Lehrstuhl für Integrierte Systeme
der Technischen Universität München

Development of a fast DRAM Analyzer and Measurement of Typical and Critical Memory Access Sequences in Applications

Simon Albert

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzende:	Univ.-Prof. Dr. rer. nat. Doris Schmitt-Landsiedel
Prüfer der Dissertation:	
	1. Univ.-Prof. Dr. Ing. Ingolf Ruge (em.)
	2. Univ.-Prof. Dr. Ing. Klaus Diepold

Die Dissertation wurde am 25.02.2008 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 23.07.2008 angenommen.

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über
http://dnb.d-nb.de abrufbar.

ISBN 978-3-89963-888-2

1. Auflage 2008

Contents

List of Figures

List of Tables

List of Acronyms

AIO	Asynchronous Input Output
AMB	Advanced Memory Buffer
AGP	Advanced Graphics Port
BL	Burst Length
CA-Bus	Command and Address Bus
CAS	Column Address Strobe
CBR	CAS before RAS refresh
CL	CAS Read Latency
CPU	Central Processing Unit
CSA	Communication Streaming Architecture
DDR	Double Data Rate
DIMM	Dual Inline Memory Module
DMA	Direct Memory Access
DQ	Data Query (Data Bus)
DRAM	Dynamic Random Access Memory
DSL	Digital Subscriber Line
DVD	Digital Versatile Disk
EDO	Extended Data Output
FB-DIMM	Fully Buffered DIMM
FSB	Fronside Bus
FPGA	Field Programmable Gate Array
FPM	Fast Page Mode
GPU	Graphics Processing Unit
HD	hard disk
ICH	IO Controller Hub
IEC	International Electrotechnical Commission
IP	Internet Protocol
ISI	Intersymbol Interference
Ki...	Kibi... (= 2^{10}) Prefix specified in IEC 60027-2
LAN	Local Area Network

LCD	Liquid Crystal Display
LSB	Least Significant Bit
LVPECL	Low Voltage Positive Emitter Coupled Logic
MAC	Media Access Control
MCH	Memory Controller Hub
Mi...	Mebi... (= 2^{20}) Prefix specified in IEC 60027-2
MMU	Memory Management Unit
NFS	Network File System
OS	Operating System
PABX	Private Automatic Branch Exchange
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PCIe	PCI Express
PDA	Personal Digital Assistant
PEG	PCI Express for Graphics
PLL	Phase Locked Loop
PXE	Preboot Execution Environment
RAS	Row Address Strobe
SDR	Single Data Rate
SDRAM	Synchronous Dynamic Random Access Memory
SMP	Symmetric Multiprocessing
SRAM	Static Random Access Memory
TLB	Translation Lookaside Buffer
UDP	User Datagram Protocol
USB	Universal Serial Bus

Chapter 1

Introduction

1.1 Evolution of Computer Systems

Since the early eighties computer system performance has increased dramatically. Starting from some few megahertz, CPUs operating frequencies reach multiple gigahertz today. Similarly DRAM sizes have increased from some few kilobytes to multiple gigabytes even in cheap desktop PCs.

In addition, penetration of daily life with digital electronic has increased substantially. Today DRAM memory is no longer used in personal computers, workstations or servers only, but can be found in a wide range of applications from personal digital assistants (PDA), mobile phones, graphics cards, networking equipment (switches, routers, ...), entertainment devices (game consoles, digital TV sets, DVD players, set top boxes, ...), or peripheral components like printers or scanners. All these systems have different requirements to the memory system: Portable equipment enforces low power consumption as devices are usually powered by (rechargeable) batteries. Graphic cards require large bandwidth, while personal computers also profit from low latencies.

The increase in overall system performance also facilitates more demanding applications like audio and video (de)coding, image processing, web servers or multitasking and multiuser environments.

These performance improvements result not only from improvements in semiconductor manufacturing technology like smaller feature sizes and new materials but also from the implementation of even more complex architectures. Systems using bus master direct memory access (DMA), symmetric multiprocessing (SMP), out-of-order and speculative execution or providing multiple execution units on one chip (superscalarity) are no longer limited to the domain of supercomputers but can be bought off-the-shelf in every computer store.

Product cycles which are often even shorter than two years impose new chal-

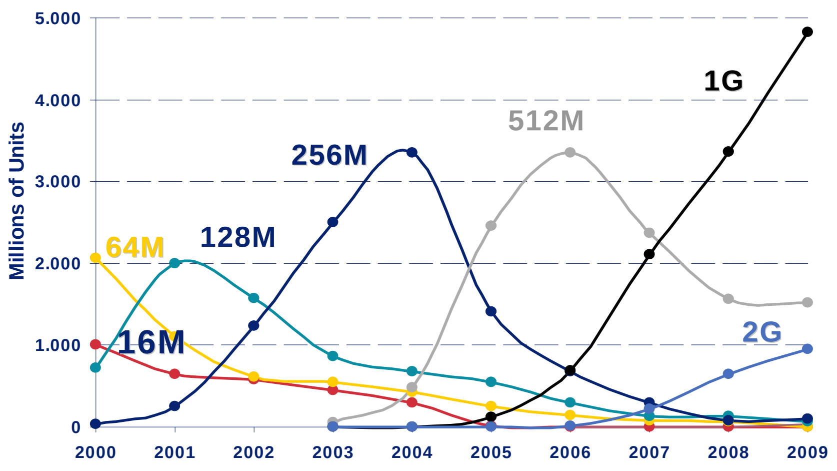


Figure 1.1: Worldwide DRAM Component Sales by Generation
(2004–2007 values are estimated) [28]

lenges to system developers and test engineers in order to keep pace with the upcoming technology changes (see figure 1.1).

The results of many traditional techniques which currently support system design engineers decisions regarding memory systems performance and power requirements have to be challenged or are no longer applicable for designing highly complex systems: Former researchers using system simulation either had the option to analyze only short program runs due to excessive simulation times of their simulation models or to extensively reduce the complexity of their simulation models making them less realistic. Researchers using measurements for memory system research also faced short measurement times due to low capturing bandwidth/low memory of their measurement equipment. The measurement setup which was developed during the thesis overcomes this severe limitation. It allows the capturing of long undisturbed command and address bus sequences from the DDR2 memory bus on real computer systems even over multiple hours.

Those *memory traces* provide system designers with a new and efficient way to gain new insights for the development of future SDRAM memory systems.

1.2 Outline

Chapter 2 provides a review of current microcomputer architectures, SDRAM memories and alternate primary memory technologies which might be used as a replacement for SDRAMs.

Chapter 3 compares the two methods used for computer architecture evaluation: simulations and measurements for its pros and cons.

Chapter 4 explains the measurement setup developed during the thesis work for capturing trace sequences at the SDRAM bus.

From the gathered access sequences, those SDRAM parameters are determined which limit the system performance most significantly. A simple statistical model is derived to estimate the effect of changes of these SDRAM parameters on the system performance. The estimates of the statistical model are then compared with real measurements.

As all measurements have been done using a multitasking operating system, the reproducability of the measurements is a concern. Chapter 6 investigates the impact of an unknown computer system starting state and limited control of task scheduling and memory assignment.

Chapter 7 summarizes the findings of the thesis and concludes with an outlook to future SDRAM research activities, which are supported by the developed measurement hardware.

Chapter 2

Memory Systems

2.1 Memory Hierarchies

Traditionally, computer storage is integrated in a hierarchical memory system consisting of different memory types with different access times and cost profiles.

This concept turned out to be effective, as usually accesses to instruction and data memory are not distributed equally over the available memory but memory access sequences comprise spatial and temporal locality. Temporal locality denotes the effect that memory locations which have been recently accessed will be accessed in the near future again. Spatial locality means that accesses to memory locations adjacent to memory locations which have been recently accessed are more probable than accesses to memory locations which are far away [26, p.47].

Therefore, it is a good idea to store data items which are used frequently in fast cache memory "near" the CPU or even in processor registers, while data items which are rarely used can reside in cheaper DRAM memory or can even be held in secondary or even tertiary storage. The transmission of data items between different levels in the memory hierarchy is done by dedicated logic or may require support from the operating system or even user mode applications depending on the level in the hierarchy (see figure 2.1).

2.2 Random Access electrically rewritable Memory

For the design of a primary memory system different types of memory can be taken into account. These memory technologies are either volatile (they loose the stored information when the power is shut off) or non-volatile (the information is kept even if no power is applied to the device).

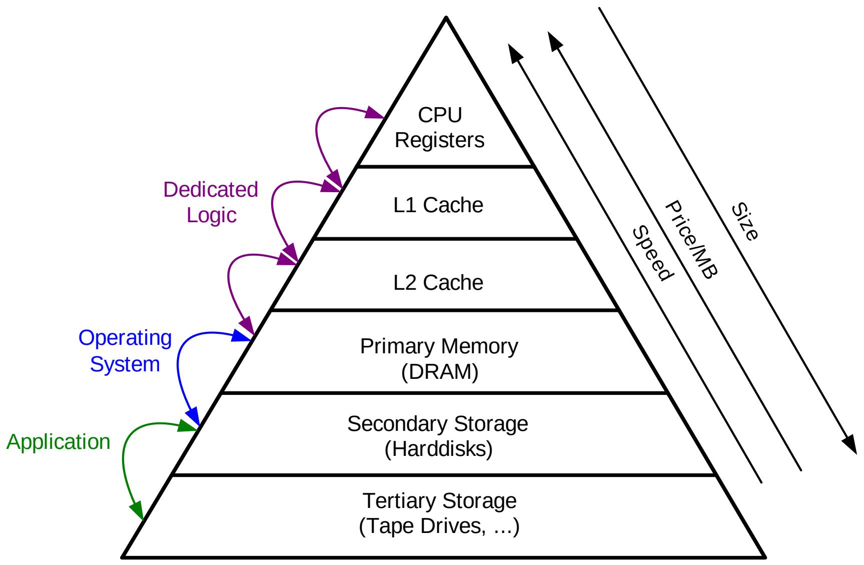


Figure 2.1: Memory Hierarchy

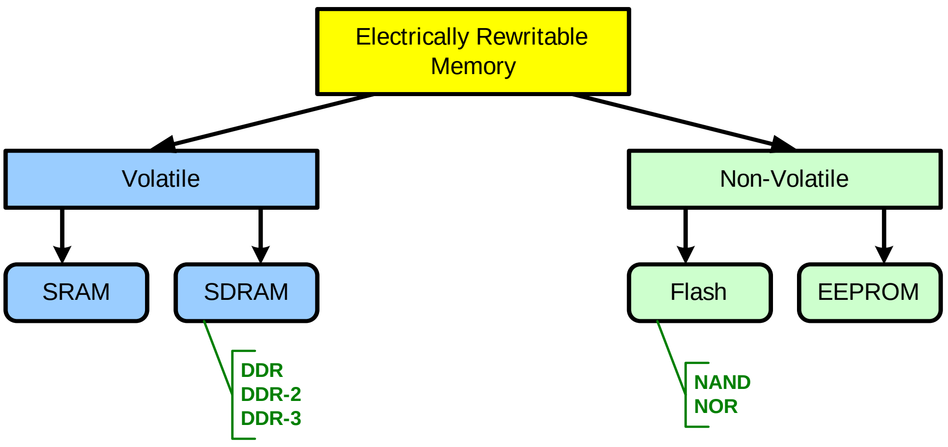


Figure 2.2: Primary Memory

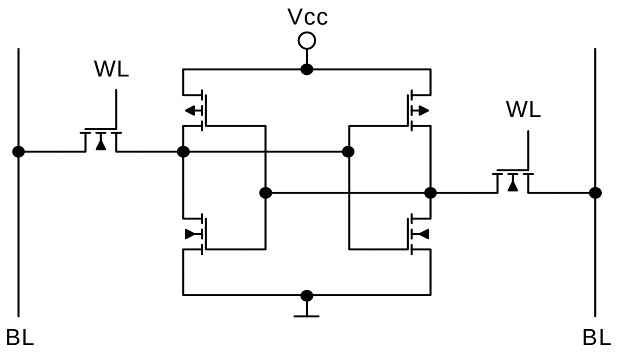


Figure 2.3: SRAM Cell
[27]

2.2.1 SRAM

The SRAM cell is a RS-flipflop consisting of four transistors plus two transistors for selecting the addressed flipflop [27, p.362]. This six transistor topology provides low access times and does not need any periodic refreshment of the SRAM cell due to the regenerative feedback of the transistors (see figure 2.3) . Unfortunately, having six transistors for one single SRAM cell leads to extremely high silicon area requirements. Today the use of SRAM memory arrays in the Gigabit range seems to be not acceptable from the cost to performance point of view. Therefore, SRAM is mostly used as cache memory only. In this application it is today almost allways implemented directly on the CPU die.

2.2.2 Flash Memory

Flash memory is an electrically erasable and programmable non volatile memory. The information is stored in an array of floating gate transistors. A NOR Flash cell consists of a single field effect transistor with two gate electrodes (control gate and floating gate). The NOR flash cell is programmed by setting a high voltage to the control gate. The electric field between gate and the drain/source path leads to a hot-electron injection onto the floating gate. To erase the cell, a negative voltage is set at the control gate, which allows a tunneling of electrons from the floating gate to the source electrode.

The captured charge on the insulated floating gate leads to a shift of the drain-source-current over gate-voltage curve, which can be detected and evaluated as either ”1” or ”0” by a sensing circuit. Todays flash memories even store multiple bits in one single cell by detecting small changes of the UI-Curve.

The charge on the floating gate remains even if the power is turned off (*non* *volatile memory*). Manufacturers often guarantee a data retention of multiple years. As programming and erasing is done by the tunnel effect, write access times are high (in the range of some µs [27, p.347]). Furthermore, programming and erasing cycles cause permanent damage to the cell. Typically after around 10^6 program erase cycles the cells content can no longer be determined reliably [27, p.348]. Therefore, some mechanisms have been invented in order to equalize the number of reprogramming cycles across the memory array by changing the address mapping of memory dynamically [39]. This mechanism is called *wear leveling*.

In addition, if erasing of single memory cells within the cell area is desired, a second transistor is required for each cell. In order to keep silicon size requirements low, most flash memory devices allow only an erase operation on multiple cells (*sector erase*).

This makes flash memory usable as non volatile data storage (e.g. in portable devices like digital cameras, MP3 players or USB sticks) but unusable as the only primary memory system due to the limited number of write cycles and the sector erase function.

2.2.3 SDRAM

The SDRAM cell consists of a single capacitor and a select transistor. The information is stored as charge within the capacitor. Due to leakage currents the capacitors need periodic refreshment of the SDRAM cell's contents. Access times are higher than for SRAM memories, but the single transistor/capacitor cell provides significant savings in silicon area size compared to SRAMs. These savings allow the manufacturing of SDRAM components currently containing up to 2^{31} SDRAM cells per chip. Thus, SDRAM memory currently is the prefered choice for primary memory systems.

2.3 Hitting the Memory Wall

Unlike improvements in CPU design, improvements in DRAM memory design are mostly driven by an increase in memory size but not by an increase in memory speed [31].

While the architecture of the processor to DRAM interface was modified several times within the last years evolving from "traditional" DRAM over fast page mode (FPM) and extended data output (EDO) DRAM to single and double data rate synchronous DRAM focusing on improvements in DRAM bandwidth, the internal memory cell array remained mostly unchanged.

While improvements in bandwidth can easily be achieved by an increase in parallelism, meaning that multiple bits are read at the same time and are then forwarded to the CPU, the duration of one single read or write access was not significantly decreased. Unfortunately, an improvement in bandwidth does not necessarily mean that CPUs execution time decreases significantly (e.g. when the access locality is poor) [31].

This leads to the situation that the memory system becomes a bottleneck in future computers and programs execution speed will be mostly determined by the performance of the memory system and not by the processors operating frequency [57].

Moreover, with the introduction of fast page mode memory current DRAMs are no longer true "random access" memories, in the sense that the access time of all data elements is always the same. Instead, the DRAM device can handle multiple transactions at the same time and access times depend on the current DRAMs state. Therefore, the memory controller has some degree of freedom regarding the scheduling of commands sent to the DRAM device.

Furthermore, DRAM market is driven by price. Although most manufacturers sell memory products with reduced latency and higher throughput, the application of these memory types is limited to specific applications like graphics cards as most performance boosts are correlated with higher pricing due to larger die sizes or increased packaging costs due to higher pin counts. In addition, computer system manufacturers often prefer highly standardized commodity products to reduce their economic dependency on specific DRAM manufacturers.

Thus, DRAM manufacturers have to evaluate proposals of DRAM interface improvements thoroughly to evaluate if the increase in system performance justifies an increase in pricing.

2.4 Computer System Architecture

2.4.1 Modern Computer Systems

Harvard vs. von Neumann Architecture

The Harvard architecture implements physically separate storage and signal paths for instructions and data, enabling the transmission of both at the same time: Systems using the Harvard architecture can be found mostly in the area of digital signal processing. Furthermore, it is used for on chip busses within the CPU. As the memory has to be allocated to the separate busses in advance, the Harvard architecture is less suitable for versatile computer systems which shall be used in a large variety of applications. By contrast, the von Neumann[1] architecture

[1]John von Neumann, Hungarian-American mathematician, 1903–1957.

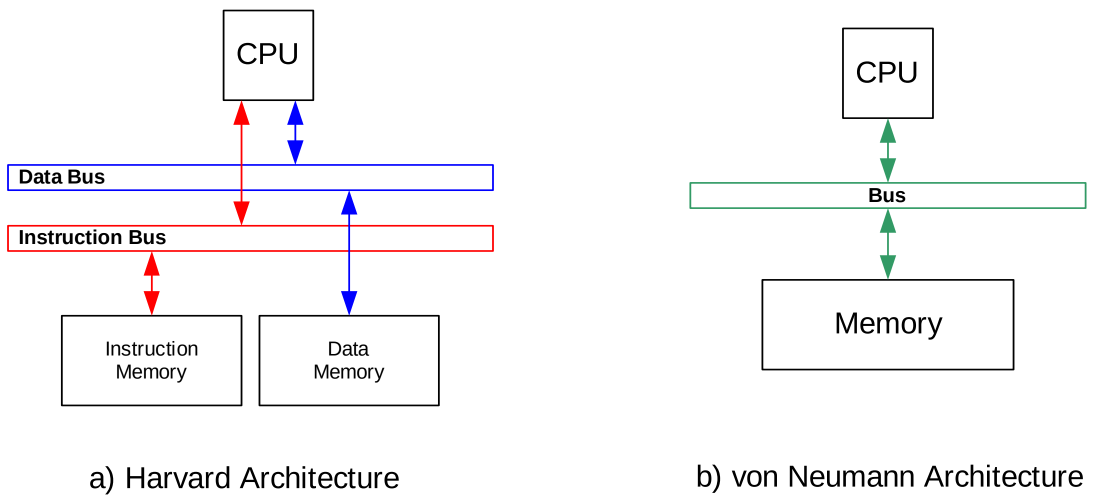


Figure 2.4: Von Neumann vs. Harvard Architecture

handles instructions and data equally and transfers both from main memory over one common databus.

Most modern computer systems are a mixture of both architectures. Within the CPU core instructions and data are handled separately up to the level 1 cache while transactions from the L1 to the L2 cache and main memory are handled over the same system bus.

Personal Computer System

Figure 2.5 shows a typical modern personal computer system. On top the CPU including L1 and L2 cache, which is connected via the frontside bus to the memory controller, is shown. The memory controller is also called north bridge[2]. Newest processor generations like the AMD Opteron include the memory controller on the CPU as well. The north bridge handles all memory transactions between the CPU (or multiple CPUs on multiprocessor main boards) and the DRAM memory. Additionally, it provides a link to the graphics card (AGP) and to the south bridge. On latest chipsets the AGP port is replaced by a more general point to point high speed interconnect called PCI Express (PCIe), which can be used either for communication with the graphics card (PCI Express for Graphics, PEG) or other peripheral components with high bandwidth requirements like Gigabit Ethernet network adapters or hard disk controllers. Some north bridges also provide an integrated graphics card, which uses the main memory as graphics DRAM (shared

[2]The company Intel uses the abbreviation memory controller hub (MCH).

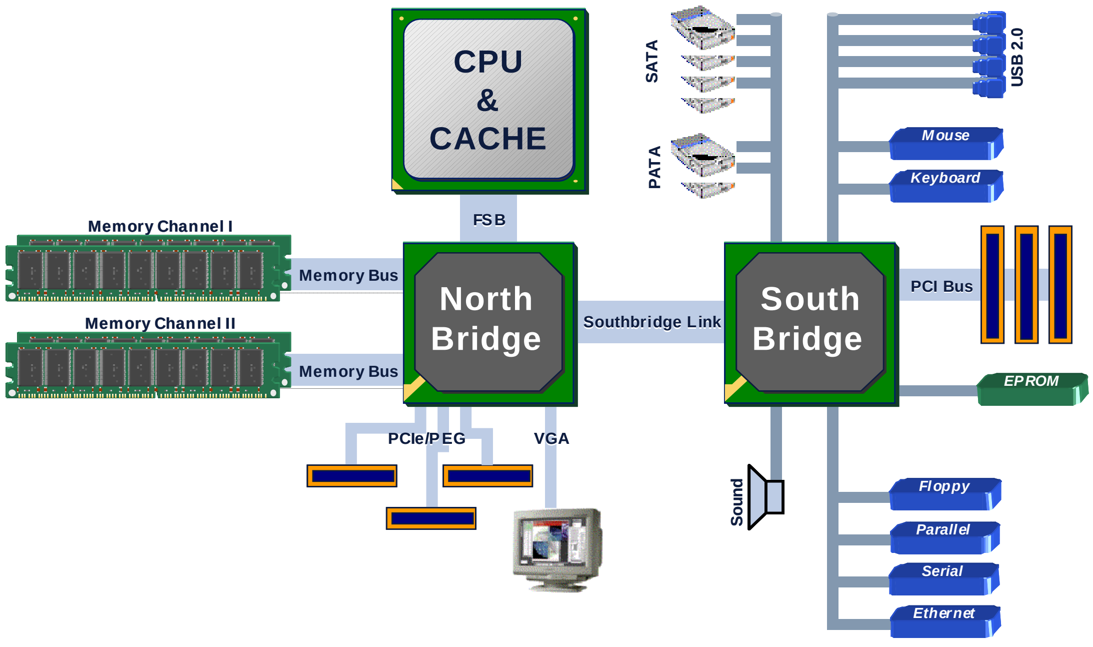


Figure 2.5: Personal Computer System

memory graphics). Some north bridges may also provide additional (proprietary) interfaces e.g. for network controllers[3].

The south bridge[4] is connected to the north bridge and contains most peripheral components found in typical PCs: the hard disk and floppy controller, USB, FireWire and PCI bus connections, (wireless) LAN, interfaces for mouse, keyboard or legacy IO (serial and parallel ports) or integrated sound. The north bridge/south bridge pair is also referred to as the PCs *chipset*.

Embedded Systems

While the concept of most embedded systems is similar to that of standard PCs, size and power requirements are a concern. Unlike PCs, the embedded CPU does not only include the processor core and the cache, but also integrates the functionality of the memory controller and peripheral components required for the particular application: LC-display, touchscreen, flash disk controller, sound, etc. for PDAs and mobile phones, various interfaces for networking or telecommunication equipment like DSL modems or PABX systems.

[3]e.g. Intel CSA (Communication Streaming Architecture).

[4]Intel also uses the term IO Controller Hub (ICH) when referring to the south bridge.

Caching

Caching provides a mechanism to hold a copy of frequently used data items in a fast static random access memory (SRAM) memory nearer to the CPU, providing data faster than standard DRAM memory. CPU references to memory for which a copy in the cache exist are served by the fast cache while references to memory locations which are currently not located in the cache are forwarded to the memory controller and reloaded from primary memory. A copy of the loaded data is placed in the cache, evicting some other data which is hopefully no longer required in the near future.

While up to the mid nineties cache memory was implemented using an external SRAM device on the computer's main board, increasing processor speeds forced L1 and L2 cache memories to be included on the processor die[5]. In addition, smaller feature sizes today easily allow the integration of cache sizes up some few Megabyte on the CPU die and can reduce system fabrication costs as well as the system size (as required by handheld equipment).

Most processors provide two separate L1 caches one for instructions and one for data items. This prevents eviction of needed instructions from the cache when large amounts of data are transferred by the CPU.

By contrast, in most systems the level 2 cache does not distinguish between instructions and data.

L1 and L2 cache operation is transparent to the user. Dedicated logic handles all transactions from the CPU to the memory controller.

In order to simplify cache design and to exploit spatial locality, the cache memory is managed on a cache line granularity. This means that the cache is divided into multiple equally sized blocks called cache lines which hold the requested data items and all data members in the direct neighborhood. Therefore, only complete cache lines are transferred from the CPU to the memory controller and vice versa[6].

If the CPU reads data, a lookup in the cache has to take place to check whether the requested memory location is in the cache or if it has to be reloaded from memory. In order to reduce the search time, the number of cache locations to which a specific address range in primary memory can be mapped has to be reduced. This number is called cache associativity[7].

Many publications have been released which investigated optimal cache replacement policies, the impact of variations in cache size, cache line size, or associativity on system performance, or which make proposals regarding cache aware design of software algorithms (e.g. [1, 43, 37, 35, 5, 24]).

[5]Chip internal interconnects are faster than external connections.

[6] During write operations, the cache controller could shorten the transfer size in order to write only data items which have been modified to the memory.

[7]Current processors provide a cache associativity of 8 to 16 [4].

Virtual Memory / Paging

Virtual memory denotes the possibility to address more primary memory than is installed physically in the computer system, by moving infrequently used data to secondary storage (*swapping*) and reloading it when it is used by the CPU. Paging divides the physical memory into equally sized pages[8]. The *memory management unit* (MMU) which is part of the CPU core maps these pages to the CPUs virtual address space. If the CPU addresses memory locations which are not mapped to physical memory, the MMU signals a trap condition to the CPU. The CPU interrupts the running process and executes an operating system procedure which is responsible for acquiring a new physical memory page (e.g. by saving the content of an infrequently used page on disk) and mapping it to the requested memory location [46, 7]. The table which is used to translate virtual addresses to physical addresses is stored in physical memory as well. In order to eliminate the need of looking up the translation rules for every memory access, the CPU memorizes the last references to memory locations in a *translation lookaside buffer* (TLB) [42].

Locality analysis of memory accesses has to consider that memory locations in the physical address space which are not located on the same memory page are not necessarily related to each other (e.g. belong to the same process).

The top section of Figure 2.12 shows how the MMU translates memory references to virtual memory to linear physical addresses.

2.4.2 Operation Modes of DDR-SDRAM

The Cell Array

Every SDRAM cell comprises a capacitor and a MOSFET as switching device (see figure 2.6). Information is stored by charging the capacitor[9]. All capacitor/MOSFET pairs are arranged in a matrix called SDRAM *bank*. The gate electrodes of all MOSFETs within one row are connected. This connection is called *wordline*. The sources of all MOSFETs within one column are also connected. This connection is called *bitline*.

Reading of memory contents is done by activating a single wordline. The mosfets build a conductive path between the capacitors belonging to that wordline and the bitlines so that the capacitors share their charge with the capacitance of the connected bitline[10].

[8] Typically 4 KiB to 16 MiB in modern CPUs / operating systems [46, p.383].

[9] A one can be represented by a fully charged capacitor, a zero may be represented by a discharged capacitor.

[10] In modern SDRAMs the storage capacitor is in the range of 20 to 40 fF, thus the load capacitance of the bitlines has to be taken into account which can be nearly an order of magnitude larger than the capacitance of the storage capacitor and coupling between adjacent bitlines has

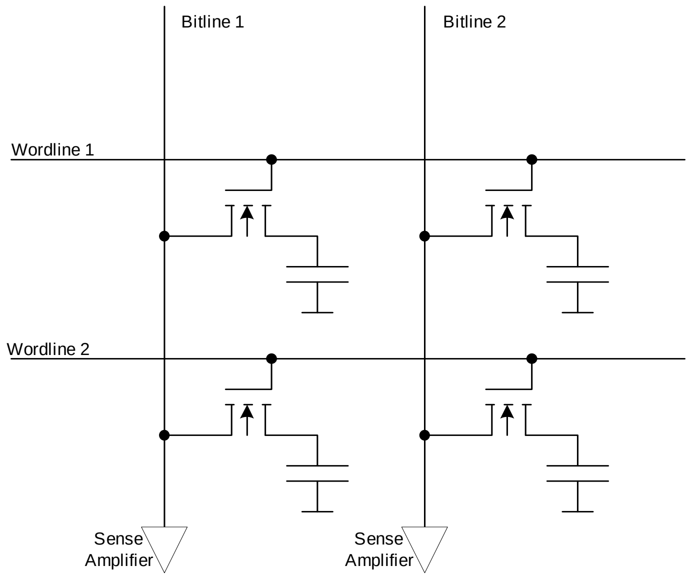


Figure 2.6: DRAM Cell Array

Depending on the previous voltage level of the capacitor, the bitline voltage increases or decreases. The cells information content is determined by sensing the voltages on the bitlines and comparing them to some mid reference voltage within the sense amplifier.

This is done for all bitlines of the SDRAM bank at the same time. The time required for activation and sensing determines the *RAS latency* or *RAS to CAS delay* (t_{RCD}).

The sense amplifier provides a positive feedback path to the cell array, refreshing the cells content while determining the voltage level on the bitline. The time required to recharge the SDRAM cells is called minimum bank activation time (t_{RAS}).

In a second step the read command selects some of the bitlines and provokes that their information is passed to the output pins. In order to enable the high throughput of modern SDRAMs, data is transferred in burst mode. This means that not only the requested data item is transferred but a complete burst of successive memory locations within the current row. DDR2 memory performs a by 4 prefetching. This means that each read command acquires the information of four consecutive addresses at the same time. The data is then time multiplexed to the output pins. Therefore, the minimum number of data items to be transferred is four (*minimum burst length*). Double data rate means that data is shifted out

to be considered as well [36].

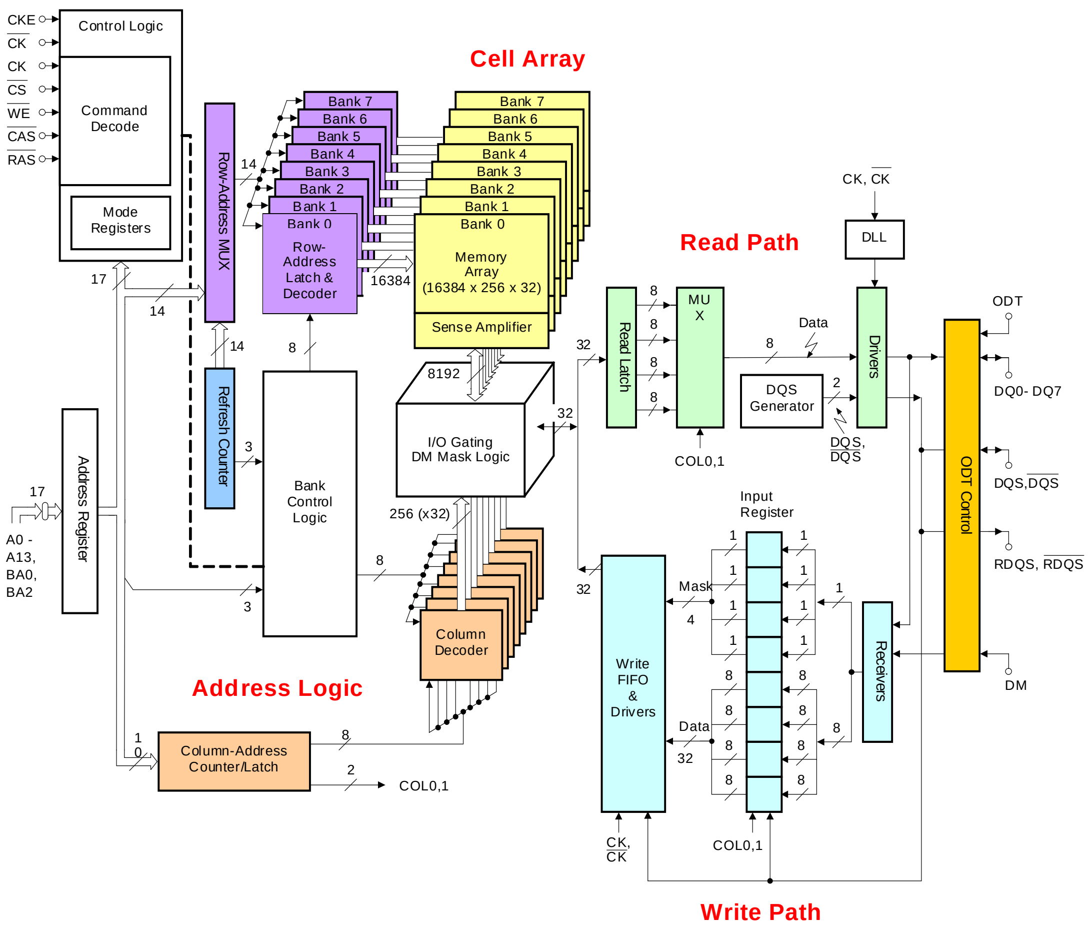


Figure 2.7: 1 Gibit DDR2 SDRAM
8 banks, 128 Mibit each, by 8 organization (16384 rows, 8192 columns) [29]

Name	**Clock Frequency**	**Data Rate/Pin**	**Core Frequency**	**Prefetch**
SDR-133	133,0 MHz	133 MBit/s	133,0 MHz	1
DDR1-266	133,0 MHz	266 MBit/s	133,0 MHz	2
DDR1-333	166,5 MHz	333 MBit/s	166,5 MHz	2
DDR1-400	200,0 MHz	400 MBit/s	200,0 MHz	2
DDR2-400	200,0 MHz	400 MBit/s	100,00 MHz	4
DDR2-533	266,5 MHz	533 MBit/s	133,25 MHz	4
DDR2-667	333,5 MHz	667 MBit/s	166,75 MHz	4
DDR2-800	400,0 MHz	800 MBit/s	200,00 MHz	4
DDR3-800	400,0 MHz	800 MBit/s	100,000 MHz	8
DDR3-1066	533,0 MHz	1066 MBit/s	133,250 MHz	8
DDR3-1333	666,5 MHz	1333 MBit/s	166,625 MHz	8
DDR3-1600	800,0 MHz	1600 MBit/s	200,000 MHz	8

Table 2.1: SDR / DDR SDRAM Data Rates and Frequencies
[11, 33]

synchronously with every rising and falling edge of the external clock line.

The SDRAM component shown in figure 2.7 provides a by 8 organization. This means the component has eight data lines each delivering a single bit at a time. Thus, the number of bits acquired during each read command is 32 ($= 8 \times 4$).

The prefetching concept makes it possible to operate the SDRAM core four times slower than the bus interface. Table 2.4.2 gives an overview of SDRAM pin bandwidth, operating frequencies, internal core frequencies, and the number of prefetched bits. As one can see, improvements in pin bandwidth over DDR1 and even single data rate SDRAM result mostly from an increase in the number of bits which are prefetched, i.e. from an increase in parallelism within the SDRAM.

The time required from issuing the read or write command to the delivery of the first data item is called *CAS latency* (CL).

Before another row can be activated, the wordline has to be deasserted and the bitlines have to be biased to a mid voltage level. This process is called *precharging* and requires some more clock cycles[11] (t_{RP}) until the new bank activation command can be issued. Figure 2.8 shows a typical activate-read-precharge cycle.

This leads to the situation that, depending on the current SDRAM state, three different "types" of memory accesses can occur:

If a memory access goes to a row which is already active, the read/write command can be issued directly. If the bank is not active, a new activation command has to be issued first. If the bank is active, but the wrong wordline is selected, the

[11]Currently around 15 ns.

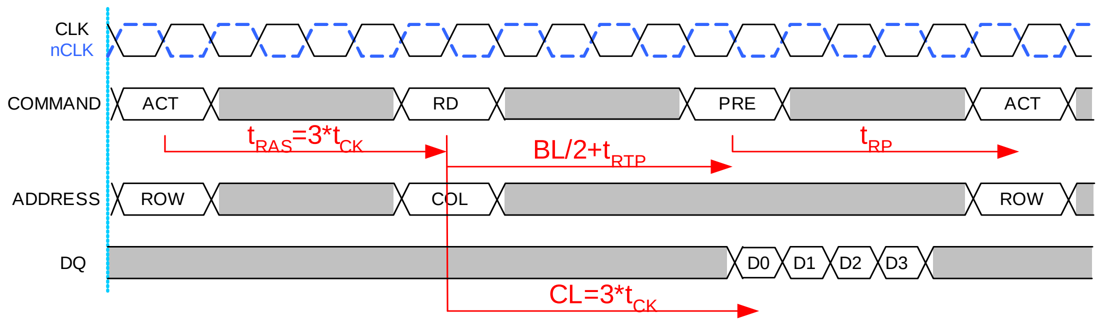


Figure 2.8: Typical Read Sequence (worst case)
CAS latency=3 cycles, RAS latency=3 cycles, burst length=4

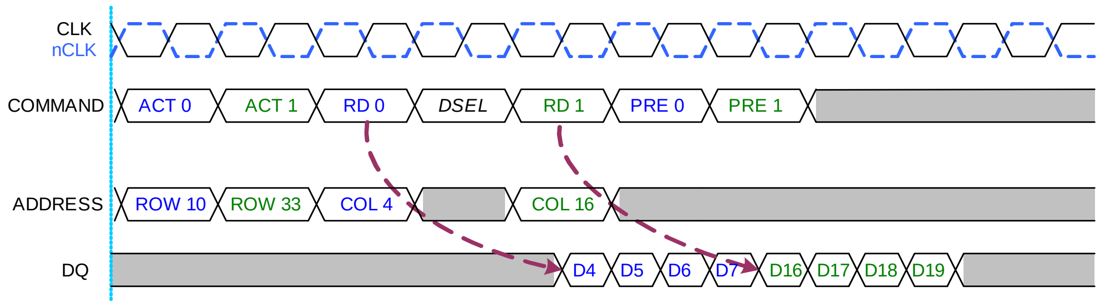


Figure 2.9: Interleaved Read Access to different Banks
CAS latency=2 cycles, RAS latency=2 cycles, burst length=4

bank has to be precharged, the new row has to be activated, and the read/write command can be issued afterwards.

Thus, modern SDRAMs expose some cache functionality: Accesses going to a row which is already open (activated) can be served fast while accesses going to different rows require more time for activating the new row and probably for precharging the old one. In some systems where space, power, or cost are a concern (e.g. PDAs) modern SDRAMs may replace the level 2 cache.

Current DDR2 SDRAMs contain multiple banks (usually 4 to 8), sharing the same address and databus (*CA-bus*). From the system developers point of view they can be regarded as multiple chips within one single package. Having multiple SDRAM banks within one chip and the clock synchronous SDRAM interface provide the possibility to perform multiple concurrent commands on different banks at the same time (as long as there is no conflict using shared resources) in order to hide SDRAM access latencies.

Figure 2.9 provides an example of an interleaved read access sequence to two different banks.

As the storage capacitors discharge over time due to leakage currents, the SDRAM content has to be refreshed periodically (in DDR2 every cell has to be refreshed every 64 ms) by activating the corresponding row. To simplify memory controller design, this can be done by issuing a *refresh command* (traditionally called CBR). A row address counter included in the SDRAM logic ensures that each *refresh command* refreshes a succeeding SDRAM row. The SDRAM refresh takes place on the same row of all SDRAM banks within the SDRAM component. It is mandatory that all banks are in a precharged state before a refresh command may be issued.

An additional power down operation mode is implemented called *self refresh*, which is used to retain the DRAMs content while the system is powered down. The *self refresh entry* command shuts down internal chip functions to reduce power consumption and ensures that the memory content is refreshed periodically using internal timers.

Figure 2.10 provides an overview of all DDR2 DRAM chips states and transitions between states. In addition to the commands described above, read and write commands can be combined with the precharge command so that the row is closed automatically after the data burst has been transmitted (Read_AP, Write_AP in figure 2.10). This simplifies memory controller design as no separate precharge command has to be issued and keeps the CA-bus unused for an additional clock cycle which would normally have been used for issuing the precharge command. Furthermore, when using auto precharging, the SDRAM's internal circuitry ensures that the SDRAM timings (t_{RAS} and t_{RTP}) are fulfilled before executing the precharge operation internally.

DRAM Modules

In order to increase the data bus width, in many applications (e.g. PCs and workstations) multiple SDRAM components are soldered on a printed circuit board (called SDRAM module). Sharing the same command and address bus, they are equally addressed at the same time and each component connects to a subset of system data bus lanes. This topology is called a *DRAM rank*.

Multiple SDRAM ranks share the same CA-bus and data bus. The memory controller addresses the ranks (and distinguishes between different ranks) by asserting chip select lines which are unique for every rank.

Modern computer systems may even include multiple CA and data busses called *memory channels*. Multiple channels do not share any resources and can be operated independently in parallel or are operated equally in order to simply increase the memory bus width (*lock step operation*).

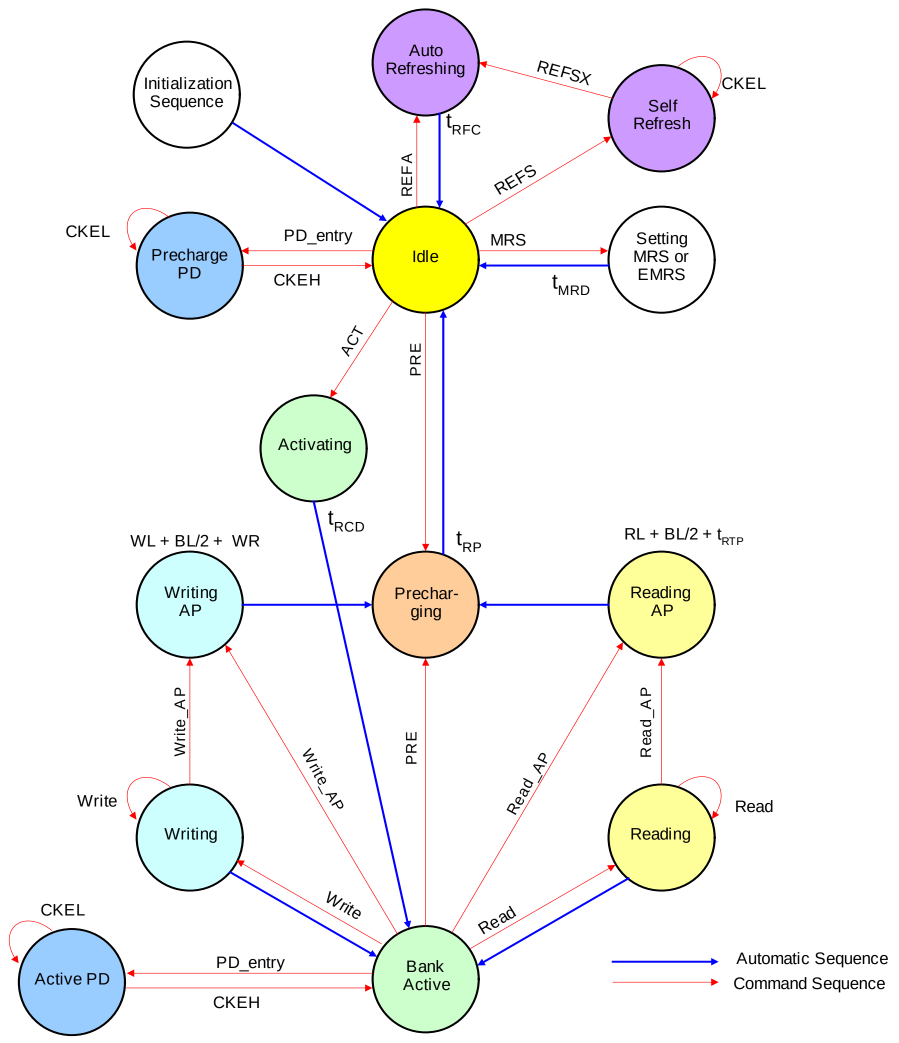


Figure 2.10: DDR2 Finite State Machine
[30]

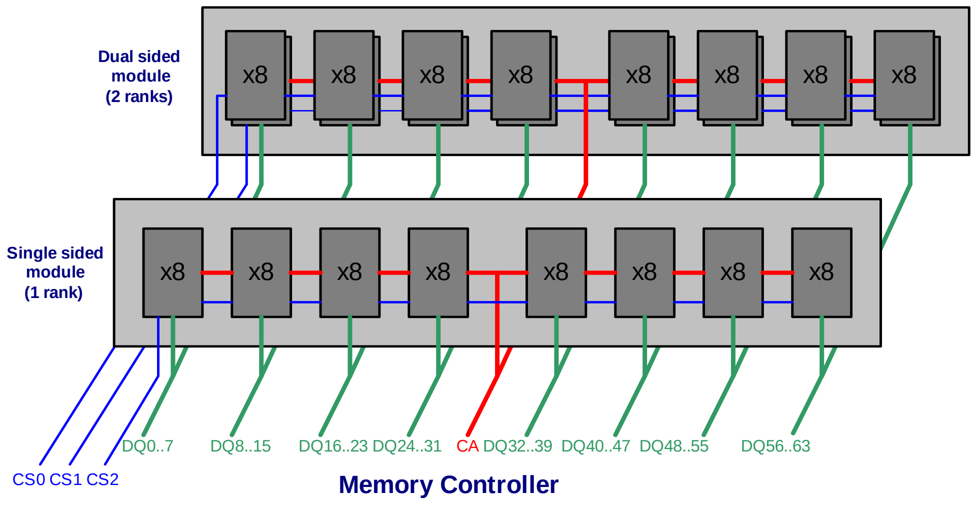


Figure 2.11: DRAM Modules, Ranks, Channels

2.4.3 Options for SDRAM Controllers

As seen in chapter 2.4.2 accessing SDRAM contents requires issuing a sequence of multiple commands. Additionally, multiple bank and rank topologies allow the execution of multiple concurrent memory requests at the same time. This leads to a large degree of freedom regarding the development of memory controllers and widens the design space for computer system developers.

Address Translation

The most obvious task for the memory controller is the address translation from a physical address to a ”SDRAM compliant” addressing scheme by means of ranks, banks, rows and columns. Usually this is done by assigning specific bits of the linear physical address to the address components of the SDRAM (see figure 2.12 for an example of the address translation accomplished by the Intel 845 chipsets north bridge). As the address translation determines how consecutive addresses in the physical address space are mapped to the installed SDRAM memory devices, banks, and rows, it has significant influence on the row hit and miss rate of the memory system.

Cache Line Splitting

In systems using caching, only complete cache lines are exchanged between SDRAM memory and the cache. As the burst length does not necessarily have to match with the cache line size, splitting the cache line transfer into multiple

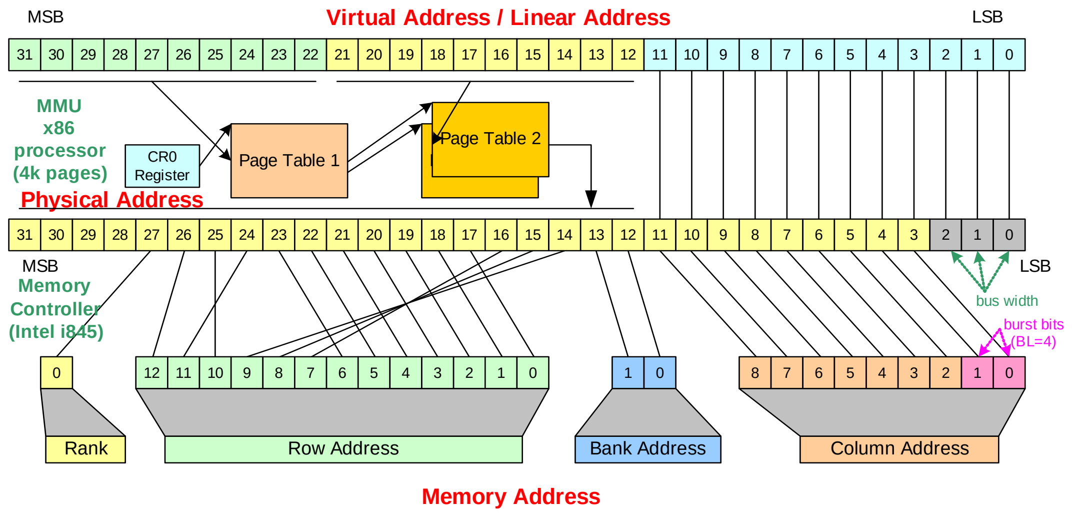


Figure 2.12: Address Translation from virtual Addresses to DRAM Addresses

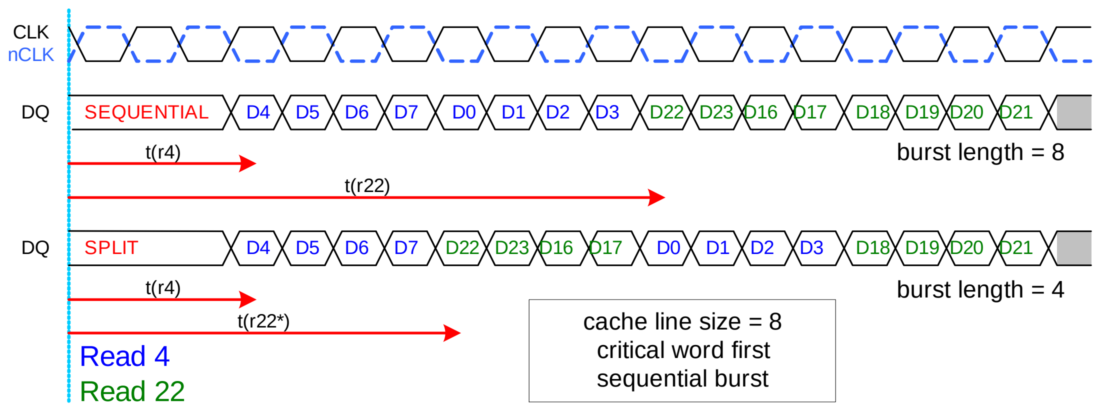


Figure 2.13: Latency Reduction by Transaction Splitting

shorter SDRAM bursts may reduce the average SDRAM latency. The arrows in Figure 2.13 show the latency of two SDRAM read requests arriving at the same time at the SDRAM controller. In the top half of the diagram the SDRAM requests are executed with a burst length of eight, which is equal to the cache line size. In the bottom half the same two requests are shown as split transaction with a burst length size of four. In both cases the critical word is transferred first.

As nothing comes for free, issuing more read or write commands with shorter burst lengths occupies the command and address bus more frequently and may stall other pending requests on other SDRAM banks.

Furthermore, the memory controller can select whether to pass the cache line content in order or to pass critical data (the one which was originally requested by the CPU) first.

Temporal Access Ordering

As the time required to access DRAM memory depends whether the corresponding row is already open or closed, it may be advisable to collect and merge multiple accesses hitting the same row and schedule them appropriately. Additionally, transactions can be priortized, e.g. read transactions have high priority as the processor core has to wait for required data while write commands may be delayed [44].

Precharge Policy

The memory controller can select between an open page and a close page policy. Open page policy means that the memory row within one bank is kept open as long as possible. The idea is to exploit temporal and spatial locality as in most applications there is a significant probability that consecutive accesses will go to the same row again, making it unnecessary to issue a new activate command. The trade-off is that, if the next request addresses another row, the bank has to be precharged first, which takes additional time until the new row may be activated.

The close page policy automatically closes every bank after one transfer. This may reduce latency in applications where consecutive requests mostly go to different rows.

In addition, memory controllers may dynamically select an appropriate precharge policy by performing estimations whether they expect future references to the same row or not (dynamic page policy). The integration of memory controllers within the CPU core simplifies this task, as the memory controller may peek at the processors state (e.g. register contents, processor pipelines, cache) to support the estimation.

Power Awareness

In some applications (e.g. mobile devices) power consumption is a concern. The memory controller may put the memory devices in a power down state in order to reduce their power consumption. In return the memory access latency increases, if the device has to be waken up from power down to fulfill a memory request. It is the memory controllers responsibility to detect phases of low SDRAM utilization and to schedule power down phases accordingly.

Refresh Policy

SDRAM rows which are not accessed have to be refreshed from time to time. During these refresh periods no further requests can be handled by the SDRAM component. Therefore, it is advisable to schedule SDRAM refreshing cycles to times when there are no memory requests pending. Estimating points in times, where SDRAM components usage will be low will be one of the goals for future memory controller designers.

Prefetching

Similarly to processors performing speculative execution, the memory controller may also try to estimate future memory accesses and perform prefetching. This mechanism may reduce latency, as data items are moved to the cache before they are referenced by the processor. Unfortunately, it also increases bus bandwidth requirements as not all prefetched data items will be used. In fact wrong guesses may even decrease system performance [12]:

- High priority requests may have to be stalled in order to complete an already running prefetching operation.
- Prefetching may crowd out vital cache line from the cache, which will be referenced soon and therefore have to be reloaded again (while the prefetched data may not be used at all).
- Prefetched data may be prefetched too early so that it may be evicted from the cache before it has been used.

Scatter Gather Operations

Some applications have inefficient access patterns, e.g. algorithms which access every n-th element within a data array[12]. Accessing only few elements of a cache

[12] One example is the vertical filtering of images (e.g. for deinterlacing of video frames).

line wastes memory bus bandwidth as most data items are not referenced. In addition, cache performance suffers as the cache gets filled with useless data.

Future memory controllers may map the memory content to the physical address space multiple times providing a different view of the memory array [59, 14]. Read accesses to this shadow address space will trigger the memory controller to assemble a cache line which may consist of data items located on different rows, banks or devices (gather operation). Similarly write accesses are distributed along the different DRAM memory locations (scatter operation).

This concept reduces bus bandwidth requirements as only used data is transferred and improves cache utilization.

Chapter 3

Evaluation of Memory Access Sequences

In order to study the behavior of a computer system, some software is executed on the device under test. At best the software should be selected for system evaluation, which will later run on the final target platform. Nevertheless, this approach is often not feasible:

1. The software is not yet available during the development phase of the target platform.
2. The hardware requirements of the software may be too high in order to be fully used in a simulator (e.g. large database applications).
3. The type of used application is not clear during the development phase (e.g. target applications of a personal computer may be text processing, computer gaming or numerical scientific computation all having different requirements on the hardware).

Therefore, studies are typically conducted with software which shall resemble the target application [34]. This software is called *workload* or *benchmark*. Many publications have been written on finding appropriate workloads for different applications and ways to determine metrics for the similarity and dissimilarity of different workloads and applications (e.g. [48, 38, 10, 15, 19, 51]). Nevertheless, even today determining appropriate workloads for system evaluation comes close to black magic and is often a source for dispute between different researchers.

Two types of benchmarks can be distinguished:

- The benchmark has a fixed task to be fulfilled (e.g. execution of an algorithm of a scientific application). A typical performance metric is the total benchmarks runtime.

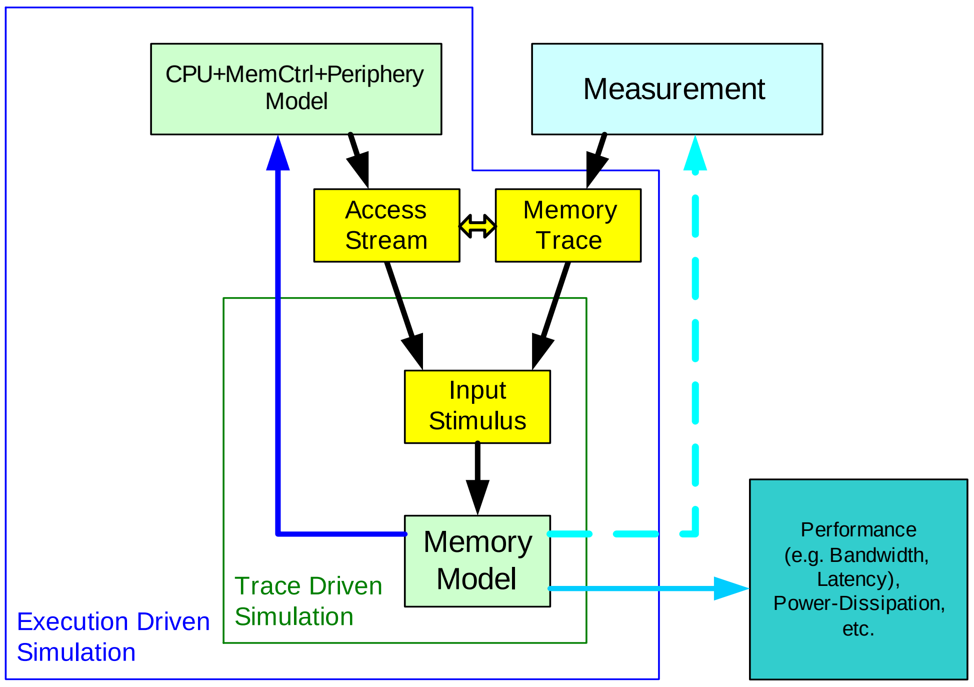

CPU+MemCtrl+Periphery Model
Measurement
Access Stream
Memory Trace
Input Stimulus
Memory Model
Trace Driven Simulation
Execution Driven Simulation
Performance (e.g. Bandwidth, Latency), Power-Dissipation, etc.

Figure 3.1: Trace driven versus execution driven Simulation

- The workload is variable. This type of workload can be found mostly in computer games. While the execution time of the workload is fixed, the system tries to perform as much computation of intermediate results as possible (e.g. in the case of a computer game the system renders as many frames on the computers screen as possible during the given time). A typical performance metric for this class of application is the number of rendered frames per unit time.

Traditionally, three techniques have been applied to study computer system behavior regarding the transactions between CPU, caches and primary memory: *execution driven simulation*, *trace driven simulation* and the *measurement of access sequences* (see figure 3.1).

3.1 Execution driven Simulation

Execution driven simulation tries to model the simulated computer systems components (CPU, cache, memory controller, buses, etc.) in software on a host systems. The applications binary is then run on the simulated target. Several simulation environments have been implemented during the last years and have been used for research purposes: The SimpleScalar Toolset [6], SimOS [45], DRAMsim[1] [53], Rascas [2], POPeye [58], or Virtutech Simics [40] just to name some few.

The main advantage of execution driven simulation is that the state of all components is accessible to the engineer as the components are only software modules running on the host computer.

In addition, it is also possible to simulate hardware which is not yet existing or to run simulations which are based on unrealistic assumptions (e.g. unlimited bus width or no memory latency) to obtain critical values regarding system behavior. This makes execution driven simulation extremely useful during the design phase of new computer systems not only for performance estimation but also for functional verification of the components.

Unfortunately, as modern computer systems complexity increases, more and more hardware features have to be rebuilt in software. For the CPU side this includes the emulation of larger instruction sets, prefetching, pipelining, speculative execution, etc. . Processor manufacturers sell different processor variants for different market segments (server, workstations, notebooks) and the processor architecture/instruction set of different manufacturers may be completely different (e.g. x86-based, ARM, PowerPC, MIPS, Coldfire, 68k) extensively increasing the implementation effort for the simulator developer when different computer systems shall be compared.

The same holds for other parts of the computer system like the memory controller or peripheral components.

Additionally, the exact functionality of the components belongs to the intellectual property of the respective manufacturer and is usually not made public. So timing accurate modeling of these components becomes difficult if not even impossible.

Thus, severe simplifications of the computer system under investigation have to be imposed. Many simulators emulate only the processor and memory controller system and are therefore restricted to the execution of user mode applications, which do not rely on peripheral hardware and on the operating system which usually handles accesses to these peripheral components. Operating system calls exercised by the application are intercepted by the simulator and replaced by a functional equivalent routine on the host system. The result is returned to the application. Thus the application can use typical operating system calls (e.g. in order to communicate with the user), although these calls do not affect the simulation results.

More elaborate simulators try to emulate peripheral components like hard disk drives and graphics cards as well. The improved hardware support enables system designers to run the desired operating system on the simulated target platform,

[1]DRAM model for other simulators like Sim-alpha [17] or GEMS [41].

enabling the simulation of operating system influences on the systems behavior. Recent studies [13] found out that omitting accesses from the OS may lead to an error in the cache miss rate of up to 100 percent even in simulations which do not use operating system calls excessively.

Unfortunately, emulation also means that tasks, which are usually handled by fast hardware on the target platform, have to be simulated by executing multiple instructions on the host platform, making execution driven simulation extremely slow. Depending on the simulations level of detail, typical factors are 10 to 10000 compared to the original system.

Therefore, system engineers have to bear programs execution time in mind and can run only short program sequences on the emulated hardware. Variations of simulation parameters (e.g. changes of the virtual computer hardware) require that the complete simulation has to be rerun.

3.2 Trace driven Simulation

Trace driven simulation tries to eliminate the need to simulate the entire target platform by acquiring a complete memory trace from the frontside bus. This can be done by measuring access sequences on a real computer system, or by logging all memory requests during an execution driven simulation. The logged memory accesses are then exercised on the simulation model of the memory system under investigation. In [52] Uhlig provides an overview of various software and hardware based approaches for acquiring and processing of memory traces.

This approach decouples the simulation of the CPU from the simulation of the memory system under investigation. It may reduce simulation times, as the CPU model has to be run only once and the gathered access sequence may be used as input stimulus for multiple memory models. Unfortunately, the memory system and the remaining computer parts are mutual interdependent. This means that parameter changes in either of them (e.g. changes in CPU frequency or the number of DRAM banks) render the memory trace invalid. Thus, some workarounds have to be applied in order to estimate the effect of memory system changes on the resulting access sequence. Chapter 5.2 tries to estimate the effect of SDRAM system changes on the SDRAM access sequence and thus the memory systems performance.

The significant interdependence of CPU and memory subsystem shall be clarified by looking at asynchronous events. Modern computer systems consist of multiple timing domains which run asynchronously to the CPU clock (e.g. the motor of the hard drive or the frame drawing of the graphics card) or are heavily influenced by user activity (movement of the mouse, reception of network packages), which determine the execution order of program sequences.

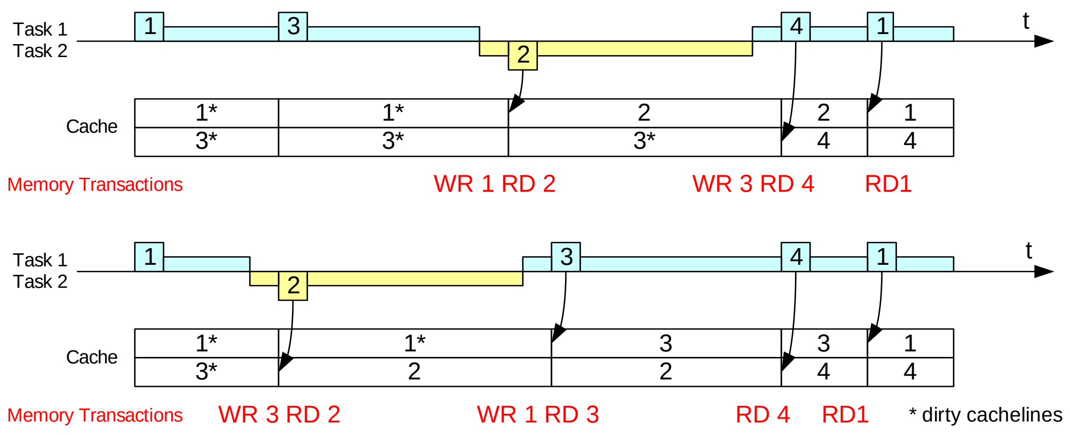


Figure 3.2: Asynchronous Timing
(Dirty cachelines are marked with an asterisk)

Even worse, the memory systems performance determines not only how fast accesses are performed and how long the CPU is stalled due to memory latency, but also what accesses are performed.

Figure 3.2 shall clarify this issue. We assume a CPU using a very small fully associative cache of two cache lines. The replacement policy is *least recently used*, so that the cache line which has not been referenced for the longer period of time is replaced. We assume that the CPUs execution context is switched due to an external event (e.g. an interrupt due to user interaction). Figure 3.2 shows the caches content at each point in time, the referenced memory locations (read request), and the bus transactions issued due to the replacement of cache lines. Depending on the point in time when the first task is interrupted, the transaction sequence is completely different, although the processor executed the same two instruction sequences and although in the end the cache contents are completely equal in both versions.

Literature studies [22, 23] found out that context switches due to exception and interrupt execution account for a significant rise of the cache miss rate, leading to an increase in data transfers from main memory to the cache and vice versa.

While one could argue that context switches may be infrequently enough so that they do not influence access sequence statistics seriously, the situation becomes even more difficult with CPUs performing speculative execution. In this case the memory systems speed determines the number of execution paths the CPU core is able to follow while DRAM memory content is acquired. The different execution paths may touch different cache locations and therefore determine the cache lines which will be replaced next.

Although trace driven simulations which do not care about this fact have to be regarded suspiciously, trace driven simulation is used frequently in the research community where execution driven simulation is not applicable due to performance or complexity limitations.

But the missing feedback path also provides some benefits for the system developer. It guarantees that the input stimulus to the memory is always the same and thus eliminates the mutual interdependence of CPU and memory system. This simplifies the evaluation of different memory systems significantly, as the memory system does not affect its own input stimulus and allows the system designer to distinguish between effects which are caused by changes of the memory system and effects which are imposed by the CPU/memory interdependence [54, pp.137].

Literature studies also tried to evaluate the possibility to build artificial access patterns by combining short access sequences acquired by execution driven simulations or measurements in order to reproduce the behavior of multitasking environments or the impact of operating system calls while simplifying the design of the simulator [22].

3.3 Measurement

The most evident approach to determine system behavior is the direct measurement of memory accesses on the real hardware platform. The results obtained from these measurements are accurate, as no simplified hardware models have to be used. Additionally, one can generate memory traces from even most demanding software on highly complex multiprocessor computer systems on the fly.

Unfortunately, also the direct measurement of access sequences has serious drawbacks:

Firstly, the hardware platform must exist and changes in the hardware platform are limited to available and compatible hardware combinations.

Secondly, data acquisition by measuring provides only a limited view on the computer system at the point where signals are probed. Correlation of the acquired data with other components states (e.g. within the CPU or cache) are impossible unless accurate software models exist for the component and the complete stimulus has been recorded.

Thirdly, the repeatability of experiments is difficult to achieve. While it is very easy to save the starting state of the computer system in a simulator, the components of the real hardware platform cannot be stopped and probed to save the state of all components (CPU, cache, etc.). The typical solution to that problem is to use long access sequences, so that the startup phase of the benchmark is not statistically relevant.

Fourthly, the effort building appropriate acquisition hardware is extremely

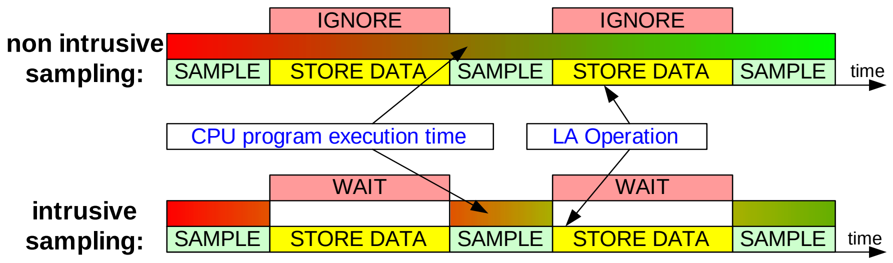


Figure 3.3: Traditional Measurement of Memory Access Sequences

high. Acquiring memory access sequences requires recording the CA-bus content (and in some applications the data bus as well) on other secondary storage types like hard disks in realtime, which provide significantly lower bandwidth than the CA-bus, is an extremely demanding task. Currently DDR2-800 memory controllers may exercise up to $400 \cdot 10^6$ commands per second on the DRAM component. Handling signals in this frequency domain does not only imposes a high complexity regarding signal integrity and processing, but also requires large storage capacities in order to sample longer periods of time.

Traditional measurement equipment like logic analyzers provide only very limited storage for sampling CA bus sequences. Current products available on the market provide a memory depth of up to 64 Megasamples [50]. The interface to store the sample buffer data to secondary storage usually provides very limited bandwidth and is therefore not suitable to transfer large amounts of memory trace data.

Most hardware probe solutions used in academia (e.g. [56, 55]) require to stop the tested target system periodically in order to be able to move the acquired data to secondary storage (intrusive sampling). Other solutions keep the system running and acquire only short sequences periodically, unloading the collected data while no acquisition is performed [52] (non intrusive sampling), and thus create discontinuities in the trace sequence (see figure 3.3).

Therefore, a high performance data acquisition system was implemented, which provides enough bandwidth to collect complete very large DRAM access sequences (of at least 960 Gigasamples) without altering the system behavior.

Unfortunately, the fact that data acquisition can be done on the fly does not necessarily mean that processing and evaluation of the acquired data afterwards can be done on the fly also. Therefore, a PC cluster is used, not only to store the acquired memory traces, but also for processing the acquired data, taking advantage of the computational power of the PC cluster.

Chapter 4

Measurement Hardware

4.1 Overview

The trace acquisition hardware mainly consists of three parts: the probe head, the FPGA platform, and the PC backend.

4.1.1 Probe Head

The SDRAM probe head consists of a six layer printed circuit board, which is placed in between the SDRAM module under test and the SDRAM component (see figure 4.2). Double ended pogo-pins, which are mounted in the PCB, form the contact between the SDRAM modules solder pads and the desoldered SDRAM component. All SDRAM signals are sent to high speed comparators (Micrel SY55857L) on the probe heads PCB, which compare them against the SDRAM's reference voltage V_{REF} and convert them to LVPECL levels, which can be transmitted over multiple centimeters of twisted pair cables to the FPGA board even at frequencies of multiple hundred Megahertz. All SDRAM lanes are length matched to minimize the skew of the CA-bus and data bus signals on the way to the FPGA.

4.1.2 FPGA Board

The FPGA platform basically consists of a ten layer printed circuit board with two high speed FPGAs (Xilinx XC2V1000), a 16 bit microcontroller with USB interface to download the FPGAs firmware and configuration data, a clock distribution network to propagate time shifted copies of the SDRAM clock signal to all FPGAs, and some power supply and monitoring logic to facilitate reference voltage generation and temperature sensing. Two additional eight layer PCBs are connected, containing an external SerDes chip, in order to provide 2 fibre optical channels of 10 Gigabit Ethernet per PCB.

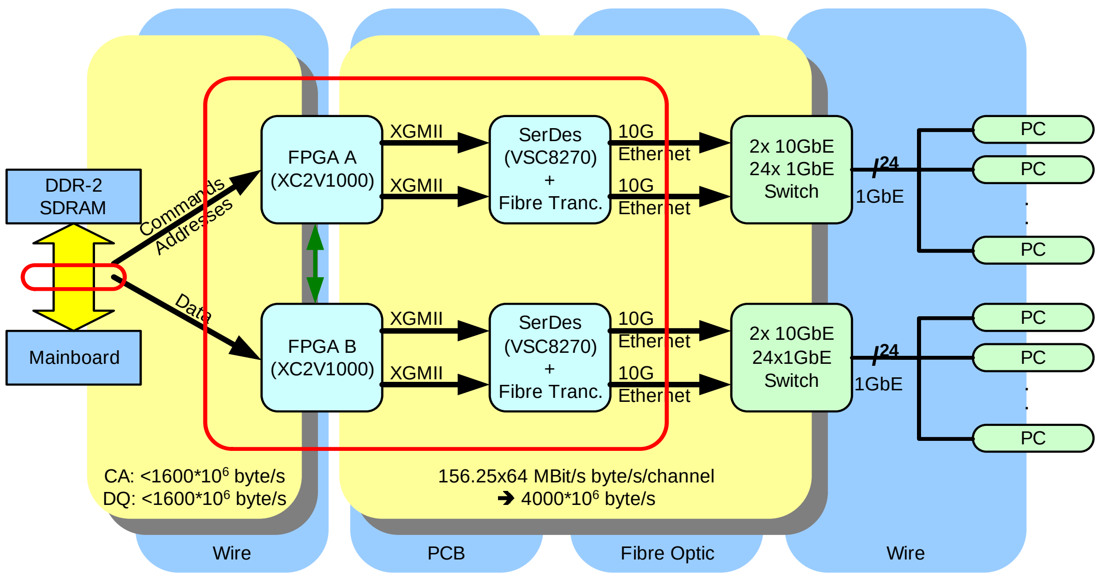


Figure 4.1: Trace Acquisition Hardware

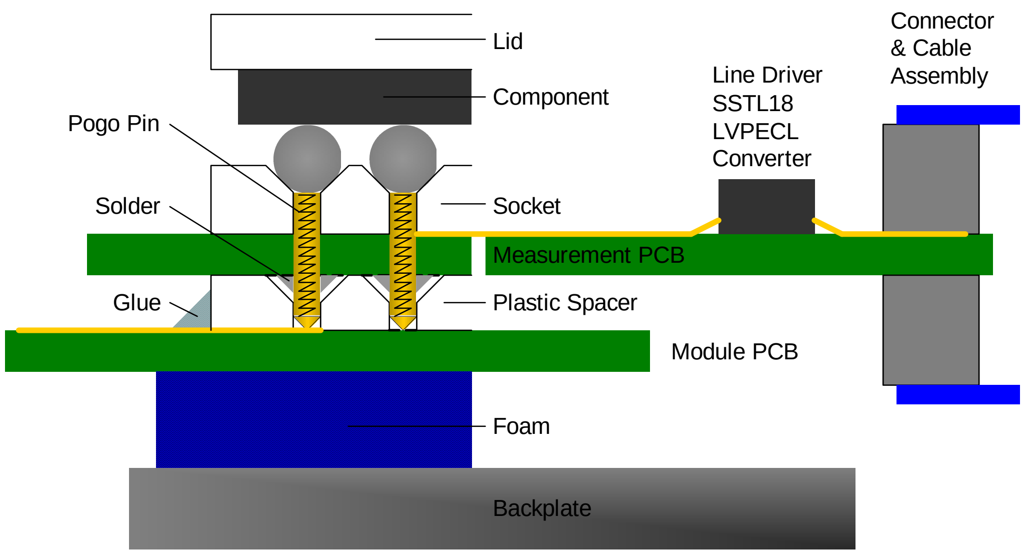


Figure 4.2: Probe Head

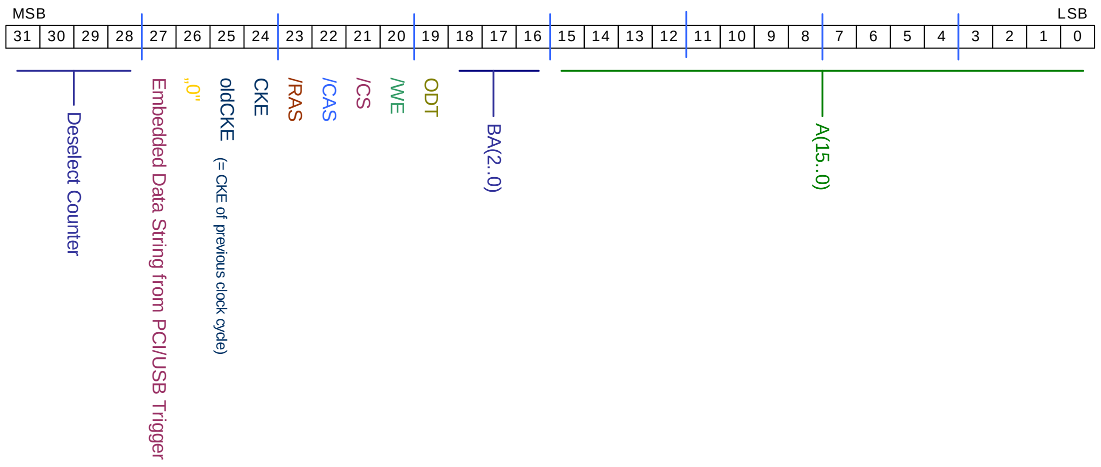


Figure 4.3: 32 bit Sample

FPGA A

The first FPGA (A in figure 4.1) latches the LVPECL signals of the SDRAM's command and address bus from the probe head. It counts the number of deselect commands between successive "useful" commands to provide information about the temporal spacing between commands, and may optionally remove the deselect commands from the captured data stream to provide a reduction of the required bandwidth for data transmission and storage.

The FPGA packages time slices of 256 samples of non deselect commands and stores them in SRAM memory within the FPGA. One single sample consists of 16 bit of address bus data, four bank addresses, the four signals of the command bus (RAS#, CAS#, CS#, WE#), ODT, CKE the value of CKE of the previous clock cycle[1], and the number of deselect cycles occurring before the command. This number is encoded in four bits. If a larger number of deselect commands occurred, deselect cycles are stored as samples within the trace file; so at least every 16^{th} SDRAM clock cycle has to be stored. Furthermore, a serial data stream consisting of six user definable bits can be embedded in the trace file. This 6 bit value can be set by the device under test via the external triggering hardware (see below) to facilitate the detection of different program execution phases. The sample vector is padded with zeros to a 32 bit value (see figure 4.3).

Ethernet, IP, and UDP headers are prepended to each time slice in order to build an Ethernet frame. These Ethernet frames are transmitted together with a CRC-32 checksum via one or two 32 bit $\times$ 156.25 MHz DDR interfaces called

[1]This simplifies analysis, as all commands can be decoded by looking at one single sample only.

	single channel		dual channel	
	raw-data	compressed	raw-data	compressed
DDR2-400	×	×	×	×
DDR2-533		×	×	×
DDR2-667		×	×	×

Table 4.1: Supported Capturing Modes depending on used Hardware Resources

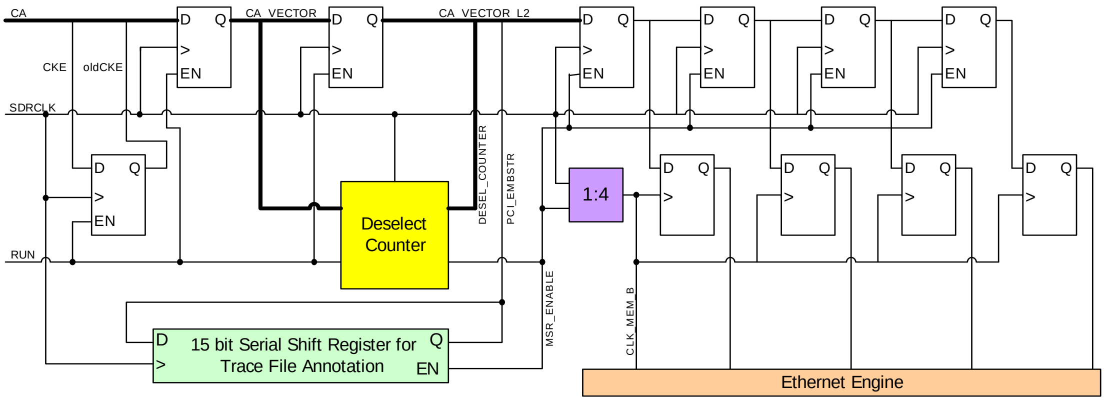


Figure 4.4: High Speed Sampling Frontend

XGMII to the external SerDes-Chip (Vitesse VSC8270). The SerDes is responsible for 8B/10B encoding, introduction of lane resynchronization symbols, and serialization of the data to 4 differential signal pairs, which are operated at 4 × 1.5625 GHz (3.125 Gbit/s). This serial interface is called XAUI.

The XAUI signals are fed into a fibre optical transceiver module, which sends them over a fibre optical link to the 10 Gigabit Ethernet uplink port of a 24 port 1 Gigabit Ethernet switch. Depending on the destination MAC address of the Ethernet frames, the Ethernet switch distributes the Ethernet frames to a cluster of PCs.

Starting with SDRAM frequencies of DDR2-533 (267 MHz), the required raw bandwidth of the CA-bus content becomes larger than the available bandwidth of one single 10GbE link, requiring the second 10GbE link. It turned out that the applied compression scheme (deselect removal) is sufficient to facilitate measurements of the CA-bus using only one single 10GbE port even at frequencies of DDR2-667 memory (see table 4.1).

Figure 4.4 shows a simplified block diagram of the high speed sampling logic implemented within FPGA A. As this part of the design has to be operated with the SDRAM's clock frequency of up to 400 MHz, it imposes the highest restrictions regarding the design complexity of the implemented logic. Furthermore, placement and routing of the logic within the high speed path becomes a demanding task in order to fulfill all timing requirements.

If the system is capturing data (RUN=1), the content of the entire CA bus (i.e. the 25 LSBs of the 32 bit sample shown in 4.3) is sampled with every rising clock edge. The CKE value of the previous clock cycle (oldCKE) is added to the vector. This ensures that power down entries end exits can be recognized during trace file analysis even when deselect removal is active. The first register stage is placed directly in the IO blocks within the FPGA. This guarantees that the routing delay is almost equal for all CA bus members (minimizing the skew between the CA bus signals).

The deselect counter evaluates the command bus signals (i.e. RAS#, CAS#, CS#, WE#) to detect deselect cycles which can be removed from the trace file. The counter appends the number of deselects to the four MSBs of the sample vector (DESEL_COUNTER). It also decides whether a sample is passed to the four stage shift register shown on the right side of figure 4.4. MSR_ENABLE is high when sampling is in progress (RUN=1) and deselect removal is disabled (i.e. all sample vectors are propagated to the shift register) or the command is not a deselect command or the deselect counter is 15 (i.e. the deselect is the sixteenth command in a continuous sequence of deselects). The annotation data is also added to the sample vector (bit 27) in the high speed domain.

Whenever four sample vectors have been propagated to the shift register, data is transferred to a set of latches. From there it is forwarded to the Ethernet engine. The Ethernet part of the design always handles four sample vectors at a time in a 128 bit vector. Demultiplexing the data stream by a factor of four reduces the clock frequency to at most 100 MHz (and even less if deselect removal is active), simplifying placement and routing of logic within the FPGA. Furthermore, demultiplexing is required in order not to exceed the maximum write rate of the FPGAs internal static RAM.

In an early design stage a set of counters was implemented to count the number of commands of every command type (activate, read, write, precharge, etc.). The counter which had to be enabled to count the particular clock cycle was determined by evaluating CA_VECTOR. The SDRAM analysis conducted in [3] used the results obtained by these counters. Although the counters were implemented using a carry-look-ahead strategy to facilitate counting at these high clock rates, they limited the performance of the FPGA acquisition hardware to frequencies of at most 267 MHz. Therefore, the counters were removed in a later design stage[2].

[2] If counting of commands is desired (e.g. for verification of recorded trace files), it is highly recommended to implement the counters in the "low speed domain" behind the 1 to 4 demultiplexer.

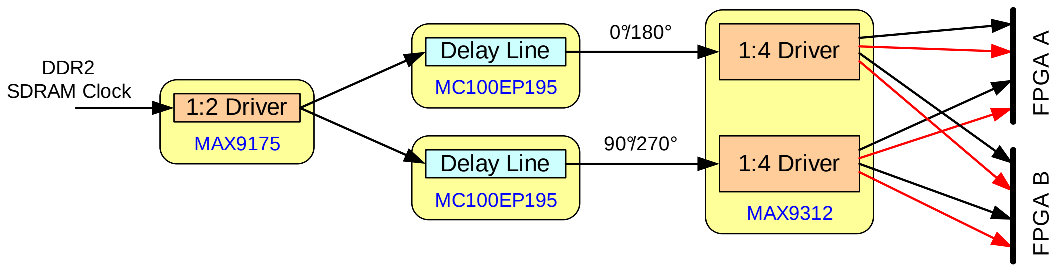


Figure 4.5: Clock Distribution Network

FPGA B

The second FPGA (B in figure 4.1) is intended to capture the data bus content of a SDRAM component with a data bus width of up to 16 bit. As there is currently no application for analyzing the transmitted data, the second FPGA was not programmed and equipped with a SerDes backend during the creation of this thesis.

Clock Distribution Network

In clock synchronous digital systems the incoming data (in this case the CA-bus and data bus information) has to be latched with the rising/falling edge of the global SDRAM clock. In this case a copy of the global clock signal has to be propagated to two FPGAs. Furthermore, phase shifted copies have to be generated to facilitate capturing of the data bus content.

Typically, designers of digital systems replicate the external clock and phase shifted copies of it via a phase locked loop (PLL) within the FPGA. This approach was not applicable here. The SDRAM specification allows entering power down states during which the memory controller does not necessarily has to provide external clocking. These power down states are rarely used in desktop systems but may occur in mobile applications (notebooks, PDAs). In periods during which no external clocking is provided the internal PLL may loose synchronization. Furthermore, the time required to lock in the PLL of the FPGA again exceeded the time allowed by the SDRAM specification.

Therefore, an external clock distribution network was implemented (see figure 4.5). On the FPGA board the SDRAM clock was replicated two times (Maxim MAX9175). The two copies are fed to two delay lines, which provide a user settable delay of the clock signals with a resoultion of 10 ps (OnSemiconductor MC100EP195) [47]. Each of the two phase shifted clock signals is then replicated four times (Maxim MAX9312). Each FPGA gets the original signal and an inverted

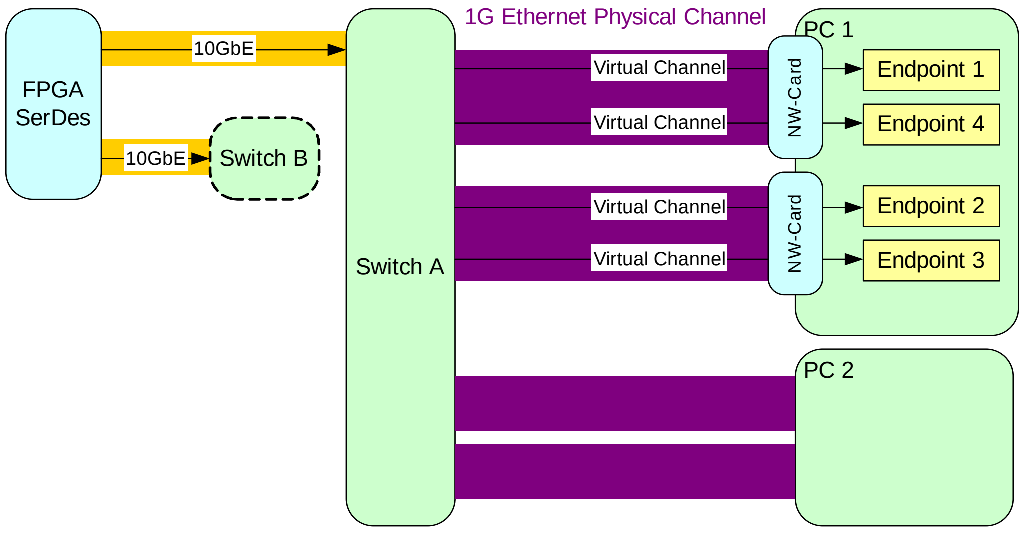


Figure 4.6: Virtual Endpoints

copy (as differential signaling is used, the inversion can be done by crossing the positive and negative signals of the differential pair) of the two delayed SDRAM clocks[3].

So each FPGA has in total four different SDRAM clock phases available. Depending on the SDRAM's speed grade under test, the delay line settings are adjusted manually by the 16 bit microcontroller.

Virtual Endpoints

For the distribution of the Ethernet frames to PCs an additional abstraction layer was introduced. All Ethernet frames are sent to "virtual endpoints" (see figure 4.6). A virtual endpoint is mainly the destination MAC and IP address of a network adapter receiving the Ethernet frames. All frames are sent over a "virtual channel" to this endpoint. These endpoints are enumerated. The FPGA maintains a list of virtual endpoints, which are served with successive Ethernet frames of trace data in a round robin fashion. Every PC of the PC cluster is responsible for storing the measured data of one or more virtual channels. This concept provides an abstraction of the data distribution from the underlying physical network hardware. By doing so, it is possible to handle multiple virtual channels by one single network adapter, providing some simple mechanism of traffic shaping and load balancing (the same network adapter can occur in the list multiple times).

[3] The inverted copies have to be generated externally, as an additional inverter in the clock path within the FPGA would introduce a significant delay (phase shift) to the inverted clock.

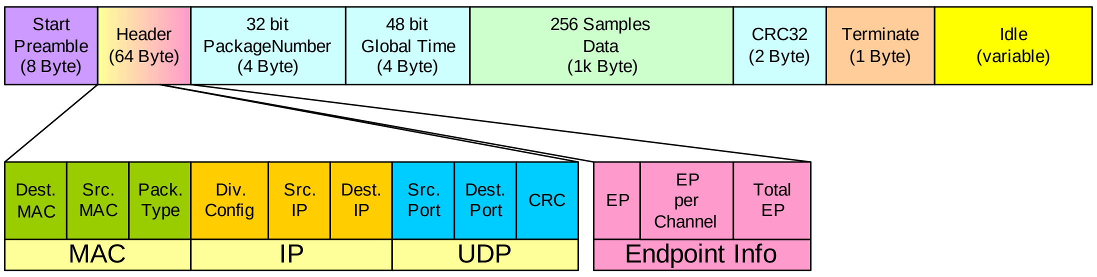


Figure 4.7: Ethernet Frame

Ethernet Framing

The sampled data is sent as UDP frames over the Ethernet network. In general UDP does not guarantee a reliable transmission over Ethernet. Nevertheless, the used Ethernet switch provides enough bandwidth for non blocking operation and, thus, no Ethernet frames are dropped in this small scale LAN application, neither by the switch, nor by the network adapters of the PC mainboards.

The complete Ethernet frame as shown in figure 4.7 starts with a MAC header containing the source and destination MAC address of the Ethernet frame. The Ethernet switch routes the frame depending on this data link layer MAC address to the desired network adapter. It is followed by an IP header containing the IP address of the FPGA platform as source and the IP address of the PC as destination. The UDP header specifies the destination port on the PC at which the capturing software is waiting for incoming Ethernet frames.

The UDP header is followed by the destination virtual channel/endpoint number, the number of virtual channels being transmitted over the respective 10G Ethernet interface, and the total number of virtual channels within the FPGAs virtual endpoint list.

A 48 bit free running clock counter operated at the Ethernet XGMII frequency (156.25 MHz) provides a timestamp, indicating the point in time, when the first sample of the Ethernet frame was captured by the FPGA.

During power down phases the DDR2 memory controller is allowed to shutdown the SDRAM clock. The missing SDRAM clock prevents the FPGA from capturing additional samples, which would allow the calculation of the total runtime. The Ethernet clock timestamps allow an estimation of the length of power down periods even when no SDRAM clock is available at the SDRAM component.

All frames sent on a 10G Ethernet interface are enumerated to allow the reordering of the Ethernet frames on the PC cluster.

The Ethernet frame is ended with the 256 samples of 32 bit trace data (1 KiB) and a CRC-32 checksum of the entire Ethernet frame.

4.1.3 PC Backend

The PC backend consists of a cluster of six PCs, which are connected to the 1G Ethernet switch over two 1G Ethernet links per PC. Special emphasize had to be put on a proper selection of the used PC mainboards, in order to make sure that both Gigabit Ethernet ports are connected to the north bridge via PCIe[4]. Each PC is equipped with four 160 GB hard disks, which are responsible for storing the incoming trace data (24 hard disk drives / 3.84 TB in total).

The PCs are operated using the Knoppix 4.0 Linux distribution as operating system. Knoppix usually runs directly from a bootable CD and, thus, does not need any write access to system directories. Writing to directories is enabled by holding modified or newly created files on a RAM disk within the PCs internal memory. This makes it possible to boot the same operating system on multiple PCs at the same time. Furthermore, Knoppix provides a sophisticated hardware autodetection mechanism, making it usable on a large variety of hardware configurations.

The Knoppix distribution is provided by a centralized server. The PCs are booted directly over the network via the PXE boot mechanism and get the linux distribution via NFS (*network file system*).

After bootup a program is run in the background, which listens on all UDP ports of the particular PC for incoming UDP frames from the FPGA platform and stores them on the hard disk drives in a round robin fashion. The hard disk drives are used without any partitions or file systems. This approach provides higher throughput as the system does not have to care about journaling or record keeping and, in addition, simplifies maintenance as one does not have to care about file naming conventions and proper partitioning.

In order to minimize the number of read and write commands being sent to the hard drives, every PC captures 1024 Ethernet frames before issuing a write/read command for all 1024 Ethernet frames (called "superblock") to the operating system (see figure 4.8). The write operation runs asynchronously to the capturing task with the Linux asynchronous IO (AIO) mechanism introduced in Linux kernel 2.6. Using AIO is required in order to ensure that multiple write operations to different hard disk drives can take place simultaneously (normal write operations would block the calling task until the write operation to the HD has finished). As capturing of Ethernet frames and writing to hard disk drives takes place in parallel, two sets of memory buffers have to be used (double buffering). While one set is written to hard disk drives, the memory of the other set is filled up with captured Ethernet frames (see figure 4.9).

The entire setup is capable to record around 900 MB/s of trace data on the fly

[4] Many mainboard manufacturers try to save money by connecting cheap Ethernet chipsets to the much slower PCI bus of the south bridge. Unlike the traditional PCI bus, only PCIe provides enough bandwidth to enable full loading of both Ethernet links.

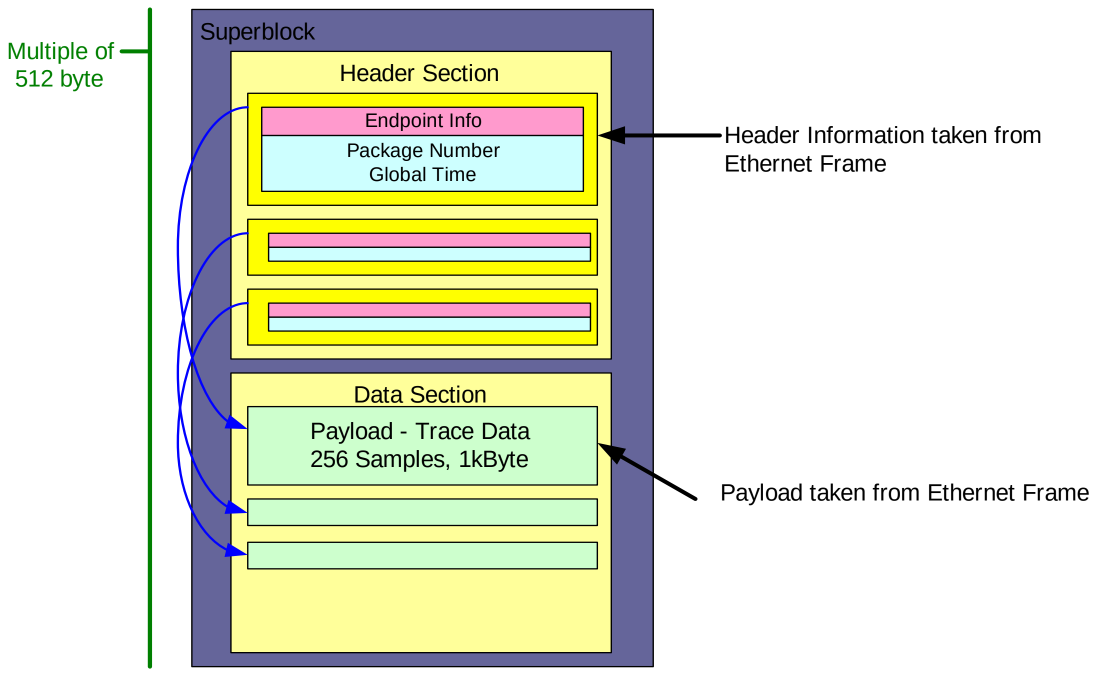


Figure 4.8: Ethernet Frames within Superblock

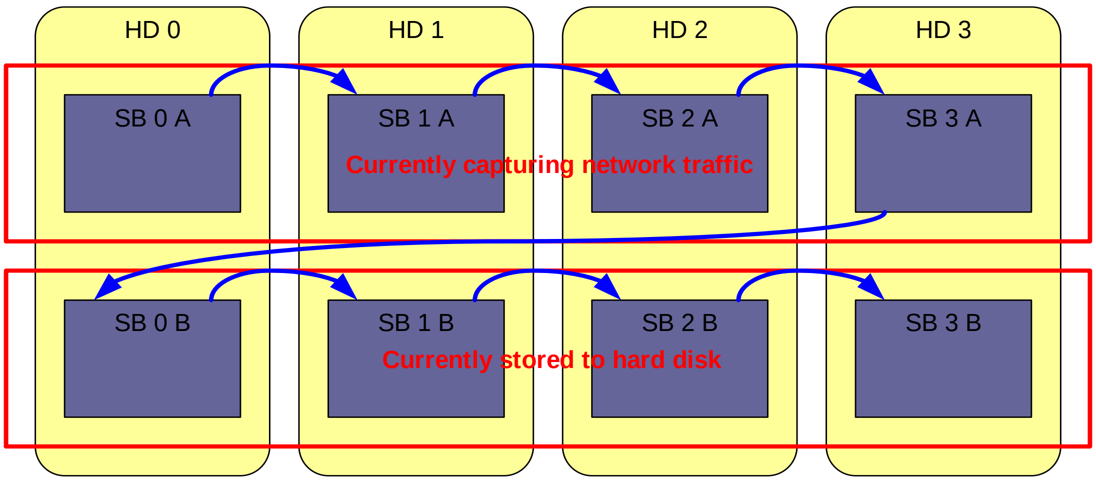

HD 0
HD 1
HD 2
HD 3
SB 0 A
SB 1 A
SB 2 A
SB 3 A
Currently capturing network traffic
SB 0 B
SB 1 B
SB 2 B
SB 3 B
Currently stored to hard disk

Figure 4.9: Distribution of Superblocks to Hard Disk Drives
(Example with four hard disks / 8 superblocks)

without loss of Ethernet frames.

After the measurement the trace data, which is distributed across the hard disks, is merged together over the 1G Ethernet network to build one single continuous trace file for further analysis. If the deselect removal is enabled, the time required to reconstruct a continuous trace file from the Ethernet frames distributed across the hard drives is typically 6 to 10 times longer than the recording time itself and is mostly limited by the network or hard disc bandwidth of the computer performing the collection.

4.1.4 External Triggering

Beside the possibility to start and stop measurements manually via the USB interface of the 16 bit microcontroller, FPGA A was equiped with an external serial interface. This serial interface allows the starting and stopping of measurements by external hardware. This feature is extremely useful as it allows the device under test to start and stop the measurements by itself (e.g. measurements can be started shortly prior to running a specific benchmark and stopped when the benchmark ends) without requiring any user interaction. Furthermore, it provides the functionality of trace file annotation: As mentioned before, a 6 bit value can be sent to FPGA A. This 6 bit value is embedded in the trace file and can be used to detect different program execution phases.

Two external interfaces have been implemented for triggering the tracer platform. One is a PCI card, which can be connected to a PCI slot of the PC under test. The PCI card is equiped with a FPGA, which serializes accesses to PCI registers in the CPUs IO address space on the card and sends them to FPGA A.

The other interface is a USB device, equipped with a chip which provides conversion functionality of USB to a variety of several different serial protocols (Future Technology Devices International Ltd. FTDI2232). It is programmed to provide the serial data stream for FPGA A.

4.2 Verification

Two methods have been applied to verify the correctness of the captured access sequences.

In order to prove the correct distribution of the Ethernet frames to the PC cluster and the reordering of the frames (and thus time slices) in the trace file, all captured access sequence were executed on a SDRAM software model which checked:

- The fulfillment of all SDRAM timings

- Possible data bus contention and fulfillment of data bus turnaround times
- Correctness of access sequences (e.g. no reads or writes to closed banks, refreshes while banks are open, etc.)

The hardware setup was capable to record even sequences in the range of $1.5 \cdot 10^{11}$ clock cycles without any errors.

In order to ensure the correct connection of the address bus from the SDRAM to the FPGA platform, a defined access sequence was executed on the device under test by a dedicated SDRAM test system (CST Inc. SP3000 DDR2-SDRAM-Tester) which has been compared with the measured access sequence. Further analysis has been done by executing SDRAM test software on a standard PC platform.

Chapter 5

DRAM Performance Analysis

5.1 Critical DRAM Timings

During the execution of a sequence of memory requests the memory controller has to insert deselect cycles due to different reasons:

1. The SDRAM specification contains ten SDRAM timings, which have to be fulfilled by the memory controller while executing a sequence of commands to a SDRAM component. A detailed description of all timings can be found in appendix A.

2. The data bus has to be free for data transmission during the data transfer phase of a read or write transaction.

3. Additional turnaround times between accesses going to different DRAM ranks and between reads and writes have to be fulfilled due to the reversal of the data bus transmission direction or in order to change the termination of the DRAM bus.

4. Many PC mainboards use a 2T rule. This means that consecutive commands are at least two clock cycles apart from each other. This allows the memory controller to reduce the switching frequency of all signals of the CA-bus but the chip select line by a factor of two, reducing the effect of signal reflections (*inter symbol interference*, ISI) significantly.

In this chapter the reasons are determined, for which the memory controller inserted additional deselect commands and which therefore limit the system performance most significantly.

These most limiting timings should be the target for optimization by SDRAM manufacturers, as a reduction of theses timings has the highest impact on the memory systems performance.

But even if SDRAM development is currently mostly driven by an increase in memory density and not by performance optimizations, SDRAM timings have to be regarded thoroughly, as changes in SDRAM sizes affect SDRAM timing parameters.

If for example the capacity of a SDRAM component is doubled, the SDRAM designer has multiple options:

Firstly, he can double the number of columns. This would lead to an increase in the current consumption of the component during bank activations and refresh cycles and would result in an increase in current consumption related timing parameters like t_{RRD} or t_{FAW}.

Secondly, he can think of doubling the number of rows. If the number of rows is doubled, the SDRAM designer has two choices: He may either double the number of sense amplifiers, which requires additional space on the chip, or he has to cope with longer bitlines. The larger capacity of longer bitlines would lead to an increase in access time and refresh time and would therefore affect t_{RCD}, t_{RAS} and t_{RFC}.

Thirdly, he may think of doubling the number of banks. This would have the least effect on SDRAM timings but requires additional chip space for signal routing and chip access logic [54].

5.1.1 Methodology

In order to determine the most critical timings in a SDRAM system, different types of applications were executed on a personal computer. The resulting memory access sequences (*trace files*) were recorded with the measurement setup of chapter 4.

Afterwards, the CA-bus utilization was determined: In clock synchronous SDRAM systems, like DDR2, one command per clock cycle is issued on the SDRAM component. This command can be either a "useful" command like a read or write request or a ND[1]-command, which does not have an effect on the SDRAM component.

In the analysis the utilization of the SDRAM's command and address bus was determined. While the CA-bus utilization for the "useful" non-ND-commands is obvious, the deselect commands may be required either in order to fulfill one of the memory system limitations of chapter 5.1 or are just introduced as no further memory requests have to be executed on behalf of the CPU (see figure 5.1).

The deselect commands within the measured sequence are then accounted to reasons why they have to be in the trace file. This is done by trying to pull in the next non-ND-command in place of the deselect (similar to the approach of [8]). It is determined whether the command could have been executed in place of the deselect command. If this would have been possible, the deselect command is not

[1]NOP and DESELECT.

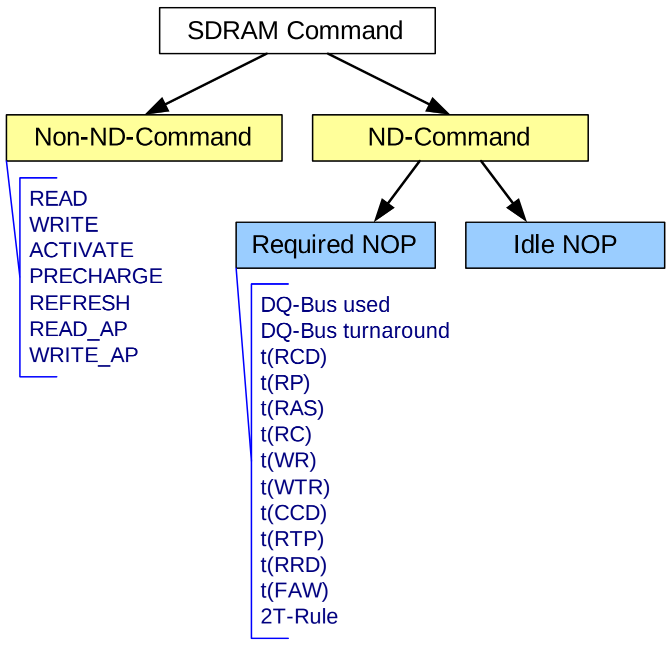


Figure 5.1: CA-Bus Utilization

required ("idle"-NOP) and may have been safely removed. Otherwise the deselect is accounted to the reasons stated above (SDRAM timings, DQ bus occupation, 2T rule violations), which would prevent a safe execution at this position.

Figure 5.2 shows an access sequence of SDRAM commands, which are issued on a single SDRAM bank. The SDRAM timings which have to be fulfilled are indicated in the Gantt[2] chart below. It is tried whether it would be possible to execute the next ND-command in place of the respective deselect (indicated by the arrows on top). The timings which would be violated by executing the non-ND-command in place of the deselect are shown on top of the time axis.

For the analysis only single rank memory systems were investigated. However, the methodology could also be applied to multi rank memory systems[3].

[2]Henry Laurence Gantt, American mechanical engineer and management consultant, 1861–1919.

[3]It shall be noted that there is a single CS# line going to each rank (refer to figure 2.11). As a result, non-ND-commands going to one rank will be seen as deselect commands by all other ranks.

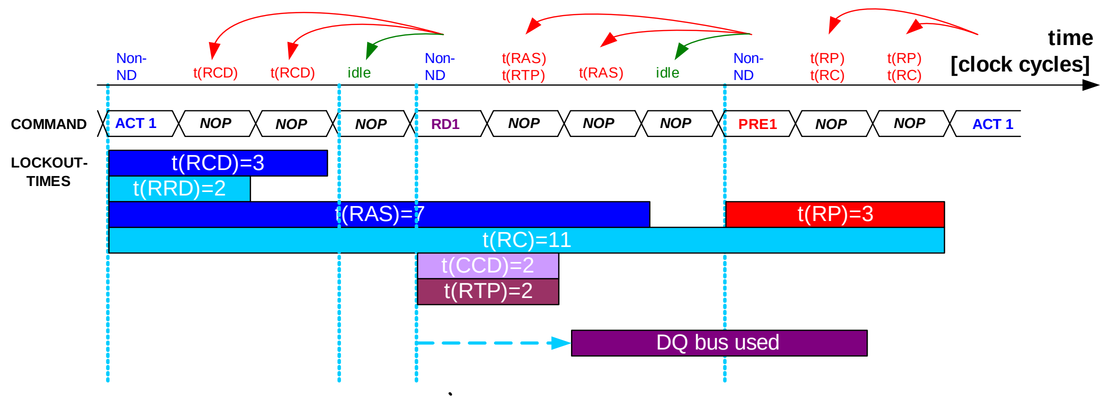


Figure 5.2: Pulling in of non-ND-Commands in Place of Deselects

Set	Benchmark	DQ-Util.	CA-Util.	WR-Ratio	Row Hit-Rate	Runtime
1	3D Mark03	48.03%	18.73%	31.68%	70.63%	346.30 s
	AquaMark	57.40%	21.80%	27.08%	72.86%	532.78 s
	CCBMPro	58.85%	21.51%	26.22%	74.59%	251.41 s
2	183.equake	39.58%	15.31%	13.29%	72.75%	132.04 s
	181.mcf	48.06%	19.45%	15.91%	69.06%	225.20 s
	189.lucas	53.09%	24.55%	31.19%	57.40%	139.02 s
	171.swim	53.67%	23.88%	29.92%	60.94%	215.61 s
	179.art	69.79%	22.62%	8.04%	85.07%	609.02 s

Table 5.1: Benchmark Set

5.1.2 Results

Two sets of applications were executed on the PC platform (see table 5.1). The first set consists of three 3D benchmarks, which were executed on platform Bb[4] (see appendix B for a detailed list of investigated systems). The second set consists of the five benchmarks of the SPEC suite, which exercise the SDRAM data bus most significantly (clusters 4, 5 and 6 in [3]). These benchmarks were executed on platform Ab.

Although the used chipset applies a 2T rule, CA-bus limitations related to this rule were ignored. The reason for this approach is that the 2T rule is not a limitation of the SDRAM itself but from the used chipset and can therefore not

[4]The nomenclation will be used in the following to refer to the pair of computer system and SDRAM device which was used for the analysis. The capital letter denotes the computer system, the small letter refers to the used memory SDRAM timings.

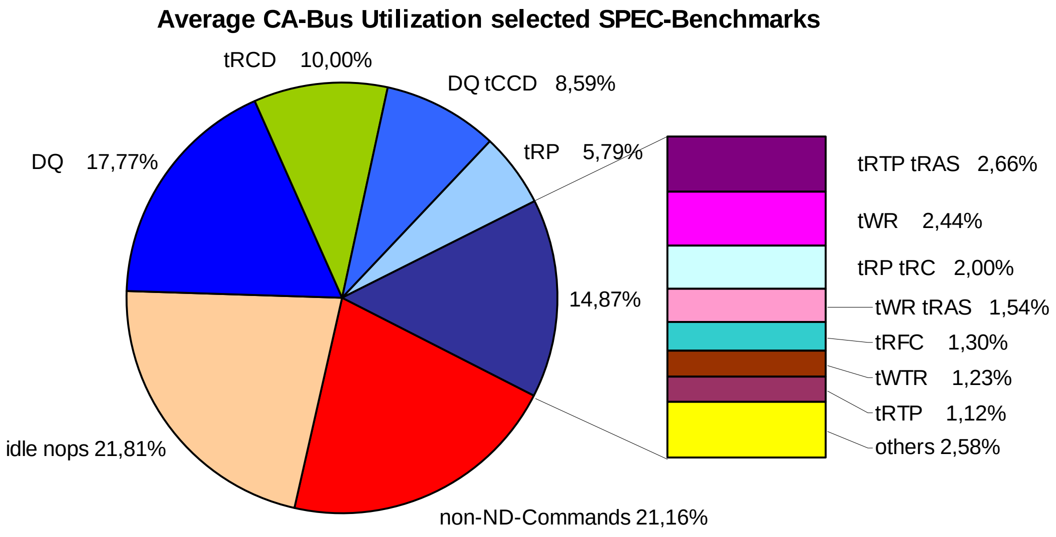


Figure 5.3: CA-bus Utilization of selected SPEC 2000 Benchmarks

be influenced by the SDRAM manufacturer. Moreover, the XT-Rule limitation affects all commands equally and, thus, its consideration would only lead to an increase in complexity due to a significant increase of timing parameter combinations occurring at the same time.

In order to weight all benchmarks equally, the CA-bus utilization was calculated for each benchmark individually. Figures 5.3 and 5.4 show the average CA-bus utilization over all benchmarks within one benchmark set.

In both cases around 20–22% of all clock cycles are spent for issuing non-ND-commands, 19–21% are ND-commands which could have been safely removed from a SDRAM point of view ("idle"-NOPs). 26–29% of all deselect commands have to be inserted due to the occupation of the data bus (sometimes overlapped with a limitation due to t_{CCD}). The remaining deselects have to be introduced due to (in order of decreasing significance):

$$t_{RCD} \Longrightarrow t_{RP} \Longrightarrow t_{RTP} \Longrightarrow t_{WR} \Longrightarrow t_{RFC} \Longrightarrow t_{WTR}$$

Around 10% of all deselects can be accounted for t_{RCD}, another 6% for t_{RP} (8% if the t_{RP} t_{RC} pair is added as well). The write recovery time t_{WR} accounts for another 3%. The remaining deselects account for more exotic combinations of SDRAM timings. Timings which are related to bank activation sequences (t_{FAW}, t_{RRD}, t_{RAS}) or activate/precharge pairs (t_{RC}) play only a minor role on the system performance. Also the turnaround times of the data bus between read and write accesses can be neglected. However, bus turnaround times may become more im-

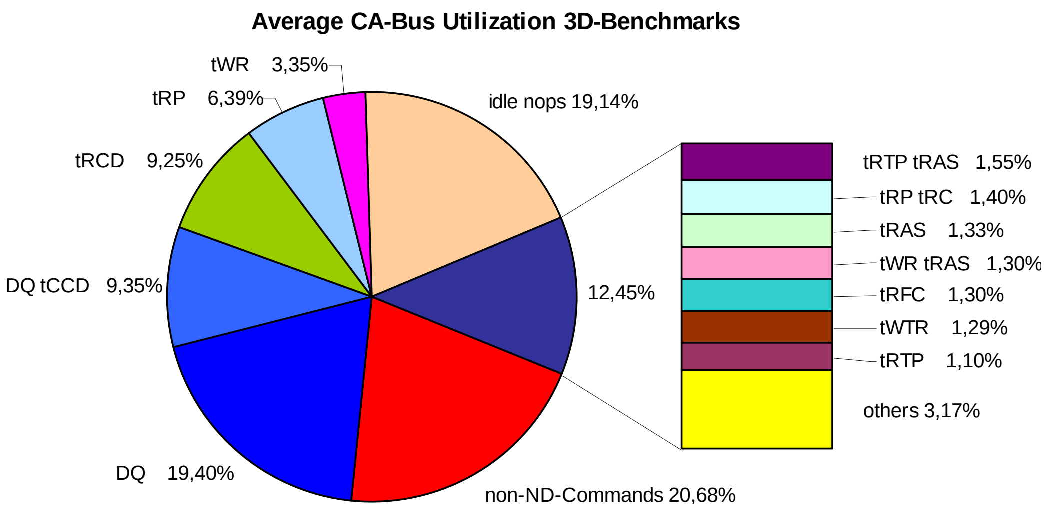


Figure 5.4: CA-bus Utilization of selected 3D Benchmarks

BM-Set	cycles affected by 2T rule	cycles affected by 2T rule only
1 (3D)	23.55%	2.87%
2 (SPEC)	21.16%	3.42%

Table 5.2: Impact of 2T Rule on System Performance

portant if the analysis would be extended to memory systems consisting of multiple SDRAM ranks.

Figures 5.3 and 5.4 show that an increase in data bus bandwidth can provide significant improvements in memory system performance. On the other hand, even if it would be possible to reduce all SDRAM timing parameters to zero, only one third of all clock cycles could be saved.

Figures 5.5 and 5.6 show the CA-bus utilization for the different benchmarks. One can see that the three 3D-benchmarks have similar behavior. For the SPEC 2000 benchmarks the CA-bus utilization depends significantly on the type of application. The cluster 6 (see [3]) benchmark 179.art produces the most traffic of all SPEC 2000 benchmarks on the data bus at a high page hit-rate. Thus, it is mostly affected by data bus limitations (DQ and DQ t_{CCD}). The benchmarks of cluster 5 (189.lucas, 171.swim) suffer from a low locality, leading to an increasing importance of SDRAM timings related to bank activations and precharge operations (t_{RCD}, t_{RP}). 189.lucas has the highest write ratio of all benchmarks and is therefore affected more seriously by write recovery (t_{WR}) and write to read turnaround times (t_{WTR}) than any other benchmark.

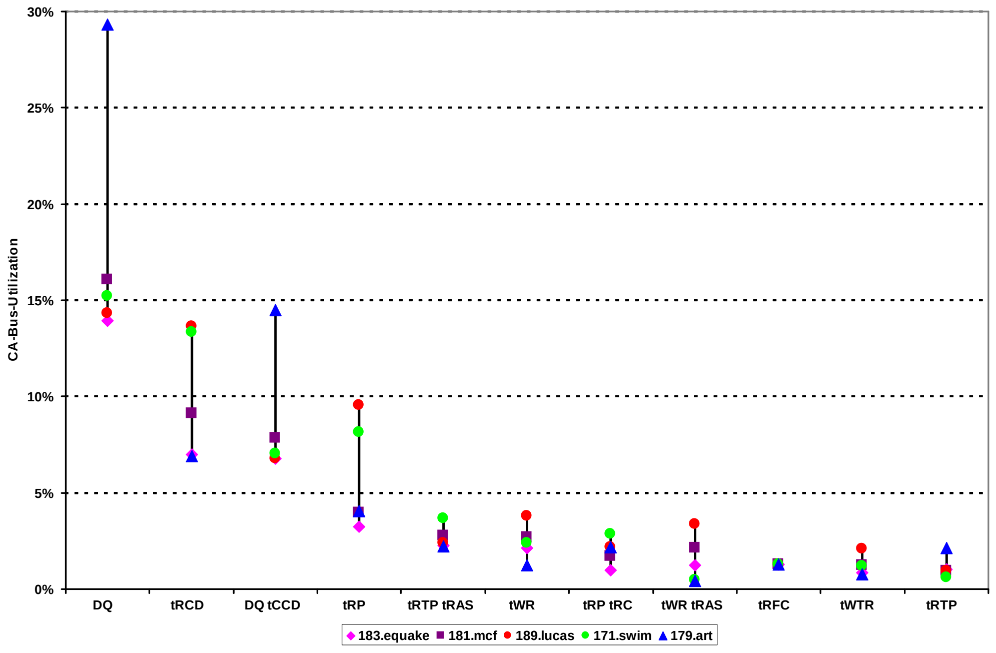


Figure 5.5: Variation of CA-bus Utilization of SPEC 2000 Benchmarks

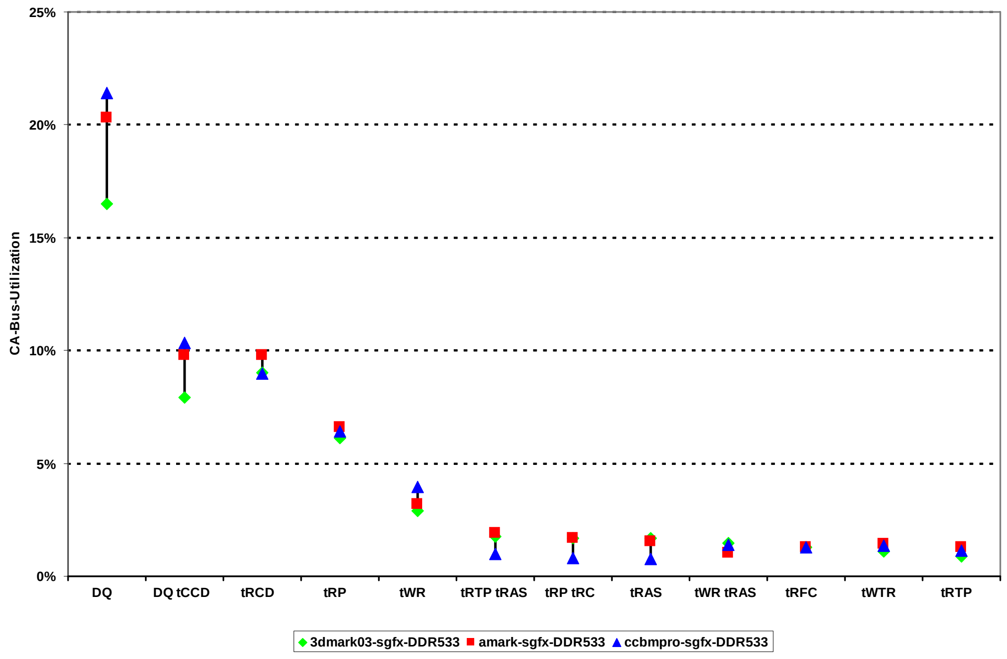


Figure 5.6: Variation of CA-bus Utilization of selected 3D Benchmarks

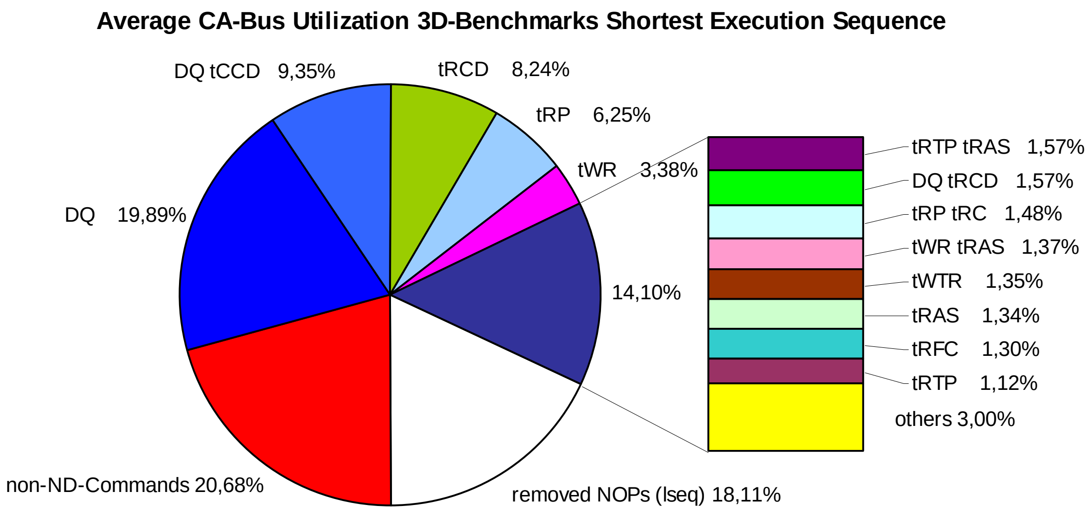


Figure 5.7: Average shortest execution Sequence of selected 3D-Benchmarks

In a second run, the same analysis was performed while applying a 2T rule. The additional 2T rule further limits the execution of SDRAM commands. As a result, some of the deselects within the access sequence can be accounted to the fulfillment of 2T rule requirements. These deselects are either taken from the pool of formerly not required deselects ("idle"-NOPs) or are from the pool of deselects which are already required to fulfill other timing restrictions.

While the first lead to an additional reduction of the memory systems performance, the latter don't limit the memory system performance as they are overlapped by other SDRAM timings.

Table 5.2 shows that nearly one quart of all clock cycles are deselects which are affected by the 2T rule. Nevertheless, less than 4% of all clock cycles contribute to the 2T rule only. The remaining deselect cycles overlap with other SDRAM timing restrictions.

5.1.3 Performance Limitations of future Memory Systems

As many publications have been created, addressing the long access times of SDRAM as limiting factor of future computer systems [57, 52], this chapter estimates the SDRAM timings, limiting the memory system performance in future computer systems.

For this approach the shortest execution sequence was determined by removing the unnecessary deselects ("idle"-NOPs) from the memory trace file during analysis. This approach can be interpreted as having an arbitrarily fast CPU (no time needed to process the requested data) and not having to wait for the data coming from the SDRAM (no impact of the CAS latency on the executed sequence). Thus, there must be no data dependency between consecutive memory accesses. Although both assumptions won't be fulfilled on a real system, the estimation can provide an upper bound for the memory systems performance. This methodology is similar to the one chosen by Wang [54]. While in the previous chapter it was only checked whether a deselect is required to fulfill timing limitations or not, now the "idle"-NOPs are removed from the access sequence. Squeezing the memory access sequence together to the shortest allowed sequence which can be executed, may lead to an increasing importance of SDRAM timings which are related to command pairs which are not consecutive (especially t_{RAS}, t_{RC}, t_{FAW}).

Figure 5.7 shows that it would have been possible to execute the memory access sequence around 18% faster. The number of removed deselect cycles is almost identical to the number of deselect commands which have been classified as unnecessary in figure 5.4.

Hence, the number of limiting SDRAM factors does not increase significantly if the CPU clock frequency is increased. This observation is supported by the relatively high row hit rate of the 3D-benchmarks (see table 5.1). As multiple requests go to the same row before a miss occurs, most timings related to non consecutive commands (e.g. t_{RAS}, t_{RC}) are already fulfilled, so that no further deselect cycles have to be introduced.

5.2 Statistical Modelling of SDRAM Parameter Changes

5.2.1 Changing SDRAM Timings

A statistical model has been derived to estimate the impact of SDRAM parameter changes on the system performance (i.e. benchmarks total runtime). The key concept of the model is that commands, which have been executed with the shortest temporal spacing which is allowed by the SDRAM specification, are limited by the memory system. A reduction of the SDRAM timing will therefore lead to a reduction of the execution time. Commands which are spaced further apart are limited by other parts of the computer system (CPU, cache, etc.) and, thus, will not be affected by changes of the SDRAM timing parameters. The shortest allowed temporal spacing of commands will be referred to as *critical timing.*

Based on the trace files generated with the measurement setup of chapter 4, the histogram of the temporal spacing between consecutive commands was created for every command pair in the command sequence of the CA-bus.

$k = 1$	DQ-Bus	RD $\rightarrow$ RD $_{\text{any bank}}$
		RD $\rightarrow$ WR $_{\text{any bank}}$
		WR $\rightarrow$ WR $_{\text{any bank}}$
		RD $\rightarrow$ RD_AP $_{\text{any bank}}$
		RD $\rightarrow$ WR_AP $_{\text{any bank}}$
		RD_AP $\rightarrow$ RD $_{\text{other bank}}$
		RD_AP $\rightarrow$ WR $_{\text{other bank}}$
		WR_AP $\rightarrow$ WR $_{\text{other bank}}$
		RD_AP $\rightarrow$ WR_AP $_{\text{other bank}}$
$k = 2$	t_{RCD}	ACT $\rightarrow$ RD $_{\text{same bank}}$
		ACT $\rightarrow$ WR $_{\text{same bank}}$
		ACT $\rightarrow$ RD_AP $_{\text{same bank}}$
		ACT $\rightarrow$ WR_AP $_{\text{same bank}}$
$k = 3$	t_{WTR}	WR $\rightarrow$ RD $_{\text{any bank}}$
$k = 4$	t_{WR}	WR $\rightarrow$ PRE $_{\text{same bank}}$
		WR $\rightarrow$ PRE_A
$k = 5$	t_{RTP}	RD $\rightarrow$ PRE $_{\text{same bank}}$
		RD $\rightarrow$ PRE_A
$k = 6$	t_{RP}	PRE $\rightarrow$ ACT $_{\text{same bank}}$
		PRE $\rightarrow$ REF
		PRE_A $\rightarrow$ ACT
		PRE_A $\rightarrow$ REF
$k = 7$	t_{RFC}	REF $\rightarrow$ ACT
		REF $\rightarrow$ REF
$k = 8$	others	

Table 5.3: Accounting of Command Pairs to SDRAM Timings

The minimum temporal spacing between most command pairs can be attributed to a specific SDRAM timing (e.g. the time between the bank activation to the first read command has to be at least t_{RCD}) or to data bus occupation. In the previous section it was shown that SDRAM timings which affect commands which are not their direct successors (i.e. t_{RAS}, t_{RC}, t_{FAW}) don't have significant impact on the system performance. Therefore, it is sufficient to look on consecutive command pairs only. Table 5.3 shows all command pairs and their relationship to a single SDRAM timing parameter.

The histograms of all command pairs contributing to one of the five most important timings which have been found in chapter 5.1 have been summed up, leading to five histograms related to SDRAM timings ($k \in [2 \ldots 7]$). In addition, one histogram containing command pairs which are limited by the data bus occupancy, like read to read command pairs ($k = 1$), and another histogram containing all command pairs which cannot be attributed to one of the five most important SDRAM timings ($k = 8$) can be created. In the following sections the reduced set of histograms will be refered to as "compressed" histograms.

From this reduced set of histograms the contribution of a single SDRAM timing to the programs execution time can be calculated.

$$T_k^O = \sum_{i=1}^{\infty} h_k^O(i) \cdot t_{CK}^O \cdot i \tag{5.1}$$

In equation 5.1 $h_k^O(i)$ denotes the histogram value in the k^{th} histogram at the i^{th} clock cycle in the original system on which the trace file was recorded; t_{CK}^O is the clock cycle time in the original system.

The total runtime T^O of the program in seconds can be calculated by adding up the contributions of all eight histograms:

$$T^O = \sum_{k=1}^{8} T_k^O = \sum_{k=1}^{8} \sum_{i=1}^{\infty} h_k^O(i) \cdot t_{CK}^O \cdot i \tag{5.2}$$

As an example, figure 5.8 shows the fraction of the overall runtime T_k^O/T^O spent on one of the SDRAM limitations of table 5.3 for one benchmark run of one of the SPEC benchmarks (181.mcf) measured on platform Ad. Where applicable each bar is split up into two sections. The left (blue) section shows the fraction of the used runtime during which the SDRAM timing is fulfilled critically. The right (green) section denotes the runtime during which the SDRAM timing is fulfilled non-critically (i.e. the commands were spaced further apart from each other than necessary).

When a SDRAM parameter is changed, two cases can occur:

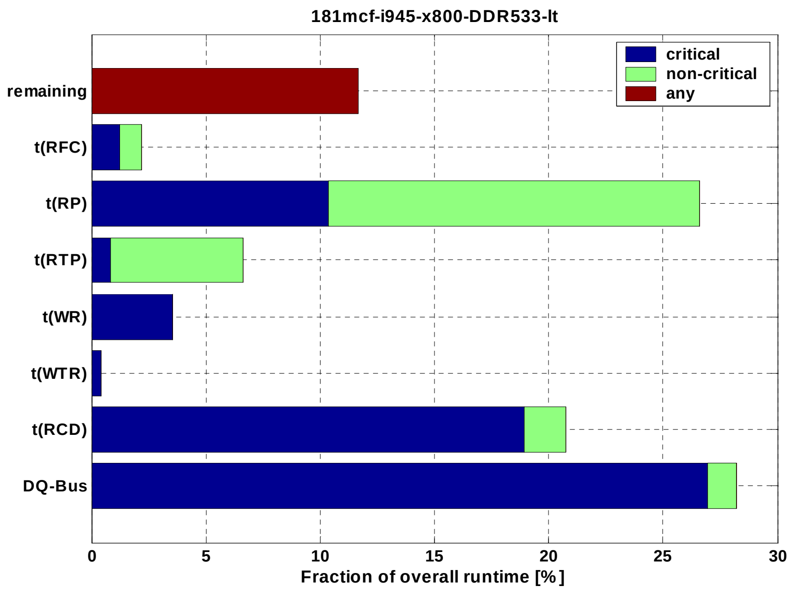


Figure 5.8: Temporal Spacing of Command Pairs belonging to a particular SDRAM Timing

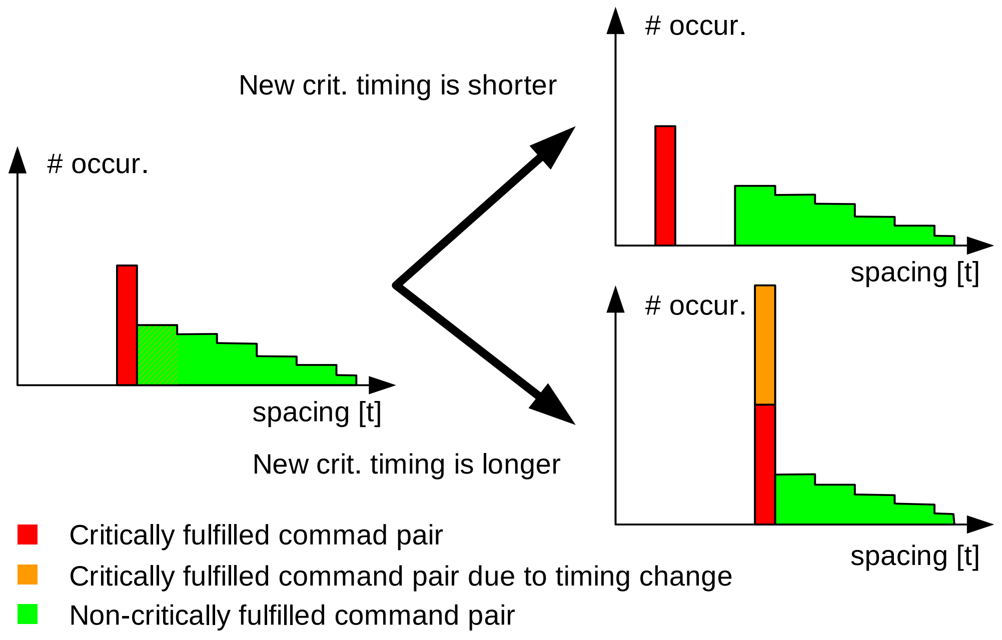

# occur.
New crit. timing is shorter
# occur.
spacing [t]
# occur.
spacing [t]
New crit. timing is longer
spacing [t]
🟥 Critically fulfilled commad pair
🟧 Critically fulfilled command pair due to timing change
🟩 Non-critically fulfilled command pair

Figure 5.9: Histogram Changes caused by Changes of SDRAM Timings

- **The new timing is shorter than the original timing**. All accesses which have been performed critically (shortest allowed timing) will be performed critically in the new system also. All accesses which have a longer than minimal temporal spacing are executed with the same timing as in the original system.

- **The new timing is longer than the original timing**. All accesses which were executed with a shorter temporal spacing in the original system have to be postponed to fulfill the new SDRAM timing and are fulfilled critically in the new system.

Figure 5.9 illustrates the histogram changes caused by changes of SDRAM timings for both cases.

The contribution of the SDRAM timing t_k to the total runtime T_k can be estimated by the equation:

$$T_k^E = \begin{cases} \underbrace{h_k^O\left(\left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil\right) \cdot \left\lceil \frac{t_k^E}{t_{CK}^O} \right\rceil \cdot t_{CK}^O}_{\text{advanced critical timing}} + \underbrace{\sum_{i=\left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil + 1}^{\infty} h_k^O(i) \cdot i \cdot t_{CK}^O}_{\text{tail of distribution}} & \text{if } t_k^E \leq t_k^O \\ \underbrace{\sum_{i=1}^{\left\lceil \frac{t_k^E}{t_{CK}^O} \right\rceil} h_k^O(i) \cdot \left\lceil \frac{t_k^E}{t_{CK}^O} \right\rceil \cdot t_{CK}^O}_{\text{accesses to be delayed}} + \underbrace{\sum_{i=\left\lceil \frac{t_k^E}{t_{CK}^O} \right\rceil + 1}^{\infty} h_k^O(i) \cdot i \cdot t_{CK}^O}_{\text{tail of distribution}} & \text{if } t_k^E > t_k^O \end{cases} \tag{5.3}$$

In the equation above t_k denotes the analog SDRAM timing from the SDRAM specification. This time is rounded up to the next clock cycle in the original clock domain. In case of data bus related stalls t_{DQ} denotes the time required for transferring a complete burst of data ($t_{DQ} = t_{k=1} = BL/2 \cdot t_{CK}^O$). As before, the total runtime can be estimated by summing up the contributions of all single SDRAM timings.

Results

Taking the measurements of four benchmarks of the SPEC benchmark suite recorded on system Ad as given, an estimation of the benchmarks runtime using equation 5.3 was performed. The SDRAM timings were varied by ±4 clock cycles with regard to their original settings.

In figure 5.10 the estimated increase and decrease of the benchmarks runtime in percent is plotted when the SDRAM timings are changed for the four benchmarks under investigation (colored bars). As the individual contributions of the SDRAM timings are independent from each other, the overall runtime change can be calculated by summing up the individual contributions of all SDRAM timing parameters.

It can be seen that the overall performance impact is mostly a linear function of the SDRAM timing. This is due to the fact that the set of "compressed" histograms has a significant peak at the critical timing (see figure 5.8) for almost all SDRAM timings while the histogram values in vicinity of the critical timing only play a minor role for the overall execution time. Accordingly, the overall runtime depends almost linearly on shifting the position of the dominating histogram peak at the critical timing.

In order to check the quality of the estimation, the estimated results were compared with benchmark runs taken on the same system but with reduced SDRAM timings (system Ab). From these measurements the "compressed" histograms of consecutive commands belonging to a particular SDRAM timing were determined as in figure 5.8. Building the "compressed" histograms allows an attribution of the overall number of saved clock cycles between the two different memory systems to the different SDRAM timings.

The performance gain/loss for each SDRAM timing due to different SDRAM timing parameters is determined by the equation:

$$\Delta T_k^{MO} = \sum_{i=1}^{\infty} \left(h_k^M(i) - h_k^O(i) \right) \cdot t_{CK}^O \cdot i \tag{5.4}$$

ΔT_k^{MO} denotes the time difference between the measured system and the original system. The relative impact of the particular SDRAM timing on the overall execution time can be determined by normalizing ΔT_k^{MO} to the execution time of the original benchmark T^O. The resulting points were also plotted in figure 5.10. The measurements on system Ab were conducted several times as they were reused in chapter 6.

Figure 5.10 shows that the estimation accuracy heavily depends on the particular SDRAM timing. The best estimation results are obtained for the t_{RCD} timing. Also the estimation for t_{WR} closely matches the measured values.

By contrast, the estimations of t_{RTP} and t_{RP} are far away from the measured results. At first glance it seems that the model of chapter 5.2 heavily underestimates the effect of SDRAM timing changes. The reason why the model performs poorly on these timings is that the command sequence is not fixed for the different benchmark runs. The reduction of SDRAM timings by some few clock cycles changes the executed command sequence significantly. The shorter SDRAM access

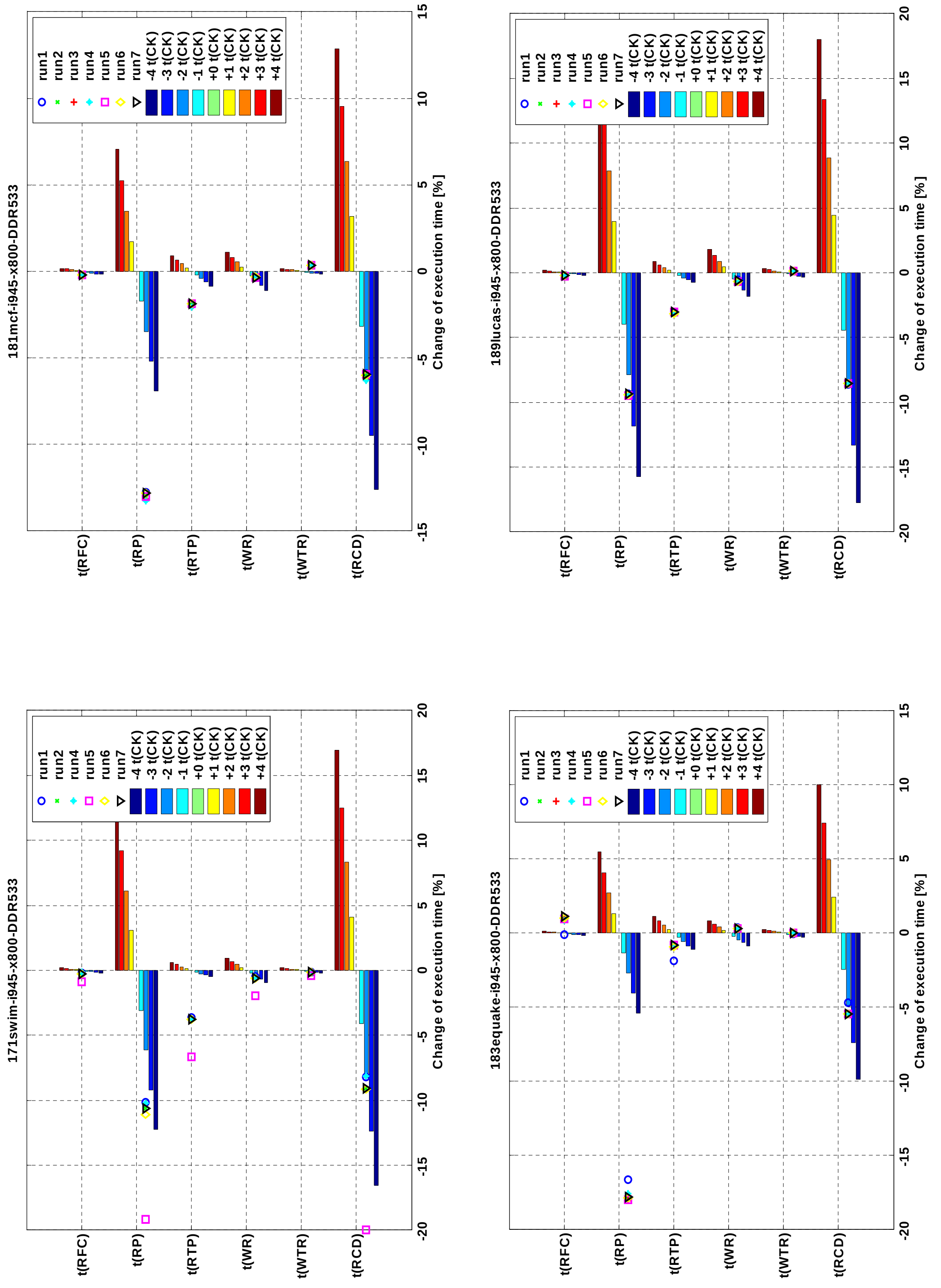


Figure 5.10: Performance Impact of SDRAM Parameter Changes

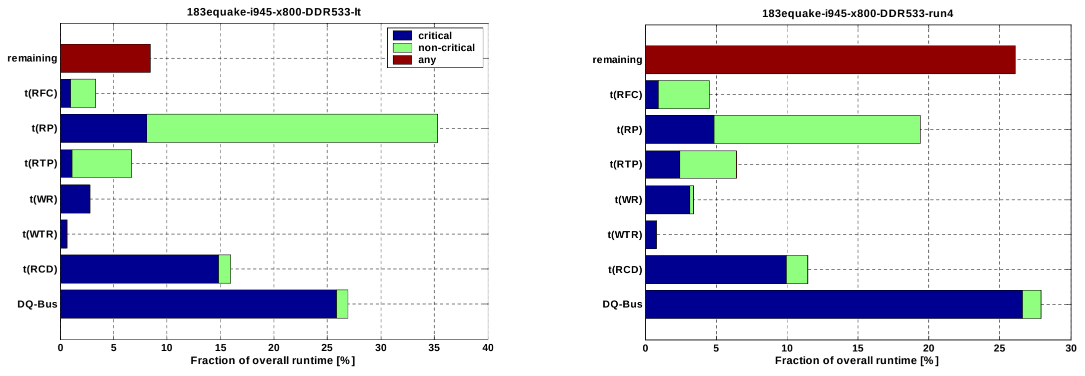


Figure 5.11: Fraction of Time spent on different SDRAM Timings

times lead to an increase in access interleaving between accesses going to different banks. The time between those accesses to different banks can no longer be attributed to one of the five most important SDRAM timings. Figure 5.11 illustrates this effect. On the left side the fraction of the benchmarks runtime spent on the different SDRAM timings is shown for 183.equake on system Ad (long SDRAM timings). On the right side the data is presented for one of the runs of the same benchmark but on system Ab. One can see that on the system using long SDRAM timings around 36% of all clock cycles are accounted to t_{RP}. When executing the same benchmark on the system with short SDRAM timings this fraction goes down to less than 20%. By contrast, the amount of clock cycles spent on command pairs which do not contribute to one of the selected SDRAM timings (these are mostly command pairs with commands going to different banks and, thus, do not impose SDRAM performance limitations) increases significantly (from 8% to over 25%). Therefore, the model does not underestimate the performance impact of particular SDRAM parameters like t_{RP}, but changing SDRAM timing parameters may change the access sequence significantly in a way that clock cycles are attributed to different histograms within the "compressed" histogram set.

As the minimum allowed refresh cycle time is equal for the original system (Ad) and the system used for comparison (Ab), the model states that the same time is spent for t_{RFC} on the fast as well as on the slow system. As the timings before, the time spent for t_{RFC} is a victim of the changes and the reordering of the commands within the execution sequence due to the changed SDRAM access times.

- Refreshing takes place periodically at a fixed rate (on average every 7.8 µs) and happens asynchronously to the program execution. When the system runs faster, the overall execution time of the benchmark decreases, and thus, the number of required periodic refreshes decreases as well. Hence, the time

Original			
System	Ab compared to Ad		
	runtime change		deviation
	estimate	real	[perc.pt.]
171.swim	-14.93%	-19.31% (run 2)	4.38%
181.mcf	-10.40%	-13.65% (run 2)	3.25%
183.equake	-8.18%	-8.06% (run 2)	-0.12%
189.lucas	-17.80%	-19.12% (run 4)	1.32%

Table 5.4: Performance Impact of SDRAM Timing Changes

used for SDRAM refreshing is lower than expected in the model when the access times decrease.

- In the model the time from refreshing the SDRAM device to the first bank activation is accounted to t_{RFC} as well as the time between two consecutive refresh cycles with no other accesses in between consecutive refresh commands.

  If access times are decreased, access bursts which were formerly executed within multiple refresh intervals on the slow system may be compressed, so that they are executed within one single refresh interval. Hence, more time is accounted to a limitation of the refresh cycle time (t_{RFC}) on the fast system than on the slow system. This effect can be observed for the 183.equake benchmark (see the t_{RFC} bar in figure 5.11)[5].

The total effect of the different contributions depends on the particular benchmark. For most benchmarks the time spent on refreshes is reduced when the access times become shorter.

Table 5.4 summarizes the estimated runtime change when switching from the slow memory system Ad to the faster system Ab. In addition, the actual change of the benchmarks runtime is shown for the measurements taken on system Ab. While in figure 5.10 it was shown that the estimation performs poorly on some SDRAM timings like t_{RP} or t_{RTP}, the estimation accuracy of the overall runtime including all SDRAM timings (see table 5.4) is higher than the individual contributions of some SDRAM timings. The reason is that wrong guesses caused by reordering of the command sequence (see figure 5.11) are largely compensated when looking at the overall benchmarks runtime.

[5] In 183.equake more than 32000 intervals of refresh to refresh pairs spaced t_{REFI} apart were measured on the fast system while only one of these pairs was found on the slow memory system.

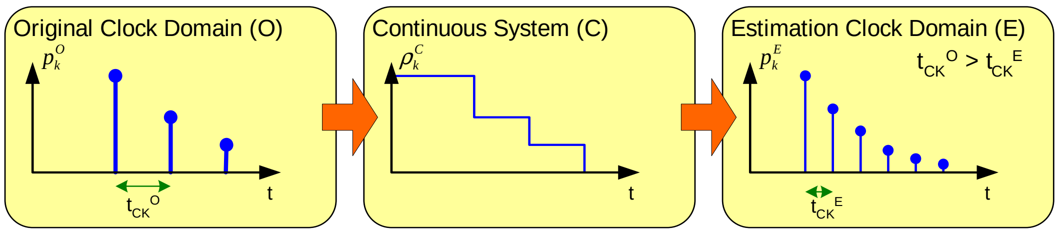


Figure 5.12: Changing SDRAM Operation Frequency: Methodology

However, for most benchmarks the model underestimated the performance gain when switching to faster SDRAM timings by up to 4.38 percentage points.

5.2.2 Changing the SDRAM Operation Frequency

Typically performance improvements of the SDRAM memory system are not restricted to single SDRAM timings. More often the operating frequency of the SDRAM interface is increased while the analog SDRAM timings remain mostly the same.

By contrast to the previous section, two additional effects can be observed when the SDRAM's operating frequency is changed:

1. As the transfer of a data burst requires a fixed number of clock cycles, the time required to transfer a memory burst is directly proportional to the clock cycle time of the memory interface and will change with the clock frequency.

2. The SDRAM memory system is a synchronous system. Changing the clock frequency leads to the situation that commands will be executed on the original system and the system for which the estimation is done at different clock cycles. The estimation of the memory systems behavior may suffer from these sampling effects. Equation 5.3 provides a simple estimation of the programs execution time in the clock domain of the source sequence. Although it can also provide an estimation of the execution time when the clock frequency of the estimation target is different from the source, rounding effects may make this estimation inaccurate.

Therefore, the algorithm of section 5.2.1 was modified in order to estimate the execution time in the clock domain of the target.

A two step approach was used. First the discrete distribution of the temporal spacing of consecutive accesses is transformed to a continuous distribution. In the second step the continuous distribution is converted back to a new discrete distribution which fits to the target clock domain. The transformation flow is depicted in figure 5.12.

For converting the formerly discrete distribution to a continuous one, it is assumed that memory requests from the CPU may occur at any time on a continuous time scale. This assumption is reasonable if the operating frequency of the CPU and cache subsystem is significantly higher that the operating frequency of the SDRAM subsystem (which it typically is in the world of multi GHz CPUs). Memory requests arriving at the memory subsystem have to be postponed up to the next clock cycle and until the critical SDRAM timing has been fulfilled before they are executed.

The probability $p_k^O(m)$ that the second access of an an access pair related to SDRAM timing k is executed on the original system with a temporal spacing of m clock cycles is given by the equation:

$$p_k^O(m) = \frac{h_k^O(m)}{H_k} = \frac{h_k^O(m)}{\sum\limits_{i=1}^{\infty} h_k^O(i)} \tag{5.5}$$

H_k denotes the overall number of access pairs accounted to SDRAM timing k. The function $p_k^O(m)$ provides an empirical estimation of the probability density function of the temporal spacing of consecutive accesses during the benchmark run. This estimation is derived by looking at the respective histograms of the long access sequences which were recorded on the computer system under test.

As the measurement reveals only the discrete distribution of the temporal spacing between consecutive commands on the device under test, an equal distribution of the command spacing of the requests arriving from the CPU between consecutive clock cycles is assumed. Consequently, in the continuous time domain the discrete probabilities (p_k^O) are distributed equally between consecutive clock cycles. Assuming an equal distribution of the temporal spacing between consecutive commands does not necessarily reflect the real distribution on the system under investigation. Nevertheless, it can be regarded as a good approximation for the real distribution if the SDRAM operating frequency is high, i.e. if the temporal spacing between consecutive samples of the discrete distribution is small.

The probability density function of the continuous distribution is given by the equation:

$$\rho_k^C(t) = \begin{cases} \dfrac{p_k^O\left(\left\lceil \frac{t}{t_{CK}^O} \right\rceil\right)}{t_{CK}^O} & \text{, if} \quad t > \left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil \cdot t_{CK}^O \\ \dfrac{p_k^O\left(\left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil\right)}{\left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil \cdot t_{CK}^O} & \text{, if} \quad t \leq \left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil \cdot t_{CK}^O \end{cases} \tag{5.6}$$

The equation on top distributes the accesses of the discrete source system equally across the time between the particular clock cycle and its predecessor. The equation below is an exception for the critical timing. Accesses which had to be postponed to fulfill the critical timing k in the source system are distributed equally on the time axis between 0 and the first clock cycle in which the second access of the command pair may take place, as its critical timing is fulfilled on the source system $(0 < t \leq \lceil t_k^O / t_{CK}^O \rceil \cdot t_{CK}^O)$.

In the second step the continuous distribution of equation 5.6 is transformed to a discrete distribution in the clock domain of the target system. All accesses which would happen at time t in the continuous time domain are postponed to be executed on the the next clock cycle of the target system. Thus, the probability that an access takes place in clock cycle n in the target system is given by:

$$p_k^E(i) = \begin{cases} \int\limits_{(i-1)\cdot t_{CK}^E}^{i\cdot t_{CK}^E} \rho_k^C(t)dt & \text{, if} \quad i > \lceil t_k^E / t_{CK}^E \rceil \\ \int\limits_{0}^{i\cdot t_{CK}^E} \rho_k^C(t)dt & \text{, if} \quad i = \lceil t_k^E / t_{CK}^E \rceil \\ 0 & \text{, if} \quad i < \lceil t_k^E / t_{CK}^E \rceil \end{cases} \tag{5.7}$$

The first equation assigns all accesses to the next possible clock cycle in the target clock domain during which they can be executed. The two equations below take care of the fact that the critical timing has to be fulfilled on the target system also. From the discrete probability density function in the target clock domain the time spent for a particular timing can be determined:

$$T_k^E = \sum_{i=1}^{\infty} p_k^E(i) \cdot t_{CK}^E \cdot i \cdot H_k \tag{5.8}$$

Equation 5.8 is a generalization of equation 5.3 for $t_{CK}^E \neq t_{CK}^O$. In the case of $t_{CK}^E = t_{CK}^O$, both equations produce the same results if $t_k^E \geq t_k^O$. For $t_k^E < t_k^O$ the results are different: In equation 5.3 it is assumed that all accesses which have been performed critically in the original system will be executed earlier and are thus shifted in the histogram. Instead, in equation 5.6 an equal distribution of accesses along the interval $[0 \ldots \lceil t_k^O / t_{CK}^O \rceil \cdot t_{CK}^O]$ is assumed in the continuous system consisting of the accesses executed critically in the original system. Accordingly, if the critical timing of the estimation becomes shorter with respect to the original settings, not all accesses are executed at the new critical timing. A detailed proof can be found in appendix C.

Figure 5.13 shows an example of the probability density functions in the different clock domains. On the positive y-axis the discrete probability density function of the original system normalized by t_{CK}^O is plotted as vertical red bars. The

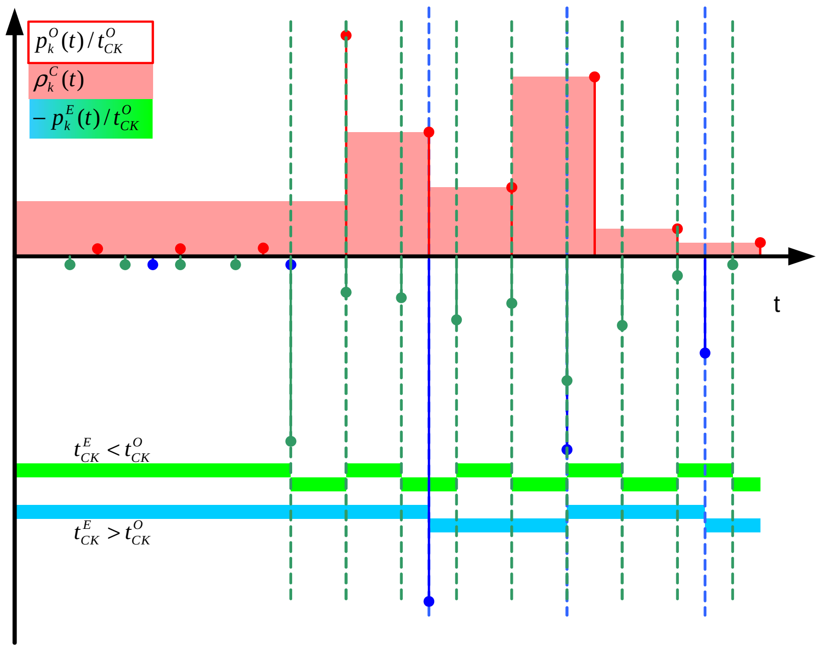
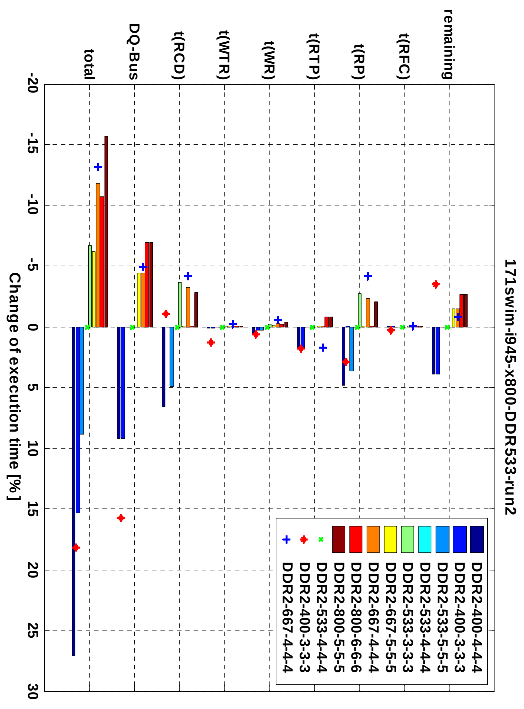
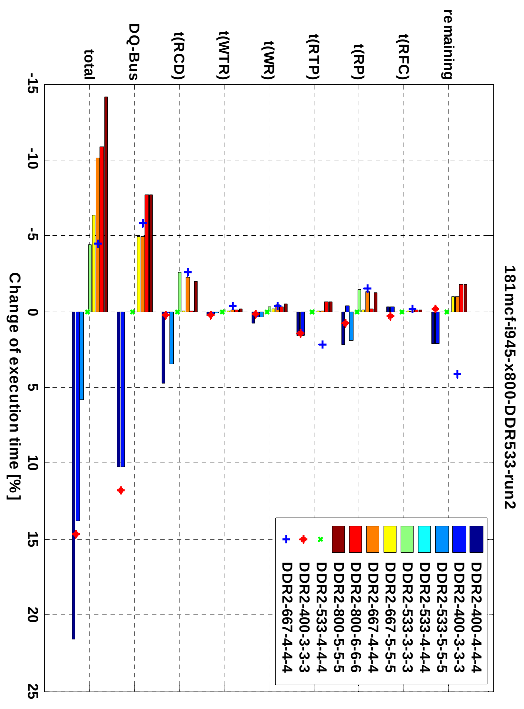
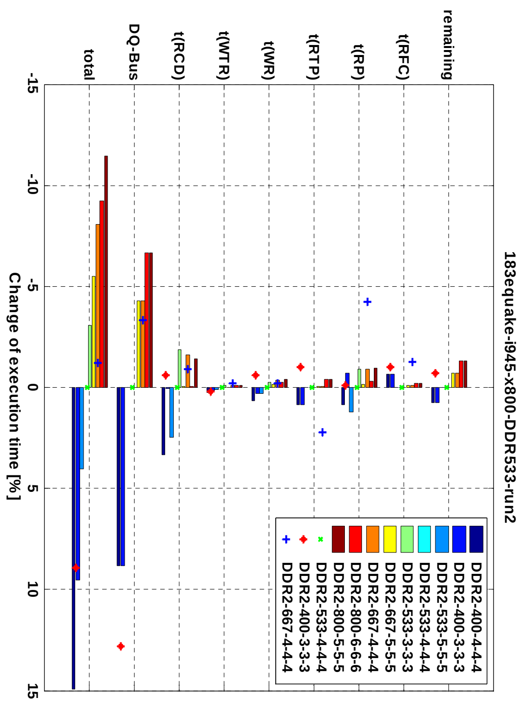


Figure 5.13: Discrete and continuous Probability Density Function of Accesses at different Clock Frequencies

SDRAM timing is executed critically in the fourth clock cycle. From the discrete probability density function the continuous probability density function is determined as described by equation 5.6 ($\rho_k^C(t)$). The area between the probability density function of the continuous system and the x-axis is plotted as light red area in the first quadrant of figure 5.13. On the negative y-axis the estimations of the discrete probability density functions are shown for higher ($t_{CK}^E < t_{CK}^O$, shown as green bars) and lower ($t_{CK}^E > t_{CK}^O$, shown as blue bars) clock frequencies. In order to maintain the scaling, both functions are also plotted normalized to t_{CK}^O. At the higher frequency, the first accesses are performed in clock cycle 5. Nevertheless, due to the higher clock frequency, accesses are executed earlier compared to the original system. At the lower frequency the critical timing is at clock cycle 3. Due to the lower operation frequency, accesses have to be delayed. The blue and green stripes at the bottom of figure 5.13 indicate the integration intervals which are used to determine the discrete probability density function in the different target clock domains: All accesses which occur within one single colored stripe are postponed and executed in the next clock cycle (i.e. at the time which limits the integration interval on the right side).







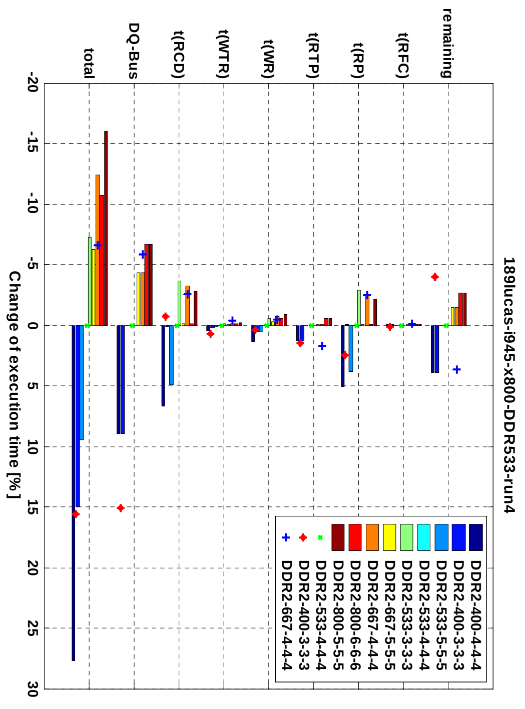


Figure 5.14: Performance Impact of SDRAM Operating Frequency Changes

Results

Using the measured access sequences of the same four benchmarks of the SPEC benchmark suite as in the previous section, an estimation of the time used for the different SDRAM timings was performed. The benchmarks were recorded with a SDRAM running at speed grade DDR2-533-444 (system Ab). As multiple runs of the benchmark were recorded at this speed grade for the repeatability study of chapter 6, benchmark runs were selected as starting point of the estimation which closely resemble the mean values of chapter 6. All speed grade/SDRAM timing combinations which are specified in the SDRAM specification [32] or are commercially available were chosen as target for the estimation. From the difference in execution time between the estimation and the measurement in the original system, the speedup of the benchmarks execution time which can be attributed to a particular SDRAM timing was calculated:

$$s_k^E = \frac{T_k^E - T_k^O}{T^O} \tag{5.9}$$

In figure 5.14 the estimated benchmarks speedup for the different SDRAM timings compared to the original system on which the access sequence was recorded is plotted (colored bars).

It can be seen that changing the SDRAM operation frequency mostly affects the data bus operation: Increasing the clock frequency while keeping the burst length equal reduces the time required for transferring a data burst and increases the memory systems bandwidth. The shorter transfer time reduces the time the system has to wait due to data bus occupation.

The second most important timing is t_{RCD}. It can also be seen that none of the remaining SDRAM timings can provide a speedup of more than 5%. This result is not surprising. As the analog SDRAM timings are mostly the same across all SDRAM speed grades, changing the clock frequency doesn't affect the system performance for these timings at all. This holds true for t_{WR}, t_{RTP}, t_{RFC} and also for t_{WTR}. The t_{WR} goes down from 10 ns at DDR2-400 to 7.5 ns for all remaining speed grades. Nevertheless, as the following read access has to be postponed up to the next clock cycle, the performance benefit is less than 2.5 ns per command pair at speed grades DDR2-533 and DDR2-667. The reduced clock cycle time at higher speed grades may slightly reduce penalties induced by the requirement of rounding the analog SDRAM timing up to the next clock cycle.

As in the previous chapter the estimation results were verified by taking measurements of the investigated benchmarks at the SDRAM target frequency of the estimation (200 MHz/DDR2-400 and 333.5 MHz/DDR2-667). The speedup of the measured system compared to the original system can be determined by the equation:

$$s_k^M = \frac{T_k^M - T_k^O}{T^O} \tag{5.10}$$

with

$$T_k^M = \sum_{i=1}^{\infty} h_k^M(i) \cdot t_{CK}^M \cdot i \tag{5.11}$$

h_k^M denotes the histogram of the temporal spacing of the command pair k while t_{CK}^M is the SDRAM's clock cycle time on the system used for verification.

The impact on the execution time for each SDRAM timing is shown in figure 5.14 also.

Comparing the estimation with the measurements reveals that the estimation only loosely fits to the measured results. Unlike the previous section, in which only particular SDRAM timings were changed, changing the clock frequency leads to a remarkable reordering of the access sequence. Furthermore, even the overall number of memory requests executed by the benchmark at different clock frequencies is at least dissimilar enough to mask the minor performance changes induced by changes of the SDRAM's operating frequency.

In order to distinguish errors of the performance estimation from errors induced by the different workloads (i.e. different command count and command pairs), the histogram of the command pairs was normalized, so that the overall number of executed command pairs on the system used for verification is equal to the number of command pairs occurring on the original system and in the estimation.

$$\tilde{T_k^M} = T_k^M \cdot \frac{\sum\limits_{i=1}^{\infty} h_k^O(i)}{\sum\limits_{i=1}^{\infty} h_k^M(i)} \tag{5.12}$$

The results after normalization can be found in figure 5.15. Table 5.5 summarizes the estimated and measured impact on the overall system performance for the normalized results and without applying normalization.

When estimating the system performance of slower SDRAM speed grades, three out of four benchmarks ran slower than predicted (i.e. the model underestimated the penalty induced by the slower access times). When normalization is applied, the estimation accuracy slightly decreases. This result shows, that on the slow system the overall number of SDRAM accesses is smaller than on the fast system.

When estimating the system performance of higher SDRAM speed grades, the model significantly overestimates the performance improvement for three out of four benchmarks. The reason is that the fast system performs more accesses during the benchmarks runtime. Thus, the measurement results have to be normalized to the same number of accesses to be comparable.

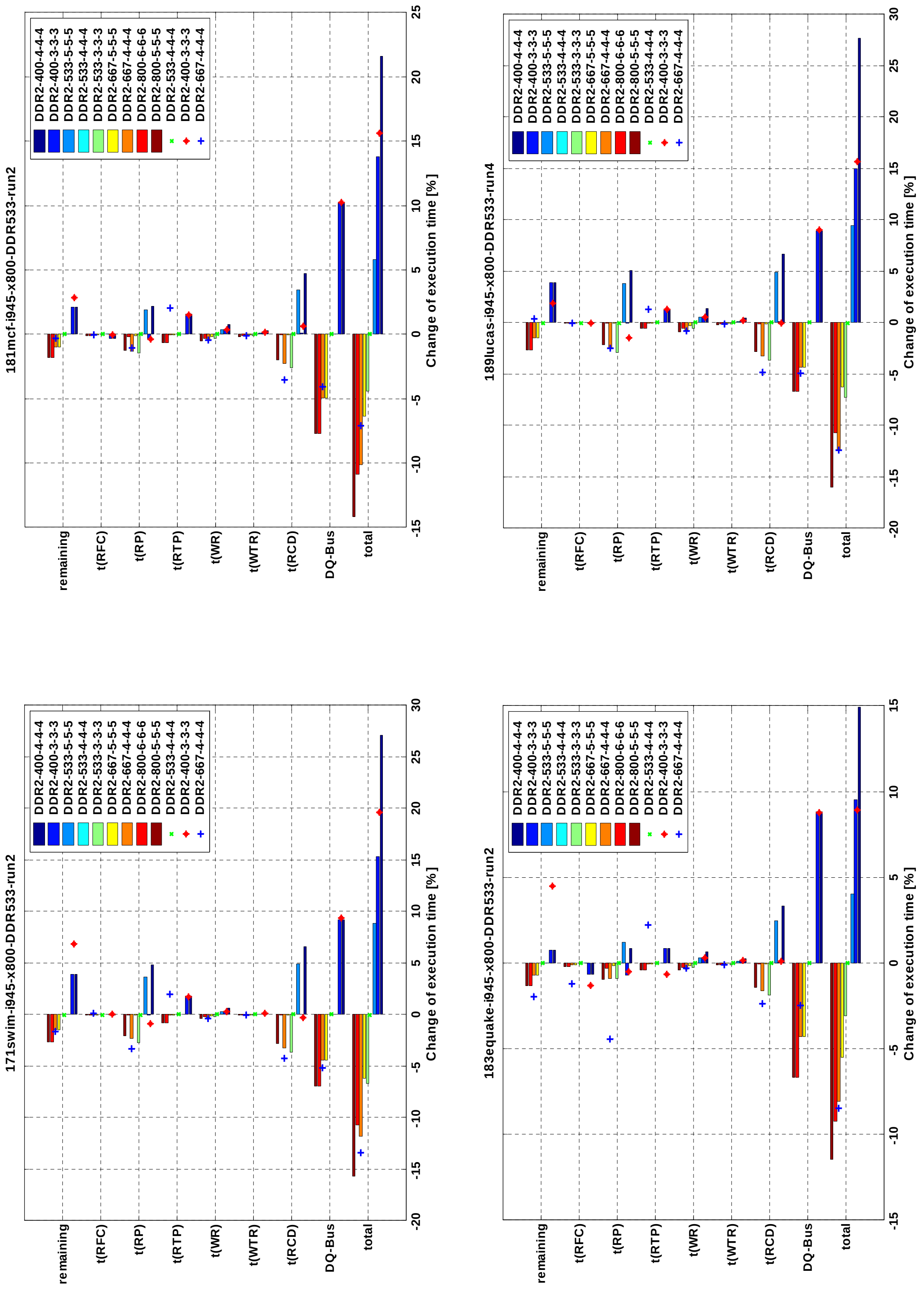


Figure 5.15: Performance Impact of SDRAM Operating Frequency Changes with normalized Access Pairs

Original						
System	Aa (DDR2-400-333)			Ac (DDR2-667-444)		
	runtime change		deviation	runtime change		deviation
	estimate	real		estimate	real	
171.swim	15.31%	18.17%	-2.86%	-11.84%	-13.13%	1.29%
181.mcf	13.83%	14.68%	-0.85%	-10.10%	-4.49%	-5.61%
183.equake	9.53%	8.93%	0.60%	-8.08%	-1.23%	-6.85%
189.lucas	15.03%	15.57%	-0.54%	-12.36%	-6.59%	-5.77%

Normalized						
System	Aa (DDR2-400-333)			Ac (DDR2-667-444)		
	runtime change		deviation	runtime change		deviation
	estimate	real		estimate	real	
171.swim	15.31%	19.60%	-4.29%	-11.84%	-13.38%	1.54%
181.mcf	13.83%	15.63%	-1.80%	-10.10%	-7.10%	-3.00%
183.equake	9.53%	8.92%	0.61%	-8.08%	-8.49%	0.51%
189.lucas	15.03%	15.63%	-0.60%	-12.36%	-12.39%	0.03%

Table 5.5: Performance Impact of SDRAM Frequency Changes

Anyway, table 5.5 shows that with normalization the estimation of the system performance deviates less than five percentage points from the measurements. In two out of four cases the estimation diverges from the measurements by less than one percentage point.

But even without normalization the runtime can be estimated with an error of less than three percentage points when estimating the performance of a lower speed grade. For three out of four benchmarks the error was even less than one percentage point. Even though the estimation of single SDRAM timings may be significantly erroneous, the overall system performance is dominated by the DQ bus utilization and t_{RCD}. For these two SDRAM limitations the estimation accuracy is high even without normalization.

However, in the unnormalized case the estimation accuracy is poor when estimating the performance of higher speed grades. Without normalization to the same number of command pairs being executed in the estimation and the system used for verification the model frequently overestimates the speedup caused by improvements of the SDRAM's access times significantly.

5.2.3 Estimating Intercommand Spacing

In the previous two sections it was shown that estimating the performance increase or decrease across different SDRAM speed grades can be done by looking at the histograms of the temporal spacing of consecutive commands. It was shown that these estimations are quite accurate compared to measurements on real computer systems.

Nevertheless, the models derived there suffer from the fact that the experimenter has to be aware of the intercommand spacing of all command pairs during the entire benchmark run in order to build the required histograms. These histograms can be created by analyzing access sequences either of recorded memory accesses on a real computer system as it was done for this theses or of access sequences obtained by simulations. In both cases long access sequences have to be analyzed.

As already shown in section 5.2.2, $p_k^O(m)$ denotes the discrete probability density function of the spacing between consecutive commands belonging to a particular SDRAM timing k in the original system. Similarly to the methodology of section 5.2.2 this discrete probability density function shall be replaced by a continuous probability density function representing the arrival of commands at the SDRAM controller. This continuous probability density function will be discretized again afterwards to match the clock domain of the system for which the performance estimation shall be performed. Opposed to the previous chapter, the continuous system is not generated by assuming a piecewise equal distribution between consecutive clock cycles. Instead an appropriate global analytical function is fitted to the points of $p_k^O(m)$ in order to obtain the continuous distribution with the probability density function $\rho_k^C(t)$. The advantage of the described approach is that the access behavior of an entire benchmark can be fully characterized by the parameters of eight analytical probability density functions.

For many SDRAM timings an exponential distribution is a reasonable assumption to describe the access behavior. The exponential distribution is typically used to describe the probability of the first occurrence of a particular event (in this case the second command of the respective command pair). The exponential distribution can be seen as continuous counterpart to the geometric distribution [20, p.279].

The exponential distribution can be applied to describe the inter command spacing of consecutive read or write requests (i.e. DQ bus limitation, t_{WTR}). Furthermore, it can be used for read/write requests which have to be initiated by a bank activation or a row replacement on the same bank (i.e. t_{RTP}, t_{WR}). As most memory systems issue a bank activation as response to a memory read/write request of the CPU or other bus master in the system, a bank activation is in most cases concluded directly with a read or write transaction to the freshly opened row. Thus, the distribution of the t_{RCD} distribution does not follow an exponential distribution, but the majority of accesses are directly initiated with the critical timing after the bank activation. Nevertheless, for the estimation of small changes of t_{RCD} an exponential distribution can be used. The optimization algorithm presented in equation 5.17 will find a relatively large coefficient for b, which will lead to an extremely steep exponential distribution, which resembles the real probability distribution of t_{RCD}.

Similarly, an exponential distribution cannot be simply applied to t_{RFC} ($k = 7$). As shown in table 5.3 both the refresh to refresh and the refresh to activate command pairs contribute to t_{RFC}.

For t_{RFC} two cases have to be investigated: For the refresh to refresh command pair most memory controllers will schedule refreshes with the time of the average periodic refresh interval (t_{REFI}) if possible. This ensures that the latency of accesses is minimal[6] while also minimizing power consumption. "If possible" means that there must be no accesses to the SDRAM rank between the two refresh cycles. Otherwise the command pair would be attributed to the refresh to activate command pair instead. For the refresh to refresh pair, the probability density function of the t_{RFC} intercommand spacing follows more likely a normal distribution with its center around the average periodic refresh interval (t_{REFI}). For this type of refreshes changing the minimum refresh interval (t_{RFC}) will not have any significant impact on the system performance. However, for benchmarks exercising the memory system significantly as those benchmarks investigated in this thesis, the number of intervals of t_{REFI} during which no memory accesses take place is negligible.

The second type of accesses which are attributed to t_{RFC} are refresh to activate pairs. For these accesses assuming an exponential distribution is reasonable. Furthermore, there may be refresh to refresh pairs which are executed with shorter temporal spacing. The memory controller may concentrate refresh cycles, e.g. because some memory refreshes had to be postponed to fulfill memory read write requests earlier.

Lastly, the exponential distribution does not fit to the histogram of temporal spacings which cannot be attributed to a particular SDRAM timing (i.e. $k = 8$). However, the results were also included in table 5.6 and figure 5.16 to demonstrate this effect. Instead of approximating this class with an exponential distribution, the overall time spent on executing those command pairs which cannot be attributed to a particular SDRAM timing should be kept constant for the performance estimation of the target system.

[6]Refreshing requires that all SDRAM banks have to be closed beforehand, so that accesses occurring after a refresh will always require a bank activation (row miss). Furthermore, memory requests arriving at the memory controller during refresh have to be postponed until refreshing has been completed.

Fitting an exponential Distribution

The probability density function of an exponential distribution is given by equation 5.13:

$$\rho_k^C(t) = b\, e^{-b\, t} \tag{5.13}$$

As in the previous chapter it is assumed that commands may arrive at the SDRAM controller at any time and will be postponed until the next clock cycle of the SDRAM clock. Thus, the probability that a command to the memory is issued at a particular clock cycle is given by integrating the continuous probability density function for the time between the particular clock cycle and its predecessor. One exception to this rule is the critical timing. As all accesses are postponed to fulfill the critical timing, the integration bounds for $i = \left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil$ have to be extended to the interval from $[0 \ldots \left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil \cdot t_{CK}^O]$.

Least square fitting is applied in order to find the parameter b of the exponential distribution which matches to the discrete distribution of the original system best. The difference between a point $p_k^O(i)$ in the discrete probability density function of the original system and the estimated continuous probability density function is given by equation 5.14:

$$\delta(i) = \begin{cases} \int\limits_{(i-1)\, t_{CK}^O}^{i\, t_{CK}^O} b\, e^{-bt} dt - \frac{h_k(i)}{H_k} & \text{, if} \quad i > \left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil \\ \int\limits_{0}^{i\, t_{CK}^O} b\, e^{-bt} dt - \frac{h_k(i)}{H_k} & \text{, if} \quad i = \left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil \\ 0 & \text{, if} \quad i < \left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil \end{cases} \tag{5.14}$$

Furthermore, $\delta(i)$ is zero for all $i < \left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil$.

The total difference Δ between the continuous distribution and the discrete distribution of the original system if given by the sum of the squares of all differences from equation 5.14.

$$\Delta(b) = \sum_{i=1}^{\infty} \delta(i)^2 \tag{5.15}$$

$$= \left(\int_{0}^{\left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil \cdot t_{CK}^O} b\, e^{-bt} dt - \frac{h_k\left(\left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil\right)}{H_k} \right)^2 \tag{5.16}$$

$$+ \sum_{i=\left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil + 1}^{\infty} \left(\int_{(i-1)\, t_{CK}^O}^{i\, t_{CK}^O} b\, e^{-bt} dt - \frac{h_k(i)}{H_k} \right)^2 = min \tag{5.17}$$

Finding the minimum of $\Delta(b)$ delivers the optimal value of the parameter b for which the estimated exponential distribution fits best to the discrete probability density function of the original system. The optimal value of b has to be determined numerically. Most methods which can be used to determine the local minimum of a function numerically require an initial guess for the value of b or an interval in which the local minimum shall be searched. The value of the parameter b of the exponential distribution is mostly determined by the accesses issued with the critical timing. At the critical timing the exponential function does not only have it's maximum but also has an extended integration interval during curve fitting. An initial estimation $\check{b}$ of the optimal parameter of the exponential distribution function can therefore be obtained by looking only at the critical timing. Disregarding the remaining points of the discrete distribution leads to:

$$\int_{0}^{\left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil \cdot t_{CK}^O} \check{b} e^{-\check{b}t} dt = \frac{h_k\left(\left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil\right)}{H_k} \tag{5.18}$$

$$\check{b} = \frac{-ln\left(1 - \frac{h_k\left(\left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil\right)}{H_k}\right)}{\left\lceil \frac{t_k^O}{t_{CK}^O} \right\rceil \cdot t_{CK}^O} \tag{5.19}$$

The initial guess $\check{b}$ can be used as starting value for a numerical approximation of the distribution's exponent b.

After determining the optimal value for b which minimizes Δ, the discrete probability density function in the target clock domain can be determined as already

	Benchmark							
	171.swim		181.mcf		183.equake		189.lucas	
k	b_{opt} $[\cdot 10^8]$	$\epsilon_k(b_{opt})$ $[\%]$	b_{opt} $[\cdot 10^8]$	$\epsilon_k(b_{opt})$ $[\%]$	b_{opt} $[\cdot 10^8]$	$\epsilon_k(b_{opt})$ $[\%]$	b_{opt} $[\cdot 10^8]$	$\epsilon_k(b_{opt})$ $[\%]$
DQ	3.587	0.8068	3.029	0.7102	2.437	1.6086	3.054	1.2066
t_{RCD}	1.776	7.6038	1.405	7.5506	1.649	6.0281	1.360	9.3081
t_{WTR}	2.488	0.0153	2.984	0.0029	2.604	0.0108	3.138	0.0019
t_{WR}	1.897	0.0256	1.476	0.0686	0.704	0.5559	1.539	0.0983
t_{RTP}	1.943	5.6175	1.972	4.1382	0.525	4.9499	2.001	5.5092
t_{RP}	2.549	1.5697	1.359	5.0565	1.102	6.3746	2.269	1.9505
t_{RFC}	0.279	0.4181	0.210	1.3331	0.113	2.9670	0.425	0.3104
oth.	4.380	3.0962	2.256	8.6393	1.938	10.4156	3.668	4.1712

Table 5.6: Optimal values for the parameter b of the Exponential Distribution

shown in equation 5.7. ϵ_k denotes the goodness of fit and is given by the root of the sum of the squared differences for the different SDRAM timings.

$$\epsilon_k(b_{opt}) = \sqrt{\Delta(b_{opt})} = \sqrt{\sum_{i=1}^{\infty} \delta(i)^2} \tag{5.20}$$

Results

The performance estimation conducted in the previous section with the results shown in figure 5.15 was repeated using exponential distributions in order to estimate the histograms of all SDRAM timings. Table 5.6 shows the optimal values of the parameter b for the different SDRAM timings.

The estimation results were compared to measurements on the real system. As in the previous section, the number of accesses measured for the different SDRAM speed grades were normalized to the number of accesses executed on the original system (see equation 5.12) to make them comparable. Figure 5.16 shows the performance change for different SDRAM speed grades compared to the original system (Ab).

For performance limitations induced by DQ bus utilization the estimation provides accurate results. Nevertheless, for SDRAM speed grade DDR2-667 the performance gain is overestimated for two out of four benchmarks (181.mcf, 183.equake).

For the second most important SDRAM timing t_{RCD} the estimation also provides accurate results. For two of the benchmarks (181.mcf, 189.lucas) the performance gain is slightly underestimated compared to the real system.

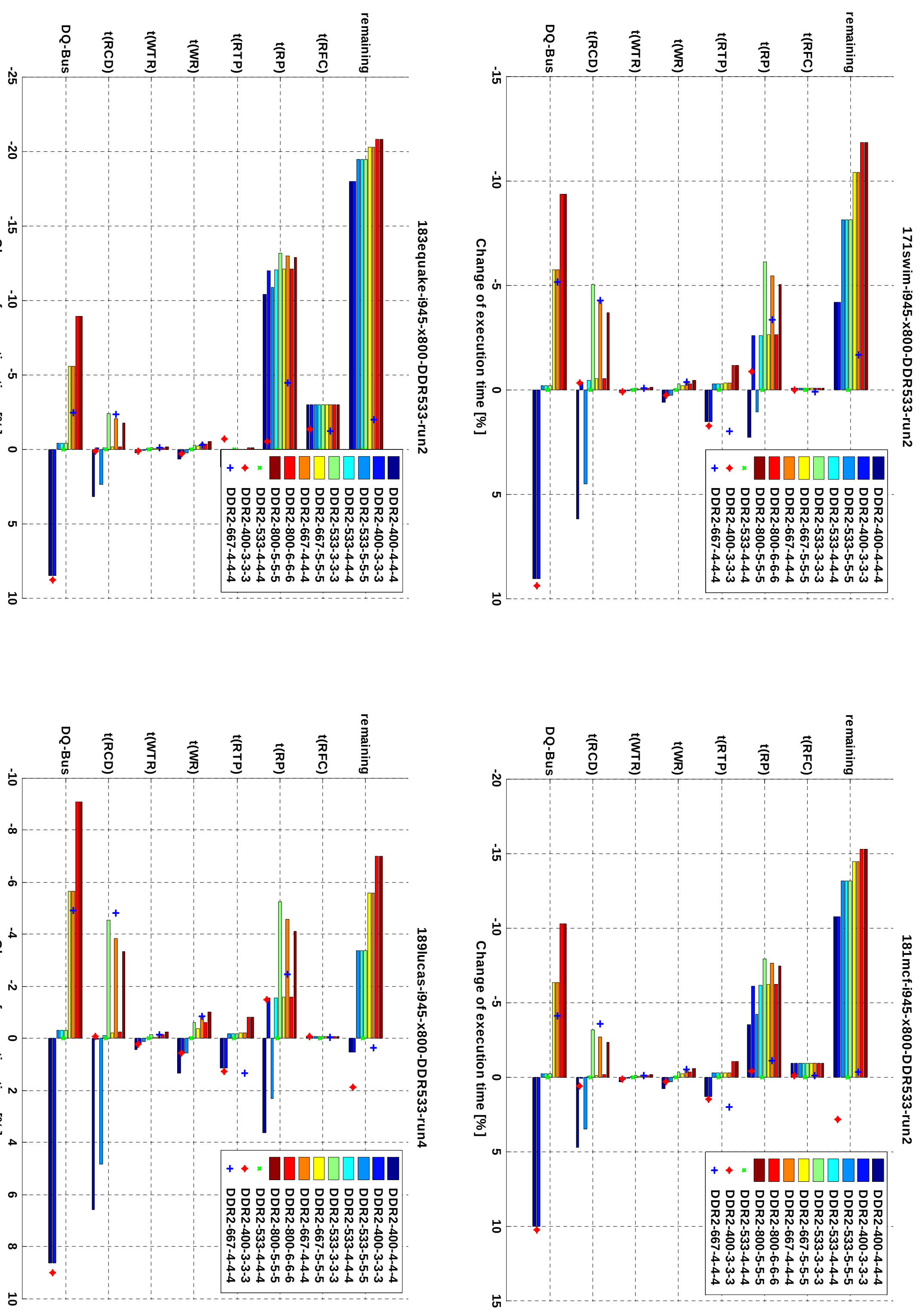


Figure 5.16: Performance Impact of SDRAM Operating Frequency Changes estimated using an exponential Distribution

The exponential distribution can also be used to model the benchmarks overall runtime accounted to t_{WTR}, t_{WR}. The estimation of t_{RTP} is accurate for lower SDRAM speed grades (i.e. DDR2-400). However, for higher SDRAM speed grades like DDR2-667 the system spends even more time with t_{RTP} than on systems using slower SDRAM memory.

For t_{RP} and t_{RFC} the estimated results do not fit to the measured results. Unlike most other timings, the discrete probability density function for t_{RP} and t_{RFC} decays slowly in the original system. Thus, the tail of the respective distribution has significant impact on the benchmarks runtime. This holds true especially for benchmarks which have a relative low SDRAM utilization (like 181.mcf or 183.equake). This slow decay makes it difficult to fit a single exponential function to all points of the histogram. Instead, for these timings a piecewise approximation of the histogram by two or more analytical functions is highly recommended[7]. Unfortunately, a piecewise approximation of the histograms introduces the problem of finding optimal interval boundaries for the sub-functions. Even worse, the solution lacks the benefit of having a small set of very simple analytical functions to describe the access behavior of an entire benchmark run.

The estimation of chapter 5.2.2 constitutes the other extreme: Here the continuous probability density function was constructed by piecewise interpolation of neighboring points of the discrete probability density function of the original system.

Figure 5.16 shows that even though the probability density functions were approximated by very simple exponential distributions, the impact of SDRAM parameter changes can be estimated with sufficient accuracy for the most important SDRAM timings. The results also show that memory performance estimation doesn't necessarily require complex models for simulating the behavior of memory access performance. Even assuming simple exponential distributions between consecutive SDRAM accesses may provide sufficient accuracy for estimating the system performance of future SDRAM systems.

[7]Looking at the respective histograms reveals that the histograms should be approximated by at least two exponential functions.

Chapter 6

Reproducibility

System simulators can start workloads from a given system state, which can be saved and restored for every simulation run. Thus, running the same benchmark multiple times in the simulation environment leads to exactly the same performance results.

Most of today's operating systems support virtual memory management. The operating system provides the user with more primary memory (virtual memory) than physically installed in the system. The operating system takes care of mapping the CPUs address space to physical memory. If an application requests more primary memory than physically available, the OS has to write currently unused memory to secondary storage (and has to load it back from there to physical memory dynamically when needed again) in order to fulfill the request. This leads to the situation that the memory map seen at the beginning of a measurement depends on the memory allocations in the past (i.e. memory requests of tasks which have been executed earlier). Furthermore, the operating systems memory allocation algorithm typically evaluates former accesses on memory pages to determine a replacement strategy for less frequently used pages. Thus, also the future assignment of primary memory to the benchmark under investigation depends on events which occurred in the past.

Unfortunately, the experimenter has limited control over this mapping of logical addresses to physical memory[1]. In addition, multitasking operating systems may run several other (system) tasks asynchronously in the background which may produce additional memory requests. These additional requests influence the memory allocation algorithm and, thus, the assignment of physical memory to the different tasks during the benchmarks execution time.

This leads to the situation that the memory map may vary between consecutive runs of the same benchmark significantly. Therefore, the impact of the unknown

[1]at least, if the operating system shall be left unmodified.

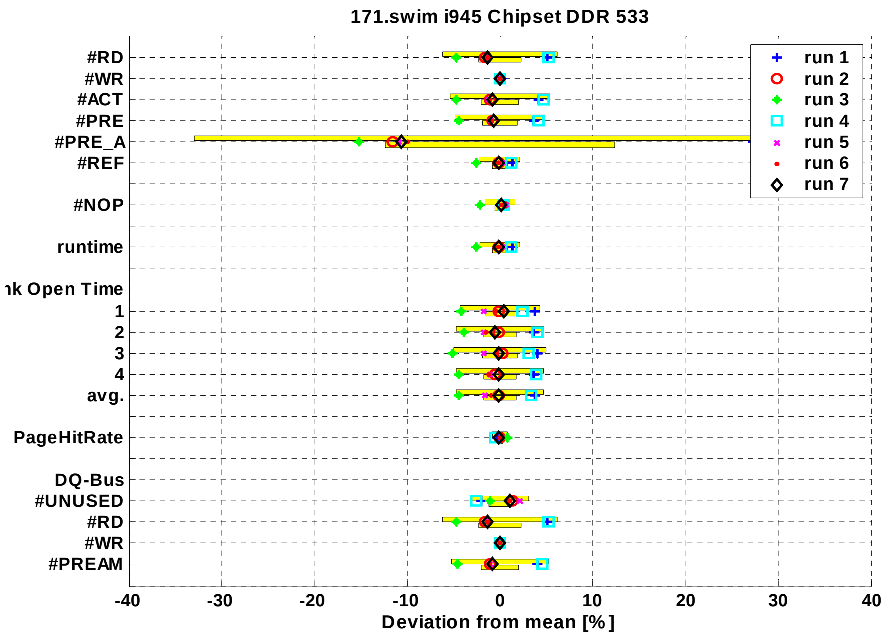


Figure 6.1: Deviation of Key Figures from Mean Value: 171.swim

starting state and the – from the experimenters point of view – random allocation of physical memory pages to the logical address space on the measured performance metrics of chapter 3 has to be determined to ensure that the measurements are representative for the workload under investigation.

Four benchmarks of the SPEC 2000 suite were executed and recorded on the same hardware setup multiple times. In order to obtain a different memory map for the different runs, other tasks were executed in between consecutive runs such as system reboots, computer games, text processing or hard disk defragmentation. Afterwards, the deviation of several key figures (number of commands, data bus utilization, bank open times) from the arithmetic mean values of all benchmark runs were determined. The distribution of the accesses across the physical address space was compared between the consecutive benchmark runs. Finally, the CA-bus utilization for the different SDRAM timings of all benchmark repetitions was evaluated as in chapter 5.1.1.

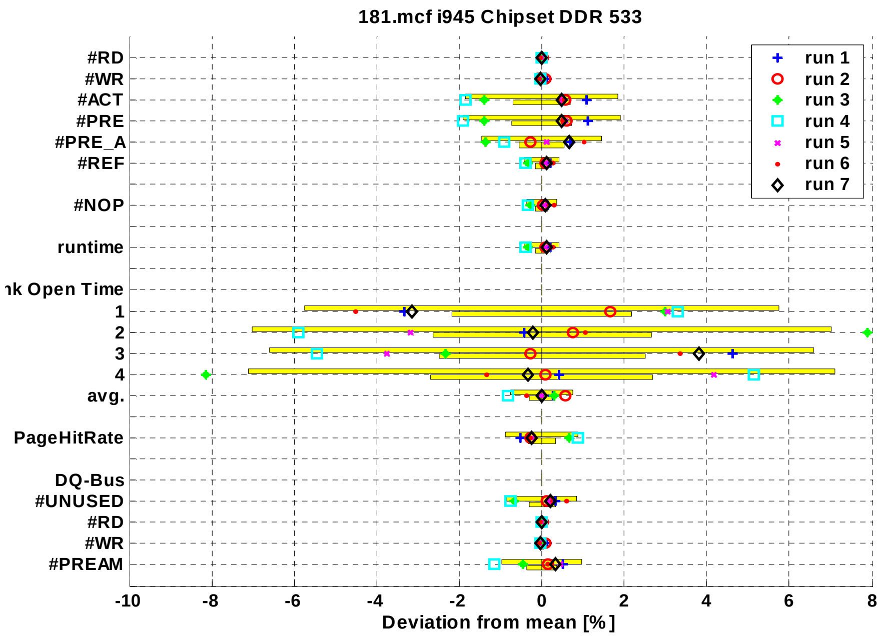


Figure 6.2: Deviation of Key Figures from Mean Value: 181.mcf

6.1 Deviation of Key Figures

In figures 6.1, 6.2, 6.3, and 6.4 the deviation of several key figures from their mean value is plotted for four memory intensive benchmarks of the SPEC 2000 benchmark suite (cluster 4 and 5 in [3]). The list of key figures consists of the number of issued commands, the benchmarks overall runtime, the bank open time for all four banks and their average, the page hit rate, and the data bus utilization.

For all key figures it is assumed that they follow a normal distribution. From the measurements the confidence interval for the mean value to a confidence level of 90% was calculated and is shown in the background of the graph (lower yellow bar). Furthermore, the confidence interval for a single benchmark run also calculated for a confidence level of 90% is shown (upper yellow bar).

For the four benchmarks the numbers of read, write, activation, and precharge commands are well within a band of $\pm 6\%$ around the arithmetic mean values. For 189.lucas and 181.mcf the deviation is even less than $\pm 2\%$. For the 171.swim and 183.equake benchmarks the number of precharge all commands may vary up to 30% from their average for some few benchmark runs. At first glance the random assignment of physical memory leads to different memory maps for the benchmarks, which makes closing more than one bank at the same time look more

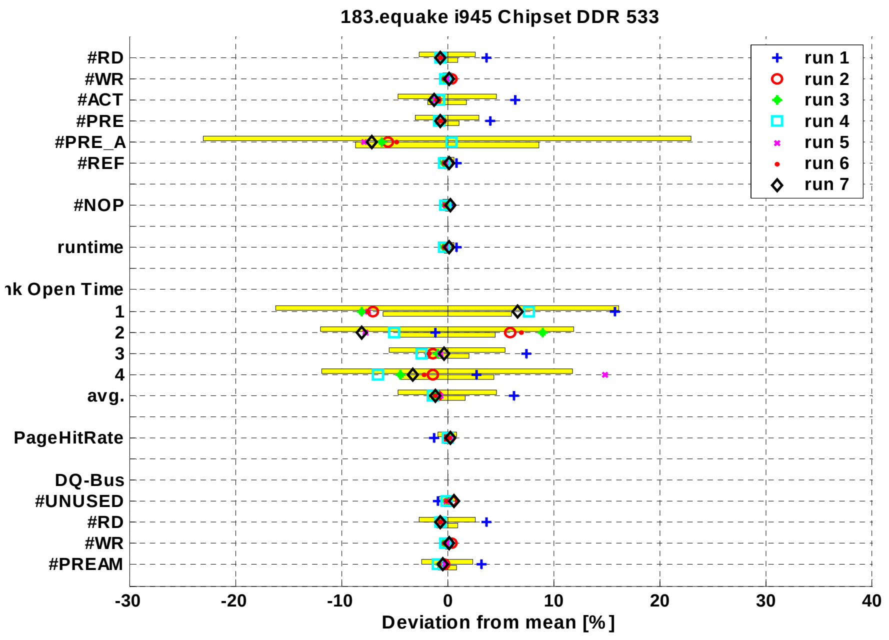


Figure 6.3: Deviation of Key Figures from Mean Value: 183.equake

beneficial to the memory controller for some benchmark runs. Nevertheless, this result shouldn't be overemphasized. Firstly, the number of precharge operations going to one single bank only is one order of magnitude larger than the number of precharge all commands. Therefore, small changes of the total number of precharge all commands lead to a large relative deviation from the average while the impact on system performance is minimal. Secondly, at least 40% of all precharge all commands close only one single bank (see figure 6.5). For three out of four benchmarks under investigation this number is even above 75%. So most of the precharge all operations could have been safely replaced by precharge commands going to one bank only.

The deviation of the DQ-bus utilization is in the same range as the CA-bus utilization. As every read and write transaction uses the data bus for transferring a single burst, an equal number of read and write requests per time also leads to an equal DQ-bus utilization.

By contrast, the deviation of the bank open times of the four benchmarks from the average is larger and is in the range of up to ±8%. Due to the random mapping of virtual memory to physical pages the bank open times may vary between different benchmark runs significantly.

An example can be found in figure 6.2. While in run 2 the banks are used

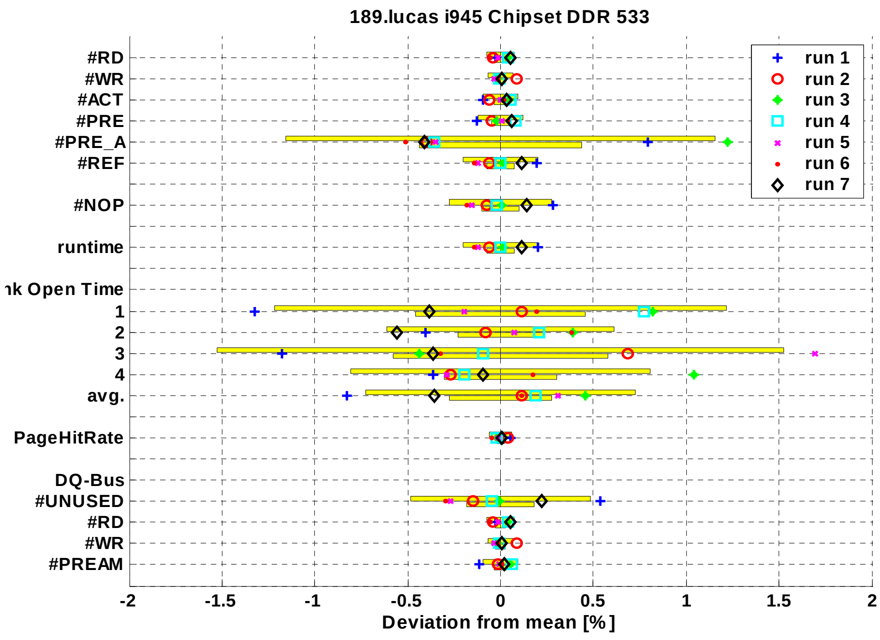


Figure 6.4: Deviation of Key Figures from Mean Value: 189.lucas

equally, benchmark runs 3 and 4 keep mostly two banks open.

On the other hand, the measured deviation of the average bank open times over all four banks is surprisingly low. This shows that despite the unawareness of the memory allocation algorithm of the hardware topology, the mapping of physical addresses to SDRAM ranks, banks, rows, and columns by the memory controller (address split) does a good job in distributing accesses equally across the four banks.

An interesting effect can be found for the 181.mcf benchmark in figure 6.2. The benchmark run 3 uses only two banks, while during run 2 SDRAM accesses are distributed equally over all banks. Although the lower number of used banks during run 3 should lead to more conflict misses (the row has to be closed in order to access another) than on runs using all four banks equally, the opposite can be observed. The number of activate and precharge commands is lower for run 2 and, thus, the page hit rate is higher. The higher locality of accesses leads to a decrease of the overall runtime and increases the memory systems performance.

The reason for this surprising result is that the page size of a virtual memory page of the x86 CPU is 4 KiB while a single row of the used SDRAM module spans 8 KiB. Therefore, one single SDRAM row contains two virtual memory pages. In the runs with a lower number of used banks the operating system allocated con-

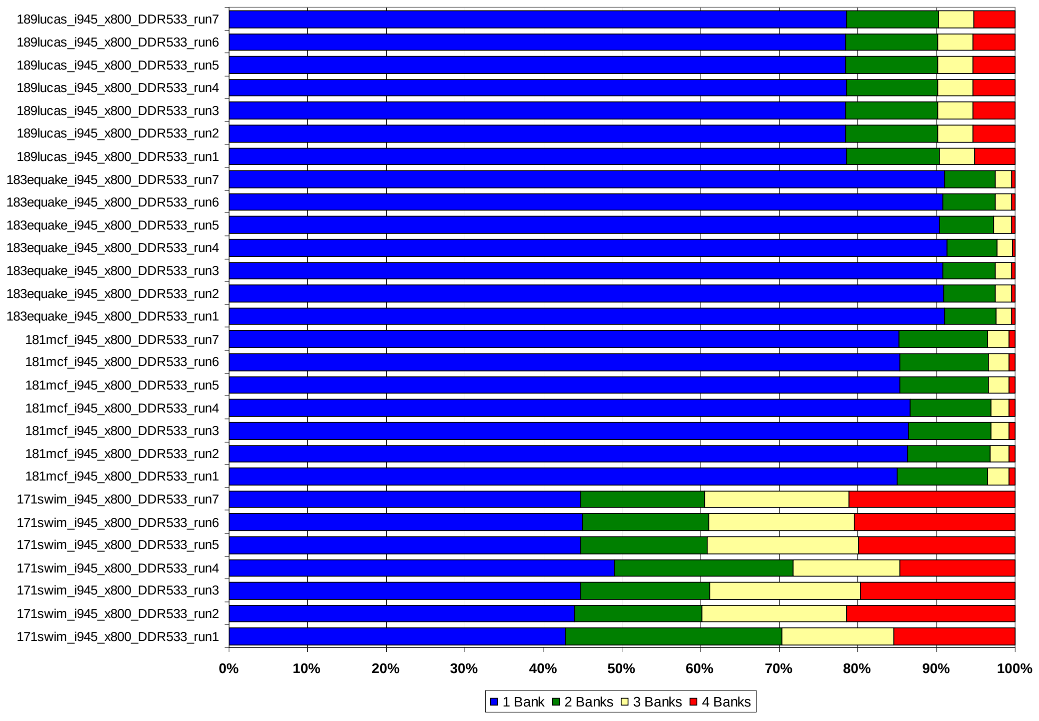


Figure 6.5: Number of closed Banks per precharge all Command

secutive virtual memory pages to the same row, increasing the locality of SDRAM accesses.

This result shows that in memory systems the size of a virtual memory page should be less or equal to the SDRAM row size of the SDRAM module. Furthermore, larger virtual pages may exploit access locality further.

6.2 Local Distribution of Accesses

In order to determine the reproducibility and, thus, the relevance of the results, not only the deviation of key figures has to be considered, but also the local distribution of accesses across the memory array. Furthermore, the distribution of the physical memory along SDRAM rows and banks has an impact on the system performance. From the performance point of view, the memory controller benefits from long SDRAM rows and a large number of banks. Both properties of a SDRAM system ensure that the memory controller may keep large chunks of memory available for direct access by the CPU wihout having to issue precharge/activate cycles for switching between different SDRAM rows.

As memory allocation algorithms of current operating systems like Microsoft Windows or Linux typically don't take the assignment of physical addresses to SDRAM ranks and banks into account, there is no explicit mechanism implemented in the operating system to distribute memory accesses to different banks equally in order to exploit temporal interleaving of SDRAM accesses [25, 46].

The system under investigation used one single rank 512 MiB SDRAM module consisting of eight 512 Mibit devices. The eight devices cover 8 KiB of physical address space per row while the page size of the Intel x86 CPU is 4 KiB. Thus, one row of SDRAM memory covers two virtual memory pages[2].

In a first step the number of virtual memory pages used by the benchmark runs was compared. The number of read or write accesses (burst transfers from or to memory) going to memory locations belonging to one single virtual memory page were counted. The resulting list was sorted by decreasing frequency of occurence. The left side of figures 6.6 to 6.9 show the distribution of accesses to virtual memory pages for all benchmark runs. For most benchmark runs the distribution of accesses across virtual memory pages is almost equal for all benchmark runs.

Opposed to this, the number of accesses memory rows may vary significantly between consecutive runs (shown on the right side of each figure).

Thus, there are benchmark runs during which the operating system assigns mostly only one single virtual memory page to a row and there are other runs, during which two pages are mapped to a single row by the operating system.

[2] In general one single memory row may cover two or more virtual memory pages, depending on the CPU and primary memory system.

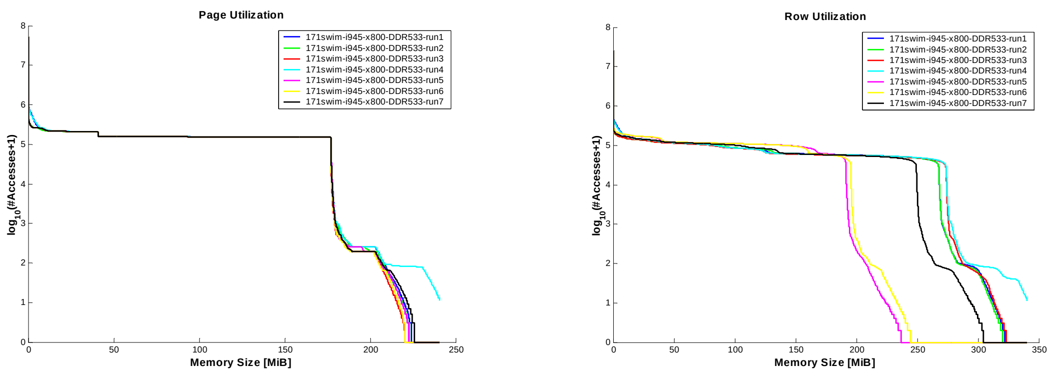


Figure 6.6: Page and Row Utilization of 171.swim

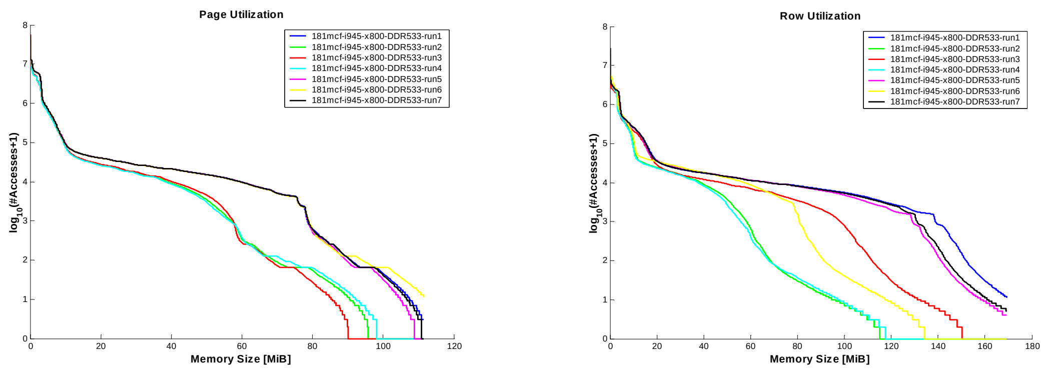


Figure 6.7: Page and Row Utilization of 181.mcf

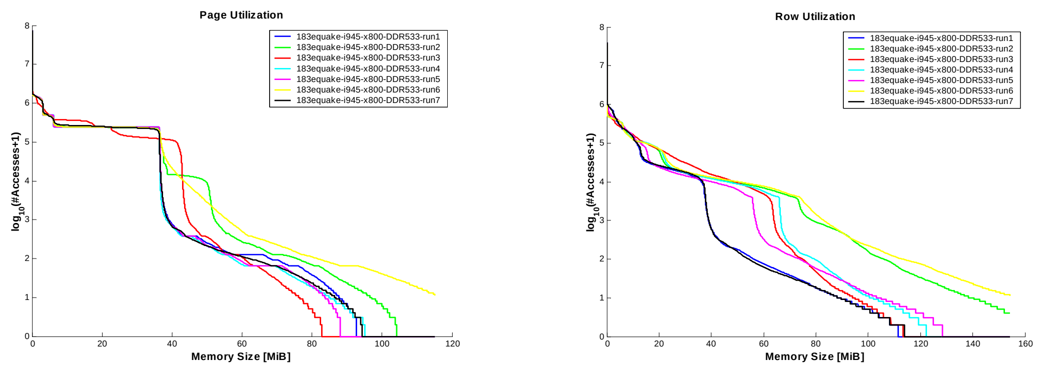


Figure 6.8: Page and Row Utilization of 183.equake

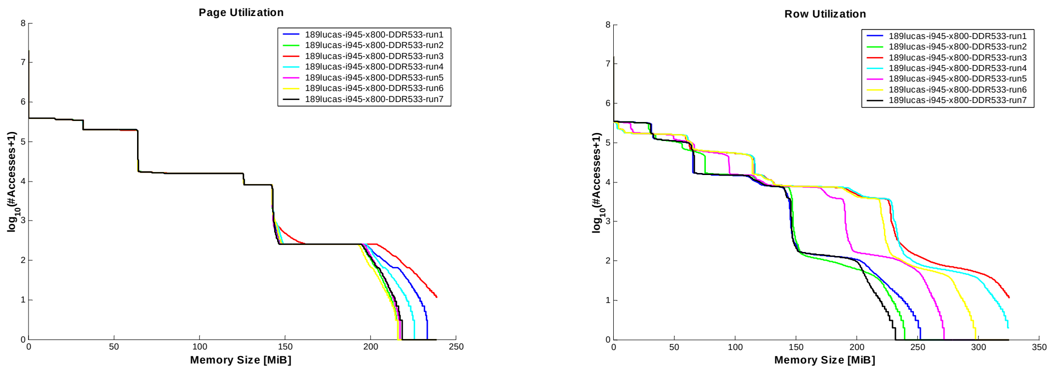


Figure 6.9: Page and Row Utilization of 189.lucas

It shall be noted, that it is assumed that each virtual memory page is mapped to a unique location in physical memory for the entire benchmark run. In effect the operating system may map multiple different virtual memory pages to the same physical memory location during the benchmarks runtime. This can happen, either because the memory utilization is not static during the benchmark run (i.e. the benchmark dynamically allocates and frees memory), or because the amount of installed physical memory is too small to hold all used virtual memory pages (*working set*) and, thus, forces the operating system to evict less used memory pages to secondary storage (disk swapping). While the first is hard to control and even to detect with the used measurement method, the latter can be controlled by installing sufficently large amounts of physical memory, so that the operating system does not have to swap memory contents to secondary storage. Figures 6.6 to 6.9 show that all benchmarks of the used SPEC benchmark suite use at most half of the installed physical memory. Thus, the swapping activity of the operating system is minimal.

Increasing the packing density of the used memory across fewer rows may reduce the number of activate/precharge cycles. In order to benefit from the higher packing density, the concept of spatial access locality must apply. So it is necessary that the two virtual memory pages mapped to a single row are to some extent correlated to each other (e.g. consecutive virtual memory pages in the processor's address space are mapped to a single row). In order to determine the correlation between the memory pages mapped to one single row, the read/write accesses going to one single virtual memory page were classified into groups and the numbers of their occurences were counted for all virtual memory pages:

1. The access is the first access going to the memory page. It required a row activation before (miss).

2. The access is the first access going to the memory page, but the row was already activated before due to an access to another virtual memory page mapped to that row.

3. The access is not the first on the memory page and, thus, the row was already activated before (hit).

Classifying the memory accesses for each memory page allows to determine the degree of correlation between memory pages mapped to a single SDRAM row. By looking at the first two types of accesses, three different classes of accessed memory pages can be determined:

1. There are read/write accesses of type 2 on the page. Thus, there are also accesses on another page located on the same memory row and there is a correlation between the virtual memory pages which allows the memory system to reduce the number of activate commands.

2. There are accesses on more than one virtual memory page located on that row (in this case: there are also accesses on the other page located on the same row), but no accesses on the page are of type 2. There is no correlation between the two pages on that row, and therefore no benefit results from holding them on the same memory row.

3. There are accesses on the page, but not on any other page located on that row (i.e. only one out of the two pages on the SDRAM component are used on the system under investigation).

Figure 6.10 shows the assignment of the accessed memory pages to their respective classes with respect to the overall number of accessed memory pages. It can be seen that during each benchmark run, the assignment of virtual memory pages to physical memory is completely different. During some benchmark runs up to 37% of all virtual memory pages share a memory row with another row and there is correlation between the two pages (i.e. there are accesses to a virtual memory page, which benefit from the fact that the respective SDRAM row has been already opened due to an access to another page on the same row). Nevertheless, on average only 12% of all virtual memory pages share a SDRAM row with another page in a way that the number of activate requests can be reduced. By contrast, on average 53% of all virtual memory pages share their SDRAM row with pages, with which they are completly uncorrelated (i.e. there are no bank activations saved at all). On average 35% of all memory pages don't share a row with another memory page.

The results clearly show that as long as the operating system is unaware of the address split (i.e. the assignment of physical addresses to SDRAM banks and rows),

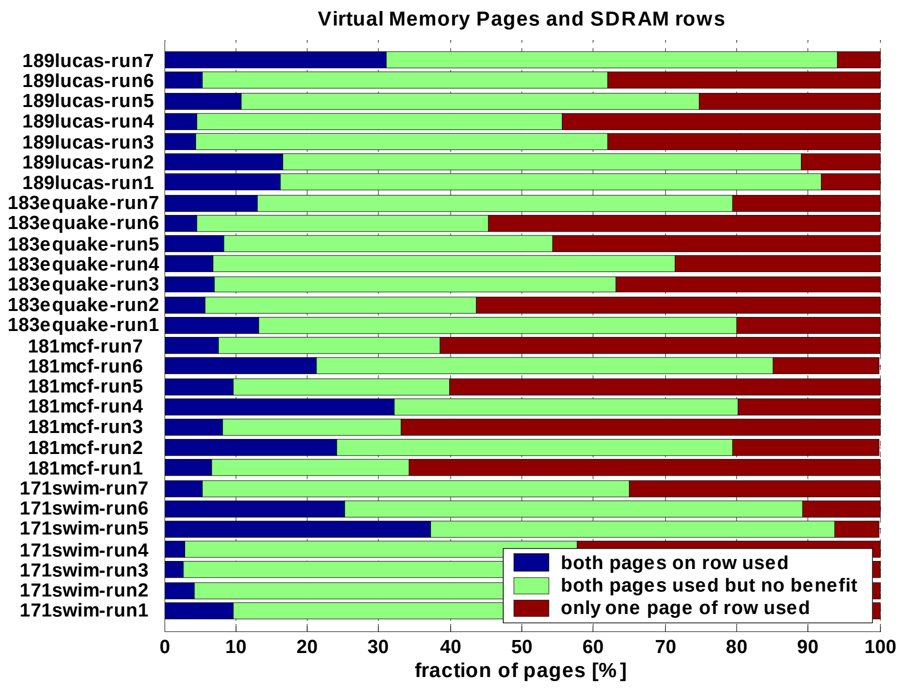


Figure 6.10: SDRAM Row Sharing of virtual Memory Pages

one cannot expect any significant benefit from placing multiple virtual memory pages on a single SDRAM row.

Designers of future memory systems should therefore make sure that either the operating systems takes the memory split into account when assigning virtual memory pages to physical memory, or the page size of the memory system should be made equal to the SDRAM row size.

Furthermore, figures 6.6 to 6.9 show that accesses to memory pages and rows are not distributed uniformly. The number of accesses going to different virtual memory pages may vary by several orders of magnitude. This result shows that even today caching mechanisms implemented in current microprocessors do not exploit locality to a large extent. In order to quantify this effect the Lorenz curve[3] is plotted for all benchmark runs. The curve shows the percentage of memory accesses as a function of the percentage of accessed virtual memory pages.

It is assumed that the number of burst transfers per virtual memory page is a random variable. Let $\vec{x} = (x_0, \ldots, x_n)$ denote the vector of the number of accesses going to a virtual memory page during a benchmark run with x_i being the number of read or write transfers going to the i^{th} least frequently accessed virtual memory page with $x > 0$. So the vectors elements are sorted in ascending order with respect to the number of accesses. In order to avoid a dependency of the Lorenz curve on the amount of available physical memory, only memory pages are taken into account which are accessed at least once during the execution of the benchmark. The Lorenz curve L is than given by the equation:

$$L\left(\frac{k}{n}\right) = \frac{\sum_{i=0}^{k} x_i}{\sum_{i=0}^{n} x_i} \tag{6.1}$$

Figure 6.11 shows that while for the 171.swim benchmark accesses are distributed equally, for the remaining benchmarks the distribution of accesses is extremely non uniform: During their execution 90% of all accesses take place on less than 30% of all virtual memory pages.

6.3 Reproducibility and SDRAM Timings

For the benchmarks of the SPEC CPU2000 suite the analysis of chapter 5.1.1 was performed on multiple runs of the same benchmark.

Figure 6.12 shows the fraction of time spent on particular SDRAM timings (as in figures 5.5 and 5.6) for consecutive runs of 171.swim. The variance between different runs of the same benchmark is less than 1% with respect to the overall benchmarks runtime.

[3] Max O. Lorenz, American economist. The Lorenz curve is typically used in economics to describe income inequalities.

The graphs for the remaining three benchmarks under investigation of the SPEC CPU2000 suite have been omitted here, as they are similar and thus do not provide additional insights.

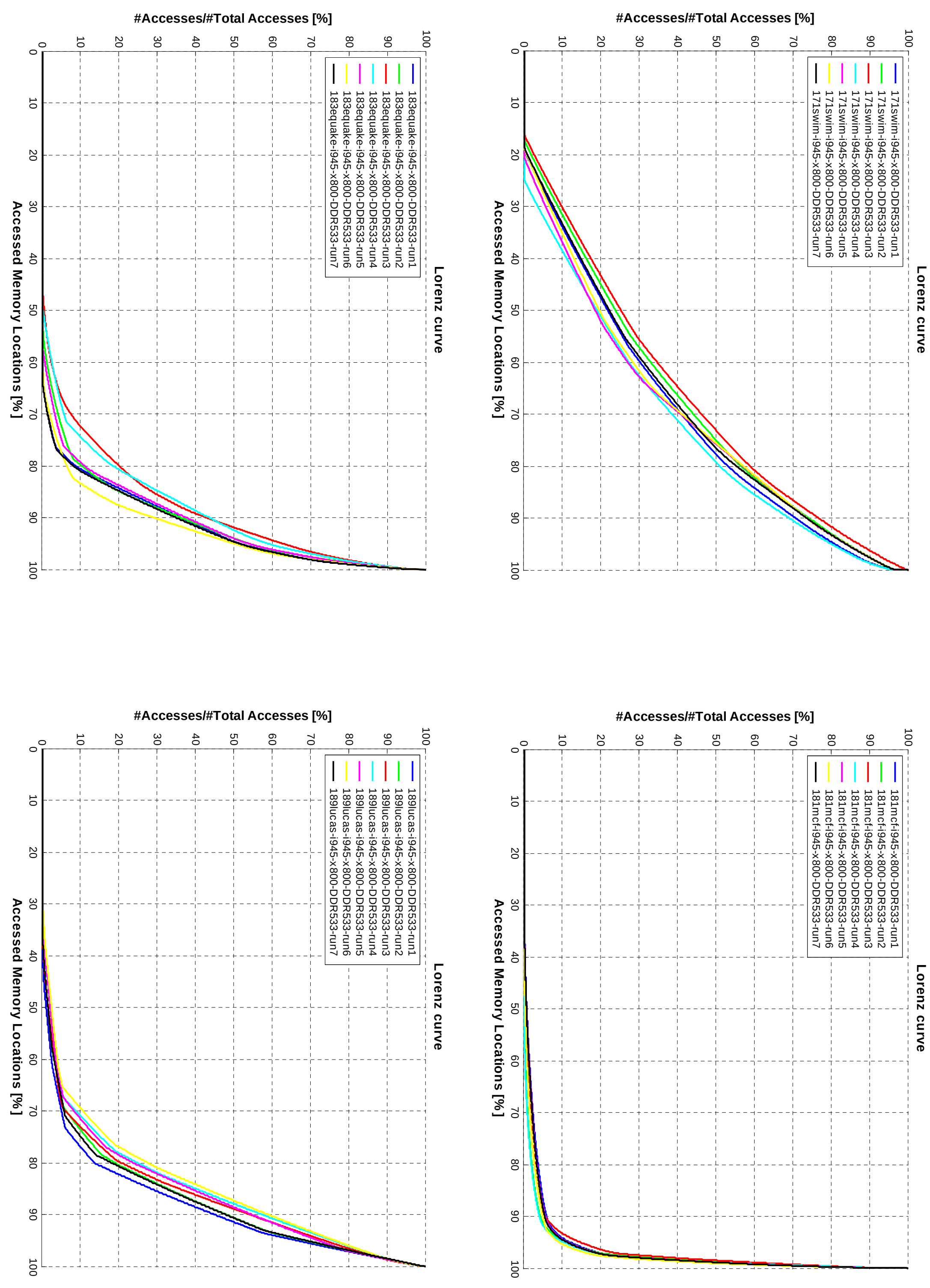


Figure 6.11: Lorenz Curves of selected SPEC 2000 Benchmarks

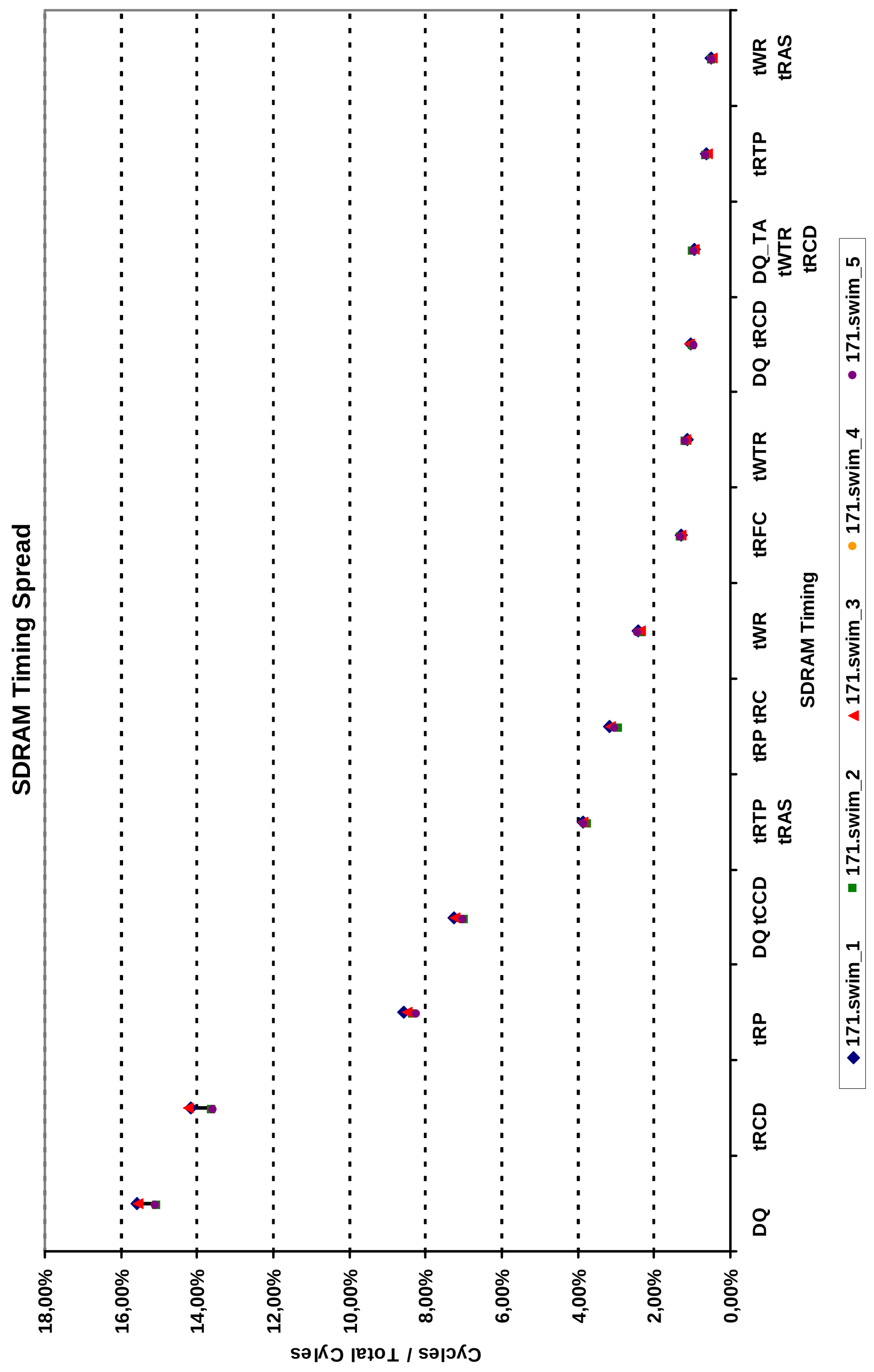


Figure 6.12: CA-Bus Utilization

Chapter 7

Conclusion

7.1 Summary

While in former times research on primary computer memory systems was mostly done using simulations, for this work a novel approach was chosen: Instead of simulating an entire computer system including CPU, memory controller, and IO devices, a measurement platform was developed. Using current FPGA and computer technology (24 hard disk drives, 6 PCs, 10 Gigabit Ethernet technology), the setup is capable to record access sequences of almost arbitrary length from a DDR2 SDRAM command and address bus on the fly without affecting program execution.

For the very first time it is possible to compare the results obtained by simulations with measurements on the real computer system.

The approach of measuring access sequences has several advantages over simulations. The most important is that no (over)simplified models of the computer system have to be used for evaluation, but evaluation of the system performance takes place on real hardware. Thus, the results obtained by measurements never suffer from unrealistic assumptions. Furthermore, it is possible to evaluate long sequences, which are much more representative, than doing only small spot checks, as it frequently has to be done in simulations due to excessive simulation times.

In order to evaluate the SDRAMs system performance, in this work new algorithms had to be developed to analyze the obtained memory access sequences. The performance of DDR2 SDRAM components is limited by a dozen SDRAM timing parameters, which have to be fulfilled between consecutive accesses. In this thesis a new method is introduced to account the time between consecutive SDRAM commands to the SDRAM timings which limit the system performance at that point in time. This approach allows a classification and prioritization of all SDRAM timings with decreasing significance regarding memory system performance. The analysis revealed, that from the bunch of SDRAM timing parameters which have to be fulfilled to operate the SDRAM within its operational range, only very few have significant impact on the system performance while the effect of most others on the system performance is negligible. The subset of relevant SDRAM timings contains only SDRAM timings which affect consecutive SDRAM commands (i.e. the SDRAM command limits the execution of its direct successor).

From the temporal distance of consecutive SDRAM commands statistical models were derived. These models allow the estimation of the memory system performance if either SDRAM timing parameters and/or the memory systems operating frequency is changed.

Estimating the performance impact of changed SDRAM timing parameters is vital for memory system designers, as it allows estimating the memory system performance of future memory systems which may not yet exist and which therefore cannot be subject of measurements.

SDRAM timing parameters may change as a side effect of the increasing capacity of SDRAM devices, changes of manufacturing technology, or may be intentionally applied by design changes. As most improvements of SDRAM timings are accompanied by increases of manufacturing costs, it is essential to estimate the impact of SDRAM timing changes on the memory systems performance prior to manufacturing.

The framework derived in the thesis provides memory system engineers with a convenient way to evaluate the increases in memory system performance of future SDRAM systems in advance. This knowledge may support their decision whether the estimated performance gain is large enough to justify an increase in manufacturing costs and/or can achieve a price premium on the SDRAM market.

Furthermore, the measurements on real personal computers delivered some unexpected results:

- Many publications claim that the memory system is the limiting factor for current and future computer systems performance due to the increasing gap between CPU and SDRAM operating frequency (e.g. [57, 31]). The measurements revealed that currently the memory system is not the limiting factor for system performance:

  The SPEC 2000 benchmark suite, which is frequently used for evaluation of computer system performance, consists of 26 benchmarks. Only a very small subset of these benchmarks exercises the memory system significantly. Four of them were selected to be worth analyzing the memory systems performance. The remaining benchmarks are still heavily CPU bound.

  The graphics benchmarks used for performance evaluation show similar results. If a graphics card is installed in the PC, most graphics rendering take place in the GPU (and the memory on the graphic adapter) while the main system memory is of minor importance for the overall system performance. In order to evaluate the memory system performance of graphic benchmarks, they had to be run on an inferior architecture with shared memory graphics (i.e. graphics rendering, displaying and computation is done entirely on the computers main memory).

Benchmarks which try to emulate typical office applications were completely unusable for memory system analysis, as the SDRAM utilization for this type of benchmark was far too low in order to identify the impact of SDRAM timing changes on the system performance.

- All benchmark runs were executed on a commercially available multitasking operating system, which manages the assignment of available physical memory to the different processes, switches between them, and handles interrupt requests from the IO systems. As the operator has very limited control over memory assignment and scheduling of processes, a large spread of performance related figures was expected when the same benchmark is run multiple times. Nevertheless, the memory system performance was still determined by the benchmark. The additional accesses to the SDRAM memory initiated by the multitasking operating system had only minor influence on the performance of the benchmark.

  This result has significant impact on system performance estimation. It shows that, if an appropriate workload is chosen for evaluation, memory system performance is determined by the used benchmark. Influences of the underlying operating system are of minor importance.

- Assigning multiple virtual memory pages to the same SDRAM row can be beneficial in terms of a reduction of time consuming activate and precharge operations. Nevertheless, as most commercial operating systems are unaware of the address split of the memory controller this effect is currently not systematically exploited.

7.2 Limitations

Although measuring access sequences on real computer systems has several advantages over simulations, there are also some drawbacks.

The most evident is that in order to measure access sequences, the hardware must already exist. Typically the performance shall not be estimated for computer systems which are already well-established, but for systems which will be built in the future. The estimation model uses data of current systems and extrapolates it to future memory systems. The accuracy of the model has been verified with measurements on similar memory systems which are currently available. However, significant architectural changes of the SDRAM memory system may render the model invalid. Fortunately in recent times SDRAM manufacturers focus more and more on some few highly standardized interface types, which are similar to each other while more exotic interfaces (e.g. RAMBUS) have disappeared from the market.

The framework introduced in the thesis strictly follows a SDRAM manufacturer's point of view. The model takes the access sequence of the CPU as given and evaluates the SDRAM manufacturer's options to improve system performance under these circumstances.

In reality not only the SDRAM manufacturer decides about improvements of the memory system. Many improvements of future memory system performance can be attributed to design changes of the memory controller (e.g. page policy, address split, power down policies, command scheduling, etc.). Unfortunately, as the memory traces are not taken from the frontside bus but from the memory bus, the view is limited to the results of the memory controllers job. The limited view makes it difficult to provide proposals for future memory controller designs. While SDRAM devices are commodity products and provide a standardized interface, in current computer systems north bridge and CPU are tightly coupled and the interface between them is proprietary to the CPU/chipset manufacturer. Capturing access sequences from the frontside bus would therefore limit the application of the measurement setup to some few processor designs.

The model provides only a very limited feedback path from the SDRAM memory to the CPU. Especially in the model the CAS latency (i.e. the time from issuing a read command to the final delivery of the data to the CPU) has no impact on the system performance. In reality, some of the accesses may be executed with non critical timing only due to the fact that required data requires significant time until it is delivered to the CPU for further processing. However, the estimation results show that the weak feedback path has only minor effect on the quality of the estimation.

7.3 Related Work

The measurement setup and the developed algorithms provide a novel and unique solution for capturing and analyzing long memory access sequences gathered on current computer systems.

In the early nineties, several attempts were undertaken to capture memory access sequences from the frontside bus of particular CPUs (see [52] for details) with additional hardware. None of the designs provided enough bandwidth for transferring full undisturbed memory traces of arbitrary length to secondary storage in realtime. The sequences were taken at the frontside bus as DRAM performance research was not popular: At that time DRAMs did not provide paging. Accordingly, accesses to the DRAM array took always the same amount of time, making memory controller research worthless. Most of the investigated CPUs are outdated today and have already been replaced by their successors.

Analysis of SDRAM access behavior is a relatively new discipline in computer architecture research and so far only very few publications have been released.

In [54] Wang presents a simulation framework for analyzing SDRAM access sequences based on "request access distances", which are to some extent comparable to the critical SDRAM timings of this thesis. In his work, he sticks to the assumption that the speed of the benchmarks execution is completely limited by the SDRAM memory system (infinitely fast CPU without any data dependencies). Starting from memory access sequences of a processor front side bus, he tries to execute the given access sequence as fast as possible on the given SDRAM memory while applying different memory controller scheduling strategies. Consequently, all memory accesses are performed critically.

Unlike this work, in which the memory controller is taken as given and the SDRAM memory is subject to optimizations (memory manufacturer's point of view), he takes the SDRAM memory as given while optimizing the memory controller side (memory controller manufacturer's point of view).

7.4 Extensions and Future Work

Power Estimation

While this thesis focuses on performance issues of current SDRAM technologies, power concerns become more and more vital to future computer systems. Three reasons can be pointed out:

1. The increase in operating frequency and integration density of CMOS devices increases their power consumption [16]. This leads to more heat dissipation within the devices and increases the effort of cooling the device.

   In addition, the increasing processing power due to the higher integration density facilitates more demanding applications which are handled by large server farms (e.g. web search engines, web servers, scientific computing, movie rendering, databases) [9, 21]. In these applications, heat dissipation is not just an adverse side effect, but becomes a serious issue.

2. Daily life is increasingly penetrated with mobile equipment. The number of mobile devices has increased significantly over the last years. Mobile phones

became a commodity product for almost anyone over the last years. As mobile equipment is powered from (rechargeable) batteries, low power consumption is mandatory to fulfill customers expectations of long runtimes of their gear.

3. Environmental protection concerns come to the focus of public perception. While the power consumption of a single SDRAM component might appear negligible at first glance, the vast number of devices integrated in electronic products sums up to a significant power consumption related to the use of SDRAM devices. In the future governmental regulations may enforce manufacturers of electronic equipment to keep the power consumption of their products below certain power limits.

While metrics for system performance exist as well as metrics of power consumption, new metrics have to be invented to specify the trade-off between system performance and power consumption [49].

Future memory controller will no longer only have to decide about the used page policy and whether to close opened SDRAM banks or not. New strategies have to be developed which enable the memory controller to find optimal points for entering and exiting power down states while minimizing the impact on the system performance.

In order to support power related research, new measurement probes are developed which shall cover measurements on mobile devices. As mobile devices are designed with an extremely high integration density (space is a major concern for mobile equipment), physical probing of the SDRAM signals at the SDRAM device becomes challenging.

Furthermore, there are different SDRAM variants on the market (e.g. mobile RAM) which use different SDRAM protocols compared to standard DDR2 components.

New SDRAM Type DDR3

In chapter 1.1 it was shown, that SDRAM product cycles are extremely short[1]. During the time this ph.d. thesis was created, the next SDRAM technology DDR3 was already in the development phase. The increased SDRAM frequency of DDR3 (see table 2.4.2) imposes new challenges on the development of measurement hardware regarding signal integrity and skew. Fortunately, modern FPGA technology tries to cope with the increase in operating frequency (e.g. latest FPGAs include technologies for single lane deskewing).

[1] The time from the specification of the DDR2 standard to the succeeding SDRAM technology DDR3 was around five years [32] [33].

Thus, a redesign of the measurement hardware is planed, using the latest FPGA technology, to enable capturing of DDR3 memory sequences. As SDRAM operation, most commands, and most SDRAM timings of DDR3 are comparable to DDR2, there will be a remarkable reuse of the measurement concept, the capturing hardware as well as the software for recording, processing, and analysis of SDRAM trace files.

Data Tracing

The measurement system cannot only be used for capturing command and address bus access sequences but could also be used for capturing data bus contents if this functionality is implemented in FPGA B. While data bus content may be of limited value for performance research, it can be useful for the verification of SDRAM devices. Read/write errors which occur only sporadically are hard to track down with traditional measurement equipment. Capturing the entire communication between the memory controller and the SDRAM device may help test engineers to track down communication errors between memory controller and SDRAM device more easily.

Reliability

SDRAM wordline drivers and address decoder have a limited lifetime (i.e. the number of read and write cycles is limited). As the lifetime of SDRAM devices is typically specified in terms of years and not in terms of the number of read/write cycles, the capturing of long access sequences may support the modelling of realistic utilization scenarios of the SDRAMs internal circuitry during memory operation in a computer system. This model may support SDRAM manufacturers to align their terms of warranty.

Workload Analysis

While single processor, single tasking execution of program code has been studied extensively, limited research has been conducted regarding the impact of the latest architectural changes like multiprocessing and multitasking environments. This is extremely important as latest personal computers are frequently delivered with multicore CPUs which are capable of running multiple tasks at the same time.

But architectural changes are not limited to the CPU side only. The increasing amount of memory implemented in new computer systems leads to more complex memory systems: Memory is no longer only distributed across multiple banks within one component but may be spread along multiple devices (SDRAM ranks) or may be even connected to separate SDRAM channels. Up to now, few research has been done to classify the goodness of memory distribution along different SDRAM banks, ranks, and channels.

A set of connector footprints has been connected to FPGA B. Instead of capturing the data bus contents, it would be possible to sample the command bus of up to three additional SDRAM ranks.

Benchmark classification

Research is ongoing to characterize access beahavior of different benchmarks. Often it is still unclear whether the benchmarks used to evaluate the performance of a computer system resemble the target application [49].

The tracing hardware provides the possibility to record access sequences not only of benchmarks but also of real (large scale) applications running on the computer system under test. Hence, it provides the opportunity to compare the memory access behavior of the application with its potential representative (benchmark).

Appendix A

DRAM Timings

A.1 Intra Bank Timings

Intra bank timings denote the minimum required temporal spacing between commands going to the same bank.

A.1.1 RAS to CAS delay (t_{RCD})

The RAS to CAS delay is the minimum time required between the activation command and the first read or write operation on that bank. During this time the transistors of one single wordline connect the capacitors to the bitlines. Charge is flowing from or to the capacitor plates, increasing or decreasing the voltage of the bitlines. The voltage change of the bitlines is then evaluated by the sense amplifiers connected to the bitlines.

A.1.2 CAS Latency (CL)

The CAS latency is the time from the read command to the output of the first data word at the DQ bus. This time is required to select the desired bitlines and to propagate the selected data to the secondary sense amplifier and the IO pins of the SDRAM device.

The CAS latency also denotes the time from issuing a write command until the memory controller may place the respective data on the data bus. For DDR2 memory devices the minimum CAS write latency is one clock cycle smaller than the CAS read latency.

Source	**Destination on Same Bank**					
	ACT	**RD**	**WR**	**PRE**	**PALL**	**REF**
ACT	t_{RC}	t_{RCD}	t_{RCD}	t_{RAS}	t_{RAS}	t_{RC}
RD		t_{CCD}	t_{CCD}	t_{RTP}	t_{RTP}	
WR		t_{WTR}	t_{CCD}	t_{WR}	t_{WR}	
PRE	t_{RP}					t_{RP}
PALL	t_{RP}					t_{RP}
REF	t_{RFC}					t_{RFC}
Source	**Destination on Different Bank**					
	ACT	**RD**	**WR**	**PRE**	**PALL**	**REF**
ACT	t_{RRD} t_{FAW}				t_{RAS}	t_{RC}
RD		t_{CCD}	t_{CCD}		t_{RTP}	
WR		t_{CCD}	t_{CCD}		t_{WR}	
PRE						t_{RP}
PALL	t_{RP}					t_{RP}
REF	t_{RFC}					t_{RFC}

Table A.1: Timings to be fulfilled between Commands [8, p.24]

A.1.3 Row Active Strobe (t_{RAS})

The row active strobe is the minimum time a bank has to be kept open after activation. t_{RAS} is required to allow the sense amplifier to refresh the SDRAMs cells content after activation.

A.1.4 Row Precharge Time (t_{RP})

The row precharge time is the minimum time required to perform the precharge operation. t_{RP} denotes the time from the precharge command to the next activation or refresh operation on the SDRAM bank. During the t_{RP} time interval the capacitance of the bitlines is charged to an intermediate voltage level.

A.1.5 Row Cycle Time (t_{RC})

The row cycle time is the minimum temporal spacing of two bank activations of the same bank. It is the sum of t_{RAS} and t_{RP}.

A.1.6 Write to Read Delay (t_{WTR})

The write to read delay is the minimum time from a write command (to be precise: from the end of the associated write data burst) to the next read command. t_{WTR} is the time required to transfer the data of one single prefetch of write data from the input buffer to the sense amplifiers in the array [32, p.31].

A.1.7 Write Recovery Time (t_{WR})

The write recovery time is the minimum time from writing to a bank to its precharge operation. It is calculated from the end of the data burst of the write command.

A.1.8 Refresh Cycle Time (t_{RFC})

The refresh cycle time is the time required for a single row refresh on all SDRAM banks. The next refresh or activation command must not be issued to an affected bank until t_{RFC} has elapsed.

A.2 Inter Bank Timings

Inter bank timings denote the minimum required temporal spacing between commands going to different banks on the same SDRAM rank.

A.2.1 CAS to CAS Delay (t_{CCD})

The CAS to CAS delay is the minimum time between two read or write commands going to a single SDRAM rank.

A.2.2 RAS to RAS Delay (t_{RRD})

The RAS to RAS delay is the minimum time between two activate commands going to the same SDRAM rank. As activation commands impose the largest energy requirements of all SDRAM commands, the t_{RRD} limitation is required in order to allow internal power generators to recover from the previous activate command.

A.2.3 Four Activate Window (t_{FAW})

Due to current consumption constraints, on DRAM devices having eight or more banks no more than four activate commands must be issued within a floating time window of t_{FAW}.

Appendix B

Configurations

B.1 PC System Configuration

Configuration	**A**	**B**
CPU	Intel P4	Intel P4
CPU Frequency [MHz]	2800	2800
Chipset	Intel 945G	Intel 945G
Graphics Card	ATI x800	internal
Operating System	WinXP Prof. SP2	WinXP Prof. SP2
Module Size	512 MiB	512 MiB
Nr. of Ranks	1	1
Burst Length	8	8

B.2 SDRAM Configuration

SpeedGrade		a	b	c	d
SDRAM	[Transfers per sec.]	400	533	667	533
CL	[Cycles]	3	4	4	6
t_{RCD}	[Cycles]	3	4	4	6
t_{RAS}	[Cycles]	3	4	4	4
r_{RC}	[Cycles]	11	16	19	16
t_{RP}	[Cycles]	3	4	4	6
t_{WR}	[Cycles]	3	4	5	6
t_{WTR}	[Cycles]	2	2	3	4
t_{RTP}	[Cycles]	2	2	3	2
t_{RFC}	[Cycles]	21	28	35	28
t_{RRD}	[Cycles]	2	2	3	2
t_{CCD}	[Cycles]	2	2	2	2
t_{FAW}	[Cycles]	8	10	13	10

B.3 Benchmarks

B.3.1 3D–Benchmarks

- **Aquamark 3**

  Aquamark is a 3D graphics benchmark which was developed by Massive Development in the year 2003. It is based on the game engine used in the computer games Aquanox 1 and Aquanox 2. The benchmark renders several scenes on screen using features of DirectX 7, 8 and 9. It renders a fixed number of images. Thus, the benchmarks runtime depends on the system configuration (frame-based rendering).

- **3D-Mark03**

  3D-Mark03 is a collection of 3D benchmarks developed by Futuremark Corp. in 2003. These include a set of four game tests. The benchmark also includes a set of CPU, feature, image quality, and sound tests [18].

  For the analysis in this thesis only the four game tests were used:

  - **Wings of Fury**

    Wings of Fury represents a flight simulator type of game. It renders several scenes using DirectX 7 having low requirements regarding the used graphics hardware.

- **Battle of Proxycon**

  Battle of Proxycon represents a first person shooter game (FPS). It renders several scene using DirectX 8 (medium requirements on graphics hardware).

- **Trolls Lair**

  Trolls Lair represents a typical role playing game (RPG). It renders several scene using DirectX 8 (medium requirements on graphics hardware).

- **Mother Nature**

  Mother Nature uses DirectX 9 technology for rendering a fotorealistic animation of a natural landscape (high requirements on graphics hardware).

All benchmarks use time-based rendering. This means that the benchmarks total runtime is equal on all systems. The computer systems performance determines how many frames a rendered per second. Thus, a faster system leads to a smoother animation and results in a higher 3D Mark score which is calculated as a weighted average of the sub tests frame rates.

- **Codecreatures Benchmark Pro**

  Codecreatures Benchmark Pro is a 3D graphics rendering benchmark developed by Codecreatures[1] in 2002. Codecreatures developed a 3D games development system. The benchmark constitutes a sample application of their system. It renders several fotorealistic scenes on the computer display consisting of hills, trees, clouds, water (including light reflections of the surroundings on the surface), and birds. Contrary to most other 3D benchmarks, Codecreatures Benchmark Pro draws a fixed number of frames on the screen, so that the system performance determines the overall runtime of the benchmark.

B.3.2 SPEC 2000 Suite

- **171.swim**

  This benchmark is a simulator for weather prediction. The used model is based on the paper, "The Dynamics of Finite-Difference Models of the Shallow-Water Equations", by Robert Sadourny, J. ATM. SCIENCES, VOL 32, NO 4, APRIL 1975.

[1]Today the company Codecreatures is part of H2Labs Creative Research GmbH.

- **179.art**

  Art is a neural network algorithm for recognizing objects in a thermal image. On the first stage the network is trained with two objects (an airplane and a helicopter). Afterwards the objects are searched in an image.

- **181.mcf**

  The benchmark is derived from a program which is used for single-depot vehicle scheduling in public mass transportation. This benchmark is the only benchmark of the SPEC suite using mostly integer arithmetics which exercises the SDRAM memory system significantly.

- **183.equake**

  Equake simulates the propagation of elastic waves on an unstructured mesh using a finite element method.

- **189.lucas**

  Performs the Lucas-Lehmer[2] test to check primality of Mersenne numbers $2^p - 1$, using arbitrary-precision (array-integer) arithmetic.

[2]François Édouard Anatole Lucas, French mathematican, 1842–1891; Derrick Henry "Dick" Lehmer, American mathematician, 1905–1991.

Appendix C

Calculations

In this section it is proven, that equation 5.8 is equal to equation 5.3 for $t^E_{CK} = t^O_{CK}$. If $t^E_k \geq t^O_k$ the results are different.

Below some abbreviations are used: τ^E and τ^O denote the critical analog timings when the second command of a particular command pair can be executed for the first time in the original system and in the sytem for which the estimation is done. This value is equal to the analog SDRAM timing rounded up to the next clock cycle.

$$\begin{aligned} t^O_{CK} &= t^E_{CK} = t_{CK} \\ \tau^E_k &= \left\lceil \frac{t^E_k}{t_{CK}} \right\rceil \cdot t_{CK} \\ \tau^O_k &= \left\lceil \frac{t^O_k}{t_{CK}} \right\rceil \cdot t_{CK} \end{aligned}$$

For $\tau^E \leq \tau^O$:

$$
\begin{aligned}
T_k^E &= \sum_{i=1}^{\infty} p_k^E(i) \cdot t_{CK} \cdot i \cdot H_k \\
&= p_k^E\left(\frac{\tau_k^E}{t_{CK}}\right) \cdot \tau_k^E \cdot H_k + \sum_{i=\frac{\tau_k^E}{t_{CK}}+1}^{\infty} p_k^E(i) \cdot i \cdot t_{CK} \cdot H_k \\
&= \int_0^{\tau_k^E} \frac{\rho_k^C\left(\frac{\tau_k^O}{t_{CK}}\right)}{\tau_k^O} dt \cdot \tau_k^E \cdot H_k + \sum_{i=\frac{\tau_k^E}{t_{CK}}+1}^{\infty} \int_{(i-1)\cdot t_{CK}}^{i \cdot t_{CK}} \rho_k^C(t) dt \cdot i \cdot t_{CK} \cdot H_k \\
&= \left[p_k^O\left(\frac{\tau_k^O}{t_{CK}}\right) \frac{\tau_k^E}{\tau_k^O} \cdot \tau_k^E + \sum_{i=\frac{\tau_k^E}{t_{CK}}+1}^{\frac{\tau_k^O}{t_{CK}}} p_k^O\left(\frac{\tau_k^O}{t_{CK}}\right) \cdot \frac{t_{CK}}{\tau_k^O} \cdot i \cdot t_{CK} + \sum_{i=\frac{\tau_k^O}{t_{CK}}+1}^{\infty} p_k^O(i) \cdot i \cdot t_{CK} \right] \cdot H_k \\
&= h_k^O\left(\left\lceil \frac{t_k^O}{t_{CK}} \right\rceil\right) \frac{\tau_k^E}{\tau_k^O} \cdot \tau_k^E + h_k^O\left(\left\lceil \frac{t_k^O}{t_{CK}} \right\rceil\right) \cdot \frac{t_{CK}}{\tau_k^O} \cdot t_{CK} \cdot \sum_{i=\frac{\tau_k^E}{t_{CK}}+1}^{\frac{\tau_k^O}{t_{CK}}} i + \sum_{i=\frac{\tau_k^O}{t_{CK}}+1}^{\infty} h_k^O(i) \cdot i \cdot t_{CK} \\
&= h_k^O\left(\left\lceil \frac{t_k^O}{t_{CK}} \right\rceil\right) \frac{\tau_k^E}{\tau_k^O} \cdot \tau_k^E + h_k^O\left(\left\lceil \frac{t_k^O}{t_{CK}} \right\rceil\right) \cdot \frac{t_{CK}^2}{\tau_k^O} \cdot \frac{1}{2} \left(\frac{\tau_k^{O\,2}}{t_{CK}^2} + \frac{\tau_k^O}{t_{CK}} - \frac{\tau_k^{E\,2}}{t_{CK}^2} - \frac{\tau_k^E}{t_{CK}} \right) + \\
&\quad \sum_{i=\left\lceil \frac{t_k^O}{t_{CK}} \right\rceil+1}^{\infty} h_k^O(i) \cdot i \cdot t_{CK} \\
&= h_k^O\left(\left\lceil \frac{t_k^O}{t_{CK}} \right\rceil\right) \frac{\tau_k^E}{\tau_k^O} \cdot \left\lceil \frac{t_k^E}{t_{CK}} \right\rceil \cdot t_{CK} + \frac{1}{2} h_k^O\left(\left\lceil \frac{t_k^O}{t_{CK}} \right\rceil\right) \cdot \left[\tau_k^O - \frac{\tau_k^{E\,2}}{\tau_k^O} + t_{CK} - \frac{\tau_k^E}{\tau_k^O} t_{CK} \right] + \\
&\quad \sum_{i=\left\lceil \frac{t_k^O}{t_{CK}} \right\rceil+1}^{\infty} h_k^O(i) \cdot i \cdot t_{CK} \\
&= h_k^O\left(\left\lceil \frac{t_k^O}{t_{CK}} \right\rceil\right) \frac{\tau_k^E}{\tau_k^O} \cdot \left\lceil \frac{t_k^E}{t_{CK}} \right\rceil \cdot t_{CK} + h_k^O\left(\left\lceil \frac{t_k^O}{t_{CK}} \right\rceil\right) \cdot \left(1 - \frac{\tau_k^E}{\tau_k^O}\right) \left(\frac{\tau_k^O + \tau_k^E + t_{CK}}{2} \right) + \\
&\quad \sum_{i=\left\lceil \frac{t_k^O}{t_{CK}} \right\rceil+1}^{\infty} h_k^O(i) \cdot i \cdot t_{CK}
\end{aligned}
$$

Opposed to equation 5.8 not all accesses which are executed with the critical timing in the original system are executed critically in the estimation as well, but only the fraction τ_k^E/τ_k^O. The remaining fraction of accesses $(1 - \tau_k^E/\tau_k^O)$ are executed later. This fraction of accesses is distributed equally across the remaining time between the new and the old critical timing. The time required to execute this remaining part, is the number of remaining accessess multiplied by the average of τ_k^O and $\tau_k^E + t_{CK}$.

Starting from equation 5.8 with $\tau^E > \tau^O$ it is

$$\begin{aligned}
T_k^E &= \sum_{i=1}^{\infty} p_k^E(i) \cdot t_{CK} \cdot i \cdot H_k \\
&= \int_0^{\tau_k^E} \rho_k^C(t)dt \cdot \tau_k^E \cdot H_k + \sum_{i=\frac{\tau_k^E}{t_{CK}}+1}^{\infty} \int_{(i-1)\cdot t_{CK}}^{i \cdot t_{CK}} \rho_k^C(t)dt \cdot i \cdot t_{CK} \cdot H_k \\
&= \left[\int_0^{\tau_k^O} \rho_k^C(t)dt + \int_{\tau_k^O}^{\tau_k^E} \rho_k^C(t)dt \right] \cdot \tau_k^E \cdot H_k + \sum_{i=\frac{\tau_k^E}{t_{CK}}+1}^{\infty} p_k^O(i) \cdot i \cdot t_{CK} \cdot H_k \\
&= \left[p_k^O\left(\frac{\tau^O}{t_{CK}}\right) + \sum_{i=\frac{\tau_k^O}{t_{CK}}+1}^{\frac{\tau_k^E}{t_{CK}}} p_k^O(i) \right] \cdot \tau_k^E \cdot H_k + \sum_{i=\frac{\tau_k^E}{t_{CK}}+1}^{\infty} p_k^O(i) \cdot i \cdot t_{CK} \cdot H_k \\
&= \sum_{i=\frac{\tau_k^O}{t_{CK}}}^{\frac{\tau_k^E}{t_{CK}}} p_k^O(i) \cdot \tau_k^E \cdot H_k + \sum_{i=\frac{\tau_k^E}{t_{CK}}+1}^{\infty} p_k^O(i) \cdot i \cdot t_{CK} \cdot H_k \\
&= \sum_{i=1}^{\frac{\tau_k^E}{t_{CK}}} p_k^O(i) \cdot \tau_k^E \cdot H_k + \sum_{i=\frac{\tau_k^E}{t_{CK}}+1}^{\infty} p_k^O(i) \cdot i \cdot t_{CK} \cdot H_k \\
&= \sum_{i=1}^{\frac{\tau_k^E}{t_{CK}}} h_k^O(i) \cdot \tau_k^E + \sum_{i=\frac{\tau_k^E}{t_{CK}}+1}^{\infty} h_k^O(i) \cdot i \cdot t_{CK} \\
&= \sum_{i=1}^{\left\lceil \frac{t_k^E}{t_{CK}} \right\rceil} h_k^O(i) \cdot \left\lceil \frac{t_k^E}{t_{CK}} \right\rceil \cdot t_{CK} + \sum_{i=\left\lceil \frac{t_k^E}{t_{CK}} \right\rceil+1}^{\infty} h_k^O(i) \cdot i \cdot t_{CK}
\end{aligned}$$

Bibliography

[1] Anant Agarwal, John Hennessy, and Mark Horowitz. Cache performance of operating system and multiprogramming workloads. *ACM Transactions on Computer Systems*, 6(4):393–431, 1988.

[2] Juha Alakarhu and Jarkko Niittylahti. DRAM simulator for design and analysis of digital systems. *Microprocessors and Microsystems*, 26(4):189–198, May 2002.

[3] Simon Albert, Sven Kalms, Christian Weiss, and Achim Schramm. Acquisition and evaluation of long DDR2-SDRAM access sequences. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software 2006*, pages 242–250, Austin, Texas, United States, March 2006.

[4] AMD Inc. *AMD Athlon 64, Product Data Sheet*, 3.07 edition, June 2004.

[5] Josep Torrellas anmd Chun Xia and Russell L. Daigle. Optimizing the instruction cache performance of the operating system. *IEEE Transactions on Computers*, 47(12):1363–1381, December 1998.

[6] Todd Austin, Eric Larson, and Dan Ernst. Simplescalar: An infrastructure for computer system modeling. *Computer*, 35(2):59–67, 2002.

[7] Art Baker. *Windows 2000 Device Driver Book.* Prentice Hall International, second edition, November 2000.

[8] Soumya Banerjee. Memory performance estimation by analysis of SDRAM system states. Master's thesis, Department of Computer Science, Technical University of Dresden, 2005.

[9] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23, 2003.

[10] Luiz André Barroso, Kourosh Gharachorloo, and Edouard Bugnion. Memory system characterization of commercial workloads. In *ISCA '98: Proceedings*

of the 25th annual international symposium on Computer architecture, pages 3–14, Washington, DC, USA, 1998. IEEE Computer Society.

[11] Georg Braun. DDR3 introduction and overview. Internal presentation, Infineon Technologies AG, 2004.

[12] Douglas Burger, James Goodman, and Alain Kägi. The declining effectiveness of dynamic caching for general-purpose microprocessors. In *Proceedings of the 23rd annual international symposium on Computer architecture*, pages 78–89, Philadelphia, Pennsylvania, United States, May 1996.

[13] Harold Cain, Kevin Lepak, Brandon Schwartz, and Mikko Lipasti. Precise and accurate processor simulation. In *Proceedings of the Fifth Workshop on Computer*, pages 13–22, Feb. 2002.

[14] John B. Carter, Wilson C. Hsieh, Leigh Stoller, Mark R. Swanson, Lixin Zhang, Erik Brunvand, Al Davis, Chen-Chi Kuo, Ravindra Kuramkote, Michael Parker, Lambert Schaelicke, and Terry Tateyama. Impulse: Building a smarter memory controller. In *HPCA '99: Proceedings of the 5th International Symposium on High Performance Computer Architecture*, pages 70–79, Washington, DC, USA, 1999. IEEE Computer Society.

[15] Jason P. Casmira, David P. Hunter, and David R. Kaeli. Tracing and characterization of windows nt-based system workloads. *Digital Technical Journal*, 10(1):6–21, 1998.

[16] Anantha P. Chandrakasan and Robert W. Brodersen. Minimizing power consumption in digital CMOS circuits. In *Proceedings of the IEEE*, volume 83/4, pages 498–523. IEEE Computer Society, April 1995.

[17] Rajagopalan Desikan, Doug Burger, Stephen Keckler, and Todd Austin. Simalpha: a validated execution driven alpha 21264 simulator. Technical Report TR-01-23, Department of Computer Sciences, University of Texas at Austin, 2001.

[18] Maneesh Dhagat. *3DMark03. Next generation 3D benchmarking.* Futuremark Corporation, Feburary 2003.

[19] Lieven Eeckhout, Hans Vandierendonck, and Koenraad De Bosschere. Workload design: Selecting representative program-input pairs. In *PACT '02: Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, pages 83–94, Washington, DC, USA, 2002. IEEE Computer Society.

[20] Ludwig Fahrmeir, Rita Kunstler, Iris Pigeot, and Gerhard Tutz. *Statistik.* Springer, Berlin, 5th edition, May 2004.

[21] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz André Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the ACM International Symposium on Computer Architecture, San Diego, CA*. IEEE Computer Society, June 2007.

[22] J. Kelly Flanagan, Brent E. Nelson, James K. Archibald, and Knut Grimsrud. Incomplete trace data and trace driven simulation. In *MASCOTS '93: Proceedings of the International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems*, pages 203–209. Society for Computer Simulation, 1993.

[23] J. Kelly Flanagan, Brent E. Nelson, James K. Archibald, and Greg Thompson. The inaccuracy of trace-driven simulation using incomplete multiprogramming trace data. In *MASCOTS '96: Proceedings of the 4th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, pages 37–43, Washington, DC, USA, 1996. IEEE Computer Society.

[24] Arijit Ghosh and Tony Givargis. Cache optimization for embedded processor cores: An analytical approach. *ACM Transactions on Design Automation of Electronic Systems*, 9(4):419–440, 2004.

[25] Mel Gorman. *Understanding the Linux Virtual Memory Manager.* Bruce Perens' open source series. Prentice Hall, 2004.

[26] John Hennessy and David Patterson. *Computer Architecture. A quantitative approach.* Morgan Kaufmann, 3 edition, 2003.

[27] Kurt Hoffman. *System Integration.* John Wiley & Sons, Ltd., 2004.

[28] Infineon Technologies AG. *DRAM Market Update*, October 2004.

[29] Infineon Technologies AG. *HYB18T1G400AF, HYB18T1G800AF, HYB18T1G160AF. 1-Gbit DDR2 SDRAM, Data Sheet*, 1.1 edition, March 2005.

[30] Infineon Technologies AG. *HYB18T512400AF, HYB18T512800AF, HYB18T512160AF. 512-Mbit DDR2 SDRAM, Data Sheet*, 1.3 edition, January 2005.

[31] Bruce Jacob. A case for studying DRAM issues at the system level. *IEEE Micro*, 23(4):44–56, 2003.

[32] JEDEC Solid State Technology Association. *DDR2 SDRAM Specification*, JESD79-2A edition, January 2002.

[33] JEDEC Solid State Technology Association. *DDR3 SDRAM Specification*, JESD79-3A edition, September 2007.

[34] Lizy Kurian John, Purnima Vasudevan, and Jyotsna Sabarinathan. Workload characterization: Motivation, goals and methodology. In *WWC '98: Proceedings of the Workload Characterization: Methodology and Case Studies*, page 3, Washington, DC, USA, 1998. IEEE Computer Society.

[35] Norman P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. *SIGARCH Computer Architecture News*, 18(3a):364–373, 1990.

[36] Bernd Klehn and Martin Brox. A comparison of current SDRAM types: SDR, DDR, and RDRAM. *Advances in Radio Science*, 1:265–271, 2003.

[37] L. I. Kontothanassis, R. A. Sugumar, G. J. Faanes, J. E. Smith, and M. L. Scott. Cache performance in vector supercomputers. In *Supercomputing '94: Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, pages 255–264, New York, NY, USA, 1994. ACM Press.

[38] Dennis C. Lee, Patrick J. Crowley, Jean-Loup Baer, Thomas E. Anderson, and Brian N. Bershad. Execution characteristics of desktop applications on Windows NT. In *ISCA '98: Proceedings of the 25th annual international symposium on Computer architecture*, pages 27–38, Washington, DC, USA, 1998. IEEE Computer Society.

[39] Karl M. J. Lofgren, Robert D. Norman, Gregory B. Thelin, and Anil Gupta. Wear leveling techniques for flash EEPROM systems, February 2005. United States Patent 6,850,443.

[40] Peter Magnusson and Bengt Werner. Efficient memory simulation in simics. In *SS '95: Proceedings of the 28th Annual Simulation Symposium*, page 62, Washington, DC, USA, 1995. IEEE Computer Society.

[41] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Computer Architecture News*, 33(4):92–99, 2005.

[42] Hans-Peter Messmer. *PC-Hardwarebuch: Aufbau Funktionsweise Programmierung: ein Handbuch nicht nur für Profis.* Addison-Wesley, 1992.

[43] Ulrich Meyer, Peter Sanders, and Jop Sibeyn. *Algorithms for Memory Hierarchies: Advanced Lectures*, volume 2625 of *Lecture Notes in Computer Science.* Springer Verlag, 2003.

[44] Scott Rixner, William J. Dally, Ujval J. Kapasi, Peter Mattson, and John D. Owens. Memory access scheduling. In *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, pages 128–138, New York, NY, USA, 2000. ACM Press.

[45] Mendel Rosenblum, Edouard Bugnion, Scott Devine, and Stephen A. Herrod. Using the simos machine simulator to study complex computer systems. *ACM Transactions on Modeling and Computer Simulation*, 7(1):78–103, 1997.

[46] Mark E. Russinovich and David A. Solomon. *Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server(TM) 2003, Windows XP, and Windows 2000 (Pro-Developer).* Microsoft Press, December 2004.

[47] Semiconductor Components Industries, LLC. *MC100EP195B: 3.3V ECL Programmable Delay Chip*, July 2006. Rev.0, MC100EP195B/D.

[48] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 45–57. ACM Press, 2002.

[49] Kevin Skadron, Margaret Martonosi, David I. August, Mark D. Hill, David J. Lilja, and Vijay S. Pai. Challenges in computer architecture evaluation. *IEEE Computer*, 36(8):30–36, 2003.

[50] Tektronix Inc. *Tektronix Logic Analyzers. Family Selection Guide*, August 2004.

[51] Niki C. Thornock and J. Kelly Flanagan. Using the BACH trace collection mechanism to characterize the SPEC 2000 integer benchmarks. In *Proceedings of the Third IEEE Annual Workshop on Workload Characterization*, pages 121–143, 2000.

[52] Richard A. Uhlig and Trevor N. Mudge. Trace-driven memory simulation: a survey. *ACM Computer Survey*, 29(2):128–170, 1997.

[53] David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Kathleen Baynes, Aamer Jaleel, and Bruce Jacob. DRAMsim: a memory system simulator. *SIGARCH Computer Architecture News*, 33(4):100–107, 2005.

[54] David T. Wang. *Moden DRAM Memory Systems: Performance Analysis and a high performance, power-constrained DRAM scheduling algorithm.* PhD thesis, University of Maryland, 2005.

[55] Myles G. Watson. Does the halting necessary for hardware trace collection inordinately perturb the results ? Master's thesis, Department of Computer Science, Brigham Young University, December 2004.

[56] Myles G. Watson and J. Kelly Flanagan. Does halting make trace collection inaccurate? a case study using Pentium 4 performance counters and SPEC 2000. In *Proceedings of the 7th IEEE Annual Workshop on Workload Characterization*, October 2004.

[57] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: implications of the obvious. *SIGARCH Computer Architecture News*, 23(1):20–24, 1995.

[58] Im Yon-Kyun, Yoon Chi-Weon, and Jung Tae-Sung. POPeye: a system analysis tool for DRAM performance measurement. In *ICVC99: Proceedings of the 6th International Conference on VLSI and CAD*, pages 590–592. IEEE Computer Society, October 1999.

[59] Lixin Zhang, Zhen Fang, Mide Parker, Binu K. Mathew, Lambert Schaelicke, John B. Carter, Wilson C. Hsieh, and Sally A. McKee. The impulse memory controller. *IEEE Trans. Comput.*, 50(11):1117–1132, 2001.