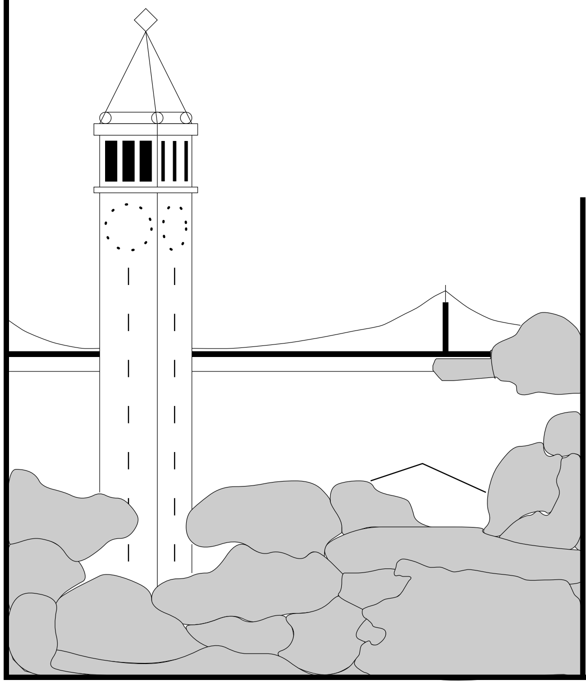
Empirical Evaluation of Global Memory Support on the CRAY-T3D and CRAY-T3E

Arvind Krishnamurthy, David E. Culler, and Katherine Yelick
Computer Science Division
University of California, Berkeley



Report No. UCB/CSD-98-991

January 1998

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Empirical Evaluation of Global Memory Support on the CRAY-T3D and CRAY-T3E *

Arvind Krishnamurthy, David E. Culler, and Katherine Yelick
Computer Science Division
University of California, Berkeley

1 Introduction

Performance prediction on parallel machines is notoriously difficult, especially using designer-supplied machine parameters for features like clock speed, network latency, and network bandwidth. The performance observed by an application programmer is a complicated function of the local memory hierarchy on each node, software overheads from the compiler and operating system, and interactions between components of the machine. As a result, the process of understanding and tuning application level performance is often an ad hoc process, lacking in the kinds of models and tools that are enjoyed by other engineering disciplines. In a previous paper [1], we proposed a "gray-box approach" for measuring machine performance through the use of micro-benchmarks, and applied it to the problem of compiling a global address space language on the Cray T3D. In this paper, we use micro-benchmarks to compare the support for a global address space on two Cray machines, the T3D and T3E.

1.1 Overview of the T3D and T3E

In the spectrum of scalable parallel machines, the T3D and T3E represent a unique point on the spectrum between distributed shared memory and pure message passing. They augment a conventional microprocessor with a small amount of hardware to allow each processor to address memory on any processor, but do not cache remote data. The resulting communication overhead is much lower than standard message passing layers, thereby encouraging applications that demand small, frequent communication.

On initial inspection, the T3E is a significant improvement over the T3D: It has a faster clock, more on-chip cache (organized in two levels), and a higher bandwidth network. Figure 1 summarizes these differences. As we will show, these advantages do not always translate to better language-level performance. First, we discuss the two machines in more detail.

	T3D	T3E
processor	Alpha 21064	Alpha 21164
clock speed	150 MHz	300 MHz
instructions per cycle (maximum)	2	4
L1 cache size	8 KB	8 KB
L2 cache size	–	96 KB
network latency	.55 us	.5 us
peak link bandwidth	300 MB/s	600 MB/s

Figure 1: *Peak performance of hardware components on the Cray T3D and T3E.*

The Cray T3D [4, 10, 11] is built around the 150MHz dual-issue Alpha 21064 processor [8, 16] and is scalable to upto 2048 processors. It has an 8 KB L1 cache, but, unlike workstations built with the 21064, no off-chip L2

*This work was supported in part by the Defense Advanced Research Projects Agency (N00600-93-C-2481, F30602-95-C-0014, F30602-95-C-0136), the National Science Foundation (CDA 9401156), Army Research Office (DAAH04-96-1-0079), U.S. Department of Energy (W-7405-ENG-48).

cache. The L2 cache is omitted to allow for lower memory latency on a cache miss. The global address space is implemented using a combination of hardware augmentations, which we refer to as the "shell." The primary component of the shell is a *DTB Annex*, which serves the role of a segment table for remote addresses. The DTB Annex is used to overcome the address range limitation of the 21064 processor. Once the Annex entries are setup, the processor can issue cached and uncached reads and non-blocking writes to remote locations. The shell also supports binding prefetch operations through a prefetch FIFO queue. In addition to these features, the machine has a DMA transfer engine (BLT), fetch-increment registers (which are accessible to remote memory operations), and a hardware barrier mechanism.

The T3D offers a large menu of primitives for implementing remote accesses, but there are performance problems and correctness issues that render some of these unusable [1]. First, it does not correctly support partial word stores, which means that data structures such as character arrays are cumbersome to implement. Second, address translation is done by the issuing processor, so it is possible to crash the program by issuing an address that is legal on the remote processor but not valid on the local one. Third, the remote prefetch mechanism has a low overhead, but the cost of Annex management and other book-keeping operations, which are necessary to avoid hazards arising out of physical synonyms, almost doubles the 600 nsec latency. Fourth, a remote write results in invalidation of the corresponding cache line, even if the target location is not being cached. Fifth, the Annex entries are accessed using special instructions, which are implemented using the opcodes for the Alpha load-locked and store-conditional instructions; the loss of these instructions affects read-modify-write sequences, which further exacerbates the lack of partial word stores. Finally, the BLT has a large startup cost, rendering it useless for anything but enormous transfers, in spite of possessing a peak bandwidth of 150MB/s.

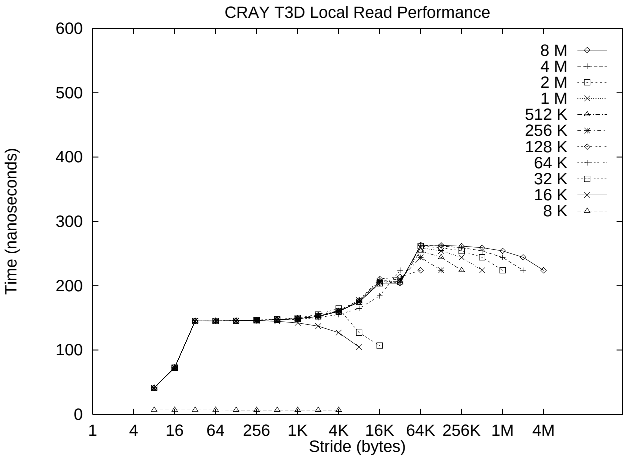
The Cray T3E [5, 15] was designed to address most of these difficulties. The T3E uses the 300MHz quad-issue Alpha 21164 microprocessor [9], which has a second level on-chip cache and added support for memory accesses, namely a write memory barrier. (Like the T3D, the T3E has no board-level cache.) The T3E introduces extra hardware logic to support virtual to physical address translation at the destination node for remote accesses; this eliminates the difficulty of addresses that are legal on one node but not on others. Language implementation on the T3E is greatly simplified by having a single remote access mechanism supported by a large set of off-chip registers, called the E-registers, as opposed to a wide variety of remote access mechanisms on the T3D. A detailed functional description of the synchronization and communication mechanisms of the machine can be found in Scott's machine overview [15].

1.2 Language overview

We evaluate the global address space support on the two machines using the primitives that underly the Split-C language [7]. Split-C is an explicitly parallel SPMD language, which provides a global address space abstraction through language features such as *global pointers* and *spread arrays*. The global address space features are similar to those in CC++ [3], AC [2], and other languages. Global pointers, which in Split-C can be distinguished from standard pointers through static typing, are wide pointers that represent a processor number and an offset within that processor's address space. Synchronous remote accesses are executed when global pointers are dereferenced within expressions. The language also provides split-phase (or non-blocking) variants of read and write, called *get* and *put*, to mask the latency of remote accesses. An explicit *sync* operation can be used to wait for split-phase accesses to complete. The language also defines a variant of write, called *store*, which avoids acknowledging the completion of a remote write, but rather increments a counter on the processor containing the target address. Both blocking and non-blocking variants of bulk transfer are also supported.

1.3 Roadmap

The remainder of this paper is organized as follows. In Section 2 we use micro-benchmarks to characterize the performance of the local memory hierarchy. In Section 3 we examine how the language's global address space abstraction could be implemented. Section 4 characterizes the performance of remote read and write operations, and identifies the performance and semantic implications for our language implementation. In Section 5 we discuss operations that mask remote memory latency. In Section 6, we study the performance of bulk transfer. Section 7 models the performance of an application kernel, and conclusions are drawn in Section 8.



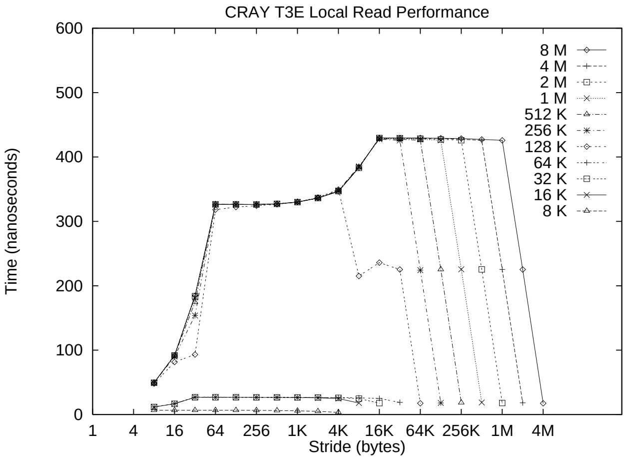


Figure 2: *Local Memory Read Performance.*

2 Local-Node Performance

We use experimental probes to identify the structure and performance of the local memory system on the T3D and the T3E. In addition to characterizing the performance of the system for different size data sets, the insights from this exercise are valuable when we study remote memory access operations, which are initiated and monitored through local memory operations on off-chip elements.

2.1 Local Read Latency

Our first micro-benchmark measures the cost of local memory accesses for a variety of access patterns. The experiment is designed to explore local node performance across a spectrum that at one end has the processor operating entirely out of its cache and at the other end has the processor operating with very little spatial locality. The micro-benchmark generates a stream of memory references whose stride and extent is varied. The stream is generated using repeated applications of the following code, which is a modified version of Saavedra's micro-benchmarks [13, 14].

```
for (arraySize = 4 Kilobytes; arraySize <= 8 Megabytes; arraySize *= 2)
    for (stride = 1; stride <= arraySize/2; stride *= 2)
        for (i = 0; i < arraySize; i += stride)
            MEMORY OPERATION ON A[i];
```

In this section, we examine the cost of executing this stimulus when the memory operation is a standard "load" operation. The results of the experiment are graphed in Figure 2. We plot the average read latencies as a function of stride and array size. In studying these graphs, we employ two general principles. First, if the read latency increases with array size for a given stride resulting in a new set of curves, it indicates an increase in conflict misses. Second, if an increase in stride causes an increase in the latency, it indicates a decrease in spatial locality that is provided by cache lines and memory pages.

Let us first examine the experimental data for the T3D. As expected, the results show that the first-level cache is 8KB and has 32 byte direct-mapped cache lines. As the array size and stride is increased, we observe inflection points when the array references are 16KB and 64KB apart. This effect exposes the structure of the DRAM organization. We can infer that the DRAM has 16KB pages with the pages interleaved across 4 memory banks. If two consecutive memory references to the same bank access different DRAM pages, the second access incurs the cost of switching the active page on the DRAM. The results also indicate the absence of a second level cache. The T3E data also reveals a first level cache that is 8KB in size with direct-mapped 32 byte cache lines. In addition, there is an on-chip second level cache, which operates with 64 byte cache lines. The second level cache size is between 64KB and 128KB with an associativity of three. We also observe that the DRAM is organized into 16KB pages.

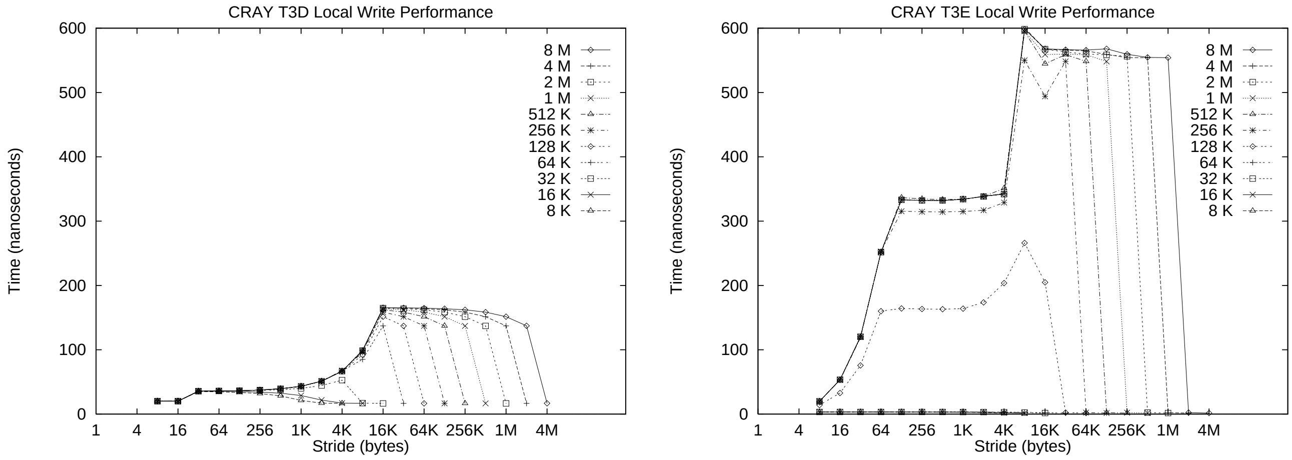


Figure 3: *Local Memory Write Performance.*

From the experimental data, it appears that T3E's L2 cache is a mixed win. The read latencies are considerably lower on the T3E when the data fits into the second level cache. However, the main memory latencies are significantly higher, which would significantly slow down codes that access large data sets. The T3E node is less "Cray-like" and more like a workstation. Also significant by their absence on both machines are latency increases due to TLB misses. This behavior indicates that both the T3D and the T3E support very large page sizes. This observation is useful when we examine the remote access mechanisms, which typically access memory mapped registers that are far apart in the virtual address space.

2.2 Local Writes

Our second experiment evaluates the access latencies for writes for the same set of access patterns. The results, which are shown in Figure 3, show dramatically different performance profiles, which can again be attributed to the miss penalty of T3E's L2 cache.

Both machines have a write-through read-allocate first level cache along with a write-buffer, which enhances performance by buffering as well as merging writes. The L2 cache on the T3E is a write-back write-allocate cache, which results in very low latencies for small data sets as the system eliminates all of the main memory accesses. However, the write-back property of the second level cache manifests itself in the relatively high latencies for large arrays where every access results in a write back of a dirty line in addition to fetching a new cache line.

3 Software Overheads in Global Addresses

One of the differences between advertised hardware performance and observed language-level performance comes from the manipulation of global addresses. On a distributed memory architecture, global addresses must contain sufficient information to indicate a processor number and a memory address within that processor's memory. A dereference of a global pointer typically requires some processing of the pointer itself, followed by read or write operation to an object in the global address space. The first part may be done by hardware or software and involves extraction of the processor number and a check to see whether the data resides on the issuing processor. In Split-C, global pointers are statically typed to be distinct from local pointers, so the overhead of global pointer dereference is not incurred for pointers that are statically known to be local and are marked as such by the programmer. In any global address space language, a key implementation challenge is to provide an efficient physical representation for global pointers and to map accesses through global pointers to the hardware primitives.

3.1 T3D Design Constraints

Two aspects of the T3D's global addressing affect our design: an external segment register table for accessing remote memory and source-based address translation. The T3D design allows remote accesses to be initiated through the standard load and store instructions on a machine-defined notion of a global pointer. However, the designers had to overcome the limitation that the Alpha 21064 processor has only 33 physical address bits coming out of the chip, which is not sufficient to address the global physical memory of a fully configured T3D. Their solution is an off-chip segment register table (DTB Annex) for providing the high order bits of a global address. The target processor number is stored in an Annex entry, whose number is appended to the virtual address (which is the local address within the target processor) to obtain a global address. When the annex entry number appearing in a global address is zero, the address names a location in local memory. The advantage of allowing local addresses to go through the annex is one of uniformity: A global address can name both local and remote locations and access them with the same set of load and store instructions.

3.2 T3E Design Constraints

The T3E has a markedly different design. There is no machine-defined notion of a global address, in spite of the increased addressing capabilities (40 physical address bits) of the 21164 processor; the hardware does not overload the use of load and store instructions for accessing remote memory. Instead, a short sequence of loads and stores to memory mapped external registers (E-register) initiate and complete a remote access. More specifically, a remote load is implemented by writing the target processor number and address to a memory mapped location followed by a load from a E-register, which returns the fetched value. A remote store requires an E-register to contain the value to be written before the store operation is initiated by writing the processor number and the address on a memory mapped location. The virtual address of the target location is communicated in its untranslated form to the destination node where the shell provides logic to do address translation.

3.3 Global Address Representations

There are two conflicting goals in representing global addresses. Since both machines require that the software identify a processor number, it is more efficient to store this value separately from the local address than to pack the two values into a single word.[1] However, the additional storage required for pointers, especially in languages with no type distinction between local and global addresses will impact memory usage and performance as pointers are moved through the memory hierarchy. The Alpha processor supports 64-bit addresses internally, and also have efficient byte manipulation instruction, so we store the processor number and address within a single word, but at a byte boundary. This strategy provides an implementation for storing and transferring global pointers that is as efficient as local pointers.

4 Remote Memory Access

In this section, we measure the hardware performance of the remote access mechanisms before determining the cost of mapping Split-C primitives onto the hardware mechanisms. We conclude with a discussion on compiler implications.

4.1 Remote Read Latency

We repeat our experiment by modifying our benchmark to issue read requests to remote memory. The results of the experiment are shown in Figure 4. On the T3D, the remote read latency is between 600 and 750ns. On the T3E, the average read latency is about 1500ns with slightly higher latencies when the stride exceeds 16KB (DRAM page miss) and when the stride is less than 64bytes.[2]

The remote read latencies are substantially higher for the T3E. The increased latencies are primarily due to the extra logic that the T3E requires for address translation on the target processor and higher main memory access costs. We have devised experiments that show that the additional cost introduced by the address translation logic is about 450ns. Our perspective is that the T3E shell pays the penalty for duplicating Alpha's efficient on-chip address translation mechanism.

[1] One could imagine a strategy on the T3D in which Annex entries are managed like registers by the compiler, but because the processor number is rarely known at compile-time, in practice each global reference results in a separate Annex setup.

[2] This behavior indicates that accessing a word in the middle of a cache line on a remote processor is slower than accessing the first word.

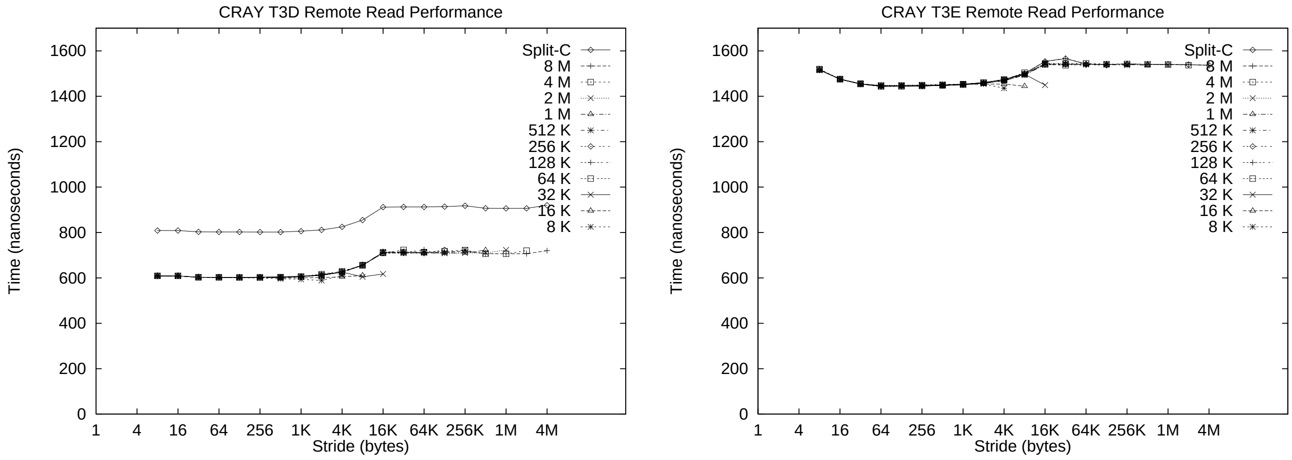


Figure 4: *Remote Memory Read Performance.*

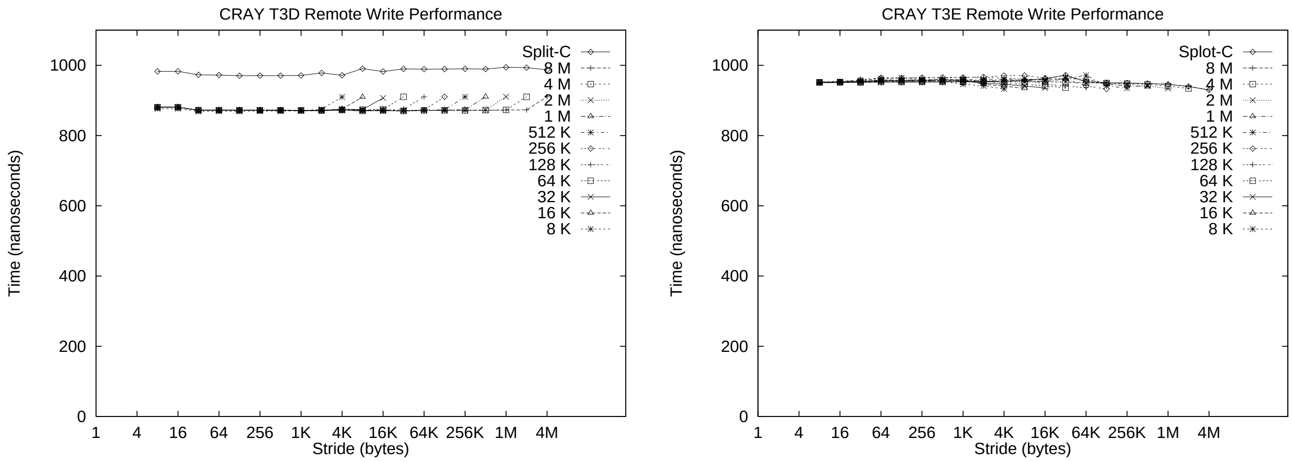


Figure 5: *Remote Memory Write Performance.*

4.2 Remote Writes

In our next micro-benchmark, we issue remote writes instead of remote reads. On both machines, an off-chip flag is polled to check for completion of the write operation. Figure 5 shows the results of the experiment. There is very little difference between the raw hardware write latencies on the two machines. This behavior is very surprising given that the remote read latency on the T3E is substantially higher than that on the T3D. A possible explanation for this behavior is that writes are acknowledged as they are received by the remote node without waiting for the corresponding address translation and memory operation to complete.

4.3 Compiler Implications

The Split-C implementation of remote accesses on the T3D includes the cost of manipulating the Annex in addition to the raw hardware costs. The total cost seen by the programmer is shown in Figures 4 and 5. Eliminating Annex setup costs requires complex program analysis (as discussed in [1]) as the compiler has to ensure that physical address synonyms are not generated. However, on the T3E, the Split-C implementation does not incur any additional cost beyond the raw hardware performance.

On the T3D, the design choices of using an Annex table and source-based address translation, in the absence of OS support, result in a rather tricky language implementation problem. Since the Annex contents determine the actual translation of a global address, and since it is impractical to access the Annex during a virtual address translation, the designers decree that the address mapping must be uniform across all processors. To preserve this property, heap memory allocation for globally visible data requires a coordinated allocation strategy across processors. Note, however, that this issue could be resolved with OS support for managing the Annex (just like the OS handles TLB misses) without requiring specialized hardware for address translation at the destination node as found in the T3E.

The Split-C write implementation for the T3D has a hidden cost that could be potentially vicious for codes optimized carefully for cache performance. When the destination processor sees only a physical address along with a write request, it flushes the corresponding cache line from the cache even if the processor is not currently caching the memory location. This issue is resolved cleanly on the T3E, where each node maintains a back-map of which lines are currently being cached and issues a probe only if it knows that the target location is being cached. We have measured the cost of the back-probe to be about 20ns (6 cycles).

4.4 Semantic Mismatches

In our earlier study, we identified two semantic mismatches between the language and the T3D machine. These mismatches continue to persist with the new machine.

Byte Writes: The 21064 processor does not support byte store operations. The processor however has a set of powerful byte manipulation instructions. A byte store could therefore be effected by a read-modify-write sequence. While this solution works on a local node, it breaks down in a multiprocessor context like the T3D. The T3E has the capacity to support remote byte store operations since a remote write operation is disguised as a store of a virtual address to a memory mapped location. Unfortunately, the shell does not support a byte store operation to a remote node. Byte writes have to be implemented on top of a more heavy-weight mechanism such as Active Messages [17]. A significant improvement though is that the handler code on the T3E can use the load-locked/store-conditional instructions, which were not available on the T3D since the shell interprets these instructions as Annex operations.

Global-Local Consistency Issues: The Split-C implementation of remote reads and writes wait for the completion of the operation. Split-C also allows the owner of a globally accessible value to access the value through local pointers. However, since loads and stores through local pointers can be reordered due to the write buffer, the implementation could cause violations of sequential consistency. This problem is generic to any MPP system that allows a remote node to access memory locations without involving the local node.[3] Since we would like to avoid the rather unfeasible solution of flushing the write buffer after every store operation, this problem indicates that the language requires an extended type system along with compiler analysis for distinguishing purely local stores from stores that could interfere with remote loads and stores.

5 Split-Phase Accesses

Split-C provides split-phase operations, *get* and *put*, to overlap the latency of remote access operations with other useful work. In this section, we measure the performance of the corresponding non-blocking machine primitives and draw conclusions on how closely the Split-C implementation tracks the performance of the hardware primitives.

5.1 Prefetching

T3D has a binding prefetch mechanism that utilizes an off-chip queue of 16 entries. A non-blocking remote load operation is initiated by issuing the Alpha "fetch" instruction on a global address. The fetched element is stored at the tail of the prefetch queue. A load instruction to a memory mapped address pops an element from the head of the queue. On the T3E, as we had discussed in the previous section, a remote read is implemented by issuing a store operation to a memory mapped shell location followed by a load operation from an E-register. The remote access latency can be masked by simply delaying the E-register load. We use a different micro-benchmark to evaluate the performance of the non-blocking prefetch operation. The experiment issues n prefetches followed by n loads from either the off-chip queue (on the T3D) or the E-registers (on the T3E). The results of varying n are shown in Figure 6. n is limited to 16 on the T3D, whereas it can be increased upto 480 on the T3E.

[3] On systems where a Split-C handler is executed by the local node, the handler would see the values languishing in the write buffer.

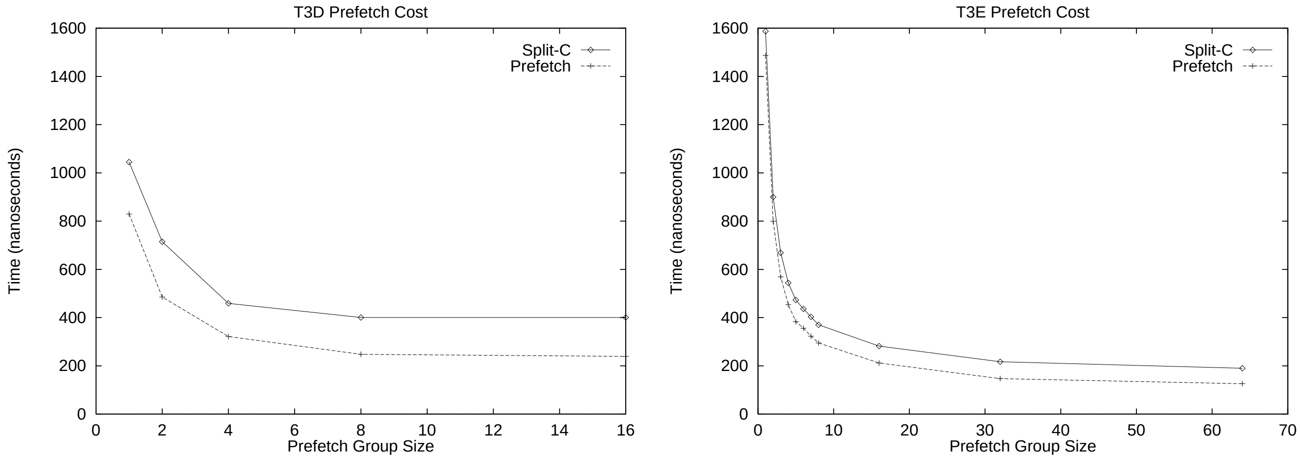


Figure 6: *Prefetch Read Performance.*

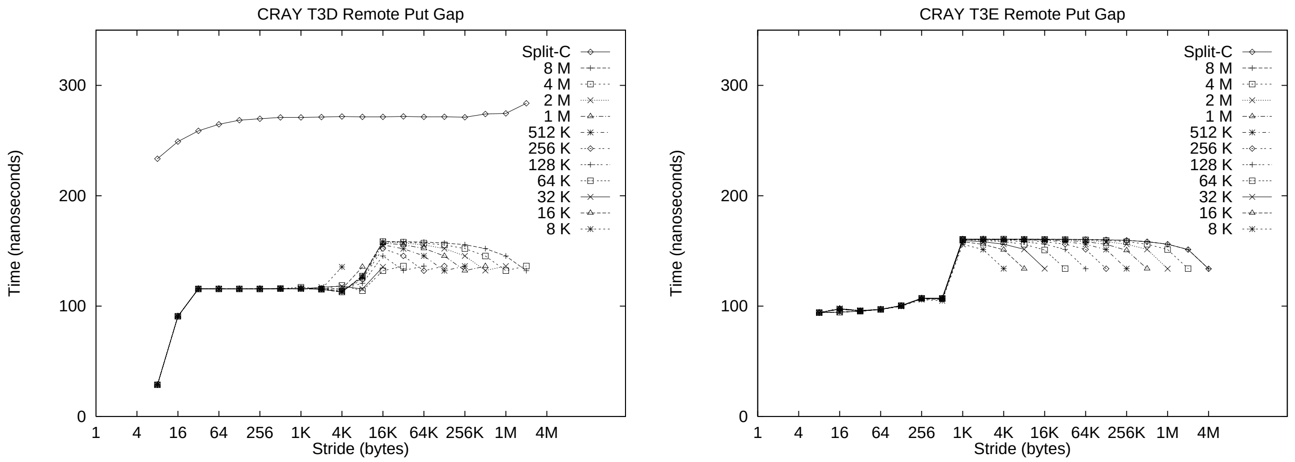


Figure 7: *Non-blocking Write Performance.*

On the T3D, most of the latency gets hidden by grouping prefetches with the cost stabilizing at about 250ns per prefetch. Since the memory access costs are less than 150ns (based on our local-node measurements), the performance limiting factor is the cost of crossing the boundary between the processor and the shell. The T3E attains a lower gap of about 130ns. Further instrumentation of the prefetch operation reveals that remote memory read accesses can be pipelined at the rate of one every 67ns, while the cost of doing a store followed by a load of an off-chip register is 120ns. We are now beginning to see the effect of having a second level cache, which increases the cost of shell operations.

5.2 Non-Blocking Writes

Recall that on both machines, an off-chip flag is polled to check for completion of a remote write operation. A non-blocking write operation simply eliminates the check. The performance results for non-blocking writes are shown in Figure 7. We observe the write merging behavior even for remote writes on the T3D. However, since a remote store operation on the T3E is disguised as a store of an address value into a memory mapped register, the remote stores do not merge in the write buffer. Consequently, the benefits of spatial locality for updating remote data structures are lost. On both machines, the memory operation at the remote node is the performance limiting factor, which is in contrast to the performance characteristics of the prefetch mechanisms.

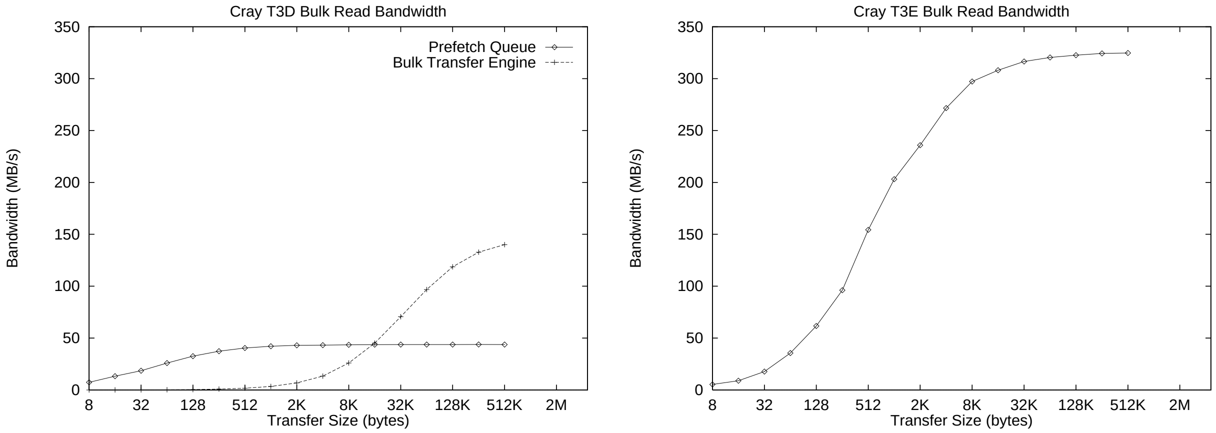


Figure 8: *Bulk Read Performance.*

5.3 Compiler Implications

As expected, a Split-C implementation of *get*s on the T3D includes the cost of updating an Annex entry. On both machines, there is an additional cost in mapping the language operation onto the machine primitives. Recall that a Split-C *get* operation specifies a local address into which a remote value is fetched. The local address, which is available when a prefetch is initiated, is required when the remote access completes. The obvious solution is for the compiler to maintain a table of these local addresses for outstanding prefetch operations. Through compiler analysis, this book-keeping could be eliminated. The compiler analysis for T3D requires accurate information as to the exact number of outstanding prefetches since the values are fetched into a dynamic address in a queue. The T3E design of using statically named E-registers makes the compiler analysis similar to the well-understood problem of register allocation for scalar processors.

Once again, we observe that the Split-C primitives, especially the *put* operation, can be implemented on top of the hardware primitives at a much lesser cost on the T3E. Though both machines have similar gap and overhead characteristics, the Split-C implementation on the T3E benefits from a closer match between the language and the machine.

6 Bulk Operations

A critical performance metric of any system is the bandwidth it can provide for bulk transfers. The Split-C operations for bulk transfers are *bulk_read*, *bulk_write*, *bulk_get* and *bulk_put* with the obvious semantics regarding synchronicity and direction of transfer. We now consider the issues that arise in implementing these operations.

6.1 Hardware Performance

The T3D provides multiple mechanisms for implementing bulk transfers. A *bulk_read* could be realized by either DMA transfers or pipelined prefetches. The drawback with DMA transfers is the cost of issuing a system call for initiating a transfer. For *bulk_write*s, we have a choice between DMA transfers and non-blocking remote put operations. Due to the write merging property of remote puts, a bulk_write implemented using remote puts attains a higher bandwidth than bulk_read does with the prefetch operation. A higher payload prefetch operation would have been useful.

The T3E design does not include the DMA engine. Instead, remote transfer using the E-registers is the only mechanism. However, one of the operations that the shell supports is a transfer of an entire cache line of data into and out of a set of E-registers, thus increasing the payload per operation. Furthermore, since E-registers can also be used for loading and storing into local memory, data can be transferred between two memory modules without entering a processor. Also, the E-register operations can be scheduled in a manner such that the cost of local memory operations are hidden by the latency of remote memory accesses. By providing more powerful

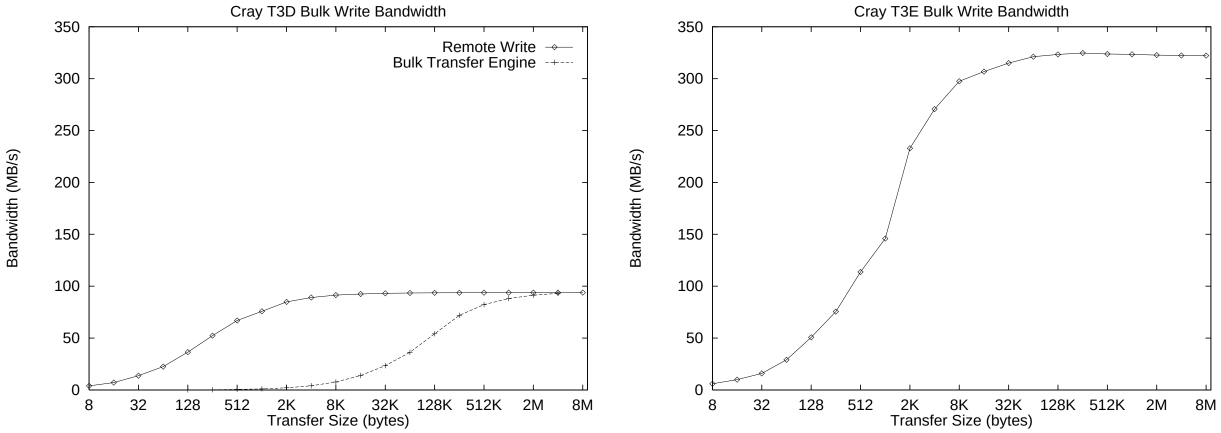


Figure 9: *Bulk Write Performance.*

operations and allowing the compiler to schedule them, T3E allows for more efficient implementations as shown in Figures 8 and 9.[4]

6.2 Compiler Implications

The performance profiles of the T3D mechanisms suggest which mechanisms to use for code generation based on the transfer size. An advanced compiler optimization that requires significant compiler analysis is to transform a *bulk-read* executing on the source processor into a *bulk-write* from the remote processor.

There is a hidden cost associated with the *bulk-read* on the T3E. Since the data never enters the processor, the processor suffers cache misses when it actually accesses the data. Given that main memory latencies are high, the effective bandwidth could be much less than the raw performance. This behavior suggests that for smaller data transfers (less than 64K), the compiler could fetch the data into its cache in addition to writing the values into memory. Since loads from E-registers can be hidden by the latency of main memory operations, this strategy is feasible without increasing the cost of bulk transfers.

7 Application Performance

In this section we build upon the performance measurements from previous sections to understand the behavior of a Split-C application kernel, EM3D, that models the propagation of electro-magnetic waves through objects in three dimensions [12]. A preprocessing step casts this problem into a simple relaxation algorithm on an irregular bipartite graph containing nodes representing electric and magnetic field values.

We first examine the local-node performance for different graph sizes to model the effect of cache performance. A useful performance metric is the average time for processing an edge in the graph. This corresponds to reading the value of a neighboring graph node and a floating-point multiply-add. The results, which are shown in Figure 10, are consistent with the absence of a second-level cache on the T3D and higher cache-miss costs on the T3E.

For a parallel implementation, the graph is represented using global pointers, and is spread across all of the processors. We have developed several versions of the application with varying degrees of optimizations. In the simplest version (Simple), the value associated with a graph node is fetched whenever one of its neighbors is updated; a remote node is fetched multiple times if it is required more than once during a single time-step. An obvious optimization is to introduce local "ghost nodes" that serve as cache-sites for remote values. This version (Bundle) benefits from reuse of cached values as well as better code generation since the compute and communication phases are separated. Both versions (Simple and Bundle) employ blocking read operations to access remote memory. Version Get optimizes the communication phase by pipelining the ghost-node fetch operations, and in version Put, the ghost-node values are pushed (using pipelined "puts") by the remote processor

[4] The $N_{1/2}$ numbers for our *bulk_read* implementation are lower than that of the vendor provided *shmem* library [15].

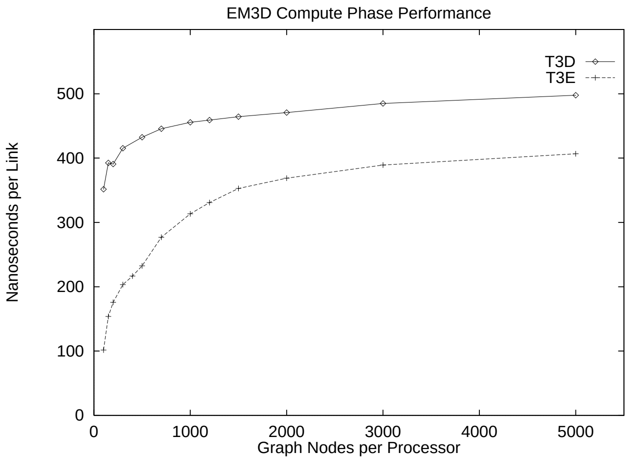


Figure 10: *Effect of Local Memory Hierarchy.*

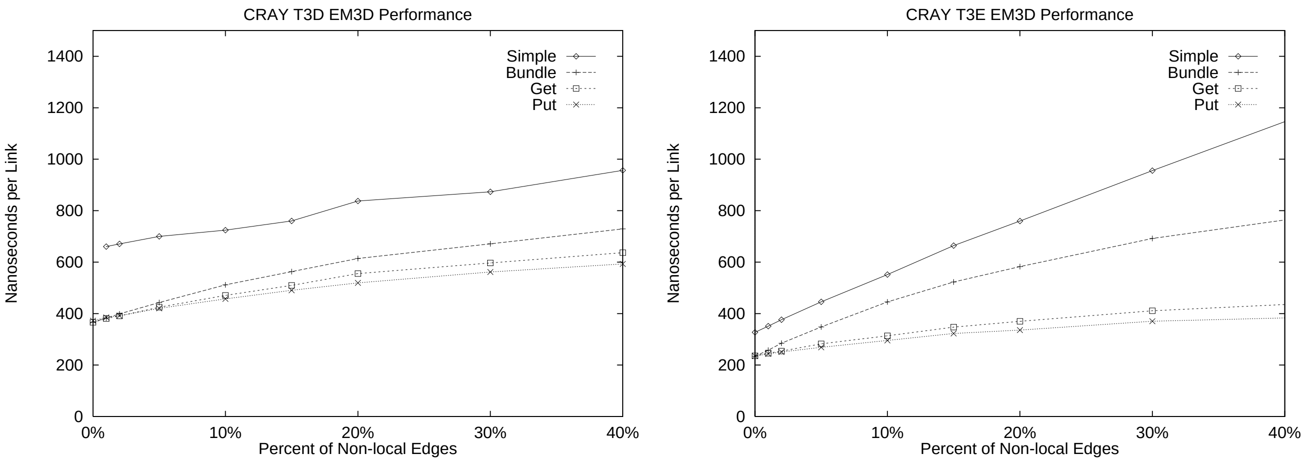


Figure 11: *Execution Costs for EM3D:* Performance results on 16 processors, for a graph size of 500 nodes per processor with each vertex having a degree of 20.

as opposed to being fetched by the source processor. The performance results for synthetic graphs with varying communication requirements are graphed in Figure 11. The performance of the different versions vary more markedly on the T3E due to higher latencies for blocking accesses and lower overheads for non-blocking operations.

8 Summary and Conclusions

In this paper, we apply a gray-box performance study to compare two machines. The contributions of this study are: (i) the first detailed characterization of language-level performance for the Cray T3E, (ii) an empirical comparison of the Cray T3E and the Cray T3D, and (iii) identification of the implementation issues for mapping a global address space onto the two machines. In comparing the two machines, we observe that the T3E has fewer options for implementing remote access and therefore a much simpler compilation strategy than the T3D. Surprisingly, the hardware-level performance of blocking remote access operations have worse performance on the T3E than the T3D, due to extra logic on the destination processor and the presence of a second level cache on the source processor. However, the language implementation adds less overhead than a corresponding implementation on the T3D resulting in better end-to-end performance for the non-blocking primitives. Consequently, the T3E performs better on applications that allow remote operations to be overlapped at a fine granularity.

The new machine also has certain architectural features that make the language implementation and programming task easier. The ability to perform address translation on the destination processor eases the compiler's responsibility in keeping address space extents uniform. The use of a back-map for invalidating cache lines during a remote write operation preserves the programmer's notion of what is in the cache. Also, the shell is accessed through standard load and store operations instead of the load-locked/store-conditional operations, which can now be used for effecting read-modify-write sequences. However, the continued lack of support for partial word stores complicates using data structures such as character arrays.

Performance characterization of parallel machines continues to be a challenging problem that is important for application writers and language implementors, as well as machine architects. In this paper we identified three levels at which performance can be measured. At the hardware component level, shown in Figure 1, we see the performance of individual system components, such as the link bandwidth and processor cycle time. At the system level, we see the effects of combining these components, such as the additional latency on the T3E relative to the T3D due to the addition of a second level cache and target-based address translation. At the language level, there is additional overhead for implementing the global address space, which is made up of Annex management on the T3D and E-register manipulation on the T3E. The insights drawn from studying the performance at each one of these levels are valuable in understanding the behavior of whole programs.

References

[1] R. H. Arpaci, D. E. Culler, A. Krishnamurthy, S. G. Steinberg, and K. Yelick. Empirical Evaluation of the CRAY-T3D: A Compiler Perspective. In *International Symposium on Computer Architecture*, June 1995.

[2] W. Carlson and J. Draper. Distributed Data Access in AC. In *Proceedings of the 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP)*, Santa Barbara, CA, July 1995.

[3] K. M. Chandy and C. Kesselman. Compositional C++: Compositional Parallel Programming. In *5th International Workshop on Languages and Compilers for Parallel Computing*, New Haven, CT, Aug. 1992.

[4] Cray Research Incorporated. *CRAY T3D Hardware Reference Manual*, 1993.

[5] Cray Research Incorporated. *CRAY T3E Hardware Reference Manual*, 1996.

[6] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Sumbramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Principles and Practice of Parallel Programming*, May 1993.

[7] D. E. Culler, A. Dusseau, S. C. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick. Parallel Programming in Split-C. In *Supercomputing '93*, pages 262–273, Portland, Oregon, Nov. 1993.

[8] Digital Equipment Corporation. *DECchip 21064-AA Microprocessor Hardware Reference Manual*, 1992.

[9] Digital Equipment Corporation. *Alpha 21164 Microprocessor Hardware Reference Manual*, 1996.

[10] R. Kessler and J. Schwarzmeier. Cray T3D: a new dimension for Cray Research. In *Digest of Papers. COMPCON Spring '93*, pages 176–82, San Francisco, CA, Feb. 1993.

[11] R. Koeninger, M. Furtney, and M. Walker. A Shared-Memory MPP from Cray Research. *Digital Technical Journal*, 6(2):8–21, 1994.

[12] N. K. Madsen. Divergence Preserving Discrete Surface Integral Methods for Maxwell's Curl Equations Using Non-Orthogonal Unstructured Grids. Technical Report 92.04, RIACS, February 1992.

[13] R. H. Saavedra, R. S. Gaines, and M. J. Carlton. Micro Benchmark Analysis of the KSR1. In *Supercomputing '93*, Nov. 1993.

[14] R. H. Saavedra-Barrera. *CPU Performance Evaluation and Execution Time Prediction Using Narrow Spectrum Benchmarking*. PhD thesis, U.C. Berkeley, Computer Science Division, Feb. 1992.

[15] S. L. Scott. Synchronization and Communication in the T3E Multiprocessor. In *Architectural Support for Programming Languages and Operating Systems*, 1996.

[16] R. L. Sites. *Alpha Architecture Reference Manual*. Digital Equipment Corporation, 1992.

[17] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: a Mechanism for Integrated Communication and Computation. In *International Symposium on Computer Architecture*, 1992.