

Stream data burst using embedded shape information

Sek M. Chai, *Member, IEEE*,
 Embedded Systems Research, Motorola Labs,
 (sek.chai@motorola.com)

Abelardo López-Lagunas, *Member, IEEE*,
 ITESM-Toluca, Mexico
 (abelardo.lopez@itesm.mx)

Abstract— Memory access patterns for streaming data are deterministic because the requested data formats (shape) are well defined. Accordingly, the memory subsystem can use the shape information to prefetch data and enable transfers that overlap computation. This paper advocates a technique to embed data shape information in physical DRAM memory to improve both latency and bandwidth. With shape data, the memory can assemble and pack requested data for transfers in a *stream-burst* format. This paper also describes an example SDRAM memory design and its expected performance for streaming applications.

Index Terms— Memory systems architecture, stream processing, memory latency, memory bandwidth, data prefetch.

I. INTRODUCTION

Current VLSI technology enables the integration of large number of processing elements to satisfy computing demands, but wire interconnect delays, poor temporal locality of data, and slow memories relative to the processing elements maintain an imbalance between processor and memory performance. Designers cope with poor memory performance issues with caching, data replication and prediction (prefetching) [1]. However, caching techniques are not effective for streaming computation due to poor temporal locality of the data [2]. Large chip area dedicated for caches and data duplication can be better utilized for processing. Furthermore, cost and power constraints contributed by large caches and complex prefetching logic limit application migration to embedded systems.

Extending the approaches in [1], this paper proposes a *stream-burst* functionality in physical DRAM memory to burst streaming data to the memory controller. This streaming data is packed with all bytes in proper alignment and in the order requested by the computing platform. To enable this feature, the DRAM memory is embedded with the shape of requested data so that the memory itself can organize, align, and pack the data for transfers. The extra logic in DRAM memory to enable stream-burst functionality can lead to more efficient bus transfers and storage buffer designs in the processor platform.

This stream-burst functionality is facilitated in the stream computation model [3]. In this model, computation is decoupled from communication, allowing hardware designers to optimize the implementation of the processor and memory subsystem separately. Data shape can be specified with stream descriptors [4,5], a mechanism for the programmer to describe format and location of data in memory. Stream descriptors have been previously described for use with a processor [4,5] and peripherals [6,7]. In this

paper, the use of stream descriptors is applied to physical memories such as SDRAM so that data is assembled and burst back to the processor as sequential streams. The structure of the paper is as follows: Section II presents the related work relevant to this paper; Section III gives a brief presentation of stream descriptors and their use in a set of image processing applications; Section IV describes an example SDRAM design using stream descriptors; Section V concludes and presents the future work for this research area.

II. RELATED WORK

A. Stream memory subsystem

Stream data has predictable access patterns which enables optimization of the memory subsystem. There is already an existing body of work related to the efficient transfer of stream data from memory to processor. In [8], stream buffers are used to separate stream data from cached data because stream data has low temporal locality and can trash cache contents. In [9], a stream memory controller prefetches and buffers stream data from memory by dynamically reordering the accesses. Readers are referred to [9] for an overview of compile-time and run-time techniques for data prefetching.

In the Impulse Adaptable Memory System, [10], shadow addresses (memory ranges not backed by DRAM) are mapped to physical addresses in order to improve bus and cache performance. The Impulse memory controller can also prefetch data from sparse DRAM locations and assemble the data as a contiguous cache line.

Stream processors also optimize data transfers as they operate on a stream of data that appears sequential but may be scattered throughout memory [11]. Examples of stream processors include RAW [12], Imagine [13], Merrimac [14], and the RSVPTM architecture [4,5]. In the stream computation model, the stream data type and movement are explicitly defined as communication between computation kernels [15]. The explicit definition allows configuration of the stream memory subsystem for optimized transfers of stream data to the computing platform. In [12], a compiler schedules a flexible network to gather data from memory locations distributed across the chip. In [13,14], stream register files are used to store and assemble data as contiguous streams. In [4,5], dedicated stream units prefetch and gather stream elements from different memory locations. The stream units use stream descriptors, which are described later in Section III.

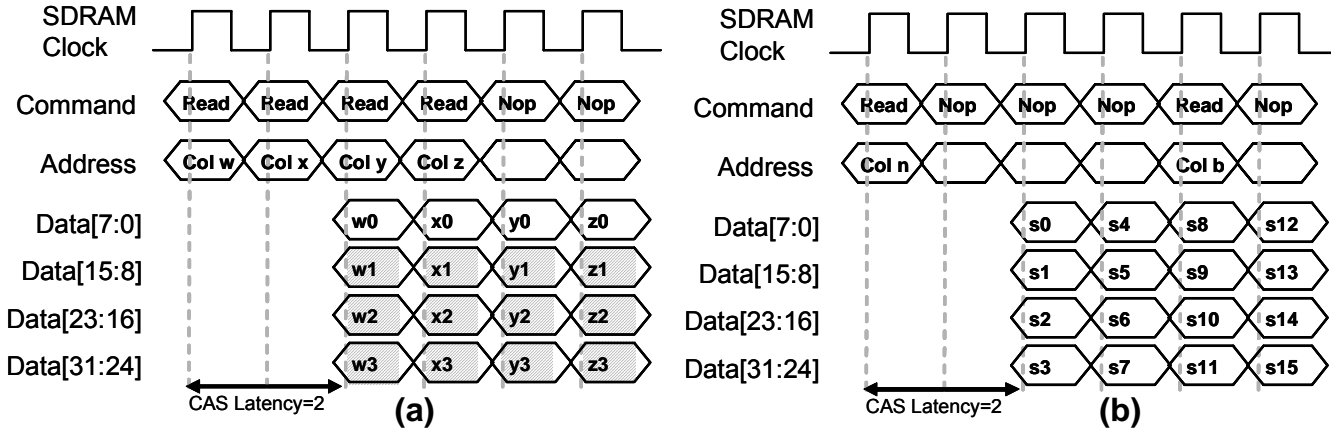


Figure 1. SDRAM commands and data transfer example.
(a) traditional random column burst, (b) proposed stream-burst

B. DRAM Data Burst

There are numerous improvements on the basic DRAM design to improve the overall throughput. Readers are referred to [16] for more details. Some updates come with relatively minor changes in their implementation, but they allow significant performance improvements. Fast page mode (FPM) keeps the sense amplifiers active over multiple column addresses, extended data-out (EDO) adds a latch between sense amplifiers and output pins to keep data valid into the next transfer phase and allow faster precharge, and burst EDO (BEDO) adds an internal counter to burst data on consecutive column addresses. Today’s synchronous DRAM (SDRAM) adds a synchronous clock and can even provide data on both edges of the clock (double-data rate or DDR-SDRAM). As shown in Figure 1a, SDRAM can burst data from the different column addresses in each bank (for row addresses that are already activated in each bank) [17]. If a different row is opened, the memory controller would first issue a precharge (PRE) command to close an already opened row. Then it would issue an active (ACT) command to open a new SDRAM row. In typical accesses, the SDRAM would be configured to burst consecutive column address from the same row and bank addresses.

Other changes include major structural modifications around the same basic DRAM design. For example, RDRAM [18] from Rambus has a specialized bus protocol that enables packet transfer over a narrow bus at high speeds. RDRAM also has more banks per megabyte than SDRAM, which lowers bank conflicts during transfers.

More radical approaches include PIM (“processor-in-memory”) designs that allow memory to perform computation [19,20]. Although the close proximity of processor to memory allows for wider buses, DRAM processes are substantially slower than logic fabrication processes.

These evolving memory technologies have been geared towards increasing bandwidth rather than latency [1], which bias towards platforms with large caches. In both SDRAM and RDRAM cases, the transferred data are not packed as a

sequential data stream for processing by a stream processor and may contain unused bytes, as shown in Figure 1a with shaded regions. Furthermore, although the command-based structure of RDRAM is amenable for streaming data, the high speed busses in RDRAM can be difficult to implement, and can exceed the power budget in embedded systems.

The proposed stream-burst feature seeks to make the data bursting protocol more efficient by leveraging stream descriptors to describe the shape of the data. As shown in Figure 1b, a stream-burst memory follows similar SDRAM operations, but its internal logic can assemble data elements for transfers in a data stream. This stream-burst feature is a form of prefetching as data elements, potentially from different column addresses, are gathered and placed onto the data stream. For streaming applications, this stream-burst may reduce the reliance on large caches because the data stream is readily consumed by the stream processor.

III. EMBEDDED DATA SHAPE INFORMATION

A. Stream Descriptors

Stream descriptors are programming interface mechanisms for the programmer to describe the shape and location of data in memory. In the stream computation model, stream descriptors decouple memory address generation from actual computation with explicit definition of the stream data. In [4,5], dedicated stream units utilize the stream descriptors to prefetch data from memory for the computing platform. Each stream unit handles all issues in loading/storing of data: address calculation, byte alignment, data ordering, and memory bus interface. In [6,7], stream descriptors are applied to peripherals such that raw data from sensors are assembled as streams for the computing

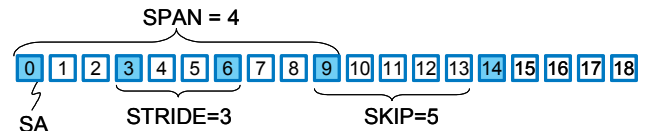


Figure 2. Example stream descriptor

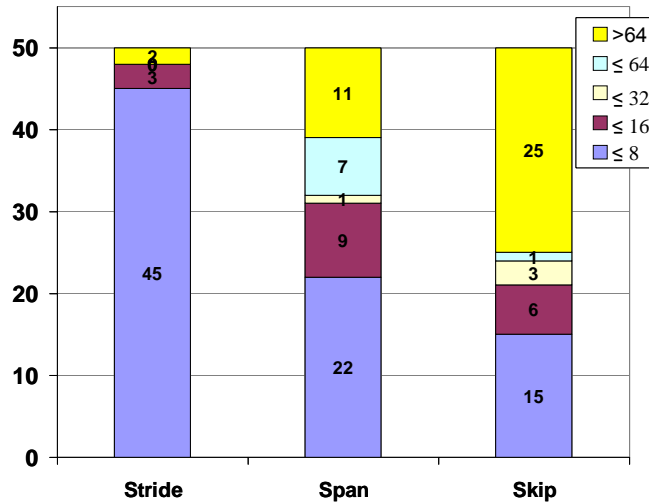


Figure 3. Example variation of stream data shape for a set of image processing applications

platform.

A stream descriptor is represented by the tuple (Type, Start_Address, Stride, Span Skip, Size) where:

- **Start_Address** (SA) represents the memory address of the first stream element.
- **Stride** is the spacing, in number of elements, between two consecutive stream elements.
- **Span** is the number of elements that are gathered before applying the skip offset
- **Skip** is the offset that is applied between groups of span elements, after the stride has been applied
- **Size** is the number of elements in the stream
- **Type** indicates how many bytes are in each element (Type is 0 for bytes, 1 for 16-bit half-words, etc.)

The Stride, Span, Skip, and Type fields define the shape of a data object. The grouping and order in which data is accessed defines a *Stream Record* and corresponds to the preferred alignment of the computation kernel. Stream records can be processed in parallel by stream processors and this explicit alignment of the data facilitates their hardware implementation by eliminating packing and unpacking instructions. Multidimensional or even non-regular shapes can be created by extending the defined semantics of each stream descriptor field. Furthermore, a stream descriptor may include additional parameters to represent movement and additional dimensionality of the stream record. Figure 2 shows an example stream descriptor.

The above stream descriptor structure allows equivalent definitions for the same data format. Programmers focus on stream descriptor definition based on the computation to be performed, and a compiler can then manipulate the descriptor to better match the data transfers on the underlying memory subsystem. For example, the compiler can adjust the stream descriptor parameters to match the width of each of the data busses throughout the interconnect hierarchy that connect the stream processor to the memory controller. In this manner, stream descriptor manipulations improve the portability of the code across different hardware

substrates and increase the level of abstraction of the programmer. The stream descriptor structures and compiler manipulations are active research areas. Regular data access patterns, such as those occurring in tight loops, have also been represented as descriptors and used for trace generation [21,22]. Readers are also referred to [5,7] for more details.

B. Benchmarks using stream descriptors

This section describes the data shapes in a set of image processing applications in order to understand the variation in stream descriptor parameters. The application set includes license plate recognition, image processing chain, MPEG4 encode, MPEG4 decode, and lane departure warning [23,5,24], which have been selected as representative benchmarks for embedded smart cameras. In most cases, these algorithms consist of a series of operations on portions of an image frame buffer, data structures related to intermediate data, or filter coefficients in memory.

Figure 3 shows an example distribution of stream descriptors in the application set. The stream descriptors are chosen from unique data accesses and not according to program flow because the execution path may vary for different data sets. In doing so, the variation and range of stream descriptor values are presented to uncover their effect on data transfers. The stride, span, and skip descriptors are placed in bins of power of two that represents the bus widths in bytes. For comparison purposes, the absolute values of the descriptors are presented to show displacement without direction of access. Correlations between the descriptors are application dependent and are not discussed as they are beyond the scope of this paper.

The Stride descriptor represents the spatial locality of the data shape. For the application set, processing is performed on region of images where pixel data is stored in row-major format. As such, stride values are small (≤ 8) such that for every data element selected, the consecutive data element

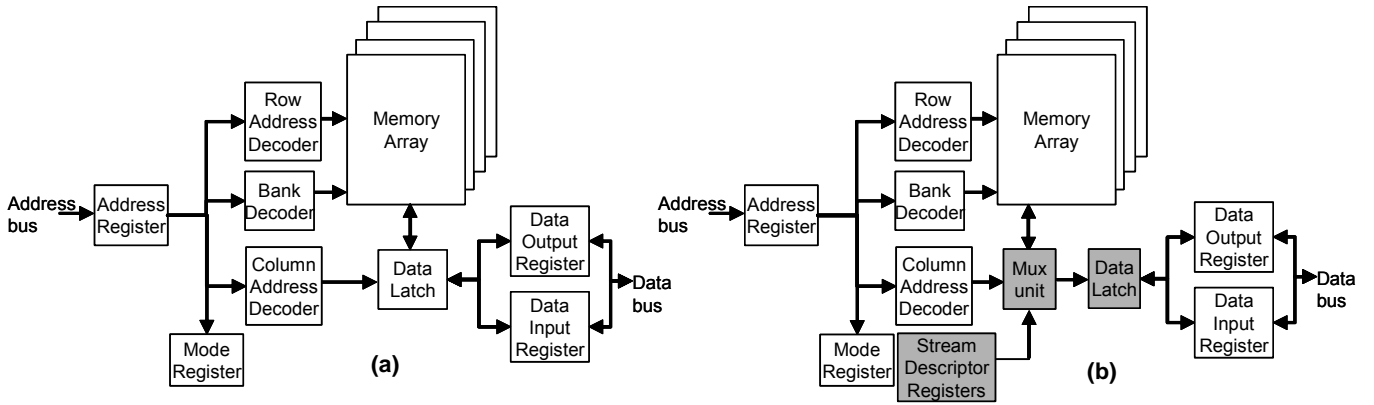


Figure 4. (a) Standard SDRAM design, and (b) SDRAM design with stream-burst functionality

will be transferred as well. For column-wise processing, the stride values are encoded with values equal to the image height, which reduces bus bandwidth utilization because consecutive elements in the stream record are not consecutive in physical memory. As such, the algorithms have been restructured to process smaller image tiles, if possible, to change data access and favor small stride values. The Span descriptor represents the number of stream elements within a collection of stream data items. Large span values with small stride values indicate the likelihood of multiple stream data elements within a burst of data from memory.

The Skip descriptor represents the spatial distances between collections of stream data items. Small skip values (≤ 8) indicate the probability of fetching the next stream data element in consecutive bursts, and large skip values indicate otherwise. It is important to note that there are mid-ranged skip values, which indicate that the next stream data element is within a 1024 byte row in SDRAM. This means that even though bursts of consecutive column addresses will not fetch the next stream data element, a new SDRAM row does not need to be precharged and activated. The Type descriptor represents the data element size. For this set of applications, stream data elements are bytes for image data, half-word for filter coefficients, and 32-bit words for buffer transfers.

Stream descriptors are used in stream units in [4,5] to prefetch data based on the shape and location of requested data. The stream units reside within the SoC to move data between storage units (embedded memories or peripherals) and stream processors. They can also be integrated in the memory subsystem in components such as tightly coupled DMA units in processors [25,26], stream buffers [8], or stream memory controllers [9,10]. To further improve performance, this paper explores the use of stream descriptors in the physical memory such as SDRAM to more efficiently burst data to the stream processor through the memory subsystem. Since the shape of requested data is deterministic, there are opportunities to reduce the number of transfer commands sent to the memory controller. As shown in Figure 3, there are many occasions where consecutive data elements are not addressable with consecutive column addresses which are used in a standard SDRAM burst. On the other hand, there are many cases where the next column

addresses can be calculated a priori such that data is packed more efficiently during transfers, enabling *stream-burst* functionality in physical memories.

IV. STREAM-BURST

This section describes the design of an SDRAM that uses stream descriptors as a prefetching and alignment mechanism. Embedding stream descriptors in physical DRAM memories is ideal since memories are the physical medium that stores the data and they represent the last architectural mechanism in data transfers. Furthermore, the basic DRAM structure already has selection logic circuits (row and column decoders) that could be reused to support stream-burst functionality. In doing so, the number of bus requests would be reduced since the set of data elements are packed on the data bus for each request. In addition, lower latency responses can be achieved since the memory can determine the next set of data elements to transfer beforehand using stream descriptors. Unlike other approaches to embedded prefetching mechanisms in DMA or memory controllers, stream-burst can reduce the number of transfers from memory for data that is not spatially co-located. This reduces bus traffic, bus contentions, and power consumption.

A. Embedding stream descriptors in SDRAM

Figure 4a illustrates the components in a standard SDRAM design [16,17]. A set of physical addresses (row address) is presented to the row decoders to select a row of data from different memory banks. Another set of physical addresses (column address) is presented to the column decoders to select a set of data equal to the bus width of the memory devices. Current SDRAM technology allows for bursting by incrementing the column addresses to transfer consecutive sets of data. While there are many different approaches to enable stream-burst functionality, the design choices presented in this paper seek to maintain the standard SDRAM design, since it is well understood and tested over many generations of commercial products.

To embed stream descriptors in SDRAM, additional logic is added to store the descriptors and to calculate the next column address. To maintain the standard DRAM

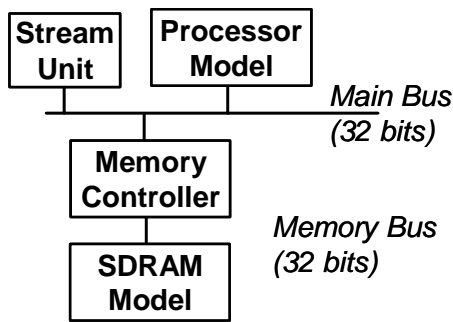


Figure 5. Simulation Platform

structure, additional logic circuits to advance row addresses are not added, even though stream descriptors can be used to determine the next row address. If row address manipulation is supported, the functionality would require changes in the memory operation timing with respect to refresh and row activation. These changes would manifest in the memory controller timing parameters as well.

In using the embedded data shape information, there is a necessary set of operational procedures which includes:

- Initialization

The SDRAM must be loaded with stream descriptor values during program execution. The compiler can schedule the processor to preload these values during run time. The number of supported data streams depends on the amount of storage for stream descriptors in the SDRAM.
- Normal operation

The SDRAM operates normally with bursts of consecutive column addresses. Maintaining the normal mode is important as random-memory access is important for cache access by the scalar processor.
- Stream-burst mode enabling

The SDRAM is enabled to burst stream data based on a preloaded stream descriptor. The SDRAM remains in this mode for all data transfers while this mode is enabled.
- Termination of stream-burst mode

The SDRAM returns to normal operation mode.

Figure 4b shows additional components to embed stream descriptors in a standard SDRAM design. This design allows the above operational procedures to enable the stream-burst functionality with minimal changes to the standard SDRAM structure and memory controller operation. Registers are included to store stream descriptor values and a mux unit is used to select different columns of data based on pre-calculated column addresses. Because row data from multiple banks (if already activated) are presented to the mux unit, the data latch unit is able to select from different data columns within opened rows from different banks. The data prefetching notion for stream descriptors in SDRAM is therefore reduced to the ability to assemble stream elements within opened row data. Stream descriptors are used to calculate the next sets of column addresses in advance of the actual requests from the memory controller.

B. Model implementation and evaluation

A simulation platform that includes the stream unit, a memory controller, a processor module, and SDRAM

module is shown in Figure 5, and is built using Verilog HDL to accurately model the data movement behavior. Verilog HDL simulations are chosen because they allow for more accurate cycle-by-cycle simulation of SDRAM behavior. The stream unit has an address generator that produces memory addresses for the memory controller to access stream data from SDRAM. The main bus is based on the Wishbone protocol [27] and operates at 200MHz.

The memory controller and processor modules are based on open source designs [28] that were modified to include circuits that measure performance and ability to issue stream descriptors. Registers have been included to store stream descriptors written by the processor during initialization. The registers are placed on the memory controller's memory map. In order to minimize design changes in the memory controller, the *load_mode_register* feature in the memory controller has been modified to transmit stream descriptor values, previously stored in memory controller registers. The stream descriptor values are placed on the data bus during the *load_mode_register* procedure, which is used to set SDRAM burst and CAS latency [17] during initialization. In typical SDRAM operation during mode register programming, the data bus is unused. For stream-burst mode, the 32-bit data bus is used to encode stride (10bits), span (10bits), skip (10bits), and type (2bits). The bit widths of the stride/span/skip stream descriptors are chosen to match the number of bits of the column address.

The memory module is the Micron 64Mb SDRAM (2M x 32 x 4 banks, 100MHz, CAS latency of two cycles) memory model [17]. The memory model has been modified to enable stream-burst transfer upon receiving the following burst mode setting: 3'b100 for stream-burst of four, and 3'b101 for stream-burst of eight. Both settings are originally reserved, but are utilized in this simulation for the stream-burst feature. Upon mode register programming, the mux unit is set to select data from a selected SDRAM column in an opened row, as described previously for Figure 4b. A new mux unit setting is then selected when the next burst is requested. The stream-burst transfer can be disabled when the mode register is programmed for normal bursts.

In this implementation of stream-burst mode, the row address is still used to select the SDRAM row from a set of memory banks. In an embodiment which is modeled for this paper, the column address is used with the mux unit as an offset to select the alignment of data on the bus. Data is then burst back to the memory controller as streams according to the request order described by the data shape embedded in stream descriptors. Unlike normal SDRAM random column read (Figure 1a), data is packed such that all bytes are used by the processor. In other embodiments, the column address is not used as it can be calculated by a preloaded stream descriptor, *start_address*.

The memory controller and SDRAM modules were chosen for initial modeling without the influence of specialized or proprietary bus protocols, memory controllers, or memory designs. For this paper, the necessary set of operation procedures to enable stream-burst

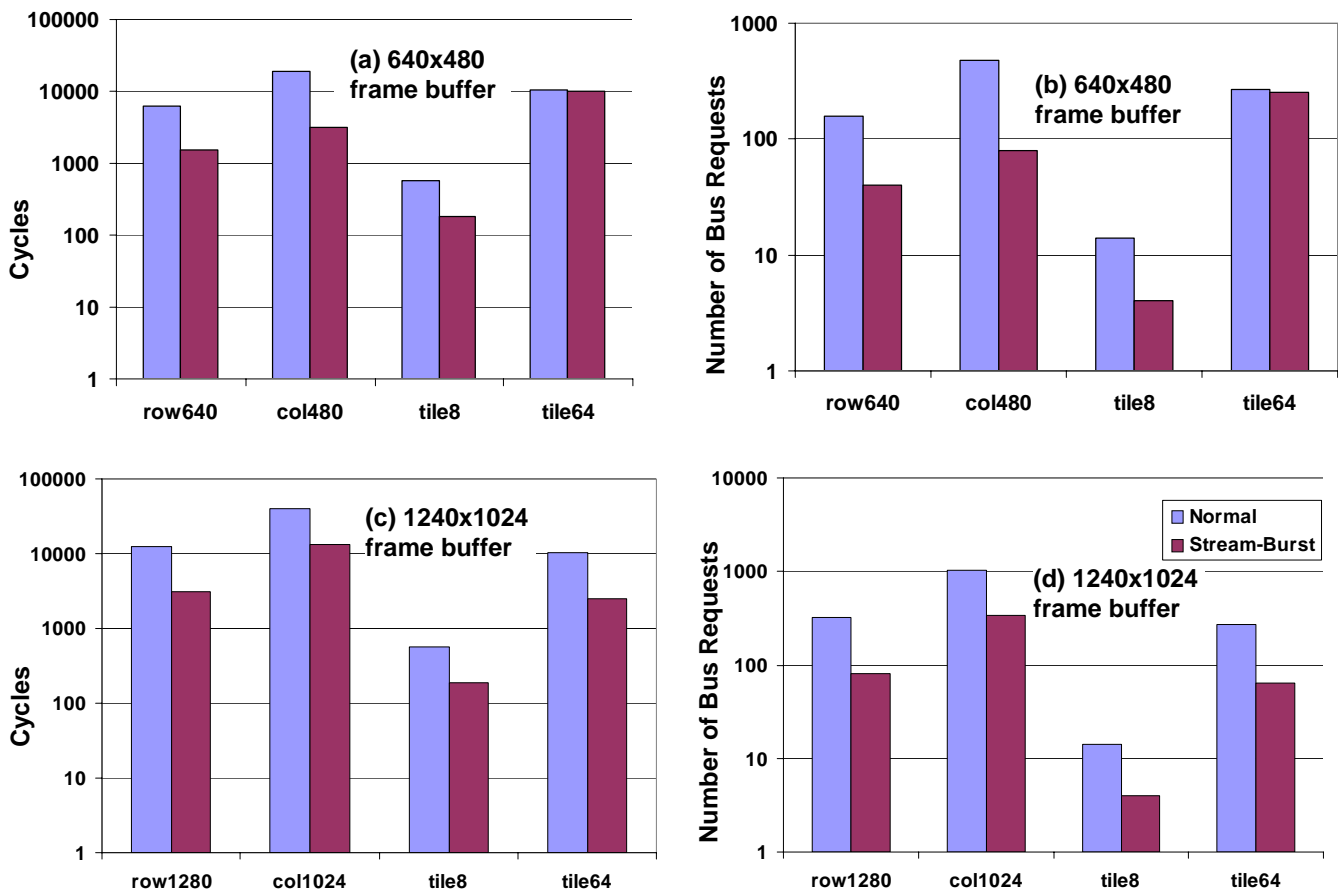


Figure 6. Simulation results of normal vs. stream burst for different data shapes,
(single row-access, single column-access, 8x8 byte tile access, 64x64 tile access in 640x480 and 1240x1024 frame buffers)

transfer is described above as a simple modification to the burst mode already present in today's SDRAM modules. Using embedded information about the shape of data in the SDRAM, a sequence of data that is packed can be streamed back to the processor through the memory controller.

Figure 6 shows simulation results in cycles and number of bus requests for different data shapes. The simulations were performed for several typical stream shapes, and not the full application. In general, the simulation shows that request latency can be reduced because the SDRAM calculates the sequence of data access (in essence, prefetching data) for a transfer. Furthermore, the number of bus requests is decreased for data accesses that have large skip values. This is especially true for image tile or column access with speedups ranging from three to six. The simulation results show that, for the stream-burst cases, there are data that can be packed within the locality of the opened row address that are not necessarily located in consecutive SDRAM column addresses. For image row-access cases, data is already packed in consecutive bursts so speedups come mainly from reduced bus requests when the compiler can not optimize for the SDRAM burst size. For larger image buffer sizes (e.g. megapixel), the size of a row of image pixel data is larger, so there is less spatial locality of data for image column access. In that case, the SDRAM could pack fewer data in a burst, and a new SDRAM row has to be activated.

The exercise in implementing the operational procedure has uncovered several enhancements to the current design. For example, the mode register is limiting the ability to preload stream descriptors into the SDRAM. The amount of bits allocated to stream descriptors is based on bus width and there are no provisions for multiple stream descriptors. Furthermore, SDRAM module organizations (i.e. wider bus formed by narrow SDRAM chipsets) may limit stream descriptor loading. Mode registers are typically programmed once after reset which does not support the ability to toggle between normal and stream-burst modes.

Other design alternatives are being considered such as using additional pins to initialize stream descriptors registers and enable stream-burst mode. For example, a set of pins using I²C transfer protocol can be used exclusively to initialize stream descriptor in lieu of the *load_mode_register* programming. These pins are already available in higher end DDR memories.

V. CONCLUSION AND FUTURE WORK

This paper introduces the notion of embedding information about the shape of data along with storage in order to improve the access time. The data shape is encoded as stream descriptors which are stored in physical memories such as SDRAM. When a stream-burst mode is enabled, the SDRAM can burst data back as a data stream, which

consists of packed data in the order requested by the processor. This concept extends the notion of prefetching data by placing the address calculation in physical storage rather than the processor, DMA unit, or memory controller. With data prefetching, system performance becomes dependent on average memory subsystem's bandwidth with less sensitivity to peak latency to access a data element. Furthermore, with stream-burst functionality, the total bus traffic is reduced and large caches are not necessary. This also can potentially reduce energy consumption.

An initial design based on standard SDRAM structure is presented to enable stream-burst functionality, which shows promise that stream-burst functionality can be implemented. Future work will include other design considerations to facilitate initialization and use of the new burst mode. Other SRAM memory structures can also be considered for stream-burst. Compiler optimizations to better manipulate stream descriptors to match the capabilities of the memory are being considered. For example, the compiler can generate different stream descriptors based on how the memory chips are interleaved and how a data stream is mapped onto multiple pages in memory. An enhanced memory controller is also being designed to better couple with the memory in burst mode. With stream-burst, stream buffers in the memory controller and SoC can be simplified. Finally, full simulations of the application benchmarks would better showcase the sustained performance benefits of this technique.

ACKNOWLEDGMENT

The authors acknowledge previous contributions by the RSVP™ design team at Motorola Labs.

REFERENCES

- [1] David A. Patterson, "Latency Lags Bandwidth," Communications of the ACM, vol.47, no.10, pp.71-75 October 2004.
- [2] Parthasarathy Ranganathan, Sarita Adve, Norman P. Jouppi, "Performance of image and video processing with general-purpose processors and media ISA extensions," *Proceedings of the 26th Annual International Symposium on Computer Architecture (ISCA'99)*, May 1999, pp. 124-135
- [3] Saman P. Amarasinghe; William Thies, "Architecture, languages and compilers for the Streaming Domain," PACT 2003 Tutorial., <http://cag.lcs.mit.edu/wss03/>.
- [4] S. Chiricescu, R. Essick, B. Lucas, P. May, K. Moat, J. Norris, M. Schuette, and A. Saidi, "The Reconfigurable Streaming Vector Processor (RSVP™)," *Proceedings of the 36th International Symposium on Microarchitecture*, December 2003, pp. 141-150.
- [5] S. M. Chai, S. Chiricescu, R. Essick, A. López-Lagunas, B. Lucas, P. May, K. Moat, J. Norris, M. Schuette, "Streaming Processors for Next Generation Mobile Imaging Applications," *IEEE Communications Magazine*, pp. 84-89, Dec 2005
- [6] S. M. Chai and A. López-Lagunas, "Streaming I/O for Imaging Applications", *IEEE International Conference on Computer Architecture for Machine Perception*, July 2005, pp. 178-183.
- [7] A. López-Lagunas and S. M. Chai, "Memory Bandwidth Optimization through Stream Descriptors", Memory Performance: Dealing with Applications, Systems and Architecture (MEDEA) Workshop, September 2005, pp. 59-66.
- [8] S. Palacharla, R.E. Kessler, "Evaluating Stream Buffers as a Secondary Cache Replacement", *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pp. 24-33, April 1994.
- [9] S. A. McKee, et. al., "Dynamic Access Ordering for Streamed Computations," *IEEE Transactions on Computers*, Vol. 49, No. 11, November 2000.
- [10] L. Zhang, Z. Fang, M. Parker, B.K. Mathew, L. Schaelicke, J.B. Carter, W.C. Hsieh, S.A. McKee, "The Impulse Memory Controller", *IEEE Transactions on Computers*, pp. 1117-1132, Nov 2001.
- [11] N. Jayasena, W. J. Dally, "Streams and Vectors: A Memory System Perspective", *Workshop on Media & Stream Processing*, Dec 2004.
- [12] Michael Bedford, et al, "Evaluation of the Raw microprocessor: An exposed-wire-delay architecture for LLP and streams," *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)*, June 2004, pp. 2-14.
- [13] Scott Rixner, William J. Dally, Ujval J. Kapasi, Brucec Khailany, Abelardo López-Lagunas, Peter R. Mattson, John D. Owens, "A bandwidth-efficient architecture for media processing," *Proceedings of the 31st annual ACM/IEEE International Symposium on Microarchitecture*, November 1998, pp. 3-13.
- [14] William J. Dally; Patrick Hanrahan; Mattan Erez; Timothy J. Knight; François Labonté; Jung-Ho Ahn; Nuwan Jayasena; Ujval J. Kapasi; Abhishek Das; Jayanth Gummuraju; Ian Buck, "Merrimac: Supercomputing with streams", *Proceedings of the SuperComputing SC'03 Conference*, November 2003, Phoenix, Arizona, pp. 35-43.
- [15] Francois Labonte, Peter Mattson, Ian Buck, Christos Kozyrakis, Mark Horowitz, "The Stream Virtual Machine," *Proceedings of the 2004 Intl. Conf. on Parallel Architectures and Compilation Techniques*, Sept. 29-Oct. 3, 2004.
- [16] Y. Katayama, "Trends in Semiconductor Memories," *IEEE Micro*, vol 17, no. 6, pp10-17, 1997.
- [17] Micron Technology Inc, "Synchronous SDRAM", DataSheet for MT48LC2M32B2, January 2002, www.micron.com/dramds
- [18] R. Crisp, "Direct Rambus Technology: The New Main Memory Standard," *IEEE Micro*, vol. 17, no. 6, pp. 18-28, 1997.
- [19] M. Oskin, F. T. Chong, T. Sherwood. "Active pages: A model of computation for intelligent memory", *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pp. 192-203, 1998.
- [20] C. E. Kozyrakis et al., "Scalable processors in the billion transistor era: IRAM", *IEEE Computer*, pp. 75-78, Sept 1997.
- [21] J. Marathe, F. Mueller, T. Mohan, B. Supinski, S. McKee, A. Yoo, "METRIC: Tracking down inefficiencies in the memory hierarchy via binary rewriting", *International Symposium on Code Generation and Optimization*, March 2003.
- [22] P. Havlak, K. Kennedy, "An implementation of interprocedural bounded regular section analysis", *IEEE Transactions on Parallel and Distributed System*, vol. 2, no. 3, pp. 350-360, July 1991.
- [23] N. Bellas, S. Chai, M. Dwyer, and D. Linzmeier, "FPGA implementation of a license plate recognition SoC using automatically generated streaming accelerators," *Reconfigurable Architecture Workshop*, April 2006.
- [24] S.Chiricescu, S.Chai, K.Moat, B.Lucas, P.May, J.Norris, R.Essick, M.Schuette, "RSVP II: A Next Generation Automotive Vector Processor," *IEEE International Vehicle Symposium*, June 2005
- [25] B. Flachs, et. al., "A Streaming Processor Unit for a CELL processor," *IEEE Solid-State Circuit Conference*, 2005, pp.134-135
- [26] ARM11Reference Manual ARM_DDI_0211_F, March 2005, www.arm.com/pdfs/DDI0211F_arm1136_r1p0_trm.pdf
- [27] OpenCores Organization, "WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores", revision B.3, September 2002, www.opencores.org
- [28] R. Usselmann, "Memory Controller IP Core", January 2002, www.opencores.org

RSVP™ is a trademark of Motorola Inc. Other product names are the property of their respective owner. A patent is pending that claims aspects of items and methods described in this paper.