Euromicro'2000, Symposium on Digital Systems Design,
Maastricht, Netherlands, Sep. 2000, vol.1, pages 282-289, © by IEEE

The Impact of Recent DRAM Architectures on Embedded Systems Performance

Matthias Gries
Swiss Federal Institute of Technology Zurich
Computer Engineering and Networks Laboratory (TIK)
CH-8092 Zurich, Switzerland
email: gries@tik.ee.ethz.ch

Abstract

Embedded computing systems utilize less cache space and fewer memory hierarchy levels than ordinary PC or workstation systems due to costs, area, and power dissipation restrictions. Consequently, they particularly depend on the performance of the underlying DRAM main memory system. Hence, two recent DRAM architectures, widely-used SDRAMs and the next generation memory Direct RDRAM, are investigated in this paper. Performance gains are revealed that can be achieved by exploiting features of recent memory interfaces with simple enhancements of current embedded memory controllers. Different approaches for memory access schemes are investigated by simulation of the DRAM architectures and the memory controller together with an out-of-order issue, superscalar CPU model running various applications. The simulations lead to the following results: using RDRAMs instead of SDRAMs improves the performance of the system by up to one third. Exploiting the multibank structure of DRAMs improves the performance more than pipelining memory transfers.

1. Introduction

Memory chips have become the main performance bottle-neck of computing systems [3, 23, 5]. Chip manufacturers have tried to bypass the weak performance of the memory core by designing complex memory interfaces which isolate the behavior of the slow memory core from the fast interconnections between the memory chip and the memory controller. This has led to a variety of memory interface implementations which essentially are based on the same memory core technology and functionality.

Recently, most manufactures of personal computer technology have decided to switch from synchronous DRAM (SDRAM) to Direct Rambus DRAM (RDRAM). The RDRAM interface is not simply another step of SDRAM interface improvement, but needs a complete redesign of the main memory system. The Rambus approach for improved memory utilization is to use narrow buses, so-called channels, at very high clock rates. In traditional SDRAM based main memory systems, several chips are needed to fit the width of the memory bus. Thus, these chips cannot be controlled concurrently. A single RDRAM chip, however, spans the whole Rambus channel so that two RDRAM chips can already be controlled concurrently. Finally, the number of internal memory banks has been increased for RDRAMs.

The performance comparisons published by SPEC [21] and in [7] suggest that workstation class computers do not depend notably on the synchronous DRAM type used. The question now arises whether embedded computing systems will be able to exploit the new features introduced by RDRAMs. In this paper, we concentrate our investigations on configurations where cache sizes and the number of main memory DRAM chips are kept small. For this class of systems we compare SDRAMs and RDRAMs for a number of applications, ranging from CPU benchmarks to real-world programs. We also investigate the performance impact of different memory access schemes which can be implemented in custom-designed memory controllers.

1.1. Related work

Detailed cycle-true simulators of CPUs have been used for a variety of investigations in order to explore future processor systems together with their memory subsystem. These studies either focus on the influence of different CPU architecture features and cache designs on the memory behavior using standardized SPEC [21] CPU benchmarks [5, 8, 22] and particular applications [20, 22] or concentrate more on the workload generated for the memory subsystem by certain applications [1]. All these studies have in common that due to large second level caches the impact of the main memory is supposed to be low. Thus, the main memory is simulated by a simple model which only consists of two to three access delay parameters. Our study models the main memory system more precisely and emphasizes the role of the memory controller.

Enhancements of memory controllers such as stream buffers [14, 10, 16] and configurable complex address remapping functions [6] are rather complex and are thus unlikely to be introduced in embedded memory controllers in the near future. These concepts need adjustments in the compiler, may generate additional run-time overhead for reconfigurations, and may introduce additional delay for applications which do not show regular memory access patterns. We restrict our study to simple memory controller architectures and enhancements which can already be found in current workstation systems.

Recent RAM surveys [18, 15, 13] focus on the functionality, the architecture, and typical applications of dynamic RAMs. However, they do not provide performance measurements under realistic workloads. An analysis of different DRAM architectures has been combined with performance investigations by simulation of a complex CPU model by Cuppu et al. in [7]. However, their objective differs from ours in that they simulate a workstation class computer with larger caches, longer cache lines, more superscalar units, and a higher clock frequency than embedded systems usually have. Moreover, we are interested in the impact of the memory controller on the performance of the whole computing system. We restrict our studies to the two most recent DRAM types, but investigate the influence of address permutations and pipelined accesses performed by the memory controller.

The main contributions of this paper are:

- performance evaluation of memory controller features and DRAM types by
- simulation of a computing system adapted to the needs in embedded systems
- running various complex applications and benchmarks
- using accurate memory controller and DRAM models.

The results can thus be used to find a suitable trade-off between system performance and memory controller complexity in embedded systems design.

This paper is organized as follows: in the next section, the characteristics of SDRAMs and RDRAMs are pointed out and the functionality of a memory controller is briefly described. Section 3 then outlines the chosen CPU simulator and the extensions implemented in order to simulate a CPU model together with detailed main memory models. Section 4 presents the results from various simulation runs. Section 5 concludes the paper.

2. Functionality of a memory subsystem

2.1. Functionality of DRAMs

Recent DRAMs have pipelined synchronous interfaces which isolate the main memory cell arrays from the memory bus signals. Read and write accesses are usually performed in burst operation mode. An arbitrary access is a two-step process. The corresponding row of a memory array must be addressed before a particular column can be accessed. The contents of the chosen row are then held by the *sense-amplifiers*. Thus, the sense-amplifiers can be seen as a single row cache of the corresponding memory bank. Accesses to that particular row are fast and just need the column access time. Accessing another row however is slow, since the current row in the sense-amplifiers must always be precharged before another row can be addressed. Modern DRAMs consist of several internal memory banks, each with its own row of sense-amplifiers. In the context of DRAM organization, an internal memory array row is often called a memory *page*.

SDRAM. SDRAM is currently the most often used synchronous DRAM type. Especially, the RAM modules according to the PC 66/ PC 100 SDRAM specification [11] are popular. SDRAMs are available on standardized modules with fixed data bus width on which several memory chips are mounted. The most popular modules are DIMMs (Dual In-line Memory Module [12]) and offer a 64 bit wide memory bus.

RDRAM. RDRAM is a recent memory specification developed by Rambus Inc. The Rambus memory bus has a width of only 16 bit. The transmission of all kinds of information needs four clock cycles for completion. Since even control signals need several clock cycles for transmission, RDRAMs are often referred to as DRAMs which use packets for communication. RDRAMs have write buffers. The memory controller must take care of the data consistency between the write buffers and the sense amplifier contents. RDRAMs are also available on memory modules, so-called RIMMs specified by Rambus Inc. Several RIMMs are interconnected in a serial fashion.

2.2. Functionality of a memory controller

The memory controller is responsible for the translation of read and write requests of the CPU into control signals for the DRAM array. In addition, the controller must satisfy the timing requirements of the memory chip because the timing of control signals is not checked by the DRAM itself. The memory controller may stall the issue of new requests by the CPU until the DRAM is again capable of accepting read or write accesses. The format of the memory accesses issued by the CPU depends on the CPU architecture and is usually set at start-up time. The size of an access equals the size of a cache line so that a single burst transfer of the length four or eight is sufficient to exchange the contents of a cache line. Embedded CPUs often use a memory controller integrated into the CPU core. Personal computing systems rather use an external controller through a front-side bus for more flexibility in the system design accepting an additional delay of one to two bus clock cycles per access.

Memory controllers without additional operation queues are not able to reschedule read and write operation requests from the CPU. Most of the memory controllers currently found in PC and embedded systems belong to this class. However, the latency of a memory operation can be reduced by using heuristics which exploit the row-wise organization of DRAMs. For instance, leaving memory rows opened as long as possible in order to save some precharge and activate operations is called an *open-page* policy. In contrast to that, a *closed-page* policy precharges an active memory row as soon as possible. This method is employed if row or bank changes appear frequently. Moreover, the controller may buffer memory requests of the CPU in an in-order queue. Such a controller is able to exploit the pipelined memory interface of modern DRAMs by overlapped processing of successive accesses. Finally, the memory controller may implement simple schemes to map physical address ranges to other regions by address permutations for interleaving bank accesses.

3. Simulation environment

3.1. Simplescalar tool-set CPU simulator

We use SimpleScalar [4], version 3.0a, for cycle-true, execution-driven simulations. The architecture of the instruction set of the simulated CPU, called Portable ISA (PISA), has been inspired by the MIPS IV instruction set [17]. The CPU's internal data path is 32 bit wide. There are separate reorder buffers for compute and load/store instructions and register renaming facilities to make out-of-order instruction execution possible. The five stages of the pipeline can be reconfigured as well as the number of execution units. The cache levels use writeback mode and non-blocking loads and stores.

3.2. Simulator enhancements

We have integrated the following DRAM and memory controller models into SimpleScalar [4].

SDRAM. We have modeled a recent PC100 [11] compliant SDRAM of the newest 256 MBit generation with a 16 bit organization, four internal memory banks, and a row size of 1 KByte. Hence, four devices have to be controlled in parallel to form a common 64 bit wide memory bus, similar to a DIMM with four mounted devices. The controller sees memory banks four times the size of a single chip's bank. The memory bus is clocked at 100 MHz as in current personal computer systems. The following timing parameters described in [11] have been modeled:

Table 1. SDRAM timing parameters.

t_{RCD}	20 ns	*row address strobe to column address strobe delay:* time to activate a row of an idle memory bank
t_{AA}	20 ns	*column address strobe delay:* time between issuing a read instruction and the appearance of the first data item
t_{RP}	20 ns	*precharge time:* time to precharge an active row
t_{DPL}	20 ns	*data input to precharge delay:* time needed from the last data bus cycle of a burst write to a following precharge of the bank
t_{WAR}	20 ns	*write after read delay:* delay being subject to bus turn around limitations of the DIMM; measured from the last burst read data bus cycle to a following burst write data bus cycle

The CPU uses a cache line size of 32 Byte which corresponds to burst transfers of length four. Refresh operations take less than 1% of the run-time of the SDRAM and have been neglected.

RDRAM. The RDRAM memory bus is 16 bit wide as is an RDRAM chip. Again, 256 Mbit devices are modeled. The Rambus channel consists of four RDRAMs. The channel configuration allows to control the banks of each memory chip individually. The chosen RDRAM [19] consists of 32 internal memory banks with a row size of 2 KByte. However, adjacent banks have to share sense amplifiers. The Rambus channel runs at 400 MHz using both edges. The following timing parameters described in [19] have been modeled, see Tab. 2.

The CPU uses a cache line size of 32 Byte which corresponds to two data packets transferred over the Rambus channel. Again, refresh operations which take 1.0% of the run-time of the Rambus channel have been neglected.

Address translation. The memory controller has the option to map the physical addresses to chip, bank, row, and column addresses of the main memory. Two variants have been implemented.

- *linear translation:* The memory controller maps physical addresses in a linear manner to the chips and banks

Table 2. RDRAM timing parameters.

t_{RCD}	17.5 ns	*row to column address strobe delay*
t_{CAC}	20 ns	*column address strobe delay*
t_{CWD}	15 ns	*column address strobe write delay:* time needed from the end of the control packet which initiated the write access to the beginning of the corresponding data packet
t_{RP}	20 ns	*row precharge time*
t_{RTR}	20 ns	*retire delay:* time needed for data to be transferred from the write buffer of the device to the corresponding sense amplifiers
t_{RDP}	10 ns	*read to precharge delay:* time needed from the end of a control packet initiating a read access to the end of a precharge control packet
t_{PP}	20 ns	*precharge to precharge delay:* for precharge control packets in the same device
t_{packet}	10 ns	time to transmit a packet

of the main memory. That is, passing the boundaries of a DRAM row results in an access of the next row in the same internal DRAM bank.

- *interleaved translation:* Memory rows are mapped in an interleaved manner to the internal banks of the DRAMs. Our generic address permutation scheme results from the row size of the DRAM. One may think of interleaving divisions of a row. These address permutations however must be individually derived from the software and data sets used. In our scheme, whenever the address passes the boundaries of a DRAM row, the next address accesses another bank and not the next row of the same bank. Obviously, this mode is beneficial if an application accesses streamed data since changing to another memory bank is usually faster than accessing another memory row in the same memory bank. Due to the shared sense amplifiers every second bank is allocated consecutively for streamed data in the RDRAM case.

Activation policies.

- *strict open-page policy:* memory rows are kept activated as long as possible by the controller. This way, accesses bounded to the current row do not need to precharge and activate the row again.

- *closed-page policy:* the memory controller precharges an active row as soon as possible. If row changes are likely to happen frequently, this mode may result in better performance because some of the precharge time can be hidden.

Overlapped processing. The controller is able to process an access on the memory bus while receiving further requests from the CPU. The CPU must support this mode of operation by non-blocking load/store execution. Then, the controller may buffer requests in order to hide some of the latency introduced by activation and precharge tasks, taking advantage of the pipelined interface of recent DRAMs. An example is given in Fig. 1. Assume that the second op-

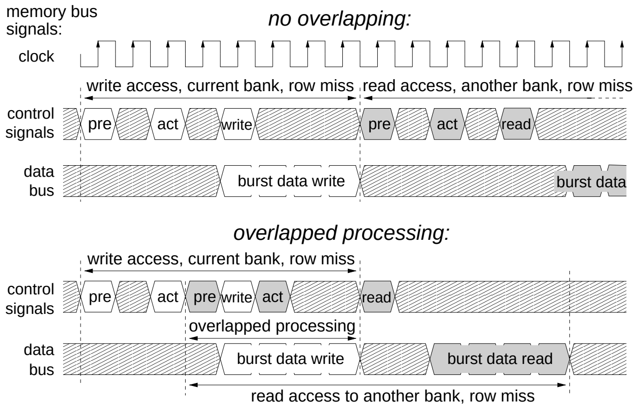

Figure 1. SDRAM, overlapped processing.

eration, the read access, is known during the processing of the first access, since the corresponding request has already been transferred from the CPU. The latency for precharging and activation can be completely hidden. The amount of clock cycles which can be overlapped depends on the current as well as the buffered accesses and the inter-arrival time of the requests at the memory controller. The requests are not rescheduled but stored in an in-order queue. Overlapped processing has not been modeled in closed-page mode. Therefore, linear and interleaved address translation are not distinguished in closed-page mode since both translation schemes result in the same latencies.

In closed-page mode, two different cases must be distinguished for SDRAMs and RDRAMs because only the request type (read or write) is of importance. Using open-page mode and SDRAMs without overlapped processing, 18 different cases are determined depending on the request type of the current and the buffered operations, the bank addresses, and the row state (row hit/miss). For RDRAMs, 36 main cases have been identified. The higher number of cases compared to the SDRAM controller is mainly caused by the shared sense amplifiers of the modeled RDRAM, the retire mechanism, and the use of several concurrent devices. If the overlapped processing mode is enabled in addition, the overlapping period may be of variable size. The period of time a request occupies the memory is determined at the arrival time of this request at the controller.

Our simulation model covers a reasonable set of memory controller features. Tab. 3 gives an overview of current SDRAM memory controllers. The support for a closed-page (*c-p*) activation policy, for an open-page policy together with the maximal number of open pages (*o-p/#pages*), and for overlapped processing is shown. Our modeled controller supports up to four open pages in the modeled SDRAM system and 64 open pages for the RDRAM channel. Recent announcements of RDRAM controllers [9] even talk about the support of several hundred open pages.

Table 3. Memory controller features.

chip name	c-p	o-p / #pages	overl.
embedded CPUs with integrated memory controller			
Hitachi SH-4/7750	yes	yes / 4	yes
Intel 80960RN	no	yes / 8	no
Motorola Coldfire	yes	yes / 1	no
Motorola MPC8xx	yes	no	no
Motorola MPC8240	yes	yes / 4	no
NEC V832	yes	yes / 1	no
stand-alone memory controllers			
Intel 440BX	yes	yes / 32	yes
Motorola MPC106	yes	yes / 2	yes
NEC VRC5074	yes	yes / 8	yes
workstation class stand-alone memory controllers			
AMD 751	yes	yes / 24	yes
Digital 21174	yes	yes / 24	yes
Via Tech. KT133	yes	yes / 16	yes

4. Experimental results

4.1. Applications

We do not only look at typical CPU benchmarks but also investigate real-world programs (see Tab. 4). Those programs which are freely available have been compiled from different sources at [2].

4.2. Simulations

Settings. In order to compare SDRAMs with RDRAMs under different memory controller configurations in an embedded system scenario, the following settings for SimpleScalar have been chosen. The CPU runs at a clock speed of 200 MHz and is two-way superscalar with out-of-order issue. Bimodal prediction with 512 entries is used. The reorder buffers have eight entries for compute and four entries for load/store instructions. The direct mapped first level cache is split into an 8 KByte instruction cache and an 8 KByte data cache. The cache line size is 32 Byte. There is no second level cache. Loads and stores are non-blocking. The following functional units are available: an integer ALU, an integer multiplier/divider, a floating-point ALU, and a floating-point multiplier/divider.

Table 4. Programs for simulation.

CPU benchmarks	
spec95 ijpeg	Image compression (input file *specmun.ppm*), integer performance rating
spec95 li	Lisp interpreter (input file *test.lsp*), integer performance rating
spec95 swim	Finite difference approximations (input file *swim.in*), floating-point performance rating
linpack	Linear algebra routines mix, floating-point performance rating
Applications	
g723	ITU-T rec. g723.1 speech decoder integer reference code. Decodes test vector *dtx63.rco* encoded at 6.3 Kbit/s.
g723enc	Corresponding encoder. The test vector *dtx53mix.tin* is encoded at 5.3 Kbit/s.
c4 1.0	Connect-four game solver. Performs search operations on hash table structures with more than one million 32 bit hash entries.
nsieve 1.2b	Computes prime numbers based on the algorithm of Eratosthenes. Arrays of several MBytes are used.
heapsort 1.0	Sorts random arrays with up to 2^{19} integers.
tfft	Fast Fourier Transform of 16 to 2^{16} samples.
tmn 3.1.2	ITU-T rec. H.263 video decoder by the Univ. of British Columbia and Telenor Research. 100 frames of a test sequence encoded at 1 Kbit per frame in qcif picture format are decoded.
tmnenc 3.1.2	Corresponding H.263 video encoder. 15 frames of a sequence are encoded at 3 Kbit per frame.
sim	Finds seven best non-intersecting alignments between two strings with 2500 characters using dynamic programming.
gzip 1.2.4a	Compresses its own distribution archive (2.5 Mbyte) using Lempel-Ziv coding.

Results. As described in section 3.2, the memory controller can apply linear (*lin*) as well as interleaved (*int*) address translation. Moreover, a closed-page (*c-p*), an open-page (*o-p*), and an open-page activation policy with overlapped processing can be chosen. We have not bounded the number of buffered requests in the performance model of our memory controller. However, the characteristics of the CPU architecture, especially the length of the reorder buffers, limit the number of pending requests. Finally, SDRAMs and RDRAMs can be controlled. The controller is supposed to be integrated in the embedded CPU.

The simulated execution times of the different programs are displayed in Tab. 5. The values have been scaled to the corresponding execution time using SDRAMs and closed-page mode. Tab. 6 provides information about program execution statistics: cache miss rates, the amount of main memory accesses scaled to the number of all executed instructions (*total*), the ratio between the number of mem-

Table 5. Simulated program execution times.

program	addr. transl.	SDRAM c-p	SDRAM o-p	SDRAM overl.	RDRAM c-p	RDRAM o-p	RDRAM overl.
heapsort	lin	1.0	1.02	1.02	0.99	1.01	1.01
	int		1.01	1.01		0.96	0.95
tmnenc	lin	1.0	1.00	0.99	0.99	0.99	0.98
	int		0.99	0.99		0.97	0.96
g723	lin	1.0	0.99	0.99	0.99	0.98	0.98
	int		0.98	0.98		0.97	0.97
g723enc	lin	1.0	0.98	0.98	0.98	0.97	0.96
	int		0.97	0.97		0.95	0.95
jpeg	lin	1.0	0.99	0.98	0.98	0.96	0.94
	int		0.98	0.97		0.93	0.91
li	lin	1.0	0.97	0.95	0.96	0.93	0.88
	int		0.95	0.93		0.85	0.80
c4	lin	1.0	0.86	0.84	0.92	0.81	0.75
	int		0.86	0.84		0.78	0.72
swim	lin	1.0	1.15	1.06	0.95	0.84	0.69
	int		1.01	0.88		0.75	0.61
sim	lin	1.0	1.09	1.05	0.98	1.00	0.93
	int		0.98	0.93		0.77	0.70
nsieve	lin	1.0	1.10	1.03	0.96	1.05	0.93
	int		0.96	0.91		0.82	0.71
tmn	lin	1.0	1.06	1.02	0.95	1.00	0.93
	int		0.89	0.87		0.81	0.74
gzip	lin	1.0	1.09	1.05	0.96	1.04	0.99
	int		1.01	0.96		0.86	0.80
linpack	lin	1.0	1.06	1.06	0.96	1.02	1.01
	int		0.92	0.91		0.84	0.82
tfft	lin	1.0	1.03	1.01	0.96	0.98	0.94
	int		1.00	0.98		0.91	0.87

Table 6. Program execution statistics.

program	cache miss [%] instr.	cache miss [%] data	DRAM accesses total [%]	DRAM accesses overl. [%]	exec. load/store rel. [%]	exec. load/store abs. $[10^6]$
heapsort	0.0	2.7	1.1	0.3	22.1	310
tmnenc	0.2	1.5	0.7	12.7	28.5	368
g723	0.5	0.1	0.6	3.0	13.4	120
g723enc	0.8	1.3	1.0	2.2	12.4	189
jpeg	0.7	3.7	1.9	17.4	25.2	147
li	4.5	2.3	6.4	9.3	47.9	495
c4	10.9	3.4	13.8	8.4	32.6	1229
swim	1.3	41.4	16.1	42.6	29.4	237
sim	2.3	17.9	11.4	18.2	36.3	1595
nsieve	0.0	25.9	5.9	45.0	11.7	113
tmn	1.2	8.5	7.1	27.3	55.5	250
gzip	0.0	17.0	5.3	28.6	28.0	138
linpack	6.1	8.9	4.5	3.7	26.9	21
tfft	1.0	7.7	5.5	15.7	38.9	80

ory accesses that can make use of overlapped processing to the number of all main memory accesses, the amount of loads and stores scaled to the number of executed instructions (*rel.*), and the absolute count of executed load and store instructions. Since the amount of memory requests which can make use of overlapped processing does not vary remarkably by changing the address translation mode or the DRAM type, only the maximum of these simulation runs is considered in the table. Finally, Tab. 7 shows hit and miss statistics for the main memory system using open-page mode. Looking at two consecutive memory requests, the next pending request may access the same row as the current access. This situation is called a hit in the current bank (*cb hit*). If the pending request wants to access a different row in the same bank, a miss in the current bank occurs (*cb miss*). Finally, the situations where the pending request will hit or miss a row in another bank than the current one are counted as accesses to another bank (*ob*). The negligible amount of accesses to idle banks is not quoted in the table.

Analysis. For four programs the execution time is almost independent on the memory and controller used: *heapsort*, the video encoder *tmnenc*, and the voice processing programs *g723* and *g723enc*. The programs show almost no misses in the cache. Although load and store instructions take around 20% of the executed instructions, only slightly more than 1% of the executed instructions result in a main memory access. This rate is too low to produce any run-time effects. For instance, looking at RDRAMs and the *tmnenc* program, the rate of expensive row misses can be reduced from 42% for linear address translation to 2% using interleaved address translation. However, there is no noticeable difference in the execution time.

For another group of programs, *jpeg*, *li*, and *c4*, using interleaved address translation always shortens the execution time. In most configurations however the differences are negligible. The programs run faster by controlling RDRAMs instead of SDRAMs. Applying overlapped processing only improves the performance of RDRAM configurations noticeably. Using open-page mode instead of closed-page mode improves the execution time for both DRAM types. The programs have in common that they generate moderate miss rates in the instruction cache and almost no misses in the data cache, see Tab. 6. Since the share of loads and stores of all executed instructions is high (up to 48% for *li*), a noticeable amount of the simulated instructions results in main memory accesses. Moreover, the accesses to main memory are very localized. At least 56% of the memory accesses hit an already activated memory row. Thus, the memory controller benefits well from the open-page policy. The remaining amount of access misses is too small to influence system performance by changing from linear to interleaved memory translation.

The remaining group of programs, consisting of *swim*, *sim*, *nsieve*, *tmn*, *gzip*, *linpack*, and *tfft*, shows a noticeable performance improvement by using overlapped processing (except *linpack*). The execution time can be shortened up to 19% for RDRAMs using overlapped processing (*swim*,

Table 7. Main memory access statistics in percent of all memory accesses.

program	addr. transl.	SDRAM [%] cb[1)] hit	SDRAM [%] cb[1)] miss	SDRAM [%] ob[2)] hit	SDRAM [%] ob[2)] miss	RDRAM [%] cb[1)] hit	RDRAM [%] cb[1)] miss	RDRAM [%] ob[2)] hit	RDRAM [%] ob[2)] miss
heapsort	lin	4.4	91.7	2.7	1.2	3.1	59.5	6.8	30.6
	int		32.2	20.5	42.9		0.8	86.4	9.7
tmnenc	lin	30.9	45.6	15.6	7.9	30.1	31.5	28.0	10.3
	int		20.9	36.2	12.0		0.5	67.8	1.6
g723	lin	71.1	25.5	2.6	0.8	69.9	26.7	2.4	1.0
	int		6.6	19.7	2.6		0.0	29.8	0.3
g723enc	lin	57.7	26.1	14.6	1.6	57.3	26.5	14.6	1.5
	int		3.7	36.9	1.7		0.0	42.6	0.0
jpeg	lin	41.0	36.2	15.5	7.3	39.2	24.3	25.5	11.0
	int		19.5	26.6	12.9		0.7	58.7	1.3
li	lin	55.5	26.8	8.9	8.8	55.0	27.3	8.2	9.4
	int		13.3	17.0	14.2		0.3	43.3	1.4
c4	lin	79.5	8.9	8.0	3.6	77.3	9.5	8.0	5.2
	int		7.5	5.5	7.5		1.3	17.0	4.4
swim	lin	13.9	85.8	0.1	0.1	9.8	10.0	62.3	17.8
	int		11.1	34.8	40.1		0.0	88.9	1.2
sim	lin	30.1	59.1	4.9	5.9	30.1	59.1	4.9	5.9
	int		9.7	29.9	30.3		0.1	69.8	0.0
nsieve	lin	1.6	98.3	0.0	0.0	1.4	88.4	1.7	8.5
	int		24.9	54.4	19.1		0.3	89.3	9.0
tmn	lin	16.1	74.0	6.0	3.9	15.8	70.9	8.1	5.2
	int		9.8	65.4	8.7		0.2	82.5	1.5
gzip	lin	7.6	92.3	0.1	0.0	6.5	93.4	0.1	0.0
	int		25.5	32.7	34.2		0.1	92.5	0.9
linpack	lin	9.6	88.0	1.7	0.7	8.8	88.9	1.7	0.6
	int		15.2	67.4	7.8		0.1	91.0	0.1
tfft	lin	29.3	62.5	4.0	4.2	28.3	63.1	4.3	4.3
	int		42.0	19.8	8.9		21.7	33.7	16.2

1) *cb hit/miss:* hit/miss in the current bank
2) *ob hit/miss:* hit/miss in another bank than the current

interleaved translation) and up to 13% for SDRAMs respectively (*swim*, interleaved translation) since up to 45.0% of all main memory accesses can make use of this mode. Interleaved address translation shortens the execution time by up to 25% (*sim*, RDRAM, with overlapped processing). Since the programs generate high miss rates in the data cache combined with a high count of executed load and store instructions, at least 5.3% of all executed instructions result in a main memory access. RDRAMs benefit more from overlapped processing since their level of pipelining can better cope with bus turn-around situations. For instance at *nsieve*, for which almost 90% of the memory transfers have been classified as *read after write* or *write after read* accesses, the RDRAM configuration shows a 13% increase in performance compared with only 6% for the SDRAM configuration. The significant performance improvements by using interleaved address translation instead of linear translation can be achieved because the programs show high DRAM miss rates in the current bank compared with the preceding group of programs. Using interleaved translation, these relatively expensive accesses can be exchanged for cheaper accesses in other memory banks. Moreover, RDRAMs benefit more from interleaving, since the high number of internal banks causes considerably less conflict misses. The high miss rate in the current bank also explains why the SDRAM configurations using open-page mode and linear translation perform poorer than the corresponding configurations using closed-page mode. A closed-page policy is able to hide some of the precharge overhead which may be caused by frequent memory row changes.

Summary. The results are summarized in Fig. 2. On the one hand, the RDRAM configuration using open-page mode and interleaved address translation always outperforms all other configurations. The minimal performance gain compared with all other configurations ranges from some percent to up to 25% for the *sim* application. On the other hand, the RDRAM more heavily depends on the chosen memory controller access scheme due to the level of pipelining and the number of memory banks involved. In the worst-case, the *sim* program looses more than 50% of its best-case performance by using RDRAMs with closed-page mode. SDRAMs using open-page mode and interleaved address translation can well compete with RDRAMs using linear translation.

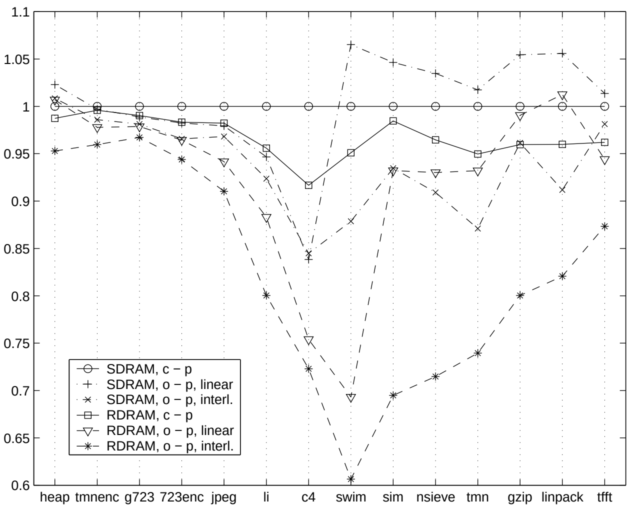

Figure 2. Normalized simulated execution times for closed-page and open-page modes with overlapped processing. SDRAM with closed-page mode is used as reference.

5. Conclusion

This paper has investigated the influence of recent DRAM technologies, SDRAMs and Direct Rambus RAMs, on embedded systems performance. Especially, the impact of the memory controller on the resulting performance has been explored by simulating a modern 32 bit out-of-order superscalar CPU architecture together with different main memory access schemes which can be implemented in custom-designed controllers. Since embedded systems use fewer cache levels and smaller cache sizes due to power, area, and cost constraints, their performance particularly depends on the main memory system. The simulation of 14 different applications showed the following results for our system configuration:

- Looking at RDRAMs, applying an open-page access heuristic results in better system performance than a closed-page access scheme. SDRAMs however do not necessarily perform better in open-page mode.

- Exploiting the internal multi-bank structure of modern DRAMs by a row-wise interleaved address translation scheme accelerates the program execution by up to 25%. The amount of resolved capacity misses thus has always outperformed the newly caused conflict misses.

- Using additional in-order request buffers to overlap the processing of successive memory accesses may decrease the execution time by up to 19% for RDRAMs and by up to 13% for SDRAMs.

- The RDRAM system outperforms the SDRAM system by up to one third using the same memory controller settings. RDRAMs benefit from their higher number of internal banks, better bus turn-around behavior, and shorter page hit latencies.

Thus, from a performance point of view, the transition from the mature SDRAM to the new RDRAM technology can be indeed beneficial for embedded systems. Besides, applying an interleaved address translation scheme has a greater impact on the system performance than exploiting the pipelined memory interfaces.

Acknowledgments. The author would like to thank M. Platzner and A. Romer for valuable comments and discussions.

References

[1] L. A. Barroso, K. Gharachorloo, and E. Bugnion. Memory system characterization of commercial workloads. In *25th International Symposium on Computer Architecture*, pages 3–14, 1998.

[2] BenchWeb. Benchmark branch of the Netlib repository, University of Tennessee. *http://www.netlib.org/benchweb/*.

[3] K. Boland and A. Dollas. Predicting and precluding problems with memory latency. *IEEE Micro*, 14(4):59–67, Aug. 1994.

[4] D. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin-Madison, June 1997.

[5] D. Burger, J. R. Goodman, and A. Kägi. Memory bandwidth limitations of future microprocessors. In *23th International Symposium on Computer Architecture*, pages 78–89, 1996.

[6] J. Carter, W. Hsieh, L. Stoller, M. Swanson, L. Zhang, E. Brunvand, A. Davis, C.-C. Kuo, R. Kuramkote, M. Parker, L. Schaelicke, and T. Tateyama. Impulse: building a smarter memory controller. In *Fifth International Symposium on High-Performance Computer Architecture*, pages 70–79, 1999.

[7] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. A performance comparison of contemporary DRAM architectures. In *26th International Symposium on Computer Architecture*, pages 222–233, 1999.

[8] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic. Memory-system design considerations for dynamically-scheduled processors. In *24th International Symposium on Computer Architecture*, pages 133–143, 1997.

[9] L. Gwennap. Alpha 21364 to ease memory bottleneck. *Microprocessor Report*, 12(14):12–15, Oct. 1998.

[10] S. I. Hong, S. A. McKee, M. H. Salinas, R. H. Klenke, J. H. Aylor, and W. A. Wulf. Access order and effective bandwidth for streams on a Direct Rambus memory. In *Fifth International Symposium on High-Performance Computer Architecture*, pages 80–89, 1999.

[11] Intel Corp. *PC SDRAM Specification, Rev. 1.7*, Nov. 1999.

[12] Joint Electron Device Engineering Council (JEDEC). *Standard 21C (JESD21C): Configurations for solid state memories: official and preliminary releases*.

[13] M. Kumanoya, T. Ogawa, Y. Konishi, K. Dosaka, and K. Shimotori. Trends in high-speed DRAM architectures. *IEICE Transactions on Electronics*, E79-C(4):472–481, Apr. 1996.

[14] S. A. McKee, R. H. Klenke, K. L. Wright, W. A. Wulf, M. H. Salinas, J. H. Aylor, and A. P. Batson. Smarter memory: improving bandwidth for streamed references. *IEEE Computer*, 31(7):54–63, July 1998.

[15] Y. Oshima, B. J. Sheu, and S. H. Jen. High-speed memory architectures for multimedia applications. *IEEE Circuits and Devices Magazine*, 13(1):8–13, Jan. 1997.

[16] M. Porrmann, J. Landmann, K. M. Marks, and U. Rückert. HiBRIC-MEM, a memory controller for PowerPC based systems. In *23rd Euromicro Conference: New Frontiers of Information Technology*, pages 653–657, Sept. 1997.

[17] C. Price. *MIPS IV Instruction Set, revision 3.1*. Mips Technologies Inc., Mountain View, CA, USA, Jan. 1995.

[18] B. Prince. *High Performance Memories: New Architecture DRAMs and SRAMs - Evolution and Function, revised ed.* John Wiley & Sons Ltd., 1999.

[19] Rambus Inc. *Direct RDRAM 256/288-MBit (512K x16/18 x 32s), Preliminary information*, Apr. 2000.

[20] M. Rosenblum, E. Bugnion, S. A. Herrod, E. Witchel, and A. Gupta. The impact of architectural trends on operating system performance. In *15th ACM Symposium on Operating Systems Principles*, pages 285–298, 1995.

[21] Standard Performance Evaluation Corporation. Open Systems Group, CPU benchmark suite. *http://www.spec.org*.

[22] K. M. Wilson and K. Olukotun. Designing high bandwidth on-chip caches. In *24th International Symposium on Computer Architecture*, pages 121–132, 1997.

[23] W. A. Wulf and S. A. McKee. Hitting the memory wall: implications of the obvious. *Computer-Architecture-News*, 23(1):20 – 24, Mar. 1995.