SQRL: Hardware Accelerator for Collecting Software Data Structures

Snehasish Kumar, Arrvindh Shriraman
School of Computing Science
Simon Fraser University
{ska124,ashriram}
@cs.sfu.ca

Vijayalakshmi Srinivasan
IBM Research
viji@us.ibm.com

Dan Lin, Jordon Phillips
School of Computing Science
Simon Fraser University
{lindanl,jjp14}@cs.sfu.ca

ABSTRACT

Software data structures are a critical aspect of emerging data-centric applications which makes it imperative to improve the energy efficiency of data delivery. We propose SQRL, a hardware accelerator that integrates with the last-level-cache (LLC) and enables energy-efficient iterative computation on data structures. SQRL integrates a data structure-specific LLC refill engine (Collector) with a compute array of lightweight processing elements (PEs). The collector exploits knowledge of the compute kernel to i) run ahead of the PEs in a decoupled fashion to gather data objects and ii) throttle fetch rate and adaptively tile the dataset based on the locality characteristics. The collector exploits data structure knowledge to find the memory level parallelism and eliminate data structure instructions.

1. Introduction

Emerging applications in a diverse set of fields depend on iterative computation over large data structures [4]. Interestingly, the main benefits of parallelizing such applications come from fetching multiple data objects simultaneously and hiding long memory latencies. Unfortunately, the load/store interface of general-purpose processors do not scale to exploit the available memory bandwidth.

We propose *SQRL*, an accelerator that integrates with the last-level-cache and enables energy-efficient iterative computation on data structures. Supporting iterative computation requires *SQRL* to adapt to the locality requirements of the compute kernel and effectively supply data as the kernel streams over the entire dataset. Inspired by the decoupled access-execute paradigm [6], *SQRL* partitions iterative computation on data structures into two regions: the data collection and the compute kernel. *SQRL* employs a data structure-specific controller (called the Collector) that traverses the data structure, fetching the elements and staging the corresponding cache lines in the LLC until the array of Processing Elements (PEs) consume them. The PEs operate on the data objects implicitly supplied by the collector. *SQRL* is aware of the compute iterations and appropriately tiles the data to fit in the LLC to prevent thrashing. Similarly, the collector is aware of the organization of the data structure and runs ahead of the compute kernel so as to prefetch the objects to hide memory latency. The collector stalls if the LLC starts getting filled with locked lines to ensure the LLC does not evict data which has not been consumed by the compute kernel.

PACT'14, August 24–27, 2014, Edmonton, AB, Canada.
Copyright 2014 ACM 978-1-4503-2809-8/14/08 ...$15.00.
http://dx.doi.org/10.1145/2628071.2628118.

2. *SQRL*: Hardware accelerator for collecting software data structures

To assist the acceleration of iterative computation on data structures, we have developed a software runtime to construct a load slice and store slice which include information on the type of data structures, the location of the data structure and the object indices needed within an iteration. Our benchmarks are implemented using C++ STL (Standard Template Library) and use templated iterators. We replace the iterators with calls to the SQRL's C++ runtime environment. We illustrate SQRL's operation using Vector Collectors for the Blackscholes benchmark. Each iteration in Blackscholes (Figure 1) reads 5 vector elements of type float, one vector of type char and writes into a float vector. The load slice of Blackscholes consists of the following vectors, spot[],strike[],rate[],volatility[], time[] and type[] and the store slice consists of price[]; all vectors employ a unit stride.

In *Blackscholes*, each of the vectors are accessed with a unit stride i.e., in the i^{th} iteration the i^{th} element is accessed. The `Keys/Iter` is a single entry with offset 0. The descriptor limits the number of vectors accessed per-loop iteration (8 in this paper). The descriptor for spot[] in *Blackscholes* would be: <LD,&spot[0], FLOAT (4 bytes), nOptions (length of vector), [0] (unit stride access) >.

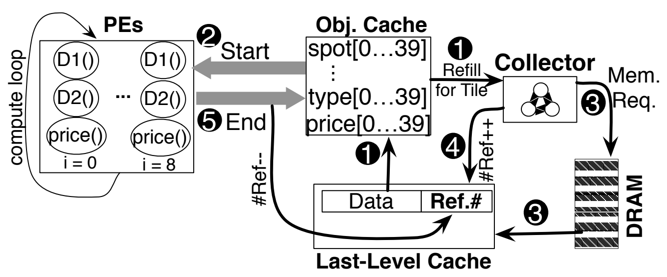
The loop is unrolled with each PE executing a single iteration of the compute kernel; PEs sync up after each iteration. Each iteration requires space for 4×5 (floats) + 1 (char) + 4 (float) = 25 bytes. The 1 KB *OBJ-$* can thus hold data for 40 iterations of the *Blackscholes* kernel (a loop tile). As shown in Figure 1, the Collector issues an asynchronous request for data (❸). Once the data is filled into the LLC, the Ref. Counter is set (❹) to ensure that the data is not evicted prior to consumption by the PEs. The collector pushes the required data into the *OBJ-$* (❶) and starts a wavefront (set of iterations) to commence execution (❷). Each wavefront consists of 8 concurrent iterations (as *SQRL* has 8 PEs). For *Blackscholes*, there 5 wavefronts are executed until all the data in the *OBJ-$* is consumed and a refill (❶) is required. The dirty data in the *OBJ-$* is written back to the LLC and the Ref. Counter is unset (❺).

3. Evaluation

We evaluate SQRL using a detailed cycle-accurate x86 simulator and the parameters are summarized in Table 1. We use Macsim [1] to model the out-of-order core(footnote), GEMS [2] to model the cache hierarchy, and DRAMsim2 [5] to model the main memory of the system. We added support for SQRL's cores into Macsim, and model the state machine of SQRL's collectors. We model core and

```
begin blackscholes
    // Initialize collectors
    c_spot = new coll(LD,&spot,FP,#length,0,VEC)
     ...
    c_time = new coll (LD,&time,FP,#length,0,VEC)
    // Run collectors in step.
    group.add (c_spot...c_time);

   // Unroll to # of PEs. Tile based on Obj.$ size.
   start(kernel(),#iterations)
end
```



Left: Blackscholes kernel modified to work with *SQRL*. ❶ Loop tile refill from LLC triggered every 40 iterations as *OBJ-$* can hold 40 iterations worth of data. ❷ Execution proceeds in tiles. In Blackscholes a tile of 40 iterations executed on the PE as 8 unrolled iterations at a time. ❸ Memory request issued by collector to fetch the cache blocks corresponding to the data structure. LLC refills from memory. ❹ collector locks line by setting Ref.# to number of elements needed in the iteration. ❺ When the compute iteration completes, the PE decrements the reference count to release the cache lines. Ref.#: 6bit field.

Figure 1: Blackscholes Execution on *SQRL*.

PE energy using McPAT using 45nm technology, and use CACTI [3] to model energy of the caches. The energy consumption of SQRL components are modeled with the templates for the register files, buffers, and ALUs from the Niagara 1 pipeline.

We evaluate the efficacy of *SQRL* on a set of 6 benchmarks, *Blackscholes, Datacube, BTree, HashTable, TextSearch, Recommender*. The workloads include multiple iterative regions and we profile the entire run apart from the initialization phase.

Table 1: System parameters

Cores	2 GHz, 4-way OOO, 96 entry ROB, 6 ALU, 2 FPU
L1	64K 4-way D-Cache, 3 cycles
LLC	4M shared 16 way, 4 Banks, 20 cycles Directory-based MESI
Energy Params	L1 (100pJ Hit). LLC (230pJ hit) L1-LLC link (6.8pJ/byte), LLC-DRAM (62.5 pJ per byte)
***SQRL* Components**	
	2GHz. 8 PEs, 4 stages in-order at 2 Ghz, 1 FPU and 1 ALU (per PE) Instruction buffer (1KB, 256 entries) INT RF: 32 entries, FP RF: 32 entries
	Obj-$ (1KB fully-assoc. sector cache, 32 tags). 1 cycle
***SQRL* Area Overhead** (at 45nm using McPAT)	
SQRL	24.099mm^2 (8 PEs)
SQRL w/o FPU	5.46mm^2

3.1 *SQRL* vs. OOO Baseline

Result 1: *SQRL* improves performance of the kernel by 18× compared to a general purpose processor. Maximum speedup (BTree): 111×. Minimum speedup (Blackscholes): 7×.
Result 2: *SQRL* is able to reduce energy in the PEs relative to the OOO core by 8× on average.
Result 3: On-chip network energy is reduced on average by 2× by eliminating L1-LLC transfers.
Result 4: *SQRL*'s shallow memory hierarchy helps reduce on-chip cache access energy by 21×.
Result 5: *SQRL* is able to attain on average a 13× increase in off-chip DRAM bandwidth. Max: 102×(BTree)

4. Summary

We have focused on exploiting data structure information to provide compute accelerators and kilo-instruction processors with energy-efficient interfaces to the memory hierarchy. We developed *SQRL*, a hardware accelerator that integrates with the LLC and supports energy efficient iterative computation on software data structures. SQRL exploits information about the computation kernel and actively manages the space in the LLC to ensure that no data fetch is wasted. SQRL deploys cache refill engines that are customized for software data structures which increases the memory level parallelism, eliminates the penalty of transfers through the cache hierarchy, and removes data structure instructions.

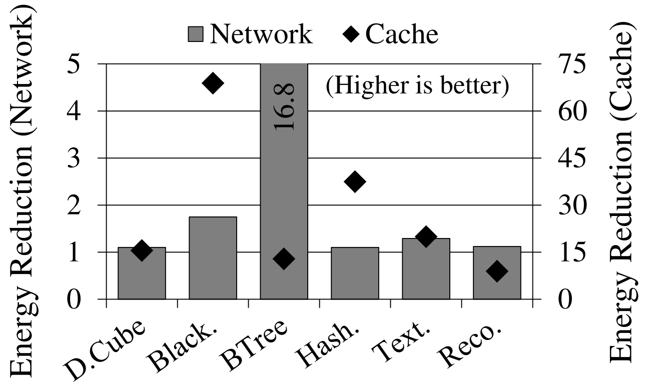


Figure 2: Energy reduction (Unit: × Times). *SQRL* vs OOO. Left Y-axis: Network energy. Right Y-axis: Cache energy.

5. References

[1] Macsim : Simulator for heterogeneous architecture - https://code.google.com/p/macsim/.

[2] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99, Nov. 2005.

[3] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In *PROC of the 40th MICRO*, 2007.

[4] P. Ranganathan. From Micro-processors to Nanostores: Rethinking Data-Centric Systems. *Computer*, 44(January):39–48, 2011.

[5] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. Dramsim2: A cycle accurate memory system simulator. *Computer Architecture Letters*, 10(1):16 –19, jan.-june 2011.

[6] J. E. Smith. Decoupled access/execute computer architectures. In *25 years of the international symposia on computer architecture (selected papers)*, 1998.