A Power-Performance Analysis of Memory-intensive Parallel Applications on a Manycore Platform

Vishal Gupta [*] Hyesoon Kim Karsten Schwan

College of Computing, Georgia Institute of Technology, Atlanta, GA, USA 30332

{vishal,hyesoon,schwan}@cc.gatech.edu

Abstract

Multicore processors have been effective in scaling application performance by dividing computation among multiple threads running in parallel. However, application performance does not necessarily improve as more cores are added. Application performance can be limited due to multiple bottlenecks including contention for shared resources such as caches and memory.

In this paper, we perform a scalability analysis of parallel applications on a 64-threaded Intel Nehalem-EX based system. We find that applications which scale well on small number of cores, exhibit poor scalability on large number of cores. Using hardware performance counters, we show that many performance limited applications are limited by memory bandwidth on manycore platforms and exhibit improved scalability when provisioned with higher memory bandwidth by varying the number of memory riser cards used. Results show significant energy savings for these memory bandwidth limited applications by regulating the number of threads used and applying dynamic voltage and frequency scaling.

Keywords Manycore, Scalability, Memory bandwidth

1. Introduction

The number of cores in modern processors are rapidly increasing, and this trend is going to continue [1]. However, application performance does not necessarily improve with increasing core count. For example, Figure 1 shows the execution time for SP (scalar pentadiagonal) benchmark from NAS parallel benchmark (NPB) suite [4] for different number of threads. As we see from the figure that execution time of SP first decreases and then starts to increase with more number of threads.

The performance of multi-threaded applications on manycore processors can be limited due to multiple factors. These bottlenecks could be due to the application structure (e.g., serial fractions, critical sections) or due to contention for shared architectural resources (e.g., cache, memory). Previous work has shown that many parallel applications can be

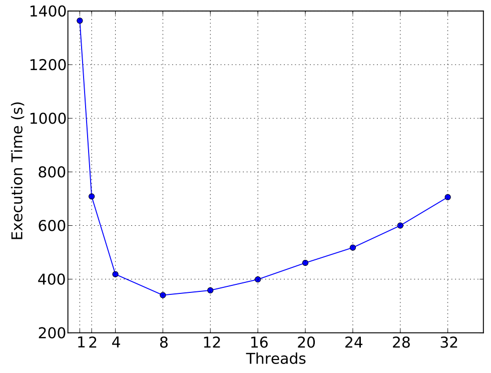


Figure 1. Performance of SP (Scalar Pentadiagonal) benchmark degrades with large number of cores

performance limited by available memory bandwidth [10]. For such applications, once memory bandwidth is saturated, any additional threads spend their time waiting for memory accesses rather than computing. These additional cores can actually deteriorate performance due to queuing delays in memory controllers.

In this paper, we perform a scalability analysis of parallel applications on a 64-threaded Intel Nehalem-EX server. Our evaluation platform consists of four eight-core Nehalem-EX processors with 64GB DDR3 RAM. With the help of hardware performance counters used for profiling, we show that the performance of many performance limited applications is indeed limited by memory bandwidth. We verify these bandwidth limitations by varying the number of memory riser cards used to plug memory in, and, thus varying the available memory bandwidth. Experimental results show that by increasing the memory bandwidth, applications exhibit performance improvement of upto 53%.

Such memory bandwidth limited applications present two opportunities for energy savings. First, by using dynamic concurrency throttling (DCT) and second, by applying dynamic voltage and frequency scaling (DVFS). Our experimental results show that memory bandwidth limited applications can save upto 59% energy using DCT and upto 17% energy using DVFS.

[*] Student author

2. Memory-intensity Analysis

In this section, we describe our methodology to analyze the memory access behavior of parallel applications.

2.1 Measurement Method

For detecting memory bandwidth consumed by an application, we use hardware performance counters. Specifically, we use *offcore_response_0* counter [1] available on Intel Nehalem processor to count total memory accesses by a single-threaded run of the application. This counter measures all the requests that are serviced by the memory subsystem.

Why not LLC-misses?
Last level cache (LLC) misses are often used to measure the memory intensity of an application. However, it does not provide the correct measure of application's memory bandwidth requirement since it only counts on-demand read/write requests which do not hit in the L3 cache. It does not measure the requests sent to the memory subsystem by hardware prefetchers which also compete for memory bandwidth. Therefore, use use offcore_response_0 counter which allows us to measure all the requests including prefetches that go to memory.

2.2 Scalability Prediction

The behavior of memory bandwidth limited applications can be modeled using Equation 1 [10]:

$$T_n = \frac{T_1}{n'} \text{ where } n' = \frac{BW_{total}}{BW_{thread}} \quad (1)$$

$$BW_{thread} = offcore\_response\_0 * cache\_line\_size \quad (2)$$

Here T_n represents application's execution time with n threads and n' denotes the number of threads which saturate memory bandwidth. n' is calculated by diving total available memory bandwidth BW_{total} by bandwidth consumed by a single thread BW_{thread}. We obtain BW_{total} experimentally and BW_{thread} by multiplying cache line size with total number of memory requests measured using performance counters as shown in Equation 2. Application performance scales linearly until it saturates available bandwidth (n' threads). After this point, any increase in number of threads does not result in performance gain. Therefore, we use Equation 1 to predict the optimal number of threads (n') for execution.

To verify the applicability of Equation 1, we use a multithreaded microbenchmark where each thread sequentially accesses (read or write) a large local array. Figure 2 shows the modeled and measured performance of read and write microbenchmarks as a function of number of threads. We can see from the figure that Equation 1 captures the experimental behavior closely. The difference between the measured and modeled values is due to queuing delays in memory controllers under high contention. We also note that read and write benchmarks show significantly different behavior. This is due to the fact each memory write operation causes two memory accesses, first to fetch data into the cache and then second, to write modified data back into memory during write-back. This effectively doubles the bandwidth requirement.

[1] We use any_request and local_dram masks with the counter.

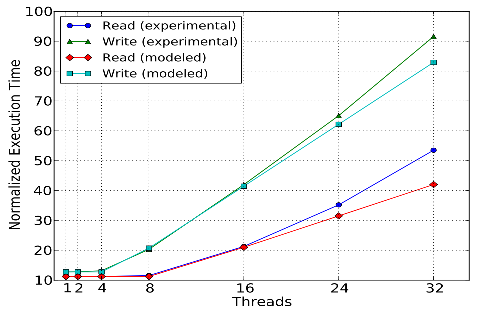


Figure 2. Modeling performance of memory bandwidth limited applications using hardware performance counters

3. Evaluation & Analysis

In this section, we present our experimental results for a scalability analysis of real world parallel applications. Our evaluation platform is a 64-threaded server consisting of four eight-core Intel Nehalem-EX processors with 64GB of DDR3 RAM (system architecture is shown in Figure 3 and configuration details are provided in Table 1). Processors access memory using a memory riser card interface which provides a read bandwidth of 10GBps. By varying the number of riser cards used, we can vary the total amount of memory bandwidth available. All the energy results presented are measured using Wattsup power meter. Our system has idle power consumption of 484W. We use OpenMP implementation of NAS Parallel Benchmarks (NPB) [4] for our evaluation. Input size for all the benchmarks except IS is class C. For IS, we use class D since class C experiments run for a very short period.

Processor	Nehalem-EX
Cores	32
H/W threads	64
Cores per socket	8
CPU Frequency	2.26GHz
LLC size	24MB
Memory	64GB DDR3
DIMMs	8
Memory riser cards	1-4

Table 1. Configuration of the evaluation platform

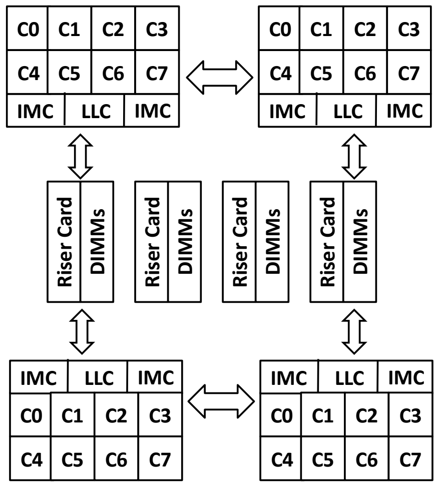


Figure 3. Architecture of our evaluation system

3.1 Performance Analysis

3.1.1 Single Riser Card

We first present results for the configuration when only one memory riser card is used. We run each benchmark by varying the number of threads from zero to 32 in multiples of four. Table 2 shows the number of threads for each application which provides the maximum performance. The table also contains results for the number of threads which minimize energy consumption. As we can see, different applications show peak performance with different number of threads. EP (embarrassingly parallel) benchmark scales well to 32 threads, while MG and SP do not scale beyond 8 cores. Figure 4 shows resultant scalability curves for each application.

	No. of Threads	
	Max. Performance	Min. Energy
BT	32	16
CG	12	12
EP	32	32
FT	20	16
IS	20	16
LU	24	16
MG	8	8
SP	8	8
UA	12	8

Table 2. Optimal number of threads for maximum performance and minimum energy

We use Equation 1 and 2 to predict the optimal number of threads (n') for each application. Table 3 shows the predicted number of threads along with memory bandwidth consumed by each NPB application. The bandwidth is measured using hardware performance counters as described in the previous section. From a comparison of this predicted value with the measured thread count (maximum performance column) in Table 2, we observe that the prediction method provides close results for all the other benchmarks except FT and IS.

	Mem. B/W (GBps)	Predicted Threads (n')
BT	0.23	42
CG	0.65	15
EP	∼0	64
FT	0.17	57
IS	0.31	31
LU	0.44	22
MG	1.2	8
SP	0.88	11
UA	0.76	13

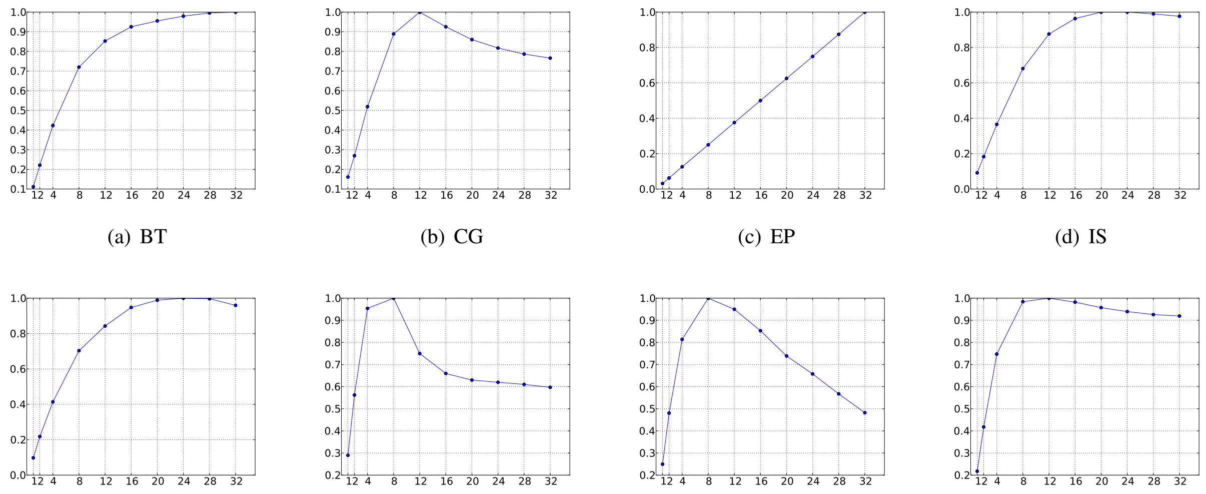
Table 3. Memory bandwidth consumed by a single-threaded run measured using performance counters and predicted optimal number of threads.

FT (fast fourier transform) and IS (integer sort) benchmarks show poor scalability in experiments, while prediction results show them to be highly scalable. This behavior can be explained as follows: The model in Equation 1 assumes that applications have a uniform memory access pattern. However, this is not true for FT and IS which have periodic peaks of memory accesses followed by durations of low memory accesses. Only these peaks saturate memory bandwidth and are not scalable, while the other parts of the computation are scalable. Therefore, a prediction that is based upon average memory bandwidth consumption will provide an incorrect prediction. For correct estimation, the prediction method needs to take these phases into account for applications with large differences in their memory access frequency over time. We plan to explore this as part of our future work.

3.1.2 Multiple Riser Cards

To further explore the impact of memory bandwidth on these benchmarks, we distribute the memory in our system across four riser cards and, thus, effectively quadruple the total memory bandwidth and perform the experiments again. With two or three riser card configurations, our system becomes asymmetric in terms of mapping from quad-socket CPU cores to two or three memory nodes. Therefore, we only report results with four riser cards.

Figure 5 shows the resultant performance improvement for this configuration. As we see from the figure, all the applications except EP show high performance improvement with four riser cards. EP (embarrassingly parallel) does not exercise memory subsystem and does not show any performance improvement with four riser cards. Table **??** lists the



(a) BT (b) CG (c) EP (d) IS

(e) LU (f) MG (g) SP (h) UA

Figure 4. Scalability curve for NPB applications (x-axis = threads (1-32), y-axis = normalized performance (higher is better)).

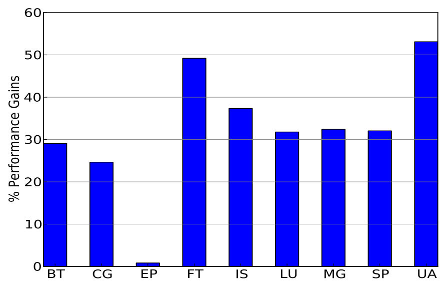
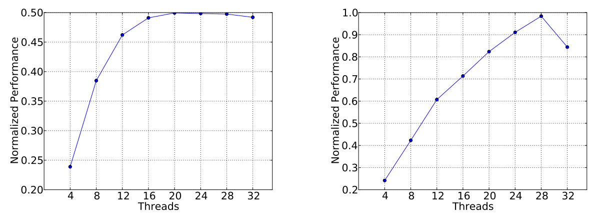


Figure 5. Performance improvement due to increase in memory bandwidth by using multiple memory riser cards.

corresponding optimal number of threads for maximum performance with one and four riser cards. Many applications show improved scalability upto 28-32 threads with four riser cards. For example, Figure 6 shows a comparison of the scalability behavior of FT with two configurations. With four riser cards, FT performance scales to 28 threads which was scalable to only 20 threads with one riser card.



(a) One riser card (b) Four riser cards

Figure 6. Improved scalability in FT due to high memory bandwidth (higher is better).

3.2 Energy Analysis

In this section, we discuss two opportunities to reduce the energy consumption of memory bandwidth limited applications and show experimental results.

3.2.1 Concurrency Throttling

Parallel applications do not benefit from any additional threads once memory bandwidth is saturated. However, these additional threads add to the power consumption. By regulating the number of threads used to run an application, also known as *dynamic concurrency throttling* (DCT), its power consumption can be reduced. Figure 7 shows energy savings for all the NPB applications that can be achieved by using optimal number of threads as compared to running an application with as many threads as CPU cores. The reported results correspond to the number of threads shown for minimum energy in Table 2.

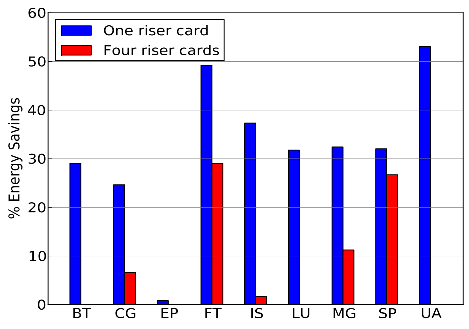


Figure 7. Energy savings using concurrency throttling to use fewer threads than cores available

3.2.2 Voltage and Frequency Scaling

An application limited by memory bandwidth spends most of its time waiting for memory accesses. Therefore, dynamic voltage and frequency scaling (DVFS) can be used to reduce application's power consumption with minimal performance degradation. Figure 8 shows corresponding energy savings of DVFS, i.e., running all the cores at a frequency of 1.06GHz. BT and IS show negative energy savings since DVFS degrades their performance severely, resulting into higher energy consumption. The figure also contains results for a system configuration with idle power of 200W.

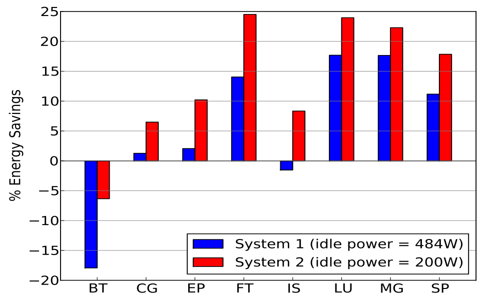


Figure 8. Energy savings by applying DVFS to memory bandwidth limited benchmarks (with 32 threads).

Since our system has high idle power (484W) due to redundant fans, power supplies and other components, this idle power component dominates the overall server power consumption which makes energy saving techniques described above less effective. A system with a lower idle power will benefit more from these techniques. Also, the optimal operating point (i.e., number of threads and CPU frequency) for minimum energy consumption is dependent on this idle power component. A lower idle power component will allow aggressive idling and scaling techniques to be applied.

4. Related Work

Substantial previous work has analyzed the sources of poor scalability in parallel applications. Several techniques have been proposed to execute parallel applications with fewer threads than cores to achieve maximum performance or minimum energy [3, 5, 9, 10]. Previous work has also proposed models to predict the optimum number of threads for an application [2, 7]. However, previous work has either relied on simulators or systems with smaller core count (upto 12) for their evaluation. In comparison, our experimental results are based on a large 64-threaded Intel Nehalem-EX server. Further, studies have been performed to analyze the memory intensity of parallel applications highlighting the bottlenecks in memory subsystem [6, 8]. Our results support their conclusions stressing the importance of regulating the number of threads used for parallel applications.

5. Conclusions & Future Work

In this paper, we performed a scalability analysis of parallel applications on a 64-threaded Intel Nehalem-EX based server. Using hardware performance counters to detect memory subsystem bottlenecks, we showed that many performance limited applications are limited by memory bandwidth. When provisioned with higher memory bandwidth by increasing the number of riser cards, applications show better scalability and performance improvements of upto 53%. We also discussed two opportunities to reduce the energy consumption of memory bandwidth limited application. By regulating the number of threads for execution, upto 59% energy savings and by applying dynamic voltage and frequency scaling upto 17% energy savings can be achieved.

As part of future work, we plan to extend our scalability analysis to other class of applications which may have bottlenecks due to shared resources like caches. We would also like to develop models to predict the performance of application performance in these scenarios.

References

[1] S. Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, New York, NY, USA, 2007. ACM.

[2] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski, and M. Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th ACM PACT '08*, 2008.

[3] S. Imamura, H. Sasaki, N. Fukumoto, K. Inoue, and K. Murakami. Optimizing power-performance trade-off for parallel applications through dynamic core and frequency scaling. In *Proceedings of the RESoLVE'12*, March 2012.

[4] H. Jin, M. Frumkin, and J. Yan. The OpenMP implementation of NAS parallel benchmarks and its performance. *NASA Ames Research Center," Technical Report NAS-99-011*, 1999.

[5] J. Li and J. Martinez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *proceedings of the IEEE 12th HPCA*, pages 77 – 87, feb. 2006.

[6] L. Liu, Z. Li, and A. H. Sameh. Analyzing memory access intensity in parallel programs on multicore. In *Proceedings of the 22nd ICS '08*, New York, NY, USA, 2008. ACM.

[7] R. Moore and B. Childers. Using utility prediction models to dynamically choose program thread counts. In *Proceedings of the IEEE ISPASS*, pages 135 –144, april 2012.

[8] M. Pavlovic, Y. Etsion, and A. Ramirez. On the memory system requirements of future scientific applications: Four case-studies. In *Proceedings of the IEEE IISWC*, nov. 2011.

[9] K. Pusukuri, R. Gupta, and L. Bhuyan. Thread reinforcer: Dynamically determining number of threads via os level monitoring. In *Proceedings of the IEEE IISWC '11*, nov. 2011.

[10] M. A. Suleman, M. K. Qureshi, and Y. N. Patt. Feedback-driven threading: power-efficient and high-performance execution of multi-threaded workloads on CMPs. In *Proceedings of the ASPLOS XIII*, New York, NY, USA, 2008. ACM.