For our project Cluster configurations built from PC components are a viable alternative for achieving speed increases. This conclusion is specific to the characteristics of our software task. The relatively large message sizes inherent in our project minemize the communication penalties associated with message passing implementations. Other applications, with smaller message traffic may perform better. The importance here is not the specifics of our project but that message passing implementations come with significant software performance penalties which may eliminate speed advantage one hopes to achieve. Quantitative analysis on a case by case basis is required. This paper has collected both the formula for such analysis and showed how they are used to perform software design trade-offs and optimize Cluster configurations.

ACKNOWLEDGEMENT
The authors gratefully acknowledge the support of the following agencies:
US ARMY Texcom Experimentation Center, Fort Hunter Liggett, Jolon, CA 93928
TRADOC Analysis Command-Monterey,Naval Postgraduate School, Monterey, CA 93943

REFERENCES:

1)J. Hennessy;D. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann Publishers, Inc., San Mateo, 1990,(see chapter 7)

2)Y. Yan; X Zhang; Y. Song, An Effective and Practical Performance Prediction Model for Parallel Computing in Nondedicated Heterogeneous NOW, J. of Parallel and Distributed Computing 38,63-80(1996)

3)Z. Xu, X. Zhang; L. Sun, Semi-empirical Multiprocessor Performance Predictions, J. of Parallel and Distributed Computing 39,14-28(1996)

4) G. F. Pfister, In Search of Clusters: The Coming Battle in Lowly Parallel Computing, Prentice Hall, New Jersey, 1995

5) PVM Parallel Virtual Machine

6) W. Gropp; E. Lusk; A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, The MIT Press, 1996

7)J. Shirley;Hu Wei; D. Magid, Guide to Writing DCE Applications, O'Reilly&Associates, Inc., Sebastapol, 1994

8) W. Baer, *Scalable Parallel Processing System for Video Realistic Real-time Battlefield Simulation*, Proceedings of the 1994 Simulation Multiconference, Hyatt Regency Aventine, La Jolla San Diego, California, April 11 - 13, 1994, p. 77, Edited by Michael J. Chinni U.S. Army AMCCOM-ARDEC

9) W. Baer ,*Implementation of a Perspective View Generator* Transputing '91, P. Welch et al. Vol. 2, pp 643-656, ISOPRESS Amsterdam, 1991

10) SPEC, Better Benchmarks, Standard Performance Evaluation Corporation, Manassas, 1997

11) S. W. Decato , Parallel Processing Performance Evaluation of Mixed T10/T100 Ethernet Topologies on Linux Pentium Systems, Thesis , Naval Postgraduate School, Monterey Ca. March 1996

12)W. R. Stevens, Unix Network Programming , Prentice Hall 1992