Shawn Koch

Mark Doughty

ELEC 525

4/23/02

A Simulation: Improving Throughput and Reducing PCI Bus Traffic by Caching Server Requests using a Network Processor with Memory

1 Motivation and Concept

The goal of this project was to show how using a network processor with external memory for caching server requests could potentially provide a significant increase in server throughput and PCI bus bandwidth. Server applications are becoming more important as the internet continues to expand. We expect that the future will continue to bring more web users each year. These users will likely turn to the internet for the latest news updates, for online shopping, for research, and countless other areas of interest. As the number of users a web server must handle increases, server throughput will become increasingly important, especially for servers handling a large number of clients. Therefore, we believe that by replacing the standard NIC with a network processor connected to an external memory used for caching server requests, we can significantly increase the throughput while also benefitting PCI bus bandwidth.

While this idea would benefit a large number of servers, there are some that might not reap benefits as great. Servers that primarily handle streaming media requests or dynamic web pages are not likely to see the maximum benefit our expected results show. We believe that servers dealing with static web pages are those that can expect to see the most improvement in throughput. However, the idea may still be applicable to non-static request handling servers with some modifications. Our simulator model, however, is based on a static

request handling server. Also of note, from a single user or client perspective, there may not be any discernible improvement in server performance. The primary benefit comes from being able to handle more requests in a shorter amount of time, thus allowing the server to increase its overall performance. So while a particular user may not be excited about the concept, websites that consistently see periods of high traffic on any given day would likely be interested in the results presented here.

The idea is to use a network processor connected to a DRAM for caching frequently requested data packets coming from a server. As a request is made, the host machine will seek the requested data from its memory and send the data out to the network processor. The host will also signal the network processor to cache the data in its own memory. The next time a request is made for the same data, the host CPU will not need to send out the data again. Rather, it will simply send out header information for the packet and information detailing the location of the data in the network processor memory as well as the length of the data. The network processor will then concatenate the header with the requested packet data and complete the response by sending it out to the Ethernet. First, this allows the CPU to handle the next request sooner rather than having to prepare the packet again for transmission. Also, since the entire packet is not required to be sent to the network processor again, the traffic over the PCI bus is reduced by the size of the requested packet that could potentially be considerable over enough requests. Thus, we also improve the bandwidth over the PCI bus.

2 Simulator Architecture

In order to test this idea, we have created a simulator for this system using C++. We have modeled the host CPU, host main memory (DRAM), host disk, network processor, network processor external memory (DRAM), PCI bus, packets, and Ethernet. We are particularly interested in measuring the average response time for a request for the system and

the total amount of traffic over the PCI bus. This will give us the percentage improvement of our network processor based simulations over our baseline simulation lacking a network processor. The CPU and NP are not modeled in detail. Rather, we have modeled a processing time for each component which is essentially a constant amount of time added to each request/response pair, and the clock times of each component. The processing time is the amount of expected overhead time to handle the request. The processing times and clock speeds can be changed in the configuration file (see attached sheet listing all parameters, note that listed parameters are default baseline parameters).

The Ethernet is modeled in terms of a request frequency and number of requests. Both are parameters that may be varied. The number of requests is simply the total number of requests that will be sent through the system. The request frequency is the rate at which requests are made on the system in terms of requests per second. The throughput is measured as the amount of time it takes from the initiation of a request until a response for the request is completed. Each request is given a unique identification number in sequential order so that the total number of requests can be monitored. The requests are also given a second ID number that is not unique. If this number is the same as that on another request, it means the requests are for the same data. This is how we determine if the network processor memory contains the request or not.

The PCI bus between the network processor and the host computer is modeled as a 64-bit bus. It is set up as a parameter, but is held as having an 8-byte transfer rate per cycle for our simulations. We monitor the total amount of data (number of bytes) that is transferred over the PCI bus in either direction. Also, we have not modeled any contention for the bus. Data can be traveling in both directions at the same time in our simulator. We would look to modeling the bus more accurately in the future.

Packets are modeled for requests and responses. Request packets are modeled as having a default request size of 1024 bytes. This means all requests require that 1024 bytes be transferred over the PCI bus from the network processor to the host machine. The other major parameter for request packets is the number of different requests. This controls the number of different IDs that are available to requests. These IDs are attached to requests at random. As an example, if the number of different requests is set to 1, the request will always be processed from the network processor’s memory after the initial miss in its memory. The response packets on the other hand are modeled with a minimum and maximum size. The defaults for the sizes are set to 1KB and 20KB respectively. The sizes are also randomly generated for packets. We arrived at these values by visiting several popular websites and looking at the sizes of all objects that loaded from the page. We noticed that most objects were of a smaller size with a median of around 10KB. We believe the websites represent a good sample of where many users and clients would connect. Finally, we have modeled an “In NP memory response size” which is set as default to 1024 bytes. This number is to account for the header information that would be transferred from the host CPU to the network processor if the requested data is found in the NP’s memory. This means that while the final response will indeed consist of the total size of the header and data held in the NP memory, only 1024 bytes will be noted as having passed over the PCI bus for the particular response.

The final main components of our simulator are the different memory units. All memory (disk, main CPU DRAM, external NP DRAM) is modeled the same way. The parameters for the memory units include a total size, a line size, and a latency. The hard disk parameter is set to indicate an infinite size. This means all requests, if not elsewhere, must be present on the hard disk. We have chosen to model our memory replacement as using a least recently used policy. The memory model is essentially a list of request IDs that are held in the

memory. These lists are searched by the system to identify whether a request can be handled from a particular memory unit or not, starting with the NP memory, then the host main memory, and finally the disk. When data is moved to a different memory unit, the available size is checked. If space for the transfer is lacking, the LRU piece of data is removed from the list and the new item is inserted (assuming the item is small enough to fit in the memory unit). The latency is the number of cycles (either CPU or NP) required to access the memory while the line size is the amount of data that is read from the memory "per latency."

The simulator itself controls the "ticking" of the CPU and NP. It is used to keep track of when the NP should "tick" (execute a cycle or work) relative to when the CPU "ticks." The final parameters allowed are primarily for debugging purposes. These can be set to allow certain information to be dumped from the simulator or can be cleared to turn off debug information. If debug information is turned on for a particular simulator component, the simulator will dump the state of the specified unit at the requested simulator "tick" frequency.

3 Experimental Methodology

We ran 26 different configuration files on our simulator that we felt gave a realistic idea of what types of system configurations were possible. Unfortunately, we did not include our disk memory model in any of the simulations mainly due to time constraints in running our simulations with the disk. In order to run all configuration files we wanted to would have required a significant amount of time. In the future we would plan to run the files with the disk included. The lack of disk means that we have forced all requests to exist in the host's main memory. Although we feel that this does not give a completely accurate idea of our system's behavior, it should actually lower our expected improvement since the host never references disk. Therefore we believe our results will be conservatively skewed. We also scaled our parameters by a common factor in order to speed up our simulation time

further. The memory line size and transfer rate across the PCI bus were scaled up. This allowed us to complete our simulations in a reasonable amount of time.

The configuration file attached shows all parameters as well as the default values we chose for our short simulations. Short simulations were run with 50,000 total requests while the four long simulations were run using 250,000 total requests. For the long simulations we varied only the memory size of the DRAM connected to the NP. We expect that adding more memory should always increase throughput since more requested data may be cached and handled by the NP. Similarly, we expect to see a reduction in PCI bus traffic as more memory is added. The baseline system used an NP memory size of 0, while the other long simulations used 1MB, 8MB, and 32MB of memory. It is of note that we do not model any space the network processor requires for firmware, temporary storage, or transmit and receive buffers as described in "Increasing Web Server Throughput with Network Interface Data Caching" (Rixner et al).

The short simulations were used to vary more parameters than simply memory size. Here we also varied clock speed of both the CPU and NP, memory latency, memory line size, memory size, number of different requests, total number of requests, request frequency, processing time, and response time. We chose to vary each parameter one at a time to try and isolate which parameter had the greatest effect on performance. We also ran a few configurations in which several parameters were varied in hopes of creating a best case system configuration.

4 Analysis

Looking at the results for the long simulations, it becomes quite apparent that connecting a larger memory to the NP results in better performance gain. The throughput for the 1MB NP memory case shows a mere 0.18% improvement over the throughput of the baseline system (throughput measured in simulator ticks, 1 simulator tick = 1 CPU tick).

Increasing the NP memory to 8MB gives a 2.71% improvement in throughput while adding a 32MB memory results in a desirable 10.89% improvement (see Fig. 1). The decrease in PCI bus traffic is more substantial in each case. Adding 1MB of NP memory reduces the PCI bus traffic slightly by 1.18% while the 32MB case shows a reduction of almost half, 45.93% (see Fig. 2). The mean request size in the long simulations was 10KB, the total number of different requests was 5000, and the number of total requests sent was 250,000. Our results indicate for the 1MB and 8MB NP memory cases, requested data was likely evicted from the cache before it was requested again. Since the 32MB case is able to hold more requests at once, it is likely that future requests will hit in the cache. Even though the 8MB case did not demonstrate significant improvement over the baseline case in our simulation, 8MB may still provide considerable gains for servers expecting many requests for the same information or for servers that deal in smaller request sizes. Lastly, the 45.93% reduction in PCI bus traffic in the 32MB case is quite encouraging since this means the host CPU would have more bandwidth to communicate with any peripherals attached via the PCI bus as necessary.

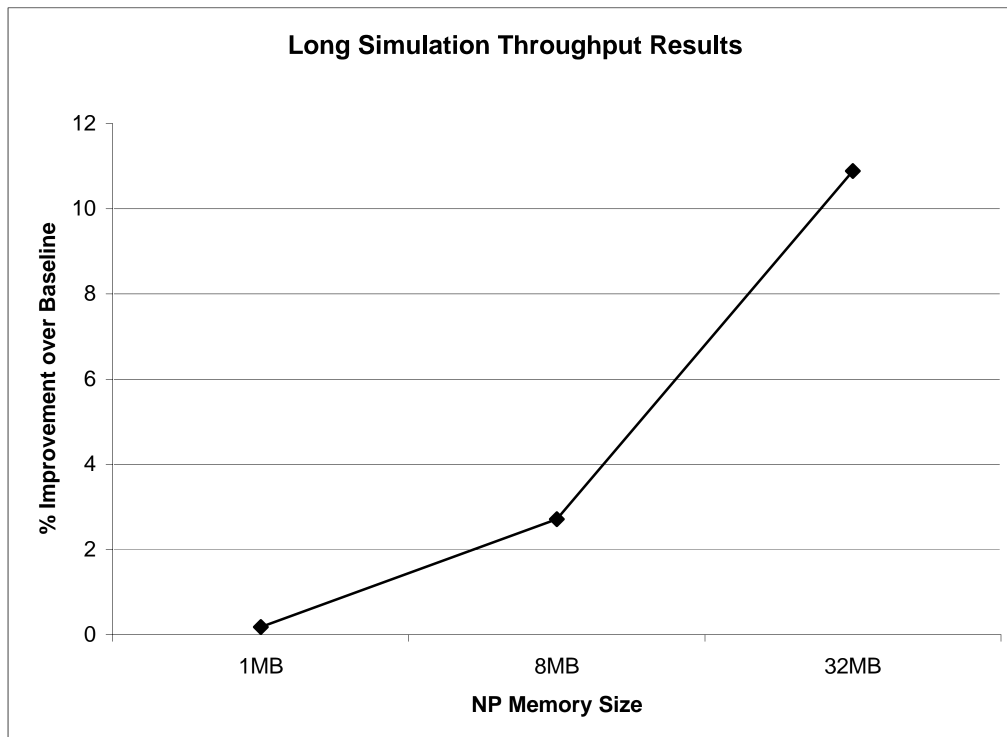

FIG. 1

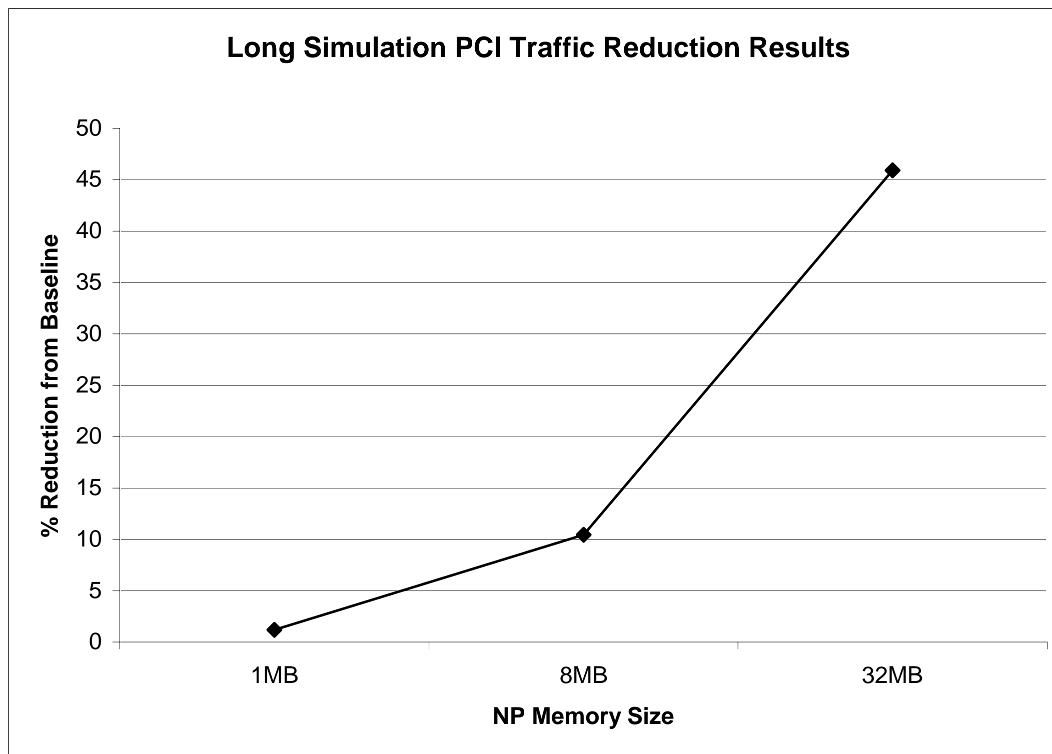

Fig. 2

The short simulations give some interesting results as well. For these simulations, the number of different requests was reduced from 5000 to 1000, so we expect to see an overall improvement in results compared to the long simulations. The 512KB, 1MB, 8MB, 16MB, and 32MB trials yielded throughput improvements of 1.41%, 1.73%, 14.12%, 18.26%, and 18.19% respectively (see Fig. 3). The traffic improvements for the five cases were 4.67%, 9.10%, 55.99%, 72.97%, and 72.97% respectively (see Fig. 4). The zero increase in performance from the 16MB case to the 32MB case is expected. Since the simulations we ran contained 1000 different requests with an average size of 10KB, all the requests should have fit into the 16MB cache. This means increasing to 32MB should not have made any improvement, which our results verified.

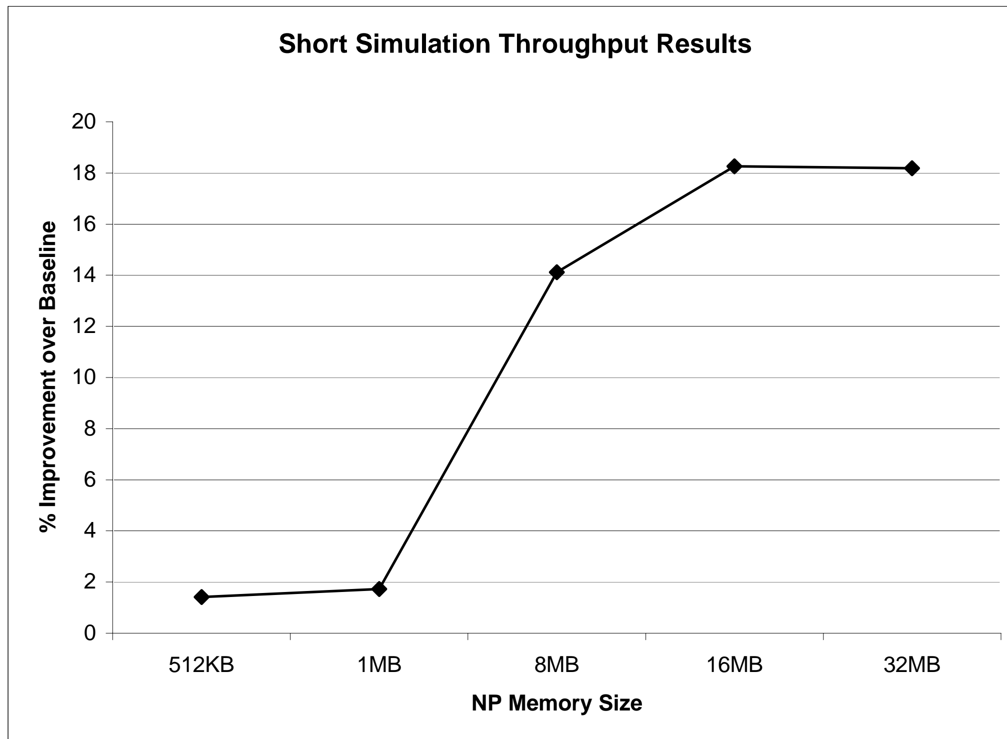

Fig. 3

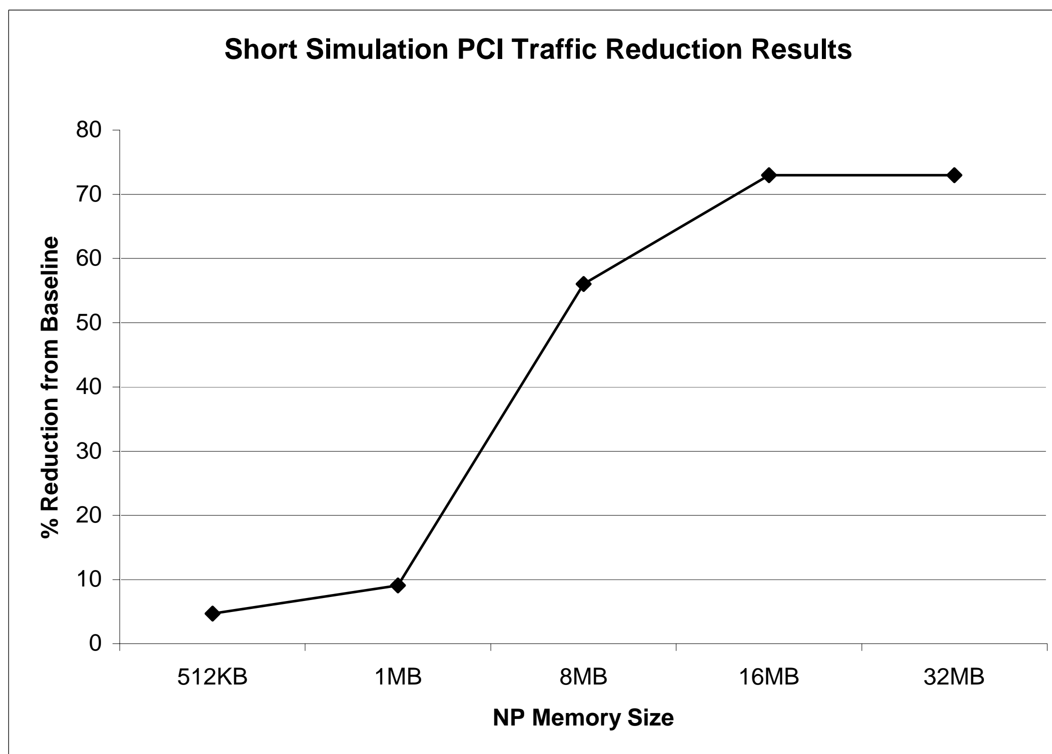

Fig. 4

We ran a group of simulations in which we varied the PCI bus speed. By doubling the transfer rate with no NP memory, we saw a performance increase of 11.08%. Similarly, by halving the transfer rate with no NP memory, we saw a performance loss of 21.87%. This demonstrates the vital importance of bus bandwidth. The addition of an 8MB NP memory in these cases result in 16.72 and 5.77 performance increases respectively for the fast and slow cases. These measurements are relative to the no NP memory, normal bus speed baseline. Notice that the addition of the 8MB NP memory more than makes up for the reduced bus transfer rate.

Simulations in which we varied the number of different requests on the 8MB NP memory machine showed how large an effect on performance this factor may have. Increasing from 1000 different requests to 5000 lowered the throughput improvement from 14.12% to 3.06%. Further increasing the unique requests to 10000 lowered the improvement over 5000 from 3.06% to 1.99%.

As far as NP processor speed, increasing the clock speed from 200MHz (baseline case, value taken from Vitesse IQ2000 NP) to 500MHz showed little throughput improvement, only 0.5. Of final note, lowering the average response size of the 8MB case benefited traffic reduction much more than overall throughput (see Table 1 showing each simulation run, what parameter was varied for the given simulation, the throughput improvement, and traffic reduction).

Table 1

Simulation Name	Parameter Varied From Baseline	% Throughput Improvement	% PCI Traffic Reduction
LongBaseline	Default Basline Settings Used, no NP mem	N/A	N/A
Long1MB	NP memory size set to 1MB	0.18	1.18
Long8MB	NP memory size set to 8MB	2.71	10.45
Long32MB	NP memory size set to 32MB	10.89	45.93
ShortBaseline	Default Basline Settings Used, no NP mem	N/A	N/A
Short512KB	NP memory size set to 512MB	1.41	4.67
Short1MB	NP memory size set to 1MB	1.73	9.10
Short8MB	NP memory size set to 8MB	14.12	55.99
Short16MB	NP memory size set to 16MB	18.26	72.97
Short32MB	NP memory size set to 32MB	18.19	72.97
ShortLowLatBase	Baseline with lower main memory latency	2.31	0.981
Short500MHzNP	8MB NP memory, NP speed = 500MHz	14.6	54.97
Short5000DiffReqs	8MB NP memory, 5000 diff. Request IDs	3.06	12.68
Short10000DiffReqs	8MB NP memory, 10000 diff. Request IDs	1.99	5.91
ShortLongNPLat	8MB NP memory, NP mem. lat. increase	-3.09	55.90
ShortSmallResp	8MB NP memory, Response size shorter	19.67	73.79
ShortBigResp	8MB NP memory, Response size longer	-17.46	-62.12
ShortBigLineSize	Baseline with increased line size for mem.	2.82	2.07
ShortBest	32MB NP memory, line size bigger NP mem	19.45	72.97
ShortFastBus	Baseline with fast bus (2x)	11.08	N/A
ShortSlowBus	Baseline with slow bus (2x)	-21.87	N/A
Short8MBFastBus	8MB NP memory, fast bus (2x)	16.72	55.99
Short8MBSlowBus	8MB NP memory, slow bus (2x)	5.77	55.99

5 Conclusion

We believe that our results show how data caching on a network interface can prove valuable for many web servers. For long simulations we demonstrated throughput improvement of up to 10.89% and for small simulations, an increase in throughput of up to 19.45%. Our hypothesis of adding more memory to achieve even greater gains was supported by our results. As long as the memory is large enough to hold a considerable amount of the request data, server performance will benefit. The large decreases in PCI bus traffic also bolster the argument for implementing this system. However, we do believe our simulation’s accuracy may need to be refined based on the CPU and NP processing times. With accurate

parameters, we believe our simulator represents a fair model of the overall system. Given more time, we would refine our model to include more details of bus contention, processing time, cache replacement policy, and disk interactions. Furthermore, we believe our simulations would only benefit by adding disk interactions to our system. This concept could prove to be a fast, low cost implementation for improving server performance in the near future.