Predicting the Performance Impact of Increasing Memory Bandwidth for Scientific Workflows

Nelson Mimura Gonzalez, Jose Brunheroto, Fausto Artico, and Yoonho Park
IBM T. J. Watson Research Center, New York, USA – E-mail: {nmimura, brunhe, fausto.artico, yoonho}@us.ibm.com
Tereza Carvalho
Escola Politécnica, University of São Paulo, Brazil – E-mail: carvalho@larc.usp.br
Charles Christian Miers, Mauricio Aronne Pillon, and Guilherme Piegas Koslovski
Santa Catarina State University, Joinville, Brazil – E-mail: {charles.miers, mauricio.pillon, guilherme.koslovski}@udesc.br

***Abstract*—The disparity between the bandwidth provided by modern processors and by the main memory led to the issue known as memory wall, in which application performance becomes completely bound by memory speed. Newer technologies are trying to increase memory bandwidth to address this issue, but the fact is that the effects of increasing bandwidth to application performance still lack exploration. This paper investigates these effects for scientific workflows focusing on the definition of a performance model and on the execution of experiments to validate the rationale for the model. The main contribution is based on two observations: memory bound applications benefit more from an increase to memory bandwidth, and the effects of improving bandwidth for a particular application gradually diminish as bandwidth is increased.**

I. Introduction

Current main memory is able to provide less than 10% of the bandwidth of high-end multi-core processors [1], which explains why for so many applications, especially scientific workflows, performance is bound by memory resources, either bandwidth or latency [2]. Moreover, intensive use of the memory due to multiple concurrent requests leads to an increase in contention latency [3], culminating in the infamous memory wall [4] in which application performance becomes completely dependent on memory speed.

Newer technologies such as the Hybrid Memory Cube (HMC) and High Bandwidth Memory (HBM) aim at providing much higher bandwidths than traditional DIMMs in order to address this memory wall. However, the benefit of adopting these technologies largely depend on the application being optimized. For some cases the idle latency will probably increase due to the higher complexity of these devices [5]. On the other hand, if applications present sufficient memory-level parallelism, shared-resource contention among concurrent requests should improve due to higher bandwidth. The fact is, the performance effects of improving bandwidth still lack exploration.

This paper presents an investigation on the effects of improving memory bandwidth to scientific workflows, then it defines a performance model that can be used to predict the impact of increasing bandwidth for a particular application. Finally, this paper presents a set of experiments conducted to validate the model and its rationale. The main contribution of this paper is based on two key observations: *(i)* Memory bound applications benefit more from an increase to available memory bandwidth than applications whose performance is not constrained by memory latency and/or bandwidth; and *(ii)* The effects of improving memory bandwidth for an application gradually diminish as the available bandwidth is increased, to the point that having more bandwidth does not affect application performance at all.

Other contributions of this paper include: *(a)* The definition of the $\eta \times F$ curve (Sections III-A and III-B) that correlates the fraction of total bandwidth initially used by an application (F) and how efficiently (η) an increase to total bandwidth is converted into performance improvement; *(b)* The definition of a model and a numerical iterative method (Section III-C) to determine the performance improvement of an application after increasing the available memory bandwidth; *(c)* The memory bandwidth experiments (Sections IV and V) that validate the two observations for a new set of applications executed in a newer environment and then compared to the values obtained by using the experimental results from Radulovic et. al. [5]; and *(d)* The methodology used in the experiments (Section VI), which can be reproduced to verify the results and to generate new data sets to be used for performance prediction purposes.

The remainder of this paper is organized as follows: Section II presents the background concepts and related work; Section III presents the performance model, including the $\eta \times F$ curve and the iterative method to predict the performance improvement; Section IV presents the methodology adopted for the experiments; Section V presents the experimental results used to validate the initial observations and the performance model; Section VI presents a comparison of these experimental results to the ones generated using the literature; and Section VII presents the conclusion and future work.

II. Background & Related Work

"The first hardware-related [design] issue is memory bandwidth: the benchmarks suggest that it is not keeping up with CPU speed. [...] If memory bandwidth does not improve dramatically in future machines, some classes of applications may be limited by memory performance" [6]. This excerpt, extracted from a 1990 publication by John Ousterhout, reveals

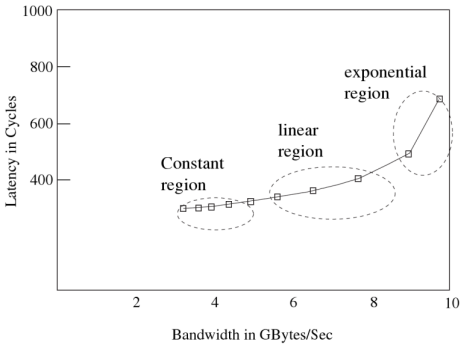

Fig. 1: Memory latency x bandwidth curve. Latency exponentially increases as the amount of bandwidth used by the application approximates the maximum sustained bandwidth. Extracted from [3] and [10].

that the disparity between CPU and memory speed was already a concern decades ago. Likewise, Richard Sites stated in 1996 that *"chips are largely able to execute code faster than we can feed them with instructions and data"* [7]. This increasing gap leveraged the relevance of the memory subsystem and its hierarchy, rendering it as the main component that significantly affects overall system performance [8].

The memory wall issue was formalized by Wulf and McKee in 1994 [4]. The rationale is based on the fact that the rate of improvement in microprocessor speed vastly exceeded the rate of improvement in DRAM memory speed over time. Average memory access time essentially depends on cache and DRAM access times [1]. Consider that cache speed matches processor's performance (e.g., on-chip cache). Even assuming a perfect cache, wherein misses are only due to accessing previously unreferenced locations, the miss rate is small but not zero. Consequently, as cache and DRAM access times diverge, the average memory access time will grow and system performance will degrade, thus hitting the infamous wall. For instance, a program with 33% of its instructions referencing memory implies on average a memory access every three instructions. The wall is hit in this case if the average memory access time exceeds three instruction times. At this point the performance is completely dependent upon memory speed, hence having a faster processor would not affect wall-clock time to execute this program under these circumstances.

Performance gap between processor and memory since 1980 has increased [1], [9]. Processor performance observed an increase of 25% per year until 1986, then 52% until 2000, then 20% between 2000 and 2005, and no change in performance (per-core) thenceforth. In contrast, memory had an overall improvement of 7% per year in terms of latency. After 2005 processor performance on a per-core basis stopped improving. Meanwhile, memory continued to slowly improve but not nearly enough to substantially reduce the gap.

DRAM main memory is able to provide less than 10% of the bandwidth of high-end multi-core processors [1]. Originally, the memory wall issue was defined in terms of latency, not bandwidth [5]. However, there is an intrinsic relationship between the two. Jacob [3] and Srinivasan [10] present a curve correlating sustained memory bandwidth and the corresponding average latency per request. An example of such curve is presented in Figure 1. The maximum sustained bandwidth of the system (10 GB/s in this example) is the maximum bandwidth that could be effectively used by an application, generally between 65% and 75% of the theoretical peak [3]. Essentially, this curve reveals the access time cost to use the available bandwidth. Jacob and Srinivasan discretize three regions from this curve:

a) Constant region: Latency is nearly constant up to 40% of sustained bandwidth. In this region the average latency is close to the idle latency of the system. Performance is not limited by memory bandwidth, either because the application is not memory bound [11] or due to abundance of resources.

b) Linear region: Latency increases almost linearly with bandwidth demand in the region comprising 40% to 80% of sustained bandwidth. Average latency starts to increase due to contention overhead introduced by numerous memory requests. Performance degradation starts to be observed and the application is considered to be moderately memory bound.

c) Exponential region: Latency rapidly increases and can be many times the idle latency when bandwidth demand is in the region between 80% and 100% of sustained bandwidth. System performance clearly is limited by available memory bandwidth and the application is completely memory bound.

DRAM latency is subdivided in two components [10]: *idle latency*, the round trip time for a memory request with no other concurrent requests to the memory controller; and *contention latency*, the overhead due to other pending requests. Ideally, applications should operate in the constant or at least linear region, otherwise contention drastically increases whilst application performance decreases and becomes memory bound.

Several technologies were proposed to address the memory wall, such as Intelligent RAM [9], Rambus Direct DRAM [12], and Double Data Rate (DDR) devices. Contemporary advancements include 3D-stacking techniques to support much higher bandwidths than traditional DIMMs, such as the Hybrid Memory Cube (HMC) [13] and the High Bandwidth Memory (HBM) [14]. The HMC should be able to deliver up to $15x$ the bandwidth of a regular DRAM, with a per-device bandwidth of 480 GB/s. Similarly, HBM maximum bandwidth reaches 256 GB/s and it can combined to other DIMMs.

These memory technology trends indicate that high-performance DRAM has always strived to provide higher bandwidth, not necessarily reducing latency. Patterson [15] shows that from 1980 to 2000 memory bandwidth observed an improvement of $123x$ while latency had a much smaller $4x$ improvement. One of the reasons is the increase in number of transistors per chip (Moore's law), benefiting bandwidth but leading to larger die sizes (e.g., 35 mm^2 DRAMs to 204 mm^2 DDR SDRAMs). The distance sets an inherent lower bound to latency, and delays on long word and bit lines represent the largest part of row access time of a DRAM.

Nevertheless, Radulovic et al. [5] point that higher bandwidth may lower average latency if the applications offer sufficient memory-level parallelism (MLP), which is related to the number of outstanding cache misses that can

be generated and executed by overlapping multiple off-chip accesses [16] (including instruction fetches, loads, and prefetches). Radulovic et al. show that 3D devices do not change the idle-system memory latency – in fact, they will probably increase baseline latency due to higher complexity. On the other hand, full-system memory latency considering shared-resource contention among concurrent memory requests should drastically improve due to the higher bandwidth, according to the bandwidth-latency curve and its three regions.

III. Memory Bandwidth Model

A. Curve $\eta \times F$

The core rationale for the proposed memory bandwidth model is based on the fact initially observed by Radulovic et al. [5] and emphasized here: memory bound applications benefit more from an increase in total available bandwidth than applications that are not memory bound. This means that there is a proportional correlation between the amount of effective bandwidth B originally used by the application and the observed improvement I_B in this effective bandwidth after increasing total available bandwidth.

The model starts with the definition of an efficiency metric η as the ratio between I_B and the increase I_S in available sustained bandwidth obtained after incrementing the total bandwidth, as defined by $\eta = \frac{I_B}{I_S}$. This metric represents how effectively the increase in sustained bandwidth (fraction of the total theoretical peak) is converted into application performance in the form of available bandwidth for memory operations and memory bound regions. A description of the variables is presented in Table 1. Parameters without the prime symbol correspond to values obtained before the increase in the memory bandwidth, while parameters with the prime symbol correspond to values obtained after the increase.

Table 1: Variables considered in the calculations.

	Description	Formula
B	**Effective** memory bandwidth used by application *before* increase in memory bandwidth.	*measured*
S	**Total sustained** bandwidth measured *before* increase in memory bandwidth.	B
F	**Fraction** of sustained bandwidth used by application *before* increase in memory bandwidth.	B/S
B'	**Effective** memory bandwidth used by application *after* increase in memory bandwidth.	*measured*
S'	**Total sustained** bandwidth measured *after* increase in memory bandwidth.	B'
F'	**Fraction** of sustained bandwidth used by application *after* increase in memory bandwidth.	B'/S'
I_B	**Increase in effective** memory bandwidth due to increase in total available memory bandwidth.	$\frac{B'-B}{B}$
I_S	**Increase in total sustained** memory bandwidth due to increase in total available memory bandwidth.	$\frac{S'-S}{S}$

For exemplifying this metric, Table 2 analysis the results obtained by Radulovic et al. [5] with different applications. These results reinforce the argument for correlating the bandwidth originally used by an application and the performance improvement obtained after increasing available memory bandwidth. For example, QE shows a relatively high initial bandwidth utilization ($B = 35.2$ GB/s, a fraction $F = 65.0\%$ of available bandwidth) and after increasing the clocks the effective bandwidth used by the application improves by $I_B = 8.6\%$, representing an efficiency of $\eta = 58.5\%$. On the other hand, GROMACS shows a lower initial bandwidth utilization ($B = 7.0$ GB/s, a fraction $F = 13.0\%$ of available bandwidth) and after increasing the clocks the bandwidth used by the application improves by only $I_B = 1.6\%$, leading to a much lower efficiency of $\eta = 10.9\%$.

Table 2: Metrics calculated for the experimental results obtained by Radulovic et al. [5].

Application	B	F	B'	F'	I_B	η
STREAM	54.1 GB/s	100.0%	62.1 GB/s	100.0%	14.7%	100.0%
QE	35.2 GB/s	65.0%	38.2 GB/s	61.5%	8.6%	58.5%
ALYA	31.1 GB/s	57.4%	32.2 GB/s	51.9%	3.7%	25.2%
GROMACS	7.0 GB/s	13.0%	7.1 GB/s	11.5%	1.6%	10.9%
NAMD	3.7 GB/s	6.8%	3.7 GB/s	5.9%	0.3%	2.0%

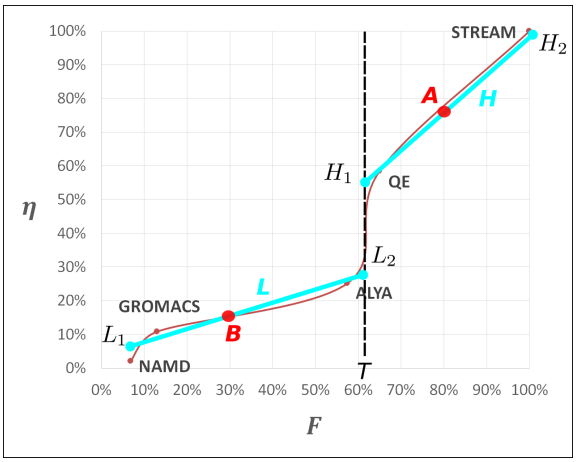

Fig. 2: Curve $\eta \times F$ showing the efficiency factor η as a function of the fraction F of sustained bandwidth originally used by the application and linearization of each region (H and L). Applications with higher F demonstrate a better efficiency to convert bandwidth increase into application performance. Regions H and L are separated by an abrupt slope marked by point T, representing the threshold between memory bound and non memory bound applications. Finally, A and B are examples of applications with memory bound application and not, respectively.

This correlation becomes even clearer after plotting these variables as depicted in Figure 2. The figure shows the efficiency factor η as a function of the initial fraction F of available bandwidth used by the application. The curve is obtained by interpolating the data points highlighted in Table 2. Applications with comparably high initial bandwidth utilization (thus higher F), such as QE and ALYA, exhibit a superior efficiency factor than applications with lower F, such as GROMACS and NAMD. Again, this implies that applications that are completely or at least partially memory bound benefit more from an increase in available memory bandwidth than applications that are not memory bound.

B. Regions

Two main regions can be distinguished in the curve. One region, comprising F between 65% and 100%, represents the memory bound applications with η between 60% and 100%. The other region, comprising F between 15% and 50%, represents non memory bound applications with a low efficiency η, between 2% and 20%. These regions can be associated to the memory regions identified by Jacob [3] and Srinivasan [10]. The region with high η corresponds to their linear and exponential regions (see Figure 1), with bandwidth utilization higher than 50% in which contention plays a key role in application performance. Conversely, the region with low η corresponds to the constant region, wherein application performance is not limited by memory bandwidth.

As observed by Radulovic et al. [5], improving memory bandwidth does not affect performance for applications in the constant region as these applications do not suffer from contention latency penalties. On the other hand, if the application is in the linear or exponential memory regions, a significant portion of latency comes from collisions from concurrent memory requests, consequently upgrading bandwidth may reduce contention and thus improve performance.

Formally, region H, delimited by points H_1 and H_2, represents the applications with high memory bandwidth utilization, or applications in the memory bound region. In contrast, region L, delimited by points L_1 and L_2, represents the applications with low memory bandwidth utilization. The line defined by T ($F \approx 60\%$ in this example) is the threshold that distinguishes memory bound applications, representing the transition point from the constant memory region to the linear and then exponential regions from Jacob and Srinivasan. These regions can be described as follows:

$$\eta = \begin{cases} \dfrac{H_{2_y} - H_{1_y}}{H_{2_x} - H_{1_x}} \cdot (F - H_{2_x}) + H_{2_y}, & T < F \leq 1 \\ \\ \dfrac{L_{2_y} - L_{1_y}}{L_{2_x} - L_{1_x}} \cdot (F - L_{1_x}) + L_{1_y}, & 0 \leq F \leq T \end{cases}$$

Assuming that $H_2 = 1$, $L_1 = 0$, and $H_{1_x} = L_{2_x} = T$:

$$\eta = \begin{cases} 1 - \dfrac{1 - H_T}{1 - T} \cdot (1 - F), & T < F \leq 1 \\ \\ \dfrac{L_T}{T} \cdot F, & 0 \leq F \leq T \end{cases} \tag{1}$$

Where L_T and H_T are the values of η immediately before and after the threshold T, respectively. These expressions can be used to calculate η for a particular value of F. For the exemplary point A, $A_x = 0.80$, $T = 0.60$, and $H_T = 0.55$. Calculating the efficiency factor using the first expression of Equation 1 results in $\eta_A = 0.78$. Contrastingly, for point B, $B_x = 0.30$ and $L_T = 0.30$, thus $\eta_B = 0.15$. This means that for application A 78% of an increase to total available bandwidth is translated into application effective bandwidth, while for B only 15% of this increase is transformed into application performance.

These curves and the threshold T define a clear transition point that distinguishes applications that are memory bound. The expressions defined in Equation 1 provide a method to determine the efficiency η with which an increase to memory bandwidth is translated into application performance. Moreover, the abrupt slope around T shows that after this threshold the effect of increasing bandwidth becomes much lower, which reflects the experiments and conclusions from Jacob and Srinivasan and their memory regions.

C. Discretization and Iterative Method

Using the results for application A from Figure 2 ($F = 0.80$, $\eta_A = 0.78$), suppose that a new increase factor is applied to memory bandwidth, doubling the available sustained memory bandwidth. Applying the efficiency factor, this means that the 78% of this increase is translated to effective bandwidth, thus $I'_B = 0.78$. Using the notation B' and B'' to indicate the application bandwidth before and after this new increase, this means that $B'' = B' + 0.78$ and $B' = 1.78B'$. However, the new sustained bandwidth is $S'' = 2S'$, and the new fraction F'' of sustained bandwidth can be generalized as:

$$F'' = \frac{B''}{S''} = \frac{B' + I'_B B'}{S' + I'_S S'} = \frac{(1 + I'_B)}{(1 + I'_S)} \cdot F' \tag{2}$$

The final expression from Equation 2 shows that except for STREAM, in which $I_B = I_S$ as $\eta = 1$, every time an increase is applied to sustained memory bandwidth the application will absorb just a part of it and, consequently, the fraction of the total available bandwidth gradually diminishes ($F'' < F'$). Moreover, this also shows that calculating the increase in effective bandwidth used by the application in a single step is not correct, as the efficiency factor is a function of the fraction F which continuously varies as the increase is applied to the application. In this sense, a simple experiment to show this discrepancy is to divide the calculation above in two steps, considering two increases of $I'_S = \sqrt{2} - 1 = 0.41$ instead of applying a single increase of $I'_S = 1$.

(0) $F' = 0.80, \eta = 0.78$
(1) $I'_S = 0.41 \rightarrow I'_B = 0.32 \therefore B'' = 1.32B'$
$F'' = 0.75 \rightarrow \eta = 0.70$
(2) $I'_S = 0.41 \rightarrow I'_B = 0.29 \therefore B'' = 1.70B'$
$F'' = 0.69$

Note that not only the final fraction F'' differs from the first calculation, but also the final value of B''. This suggests that the calculation should be done in parts, and the smaller the increments the more precise the calculation should be. It is possible to control these increments by gradually increasing the sustained bandwidth value. Let S_F be the final value for the sustained bandwidth, S_i the intermediate values, and α

the increment factor. Breaking the calculation in two steps, for example, leads to the following:

$$\begin{aligned} S_0 &= S' \\ S_1 &= S_0 + S_0 \cdot \alpha \quad &= S_0 \cdot (1+\alpha) \\ S_2 &= S_1 + S_1 \cdot \alpha \quad &= S_1 \cdot (1+\alpha) \quad &= S_0 \cdot (1+\alpha)^2 \\ &= S_F \end{aligned}$$

This expression can be extended for k increments of α:

$$\begin{gathered} S_i = S_0 \cdot (1+\alpha)^i,\ 1 < i < k \\ S_0 = S',\ S_k = S_F \\ \alpha = \sqrt[k]{\frac{S_F}{S_0}} - 1 \end{gathered} \tag{3}$$

For each iteration i the value of S_i is incremented by α, consequently $I_S = \alpha$. Applying the currently known efficiency factor, η_{i-1}, it is possible to calculate the new value for B_i and then F_i using expressions from Equation 3:

$$\begin{gathered} B_i = \alpha \cdot \eta_{i-1} \cdot (B_{i-1} + 1) \\ F_i = \frac{B_i}{S_i} = \frac{\alpha \cdot \eta_{i-1} \cdot (B_{i-1}+1)}{S_0 \cdot (1+\alpha)^i} \end{gathered} \tag{4}$$

The expressions from Equation 4 calculate the value of the effective bandwidth B used by the application and its representative fraction F for iteration i. The value of F_i is then used to obtain the new value for η_i, using the expressions from Equation 1. A summary of all these steps is presented in Algorithm 1. After completing the final loop, B_k represents the final effective bandwidth. The overall speedup is calculated by comparing this number to the initial value B_0. Moreover, the final value for the fraction F_k represents the new point in the curve that the application is located at.

Algorithm 1 Iterative method to compute B.

1: $B_0, S_0, S \leftarrow$ *bandwidth values*
2: $F_0 \leftarrow$ Table 1
3: $\eta_0 \leftarrow$ Equation 1
4: $k \leftarrow$ *number of iterations*
5: $\alpha \leftarrow$ Equation 3
6: **for** i=1 **to** k **do**
7: $S_i \leftarrow$ Equation 3
8: B_i, $F_i \leftarrow$ Equation 4, *using* η_{i-1}
9: $\eta_i \leftarrow$ Equation 1, *using* F_i
10: **end for**

A visual example of this dislocation over the curve is observed in Figure 3. This example is based on the numerical results of applying $I_S = 1$ in two steps ($\alpha = 0.41, k = 2$). The initial value of F is 0.80, then it becomes 0.75 after the first iteration, and 0.69 after the second one. Visually, this is represented by dislocating the point A to the left on the X-axis (F). Its final state is represented by point A'. This effect corroborates the observation regarding the diminishing returns of applying a speedup factor to memory bandwidth. Given

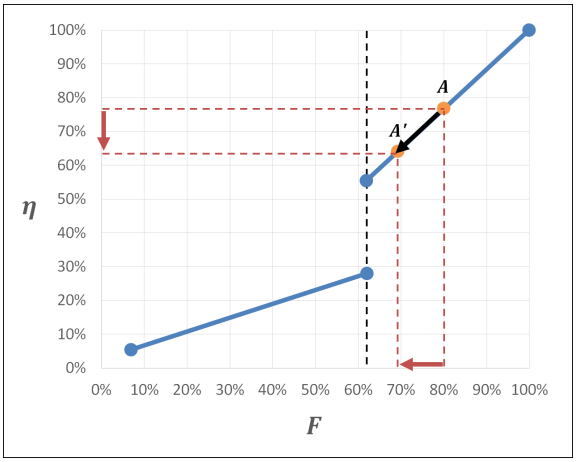

Fig. 3: Example of the dislocation over the curve.

the same application and infrastructure (run time) properties, the effects of increasing bandwidth gradually decrease to the point that having more available bandwidth do not improve application performance at all (i.e., application reaches a point in the curve with very small η, requiring a very large increase factor to yield a substantial I_B). This corresponds to the application reaching the constant memory region from Jacob [3] and Srinivasan [10].

IV. Experimental Methodology

Section III presented the memory bandwidth model proposed in this work. The model was built based on the concepts exposed in Section II and using the experimental results obtained by Radulovic et al. [5]. The next step of this investigation was to validate the model and the assumptions by executing experiments on a new hardware and software platform to observe the same patterns and behaviors. This section presents the methodology used for these experiments, including the experimental platform (IV-A), the method to vary bandwidth (IV-B), the applications selected for testing (IV-C), and the tools developed for analysis (IV-D).

A. Platform

The experimental platform is based on the IBM Power System S822LC model 8335-GTA (codename **Firestone**) [17]. Sixteen nodes were available for the tests, but only one node was used per test. Each node contains two POWER8 sockets at 3.5 GHz with ten cores each and eight hardware threads per core, totalling 160 hardware threads per node. Moreover, each node has 512 GB of main memory subdivided in eight memory risers, as illustrated by Figure 4. The theoretical peak bandwidth between a socket and a riser is 28.8 GB/s.

B. Bandwidth Variation

The methodology used to vary bandwidth for our experiments is different from Radulovic et al. [5]. They varied bandwidth by changing the memory clocks (from 1066 MHz to 1333 MHz). In our case the bandwidth was varied by physically removing memory risers, leading to different bandwidth

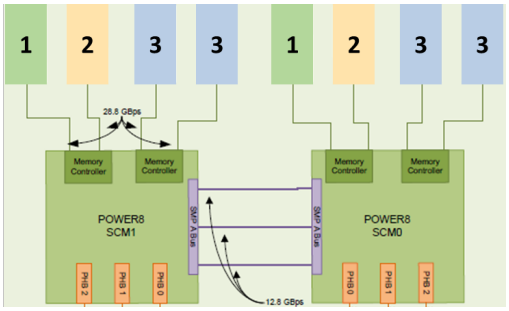

Fig. 4: Detail of Firestone node showing memory risers connected to two POWER8 sockets. Extracted from [17].

configurations. Risers are organized in three groups: #1, #2, and #3 (see Figure 4). Risers #1 must always be present and removal must start from group #3, then group #2. All risers from the same group must be removed, leading to the possible configurations listed in Table 3. The experimental results presented in this paper correspond to the transition from configuration *II* to *I*, going from half to full bandwidth.

Table 3: Possible configurations for memory risers.

	Configuration	Total Peak	Relative
I	#1 + #2 + #3	230.4 GB/s	100%
II	#1 + #2	115.2 GB/s	50%
III	#1	57.6 GB/s	25%

C. Applications

Four applications were selected: STREAM [18], GTC-P [19], HPCG [20], and MILC [21]. STREAM is a well known benchmark for measuring sustained memory bandwidth. The others are part of the applications and benchmarks for the Crossroads/NERSC-9 procurement (APEX), thus representing the applications that will be executed or used to evaluate future systems and architectures. GTC-P, HPCG, and MILC were executed with small problem sizes to fit a single node. STREAM was executed to yield results relative to an array size of one billion elements each (24 GB of total memory utilization). All applications were compiled and executed with MPI support. Memory and MPI process affinity was guaranteed via *rankfiles*, mapping each MPI process to a single core. This allows the execution of experiments with different process/core configurations and also different memory bandwidths. The final results presented in this paper are relative to runs with 20 MPI processes to reach maximum bandwidth utilization, except for MILC that was executed with 16 cores (1 MPI process per core).

D. Tools

Sustained bandwidth S can be easily measured by STREAM. The instrumentation tools provided by the Barcelona Supercomputing Center (BSC) were used to measure the effective bandwidth B used by applications. In particular, Extrae was used to collect information from three POWER8 performance counters: `PM_MEM_READ`: Loads from memory; `PM_MEM_PREF`: Prefetches from memory; and `PM_L3_CO_MEM`: Cast-outs from L3 to memory. Results were processed and grouped using intervals of two seconds, although data were collected with a much higher resolution (μs and ns), generating very large trace files (larger than 50 GB).

V. Results

The results for STREAM, GTC-P, HPCG and MILC are decpited in Figures 5, 6, 7 and 8, respectively. In Figure 5 we observe the results running STREAM using full and half risers. Each graph shows the effective read, write, and total bandwidth. The X-axis represents the interval numbers, each interval being two seconds long. For all applications the read bandwidth is higher than the write bandwidth, and the total bandwidth is the sum of both components.

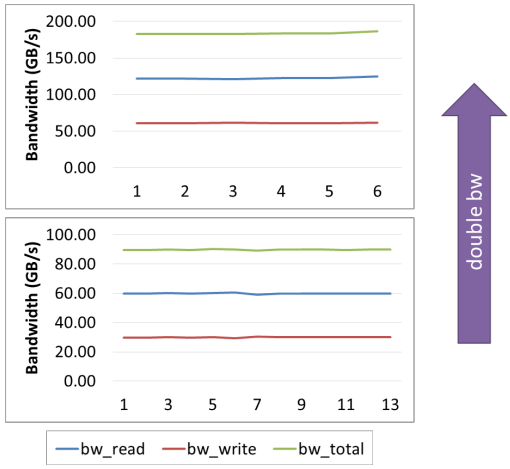

Fig. 5: Results for STREAM using full and half risers.

A summary of these results is presented in first line of Table 4. For full risers configuration the theoretical peak bandwidth is 230.40 GB/s and the sustained bandwidth S measured by STREAM was 183.84 GB/s, around 80% of the peak (consistent to the observations from Jacob [3]). For half risers the theoretical peak is 115.20 GB/s and the value measured by STREAM was 89.77 GB/s, around 78% of the peak. The increase I_S in sustained bandwidth is 104.79%, which is a little bit over 100% that would represent doubling the bandwidth. This shows that removing the memory risers effectively changes the memory bandwidth available to applications. STREAM was also used to calibrate the instrumentation tool developed for collecting the performance counters for the other applications. The discrepancy between the results reported by STREAM and the values calculated based on the collected metrics is lower than 5%.

GTC-P (Figure 6) uses a very small portion of memory, with a total of 15.74 GB/s for half risers (17.53% of sustained bandwidth) and 16.36 GB/s for full risers (8.90% of sustained bandwidth). A summary is presented in Table 4.

Compared to STREAM (Figure 5) and GTC-P (Figure 6), HPCG (Figure 7 and Table 4) shows a slightly higher bandwidth fluctuation during execution. The write bandwidth is

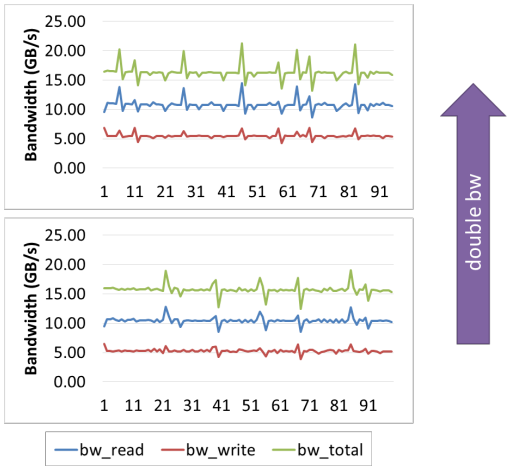

Fig. 6: Results for GTC-P using full and half risers.

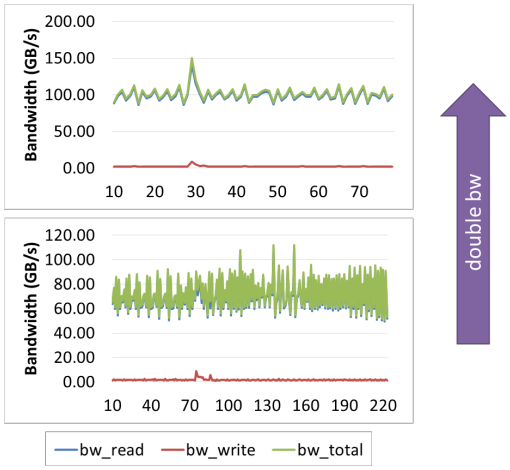

Fig. 7: Results for HPCG using full and half risers.

almost irrelevant compared to the read portion. Finally, MILC (Figure 8) clearly is the application with longest duration among the ones executed for these experiments. A summary of the calculations for MILC is presented in Table 4. Observe that MILC was executed with 16 cores, thus the sustained bandwidth was 70.06 GB/s and 142.34 GB/s for half risers and full risers, respectively.

VI. Analysis

A summary of all experimental results is observed in Table 4. This table shows the initial effective bandwidth B, the fraction F of sustained bandwidth, the new effective bandwidth B' (after increasing the available bandwidth by restoring the full risers configuration), the new fraction F',

Table 4: Summary of all experimental results.

	B (GB/s)	F	B' (GB/s)	F'	I_B	η
STREAM	89.77	100.0%	183.84	100.0%	104.79%	100.0%
MILC	56.25	80.29%	91.52	64.29%	62.68%	60.76%
HPCG	69.30	77.19%	101.75	55.35%	46.83%	44.69%
GTC-P	15.74	17.53%	16.36	8.90%	3.92%	3.74%

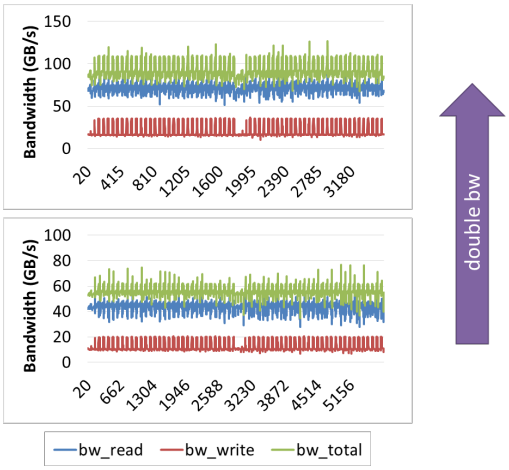

Fig. 8: Results for MILC using full and half risers.

the increase in bandwidth I_B, and the efficiency factor η for each application. Note that $I_S = I_{B_{\text{STREAM}}} = 104.79\%$.

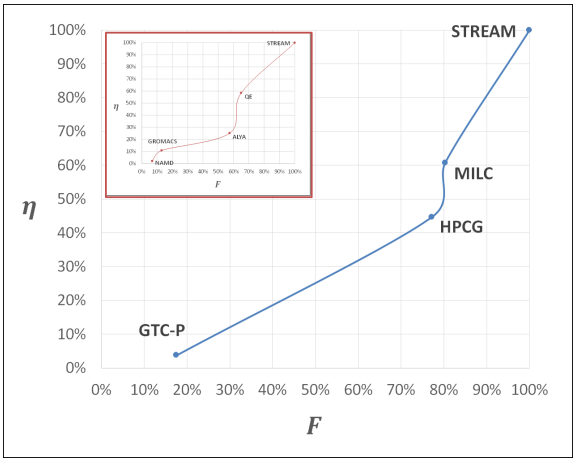

Fig. 9: Curve $\eta \times F$ for experimental results. A miniature of the $\eta \times F$ curve for the experimental results from Radulovic et al. was added too.

Plotting the $\eta \times F$ curve for these values leads to the graph presented in Figure 9. This figure also includes a miniature of the $\eta \times F$ curve for the experimental results from Radulovic et al. The results demonstrate a similar pattern of decreasing efficiency of transforming bandwidth increase into application performance depending on the amount of memory bandwidth the application originally uses. In these experiments the sustained bandwidth was doubled, but GTC-P effective bandwidth utilization did not double – it actually increased by only 4%. Even MILC, with an initial bandwidth utilization of 80%, observed an increase of 64% in effective bandwidth. Based on these numbers, increasing the available bandwidth even by a substantial amount is not enough by itself to improve application performance. As mentioned by Radulovic et al. [5], even when increasing bandwidth mitigates memory collisions other components can still limit performance.

Moreover, comparing to the curve plotted using the experimental results from Radulovic et al., a similar behavior

is observed, but not the exact same curve with the same $F \rightarrow \eta$ correlation. Both curves present two main linear regions separated by a steep slope – for Radulovic et al. around $F = 60\%$, and for the results of this paper around $F = 80\%$. This suggests that the curve parameters might depend on several factors, such as machine architecture, hardware, memory hierarchy, underlying software, etc. Consequently, to define an accurate and adequate methodology for performance prediction, especially for memory bound regions of applications, it is important to execute these applications and benchmarks in a modern machine with a configuration similar to the target system. Then, based on these experimental results, the $\eta \times F$ curve can be traced in order to provide a baseline to predict performance for the memory bound regions of applications.

VII. Conclusion and Future Work

This paper presented an investigation on the effects of memory bandwidth on application performance, more specifically on scientific benchmarks. At first the paper provided an analysis of background concepts and related work to formulate a performance model for memory bound regions of applications. This model is based on two main observations: *(i)* Memory bound applications benefit more from an improvement to total available bandwidth than applications that are not memory bound; and *(ii)* The performance gains of improving available memory bandwidth gradually diminish as this bandwidth increases, which is graphically represented by the dislocation to the left (lower fraction F and lower efficiency η) on the $\eta \times F$ curve.

The model culminated in the definition of a iterative method to calculate the performance gain (measured in the form of effective bandwidth used by the application) using variables such as the initial bandwidth, the sustained bandwidth, and the parameters of the $\eta \times F$ curve. The model and its foundations were then tested via experiments conducted on newer hardware and software. The results exhibited a similar behavior to the one observed in the curve plotted using experimental results from the related work from Radulovic et al. [5]. On the other hand, the disparities between the curves suggest that the behavior might depend on several factors, such as hardware and software properties, and even application configurations. Consequently, an accurate prediction model for memory bound regions requires extensive experimentation to determine the best $\eta \times F$ curve to be used. Moreover, improving application performance requires a balanced system able to explore potential enhancements to the memory subsystem. Future work includes execution of further experiments with different applications and different hardware setups. Experiments will cover different application configurations, problem sizes, and parallelism options (e.g., threading [22], [23]).

Acknowledgment: We thank David Montoya and Gary Grider for the feedback during the creation of the performance model and the execution of the experiments. We also thank Karen Bard and Dave Singer for the support during the configuration of the Firestone cluster.

References

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[2] Z. Guz *et al.*, "Threads vs. caches: Modeling the behavior of parallel workloads," in *2010 IEEE International Conference on Computer Design*, Oct 2010, pp. 274–281.

[3] B. L. Jacob, *The Memory System: You Can't Avoid It, You Can't Ignore It, You Can't Fake It*, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2009. [Online]. Available: http://dx.doi.org/10.2200/S00201ED1V01Y200907CAC007

[4] W. A. Wulf and S. A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, mar 1995. [Online]. Available: http://doi.acm.org/10.1145/216585.216588

[5] M. Radulovic *et al.*, "Another Trip to the Wall: How Much Will Stacked DRAM Benefit HPC?" in *Proceedings of the 2015 International Symposium on Memory Systems*, ser. MEMSYS '15. New York, NY, USA: ACM, 2015, pp. 31–36. [Online]. Available: http://doi.acm.org/10.1145/2818950.2818955

[6] J. Ousterhout, "Why aren't operating systems getting faster as fast as hardware," in *In Summer USENIX90*. Citeseer, 1990.

[7] R. Sites, "It's the memory, stupid!" *Microprocessor Report*, vol. 10, no. 10, pp. 2–3, 1996.

[8] B. Jacob *et al.*, *Memory Systems: Cache, DRAM, Disk*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[9] D. Patterson *et al.*, "A case for intelligent RAM: IRAM," *IEEE Micro, April*, 1997.

[10] S. Srinivasan, "Prefetching vs The Memory System: Optimizations for Multi-Core Server Platforms," Ph.D. dissertation, University of Maryland, 2007.

[11] A. Hutcheson and V. Natoli, "Memory Bound vs. Compute Bound: A Quantitative Study of Cache and Memory Bandwidth in High Performance Applications," Stone Ridge Technology, Tech. Rep., jan 2011. [Online]. Available: http://stoneridgetechnology.com/wp-content/uploads/2014/12/ComputevsMemory.pdf

[12] R. Crisp, "Direct rambus technology: The new main memory standard," *IEEE Micro*, vol. 17, no. 6, pp. 18–28, 1997.

[13] Hybrid Memory Cube Consortium, "Hybrid Memory Cube Specification 2.1," Oct. 2015. [Online]. Available: http://www.hybridmemorycube.org/specification-v2-download-form

[14] JEDEC, "High Bandwidth Memory (HBM) DRAM," Nov. 2015. [Online]. Available: https://www.jedec.org/standards-documents/results/jesd235

[15] D. A. Patterson, "Latency Lags Bandwith," *Commun. ACM*, vol. 47, no. 10, pp. 71–75, Oct. 2004. [Online]. Available: http://doi.acm.org/10.1145/1022594.1022596

[16] Y. Chou *et al.*, "Microarchitecture Optimizations for Exploiting Memory-Level Parallelism," *SIGARCH Comput. Archit. News*, vol. 32, no. 2, pp. 76–, Mar. 2004. [Online]. Available: http://doi.acm.org/10.1145/1028176.1006708

[17] A. Caldeira *et al.*, "IBM Power Systems S822LC – Technical Overview and Introduction," IBM, Tech. Rep., December 2015. [Online]. Available: http://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/redp5283.html

[18] J. D. McCalpin, "Sustainable Memory Bandwidth in Current High Performance Computers," Silicon Graphics, Inc., Tech. Rep., Oct. 1995. [Online]. Available: http://www.cs.virginia.edu/~mccalpin/papers/bandwidth/bandwidth.html

[19] NERSC, "GTC-P." [Online]. Available: http://www.nersc.gov/research-and-development/apex/apex-benchmarks/gtc-p/

[20] ——, "HPCG." [Online]. Available: http://www.nersc.gov/research-and-development/apex/apex-benchmarks/hpcg/

[21] ——, "MILC." [Online]. Available: http://www.nersc.gov/research-and-development/apex/apex-benchmarks/milc/

[22] W. Wang *et al.*, "Predicting the memory bandwidth and optimal core allocations for multi-threaded applications on large-scale numa machines," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 419–431.

[23] ——, "Dramon: Predicting memory bandwidth usage of multi-threaded programs with high accuracy and low overhead," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2014, pp. 380–391.