A Data Layout Transformation (DLT) Accelerator: Architectural support for data movement optimization in accelerated-centric heterogeneous systems

Tung Thanh-Hoang[1], Amirali Shambayati[1], and Andrew A. Chien[1, 2]

[1]Department of Computer Science, University of Chicago, Chicago, Illinois, USA
[2]Argonne National Laboratory, Chicago, Illinois, USA
{hoangt, amirali, fywkevin, achien}@cs.uchicago.edu

***Abstract*—Technology scaling and growing use of accelerators make optimization of data movement of increasing importance in all computing systems. Further, growing diversity in memory structures makes embedding such optimization in software non-portable. We propose a novel architectural solution called Data Layout Transformation (DLT) associated with a simple set of instructions that enable software to describe the required data movement compactly, and free the implementation to optimize the movement based on the knowledge of the memory hierarchy and system structure.**

The DLT architecture ideas can be applicable to both general-purpose and accelerator-based heterogeneous systems. Experiment results first show that the proposed DLT architecture can make use of the full bandwidth (>97%) of a wide range of memory systems (DDR3 and HMC) while its implementation cost is relatively low, occupying only 0.24 mm^2 and consuming 75mW at 1GHz in 32nm CMOS technology. Our evaluation of using the DLT accelerator in accelerated-based heterogeneous system across DDR3 and HMC memory shows that the DLT can enhance system performance in range of 4.6x-99x (DDR3), 4.4x-115x (HMC) which turns out 2.8x-48x (DDR3), 1.4x-39x (HMC) improvement for energy efficiency.

I. Introduction

The end of Dennard scaling has risen *power wall* challenge [1] which is a primary motivation for the design of high performance energy efficiency systems in exascale computing era. In general, system energy is composed of two major costs, one is consumed to process data and the other is spent to move/transform data in adequate destination/layout for computation. Currently, the relative energy cost of data movement over computation in CPU/GPU is significant [2].

On the other hand, minimizing energy cost of data movement becomes more critical for heterogeneous systems. In such heterogeneous systems, intensive computation can be easily accelerated by using multi-core, GPUs or ASIC accelerators etc. resulting in the extreme reduction of energy cost of computation. On the other hand, the challenges to improve system performance and energy efficiency is shifting to optimizing data movement among system components.

In reality, the memory hierarchy of commodity processor is not efficiently designed for access. For example, accessing stride data and transforming data layout via caches lead to performance reduction and energy inefficiency due to the low latency and data locality of caches. To overcome these bottlenecks of caches, scratchpad memory (or local memory) [3] has been used because it shows predictable performance, low latency and high energy efficiency. However, leveraging the advantage of local memory in system perspective would requires more investigations.

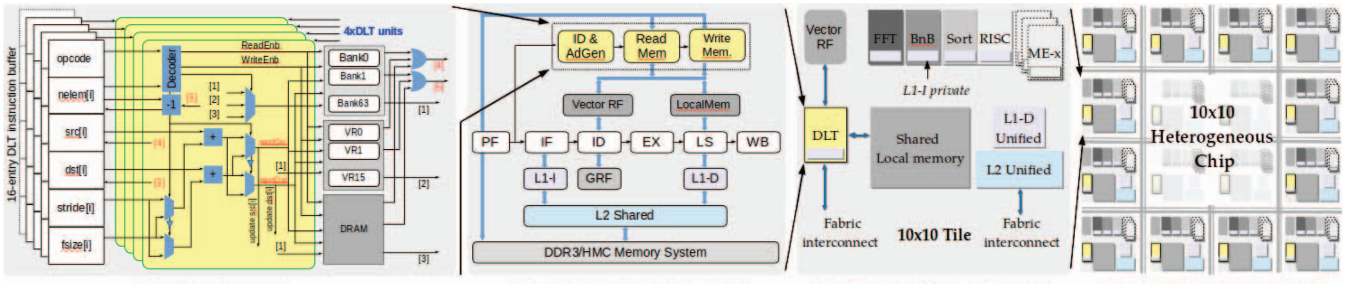
To date, there are two challenging questions regarding *the reduction of energy cost due to data movement at system level*: *i)* How to minimize the amount of moving data? *ii)* How to move data more efficiently?

There has been a number of prior works which tried to solve these questions, for example, latency avoidance techniques, waste reduction techniques (e.g. by redesigning memory systems), hybrid techniques (e.g. by aggregating DMA engines, or using gather-scatter DMA engine), and adding specialized hardware for Processing-In-Memory (PIM). Although these works can efficiently handle data movement cost, they still incur several bottlenecks such as: the miss rate of predictor of prefetcher, high design efforts and implementation cost to redesign memory system, usage overhead due to (i.e. gather-scatter DMA), and low programmability (i.e. PIM).

In this paper we propose a novel architecture called Data Layout Transformation (DLT), a specialized accelerator, which can systematically minimize data movement and move data more energy efficiently. The key points of DLT architecture are simple hardware, instruction level integration, and memory-oriented design. As a result, the DLT accelerator has high programmability and bandwidth efficiency which can mitigate the energy limits of data movement inside accelerator-based heterogeneous systems. Evaluation shows good system performance improvement and energy efficiency when the DLT accelerator is used to complement others accelerators in 10x10 architecture [4] as a case study of heterogeneous system.

Specific contributions include:

1) Microarchitecture, instruction set design and hardware implementation of the DLT accelerator that can accelerate a wide range of data movements (unit stride, non unit-stride, transposition) among storage system components.
2) Evaluation of DLT implementation in 32nm CMOS shows area only 0.24 mm^2 and low power (75mW, 1GHz). Furthermore, DLT can utilized the bandwidth of diverse memory systems (e.g. 97% and 98% bandwidth efficiency for DDR3 and HMC respectively)
3) Finally, DLT really shines in highly-accelerated systems where memory-limited execution is common. Adding DLT to the 10x10 accelerator-centric system can improve system performance by *4.6x-99x (DDR3), 4.4x-115x (HMC)* which yields *2.8x-48x (DDR3), 1.4x-38x (HMC)* improvements on energy efficiency.



(a) Mircoarchitecture (b) High-level integration of DLT (c) DLT integration in 10x10 core. (d) 10x10 heterogeneous arch.

Fig. 1: Design, integrate and federate DLT accelerator for heterogeneous accelerator-centric systems.

II. Data Layout Tranformation Accelerator

A. Microarchitecture

At the center of this work is the DLT accelerator shown in Figure 1. For high-level integration (Figure 1b), the DLT accelerator is tightly integrated with the pipeline of a simple RISC processor. Therefore, the DLT accelerator can take the advantage of instruction-level integration such as high programmability, low usage overhead (e.g. reduce performance cost to access system components) [1].

In microarchitecture level (Figure 1b), the key component of DLT accelerator is DLT lanes. Each lane is composed of a buffer that keeps DLT instructions, address generation unit and a simple decoder. Each entry of buffer is 99 bits length which consists of 3-bit instruction opcode, 4-byte source address, 4-byte destination address and 4-byte DLT descriptor thus the size of buffer is 200 bytes in total. The address generation unit contains small size adders/subtracters which are used to calculate new nextSrc, nextDst address and decrease of nelem[i] (as the bookkeeper of a gather-scatter instruction) respectively. These values are used to update buffer's fields under the control of ReadDone/WriteDone signal issued when data is completely read/written from source/to destination address. The decoder generates enable read and write signals for memories and vector register file, and routes read/write addresses via multiplexers. When a particular DLT instruction executed by a DLT lane is completed (i.e. nelem[i] is decreasing to zero), this DLT lane is released for a new gather/scatter instruction.

For parallelism, the DLT accelerator can simultaneously execute 4 gather-scatter instructions which provides at least 256GBps and 512GBps request bandwidth to DDR3 and HMC respectively. Therefore, the DLT accelerator can saturate the peak bandwidth of memory systems while minimizing hardware implementation cost.

B. Software Interface

To make use of the DLT accelerator, we develop C-intrinsics that map directly to DLT instructions (see Table I). The *dlt_form_descriptor* intrinsic compacts the number of elements needed for gather-scatter, element stride and element size into a 32-bit descriptor. The *dlt_gather* and *dlt_scatter* intrinsics support gathering/scattering data between off-chip and local memory. The source and destination address pointers are provided via general purpose registers thus it can support structural datatypes if its size can be described by DLT's descriptor. Notice that, the address space of local memory is mapped into physical memory address thus *dlt_gather* and *dlt_scatter* intrinsics can be used to rearrange data (e.g. transposition) within the local memory when both source and destination address locate in the local memory. Designed in the same fashion, two other intrinsics *dlt_vgather* and *dlt_vscatter* can gather-scatter data between the local memory and wide vector register file.

We provide a set of memory synchronization (fencing) instructions in that *dlt_gather_fence*/*dlt_scatter_fence* instruction can fence only memory instructions of RISC until all concurrent gather and scatter instructions are completed. Higher priority *dlt_fence* instruction will fence all types of memory instruction issued by either RISC processor or DLT accelerator. Meanwhile *dlt_flush* instruction allows flushing data through cache hierarchy to off-chip memory before it can be gathered/scattered thus memory consistency is guaranteed.

To this end, by design an efficient ISA which is mapped to flexible C-intrinsic functions, our DLT accelerator can offer high programmability.

C. System-level Integration

The energy inefficiency of data movement becomes critical for accelerator-based heterogeneous systems where memory-limited execution is common [2]. We have integrated the DLT accelerator into 10x10 architecture [4], as a case study of federated accelerator heterogeneous systems. The ultimate goal of 10x10 is to bring both programmability and energy efficiency for accelerators-centric heterogeneous systems. The DLT accelerator can optimize data movement to complement other compute-intensive accelerators of the unicore and multicore 10x10 architecture shown in Figures 1c and 1d.

III. Methodology and Experiments

A. System Simulation Platform

We build a cycle-accurate simulator by integrating both commercial and open-source tools. The core-processor, register file, BnB and DLT, accelerators are designed in LISA language which can be compiled into synthesizable RTL using Synopsys Processor Design toolchain. Others accelerators, FFT and MergeSort, are designed in RTL for high performance then integrated into system simulator via LISA wrapper for cycle-accurate simulation. Our simulator provides instruction and cycle counts that then calibrate using the power system component extracted from the synthesis using TSMC-based 32nm cell library.

Regarding system power modeling, we estimate cache power and timing using Cacti 6.5 [6] and using DRAMSim2 [7] to model DDR3 memory system2 in 32nm process. In

[1]Notice that DLT setup takes only ∼ 4 cycles via simple *dlt form descriptor* while the loosely-coupled accelerators such as EDMA [5] incurs ∼300 cycles.

TABLE I: Micro-Instruction Set Architecture (ISA) of the DLT accelerator.

Intrinsic	Instruction	Functional description
int dlt_form_descriptor(int nelem, int stride, int fsize)	FORMDESC R1 R2 R3	Pack (number_of_elems, stride, elem_size) into a 32-bit descriptor
void dlt_gather(ptr* dstA, ptr* srcA, int desc)	GATHER R1 R2 R3	Gather data described by *desc* from *srcA* to *dstA*
void dlt_scatter(ptr* dstA, ptr* srcA,int desc)	SCATTER R1 R2 R3	Scatter data described by *desc* from *srcA* to *dstA*
void dlt_vgather(vrDst, ptr* srcA,int desc)	VGATHER V1 R1 R2	Vector gather data described by *desc* from *srcA* to *vrDst*
void dlt_vscatter(ptr* dstA, vec vrSrc,int desc)	VSCATTER R1 V1 R2	Vector scatter data described by *desc* from *vrSrc* to *dstA*
void dlt_gather_fence()	GATHERFENCE	Fence normal Ld/St instructions until completing all concurrent gather instructions
void dlt_scatter_fence()	SCATTERFENCE	Same function as dlt_gather_fence() but for scatter instructions
void dlt_flush(ptr* addr1,ptr* addr2)	FLUSH R1 R2	Flush data in caches within address range (*addr1, addr2*)
void dlt_fence()	FENCE	Avoid execution of ALL memory instructions.

(*) The ISA of FFT, Sort, and BnB accelerators are designed in the same fashion as DLT, as parts of 10x10 architecture. /

addition, we developed in-house model for HMC memory [8] and validated the energy efficiency of DDR3 and HMC models to the published data [9].

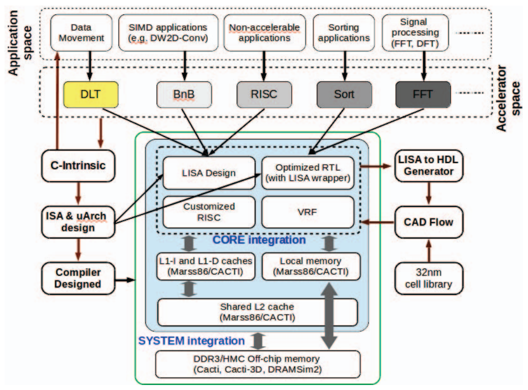

Application space
Data Movement
SIMD applications (e.g. DW2D-Conv)
Non-accelerable applications
Sorting applications
Signal processing (FFT, DFT)
Accelerator space
DLT
BnB
RISC
Sort
FFT
C-Intrinsic
ISA & uArch design
Compiler Designed
LISA Design
Optimized RTL (with LISA wrapper)
Customized RISC
VRF
CORE integration
L1-I and L1-D caches (Marss86/CACTI)
Local memory (Marss86/CACTI)
Shared L2 cache (Marss86/CACTI)
SYSTEM integration
DDR3/HMC Off-chip memory (Cacti, Cacti-3D, DRAMSim2)
LISA to HDL Generator
CAD Flow
32nm cell library

Fig. 2: HW/SW co-design flow for the 10x10 architecture.

TABLE II: Simulation platform configuration.

Parameter	Value
Process	32nm, TSMC-based
System clock rate	1Ghz
Core type	In-order, MIPS-like ISA, 5-stage pipeline
Vector register file	256B x 16 registers
Local memory	64 banks and 256B IO (4Bx16k entry/bank)
Cache hierarchy	L1-I: 32KB, 2-cycle latency, L1-D: 24KB, 2-cycle latency Shared L2: 512KB, 10-cycle latency
Main memory	
DDR3	2GB, 4-rank, 8-device, 667 Mhz, 10.6 GB/s
HMC	4GB, 4-rank, 8-device, 16 vaults, 1.25 Ghz, 128 GB/s

B. Evaluated Designs and Applications

In our evaluation, we consider the following designs, each of them can run on both DDR3 and HMC memory systems.

- **Baseline**: baseline RISC32 without any accelerator.
- **Accel** has one of the FFT, BnB or MergeSort accelerators without DLT accelerator added.
- **DLT+Accel** consists of the DLT accelerator in combination with one of the other accelerators.

Regarding benchmark, we evaluate 5 applications which belong to PERFECT bench [10] and widely used in domain-embedded systems: 2D FFT (2DFFT), Discrete Wavelet Transform (DWT), 2D Convolution (2DCon), Matrix Multiplication (MM), Merge-sort (MS).

Table III summarizes the system configurations applications mapped to the evaluated accelerators while Figure 4 shows complete HW/SW co-design flow of accelerator-centric 10x10 architecture and the mapping capability of DLT.

IV. Experimental Results

A. Hardware Implementation

In order to evaluate the cost of integrating DLT, we design the DLT accelerator in LISA, compile into Verilog by using HDL Generator and synthesiz with 32nm cell library and

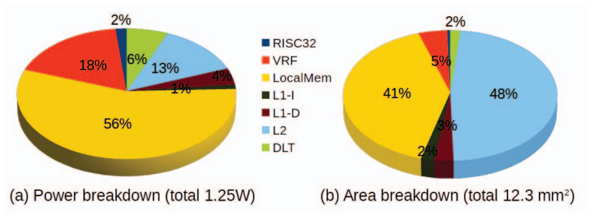


Fig. 3: HW evaluation of processor-integrated DLT accelerator.

Design Compiler. The area and power breakdown of DLT compared to other system components are shown in Figure 3. In the bottom line, the DLT accelerator can operate at 1GHz and take only 6% (75mW) of system power and 2% (0.246 mm^2) of system area which are relatively low costs for integration.

B. Bandwidth Efficiency of DLT

We study the memory bandwidth efficiency achieved by the DLT accelerator with DDR3 and HMC memory systems. By varying *stride* we can evaluate the impacts of memory array organization. Meanwhile, changing *fsize*, as the payload of DLT request, can expose the limit of memory interface. From evaluation result, we observed that DLT can achieve bandwidth efficiency up to 97% and 98% for DDR3 and HMC memory systems respectively.

C. Performance and Energy Efficiency of DLT in 10x10

Figure 4a shows the performance of 10x10 architecture using the DLT accelerator. For DDR3 memory, using DLT (**DLT+Accel**) yields 4.6x-99x compared to system without DLT support (**Accel**). When high bandwidth HMC system is used, DLT can deliver further performance improvement in range of 4.4x-115x.

As shown in Figure 4b, **DLT+Accel** design achieves energy efficiency as mush as 2.8x-48x and 1.4x-39x higher than **Accel** which is not supported by DLT. Notice that the energy efficiency of **DLT+Accel** design with HMC memory is less than ones with DDR3 since HMC is more energy efficient than DDR3 thus the energy cost of data movement with HMC, as a fraction of total system energy, is smaller.

Looking at evaluated benchmarks, MatrixMult shows lowest performance and energy improvements than the others because it is highly compute-intensive thus benefits of DLT is decreasing in both memory systems. In contrast FFT and MergeSort are data-intensive applications thus the DLT accelerator can provide highest performance and energy efficiency.

V. Related Work

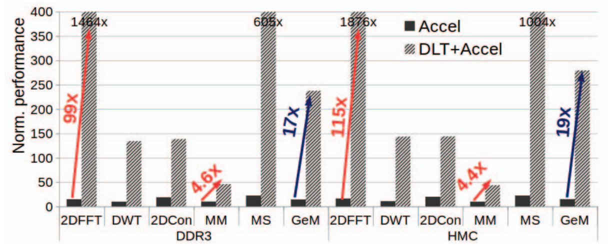
In this section, we consider the DLT accelerator in the context of related works divided into the following categories

Latency avoidance techniques avoid movement latency by early fetching data [11, 12] or rapidly transforming data

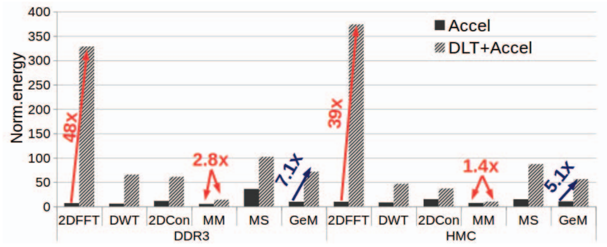
TABLE III: Evaluated designs and benchmarks (LM=Local Memory, MM=Main Memory and VR=Vector Register)

Application *(1)*	Workload size	Type of data movement	Evaluated designs	
			Accellerator only	DLT+Accellerator
2DFFT *(2)*	4kx4k, fixed 16-bit sample	Gather/scatter data between LM and MM, and transpose data inside LM	FFT (64-point kernel)	DLT+FFT
DWT	1080x1920, int	Gather/scatter data between LM and MM, and transpose data inside LM	BnB (2048-bit vector)	DLT+BnB *(3)*
2DCon	1080x1920, int	Gather/scatter data between LM and MM	BnB	DLT+BnB *(3)*
MS	1k streams, 1k int/stream	Gather/Scatter data between LM and MM	Sort (4-way x 16 lanes)	DLT+Sort
MM	4kx4k, int	Gather/Scatter data between LM and MM, LM and VR	BnB	DLT+BnB

(1) Tile-based implementation is used to exploit local memory for fast streaming data to accelerators by using DLT.
(2) Benchmark 4kx4k 2D-FFT is built by scaling 4k 1D-FFT function shown in Listing 1.
(3) Data movement latency using DLT can be overlapped by acceleration execution time to futher improve system peformance and energy efficiency.



(a) Performance, normalized to **Baseline** (GeM = Geometry Mean).



(b) Energy effiency, normalized to **Baseline** (GeM = Geometry Mean).

Fig. 4: Performance and energy efficiency of DLT accelerator in 10x10 architecture with DDR3 and HMC memory systems.

layout (e.g leveraging multithreading [13, 14]) However, the performance and energy efficiency of these approaches suffer from the miss rate of predictor (for prefetchers) or the long latency and low bandwidth utilization of caches (for multi-threading architectures). Futhermore, the high cost of hard-ware predictors, address generator to support arbitrary access patterns may reduce the energy efficiency of system.

Waste reduction techniques eliminate moving unneces-sary data between system components in order to improve performance and energy efficiency. These techniques require redesigninf array organization [15, 16] or interface of memory systems (e.g. package-based protocol in HMC [8])

Hybrid techniques combine latency avoidance and waste reduction mechanisms For example, Cell processor [17] can aggregate the bandwidth of multiple DMA engines to ac-celerate data movement. Another approach is gather-scatter DMA [5] various data movement types (e.g. gather-scatter data for caches) with flexible descriptors. Very recent work [18] proposed a data reorganization accelerator integrated into logic die of 3D-stacked memory. However, this approach requires high modification in logic layer. Furthermore, compared to the proposed DLT , this work does not support gather/scatter for wide vector which can limit performance improvement in SIMD applications.

VI. Summary and Future Work

This paper presented Data-Layout-Transform (DLT), a memory-oriented, highly programmable accelerator architec-ture for optimizing data movement across system components. The proposed DLT accelerator has relative small area and achieves high bandwidth utilization for varied memory systems (e.g. 97% and 98% for DDR3 and HMC respectively). More-over, it can complement other compute-intensive accelerators of 10x10 heterogeneous architecture to improve both system performance and energy efficiency as much as, in geometry mean, 17x (DDR3), 19x (HMC) and 7.1x (DDR3), 5.1x (HMC) respectively.

Interesting future works include: compiler support to detect the code spots to apply DLT intrinsics, support 2D transpose in single instruction or emsemble multiple DLT accelerators.

VII. Acknowledgment

Funding of this work was provided in part by the DARPA-PERFECT under award HR0011-13-2-0014 and the NSF under award NSF OCI-10-57921.

References

[1] H. Esmaeilzadeh et al. “Dark silicon and the end of multicore scaling”. In: *ISCA*. 2011.
[2] B. Dally. *NVIDIA’s Path to ExaScale*. GPU Tech., SC. 2014.
[3] S. Kayla et al. “Efficient HPC Data Motion via Scratchpad Memory”. In: *DISCS*. 2012.
[4] A. A Chien et al. “10x10: A General-purpose Architectural Approach to Heterogeneity and Energy Efficiency”. In: *ICCS*. 2011.
[5] Texas Instruments. *TMS320C6678 Memory Access Performance*. Tech. rep. 2011.
[6] N. Muralimanohar et al. *CACTI 6.0: A Tool to Model Large Caches*. Tech. rep. HP Lab., 2009.
[7] P. Rosenfeld et al. “DRAMSim2: A Cycle Accurate Memory System Simulator”. In: *Computer Architecture Letters* (2011).
[8] *Hybrid Memory Cube Spec. v1.1*. Tech. rep. 2014.
[9] B. Shekhar et al. “Exascale Computing - A Fact or a Fiction ?” In: *Technical talk at IPDPS*. 2013.
[10] *PERFECT benchmark*. Online, http://hpc.pnl.gov/PERFECT/.
[11] T. C. Mowry et al. “Design and Evaluation of a Compiler Algorithm for Prefetching”. In: *ASPLOS*. 1992.
[12] V. Jiménez et al. “Making Data Prefetch Smarter: Adaptive Prefetching on POWER7”. In: *PACT*. 2012.
[13] V. Andrey. *Multithreaded Transposition of Square Matrices with Common Code for Intel Xeon Processors and Intel Xeon Phi Co-Processors*. Download link http://research.colfaxinternational.com. 2013.
[14] I.-J. Sung et al. “In-place Transposition of Rectangular Matrices on Accelerators”. In: *PPoPP*. 2014.
[15] D. H. Yoon et al. “Adaptive Granularity Memory Systems: A Tradeoff Between Storage Efficiency and Throughput”. In: *ISCA*. 2011.
[16] D. H. Yoon et al. “The Dynamic Granularity Memory System”. In: *ISCA*. 2012.
[17] S. Williams et al. “Scientific Computing Kernels on the Cell Processor”. In: *Int. J. Parallel Program.* (2007).
[18] B. Akin et al. “Data Reorganization in Memory Using 3D-stacked DRAM”. In: *ISCA*. 2015.