Scalability of Graph Algorithms on Eldorado

Keith D. Underwood Megan Vance Jonathan Berry
Bruce Hendrickson
Sandia National Laboratories*
P.O. Box 5800, MS-1110
Albuquerque, NM 87185-1110
{kdunder, mlvance, jberry, bahendr}@sandia.gov

June 1, 2006

Abstract

The Cray MTA-2 system provides exceptional performance on a variety of sparse graph algorithms. Unfortunately, it was an extremely expensive platform. Cray is preparing an Eldorado platform that leverages the Cray XT3 network and system infrastructure while integrating a new revision of the MTA-2 processors that is pin compatible with the AMD Opteron socket. Unlike the MTA-2, this platform will have a more constrained network bisection bandwidth and will pay a high penalty for random memory accesses. This work attempts to assess whether the Eldorado platform will scale and provide the level of performance that the MTA-2 platform did.

1 Introduction

Algorithms that operate on sparse graph structures are of particular intrest to the informatics community. Core algorithms include operations such as connected components, S-T connectivity, sparse matrix vector multiplication, and subgraph isomorphism (see *Graph Software Development and Performance on the MTA-2 and Eldorado* by Jonathan Berry in CUG 2006). These types of algorithms perform extremely poorly on the commodity processors that make up many of todays supercomputers. They tend to make somewhat random references to data that is distributed across the entire system. Thus, their performance is dominated by both the latency of remote accesses and the rate at which those accesses can occur (message rate).

One architecture that performs extremely well on these algorithms is the MTA-2. A single 220 MHz processor on the MTA-2 achieves comparable performance a 3 GHz Pentium-4. More importantly, because the MTA-2 is designed to provide remote load/store operations at a high rate and is designed to tolerate remote latency, it scales dramatically better than systems based on commodity microprocessors and commodity networks. The downside of the MTA-2 was that the overall system was extremely expensive. To address this issue, Cray is introducing the Eldorado system that places MTA-2 processors into AMD Opteron sockets and leverages the much lower cost infrastructure of the Cray XT3 system.

Unfortunately, the aspects of the system that reduce the cost also put the performance at risk. The network cannot provide full bisection bandwidth (a feature that MTA-2 leverages heavily). The memory system has changed from one that could sustain full-rate random accesses to the standard DRAM used by an AMD Opteron. Eldorado will even include a cache (of sorts). This combination of changes calls into question the potential for Eldorado to provide the same orders of magnitude advantage (at scale) that the MTA-2 could. This paper presents simulation and analysis that indicate that the Eldorado platform will perform surprisingly well in the face of its limitations.

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

2 Methodology

The performance of Eldorado systems is constrained by a number of interdependent factors. While Cray has an existing MTA-2 simulator, it does not account for any of the variables associated with Eldorado and is not generally available. While it is currently not possible to simulate all of the factors affecting performance simultaneously, good approximations can be achieved by simulating some of the factors independently and unifying the simulations through analysis. The approach was to analyze some graph algorithms, simulate the performance implications of the DRAM subsystem, and simulate the performance of the network at 512 nodes.

2.1 "Application" Data

Our first objective was to obtain reliable baseline data on the behavior of algorithms from the graph library. Since the Cray MTA-2 simulator was not available to us, we needed other methods to obtain sufficient data on the behavior of the application. The first step was to understand the rate at which applications make memory references. This data was obtained from the standard MTA-2 performance tools as the percentage of VLIW instructions that make a memory reference. While this does not provide an exact representation of the pattern of memory access instructions, the percentage (typically 50% or more) is high enough to imply that a statistical representation is sufficient.

Knowing the requested memory access rate is insufficient to fully understand the implications of the architecture for the application. Unlike the MTA-2, Eldorado is a non-uniform memory architecture with a much greater penalty for remote accesses (in terms of both latency and access rate) than for local accesses. Thus, a second important criteria is the percentage of those accesses that are local and the percentage that are remote. Cray instrumented their simulator to distinguish stack accesses (effectively, local accesses on the Eldorado platform) from heap accesses (remote accesses on the Eldorado platform). It was also necessary to capture the full address trace to perform some simulations of the eventual cache architecture. While the stack accesses are not the only accesses that can be made local on Eldorado, making other accesses local will require changes to both the system software and programming model that cannot be measured on the MTA-2; thus, the data we have available is used to consider a worst case, where only the stack accesses are local. However, there seems to be only very limited scope for exploiting local memory allocation in graph operations on highly unstructured graphs.

2.2 Cache Experiments

One of the many aspects of Eldorado that differs from the MTA-2 is the memory system. Rather than having memory distributed throughout the fabric (independent of any node) that can deliver full bandwidth with random accesses, Eldorado has a DRAM attached to each node that has limited random access characteristics. To better use this DRAM, Eldorado incorporates a buffer in the memory controller that is managed as a cache of the local DRAM only. The performance of this "cache" is critically important to the performance of the machine as a whole.

To measure the performance of the cache, memory access traces were taken from a single processor on the MTA-2 simulator. Each access in the trace was flagged as a local (stack) or remote access. It is expected that numerous threads will execute simultaneously on one Eldorado processor. While the trace from a single processor would certainly capture numerous threads executing, it is possible that the relatively small graphs used in the simulation environment would not lead to full utilization of the processor. To explore the implications of this possibility, the single processor data was used to generate multiple threads of access. Measurements were taken (in powers of 2) for each possibility between 1 and 128 replications. Each replication of the addresses was offset by a constant value (4999872) from the previous replicate to prevent the artificial introduction of cache hits. The constant was chosen to be large, 64 byte aligned, but not 32 KB aligned. This was done to prevent breaking natural alignment without introducing artificial aliasing pressure on the cache.

Typical codes will have a constraint on the lookahead that can be exploited. They will also have a combination of local (stack) and remote (heap/dataset) accesses. These factors skew the timing of the threads and pollute the cache with network traffic. Thus, the simulator imposed a lookahead constraint of 4 on the threads and, for remote accesses, imposed a statistical remote latency. Simultaneously, the cache was exposed to high-rate random accesses to provide a "polluting" factor.

2.3 Network Simulation

The second major factor in Eldorado performance that differs dramatically from the MTA-2 is the network. While the MTA-2 used a full bisection bandwidth modified Cayley graph network, Eldorado uses the 3D torus network from the Cray XT3. The implications of the torus vary depending on the exact machine size, but the experiments here assume an $8 \times 8 \times 8$ topology of 512 nodes. Note that this is not a natural machine size for the Eldorado infrastructure, but it is not exceedingly unreasonable to build.

The simulation of the network performance starts at the edge of the processor with the HyperTransport (HT) link into the Seastar network chip. The Eldorado processor will run at 500 MHz, but the HT link nominally runs with a 2 byte wide per direction, 1.6 GigaTransfer/s rate. Nominally, this provides 3.2 GB/s of bandwidth per direction. Realistically, this never happens. There are HT packets involved that impose 8 bytes of overhead per transaction, and an HT cave on the Seastar that will impose some overhead. Since our current simulator only permits clocks that are integer multiples of each other, we simulated this interface as a 2 byte wide (per direction) link and explored the parameter space from 0.5 GT/s to 1.5 GT/s. None of the HT rates impacted the results, so the latter was used to prevent introducing an artificial constraint. Based on our experience with the Cray XT3 network and our discussions with Cray, we believe this to be reasonable, but not exact. The link was simulated with a latency of 250 ns in one direction.

The router may be one of the most difficult parts to simulate correctly, but it is also the most crucial. All of the router queue depths are modeled as are the approximate latencies moving from one queue to the next. The queue arbitrations use round-robin arbitration. In addition, we make the assumption of a strict X, Y, Z dimension order routing algorithm. Alternatives, such as the (X+, Y+, Z+), (X-, Y-, Z-) approach used in the T3E, may provide better results under some load scenarios. Also, this does not account for any gains that could be achieved by the virtual channel spreading used in the T3E systems or the losses that might occur if a node were down. Unlike the T3E, adaptive routing was not available.

The network link itself has a 3.84 GB/s data rate that is achieved using high speed SERDES. Less than 3 GB/s is available for actual data transfer because of the reliability protocol on the link. Due to limitations in the simulator, the link is modeled as a 4 GB/s link with overhead added to represent the impacts of the link protocol. Thus, the link may be modeled as 9% too slow. The bit error rate (and corresponding impact on link bandwidth) is not modeled; however, this is a 10^{-12} event and should not be a major factor. Serialization and deserialization latency is approximately modeled as part of the latency associated with controlling the link, which is the bulk of the 52 ns router latency.

2.4 Network Traffic Generation

Network traffic was generated using statistical methods that are designed to match our best understanding of the algorithms analyzed. The nominal request rate was taken to be the fraction of instructions including a memory reference multiplied by the clock rate of the Eldorado processor. This yielded a sweep space of 150, 230, 300, and 400 million references per second. The local versus remote (stack versus heap) percentages were swept from 10% local to 80% local, since coding styles can be changed to increase the amount of local data used. The distribution of network accesses was assumed to be purely random. That is, the hash algorithm used to distribute memory on the MTA-2 processor is assumed to work and the code is assumed to have no significant hotspots. Sweeps to determine the impact of significant hotspots were done by probabilistically sending a higher fraction of the traffic to a hotspot.

The traffic generated at each node was assumed to originate from some number of threads running on the Eldorado processor (32, 64, or 128). Each thread was assumed to be a stream of random addresses. Lookahead was imposed on each thread for lookaheads of 2, 4 and 8. Threads were only scheduled if they had sufficient lookahead remaining.

One notable characteristic that was not captured was the change from a hash on an eight byte boundary to a hash on a 64 byte boundary. To date, we have not been sufficiently confident in our ability to assess the locality in the remote reference stream to create a model of the implied changes in traffic pattern. Realistically, it is possible that no such locality exists, but we have not been able to confidently declare that either.

2.5 Memory Effects on Network Simulation

Network requests must be serviced by the memory subsystem on the Eldorado chip. Unfortunately, to integrate 512 full memory system simulations with the simulation of the network would have yielded prohibitive simulation times on a serial simulator. Thus, network simulations were run with statistical memory characteristics and swept over several values. Memory characteristics were determined by average cache hit rate giving each access a probability of hitting the cache. Misses consumed part of a simulated memory bandwidth and incurred a higher delay, which was also impacted by contention. Network accesses were assumed to never hit the cache (and only pollute it) while local accesses were studied over several cache hit rates.

2.6 Simulation Durations

When simulating the cache and memory subsystem, entire traces were simulated; however, when simulating a 512 node network, it is necessary to choose a shorter simulation duration. Given the constant, uniform, statistical nature of the network traffic generation, we believe it is reasonable to reach "steady state" points relatively quickly (in terms of simulated cycles). To define steady state, we monitored checkpoint dumps for multiple checkpoints. Where measured network characteristics did not change for several checkpoints, we declared that to be steady state. For all constant traffic models, steady state was reached within 1 million cycles.

3 Results

To set the results of the performance analysis in context, it is useful to first examine the difference between the MTA-2 and the Eldorado platforms. Table 1 lists the access rates for the MTA-2 and Eldorado for comparison. The architectures are different in numerous ways. The processor clock has gone up by over a factor of two. The architecture has shifted from uniform memory access (UMA) to non-uniform memory access (NUMA) and from full random access bandwidth to limited random access bandwidth. This change brought the introduction of a 128 KB, 4-way set associative cache with 64 byte cache lines. Finally, the network technology changed and that changed the relative bisection bandwidth per processor from 220 Mref/s to 75 Mref/s. In addition, bisection bandwidth per processor is linear with the number of processors on the MTA-2, but *decreases* as the system size of the Eldorado grows beyond 512 nodes.

3.1 Application Characteristics

Table 2 presents characteristics measured from several graph kernels. Three versions of the connected components kernel are presented: the Bully algorithm (the best performing algorithm on the MTA-2), the Kahan algorithm, and the simple algorithm. The S-T connectivity kernel is presented for the "small" case (less than 30 nodes visited), the "medium" case (1000-2000 nodes visited), and the "large" case (at least 10000 nodes visited). The other two algorithms presented are sparse matrix vector multiply and subgraph isomorphism.

Two measurements are presented for each algorithm along with three derived metrics that are related to Eldorado. The first column presents the percentage of VLIW instructions that include a memory reference. The second column is the percentage of those memory references that go to the stack and, therefore, will be to local DRAM on Eldorado. The final three columns translate these characteristics to an access rate (based on the clock rate of Eldorado) and break that access rate into two categories: local and global. Local accesses go to the DRAM directly and global accesses must use the network interface.

Referring back to Table 1, we can compare the demands of the applications to the capabilities of the platform. In the best case, when most accesses hit the cache, an Eldorado processor can service 500 million memory references per second from the combination of the network and local processor. In the worst case, when no accesses hit cache, the node can service 100 million memory references per second from the DRAM directly. Given the nature of the network requests, network requests are highly unlikely to hit cache and the network could be requesting up to 75 million references per second and that will draw directly from the 100 million references per second the DRAM can service. The remaining 25 million reference per second plus any data serviced by the cache must be sufficient for the local accesses.

Clearly, the network will not be sufficient to sustain most of the codes at maximum rate. What is slightly more subtle, however, is that it is also possible for the DRAM to become the limiting factor if

Table 1: MTA-2 and Eldorado characteristics

Property	MTA-2	Eldorado
Clock	220 MHz	500 MHz
Local Memory Rate (Best)	N/A	500 Mref/s
Local Memory Rate (Worst)	N/A	100 Mref/s
Data "Cache"	N/A	128 KB, 64B line
Topology	Modified-Cayley	3D-Torus
System Size / Simulated Size	40 nodes	512 node (radix-8 torus)
Remote Memory Rate (Net, Best)	220 Mref/s	75 Mref/s
Bisection BW	$3.5GB/s \times P$	$15.3GB/s \times P^{2/3}$

Table 2: Memory access characteristics of several kernels

Kernel Name	% Memory References	% Stack	Access Rate Mref/s		
			Total	Global	Local
Connected Components: Bully	59	46	295	159	136
Connected Components: Kahan	60	53	300	141	159
Connected Components: Simple	56	49	280	143	137
S-T Connectivity: Small	75	10	375	338	37
S-T Connectivity: Medium	60	28	300	216	84
S-T Connectivity: Large	60	32	300	204	96
Sparse Mat. Vect.	46	53	230	108	122
Subgraph Isomorphism	30	34	150	99	51

Table 3: Cache hit rate of several kernels for several replication degrees

	Replications			
Kernel Name	64×	16×	4×	1×
CC: Bully	20%	63%	85%	99%
CC: Kahan	13%	52%	79%	92%
CC: Simple	17%	56%	82%	92%
S-T Connectivity	85%	95%	99%	99%
Sparse Mat. Vect.	70%	85%	93%	99.9%
Subgraph Iso.	63%	69%	85%	87%

the cache hit rate is insufficient for local accesses.

3.2 Caching Simulation

Table 3 shows the cache hit rates for each of the graph kernels under varying degrees of trace replication. Small, medium, and large cases for the S-T connectivity kernel could not be easily separated. Interpretation of this data requires considering two factors: how busy was the single node simulation and how busy will the Eldorado node be. Both of these are factors that are difficult to determine, so the caching simulation swept over a range of possibilities by replicating the work that was done by the code in the simulation. For each kernel, the columns represent the cache hit rate when we assume that Eldorado will have to be 64× as busy (or 16×, 4×, or 1×) as the MTA-2 simulation was. Using the Eldorado properties, the application properties, and the cache hit rates, we can determine whether the DRAM will be the first bottleneck. For the local DRAM to be fast enough that it is not the bottleneck, it must be able to service the local node's local request rate. If we start by assuming that the network is consuming 75% of the local DRAM bandwidth[1], then after subtracting what hits the cache, the local request rate must be below 25 Mref/s. Table 4 shows the number of local DRAM accesses each kernel would need assuming the cache hit rates in Table 3.

The data in Table 4 indicates that if Eldorado has no more than 4× as many threads per nodes as the MTA-2 simulation, the cache should perform adequately. Higher rates may lead to some limitations from the DRAM; however, even at 16× replication, the network is likely to be as limiting or more limiting that the DRAM system.

[1]Because it is making 75 Mref/s that are uncacheable

Table 4: DRAM access rate (Mref/s) needed

	Replications			
Kernel Name	64×	16×	4×	1×
CC: Bully	108	50	20	1.3
CC: Kahan	138	76	33	12.7
CC: Simple	94	50	20	9
S-T Connectivity	14	5	1	1
Sparse Mat. Vect.	37	18	9	0.1
Subgraph Iso.	19	16	8	7

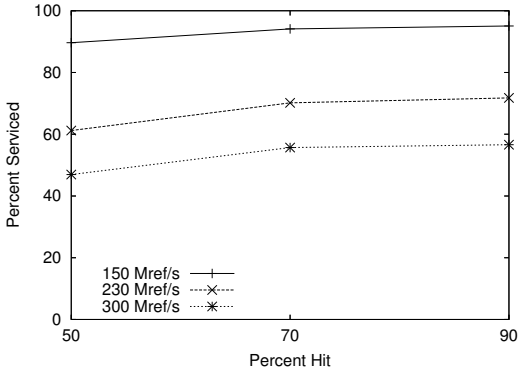


Figure 1: Impact of cache hit rate

To clarify the implications of the cache hit rate, consider the graph in Figure 1. This graph holds the lookahead constant (8), the local access percentage constant (60%), and the thread count per processor constant (64). What varies is the access rate (the various lines) and the cache hit rate. The metric is the percentage of the desired access rate that can be serviced. Thus, a value of 50% translates into a 2× performance hit from ideal. For higher access rates, the difference in a low cache hit rate and a high cache hit rate is significant. Despite the fact that higher access rate applications are predominantly bound by the network bisection bandwidth, dropping the cache hit rate from 90% to 50% in this scenario can drop the overall service rate from 56% to 46% — an 18% drop in overall performance.

3.3 Network Simulation versus Algorithms

A wide range of application configurations were simulated to produce over 1300 data points for a 512

node ($8 \times 8 \times 8$) system[2]. For the sake of brevity, this section presents only those data points in the space immediately around the current characteristics measured for the algorithms discussed in this paper. For each application, we assume the "typical" lookahead is 4. The typical number of threads per processor is assumed to be 64, which should generally be a reasonable number. The access rate, local reference percentage, and cache hit rate are taken from the application measurements above. Unfortunately, the current analysis was performed on graph kernels using an MTA-2 programming model and optimized for an MTA-2 system. The programming model of Eldorado and optimizations made for Eldorado may shift any number of these properties. Thus, for each kernel, the impact of moving along each of two key planes of interdependent variables is graphed. The first plane presented is the access rate/local reference percentage plane to explore the impacts of adding or removing traffic from the network. The second plane presented is the lookahead/threads per processor plane to explore the impacts of available concurrency on the results.

3.3.1 General Trends

As discussed earlier, cache hit rate can be critically important. In cases where the stack percentage was moderately high, the DRAM reference rate can easily become the primary bottleneck in the system if the cache hit rate is insufficient. The definition of "insufficient" depends on the requested access rate and the stack percentage, but, as an example, a 50% hit rate is insufficient when the stack percentage is 50% and the access rate is 230 Mref/s.

Stack percentage controls the point at which the network bisection bandwidth saturates; thus, it tends to be the primary bottleneck in the system. The 230 Mref/s to 300 Mref/s memory instruction rate means that the typical 50% stack percentage seen in the graph kernels will directly lead, in and of itself, to approximately a 2× performance penalty on the kernels at a system scale of 512 nodes. On a positive note, this is comparable to the performance that might be expected from a current MTA-2 system. Also, with the right development environment, many of the algorithms are expected to be able to achieve a somewhat higher "local" memory percentage beyond the current stack memory percentage.

Lookahead and the number of threads per processor are tightly coupled parameters. A smaller number of threads leads to the need for higher lookahead. Similarly, smaller lookahead requires more threads. Generally speaking (although certainly not universally), a lookahead of 4 looks achievable. With a lookahead of 4, 64 threads generally look sufficient.

[2]Larger systems may experience significantly worse results.

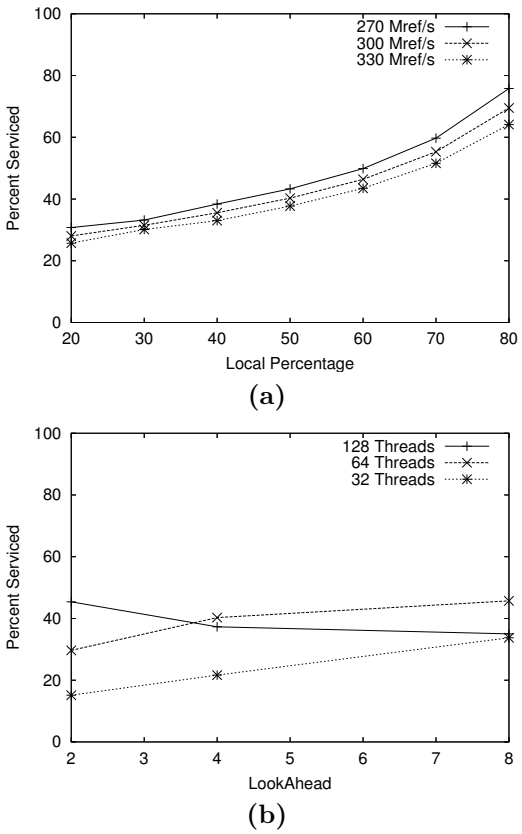


Figure 2: Impacts of **(a)** changes in global request rate and **(b)** changes in concurrency on Connected Components

3.3.2 Connected Components

For practical purposes, the properties of the three connected components algorithms are very similar. About 60% of the instructions reference memory and about 50% of those are local. That leads to a request rate for the network of approximately 150 Mref/s and a local reference rate of about 150 Mref/s for a total request rate of 300 Mref/s. Given the data from Table 3, we will assume a cache hit rate of 70%. Simulations have indicated that the network bisec-

tion bandwidth is on the order of 67 Mref/s[3], and so we would expect 44% of the request rate of these applications to be serviced. That translates into approximately a 2.3× performance penalty from what the peak would be. It is also possible that changes to the system software or the compiler to better exploit local memory could make a significant difference in these codes by shifting more of those accesses to the local memory. As a caveat here, while the "Simple" connected components algorithm looks better in the ways that are relevant to scalability on Eldorado, it is less parallel than the "Bully" algorithm and *would not scale*. Thus, we feel that Bully is a better, general-purpose method.

Given that the shift to Eldorado may change some of the salient features of the graph kernels, we have explored the implications of increasing or decreasing the access rate by 10%. We have also explored the entire spectrum of local percentages for each of these access rates. The results depicted in Figure 2(a) are unsurprising. Lowering the memory demands of the application improves the performance and raising the memory demands decreases the performance. The performance delta is of approximately the same magnitude as the change in memory demands. The same holds true for moving more of the references to local memory. To the right of the curve, however, we can observe that there may be value in increasing the number of total references if that can be used to increase the percentage of references that are local.

The impact of concurrency is generally intuitive, but points out the need to find high levels of concurrency in the application. That said, 64 threads per processor with a lookahead of 4 seems to be sufficient to saturate the network. Indeed, providing significantly more concurrency (128 threads, lookahead of 8) seems to reduce performance. While this seems counterintuitive, it appears multiple times in the simulations. Our best current understanding indicates that this is a real result that is caused by the extremely non-linear relationship between network delay and offered load. As the concurrency reaches its highest level, the processors can offer dramatically more load than the network can accept.

3.3.3 S-T Connectivity

Many of the S-T connectivity scenarios are very similar to the connected components scenarios from a perspective of how the kernel uses the machine.

[3]As noted in the methodology, this number may be up to 10% too low.

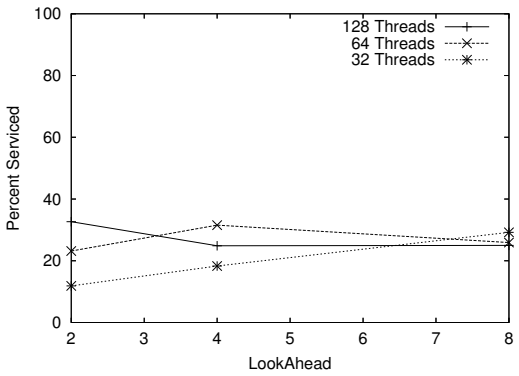


Figure 3: Impacts of concurrency on S-T Connectivity (medium)

That is, the memory reference rate is approximately the same and the percentage of local accesses falls within the ranges graphed in Figure 2(a); however, since the concurrency impacts can be slightly different when the local percentage changes, those are graphed in Figure 3.

In the case graphed in Figure 3, we see a similar behavior to Figure 2(b). If too much concurrency is exposed, network performance suffers. That is, if the offered load to the network is too high, the network delay grows dramatically. The break points differ here because the percentage of the memory request traffic that goes to remote nodes is higher. Generally speaking, at this level of load, we expect as much as a 3× performance hit against the theoretical maximum.

In scenarios where the number of nodes visited by the S-T connectivity algorithm is small, we see an extreme example of a poor match to the Eldorado architecture. Approximately 75% of the instructions reference memory and approximately 10% of those are local. In our design space exploration, we reduced the extreme slightly to an 80% reference rate with 20% of those being local. That leads to a request rate for the network of approximately 320 Mref/s and a local reference rate of approximately 80 Mref/s for a total request rate of 400 Mref/s. Given the data from Table 3, we will assume a cache hit rate of 90%. Simulations have indicated that the network bisection bandwidth is on the order of 67 Mref/s, and so we would expect 21% of the request rate of these applications to be serviced. That translates into approximately a 5× performance penalty from what the peak would be.

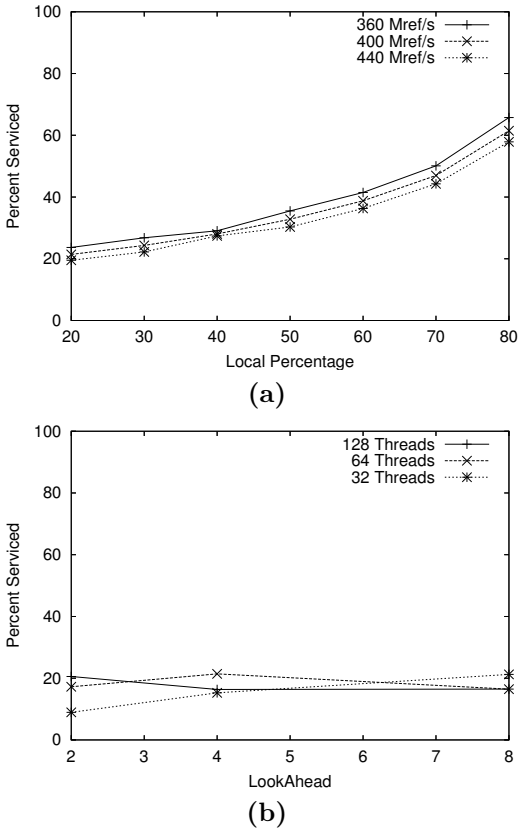


Figure 4: Impacts of **(a)** changes in global request rate and **(b)** changes in concurrency on S-T Connectivity (small)

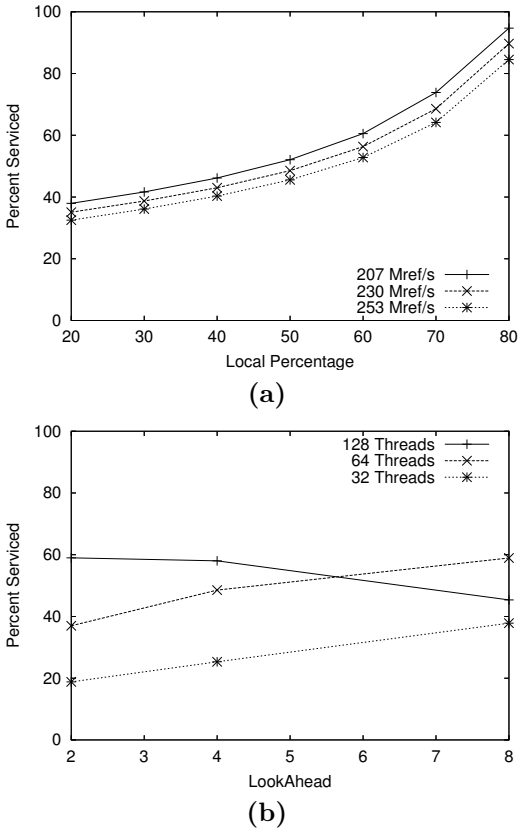


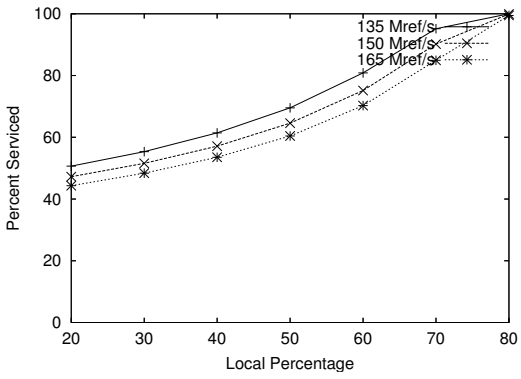
Figure 5: Impacts of **(a)** changes in global request rate and **(b)** changes in concurrency on Sparse Matrix Vector Multiply

The impacts of changes in rate follows similar trends to what was seen with the connected components algorithm; however, the right side of the curve is even steeper. The impact of concurrency on this particular example (Figure 4) are virtually non-existent. The network is so overwhelmed that reducing concurrency only has an impact at the lowest level (32 threads, lookahead of 2).

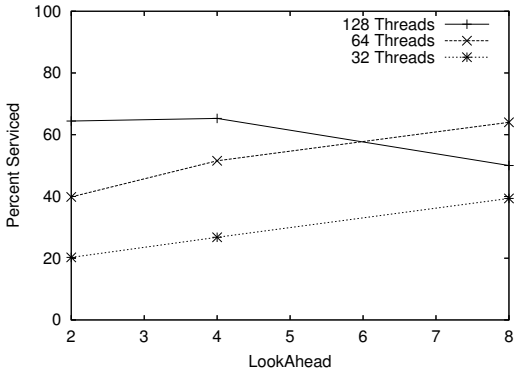
3.3.4 Sparse Matrix Vector Multiply

Sparse matrix vector multiply is one of the particularly good matches between the Eldorado architecture and an algorithm implementation. Only 46% of the instructions reference memory and over 50% of those are local. That leads to a request rate for the network of approximately 108 Mref/s and a local reference rate of approximately 122 Mref/s for a total request rate of 230 Mref/s. Given the data from Table 3, we will assume a cache hit rate of 90%. Simulations have indicated that the network bisection bandwidth is on the order of 67 Mref/s, and so we would expect over 62% of the request rate of these applications to be serviced. That translates into a mere 1.6× performance penalty from what the peak would be.

The access rate data in Figure 5(a) paints a promising picture. With an application that is this well matched to the architecture, relatively small improvements in the application behavior could lead to dramatically better application performance. This is particularly promising given that sparse matrix vector multiply has been successfully implemented on numerous distributed memory architectures. With an Eldorado programming model that should be amenable to distributed memory techniques, it should be possible to achieve all of the potential performance on this application.



(a)



(b)

Figure 6: Impacts of **(a)** changes in global request rate and **(b)** changes in concurrency on Subgraph Isomorphism

The concurrency data in Figure 5(b) is also strikingly different from that of the network bound algorithms. When the network is not significantly constricting remote memory requests, the amount of concurrency exposed becomes critically important. It is possible to see a factor of 1.5 to 2 difference in application performance based on the exposed concurrency.

3.3.5 Subgraph Isomorphism

Subgraph Isomorphism is perhaps the epitome of a good match to the Eldorado architecture, thanks to a filtering step which dominates its performance on our sample data. A mere 30% of the instructions reference memory, but only 34% of those are local. Nonetheless, the total request rate is only 150 Mref/s — the lowest of any of the kernels. This also yields the lowest network request rate (99 Mref/s) and one of the lowest local request rates (51 Mref/s). All of this is accompanied by a cache hit rate of at least 70%. Together, these translate into an algorithm that should run within 1.5× of the peak performance.

The most promising part of the subgraph isomorphism kernel is that it is highly likely that the percentage of memory references that are local could be increased to 60 or 70%, at which point the network could satisfy most of the requests[4]. This could be achieved by moving a single copy of the subgraph to a "local" memory location that can be accessed by all the threads running on a given processor.

3.4 Load Imbalance Issues

One of the major concerns with the Eldorado network is the potential for a network interface to be unable to consume the traffic it receives at a sufficient rate. This is a typical network hot spot scenario that could lead to traffic congestion in the network that affects the performance of the system as a whole. Figure 7 depicts results from two experiments that were used to explore this possibility. The Y-axis is the same percent serviced used in earlier graphs and the X-axis is a metric of overload. Percent overload is the percent of the traffic that is sent to the target set of nodes *beyond their fair share*. As an example, in a 512 node system, each node should send approximately 0.2% of its traffic to every other node. If a node has a *percent overload* of 2 with a target node set of 2 nodes, it will send approximately 1.2% of its traffic to each of those two nodes and will reduce the traffic to the other 510 nodes equally. Figure 7 graphs the impact on the overall average, the source nodes, the target nodes, and the "other" nodes (nodes that are neither source nor target).

Figure 7(a) shows the scenario where a group of 8 source nodes send overload to 2 target nodes[5] . This is a "small, bad application" scenario, where one application runs on a small number of nodes and behaves badly. We can see that neither the average nor the non-participating nodes are particularly affected. Even the source nodes are not dramatically affected until the overload is extreme. The target nodes even seem to have a slight improve-

[4]If the simulated network rates do indeed turn out to be low, this point could be reached with 50% of the memory references being local.

[5]Request rate is fixed at 300Mref/s, local percentage at 50%, cache hit percentage at 90%, and lookahead at 4. These are reasonable application parameters.

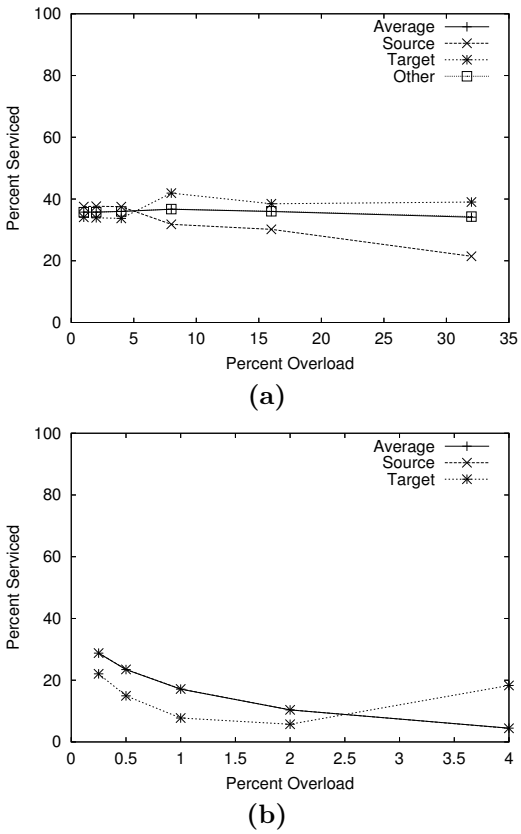


Figure 7: Impacts of **(a)** constrained (8-to-2) overload and **(b)** widespread (510-to-2)overload

ment. Clearly, a single, small job is unlikely to have a particularly detrimental effect on the system as a whole.

In contrast, Figure 7(b) paints a picture where 510 nodes are sources that send overloads to 2 target nodes. Even the 0.25% overload scenario shows a significant degradation. At 0.5% overload, the average service rate has dropped to only 57% of the peak service rate. That is a 1.5× performance hit for an increase of 1 memory reference in 200 going to a pair of nodes. Clearly, a small network hot spot in a code running on a 512 node machine can have dramatic effects.

4 Conclusions

While the Eldorado platform shares many of the characteristics of the MTA-2 platform, it also has dramatic differences that create cause for concern about the overall performance of the platform. These changes include detrimental changes to the memory hierarchy and the network topology – the two most performance critical aspects of the machine. Simultaneously, the processor performance (in terms of clock rate) on the Eldorado platform has been significantly increased.

To assess these concerns, this paper presents simulations of the various components of the system and ties those results together through analysis. Four algorithms were analyzed and statistically simulated at 512 nodes to characterize their interaction with the network and the memory system in Eldorado. The results were promising in that the limitations of the network and DRAM do not seem to impose dramatic constraints on their performance. Two algorithms (subgraph isomorphism and sparse matrix vector multiply) appear to be capable of achieving performance that is only 35% below the peak performance of the processors. The connected components algorithms and the S-T connectivity algorithm do not perform as well, but they are still expected to be within 60-75% of the peak processor performance. While this sounds low, it should translate to performance that is on par with the MTA-2. Thus, we do not see a particularly severe *architectural* limitation to the scaling of the Eldorado platform to 512 nodes.

There are many aspects of scalability that extend beyond just the machine architecture. Issues of thread creation and thread management along with a variety of run-time system issues and compiler issues can put scalability at risk. These issues cannot be directly evaluated for the Eldorado platform and are important considerations for future work.