Adaptive Memory Power Management Techniques for HPC Workloads

By

Karthik Elangovan

A thesis submitted to the

Graduate School – New Brunswick

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree

Master of Science

Graduate program in Electrical and Computer Engineering

Written under the direction of

Manish Parashar

And approved by

_____

_____

_____

_____

New Brunswick, New Jersey

JANUARY 2012

ABSTRACT OF THE THESIS

Adaptive Memory Power Management Techniques for HPC Workloads

By Karthik Elangovan

Thesis director: Manish Parashar

The memory subsystem is responsible for a large fraction of the energy consumed by compute nodes in High Performance Computing (HPC) systems. The rapid increase in the number of cores has been accompanied by a proportional increase in the DRAM capacity and bandwidth. Thus, the memory system consumes a significant amount of the power budget available to a compute node. There is a broad research effort focused on power management techniques using DRAM low-power modes. However, memory power management still presents many challenges towards Exascale. In this thesis, the potential of Dynamic Voltage and Frequency memory Scaling (DVFS) is studied considering the ability to select different frequencies for different memory channels. The approach adopted is based on tuning voltage and frequency dynamically to maximize the energy savings while maintaining performance degradation within tolerable limits. It was observed that HPC workloads rarely require maximum bandwidth, and the bandwidth demand placed by applications is spread over different channels. Also, HPC applications do not use all the bandwidth in a sustained manner, and they have phases where this bandwidth demand is not at its peak**.** Hence applications can tolerate reduction in bandwidth to save energy. Channel access patterns of applications are studied to determine the potential additional energy savings by controlling channels independently. Evaluation of proposed DVFS algorithm is conducted through a novel hybrid evaluation methodology that includes simulation and executions on real hardware. Results show the large potential of adaptive memory power management techniques based on DVFS for HPC workloads.

ACKNOWLEDGEMENTS

I would like to express my gratitude to all the people who have made this work possible. Especially, I would like to take this opportunity to extend my deepest gratitude to my advisor, Dr. Manish Parashar, for his orientation, advice and support.

I am especially grateful to Dr. Ivan Rodero, for giving me the chance to do this master thesis by introducing me to work related to DRAM Power Management, and providing a great environment to make it possible. I am also grateful to Dr. Yanyong Zhang for taking time off her busy schedule to review my thesis

Many thanks to Dr. Francesc Guim, who has helped me in the development of this project. I really want to thank him for all his help and feedback. I would also like to thank all the members of TASSL research group for their invaluable discussion and help whenever I needed it. My deepest gratitude goes to my Mom and Dad for their unflagging love and support right from the beginning of my education to this day; I could never have come so far without their encouragement. I would also like to thank my uncle and his family, especially my favorite nephews for their love and support over the last 2 years. I would like to thank my roommates and friends in and around cedar lane for keeping my spirits lifted during the period of my study here. I thank God for enabling me achieve what I have achieved and also for keeping an eye on me these last few years.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1. MOTIVATION

High Performance Computing (HPC) evolved over the past decades into increasingly complex and powerful systems. After attaining Gigaflops and Teraflops performance in the mid 80's and the last 90's, respectively, the Petaflops barrier was crossed in 2008 with the IBM Roadrunner system. Highend HPC systems power demand is increasing eight-fold every year [1], and today consume several MWs of power, enough to power small towns, and are in fact, soon approaching the limits of the power available to them. The cost of power for this and similar HPC systems runs into millions per year. Therefore, more energy efficient systems can reduce the data centers total cost of ownership (TCO) since the reduction in energy consumption translates to lower energy costs and also reduced costs for cooling and other packaging infrastructure. Furthermore, electricity savings could lower the nation-wide carbon dioxide emissions. Reaching Exaflops performance by the end of the decade, and enabling the sustained performance scaling of smaller systems require significant research along different dimensions including power efficiency. Building an Exascale machine on a power budget of 20MWrequires a 200x improvement in energy per instruction. However, only 4x is expected from improved technology, while the remaining 50x must come from architecture and circuits improvements.

The memory subsystem is responsible for a large fraction of the energy consumed by compute nodes in HPC systems. Barrosso et al. [2] indicate that over the years CPU has evolved to an extent where it accounts for only 50% of energy consumption. DRAM manufacturers have optimized their designs for increased bandwidth to address memory wall issues. But little has been done to optimize design to tackle the Power Wall problem. Hence main memory has been a significant contributor towards total energy consumption. Therefore, reducing energy consumption through intelligent control of all subsystems [3] is critical. The rapid increase in the

number of cores has been accompanied by a proportional increase in the DRAM capacity and bandwidth. For these new architectures the amount of power required by the memory subsystem is an important percentage of the power budget that computer nodes have. Therefore, the design of novel memory power management techniques has become crucial since efficient power management techniques can allow to save power or transfer it to other components of the node.

1.2. PROBLEM DESCRIPTION

The brief survey of prior work reveals that:

1. DRAMs provide multiple operating modes. Each mode has different power consumption. A very coarse level of power control is exposed to memory controller through these modes – picking an operating state depending upon activity of a particular device. For instance unused modules can be sent to idle modes reducing power consumption considerably. But the granularity of power control is not very fine. Besides DRAM cannot perform all the functions in idle mode that it can perform when it's active. So memory controller has to cleverly decide the state in which devices should operate.
2. As pointed out earlier there is a very coarse level of power control. Even though there are multiple modes (with varying degree of power consumptions) each of these modes have only a limited degree of operational capabilities compared to the active state. Hence the memory controller is able to send the devices only to shallow low-power states. The reason for this inability to exploit Nap or lowest power modes are due to penalty associated with switching states. Memory controller has to carefully select operating state to minimize frequent unwanted switches.

3. Inability to exploit these low power states is primarily because of the latency to enter and exit these modes. The memory controller is usually aware of these latencies. Hence most of the policies for choosing occupational states are based on monitoring current activity and selecting states for the next phase. Power is added as a constraint while choosing states.
4. The application should exhibit enough idle periods for effective policies based out of these low-power modes. DRAM devices can transition to low power modes during these periods effectively reducing power consumed. But finding enough idle limits the possibilities of conserving power with low-power modes.
5. A radical approach that opens up even more opportunities to reduce power is throttling DRAM commands in memory controller. Throttling can help reduce energy. Also as suggested by [23] Power Shifting can be used to change power gears of other parts of the computing system. But throttling can have negative effects on performance.
6. Research efforts on DRAM architectural changes can have negative effects on yield [27]. Architectural changes can take time to seep into the market provided there is no negative impact on yield.

Due to absence of opportunities to lower down power with current power control methods it is necessary to shift from the conventional methods of power control. An approach that can be applied across current class of DRAMs and architectural changes that are yet to come is what is currently needed. Energy proportional computing (DVFS) has provided considerable power savings to CPU hence extending the same principle to main memory should provide considerable savings.

Most scientific applications are iterative which means that they apply the same algorithm several times to the same data set. In particular, the data is processed repeatedly until number

of iterations reaches a specific value or until the value of some parameters converges (for instance, when the error converges to certain value). Hence HPC applications exhibit good locality which means that the last level cache captures most of the accesses. Thus the bandwidth demand placed on main memory is not at its peak during the entire period of execution. This behavior clearly shows that main memory need not be operated at maximum voltage and frequency levels all the time. DVFS can be extended to main memory to reduce energy consumption.

1.3. OBJECTIVES & APPROACH

This study is based on three observations:

1. Reducing frequency and voltage reduces power considerably.
2. The time it takes to perform read or write operations does not vary too much when we change the frequency.
3. Iterative behavior exhibited by HPC application further aids tuning voltage and frequency dynamically. Applications do not demand peak bandwidth all the time. Hence reduced stress on main memory provides opportunities to operate the channel at a lower voltage and frequency level.

Dynamically scaling Voltage and Frequency of main memory to contain power can be done at the cost of losing a preset amount in performance. Applications can trade bandwidth and sustain reduction in performance for saving energy. Recent design of processors incorporates multiple on-chip memory controllers and multiple channels for more parallelism. Each channel can be observed and depending upon the activity on that channel memory devices connected can be operated at appropriate voltage and frequency levels. To keep the degradation within

limits, while evaluating potential of doing DVFS, a performance guarantee algorithm has been built on top so that the frequency selection process can prevent too much loss in performance.

A novel hybrid evaluation methodology has been adopted that includes simulation as well as executions in real hardware. This approach is used to study the application channel access patterns, and to evaluate three possible frequency selection strategies and two different algorithms for mapping physical memory addresses to channels. The proposed control techniques provide around 55% energy savings on average with only around 5% degradation of execution time on average, using the NAS Parallel Benchmarks. It is also observed that controlling the channels independently provides considerable savings in comparison to controlling the frequency of all the channels together when applications exhibit uneven load.

1.4. CONTRIBUTIONS

The main contributions of this thesis are summarized as follows:

1. Study of memory access patterns, bandwidth and channels usage for HPC workloads.
2. Design of a control algorithm for adaptive memory power management.
3. A novel hybrid evaluation methodology that includes simulation and executions on real hardware.
4. Evaluation of the proposed strategy on DDR3 SDRAM with number of memory channels, frequency selection strategies, and physical address to channels mapping.

2. BACKGROUND

2.1 DRAM ORGANIZATION

Data in a DRAM device has an array structure with multiple rows and columns. The intersection of a row and column is a storage cell, which is periodically refreshed because of gradual data discharge. A row and column address can fetch only a single bit from a single device in the array. Usually a number of arrays are operated in parallel on each access. DRAM arrays that act in unison on a request belong to the same bank, for instance in a x4 DRAM four arrays act in parallel on each access. The notation also indicates the column width (e.g., x8, x16, x32 and x64 devices are commercially available). Fig. 1 shows DRAM arrays grouped together.

Commercially available DRAMs are generally distributed as memory modules (DIMM) with DRAM devices and all the other circuitry built-in. Our discussion pertains to class of JEDEC DDR SDRAMs. Fig. 1 shows DRAM and all the other components.

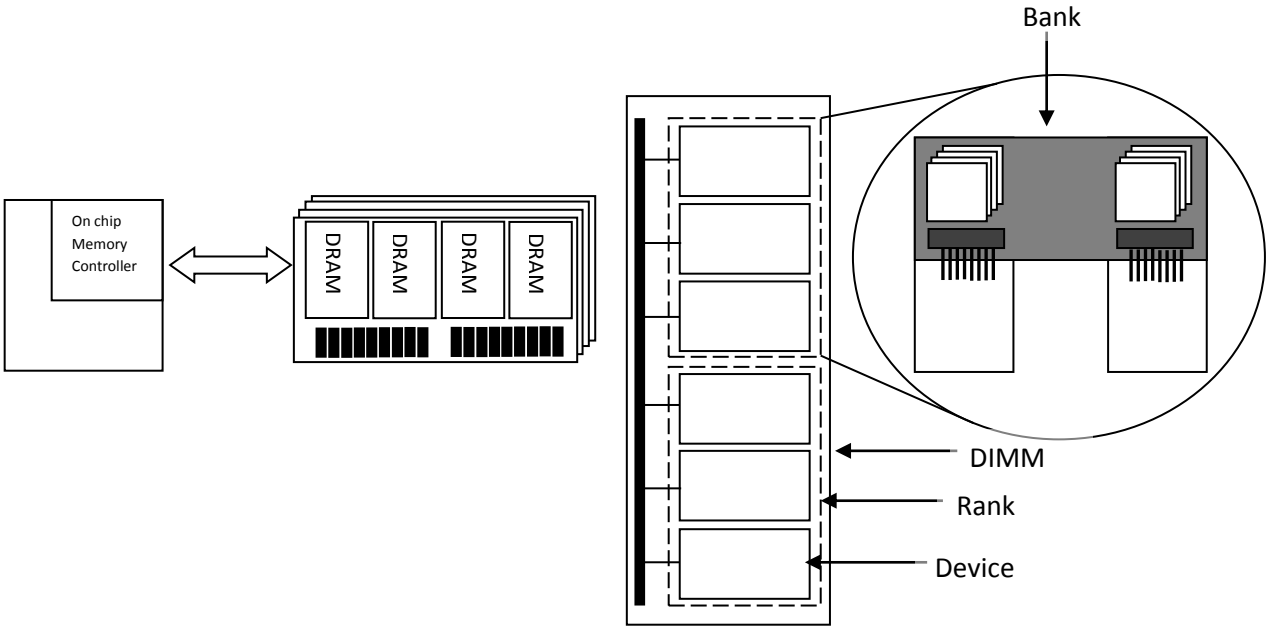

Figure 1: (Left) DRAM subsystem with DRAM modules. (Right) DIMM organization with individual rank, bank and arrays inside each bank.

Due to the DRAM organization the physical addresses generated by applications cannot be used to directly access DRAM, instead processors interface with DRAM through memory controller which relieves the socket modules from directly interfacing the DRAM. The memory controller has to convert the physical addresses to a set of commands that DRAM device understands. Physical addresses are split by the memory controller to identify a bank, then a row and column within a bank.

Every DRAM operation (read/write) begins with an activate command, which requires selecting a row with an appropriate row address. The content of the entire row is transferred to the sense amplifiers. Every bank has separate sense amplifiers and each bank can be activated independently. Once the entire row has been read it is said to be open and the bank is said to be active. The next step is to select a column and then the content of the column is placed on the data bus. DDRx SDRAMs transfer data on both edges of the clock cycle. The final step is to restore the data back to the cells with a precharge operation that closes a row.

The set of commands described previously can be pipelined when accessing different banks subsequently. But banks contend for access to the data bus. This is usually exploited by memory controllers to increase the bandwidth.

The memory controller can close the row after a read/write access with a precharge command. In closed page DRAMs, rows are immediately closed after a row access command, while in open page DRAMs, rows remain opened expecting subsequent hits to the same row. In this study we evaluate only closed page RAMs.

Fig. 2 shows DDR3 device with eight separate banks. Each bank is capable of outputting 64 bits. The figure also shows separate registers to hold row and column address. On an activate command the row address is used to activate a row and read the contents to the sense

amplifiers. On receiving the column address the appropriate data is then read out to the data bus through I/O gating.

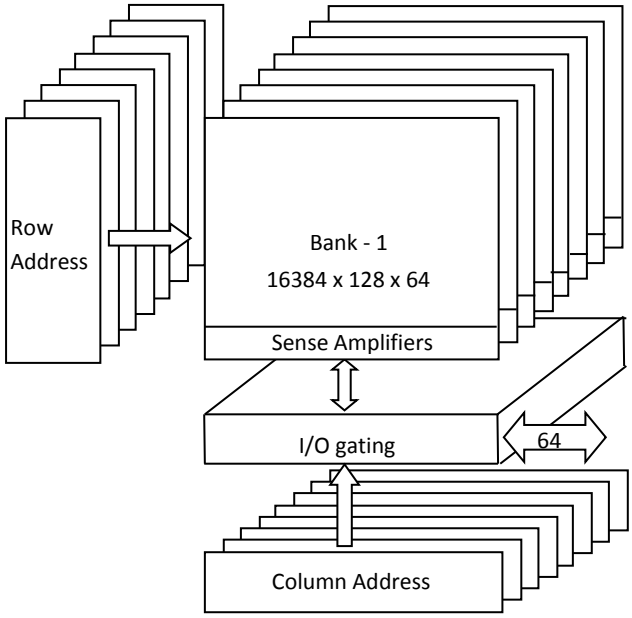

Figure 2: DDR3 SDRAM with 64 bit data out. 16384 rows and 128 columns and 8 banks

The data out capacity of this DRAM is 8 bytes. But cache lines requests are for 64 bytes. DRAMs provide ability to read in burst on single command. The device shown in Fig. 2 can provide 8 bytes at a time. Hence data (an entire cache line) can be sent back in 4 cycles (because DDRx can transfer over both the edges of the clock).

2.2 DRAM TIMING

The basic timing parameters for generic DRAM commands that will be used throughout this thesis are the following:

- t_{RCD} : Time taken to move an entire row of data to sense amplifiers.
- t_{CAS} : Time interval between column read command and availability of data in data bus.
- t_{CWL} : Time interval between column write command and availability of data in data bus.
- t_{RAS} : Time interval between row access and data restoration back to the cells

- t_{RP} : Time taken to precharge a row
- t_{RC} : Minimum time interval that has to elapse between different accesses to same row in a bank
- t_{WR} : Time it takes to restore the write data from sense amplifiers back to the cells.

Every read operation takes row activation (t_{RCD}), column read (t_{CAS}), data out (t_{BURST}) and precharge (t_{RP}). Total time is, $t_{RCD} + t_{CAS} + t_{BURST} + t_{RP.}$ DDR3 devices can perform each of these operations in about 10ns. Timing parameters for write operation remains the same except column access takes t_{cWD} and there is a t_{WR} additional waiting period after end of last data into sense amplifiers.

2.3 VOLTAGE AND FREQUENCY SCALING

The JEDEC memory standards [4] are the specifications for semiconductor memory circuits and similar storage devices adopted by the JEDEC Solid State Technology Association. They allow changing the input memory frequency, but the transition should respect the timing parameters specified by the standard; however, commercially available DRAM devices do not support multiple voltage domains. In this thesis, we assume the DRAM to operate properly within the voltage levels configurable using the BIOS settings. Table 1 lists the voltage and frequency combinations considered in this study for DDR3 SDRAM along with the power breakup of a 2GB DIMM (256x64)(according to [33]. Note that the voltage range [1.575V–1.425V] is the allowable voltage range of the power supply.

Frequency **(MHz)**	Voltage **(V)**	P_{ds}(PRE_STBY) **(mW)**	P_{ds}(ACT_STBY) **mW**	P_{ds}(ACT) **mW**	P_{ds}(WR) **mW**	P_{ds}(RD) **mW**	P_{dq}(RD) **mW**	P_{dq}(WR) **mW**
933	1.575	252	466	226	919	793	96	620
800	1.5375	240	420	170	660	720	91	591
667	1.5	229	400	104	457	514	87	562
533	1.4625	184	326	91	380	435	83	535
400	1.4250	131	232	64	271	310	79	508

Table 1: Voltage and frequency levels

3. RELATED WORK

Over the last years, processors have evolved to become energy efficient supporting multiple operating modes. At a very coarse level, power management at the system level consisted on monitoring the system load and shutting down unused clusters or transitioning unused nodes to low power modes [5]. Such algorithms are based on the availability of sufficient idle periods and propose to batch work to increase the probability of such idle periods. Computing nodes incur latency only when they exit from these low-power modes. Other studies [6], [7], [8], [9] have proven that dynamically varying the voltage and frequency proportional to system load is effective in reducing energy consumption. DVFS provides power savings at the cost of a small increase in execution time. Other approaches conducted on power management techniques are focused at the processor level, for example, Cai et al. [10] propose a DVFS techniques based on the hardware thread runtime characterization.

The previous approaches tackled the energy consumption optimizations focused in the computing elements. Memory devices, which were once considered to be undesirable candidates for power consumption, began to significantly contribute to overall system energy consumption. Just like processors, DRAM devices have several low power modes. Delaluz et al. [11] present software and hardware assisted methods for memory power management. They studied compiler-directed techniques [12], [11], as well as OS-based approaches [13], [14] to determine idle periods for transitioning devices to low power modes. Even though this approach is very conservative, switching to/from low-power modes during unwanted periods can be prevented; however it is not very effective on multi-core systems. Cho et al. [15] studied assigning CPU frequencies for DVFS that is memory-aware, because focus of all prior work was on optimal assignment of frequencies to CPU, thus ignoring memory. Huang et al. [16] proposed

a power-aware virtual memory implementation in the OS to reduce memory energy consumption. Fradj et al. [17], [18] propose multi-banking techniques that consist of setting individually banks in lower power modes when they are not accessed.

Self-Monitoring [11] approach can be effective with a control algorithm to choose between power modes for memory devices. The key is to determine idle periods to transition devices to low-power modes; however, self-monitoring control algorithms are threshold-based, therefore, an algorithm that transitions into low-power modes too frequently can increase latency, while a very sluggish algorithm can miss out on opportunities to save power. Li et al. [19], [20], [21] proposed a self-tuning energy management algorithm to provide performance guarantees. The algorithm tunes threshold parameters at different points of execution. Diniz et al. [22] studied dynamic approaches for limiting the power consumption of main memory by limiting consumption and adjusting the power states of the memory devices, as a function of the memory load. Hur et al. [23] took an entire different approach towards DRAM power management. DRAM commands are delayed in memory controller to increase idle periods, and to exploit low-power modes of DRAMs. Commands are delayed in memory controller for a certain number of cycles which is determined by a delay estimator.

Several architectural changes have also been proposed. Rank subsetting [24] and Mini-Rank [25] tackle energy consumption constraint by dividing a rank into subsets. This approach reduces the number of chips which are put to work at each memory access. Udipi et al. [26] suggested changes to internal organization of DRAM devices. The authors argue that an open-paged policy does not present any improvements for a multi-core architecture because an opened row has one or two hits on average. Building on that argument it is unnecessary to read all the bits to the

sense amplifiers, thus reducing the number of bits that are touched can decrease the energy consumption.

Deng et al. [27] proposed dynamic frequency scaling of the main memory. The frequency of all memory channels is changed to provide energy savings with guaranteed performance degradation. Even though the work is similar, the focus of this study has been clearly on evaluating the potential of controlling the voltage and frequency of all channels independently.

4. ADAPTIVE MEMORY POWER MANAGEMENT

This work has been primarily motivated by two behaviors of applications: (1) bandwidth demand, and (2) the memory access patterns. As mentioned previously, applications rarely demand peak bandwidth from main memory since on chip caches work efficiently for capturing accesses to main memory. LLC misses of an application can be used to derive bandwidth demand since it has positive correlation with bandwidth. Fig. 3 shows misses access, expressed as Misses per Kilo Instructions (MPKI) of two NAS benchmarks that were collected using CMPSim as described in Chapter 5, and demonstrate the time varying behavior of both applications. There are periods of high and low bandwidth demand, applications will not incur significant penalty by switching to a lower voltage and frequency during these low bandwidth demand periods.

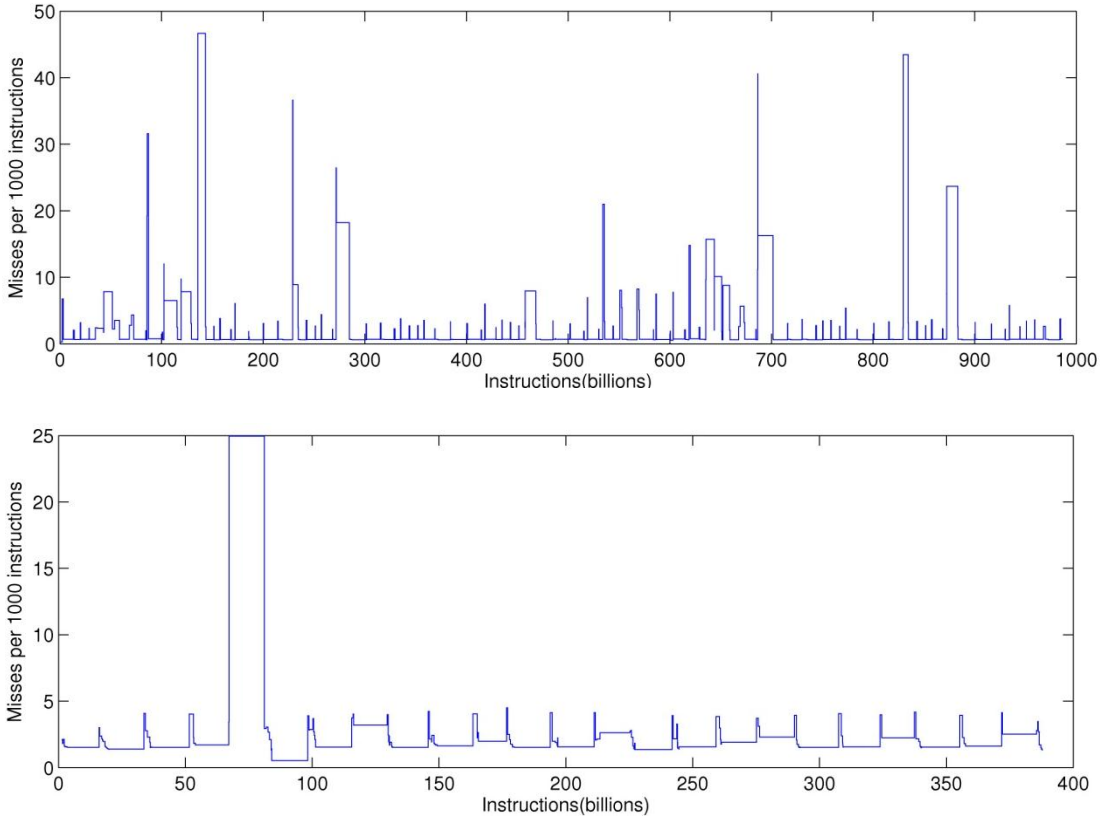

Figure 3: MPKI of BT (top) and FT (Bottom)

4.1. CHANNEL MAPPING FROM PHYSICALL ADDRESS

As described previously the processor cannot directly access DRAM devices. The physical address is sent to the memory controller which splits the address to first find the physical channel, then the rank, and then the bank within rank. The process of identifying the channel number is proprietary to each memory controller design, for instance, considering the example shown in Figs. 4 and 5 the physical address in divided to address different parts of the device, and the Row ID is mapped to the most significant bits so that consequent addresses go to the same row.

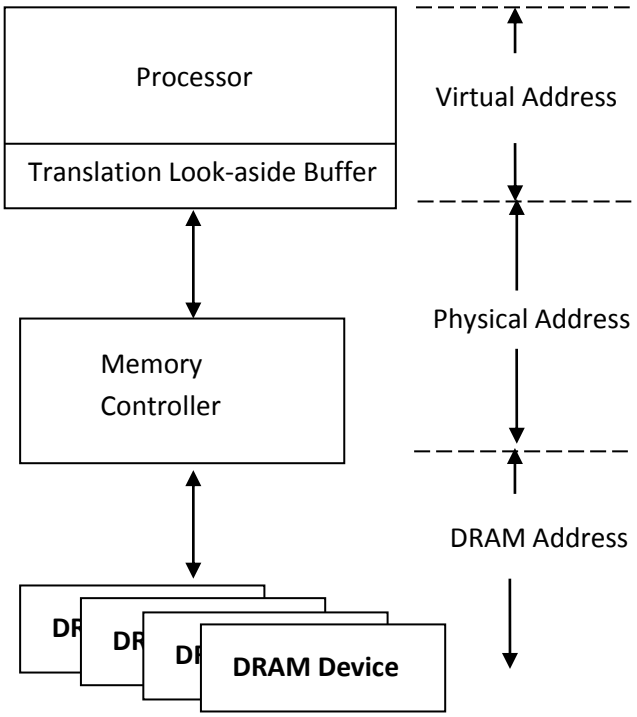

Figure 4: Addresses seen by different layers of hardware

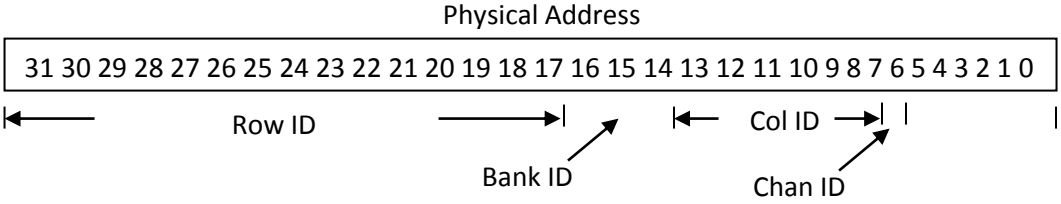

Figure 5: Channel Mapping

Two different algorithms for mapping physical addresses to channels have been considered:

- **Default**: Accesses are clustered to certain channels thus reducing activity on other channels. Allowing one to tune down the voltage and frequency of channels which are relatively lightly loaded. For instance, with a 4GB physical memory and 8 channels the first 512MB can be mapped to first channel, the next 512MB to second channel, etc.
- **Interleaving**: Accesses are distributed across different channels. This helps distribute load equally across different channels by mapping consequent cache lines to different channels. Figs. 6–14 illustrate how the algorithm for mapping physical memory addressed to channels can significantly affect the memory access pattern, and presumably the application behavior.

4.2. MEMORY ACCESS PATTERNS

Memory access patterns of three NAS Benchmarks were collected using mptrace and CMPSim (see Chapter 5). CMPSim dumps processor specific information whenever a hardware thread hits a billion instructions. Mptrace is used to obtain the physical addresses accessed. The data is then processed to obtain the channels access patterns using different channel mapping algorithms. Figs. 6, 7 and 8 show the memory access patterns for four, eight and sixteen channels. Access patterns were collected for different regions (i.e., a block of instructions) of the application. The exact number of application instructions for a region is not fixed, and varies with the CMPSim output dump frequency (i.e., when a thread hits 109 instructions). The memory access patterns exhibited by applications motivate the use of the adaptive techniques proposed in this work. Peak bandwidth is not always demanded by applications and there is unequal distribution of accesses across channels. This asymmetry presents opportunities to control the channels independently.

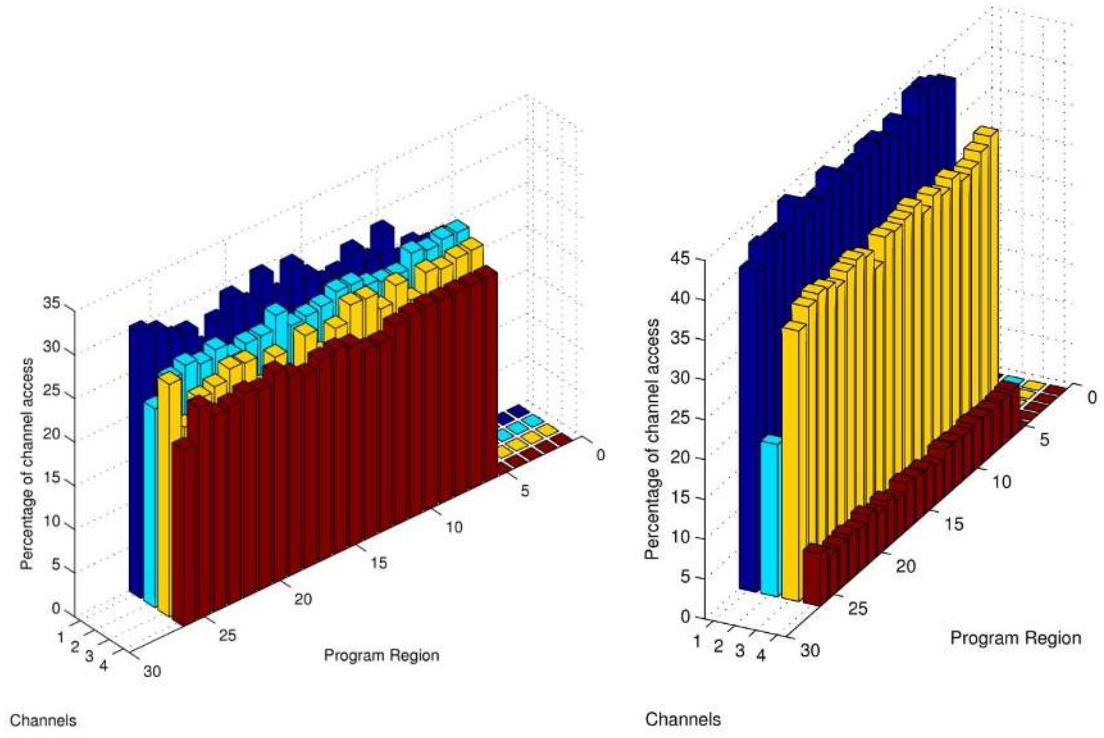

Figure 6: Access Pattern of BT benchmark with 4 channels. Interleaved Mapping (left) and Default mapping(right)

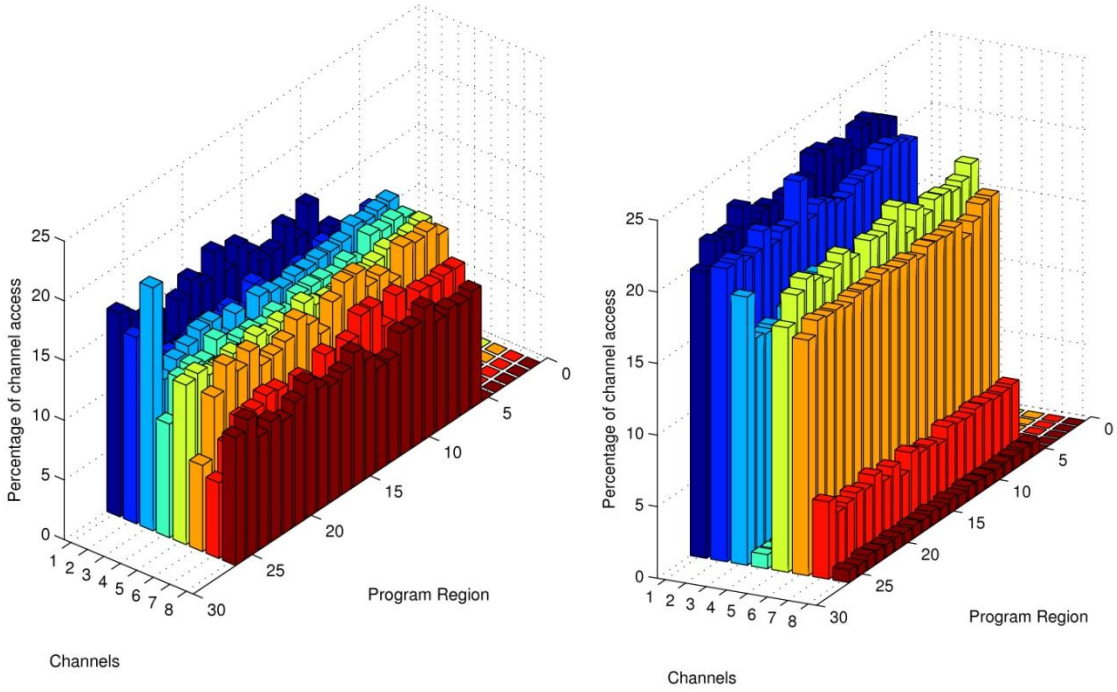

Figure 7: Access Pattern of BT benchmark with 8 channels. Interleaved Mapping (left) and Default mapping(right)

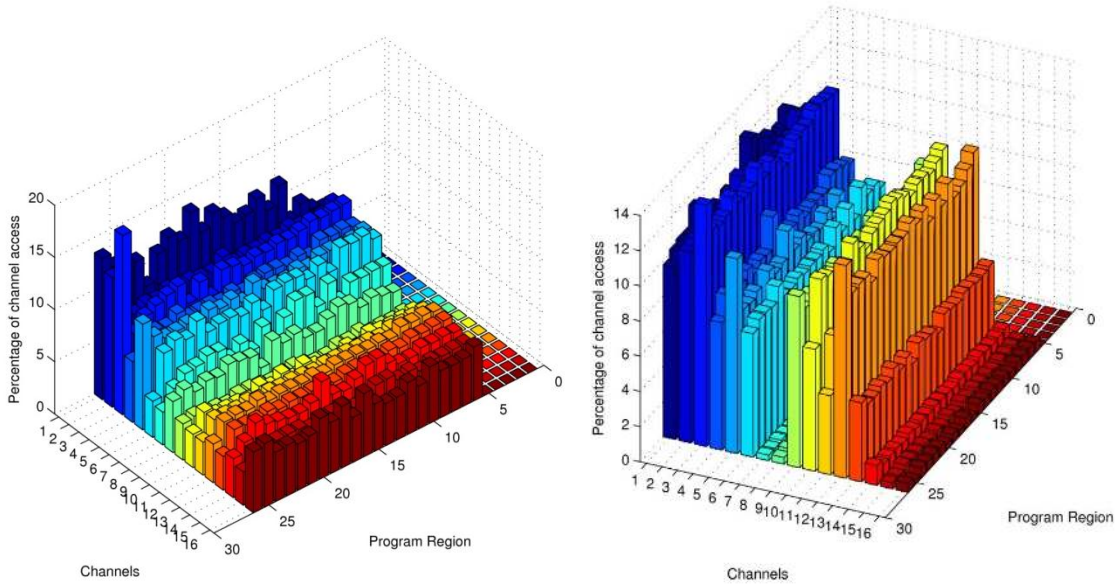

Figure 8: Access Pattern of BT benchmark with 16 channels. Interleaved Mapping (left) and Default mapping(right)

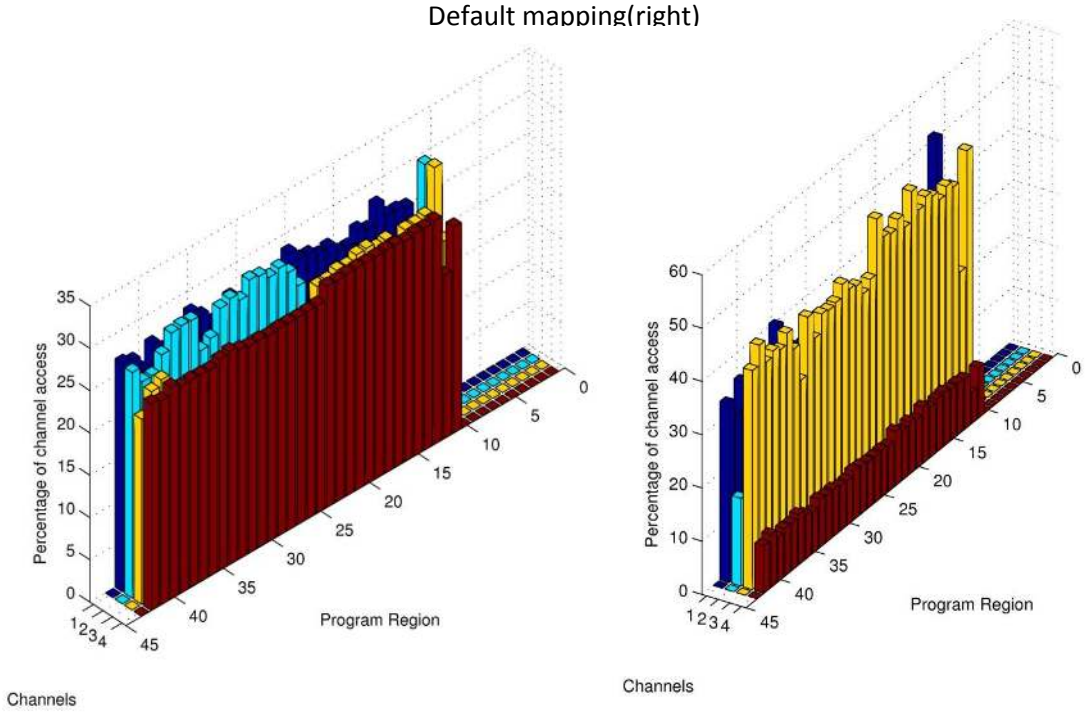

Figure 9: Access Pattern of FT benchmark with 4 channels. Interleaved Mapping (left) and Default mapping(right)

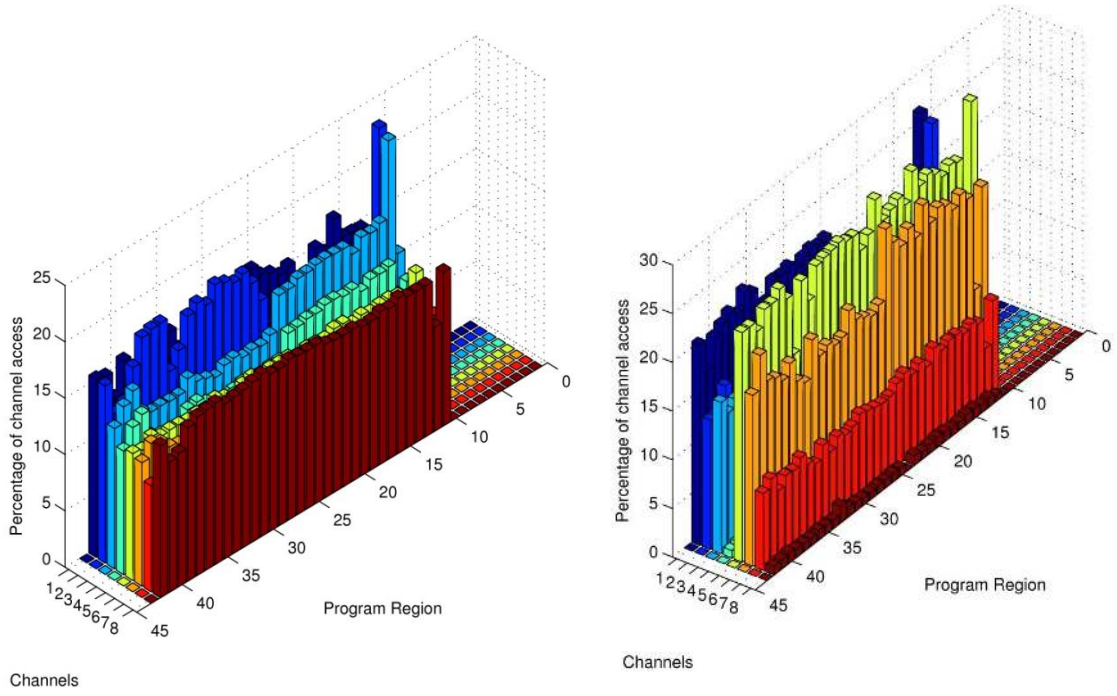

Figure 10: Access Pattern of FT benchmark with 8 channels. Interleaved Mapping (left) and Default mapping(right)

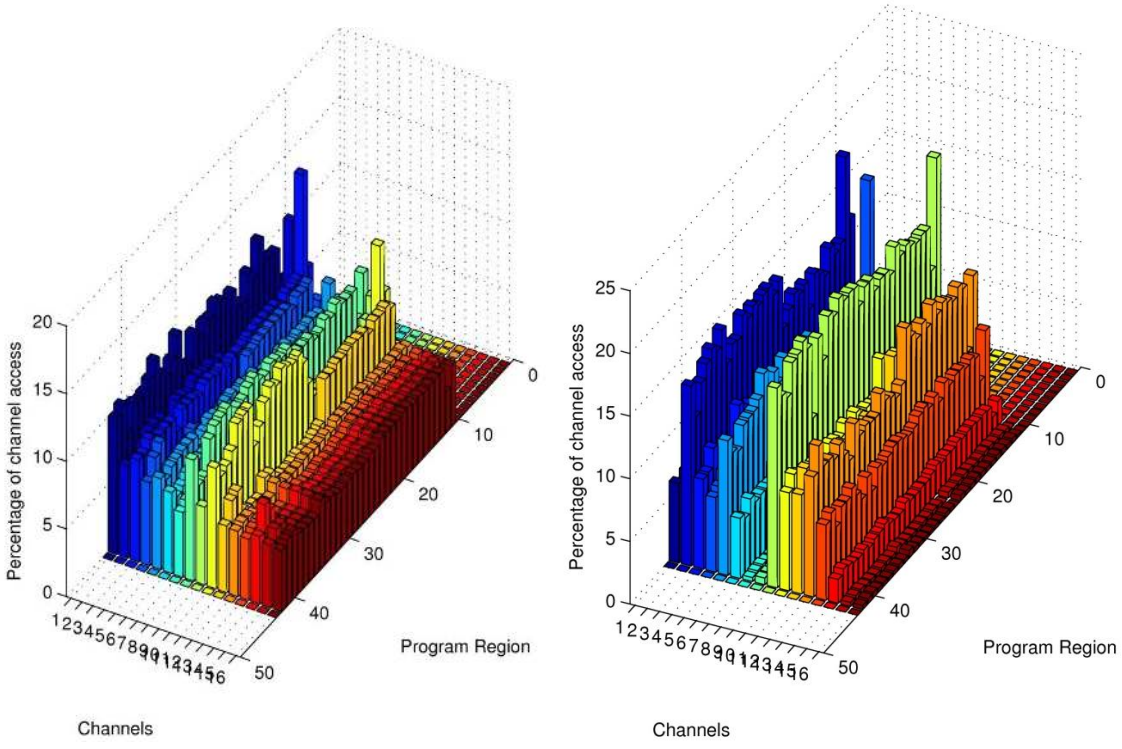

Figure 11: Access Pattern of FT benchmark with 16 channels. Interleaved Mapping (left) and Default mapping(right)

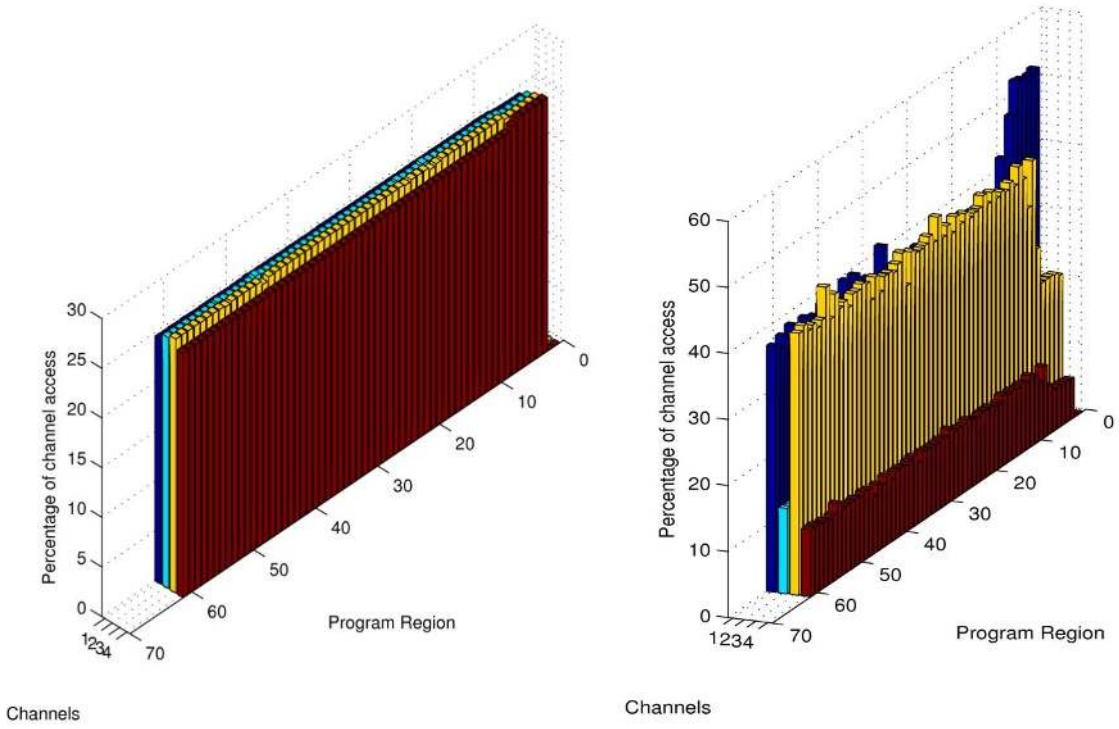

Figure 12: Access Pattern of CG benchmark with 4 channels. Interleaved Mapping (left) and Default mapping(right)

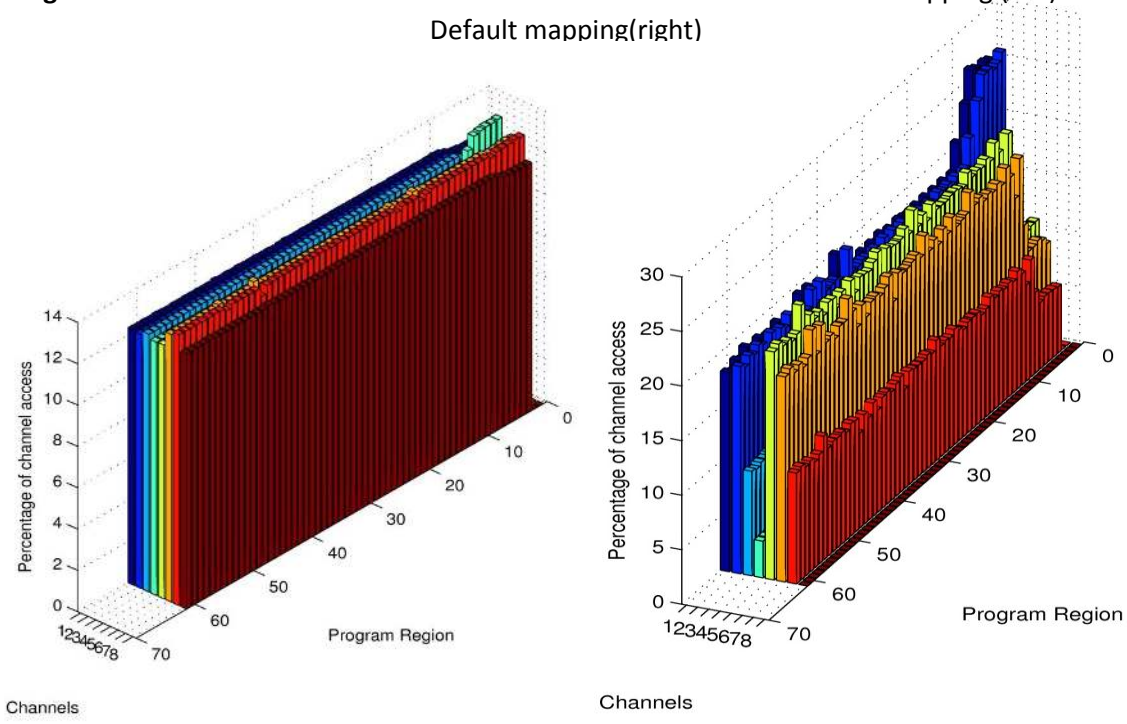

Figure 13: Access Pattern of CG benchmark with 8 channels. Interleaved Mapping (left) and Default mapping(right)

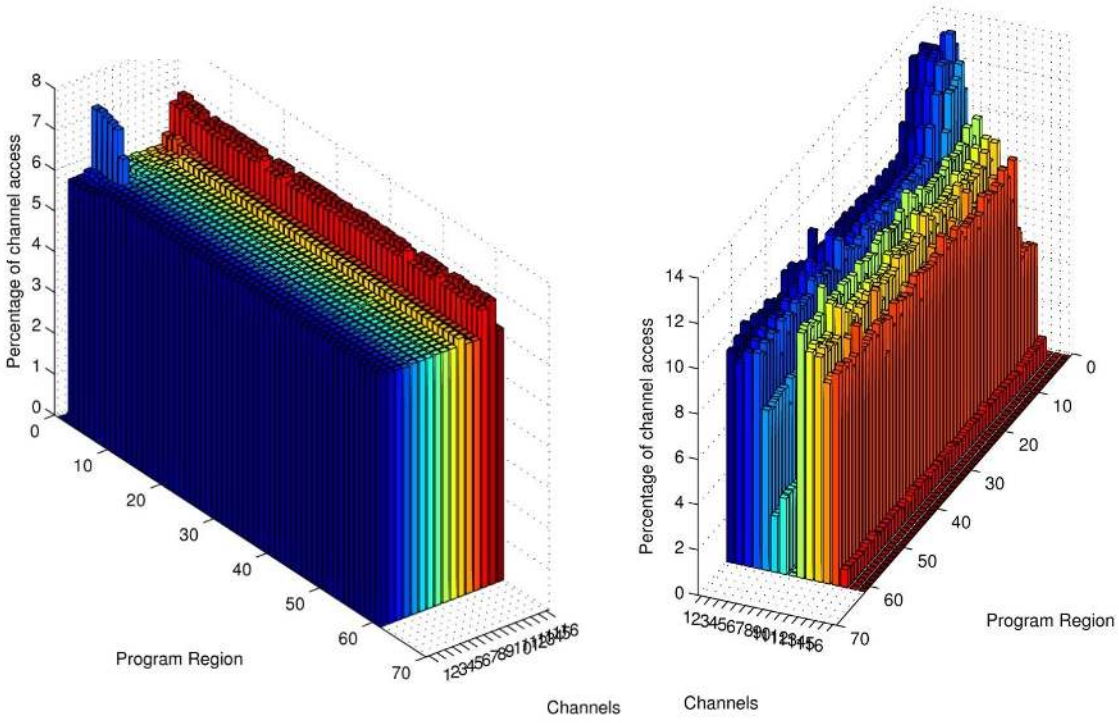

Figure 14: Access Pattern of CG benchmark with 16 channels. Interleaved Mapping (left) and Default mapping(right)

The figures show the applications exhibiting different characteristics with the two mapping algorithms. All the three applications streamline equal traffic into all channels with 4 and 8 channels when interleaved mapping is employed. But when the number of channels is increased to 16, even with interleaved mapping the applications show affinity to access certain group of channels. Dynamically adjusting the frequency of memory channels will not affect the performance of the applications significantly since there are periods of high and low bandwidth demands. Additional observation of access patterns shows that channels can be further tuned independently because of unequal distribution of traffic.

4.3. CONTROL ALGORITHM

The control algorithm selects the operating voltage and frequency of main memory. It is invoked on certain points along the execution of the program, and uses observed system parameters

during current phase of execution to obtain possible energy savings attainable at different operating states. The operating state here refers to voltage and frequency levels of channels. The control algorithm uses the power model described in Chapter 5.3. It is possible to compute the input parameters of the power model from the execution time. However, to prevent the power control algorithm from calculating this parameter, it is advisable to use a counter. In addition, the hardware should also have counters for LLC misses and total instruction executed. The counter for LLC misses should be maintained on a per-channel basis. To calculate the power dissipation at other frequency settings the execution time at all the other frequencies should be found. Execution time in total number of cycles (TNC) is given by Eq. 1 where I_{CPU} is the total non-memory instructions, I_{mem} is the total memory instructions, CPI_{mem} is cycles per memory instructions that can be found using the performance model (described in Section V-C. If the control algorithm is invoked at fixed intervals TNC will be known for the current state. CPI*CPU* can be computed using Eq. 1. CPI_{CPU} is constant at all the other states since changing frequency will not affect non-memory instructions.

$$Total\ Number\ of\ Cycles = \left\{\left[\begin{matrix}Total\ Non-memory\\ Instructions\end{matrix}\right] \times \left[\begin{matrix}Cycles-per\ Non-memory\\ Instruction\end{matrix}\right]\right\} + \left\{\left[\begin{matrix}Total\ memory\\ Instructions\end{matrix}\right] \times \left[\begin{matrix}Cycles-per\ memory\\ Instruction\end{matrix}\right]\right\}$$

$$Total\ Number\ of\ Cycles = (I_{CPU} \times CPI_{CPU}) + (I_{Mem} \times CPI_{Mem})$$

Equation 1: Total Number of Cycles

The control algorithm performs the following sequence of steps when it is invoked:

Step i: Read all the counters.

Step ii: Use the performance model to calculate the execution time of the current phase at all the possible voltage and frequency levels.

Step iii: Calculate the power consumed by the other operating states.

Step iv: Calculate the energy savings for all the states, i.e., savings compared to operating all the channels at maximum frequency.

Step v: Choose a state for the next phase that maximizes energy savings while keeping CPI degradation within a specified limit.

With M possible operating frequencies and n channels we can have M^n possible states. The control algorithm should calculate energy savings and performance degradation for all possible states before it can select an operating state for the next phase. Execution time of the control algorithm primarily depends on the number of possible states for the next phase. Since this increases the execution time of the control algorithm exponentially with increase in number of channels, three possible frequency selection strategies have been considered:

- **Exhaustive Search**: All the frequencies are considered in this method (M^n possible states). The cost of executing the control algorithm with this scheme becomes very high with 8 and 16 channels.
- **3 Levels**: Three frequency levels can be considered at a time, for instance, if we start with 933MHz then 933MHz, 800MHz and 667MHz are considered for the next phase. If 667MHz is selected for the next phase then 800MHz, 667MHz and 533MHz are considered for the next phase. With such a scheme the time complexity of the algorithm is reduced considerably. Number of operating states is reduced to 3^M.

- **Ganged**: The number of possible operating states can be greatly reduced by slaving all the channels together, even when all the five frequencies are considered, there can be only 5 possible states. A slight variant of ganged in which instead of slaving all the channels together only certain subset of channels are tied together. Two variables, M and n, decide the number of possible states. Three level search reduces M to bring down the number of possible states. But if we group channels we can also reduce n. For instance with 16 Channels, we can have eight channels in a group effectively creating two virtual channels out of sixteen channels or four channels can be grouped effectively creating 4 virtual channels. Channels are sorted based on their respective channel access ratios and then grouped.

5. EVALUATION METHODOLOGY

Since the actual implementation of DVFS is not available for currently available hardware, the evaluation of the control algorithm is performed using simulation. However, part of input data of the control algorithm is obtained using tools that run on the actual execution platform for which the simulator is configured (see Fig. 15). The CMPSim simulator is used to capture the time varying behavior of the applications. The simulator was configured to produce statistics when any hardware thread reaches 10^9 instructions.

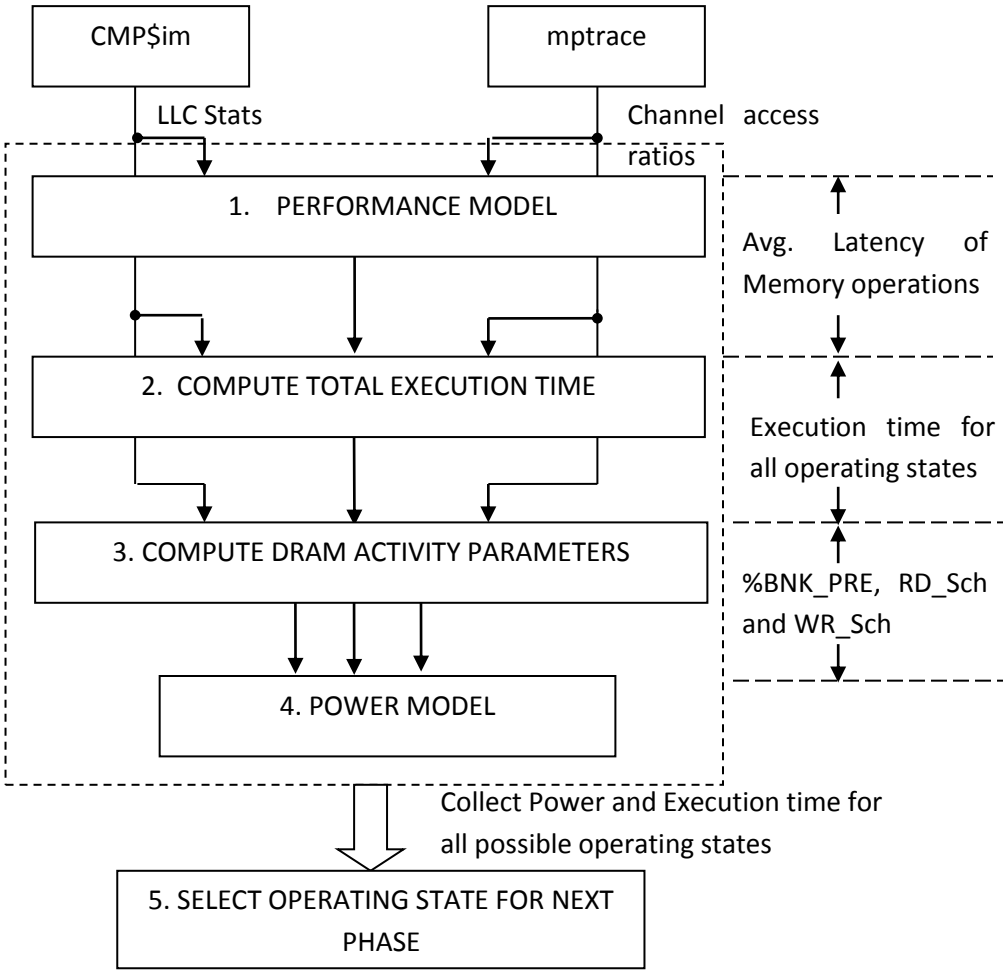

Figure 15: Block diagram of the evaluation methodology

Since CMPSim can model the application behavior only till the last level cache (LLC), mptrace is used to collect the physical address traces of all the benchmarks. Results from both CMPSim and mptrace are used as input by the control algorithm along with a performance and a power model. The performancemodel is used together with the LLC statistics (CMPSim) and channel access ratios (mptrace) to find the latency of read/write operations for all operating frequencies. After obtaining the average latency for a given frequency (f_{sys}) from the performance model, the actual execution time of the program at f_{sys} can be computed using Eq. 2.

$$N_c' = N_c - (350 \times Misses) + (Latency \times Misses)$$

N_c' - Cycles elapsed after compensating for LLC Misses

N_c' - Cycles elapsed before compensating for LLC Misses.

Equation 2: Total Execution Time

In Eq. 2, N_c' is the total execution time (in cycles) after compensating for LLC misses, and N_c is the total execution time (in cycles) produced by the simulator assuming that LLC misses are penalized with a 350 cycle latency. The execution time is then used to find the activity parameters that are computed for all possible operating states. Next, the power model is used to obtain the power dissipation, and the last step is selecting a state that provides maximum energy savings within allowable CPI degradation limits.

5.1. PERFORMANCE MODEL

CMP$im assumes a main memory access latency of 350 cycles for both read and write operations. Hence CPI provided by CMP$im is far from accurate. The figure below shows two memory controllers independently controlling channels.

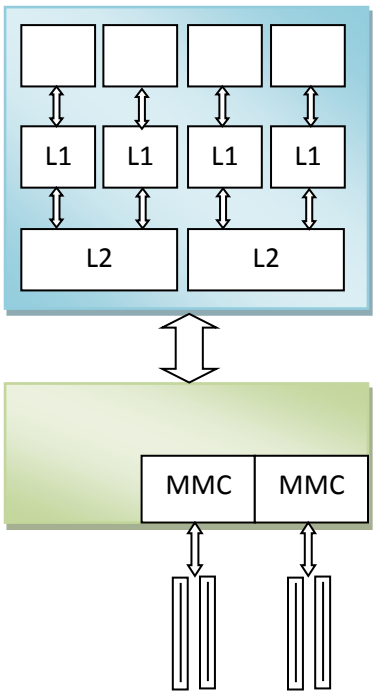

Figure 16: 4-Core Processor with two memory channels

A memory access (read/write) missing the last level cache will incur the delays shown in Eq. 3.

$$Latency = t_r + t_d \qquad for\ reads$$

$$= t_w + t_d \qquad for\ writes$$

Equation 3: Delays

Where t_R – Time for read access

t_W – Time for write access

t_d – Avg. Queuing delay incurred by a mem access

t_r and t_w include the time taken for complete transfer of data from a DRAM device back to (or from) CPU. We need a good estimate of t_d to calculate the total delay. Simplifying the operation of memory controller the queues in memory controller are shown below

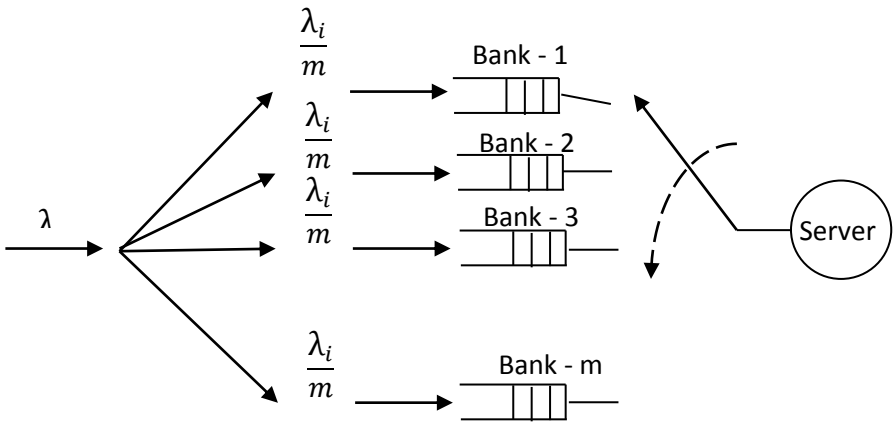

Figure 17: Simple model for memory controller

The memory controller maintains one queue per bank. λ_i is the arrival rate of requests in channel – i. We assume that all the banks are equally accessed. Hence the traffic bifurcates into all the queues equally. The request at the head of each queue is serviced on a round-robin basis. Time taken to service a request includes the complete access time (for read and write operations.). We model a closed page DRAM. A read access has row activation (t_{RCD}), column access (t_{CL}), data transfer (t_{BURST}) and a precharge command to close the row (t_{RP}). The timing parameter for a write operation is the same except that the column access takes up t_{CWL} and there is a t_{WR} interval between t_{BURST} and t_{RP}. Let t_R (t_W) denote the complete interval of time required to perform read (write) operation. So the memory controller has Poisson arrivals, generally distributed service time, single server and m traffic streams. This is an M/G/1 queue

with m users polling for the server. We can now estimate the average waiting time of a request in a queue (t_d). This can be used to calculate *Latency*.

The Probability Mass Function of service time of a request is shown in Eq. 4.

$$f_S(s) = \begin{cases} p_R, & s = t_R \\ p_W, & s = t_W \end{cases}$$

$$\overline{S} = E\{S\} = \frac{1}{\mu}$$

$$\overline{S^2} = E\{S^2\}$$

Equation 4: Probability Mass Function

Since we assume equal bank access, the waiting time of a request in the queue is given by waiting time of M/G/1 queue as shown in Eq. 5.

$$t_d = \frac{\lambda \overline{S^2}}{2(1 - \rho)}$$

Equation 5: Waiting Time

Where,

p_r - Probability that an access is read

p_w – Probability that an access is write

λ, p_r and p_w can be calculated from data collected from CMP$im and mtaptrace

CMP$im outputs instructions executed, LLC Misses on a per thread basis. This can be used to calculate the average interval between any two LLC misses as shown in Eq. 6.

$$\left.\begin{matrix} Average\ gap\ between \\ any\ two \\ LLC\ Misses \end{matrix}\right\}$$

$$= \frac{Intstructions\ Executed}{LLC\ Misses} \times \frac{Cycles - (350 * LLC\ Misses)}{(Instructions\ Executed - LLC\ Misses)} \times \frac{1}{f_{CPU}}$$

Let us denote this by τ.

$$\lambda_j = \frac{\alpha_{1j}}{\tau_1} + \frac{\alpha_{2j}}{\tau_2} + \ \ldots + \frac{\alpha_{kj}}{\tau_k}$$

Equation 6: Gap between two LLC Misses

Where, α_{kj} is fraction of accesses from thread-k going to channel j. This is obtained from mtaptrace results. Now, λ can be used in eq() to calculate average waiting time of a request to channel - j. t_d can be used to correct the total number of cycles elapsed.

5.2. POWER MODEL

In our simulations we have calculated active power (P_{ACT}), background power (P_{ACT_STBY} and P_{PRE_STBY}), read and write power (P_{RD} and P_{WR}) and termination power(P_{term}) according to the model for memory power described in [33]. The specific parameters required by the DRAM power model are listed below.

- **BNK PRE** Percentage of cycles that DRAM spent in pre-charge mode.
- **RD_Sch** Percentage of DRAM cycles that were outputting read data.
- **WR_Sch** Percentage of DRAM cycles that were outputting write data.

BNK PRE is used to compute background power while RD_Sch and WR_Sch are used to compute read, write and termination power. These parameters are described in detail in [33], and they can be computed using the performance model after obtaining the average latency of read and write operations. Eq. 7 computes the energy consumption, where P^f_{total} represents the total power dissipation, and T_{exec} the execution time at frequency f.

$$E_f = P_{total}^{f} \times T_{exec}$$

Equation 7: Energy Consumption

5.3. CMPSim SIMULATOR

CMPSim is a PIN [28][29] tool that intercepts memory access operations that are fed to a Chip Multiprocessor (CMP) cache simulator [30]. The model implements a detailed cache hierarchy with DL1/IL1, UL2, UL3 and memory. The simulator can be configured to model complex cache hierarchies, e.g., a SMP machine with 32 cores sharing the L2 and L3. In fact, the recent processor architectures can be modeled using CMPSim.

CMPSim can capture cache behavior of single and multithreaded workloads. CMPSim can gather a wide variety of statistics for an application, which are saved to an output file periodically. The log file contains information about instruction profile, total number of cache accesses and misses at all levels, and cache sharing between multiple threads, etc. Moses et al. [31] present a very detailed study of CMPSim.

5.4. MPTRACE

The Intel PIN project aims to provide dynamic instrumentation techniques to gather information about the instructions that applications execute. PIN API provides mechanisms to implement callbacks that are called where specific events occur on the execution of the target application (i.e., execution of memory access operation). Other tools that profile the the applications memory access can be found in the PIN SDK. However, no PIN tool or similar instrumentation tool has been provided to profile the physical memory accesses that applications request. Mptrace is a PIN-based tool that allows intercepting the processes memory accesses, and translating the virtual addresses to physical addresses. It uses the page map file [32] system to

translate the virtual address to physical address. The pagemap file system was released with version 2.6.25 of the Linux kernel and can be accessed through the */proc/pid/pagemap* file. As is described in the kernel source [32], this file allows a user space process to find to which physical frame each virtual page is mapped. It contains one 64-bit value for each virtual page, containing the following data (from *fs/proc/task mmu.c*):

- Bits 0-54: page frame number (PFN) if present
- Bits 0-4: swap type if swapped
- Bits 5-54: swap offset if swapped
- Bits 55-60: page shift (page size = 1 page shift)
- Bit 61: reserved for future use
- Bit 62: page swapped
- Bit 63: page present

Using the pagemap system, the mptrace PIN tools provides several functionalities to characterize how the applications access the physical memory pages. The format and information required is highly customizable, it provides information related to cache access (way and set), and memory accesses (physical page address). It also provides ways to reduce the amount of generated information, such as, sampling and trace disabling when the application loads data, or the caches are warming up. The current implementation of mptrace provides mechanisms to characterize the memory accesses on the flight. Thus, this PIN tool can provide summarized information about how an application is using the main memory. For example, it provides page access histograms, or clusters of memory regions accessed during an interval of time.

6. RESULTS

NAS Parallel Benchmarks (class B) were used to evaluate the potential of adaptive DRAM power management. CMPSim was first used with the architecture configuration shown in Table 2 to obtain statistics of the benchmarks execution (the characteristics of the different benchmark are shown in Table 3). Then, memory access patterns were obtained by running the benchmarks using mptrace on actual hardware (same configuration as shown in Table 2). Result were obtained for 4, 8 and 16 DDR3 channels with 2GB in each channel (single rank) using the power model described in [33]. Discussions pertaining to Energy consumption are only those of DRAM and not the whole system. Energy savings reported are savings with respect to operating all the channels at maximum frequency.

Feature	Specification
Cores	8 Cores, 2 HW threads per core, 2.4GHz
L1 Cache	32KB, 8-way set associative
L2 Cache	256KB, 16-way set associative 5cycles/hit
L3 Cache	16MB, 4-way set associative 15 cycles/hit

Table 2: System Specifications

BENCHMARK	DATA SET CLASS	INSTRUCTIONS
BT	B	70 billion
FT	B	79 billion
CG	B	66 billion

Table 3: Characteristics of the different NAS benchmarks

Fig. 18 shows energy savings obtained on a 4 DDR3 Channel system. All the three frequency search methods can be used on a 4-channel system. Almost equal energy savings are obtained

with both the mapping algorithms. Energy consumed by main memory is significantly reduced in case of all the three benchmarks. Average energy savings obtained with BT, FT and CG are 43.22%, 51.58% and 52.30%. It can be also seen in Fig. 16 that the increase in execution time is well within limits. Results obtained for both the mapping algorithms follow the same trend. Average energy savings obtained by controlling the channels independently are 44.85%, 51.09% and 52.57% while jointly controlling the frequency of all the channels reduces the energy consumption by 39.95%, 52.56% and 51.75%. There is significant improvement in energy savings when voltage and frequency levels of channels are tuned independently

BT has higher energy savings with 3 Level Search compared to exhaustive. The reason for this can be deduced by looking at Figs. 19, 20 and 21 showing operating frequency of channels.

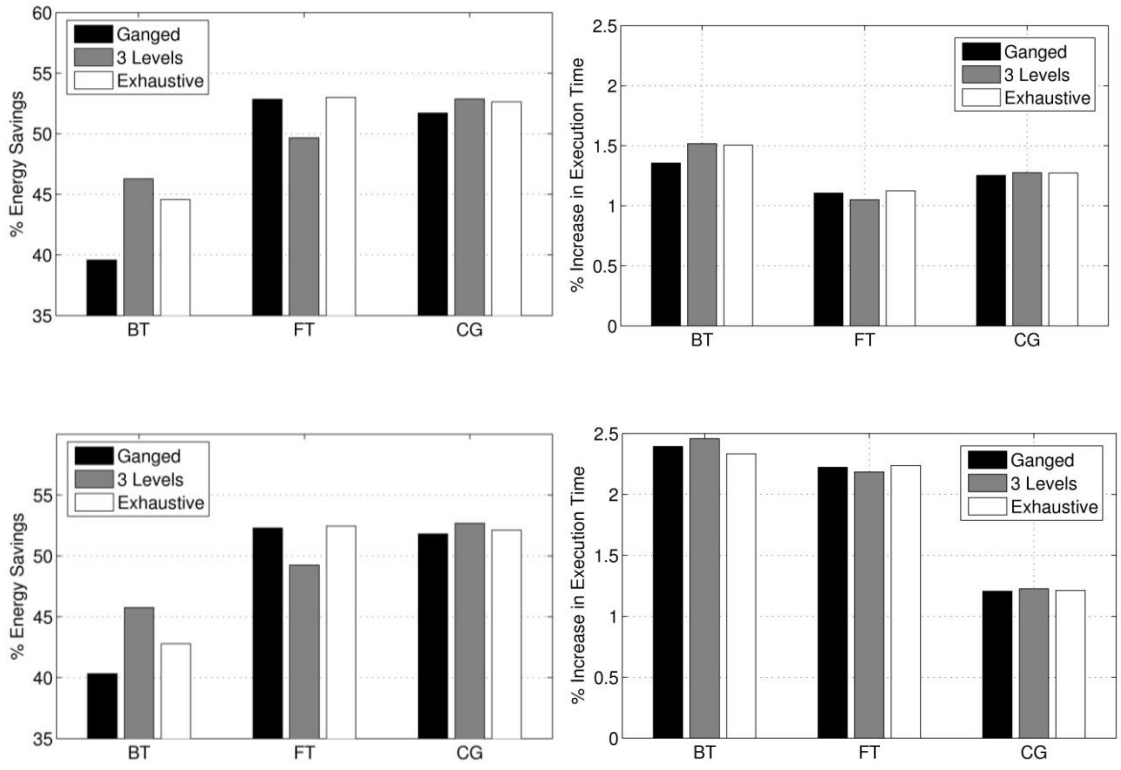

Figure 18: Percentage Energy Savings (left) and Percentage increase in execution time (right) of two mapping algorithms – Default (top) and Interleaved mapping(bottom)- 4 Channels

Operating frequencies are more are less the same except for the 900MHz switch which was avoided by three level search. Moreover looking at percentage increase in execution time exhaustive search gives the lowest increase while three level search's increased energy savings comes with higher execution time. Figs. 22, 23 and 24 shows the channel operating frequencies of FT for three different frequency selection methods. Once the system transitions to the lowest frequency all the frequency search methods operate more or less at the same frequency. FT Exhaustive search itself does not perform many frequency switches. This explains the reason behind almost equal energy savings with Exhaustive and Ganged search. The system transitions to the lowest frequency level in all the three cases but three level search gives lower energy savings because to make the transition to 400MHz all the channels should first transition

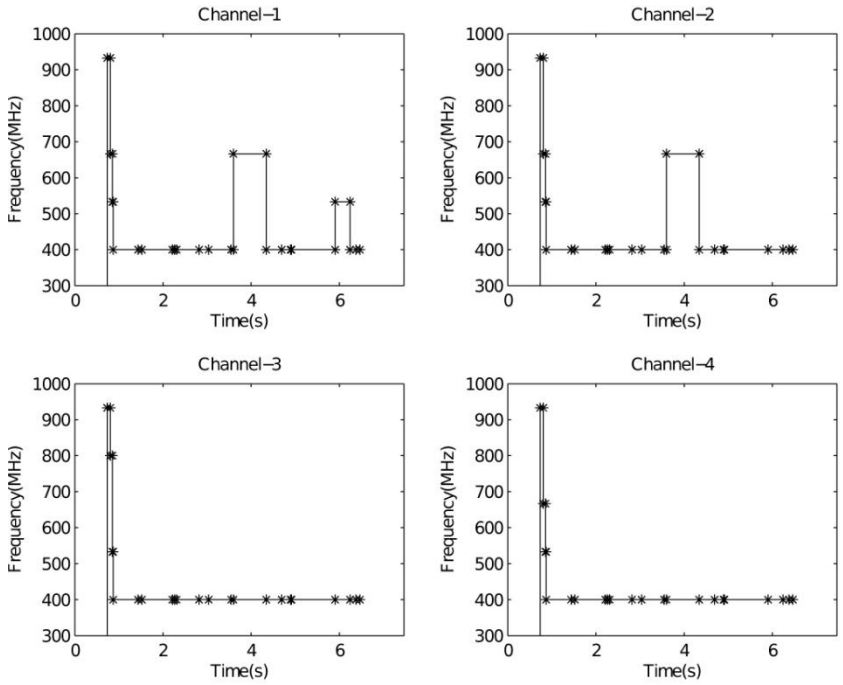

Figure 19: Frequency of Memory Channels of BT benchmark – Three Level Search

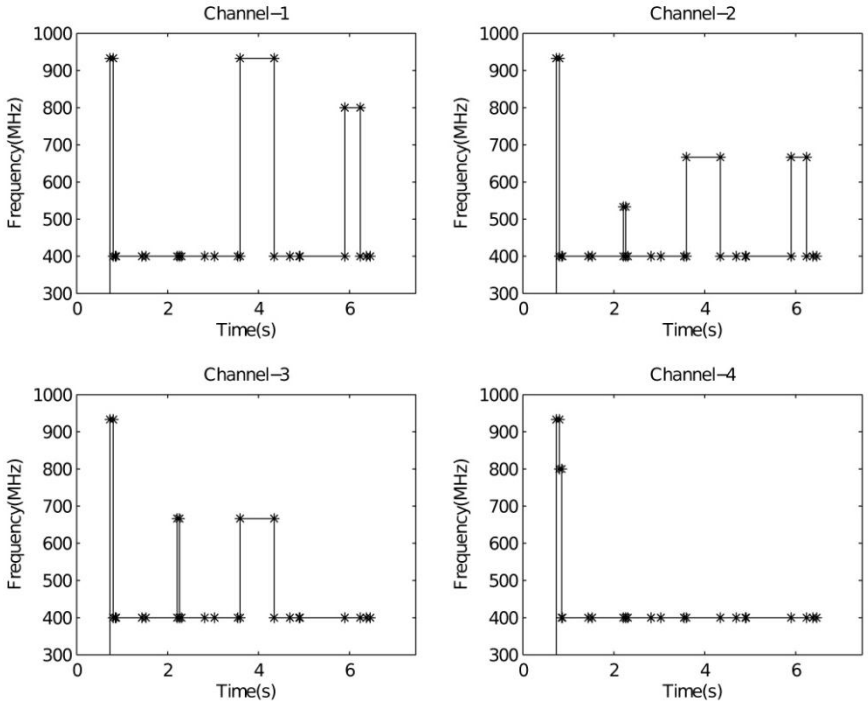

Figure 20: Frequency of Memory Channels of BT benchmark – Exhaustive Search

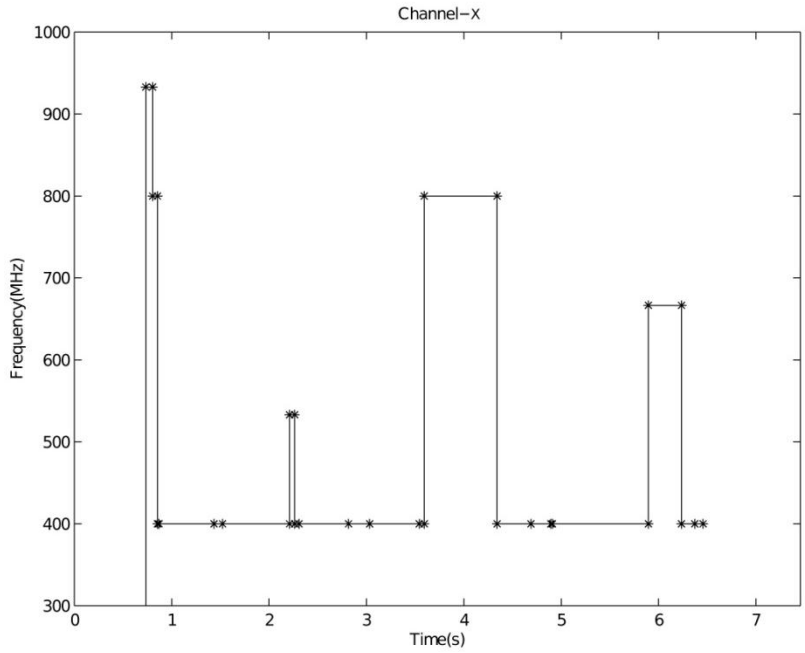

Figure 21: Frequency of Memory Channels of BT benchmark – Ganged

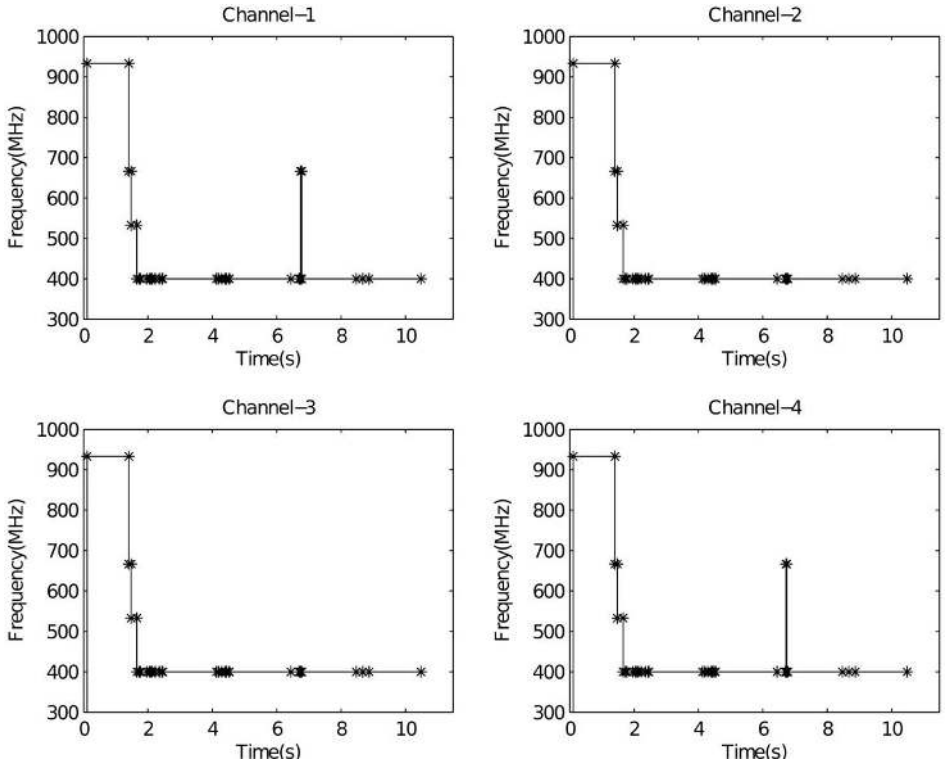

Figure 22: Frequency of Memory Channels of FT benchmark – Three Level Search

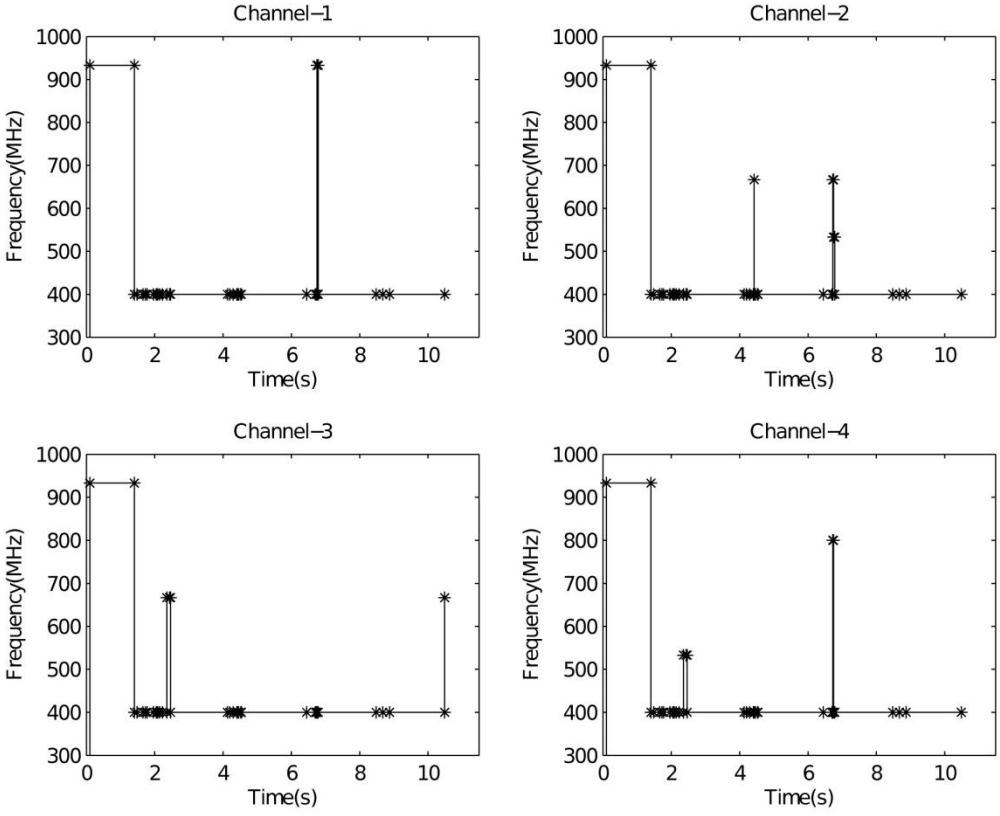

Figure 23: Frequency of Memory Channels of FT benchmark – Exhaustive Search

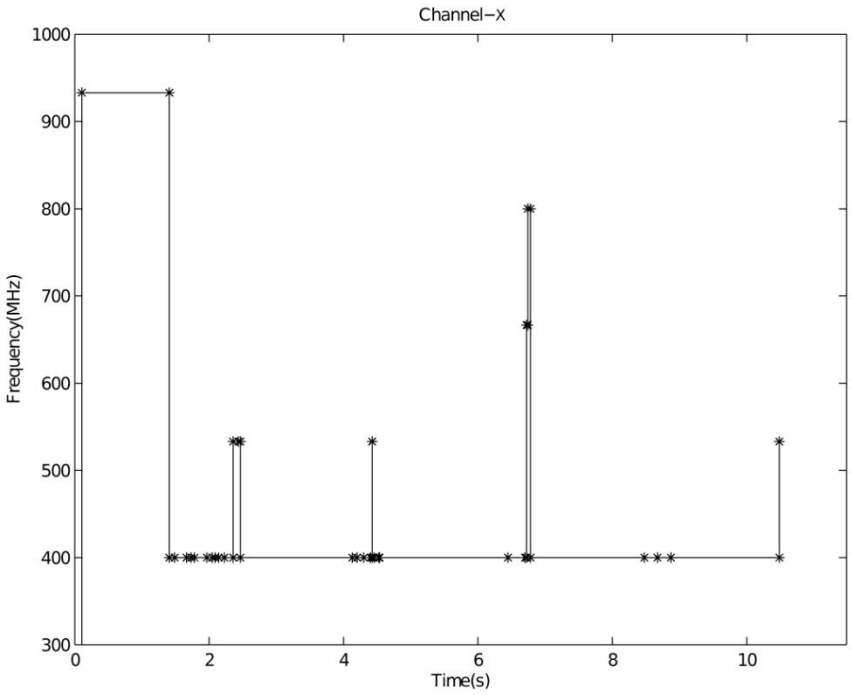

Figure 24: Frequency of Memory Channels of FT benchmark – Ganged

to 600MHz and then to 533MHz. Time spent in these two frequency levels increases the energy consumption of three level search. The same discussion can be extended to explain results obtained with default mapping algorithm.

Fig. 25 shows the energy savings obtained with 8 Channels. Average energy savings of BT, FT and CG are 43.64%, 51.95% and 52.93%. Controlling the voltage and frequency of channels independently gives 48.55%, 51.21% and 53.75% reduction in energy consumed while slaving all the channels together gives 38.73%, 52.68% and 52.09% energy savings. Energy savings obtained with BT is higher when channels are controlled independently with three level search. This holds for both the mapping algorithms. But BT shows the highest difference in energy savings between ganged and three level search with the default mapping algorithm. This is due to additional imbalance in channel traffic created by default mapping. Controlling channels

independently will present more opportunities to save power since channels which have very low traffic can be operated at the lowest frequency while the frequency of ones that are considerable loaded can be increased. This is not possible when all the channels are slaved together.

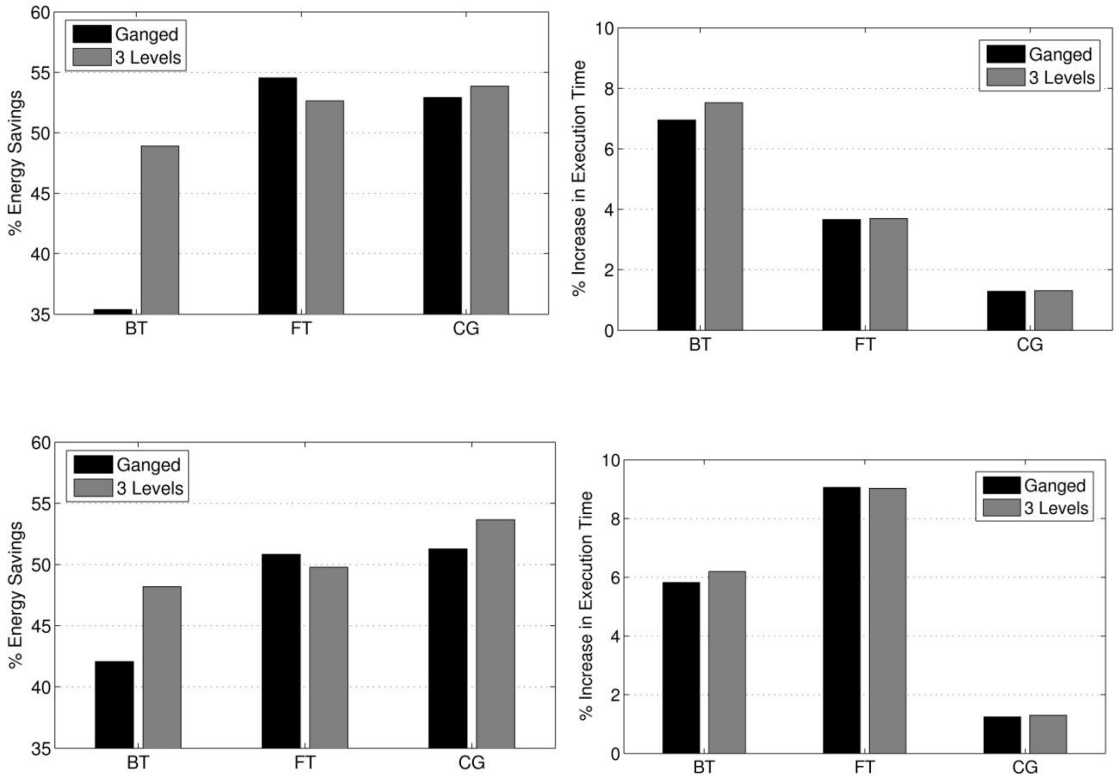

Figure 25: Percentage Energy Savings (left) and Percentage increase in execution time (right) of two mapping algorithms – Default (top) and Interleaved mapping (bottom) - 8 Channels

Figs. 26 and 27 shows operating frequencies of FT with Ganged and three level search. Three level search gives lower energy savings because all the channels have to transition to 600MHz and then to 533MHZ before going to the lowest operating frequency. The fraction of time spent in two intermediate frequencies levels contribute significantly towards total energy consumption thus lowering energy savings of three level search.

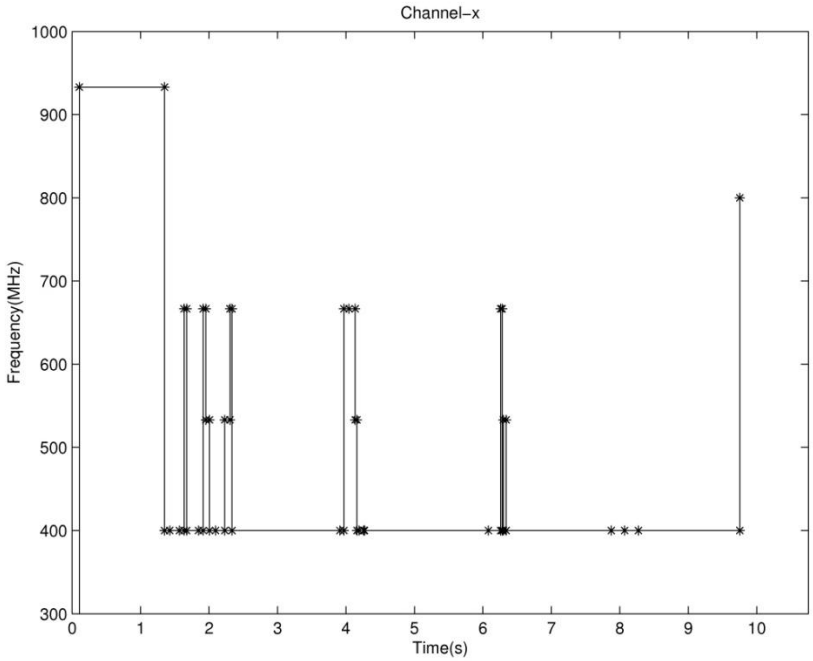

Figure 26: Frequency of Memory Channels of FT benchmark – Ganged

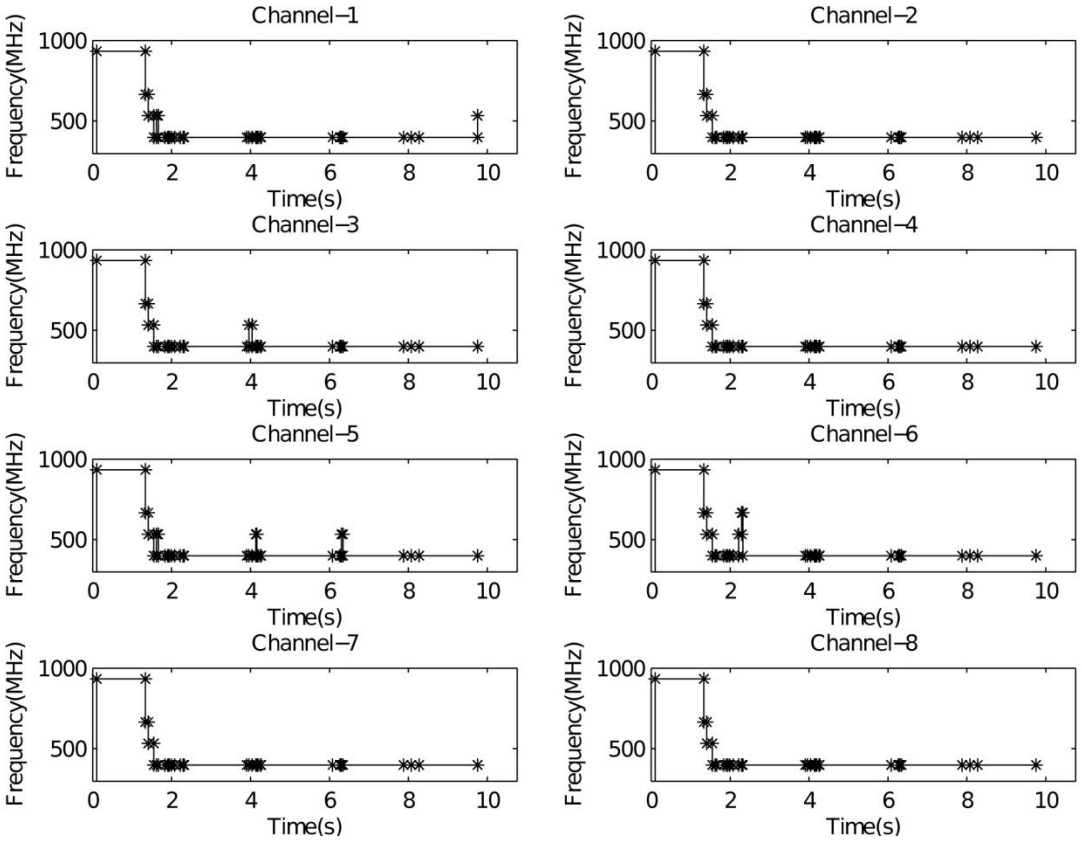

Figure 27: Frequency of Memory Channels of FT benchmark – Three Levels Search

Fig. 28 shows energy savings with 16 channels. All the three benchmarks give increased energy savings with interleaved mapping. With 16 Channels all the channels must be slaved together since it is not feasible to use exhaustive or three level search. As show in Fig. 14 default mapping completely lowers down activity in some channels and the distribution of traffic is significantly unequal across all the channels. Distribution of traffic with interleaved mapping is much more suited for ganged search.

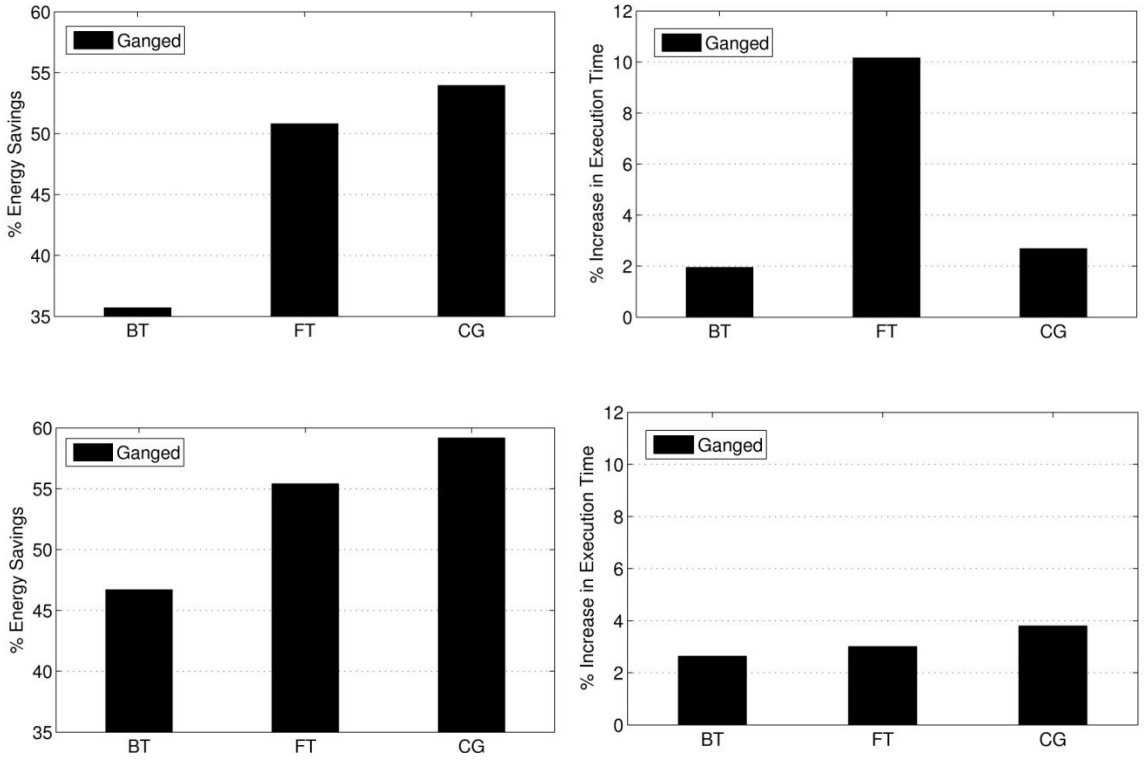

Figure 28: Percentage Energy Savings (left) and Percentage increase in execution time (right) of two mapping algorithms – Default (top) and Interleaved mapping (bottom) - 16 Channels

Two variants of ganged were also implemented for default mapping algorithm. With Ganged – 8, 8 Channels were grouped together (out of 16) effectively creating two virtual channels. Similarly with Ganged – 4, 4 Channels were grouped together creating four virtual channels. Energy savings obtained with Ganged – 8 and Ganged – 4 are show in Fig. 29. Grouping the channels is

as effective as controlling the channels independently. The reason behind lower energy consumption can be derived by looking at memory access patterns of benchmarks. With 16 channels all the benchmarks does not equally utilize all the channels. There are some channels that are almost idle. Coupling all the channels together hinders the control algorithm from lowering the frequency of certain channels with reduced activity. Hence grouping a subset of channels together provides freedom to the algorithm to operate different groups of channels at different frequencies which effectively reduces the energy consumption. Average energy savings of Ganged – 4 and Ganged – 8 are 44.56%, 52.56%, and 54.63%. It is clearly evident that significant savings in energy consumption is obtained by grouping channels together. Even though frequency of each channel is not changed independently, which may provide even higher energy savings but it is infeasible with 16 channels considering the sheer number of possible states the control algorithm has to consider, grouping the channels is as effective as exhaustive frequency search.

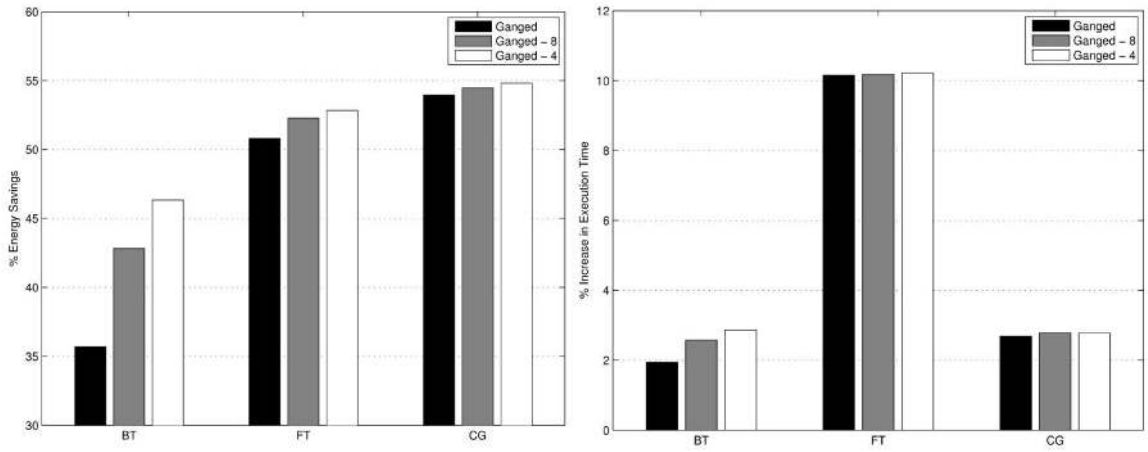

Figure 29: Percentage Energy savings (left) and Percentage Increase in Execution time (right)

On an average with 4 channels the applications give 48.1% energy savings with ganged search and 49.1% when the channels are controlled independently. When number of channels is

increased to 8 with ganged search average energy savings obtained is 48% while independently controlling channels gives 50.5%. These figures are for interleaved mapping with default mapping average energy savings are 48% with ganged search and 49.8% when the channels are controlled independently. Increasing number of channels to 8 increases energy savings from 47% (ganged) to 51.8% (independently controlled). Results from both the mapping algorithms shows that controlling channels independently becomes very effective with 8 or more channels.

7. CONCLUSIONS AND FUTURE WORK

In this thesis, Dynamic Voltage and Frequency memory Scaling was proposed to reduce energy consumption considering the ability to select different frequencies for different memory channels. The analysis of HPC applications memory bandwidth demand showed that there are significant fluctuations in the memory bandwidth demand over time, and that the memory traffic is unequally distributed to all channels. The results obtained with different number of channels, mapping algorithms and frequency selection methods show that DVFS is an effective technique to significantly reduce the energy consumed by main memory while maintaining performance degradation within tolerable limits. The results also showed that controlling the channels independently provides considerable savings with respect to controlling the frequency of all the channels together, and controlling the channels independently is more effective when number of channels is larger.

As a part of future work, the current approach will be extended with an even fine-grained simulations and a sophisticated performance model that incorporates complex scheduling strategies used by modern memory controllers. Additional benchmarks that might exhibit higher memory access imbalance will be considered as well as additional parameters such as different number of cores and different memory technologies. Finally, control algorithm will be improved with predictive strategies, such as those based on phase detection techniques.

8. PUBLICATIONS

The research presented in this thesis has resulted in the following paper, which is currently under review.

- K. Elangovan, I. Rodero, M. Parashar, F. Guim and I. Hernandez, “Adaptive Memory Power Management Techniques for HPC Workloads”, 18th International Conference on High Performance Computing, HiPC 2011.

REFERENCES

[1] “Report to congress on server and data center energy efficiency,” U.S. Environmental Protection Agency, Tech. Rep., August 2007.

[2] L. A. Barroso and U. Holzle, “The case for energy-proportional computing,” *Computer*, vol. 40, pp. 33–37, 2007.

[3] I. Rodero, S. Chandra, M. Parashar, R. Muralidhar, H. Seshadri, and S. Poole, “Investigating the potential of application-centric aggressive power management for hpc workloads,” in *International Conference on High Performance Computing (HiPC)*, 2010, pp. 1 –10.

[4] “Jedec. ddr3 sdram standard,” 2009.

[5] E. N. Elnozahy, M. Kistler, and R. Rajamony, “Energy-efficient server clusters,” in *Proceedings of the 2nd international conference on Power-aware computer systems*, 2003, pp. 179–197.

[6] N. Kappiah, V. W. Freeh, and D. K. Lowenthal, “Just in time dynamic voltage scaling: exploiting inter-node slack to save energy in MPI programs,” in *ACM/IEEE conference on Supercomputing (SC)*, 2005, p. 33.

[7] R. Ge, X. Feng, and K. W. Cameron, “Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters,” in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, 2005, p. 34.

[8] F. Yao, A. Demers, and S. Shenker, “A scheduling model for reduced cpu energy,” in *36th Annual Symposium on Foundations of Computer Science*, 1995, p. 374.

[9] D. Zhu, R. Melhem, and B. R. Childers, “Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real time systems,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, July 2003.

[10] Q. Cai, J. Gonzalez, R. Rakvic, G. Magklis, P. Chaparro, and A. Gonz´alez, “Meeting points: using thread criticality to adapt multicore hardware to parallel regions,” in *International Conference on Parallel Architectures and Compilation Techniques*, 2008, pp. 240–249.

[11] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, “Hardware and Software Techniques for Controlling DRAM Power Modes,” *IEEE Trans. Comput.*, vol. 50, no. 11, pp. 1154–1173, 2001.

[12] V. Delaluz, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, “Energy-oriented compiler optimizations for partitioned memory architectures,” in *International conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES’00)*, 2000, pp. 138–147.

[13] V. Delaluz, M. Kandemir, and I. Kolcu, “Automatic data migration for reducing energy consumption in multi-bank memory systems,” in *39th Design Automation Conference (DAC’02)*, 2002, pp. 213–218.

[14] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, “Scheduler-based DRAM energy management,” in *39th Design Automation Conference (DAC’02)*, 2002, pp. 697–702.

[15] Y. Cho and N. Chang, “Memory-aware energy-optimal frequency assignment for dynamic supply voltage scaling,” in *International Symposium on Low Power Electronics and Design (ISLPED’04)*, 2004, pp. 387–392.

[16] M. C. Huang, J. Renau, and J. Torrellas, “Positional adaptation of processors: application to energy reduction,” in *30th International Symposium on Computer Architecture (ISCA’03)*, 2003, pp. 157–168.

[17] H. B. Fradj, C. Belleudy, and M. Auguin, “System level multi-bank main memory configuration for energy reduction,” in *International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2006, pp. 84–94.

[18] H. B. Fradj, C. Belleudy, and M. Auguin, “Multi-bank main memory architecture with dynamic voltage frequency scaling for system energy optimization,” in *Euromicro Conference on Digital System Design (DSD)*, 2006, pp. 89–96.

[19] X. Li, Z. Li, F. David, P. Zhou, Y. Zhou, S. Adve, and S. Kumar, “Performance directed energy management for main memory and disks,” in *11th International conference on Architectural support for programming languages and operating systems*, 2004, pp. 271–283.

[20] X. Li, Z. Li, Y. Zhou, and S. Adve, “Performance directed energy management for main memory and disks,” *ACM Transactions on Storage*, vol. 1, no. 3, pp. 346–380, 2005.

[21] X. Li, R. Gupta, S. V. Adve, and Y. Zhou, “Cross-component energy management: joint adaptation of processor and memory,” *ACM Trans. Archit. Code Optim.*, vol. 4, no. 3, p. 14, 2007.

[22]B. Diniz, D. Guedes, W. Meira, Jr., and R. Bianchini, “Limiting the power consumption of main memory,” in *34th International Symposium on Computer Architecture (ISCA’07)*, 2007, pp. 290–301.

[23] I. Hur and C. Lin, “A comprehensive approach to DRAM power management,” in *14th International Conference on High-Performance Computer Architecture (HPCA)*, 2008, pp. 305–316.

[24] H. Zheng, J. Lin, Z. Zhang, and Z. Zhu, “Decoupled dimm: building high-bandwidth memory system using low-speed dram devices,” in *36th International symposium on Computer architecture*, 2009, pp. 255–266.

[25] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu, “Mini-rank: Adaptive dram architecture for improving memory power efficiency,” in *41st IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 210–221.

[26] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, “Rethinking dram design and organization for energyconstrained multi-cores,” in *37th International symposium on Computer architecture*, 2010, pp. 175–186.

[27] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, “Memscale: active low-power modes for main memory,” in *6th International conference on Architectural support for programming languages and operating systems*, 2011, pp. 225–238.

[28] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, , and K. Hazelwood, “Pin: building customized program analysis tools with dynamic instrumentation.” ACM SIGPLAN Conference on Programming Language Design and Implementation, 2005.

[29] K. Hazelwood, G. Lueck, and R. Cohn, “Scalable support for multithreaded applications on dynamic binary instrumentation systems,” in 2009 International Symposium on Memory Management (ISMM), Dublin, Ireland, June 2009, pp. 20–29.

[30] A. Jaleel, R. S. Cohn, C. keung Luk, and B. Jacob, “Cmpsim: A pin-based on-the-fly multi-core cache simulator,” in Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), 2008.

[31] J. Moses, K. Aisopos, A. Jaleel, R. Iyer, R. Illikkal, D. Newell, and S. Makineni, “Cmpschedsim: Evaluating os/cmp interaction on shared cache management,” in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009, pp. 113 –122.

[32] “Pagemap - Linux Kernel - Documentation / vm / pagemap.txt,” January 2011.

[33] Micron, “Calculating memory system power for ddr3,” July 2007.