A First Look at T3E UNICOS/mk Performance

John Melom, Cray Research, Inc.

ABSTRACT: *Performance characteristics of UNICOS/mk running on T3E's will be presented. Performance data related to single and multi PE applications will be shown along with characteristics of scalability related to I/O, file systems, and PEs.*

Introduction

UNICOS/mk and the Cray T3E provide significant new opportunities for parallelism. This paper will look at ways in which UNICOS/mk system architecture takes advantage of the parallelism available in the T3E. The focus will be on the disk I/O path, since that is a critical factor in operating system performance.

UNICOS/mk Disk I/O Request Path

Figure 1 shows the UNICOS/mk disk I/O request path. In the figure, boxes represent individual processing elements (PEs) in a T3E.

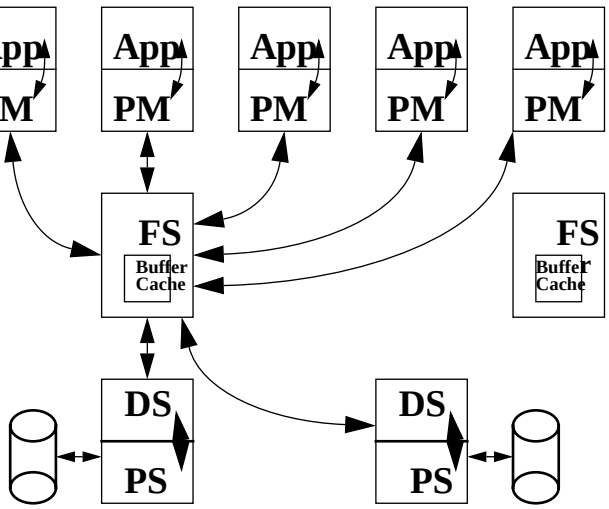

Figure 1: UNICOS/mk Disk I/O Request Path

The top row of boxes represents PEs running user applications (App). This can either be a single multi-PE application, or multiple smaller applications. When a user issues a disk I/O system call such as read() or write(), the UNICOS/mk process manager (PM) interprets the system call and decides what needs to be done with the system call.

The middle row of boxes in figure 1 represents PEs running the UNICOS/mk file server (FS). The FS consists of the UNICOS/mk code that manages file systems. Thus, the file system buffer cache resides on the FS PE and is managed by the FS. The FS processes file related system calls such as open(), close(), read(), write(), lseek(), and others. When the PM receives one of these system calls, it will forward the request to the FS. This is indicated in figure 1 by the arrows connecting the application PEs to the FS PE on the left. Note that in figure 1, two FS PEs are pictured. The capability to have more than one FS PE per system will not be available in the first release of UNICOS/mk, but is planned for a subsequent release. In the first release of UNICOS/mk, all file systems will be managed by a single FS in one PE.

If, in the case of a read() or write() system call, the FS can satisfy the request from the buffer cache, the data will be copied from the buffer cache to memory in the user application's PE. Otherwise, the FS will need to obtain the requested data from disk. The FS will forward an appropriate request to the disk server (DS) to obtain the data.This is shown in figure 1 by the arrows connecting the FS PE to the DS/PS PEs.

The bottom row of boxes in figure 1 represents PEs running the UNICOS/mk disk server (DS) and packet server (PS). The DS manages disks, while the PS is the physical network manager (GigaRing). Note that there can be multiple DS/PS PEs. The DS will make a request to the PS which will make a data transfer request to the disk. The hardware allows direct memory access (DMA). This means the data transfer path will be direct from the disk (or disk channel) to the destination PE. Data transfer will often have a more direct path than the I/O request path. If the I/O request is a buffered I/O request, the data will be read into or written from the FS buffer cache. If the I/O request is a raw I/O request, the data will be read from or written directly to the application PE to or from disk.

This disk I/O path exhibits parallelism at all levels. Multiple application PEs can issue disk I/O requests. A future release of

UNICOS/mk will allow parallelism across file servers. The DS/PS level allows parallelism across disk sets or at the disk slice level within a striped set. Users have access to the DS/PS parallelism through system defined disk striping, or through user defined striping. User defined striping is available in UNICOS/mk through the ASSIGN command just as it is available in UNICOS today.

The path shown in figure 1 does not provide for parallelism at the file server level within a file system. The optimization described in the next section addresses this.

UNICOS/mk Disk I/O Request Optimization

The file server assistant(FSA) was introduced to shorten the disk I/O path, and to remove the necessity of routing all disk I/O requests within a file system through the FS PE. It must be noted that the FSA can only be used for certain types of disk I/O requests. The restrictions will be described later. Figure 2 shows the disk I/O request path for requests that can be processed by the FSA.

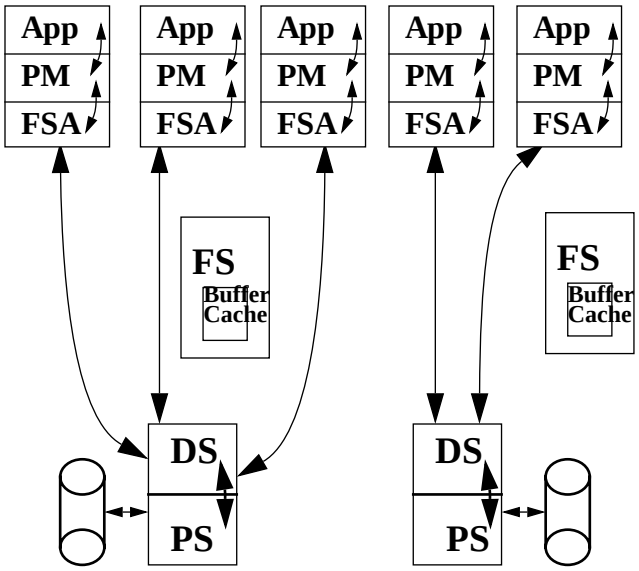

Figure 2: UNICOS/mk Disk I/O Request Optimization

A comparison of figure 1 and figure 2 shows two significant differences. First, the application PEs all have an FSA in addition to a PM. Second, the arrows showing the request flow bypass the FS PE and go directly to the DS/PS PEs.

The disk I/O request path again starts with the user application making a read() or write() system call. The system call is interpreted by the PM. If the request cannot be processed by the FSA, the PM forwards the request to the FS and the path is the same as that described in figure 1. If the disk I/O request can be processed by the FSA, the PM passes the request to the FSA. The FSA does the processing that would otherwise be performed by the FS. The FSA then passes the request to the DS/PS, which will manage the disk I/O request as before. As in the FS I/O path, the DMA data transfer will cause data to be read from or written to the disk directly to or from the application PE's memory.

The FSA cannot be used for buffered I/Os. This makes sense since the FSA I/O path bypasses the FS PE, which contains the buffer cache. The FSA can only be used for raw I/Os. These I/Os must be well-formed. The FSA can process read(), write(), reada(), writea(), listio(), and lseek() requests. These are the highest volume disk I/O requests. As a consequence, parallelism and a shorter path can provide the greatest benefit in terms of overall system throughput. Requests such as open() and close() must be processed by the FS. The file from which I/O is requested must be opened with the O_RAW, O_WELLFORMED, and O_PARALLEL flags specified. FSA usage will also be supported in the Cray libraries in UNICOS/mk.

If an FSA request attempts to extend a file (write() past current end of file), that request is passed to the FS. This detour makes the I/O path for that request longer than the straight FS path. Thus, there is a performance cost for FSA file extension requests. For this reason, an application that does frequent file extensions will probably get the best performance without using the FSA path (O_PARALLEL is not specified when the file is opened). If the ultimate file size is known, the file can be preallocated, so the file extensions will not occur. In this case, the FSA path will likely provide the best performance.

The FSA disk I/O path provides for parallelism at all levels. It applies to those cases where a high disk I/O request rate is most necessary. In addition, the restrictions and user interface are similar to those required by UNICOS today, so it is our belief that the FSA will provide convenient access to highly parallel I/O on the T3E.

Preliminary UNICOS/mk Performance Results

We will consider some preliminary UNICOS/mk performance results in this section. Before attempting to apply these results to any production environment, it is important to keep the following points in mind:

- The tests were run on a T3D running UNICOS/mk. There are no plans to release UNICOS/mk to run on a T3D, so no T3D specific performance optimizations were done in UNICOS/mk. The T3D system was used for internal development purposes while T3E systems were not readily available. One would expect UNICOS/mk running on a T3E with T3E specific optimizations to perform better than the results we will show here.
- The version of UNICOS/mk run on the test system was compiled with symbols and debug on. This is not the compilation optimization level at which UNICOS/mk will be released. One would expect a version of UNICOS/mk compiled for production to perform better than this system.

The test system configuration was limited, so we will not be able to present results for all the interesting cases. We will present results for single stream buffered I/O, multiple stream buffered I/O, and single stream raw I/O. Single stream tests mean that a single application PE is initiating requests. Multiple stream tests mean that multiple application PEs are concurrently issuing requests.

Single Stream Buffered I/O

Figure 3 shows both the I/O request rate and data transfer throughput rate for single stream buffered I/O. The test did 100 synchronous sequential buffered I/O read requests.

Values on the x axis show the number of kilobytes per request for each execution of the test. The left hand y axis provides the scale for the I/O request rate in terms of I/Os per second. The dashed line in the graph shows the measured I/O request rates. The right hand axis provides the scale for the data transfer throughput rate in terms of megabytes per second. The solid line in the graph shows the measured data throughput rate.

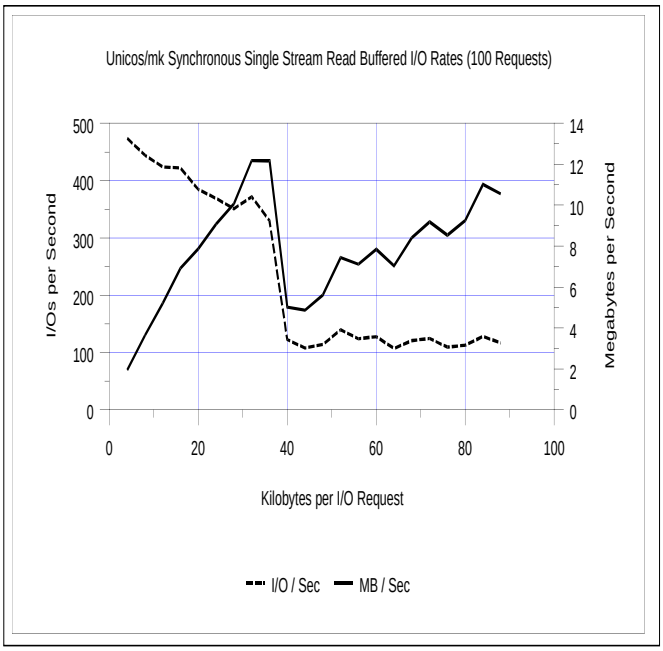

Figure 3: Single Stream Buffered I/O

Because this test is single stream, the data presented in figure 3 does not represent system capacity limits, but rather shows the effects of request latencies.

As you view the chart from left to right, you see the I/O request rate gradually dropping and the data throughput rate increasing. Initially, this single stream test I/O rate is limited by the sum of the CPU processing times in the application PE and the FS PE. As the request size is increased, the data copy time becomes more of a factor. This causes the request rate to slowly decline. Because more data is transferred per request, we see the data transfer throughput rate increase.

Between 30 and 40 kilobytes per request, we see the data transfer throughput rate level off and then dramatically drop off. This is because at these request sizes, 100 requests exceed the size of the buffer cache. Hence, we see the effect of more and more requests having to be satisfied from disk rather than from the buffer cache. After 40 kilobytes per request, the test is dominated by disk I/O, so the request rates level off and the data transfer throughput rates rise as the request size is increased. At high enough transfer sizes, the data transfer throughput rate reaches the disk data transfer limit.

Multiple Stream Buffered I/O

Figure 4 shows the data transfer throughput rate for multiple stream buffered I/O. This test did 5000 synchronous read() requests and 5000 synchronous write() requests interspersed with lseek() requests which caused all requests to do I/O to the same address. This test design avoids buffer cache overflow to allow measurement of pure buffer cache I/O limits. The I/O requests were 16 kilobytes in all cases.

The x axis shows the number of concurrent streams running for a particular test. The y axis shows the measured I/O request rate in terms of I/Os per second.

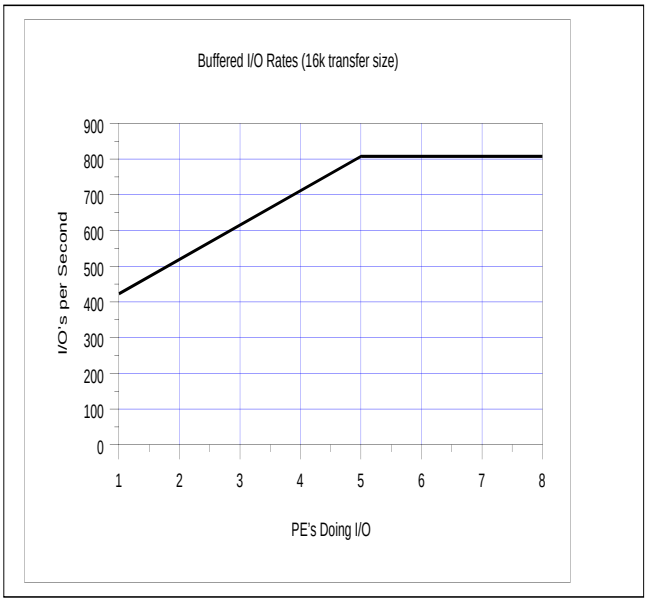

Figure 4: Multiple Stream Buffered I/O Rate Limits

The measured I/O request rate for one stream is approximately 425 requests per second, which matches closely with the I/O request rate seen in figure 3 for 16 kilobyte requests. The I/O request rate peaks at slightly over 800 requests per second at five concurrent streams. At this point, the FS PE CPU is 100% utilized. It is important to note, these results were obtained from a T3D running unoptimized UNICOS/mk.

Single Stream Raw I/O

Figure 5 shows both the I/O request rate and data transfer throughput rate for single stream synchronous raw I/O read requests from a DD60. The test system had the FS, DS, and PS all located on the same PE. This test used the FSA path. However, because the test has a single I/O stream, and because the FS, DS, and PS are located on a single PE, the results would not be significantly different if the standard FS I/O path were used.

Values on the x axis show the number of kilobytes per request for each execution of the test. The left hand y axis provides the scale for the I/O request rate in terms of I/Os per second. The dashed line in the graph shows the measured I/O request rates. The right hand axis provides the scale for the data transfer throughput rate in terms of megabytes per second. The solid line in the graph shows the measured data throughput rate.

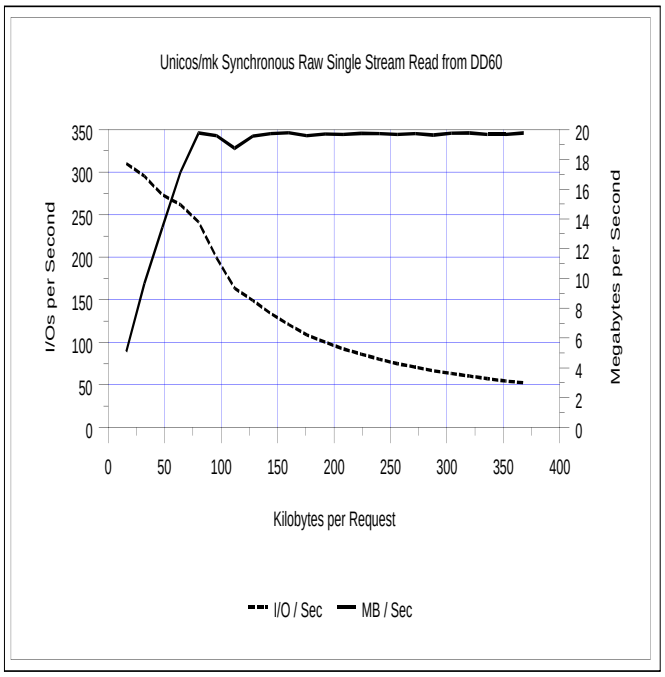

Figure 5: Raw I/O Single Stream Throughput

For small request sizes, the test is limited by the sum of the CPU times of the application PE, the FS/DS/PS PE, and the disk latency and data transfer time. Since this involves more servers than the buffered I/O path plus a disk, it is understandable that the measured raw I/O request rate is less than the measured buffered I/O request rate. It is important to note, this test does not measure the I/O request rate maximum, since it is single stream, not multiple stream. As the data transfer size is increased, the request time is increasingly dominated by disk data transfer time, until the disk throughput limit of 20 megabytes per second is reached. The decreasing I/O request rate is a natural consequence of the data transfer throughput limit. Since there is a fixed data throughput limit, and we are transferring more data per request, the I/O request rate declines.

Summary

In this paper we have shown how the design of UNICOS/mk takes advantage of the parallelism available in the Cray T3E. We have presented results that show even an unoptimized UNICOS/mk running on a T3D can provide high data throughput. During the course of our discussion, we pointed out that users can take advantage of the optimized UNICOS/mk I/O paths through interfaces similar to those they are already using in UNICOS.

Credits

Thanks to Bob Albers and Dave Henseler for providing the data used in this paper.