ELDORADO

John Feo, David Harper, Simon Kahan, Petr Konecny
Cray Inc.

ABSTRACT: *This paper introduces Eldorado, a third generation multithreaded architecture. Previous Cray multithreaded systems were plagued by unreliable hardware and high costs. Eldorado corrects these problems by using many parts built for other commercial systems. Its compute processor is a 500 MHZ multithreaded processor architecturally similar to the MTA-2 processor; but its interconnection network, I/O subsystem, and service processors are borrowed from other Cray systems. Eldorado retains the programming model, operating system, and tools of the MTA-2. It has the same capability as the MTA-2 to tolerate latencies and achieve high performance on programs that run poorly on SMP clusters. We present several programming examples to illustrate performance and scalability in the presence of high memory and synchronization latencies.*

KEYWORDS*: Multithreaded architectures, heterogeneous architectures, multithreaded processing, performance studies.*

1. Introduction

Eldorado is the third generation multithreaded architecture built by Cray Inc. The two previous generations, the MTA-1 [6] and MTA-2 systems [2], were fully custom systems that were expensive to manufacture and support. Eldorado makes use of many commodity parts and subsystems built for other commercial systems, thereby, significantly lowering its cost while maintaining the high performance, scalability, and simple programming model of the two previous generations.

The MTA-1 manufactured by Tera Computer Company was a custom system consisting of twenty-four GaAs ASICs, a 52-layer network board, and tens of thousands of connections. The system was large, complex, and power hungry. The ASICs were difficult to manufacture and unreliable. While the system did support some evaluation work [9], neither the hardware nor the operating system reached a state capable of supporting production work.

The MTA-2 developed by Cray Inc is an entirely CMOS implementation of the MTA-1. The CMOS design greatly simplifies all aspects of the machine making it both manufacturable and reliable. In 2002, two MTA-2 systems were delivered to customers. These machines remain in use today supporting research and production work [7,8]. Unfortunately, the high cost of the MTA-2's custom design made it difficult to sell.

Eldorado reduces system costs by leveraging the development of the Red Storm system [1] Cray has built for Sandia National Laboratory and has brought to market as the XT3 [3]. The XT3 is a large-scale distributed memory system supporting the MPI programming model. In a nutshell, Eldorado is an XT3 with MTA processors in place of AMD Opteron processors on the compute nodes: this replacement of processors transforms the distributed memory XT3 into a shared-memory Eldorado.

The components of early computer systems ran at the same speed. Processors ran at about the same speed as the memory systems from which they read and wrote data. Unfortunately, over the past two decades processor speeds have increased several orders of magnitude while memory speeds have increased only slightly. Processors are now starved for data. The conventional solution is to build hierarchical memory systems with two or three levels of cache, local memory, and remote memories. Only the first level cache is able to keep up with the processor. Any memory reference that misses that cache stalls the processor. The problem is exacerbated in large SMP clusters where more than 99% of memory is remote and messages most be passed between processors to move data.

Hierarchical memory systems with fast processors and slow networks present an almost insurmountable programming problem to users. To get even minimal performance and scalability, programmers must write code that reads and writes memory in cache order, reuses each data word many times, minimizes communication to large block transfers, minimizes synchronization operations, uses static data structures, and is load balanced. These constraints shrink the space of scalable algorithms and increase the size of programs. The rich literature on parallel algorithms is worthless. Many algorithmic techniques learned in graduate school, such as recursion, linear programming, branch-and-bound, and PRAMs, cannot be implemented efficiently on conventional systems.

The MTA-1 and the MTA-2 established multithreading as an efficient technique to hide system latencies [7,8,9]. By maintaining multiple threads of computation per processor and global shared memory, MTA systems use parallelism to tolerate latencies and solve the programming problem. When the executing thread must wait for a memory operation to complete or a synchronization event to occur, the processor switches to another thread and executes its next instruction. As long as at least one thread has a ready instruction, the processor remains busy. The system's shared memory lets any processor read and write any word and, most importantly, lets any processor execute any thread. Thus, workloads can be dynamically balanced without penalty. The programming problem reduces to writing code with sufficient parallelism to meet the "one ready instruction per cycle per processor" requirement. All the constraints listed in the previous paragraph are no longer important. Any algorithm with a high degree of parallelism will run well and scale on an MTA. To the degree that the XT3 network scales, the same is true for Eldorado.

In the next section, we present Eldorado and carefully point out the tradeoffs we made by adopting the XT3 infrastructure. In the third section, we discuss several programming problems suitable for Eldorado. We give source code and performance numbers for the MTA-2, and predict performance on Eldorado. We conclude with a roadmap for multithreaded systems.

2. Eldorado

Eldorado is a shared memory system that efficiently exploits large-scale fine-grain parallelism through architectural-level synchronization and scheduling. The Eldorado hardware infrastructure is based on the Cray Red Storm project [1], while the processor and system software are derived from the Cray MTA-2 Project [2]. As illustrated in Figure 1, an Eldorado node consists of compute and service modules. A compute module has four Cray MT processors with commodity DIMM memory, powered by Cray's MTX operating system. A service module consists of two AMD Opteron processors, four PCI-X interfaces, and commodity DIMM memory. The service nodes run the LINUX operating system. Both types of modules have 4 network interface chips. Table 1 compares Eldorado to the MTA-2.

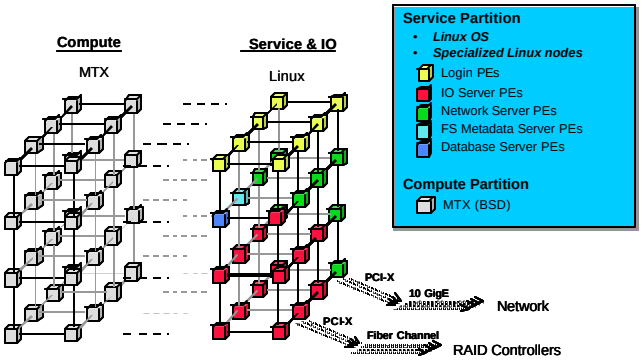

Figure 1. Eldorado system architecture

Table 1. Comparison of MTA-2 and Eldorado

	MTA-2	Eldorado
CPU speed	220 MHz	500 MHz
Maximum system size	256 processors	8192 processors
Maximum memory capacity	1 TB (4 GB/P)	128 TB (16 GB/P)
TLB reach	128 GB	128 TB
Network topology	Modified Cayley	3D Torus
Network bisection bandwidth	3.5 GB/s * P	15.3 GB/s * $P^{2/3}$
Network injection rate	220 M wrds/P	Variable (see Table 2)

2.1 MT processors

The architecture of an MT (multithreaded) processor is shown in Figure 2. Each processor has 128 hardware streams and a 64 KByte, 4-way associated instruction cache shared by all 128 streams. A hardware stream includes 32 general-purpose registers, a target register, and a status word that includes a program counter. Each stream holds the context of one thread. The processor executes an instruction from a different stream on every cycle in a fair manner. Only if no stream has an instruction ready to execute does the processor stall. There are three pipelined functional units, M, A, and C. Each instruction word may include operations that exercise all units. The M unit can initiate a read or write operation per cycle, the A unit can initiate a fused multiply-add, and the C unit can initiate either a control or an add operation. The cycle speed is 500 MHZ, so the processors have a 1.5 GFLOP/s peak performance (up from 660 MFLOP/s on the MTA-2).

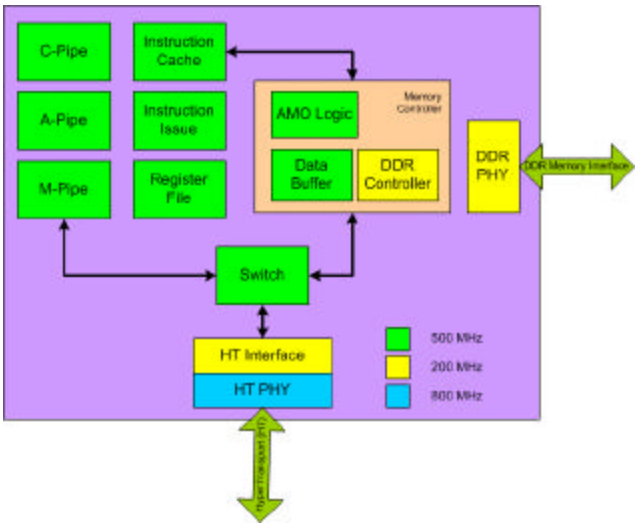

Figure 2. MT processor block diagram

2.2 Service processors

The hardware implementing Eldorado's service nodes is identical to the service node hardware on the XT3. There are two Opteron sockets on the module; each can be populated with either single or multi-core processors. Each socket connects with four commodity DIMM sockets, with a network interface (Seastar2), and with a PCI-X interface. The reuse of the existing hardware design allows Eldorado to also reuse much of the software stack running on the service nodes. This stack includes a Linux kernel, job launch and batch facilities, and I/O software including the high-performance parallel Lustre file system.

2.3 Memory system

The MT processor memory system is a global, shared address space accessible by all MT processors. There are as many memory modules as there are processors. Each module can be configured to be from 4 to 16 GBytes in size. All memory words are 8 bytes wide and the memory is byte-addressable. Associated with every word are several additional bits: a full-and-empty bit, a forwarding bit, and two trap bits. Memory is implemented using commodity DDR components and is protected against single bit failures. Each module has a single access port and is 128 bits wide. Service processors can access MT processor memory only by message passing. The Seastar2 ASIC contains a DMA engine that performs all transfers between MT processor memory and Opteron memory.

Each MT processor memory module has a 128 KByte, 4-way associated data cache. A line of 8 words (64 bytes) is transferred to the cache whenever a word of the line is accessed. Only the accessed word moves beyond the buffer, so network bandwidth is not wasted moving words not requested by a processor. This buffer is logically part of the memory system and not the processor; it is not a data cache in the conventional sense; there are no cache coherence issues. The buffer simply retains recently accessed words and is used to reduce the bandwidth required from the DRAM pins. To adapt to the commodity memory technology used in Eldorado, logical addresses are hashed to physical addresses in 8 word blocks rather than the word granularity hashing that was used in the custom MTA-2 memory system.

At boot time, the system can divide the MT processor memory into global and local memory segments. While the user will see only global shared memory, we plan to enhance the compiler and runtime system to exploit any local memory set aside at boot time. Certainly, it makes sense to spill registers, reserve stack space, and allocate thread private data in local memory rather than disperse them around the system as is done in the MTA-2.

2.4 Interconnection network

The network topology is a 3D torus. The original Red Storm network design was optimized to support message passing and although the raw performance of the network is in excess of 37 Gb/s per link per direction, Eldorado communication suffers some inefficiency due to suboptimal packet formats. It was not within the scope of the Eldorado project to re-implement the network to mitigate this problem. Despite this shortcoming, the network bandwidth is reasonably well matched to other paths in the system. Communication in the Eldorado network is protected from error on a per link basis. The network also supports all classes of Eldorado operations including the atomic fetch-and add operations that are part of the MTA instruction set.

2.5 Speeds and feeds

Figure 3 shows some of the important performance metrics of an Eldorado node. As mentioned previously, the execution pipes are capable of a 1.5 GF peak performance. Memory performance is more complex to analyze. Previous experience with the MTA-2 hardware and the class of algorithms that have been run successfully on the architecture suggests a useful memory performance metric is the sustainable rate at which uniformly distributed memory operations can be injected into the system. Figure 3 shows the four points at which this metric is determined for a system. First, the DDR memory interface supports 100 million memory operations per second (assuming DDR3200 technology). Second, the interface to the memory cache supports 500 million operations per second. Third, the HyperTransport interface to the Seastar2 network interface supports 140 million operations per

second. The fourth interface is the interface to the system network.

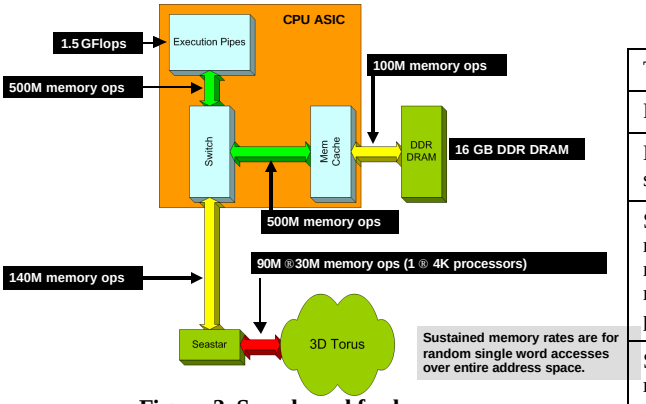

Figure 3. Speeds and feeds

The network interface is capable of supporting an issue rate of approximately 90 million memory operations per second. This interface represents the performance-limiting factor when performance is defined as described above. However, because the most interesting model of operation assumes uniformly distributed addressing, a more important performance characteristic of the network is its bisection bandwidth. The bisection bandwidth of a 3D torus scales with $P^{2/3}$, implying that the sustainable memory issue rate in the system will not scale linearly with system size. This condition represents a significant departure from the MTA-2 implementation that provided bisection bandwidth that scaled linearly with the number of processors in the system. Table 2 shows the impact of the sublinear bisection scaling at several system sizes. It is also interesting to note the penalty paid by having torus dimensions that are suboptimal for a given system size, an artifact of real world packaging issues. Despite these shortcomings, the larger system sizes offer a considerable capability to move random words of data.

2.6 I/O and file system

One deficiency of both the MTA-1 and MTA-2 systems is slow performance on serial scalar code. Consequently, OS code and in particular file system code performs poorly. Although the Eldorado processor clock frequency is twice that of the MTA-2 the clock is still 4 to 8 times slower than the clock speed of mainstream microprocessors. This performance differential makes it prudent for Eldorado to offload I/O operations to the service nodes, leveraging several technologies: the significant I/O software work done for Red Storm, the high serial code performance of commodity microprocessors, and the continuing improvement of I/O devices and drivers in commodity environments (Linux/PCI-X). I/O related system calls are packaged into messages and sent to one or more service nodes. Messages are decoded and handed to Linux and the file systems hosted on the service nodes.

Table 2. Estimated Eldorado scaling

Topology	6x12x8	11x12x8	11x12x16	22x12x16	14x24x24
Processors	572	1056	2112	4224	8064
Memory size	9 TB	16.5	33	66	126
Sustainable remote reference rate (per processor)	60MW/s	60	45	33	30
Sustainable remote reference rate (aggregate)	34.6GW/s	63.4	95	139.4	241.9
Relative size	1.0	1.8	3.7	7.3	14.0
Relative performance	1.0	1.8	2.8	4.0	7.0

2.7 Power and cooling

Eldorado power and cooling is directly leveraged from XT3. Power is delivered as 48 VDC to the edge of the board. Redundant voltage regulator modules (VRMs) convert the 48 VDC to 1.8 VDC, 2.5 VDC, and 3.3 VDC. An embedded control unit monitors the output of each VRM. The control unit is capable of handling exceptional conditions by powering down the module and at all times communicates status to a system console.

Cooling is accomplished with chilled air and heat sinks attached to the processor and network interface ASICs. Air is driven vertically through the cabinet (24 modules stacked 3 high, 8 wide) using a single fan located in the base of the cabinet. The cooling problem for Eldorado (given the existence of XT3) was not a particular challenge because the MT processor consumes about 50% of the power required by the Opteron based XT3 design.

3. Program Examples

Multithreaded architectures present a unique programming environment to the user. Since parallelism is the only criterion for good utilization, the programmer's primary interest is to maximize parallelism. If sufficient parallelism exists, processors will always have an instruction to execute, and execution time equals the product of the number of instructions executed and the machine's cycle time. As with the MTA-2, Eldorado will be most cost-effective on codes that run poorly on SMP

clusters. Such codes have one or more of the following features: larger number of cache misses; severe load imbalances; sparse, dynamic, or adaptive data structures; high communication to computation ratio; or fine-grain synchronization.

In addition to eliminating many of the programming issues that make parallel programming so difficult on SMP clusters, Eldorado makes the compiler and runtime system responsible for implementing parallelism. It is the compiler, and not the user, that inserts instructions to reserve streams, divide loop iterations among the streams assigned, allocate stack space for each stream, pass global values, and recover all but one stream when the parallel work completes. The compiler recognizes most reductions and linear recurrences, and automatically generates codes to implement the operations in parallel and without race conditions. Where the compiler is unable to recognize parallelism automatically, the programmer may provide guidance via a wide variety of pragmas; thus, codes written in OpenMP and other shared-memory paradigms port readily with the caveat that greater parallelism is required than on other systems to achieve comparable performance.

Eldorado will run the MTA-2 operating system, a multithreaded Unix. It will inherit the compilers, libraries, programming tools, and runtime system developed for the MTA-2. One of the tools is CANAL, a static compiler analysis tool. It generates a report stating how each loop is compiled. For innermost loops it gives the number of instructions, memory operations, floating point operations, the compiler's estimate of the number of streams required to hide latency, and other statistics. If latencies are hidden, then a loop's execution time is the product of the trip count, number of instructions, and the machine's cycle time. Any difference in this product and the actual execution time implies that all latencies are not hidden. The user can then increase the number of streams or look for ways to increase the loop's parallelism.

On Eldorado, the analysis is more complicated because the network's bandwidth does not match the processor's appetite for data and it does not scale with the number of processors. Moreover, if the compiler and runtime system use local memory for register spills, stack space, and thread private data, then not all memory operations will go out over the network. We have developed a calculator to predict Eldorado performance. Its inputs are: trip count, number of instructions, number of memory operations to global memory, number of memory operations to local memory, cycle time, and system configuration. In the subsections that follow, we use the calculator to estimate performance on Eldorado. In each case, we assume all memory operations in an inner loop are to global memory: consequently, our estimates are conservative.

3.1 Sparse matrix multiply

Let $\boldsymbol{C} = \boldsymbol{A} * \boldsymbol{B}$ where $\boldsymbol{A}$ is a sparse n x m matrix. To save space we store only the nonzeros of $\boldsymbol{A}$ in packed row form, transforming $\boldsymbol{A}$ into a column vector of size nz (number of nonzeros), and introducing two new vectors: ***rows***, the start of each row in A; and ***cols***, the column index of the nonzeros. The MTA-2 code is:

```
#pragma mta use 100 streams
#pragma mta assert no dependence
for (i = 0; i < n; i++) {
  int j;
  double sum = 0.0;
  for (j = rows[I]; j < rows[i+1]; j++)
    sum += A[j] * B[cols[j]];
C[i] = sum;
}
```

There are two points to note about the code. First, the code is identical to what one would write for a sequential implementation. A recurring theme in multithreaded programming is that parallel code is no more complicated than sequential code. Second, the code is devoid of implementation details except for the two pragmas. The "no dependence" pragma assures the compiler that the memory spaces for $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are disjoint and do not overlap. The "request 100 streams" pragma asks the runtime system for 100 streams per processor to execute the loop.

The CANAL report is:

```
      | #pragma mta use 100 streams
      | #pragma mta assert no dependence
      | for (i = 0; i < n; i++) {
      |     int j;
3 P   |     double sum = 0.0;
4 P-  |     for (j = rows[i]; j < rows[i+1]; j++)
      |         sum += A[j] * B[cols[j]];
3 P   |     C[i] = sum;
      | }

Parallel region   2 in SpMVM
       Multiple processor implementation
       Requesting at least 100 streams
Loop   3 in SpMVM at line 33 in region 2
       In parallel phase 1
       Dynamically scheduled
Loop   4 in SpMVM at line 34 in loop 3
       Loop summary: 3 memory operations,
       2 floating point operations, 3
       instructions, needs 30 streams for full
       utilization, pipelined
```

We learn from the report that the i loop runs parallel and the j loop runs sequential. The inner loop consists of three instructions that execute three memory operations and two floating-point operations. The compiler chooses dynamic scheduling and estimates that 30 streams are sufficient to hide latencies. The scheduling decision is a good one since the work per iteration varies with the number of nonzeros

per row. The estimate of the number of streams is too low, so the need for the second pragma.

Table 3 gives the performance of the code for $n = m = 1$M. The number of nonzeros on each row is uniformly distributed between 0 and 1000. For the particular problem, nz = 499,902,410. On the MTA-2, the minimum execution time for the code is 6.82 seconds---the product of the number of instructions in the j loop (3), the number of times the j loop executes (nz), and the cycle time of the machine (220 MHz). Dividing minimum time by actual time shows that the machine is 96% utilized, so latencies are hidden. Since the innermost loop executes one memory operation per cycle, performance on Eldorado is limited by the network injection rate. Scalability degrades as the injection rate decreases with increasing number of processors.

Table 3. Sparse matrix-vector multiply

System	T (sec.)
IBM Power4 1.7 GHz (1 P)	26.10
MTA-2 (1 P)	7.11
MTA-2 (2 P)	3.59
MTA-2 (4 P)	1.83
MTA-2 (8 P)	0.94
Eldorado (576 P) *(estimated)*	0.043
Eldorado (2112 P) *(estimated)*	0.016
Eldorado (8064 P) *(estimated)*	0.006

3.2 Linked list search

Given N linked-lists each of length N count the number of list elements whose data field is equal to a particular target value. This problem is representative of applications that involve a lot of pointer chasing. Consider the following straightforward solution:

```
      | int count_data(List **ListHeads,
      |                int num_lists, int target) {
      |   int i; int sum = 0;
      |   for (i = 0; i < num_lists; i++) {
2 P   |     List *current = ListHeads[i];
3 PX  |     while (current) {
3 PX:$|       if (current->data == target) sum++;
 ** reduction moved out of 1 loop
3 PX  |       current = current->link;
      |   } }
      |   return sum;
      | }

Loop   2 in count_data at line 10 in region 1
       In parallel phase 1
       Dynamically scheduled
Loop   3 in count_data at line 12 in loop 2
       Loop summary: 2 memory operations, 0
       floating point operations, 4 instructions,
       needs 68 streams for full utilization,
       pipelined, 1 instructions added to satisfy
       dependences
```

Cray's MTA-2 compiler automatically parallelizes this code as is, without any pragmas. The compiler introduces a reduction to avoid hot-spot contention when incrementing the sum variable.

From the Canal output, we estimate the code's execution time. The compiler reports 4 instructions in the inner loop. Since the loop is executed N^2 times, $4N^2$ instructions are issued. If there are P processors each running at 220 MHz, then we expect the time to be: $(4N^2)/(220{,}000{,}000\ P)$ seconds. In particular, for N = 5,000 we expect 0.45 seconds for P = 1 and .045 seconds for P = 10. If we double N, we expect the time to go up by a factor of 4 to 1.8 and 0.18 seconds, respectively. The measured MTA-2 results are reported in Table 4, and agree with these estimates. Eldorado results are predicted based on the calculator mentioned previously. The much longer execution times on cache-based systems, despite significantly faster processors, show how ineffective these systems are on codes that routinely miss cache.

Table 4. Linked-list search

System	Time (sec) N = 5000	Time (sec) N = 10,000
SunFire 880 MHz (1 P)	9.3	107.0
Intel Xeon 2.8GHz (1 P)	7.15	40.0
MTA-2 (1 P)	0.485	1.98
MTA-2 (2 P)	0.053	0.197
Eldorado (576P) *(estimiated)*	0.0014	0.0058
Eldorado (2112 P) *(estimiated)*	0.0005	0.0020
Eldorado (8064 P) *(estimiated)*	0.0002	0.0008

3.3 Random access

RandomAccess is one of the HPC Challenge Benchmarks [5]. It is a variant of the famous GUPS benchmark that repeatedly performs a commutative update operator on data at randomly selected locations in a large table. In this case, the operator is bit-wise XOR.
The MTA-2 code for RandomAccess is:

```
#define NEXTRND(x) (((x)<<1) ^ ((x)<0 ? 7 : 0))
  int i, j;
  int bigstep = 262144;
  int tablesize = 4294967296  /* 2^32 words */
  int nstrs = (4 * tablesize)/bigstep;
#pragma mta assert parallel
#pragma mta dynamic schedule
#pragma mta use 100 streams
  for (j = 0; j < nstrs; j++) {
      v = random_start(bigstep * j);
      for (i = 0; i < bigstep; i++) {
          v    = NEXTRND(v);
          data = Table + (v & (tableSize-1));
          val  = readfe(data);
          writeef(data, val^v);
  }   }
```

The synchronized read and write operations, *readfe* and *writeef*, assure the atomicity of the update. *readfe* returns the value stored at the memory address when the full/empty is one and sets the bit to zero. *writeef* stores the value at the memory address when the full/empty is zero and sets the bit to one. Although the HPC Challenge definition allows for 1% error in the accumulated answer, the MTA-2 (and Eldorado) can compute 100% accurate values at no cost. The synchronized read and write instructions are no more expensive to issue than normal operations; however, they may not return for many hundreds of cycles. Regardless of when the operations return, the issuing processor continues to execute instructions as long as there are instructions ready to execute. So like all other latencies, synchronization latencies are hidden with parallelism.

The MTA-2 rates for RandomAccess far exceed all other platforms (Table 5). The rate is a predictable consequence of the code generated: five instructions for the inner loop body, implying 220M / 5 = 44M updates per second per processor. The presence of the random number generator dilutes the non-local character of the benchmark and makes it as much a test of floating point speed as of random memory access. Moreover, the random number generator generates a uniformly distributed stream of memory accesses guaranteeing good load balance. This fact has little consequence for the MTA-2 and Eldorado, but improves performance on SMP clusters.

For Eldorado, we have 5 instructions and 2 memory operations in the inner loop. Eldorado keeps pace with the MTA-2 up to 256 processors; but for larger systems its performance falls off with decreasing network bandwidth. The fallout is less than the falloff in the network (last line of Table 5) because the benchmark computes its random numbers on-the-fly, decreasing the density of memory operations.

Table 5. RandomAccess

System	Giga updates per second
Cray X1 800 MHz (60 P)	0.0031
IBM Power4 1.7 GHz (256 P)	0.0055
MTA-2 (2 P)	0.041
MTA-2 (5 P)	0.204
MTA-2 (10 P)	0.405
Eldorado (576 P) *(estimated)*	17.32
Eldorado (2112 P) *(estimated)*	47.57
Eldorado (8064 P) *(estimated)*	121.0

4. Conclusions

This paper has introduced Eldorado, the third generation multithreaded computer system built by Cray Inc. While retaining the programming model and latency tolerant capabilities of previous generations, the system improves reliability and manufacturability and reduces costs by borrowing parts from other commercial systems. The system is essentially an XT3 with MT processors in place of AMD Opteron processors.

Although the system does not provide the computation to communication balance of earlier systems, its multithreaded processor and support for fine-grain synchronization can still hide most of the memory and synchronization latency in codes that have a large number of cache misses, use non-array data structures, or have severe load imbalances. The critical bottleneck is the

injection rate of memory operations into the network. We may be able to lessen this bottleneck by enhancing the compiler and runtime system to exploit local memory set aside at boot time.

The use of multithreaded processors seems to be our best hope of overcoming the increasing gap between processor and memory speed. Multithreading will play a prominent role in the design of Cascade, Cray's HPCS system [4]. This system includes heavy-weight processors, very light-weight processors embedded in the memory system, and commodity processors. The light-weight processors will be heavily multithreaded permitting the processors to tolerate long latencies and execute fine-grain operations. It is likely the heavy-weight processors will be multithreaded, but to a lesser extent, in order to hide the latency of loading and storing the large data sets on which they execute. As the first heterogeneous system consisting of multithreaded and commodity processors, Eldorado will serve as an important test platform for Cascade. Most importantly, it will establish multithreading as a serious alternative to distributed memory, message passing programming.

References

[1] Cray Red Storm Project [Online]. http://www.cray.com/products/programs/red_storm/.
[2] Cray MTA-2 System [Online]. http://www.cray.com/products/programs/mta_2/.
[3] Cray XT3 System [Online]. http://www.cray.com/products/xt3/.
[4] Cray Cascade Project [Online]. http://www.cray.com/products/programs/cascade/.
[5] HPC Challenge Benchmarks [Outline]. http://icl.cs.utk.edu/hpcc/.
[6] Alverson, A, et. al., "The Tera computer system." in *Proceedings of the 4th International Conference on Supercomputing*, ACM Press, 1990.
[7] Anderson, W., et. al. "Early experience with scientific programs on the Cray MTA-2." in *Proceedings of SC '03*, ACM Press, 2003.
[8] Bokhari, S.H. and J.R. Sauer, "Sequence alignment on the MTA-2." *Concurrency and Computation: Practice and Experience*, **16**, pp. 823-839, 2004.
[9] Pfeiffer, W., A. Snavelly, and L. Carter, *Evaluation of a multithreaded architecture on defense applications*, SDSC Technical Report, 1999.

Contact Information

The authors can be contacted as follows:

John Feo, ***feo@cray.com***; David Harper, ***dth@cray.com***; Simon Kahan, ***skahan@cray.com***; Petr Konecny, ***pekon@cray.com***.