Performance Considerations for I/O-Dominant Applicationson Parallel Computers

Anatholy F. Dedkov and Douglas J. Eadline,
Paralogic, Inc. 115 Research Dr. Bethlehem PA 18015 USA
(610) 861-6960 (voice) (610) 861-8247 (fax)
deadline@plogic.com adedkov@plogic.com

Key Words: Parallel, I/O, Performance, Cache, Simulation

Abstract: A general law is proposed that states "a large numbers of slower processors may be better than a small number of faster processors for I/O-dominant applications". The need for such a guideline is demonstrated because simple linear sums of individual processor performances do not provide an accurate estimation of I/O performance for a parallel computer. Furthermore, the law was formulated to allow better cost estimations when choosing the number and type processor for a Massively Parallel (MP) I/O application. The law is confirmed with a simple proof, analytical model, simulation, and benchmarking. A Distributed Cache Subsystem (DCS) technique is proposed to further improve the performance of the MP computers running I/O-dominant applications. Using simulations and benchmarks the DCS technique has shown the potential to achieve very high performance using standard sequential file systems. The general conclusion proposed is that a MP computer containing a small number of high-performance processors may not be the best choice for I/O-dominant applications area.

1. Introduction

Historically massively parallel (MP) computing has been successfully used for scientific, i.e. CPU-dominant, applications where MP computers play the role of "number crunchers". Although it was observed that many scientific applications have very large I/O requirements [1], the I/O subsystems were considered as auxiliary devices whose main goal is to load processing units with the sufficient input data. In recent years, decision support systems based on very large legacy database processing [2] have entered the mainstream commercial market and are expected to show continued growth. This class of applications has both huge I/O requirements and large CPU requirements. We call these applications I/O-dominant. Because these new MP applications require both CPU and I/O performance, a simple cumulative sum of individual processor performance is not an effective method to determine peak performance capabilities. This research attempts to clarify the evaluation criteria that should be used to evaluate I/O-dominant MP systems and to suggest one possible way to satisfy enormous I/O requirements without expensive high-performance hardware or parallel file systems.

2. Performance evaluation

We consider typical I/O-dominant behavior to be very close to the general model of most business applications where the application periodically reads a data block from (or writes the data block to) the storage devices and then spends some time processing the block. The average time needed to perform one I/O operation is **Tio**. The average amount of processing needed to process one data block is **Tcpu**. Actually **Tcpu** has the "millions of machine instructions" dimension, but we use "time" dimension to normalize processor speed. For simplicity we assume **Tcpu** is one for the slowest processor. Other processors performances are normalized to this processors performance using the undimensioned value called **Sp**.

We consider the ideal MP computer consists of **Np** processing nodes, **Nio** I/O nodes operating in parallel and an interprocessor communication (IPC) network operating with the average latency time **Ttr**. The overall parallel

computer's performance index widely used in the MP scientific computing is the sum of all processing nodes' performances

$$\mathbf{Ptotal = Np * Sp} \qquad \mathbf{(1)}$$

In the scientific computing field, the parallel computer having a larger overall performance index is considered to be more powerful. We demonstrate below that this statement is not appropriate for I/O-dominant applications.

For the I/O-dominant application, the natural performance index **P** should be considered to be the number of data blocks being processed in one time unit. A simple evaluation of the performance index **P** follows. Considering the single processor computer

$$\mathbf{P = \frac{1}{Tio + Tcpu/Sp}} \qquad \mathbf{(2)}$$

Considering the MP computer and assuming all jobs to be ideally distributed between processing nodes we may estimate the performance as follows.

a. **Ttr** does not depend on the processor speed

$$\mathbf{P = \frac{Np}{Tio + Ttr + Tcpu/Sp}} \qquad \mathbf{(3a)}$$

b. **Ttr** decreases proportionally to **Sp**. (i.e. the IPC network transmission rate increases with increasing processor speed).

$$\mathbf{P = \frac{Np}{Tio + Ttr/Sp + Tcpu/Sp}} \qquad \mathbf{(3b)}$$

In the (3a and b) we also assume that I/O capabilities of the MP computer are sufficient to serve each I/O request with no additional delays caused by I/O queues. This assumption allows us to add individual performances of all processing nodes. This assumption and formula (3) are an oversimplification, but it allows us to demonstrate an important observation.

Consider the performances of two MP computers with the same overall performance index (1).

$$\mathbf{Ptotal1 = Np1 * Sp1}$$
$$\mathbf{Ptotal2 = Np2 * Sp2}$$
$$\mathbf{Ptotal1 = Ptotal2}$$

Attempting to compare the performances of these two MP computers on I/O-dominant applications we should estimate **P1**-**P2**, where **P1** and **P2** are calculated according to (3a).

$$\mathbf{P1 - P2 = \frac{Np1}{Tio + Ttr + Tcpu/Sp1} - \frac{Np2}{Tio + Ttr + Tcpu/Sp2} =} \qquad \mathbf{(4)}$$

$$\mathbf{= \frac{Np1 * Sp1}{Sp1*(Tio+ Ttr)+Tcpu} - \frac{Np2 * Sp2}{Sp2*(Tio+ Ttr)+Tcpu} =}$$

$$\mathbf{= Ptotal * \frac{(Tio+Ttr) * (Sp2-Sp1)}{(Sp1*(Tio+ Ttr)+Tcpu) * (Sp2*(Tio+ Ttr)+Tcpu)}}$$

Considering the difference (**P1**-**P2**), which if positive indicates better performance, we see that its sign depends only on the sign of (**Sp2**-**Sp1**) as all other terms are positive. This result means that **P1** is greater than **P2** when **Sp1** is less than **Sp2** and vice versa. In addition, **P1** is equal to **P2** if and only if **Sp1** is equal to **Sp2** (**Np1** is equal to **Np2** respectively). Using (3b) we could obtain the same conclusion. Therefore, taking into consideration the underlying assumptions stated above we may formulate the rule:

For the two given parallel computers with the same cumulative CPU performance index, the one which has slower processors (and a probably correspondingly slower interprocessor communication network) has better performance for the I/O-dominant applications (while all other conditions are the same except the number of processors and their speed).

The assumption of sufficient I/O power might seem unrealistic because an I/O-dominant workload will cause overload conditions on I/O nodes. In order to take this into consideration, we developed more complex analytical and simulation models instead of the simplest mean-value analysis above.

3. An analytical model

To account for the limited I/O capabilities of the real MP computer we use the well-known central server model with N-channel server [3]. The workload is described as above with **Tio** and **Tcpu**. An additional assumption is that there are only **Nio** I/O nodes in the MP computer.

Using Scherr's [3] model we can compute probabilities $\mathbf{p_i}$, where $\mathbf{p_i}$ is the probability of **i** I/O nodes being busy. The utilization coefficient of I/O nodes can then be computed. The performance of the MP computer under the given workload will be proportional to this coefficient. We normalize the computed performance by the single processor computer performance (See Table 1, row (1,64)). The data in Table 1 demonstrate the results computed on Scherr's model for the different workloads and different pairs (**Np,Sp**). For all pairs **Np*****Sp**=Const. **Nio**=8. We also assume that each processing node cannot perform I/O operations in parallel, i.e., only one I/O node is involved in each I/O operation. Accounting of the IPC network latency is done by summarizing **Tcpu** and **Ttr**, i.e. we consider the case when faster processors are connected with an accordingly faster network. We assumed **Ttr**=1, i.e., the slowest processors transmission time is equal to the I/O time.

We see that for CPU dominant workload (**Tcpu**>=100) the performances are approximately the same for all values (**Np**,**Sp**) while for I/O-dominant workloads a larger numbers of slower processors yield better performance. These results confirm that the rule presented above is correct even for MP computers with limited I/O capabilities. The fact that 64 processors provide better performance than four processors is obvious, but the fact that these 64 processors could be 16 times less powerful, and be connected with a 16 times slower network is not fully evident.

Another interesting question which we could answer using this simple analytical model is "What number of slower processors could provide the same performance as the given n number of fast processors?". To answer this question, assume we have four processors in the MP computer. Each processor is of speed 32. **Nio**=4, i.e. I/O capabilities are sufficient to serve I/O requests with no queues. **Tio**=1, as before. The computer with slower processors has the slower IPC network.

Table 1. The estimated relative performances for MP computers with eight I/O nodes.

	Tcpu					
(Np,Sp)	**0.01**	**0.1**	**1**	**10**	**100**	**1000**
(1,64)	1.00	1.00	1.00	1.00	1.00	1.00
(2,32)	2.00	1.99	1.97	1.76	1.24	1.03
(4,16)	3.98	3.96	3.81	2.84	1.41	1.05
(8, 8)	7.91	7.83	7.15	4.09	1.52	1.06
(16, 4)	8.01	8.02	8.14	5.24	1.58	1.06
(32, 2)	8.01	8.02	8.14	6.09	1.61	1.06
(64, 1)	8.01	8.02	8.14	6.62	1.62	1.06

Note! These are the relative performances. They are normalized in each column by different values so different columns must not be compared with each other.

We see from Table 2 that for highly I/O-dominant workloads the performance is almost insensitive to the processor speed and the transmission rate of the IPC network.

The analytical model has two underlying assumptions that may not be very realistic. First, when some processor needs to perform an I/O operation it chooses *any* free I/O node. Second, the IPC network latency time is assumed to be constant. In fact, the contention between processors could cause some excessive delays in the interprocessor links. Besides, more numerous IPC networks require, in general, more messages retransmissions to deliver the message to the destination. To eliminate these assumptions, a simulation model was developed.

Table 2. The amount of slower processors required to provide the same performance as a four node MP computer with processor speed 16 and four I/O nodes.

	Tcpu					
Sp	**0.01**	**0.1**	**1**	**10**	**100**	**1000**
16	4	4	4	4	4	4
8	5	5	5	6	8	8
4	6	6	6	9	15	16
2	7	7	8	16	28	32
1	9	10	12	29	56	63

4. Simulation Model

In addition to the described behavior of the analytical model, the simulation model includes the following new assumptions:

a. The I/O request is directed to the given I/O node. If the I/O node is busy the request will stay in the queue even when other I/O nodes are free.

b. The IPC network with the hypercube topology is simulated with sufficient accuracy. Real algorithms of message delivery are employed.

c. The simulation model allows more than one compute process to run on a single processing node. The influence of multiprogramming to the performance could then be studied.

d. Other features of the simulation model are explained later in the paper.

The simulation run length is 100,000 I/O operations. The performance is estimated based on the simulated time required to perform these operations. Table 3 shows the simulation results for the same cases as Table 1. Simulation shows that the analytical model yields overstated performance estimations for the I/O-dominant workload (**Tcpu**<100). This difference is due to inclusion of features **a**. and **b.** above that were not previously considered in the analytical model.

The simulation results confirm the general results of the analytical model and support the proposed rule. The next question we attempt to answer is "How to improve the parallel computer's performance for an I/O-dominant workload?".

Table 3. The simulated relative performances for MP computers with eight I/O nodes.

		Tcpu					
(Np,Sp)	**Ttr**	0.01	0.1	1	10	100	1000
(1,64)	0.016	1.00	1.00	1.00	1.00	1.00	1.00
(2,32)	0.031	1.74	1.74	1.73	1.61	1.23	1.03
(4,16)	0.063	2.82	2.78	2.74	2.38	1.40	1.05
(8, 8)	0.125	4.05	4.07	3.96	3.22	1.50	1.06
(16, 4)	0.250	5.40	5.33	5.20	4.00	1.56	1.06
(32, 2)	0.500	6.38	6.31	6.26	4.62	1.59	1.06
(64, 1)	1.000	6.94	6.82	6.75	4.73	1.54	1.03

Note! These are the relative performances. They are normalized in each column by different values so different columns must not be compared with each other.

5. Approaches to Improve Performance

As we consider equation (3) to be a rough but appropriate estimation of the parallel computer's performance under the I/O-dominant workload, we can reach some general conclusions on about additional performance enhancement. Because most application developers have no direct ability to alter or improve parallel hardware, methods for improved performance will focus on issues under the control of the application developer. Considering equation (3) the following methods can be suggested:

a. To increase **Np**. **Np** is usually limited with the parallel hardware being employed. This factor is only under control when choosing the parallel hardware. The consideration above shows that it would be more advantageous to use larger values of **Np** even when processors are not very fast.
b. To increase **Sp**. As was shown above increasing of **Sp** brings very small benefit to the overall performance on I/O-dominant applications. If, for instance **Tio**=1 and **Tp**=1, then increasing **Sp** *ad infinitum* could only double the performance.
c. To decrease **Tp**. Since **Tp** (the number of machine instructions needed to process one data block) is the property of the given application it could be decreased using conventional programming methods (advanced programming techniques, optimization, etc.). We don't consider these ways in the paper.
d. To decrease **Tio**. This way seems to be most appropriate. There are three possible ways to decrease **Tio**.
    1. To use faster (and more expensive) storage devices.
    2. To use sophisticated parallel file systems [4]. Again, this is often under control of the parallel hardware vendors and not a direct solution available to the application developer.
    3. To use the Distributed Cache Subsystem (DCS) technique described below which allows significant reductions of I/O delays. This method is not dependent on hardware and is potentially applicable by any application developer.

6. Distributed Cache Subsystem (DCS)

We consider below the kinds of parallel applications having the following properties:

a. The application is I/O-dominant.
b. The application processes very large external databases.
c. The application requires multiple accesses to some (or all) of the databases data blocks (Note that when each block is accessed once only, **any** caching scheme is not efficient).

The conventional well-known way to decrease the time of I/O operations is disk caching. Both I/O nodes and processing nodes are usually equipped with their proprietary disk caches which keep most recently used data blocks. The concept of Distributed Cache Subsystem (DCS) being introduced is based on dedicating some part of the processing nodes to serve as specialized memory pools (DCS nodes). Their only goal is to serve I/O requests from the processing nodes and store most recently used data blocks. We assume that images of the cached files are stripped between DCS nodes, i.e. first data block of each file is kept on first DCS node, the second data block is on the second DCS node, etc. So all DCS nodes are coordinated this way and there is no data duplication as may occur in the local caches of the processing nodes. When some processing node is required to read a data block it determines what DCS node holds the date and sends an I/O request to the given DCS node. All DCS nodes act as passive servers; they wait for the I/O requests. Receiving the request, the DCS node determines if the required data block is in its cache. If it is, it sends the data block to the requester. If it is not, it reads the data block from the I/O node, sends it to the requester and stores it in the cache. The DCS is one more intermediate layer between processing nodes and I/O nodes as shown in Figure 1

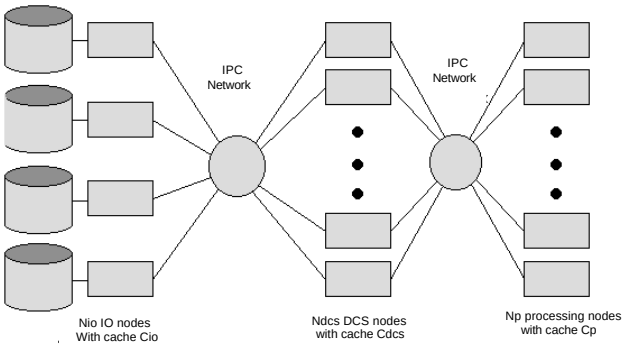

Figure 1. The general overview of MP computer with DCS nodes.

At the first glance, there appears to be no benefits of the DCS if we could allocate large amounts of each processing nodes memory for a local disk cache. To test this idea, consider a simple analysis of this problem. Assume there is some large database distributed uniformly between I/O nodes. The data volume is **Vd**. Let's

estimate the mean time, **Tacc**, needed to access the data block. We consider the stationary condition when all caches are saturated already.

Case I. No DCS

$$Tacc = P1*0 + (1 - P1) * (P2*Ttr + (1 - P2)*(Tio + Ttr)) \quad (5)$$

where:

P1 is the probability that data is found in the local cache (access time is zero)

P2 is the probability that data is found in the I/O node cache (access time is the transfer time only)

(1-P2) is the probability that data not found in the I/O node cache (access time is the time to perform the I/O operation and the transfer time)

If we consider the worst case (data requests are distributed amongst the database as uniformly distributed random numbers) then we may estimate these probabilities as

P1 = Cp / Vd ,where **Cp** is the processing nodes local cache capacity
P2 = Cio / (Vd/Nio) , as total data volume is distributed between I/O nodes, **Cio** is the I/O nodes cache capacity

Case II. DCS

$$Tacc = P1*0 + (1 - P1) * (P4*Ttr + (1-P4) *(P2*Ttr + (1 - P2)*(Tio + Ttr))) \quad (6)$$

where

P4 is the probability that data is found in the Distributed Cache. As there is no data duplication, we could estimate

$$P4 = \min(1 , Ndcs * Cdcs / Vd) \quad (7)$$

We see that in the conventional case **Tacc** (5) is not influenced by the number of processing nodes. Even if we have a large number of processing nodes with the large local caches it will not reduce **Tacc**. Increasing the number of I/O nodes decreases **Tacc**, however, as it would decrease the probability **P2**. Unfortunately this factor is out of the application programmers control.

In case of DCS, the opposite is true. We could decrease **Tacc** by increasing **Ndcs**. When **Ndcs * Cdcs > Vd** the probability **P4** will be equal to one, so all I/O activity will be eliminated. This condition results when the total database is placed into the DCS caches. Even if this condition is not reached, DCS could decrease Tacc significantly. Consider the following example. Assume some artificial values; **Vd** = 4000 MB, **Nio** = 4, **Cio** = 10 MB, **Np** =512, **Cp**= 4 MB, **Tio** = 1, **Ttr** = 0.1, **Ndcs**= 100, **Cdcs**=20MB, then we have **Tacc** = 1.089 for the no DCS case and **Tacc** = 0.594 for the DCS case. We see that in the no DCS case local disk caching is almost useless as **Tacc** is almost equal to **Tio+Ttr**, while DCS could decrease **Tacc** about twice. This condition is true for the very large databases. For small databases, the database can be efficiently cached in I/O node caches so the dedicated DCS nodes are almost useless. For instance, let **Vd** = 40 MB. We have **Tacc** = 0.09 for both cases (DCS and no DCS).

Based on the above analysis, it can be concluded that the DCS technique is efficient for a parallel I/O-dominant application processing very large databases on MP computers with large number of nodes, as the number of DCS nodes has to be large enough to hold a significant part of the database being processed. Another case when the DCS technique could be very efficient is when the MP computer does not have a high-performance parallel I/O subsystem. For instance, if all I/O is performed with the single workstation-class host computer, the DCS technique is applicable even for databases of moderate and small size.

The estimations of the data access times above are very rough. We assumed the IPC network latency time is constant while this is often not the case. Indeed, the IPC network is often a bottleneck preventing performance increases. Furthermore, when we employ the DCS technique the DCS nodes play the role of additional " I/O nodes". Using a large number of DCS nodes (**Ndcs** instead of the small number of I/O nodes) we can increase the overall data throughput. In general, the overall data throughput between I/O nodes (DCS nodes) and the processing nodes is

*Overall data throughput = number of I/O (DCS) nodes * throughput of one inter processor link*

The advantage of the DCS is that we could use an optimal number of DCS nodes to achieve the maximal data throughput instead of using the limited number of I/O nodes (or one host computer, in the worst case). The advantage offered by a DCS also depends on the individual application and the characteristics of the target MP computer.

When we decide to employ the DCS technique the problem to be solved is: "What is the optimal number of DCS nodes?". Indeed, when the number of DCS nodes is small the DCS is not efficient enough as it cannot hold enough of the database. Besides, long request queues could result when the number of DCS nodes is small. When the number of DCS nodes is large it processes I/O requests efficiently but the remaining nodes cannot provide enough processing power. The problem is that by increasing the number of DCS nodes we decrease the number of actual processing nodes. Therefore, an optimal number of DCS nodes must exist. The optimal proportion between DCS nodes and processing nodes could be found using the simulation methods or experiments with the real application.

7. DCS simulations

To study the areas of DCS applicability and to demonstrate the benefits of DCS technique we have used the simulation model described above. Although it was not mentioned, this model can simulate DCS and consider the sizes of caches (both processing nodes local caches and I/O nodes caches).

We consider the following assumptions. The MP computer has four parallel I/O nodes each with 16MB. Each DCS node (if any) has 16MB cache. Each compute node has 2MB local cache. Database sizes are 10MB to 1,000 MB. Data transfer time in interprocessor link is constant **Ttr**=0.1. The performances are normalized to the single processor computer's performance row (1,64). The numbers of the DCS nodes were chosen to be quasi-optimal using the analytical model. Actual performances of the DCS could be slightly higher than shown. As we would like to obtain preliminary estimations only, the actual achievable performances are of no importance for our consideration. The simulation results are in Table 4. The numbers of DCS nodes used in simulations are shown in Table 5.

Table 4. Comparison of relative performances for parallel MP computers with four parallel I/O nodes.

Database size =1,000MB

	Tp=0.01		**Tp**=0.1		**Tp**=1	
(Np,Sp)	**no DCS**	**DCS**	**no DCS**	**DCS**	**no DCS**	**DCS**
(1,64)	1.00	-	1.00	-	1.00	-
(2,32)	1.70	1.04	1.70	1.04	1.68	1.04
(4,16)	2.55	1.46	2.55	1.46	2.55	1.17
(8, 8)	3.41	2.55	3.42	2.42	3.44	1.17
(16, 4)	3.97	4.93	4.00	4.74	4.05	2.22
(32, 2)	4.44	10.53	4.46	9.40	4.38	4.34
(64, 1)	4.56	26.43	4.61	21.07	4.68	7.93

Database size =100MB

	Tp=0.01		**Tp**=0.1		**Tp**=1	
(Np,Sp)	**no DCS**	**DCS**	**no DCS**	**DCS**	**no DCS**	**DCS**
(1,64)	1.00	-	1.00	-	1.00	-
(2,32)	1.67	0.61	1.67	0.60	1.58	0.59
(4,16)	2.47	0.97	2.51	0.98	2.34	0.68
(8, 8)	3.39	2.48	3.42	1.93	3.29	0.69
(16, 4)	4.29	16.35	4.32	12.42	4.24	1.52
(32, 2)	5.08	30.02	5.06	33.65	4.90	4.90
(64, 1)	5.59	52.76	5.54	52.98	5.54	24.27

	Database size =10MB					
	Tp=0.01		**Tp**=0.1		**Tp**=1	
(Np,Sp)	**no DCS**	**DCS**	**no DCS**	**DCS**	**no DCS**	**DCS**
(1,64)	1.00	-	1.00	-	1.00	-
(2,32)	1.91	1.88	1.89	1.83	1.78	1.50
(4,16)	3.33	2.26	3.28	2.19	2.81	2.86
(8, 8)	6.44	3.34	6.23	4.29	4.46	4.25
(16, 4)	11.31	5.41	10.42	6.49	6.04	5.33
(32, 2)	17.75	9.84	16.29	10.80	7.48	6.50
(64, 1)	22.11	16.02	20.20	16.29	7.96	7.55

Table 5. The numbers of DCS nodes used for simulations in Table 4.

	Ndcs		
Np	**Tp**=0.01	**Tp**=0.1	**Tp**=1
2	1	1	1
4	2	2	1
8	4	3	1
16	8	6	2
32	15	11	4
64	31	22	7

As expected, the DCS yields no benefit for small databases when I/O nodes' caches are sufficient to provide efficient caching. Also, DCS is inefficient for small numbers of nodes. DCS provides valuable speed-up when the number of the DCS nodes is more than the number of I/O nodes and especially when the cumulative volume of the DCS caches is near or larger than the database's size. Nevertheless, we see that DCS could provide a five times speed-up even when the cumulative volume of DCS caches is two times less than the database's size. The effect of DCS is larger for highly I/O-dominant workload (**Tp**=0.01 and **Tp**=0.1) than for the workload with large compute part (**Tp**=1). It is also demonstrated that the behavior of the MP computer with DCS complies to the rule being formulated above.

It seems to be the advantage of the DCS technique that it is employed not as a replacement of the current parallel I/O subsystems but in conjunction with them instead. It allows the application developer to check if the DCS is efficient under the given conditions and to decide whether to employ the DCS.

An interesting case is when there is no I/O subsystem available to the application programmer. This case may result when it is too costly (or impossible) for "Legacy Data" to reside in the file system of the MP. To estimate the effect of DCS for a MP computer with slow non-parallel I/O, we have performed one more suite of simulations (see Table 6). The different assumptions are: all processors are considered to be equal (**Sp**=1). There is only one I/O node with 4MB cache. It is the case when all I/O is performed with the host-machine to which an MP computer is attached.

Table 6. Comparison of relative performances for a parallel MP computer when I/O is performed with the single host.

	Database size =1,000MB					
	Tp=0.01		Tp=0.1		Tp=1	
(Np,Sp)	**no DCS**	**DCS**	**no DCS**	**DCS**	**no DCS**	**DCS**
(1, 1)	1.00	-	1.00	-	1.00	-
(2, 1)	1.10	1.02	1.17	1.00	1.64	1.00
(4, 1)	1.11	1.41	1.20	1.47	2.08	1.94
(8, 1)	1.12	2.48	1.20	2.52	2.11	2.12
(16, 1)	1.11	4.84	1.19	4.86	2.09	3.93
(32, 1)	1.01	10.12	1.21	9.94	2.12	7.92
(64, 1)	1.10	25.36	1.19	22.19	2.11	14.34

Database size =100MB

	Tp=0.01		Tp=0.1		Tp=1	
(Np,Sp)	**no DCS**	**DCS**	**no DCS**	**DCS**	**no DCS**	**DCS**
(1, 1)	1.00	-	1.00	-	1.00	-
(2, 1)	1.09	1.12	1.17	1.11	1.64	1.06
(4, 1)	1.10	1.79	1.20	1.63	2.09	2.14
(8, 1)	1.10	4.59	1.21	1.85	2.18	2.48
(16, 1)	1.12	29.87	1.21	22.78	2.17	5.35
(32, 1)	1.12	55.06	1.22	59.87	2.16	16.49
(64, 1)	1.11	98.17	1.33	106.69	2.15	87.72

(Table 6. Continued)

Database size =10MB

	Tp=0.01		**Tp**=0.1		**Tp**=1	
(Np,Sp)	**no DCS**	**DCS**	**no DCS**	**DCS**	**no DCS**	**DCS**
(1, 1)	1.00	-	1.00	-	1.00	-
(2, 1)	1.12	6.32	1.28	3.73	1.76	1.45
(4, 1)	1.17	7.99	1.37	5.86	2.71	4.21
(8, 1)	1.20	12.05	1.40	12.94	3.23	9.58
(16, 1)	1.20	19.72	1.39	22.23	3.25	18.59
(32, 1)	1.18	36.16	1.41	41.38	3.25	35.82
(64, 1)	1.21	60.30	1.42	72.08	3.24	70.76

As expected we see that the MP computer with the I/O being performed with the single host-machine is absolutely inappropriate to carry the I/O-dominant workloads. The same computer being equipped with the DCS could provide highly efficient execution of the I/O-dominant applications. Actually, the estimations in Table 6 are overstated as we don't consider the initial time needed to saturate disk caches. This time might be significant while using the slow I/O node.

An interesting effect is that 64 processors could provide more than 64 times speed-up. This effect is due to having a sufficient number of DCS nodes to eliminate any actual I/O activity compared to the (1,1) case where I/O activity with **Tio**=1 is high.

8. Additional ways to improve the DCS performance

If the MP computer nodes are equipped with large enough memories then there are two additional approaches to improve performance of the MP computer with DCS.

1. If the parallel I/O-dominant application does not use the all nodes memory then conventional multiprogramming techniques could be employed to improve the performance. Two or more instances of the application (compute processes) run on the same compute node. As a compute node is usually under-loaded these additional processes could share the processor efficiently. This technique has been employed for at least thirty years for conventional computers but is employed rarely in parallel computers as it is inefficient for CPU dominant scientific applications. Due to space limitations we do not show these results, however, we have found that for highly I/O-dominant workloads use of multiprogramming could improve the performance about twice.
2. Under the same condition (low memory consumption of the parallel application) another way to improve performance may be employed. We considered the case when DCS runs on the dedicated nodes. If there are sufficient amounts of free memory on compute nodes, then DCS processes could run on compute nodes. So two processes run on the same node: the application process and the DCS process. This will improve the performance as maximal numbers of both DCS and compute nodes are used. The problem of determining the optimal number of DCS nodes is completely eliminated.

For instance, if the MP computer consists of 128 nodes with 16MB of memory each and the application requires 6MB, then 10MB on each node could be allocated for the DCS process. If we use 50 dedicated DCS nodes we have only 78 actual compute nodes and the cumulative DCS cache of size 800MB. Using non-dedicated DCS nodes we have 128 compute nodes and 128 DCS nodes with the cumulative DCS cache of size 1,280MB.

In our models we considered the "low memory" case and did not use the above "ample memory" assumption. It might be expected that this will improve the performance significantly and we expect to study this case further. It is hoped that when considering such cases the analysis presented here will allow decisions to be made on "price-to-performance" arguments and not on unconfirmed assumptions For example, before investing in the machine above (2048 MB of RAM; 16MB per node) the customer should expect at least four times better performance than a machine that uses one quarter as much memory (512MB of RAM; 4MB per node) .

9. Experiments

To further test our results, we developed some benchmarking programs for a parallel computer. The benchmarking was done on a nCUBE-2 massively parallel computer using the special synthetic workload program. This program imitates both the DCS nodes and compute nodes. The different processor speeds were imitated by varying the length of the dummy loop representing the compute part of the workload. The performance was estimated with the real time needed to perform a given number of 2K blocks transmissions between DCS nodes and compute nodes. The predicted performance was estimated using the simulation model with input parameters reflecting the real benchmark program parameters. The total data transfer between DCS nodes and compute nodes was 200MB in each experiment. Knowing the time needed to run the benchmark program we could estimate the actual data throughput. Predicted performance was estimated in Table 4 (Database size=10MB, Tp=0.1) data being normalized by row (4,16). The results in Table 7 confirm the general rule above and show sufficient compliance with the simulation results. These benchmarks demonstrate that our simulation model reflects real system behavior with sufficient accuracy and could be used to predict other system performance.

The described DCS technique has been implemented in Paralogic's n-parallel PROLOG [5]. Several benchmarks have been developed and confirm the results here. Interestingly, hyper speed-up was noted and almost linear scalability was observed. An additional important point is use of a scalable programming language to develop the compute portion of the application. As has been noted, an optimal combination of DCS and compute nodes will result in the best performance. This ratio may vary from application to application. If the application is not scalable (i.e. locked to a specific number of computing nodes), then a compute bottle-neck may result. In this case, n-parallel PROLOG is scalable and provide applications that run on one to "n" processors. By using n-parallel PROLOG programmer responsibilities for load balancing, inter processor communication, data consistency, and the DCS are hidden from the user. On the other hand, using more conventional programming languages to create applications that take advantage of the results presented above would be extremely difficult to create.

Table 7. The experimental results obtained for the synthetic workload benchmark program.

(Np,Sp)	Ndcs	Time (sec)	Normalized perf.	Predicted perf.	Cumulative throughput (MB/sec)	DCS node throughput (MB/sec)
(4,16)	2	97.2	1.0	1.0	2.06	1.03
(8, 8)	3	49.7	2.0	2.0	4.02	1.34
(16, 4)	5	30.2	3.2	3.0	6.62	1.32
(32, 2)	8	20.6	4.7	4.9	9.71	1.21
(64, 1)	10	16.0	6.1	7.4	12.50	1.14

10. Conclusions

1. The proposed general law was formulated to allow better estimations of the performance (speed/cost) of MP computers for I/O-dominant applications.
2. This law is confirmed with the simple proof, analytical model, simulation and benchmarking.
3. The Distributed Cache Subsystem technique is proposed to improve the performance of the MP computer running I/O-dominant applications. It is shown with simulations and benchmarking that this technique could be employed efficiently to achieve high performance for parallel I/O-dominant applications even on MP computers with slow non-parallel I/O.
4. It is shown that MP computer containing a small number of high-performance processors is not the best choice for I/O-dominant applications area. The large array of slow (and inexpensive) processors equipped with the high-performance parallel I/O subsystem and/or Distributed Cache Subsystem could provide a larger absolute performance and better price-to-performance index.

References

[1] Ethan L. Miller and Randy II. Katz. Input/output behavior of supercomputing applications. - In *Proceedings of Supercomputing* '91, pp. 567-576, November 1991

[2] Development in Commercial Parallel Processing. A User Perspective. - Gartner Group Consulting Services, September 1994.

[3] Scherr A. An Analysis of Time-Shared Computer Systems. - MIT Press, Cambridge, MA 1967

[4] Thomas H. Cornen, David Kotz. Integrating Theory and Practice in Parallel File Systems. - Unpublished manuscript. Available at *unix.hensa.ac.uk:/parallel/documents/pario/DAGS-conference/cormen:integrate.ps.Z.*

[5] Douglas J. Eadline. Implementing Prolog on Distributed Systems: n-parallel PROLOG. - In *Proceedings of the ILPS'94 Post-Conference Workshop*. Ithaca, NY, 1994, pp. 130-140, E.Pontelli and G.Gupta, eds.

