Hash Table Scalability on Intel PIUMA

Balasubramanian Seshasayee Joshua Fryman Ibrahim Hur
Intel Corporation, Hillsboro, OR, USA
Email: {bala.seshasayee, joshua.b.fryman, ibrahim.hur}@intel.com

***Abstract*—The Intel PIUMA (Programmable and Integrated Unified Memory Architecture) is a scalable, massively multi-threaded architecture designed to operate on unstructured data, with a global address space, fine-grain memory access and various novel features for latency hiding during data movement. Hash tables are a commonly used data structure with unstructured data, hence it is imperative that the performance and scaling for hash table usages are optimized for this architecture. We study three different hash table implementations on a PIUMA simulator to show that a dual-atomics based implementation, a unique feature in PIUMA, performs competitively both at larger scales and under hash collisions. Our implementations are able to achieve strong scaling up to 16,384 hardware threads.**

***Index Terms*—Hash table, scalability, graph analytics**

I. Introduction

PIUMA is designed for efficient execution of workloads operating on unstructured data, such as graph analytics. It borrows several design principles from the Cray XMT architecture [1], and comprises four 16-way threaded compute pipelines and two single threaded pipelines in each core, along with a scratchpad and a few special-purpose engines, with a scalable, hierarchical network fabric connecting the cores. Since data access latency is a key bottleneck in workloads operating on unstructured data, the architecture has several features to mitigate this latency, such as massive multi-threading, fine-grained memory access, data movement offload engines and a software managed scratchpad. Specifically, for implementing atomic update operations needed for concurrent hash table insertions, it offers the following mechanisms:

Atomics - atomic instructions on PIUMA are performed by the atomic unit, which is separate from the pipelines, allowing them to proceed asynchronously where possible.

Dual atomics - PIUMA supports a family of operations termed dual-atomics in its instruction set, wherein an additional operation such as a load or store is carried out in conjunction with an atomic operation. This avoids a round trip between back to back operations, as is common in hash table insertions. However, the two addresses that the instruction operates on, must reside in the same cache line.

Queue engines - Hardware managed queues are available for software use, by configuring them using programmable registers and to enqueue and dequeue data atomically, using special instructions. The pipeline can make independent progress by offloading queueing to these engines.

This research was, in part, funded by the U.S. Government. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

II. Hash table Implementations

Workloads operating on unstructured data, like graph algorithms (e.g., graph2vec [2], TIES [3], SpGeMM [4]) make extensive use of hash tables. Typically, these workloads' uses of hash tables occur in phases, where in one phase the hash table is built, and in a later distinct phase, its entries are looked up. In algorithms with iterative computations, the hash table is typically cleared after the completion of lookups, then rebuilt based on the next iteration of data. Given such a use-case, the implementations of concurrent insertions and lookups gain significance.

Various design choices for hash tables exist, based on the data layout, hashing function, collision resolution, among others. We choose an open addressing implementation for its lower overheads, especially at lower load factors, and a computationally simple hashing function [5]. Unlike in conventional architectures where computation is relatively cheap and data movement is limited by latency, in PIUMA the data access latency is amortized, however arithmetic computations (such as those comprising complex hashing functions) can have higher overheads due to the fine-grained 16-way multi-threading. We use linear probing to resolve collisions due to its simpler implementation, however more elaborate techniques such as Robin Hood hashing [6] can also be used with minor modifications. We consider 64-bit each key-value pairs for storage, hence insertion consists of an atomic compare exchange which if successful, is followed by a store, and an attempt to insert at the next location if not. Alternatively, these two operations can also be performed in one shot with a dual atomic instruction. A third option, which uses the queue engine to serialize insertions into a hash table, is found to be less performant than atomics and dual-atomics. Lookups are very straightforward, based on linear probing and are not explored in this paper due to space constraints.

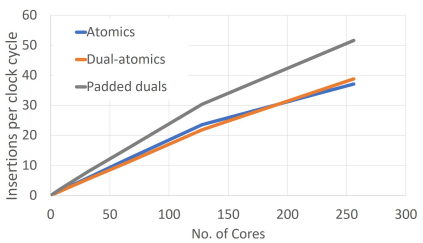

Fig. 1: Performance under no contention

978-1-7281-9219-2/20/$31.00 ©2020 IEEE

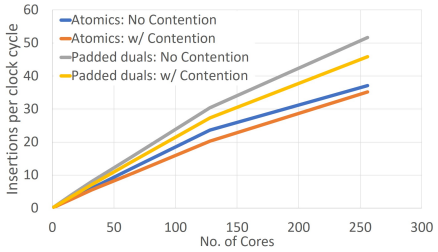

Fig. 2: Performance under contention

III. Experimental Results

The implementations discussed in Section II are developed using the PIUMA toolkit, a software library that enables SPMD programming on PIUMA, with a custom LLVM-based compiler. The resulting program is executed on an in-house version of Sniper [7] developed for this architecture (and validated against the PIUMA RTL models for accuracy), to gather execution and instruction timing, bandwidth utilization, among other statistics.

PIUMA is designed for high data throughput to amortize the latency of operating on unstructured data, with the memory and network bandwidths provisioned according to the number of processing units. Scalability in this architecture is achieved by saturating the memory bandwidth. For hash table insertions using the atomic operation, the insertion loop has the equivalent of 4 memory loads every 13 instructions. This is sufficient to saturate the bandwidth, since the per-core memory bandwidth is provisioned for a memory load per 4 instructions.

Dual atomics save bandwidth by avoiding a round trip between the pipeline and the remote memory controller. However, their encoding in PIUMA requires the addresses to be specified as offsets from cacheline boundary, adding overheads in organizing the operands to the instruction. These overheads could impact the gains arising from avoiding the round trip. In an improvement to dual-atomics, termed *padded duals*, the hash table entries are pad-aligned to cacheline boundaries, to ensure the offsets in the instruction encoding are always zero, but also increasing the size of the hashtable as a consequence. This change results in a shorter instruction sequence in the critical path, at the expense of extra unused space between hashtable entries.

For our evaluations, we perform concurrent insertions of uniformly distributed keys from all multi-threaded pipelines into a 4GB hashtable until it gets half-full, and measure the insertion rate while varying the number of cores from 1 to 256 (a PIUMA node). With 64 threads enabled on each core, a node comprises 16,384 threads. Fig. 1 compares the performance of atomics against dual-atomics, as well as the padded duals implementation. Atomics narrowly outperform dual-atomics at lower scales, as the overheads involved in emitting the right type of dual atomics instruction outweigh the gains from avoiding a memory round trip. At the node level the trend reverses: with the hash table striped across the two memory controllers available at this scale, the gains due to avoidance of a round trip with dual atomics on remote memory controllers start to be substantial. Padded duals outperform both these implementations at all scales. Introducing hash collisions ($\sim 12\%$) to this setup (Fig. 2) results in a mostly uniform performance drop arising from contention, regardless of the implementation. However, dual atomics generate a lower bandwidth utilization despite the contention.

Finally, we study the scalability of hash insertions in a practical workload, *graph2vec*, using a real world dataset for malware classification based on API dependency graphs. This dataset has 12M vertices and is 93MB in size. We simulate a portion of the modified version of the algorithm, where each of the various subgraphs is encoded as a 64-bit integer value and inserted into the hash table, using a prime modulo as the hash function. This workload uses a modified insertion algorithm, wherein the previously inserted value is returned in the case of a collision. The results in Fig. 3 show dual atomics to be outperforming, due to a high collision rate ($\sim 44\%$) in this dataset.

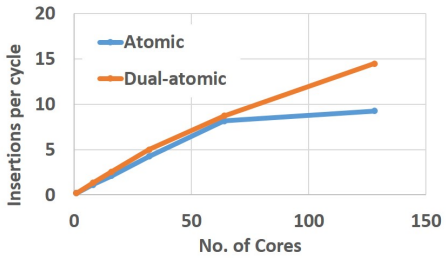

Fig. 3: Insertions with *graph2vec* on malware dataset

IV. Conclusion

Dual-atomics, a new feature of the PIUMA system, outperforms atomics-based hash table insertions at larger scales and under hash collisions, even as both forms demonstrate close-to-linear strong scaling. Dual atomics require pad-aligning hashtable entries to avoid instruction-encoding related overheads, and the use of alternative encodings to avoid these overheads is currently being explored. Future work in this direction involves extending the dual-atomics based hash table implementation to other workloads and at performance trends at scales larger than a PIUMA node.

References

[1] A. Kopser and D. Vollrath, "Overview of the next generation Cray XMT," in *Cray User Group Proceedings*, 2011, pp. 1–10.
[2] A. Narayanan *et al.*, "graph2vec: Learning distributed representations of graphs," *CoRR abs/1707.05005*, 2017.
[3] N. K. Ahmed, J. Neville, and R. Kompella, "Network sampling: From static to streaming graphs," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 8, no. 2, p. 7, 2014.
[4] Y. Nagasaka, S. Matsuoka, A. Azad, and A. Buluç, "High-performance sparse matrix-matrix products on Intel KNL and multicore architectures," in *Proceedings of ICPP*, 2018, pp. 1–10.
[5] D. Knuth, "The art of computer programming, 2nd ed." vol. 3, p. 516.
[6] P. Celis, P.-A. Larson, and J. I. Munro, "Robin hood hashing," in *Proc. of Annual Symposium on Foundations of Computer Science*, 1985.
[7] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *SC*, 2011, pp. 52:1–52:12.

978-1-7281-9219-2/20/$31.00 ©2020 IEEE