Efficient Compute at the Edge: Optimizing Energy Aware Data Structures for Emerging Edge Hardware

Amin M. Khan, Ibrahim Umar, Phuong Hoai Ha
Department of Computer Science
UiT The Arctic University of Norway
Tromsø, Norway
{amin.khan, ibrahim.umar, phuong.hoai.ha}@uit.no

***Abstract*—The advent of exascale computing, with the unparalleled rise in the scale of data in Internet of Things (IoT), high performance computing (HPC), and big data domains, both at the center and the edge of the system, requires optimal exploitation of energy-efficient computing hardware dedicated for edge processing. Emerging hardware for data processing at the edge must take advantage of advanced concurrent data locality-aware algorithms and data structures in order to provide better throughput and energy efficiency. Their design must be performance portable for their implementation to perform equally well on the edge hardware as well as other high performance computing, embedded and accelerator platforms. Concurrent search trees are one such widely used back-end for many important big data systems, databases, and file systems. We analyze DeltaTree, a concurrent energy-efficient and locality-aware data structure based on relaxed cache-oblivious model and van Emde Boas trees, on Intel's specialized computing platform Movidius Myriad 2, designed for machine vision and computing capabilities at the edge. We compare the throughput and energy efficiency of DeltaTree with B-link tree, a highly concurrent B+tree, on Movidius Myriad 2, along with a high performance computing platform (Intel Xeon), an ARM embedded platform, and an accelerator platform (Intel Xeon Phi). The results show that DeltaTree is performance portable, providing better energy-efficiency and throughput than B-link tree on these platforms for most workloads. For Movidius Myriad 2 in particular, DeltaTree performs really well with its throughput and efficiency up to 4× better than B-link tree.**

***Index Terms*—edge computing, energy efficient algorithms, concurrent algorithms, data locality, binary search trees**

I. Introduction

The trend of data intensive and data centric computing, with rising to the exascale, along with the applications in Internet of Things (IoT), smart grid, health technology and personalized medicine, are bringing the focus back to having intelligence and computing at the edges of the network. Arising from the concerns for data security, privacy, control and ownership, along with the plain impracticalities and costs involved in moving the increasing size of data around, this is also necessitated by the latency and performance requirements of applications from cloud gaming, machine vision, video analytics, cloud robotics and time critical industrial applications.

This has given rise to interesting and novel approaches in designing hardware for efficient computing and processing capabilities. One of the promising technology in this domain is the Movidius Myriad platform [1], which provides always-on low powered machine vision and deep learning capabilities for mobile and edge applications. Such edge platforms can play an important role in performing data analytics and data intelligence at the edge [2]. They can also help to increase adoption and deployment of big data technologies in volunteer computing and community edge cloud platforms [3], [4] for applications into personal health, smart homes, smart cities, and IoT.

Exploiting the full potential of such edge platforms also requires that the data structures and algorithms from traditional computing platforms adapt well to the new hardware, so that their performance remains portable. In this paper, we focus on concurrent search trees that play a crucial and central role in the back-end design of databases, key-values stores, file systems, system schedulers, big data, and many other applications. We explore and compare the performance and energy efficiency profile of Movidius Myriad platform with other state-of-the-art high-performance and personal computing platforms.

We have compared DeltaTree [5]–[7], a performance-portable fine-grained locality-aware concurrent search tree based on van Emde Boas layout, with B-link tree [8], a highly concurrent B+ tree, on the specialized computing platform Movidius Myriad 2, and also experimentally evaluated them on a high performance computing (HPC) platform (Intel Xeon), an ARM embedded platform, and an accelerator platform (Intel Xeon Phi). Our results show that DeltaTree performs well, and its energy efficiency and throughput are up to 4× better than B-link tree. This indicates the huge potential in developing energy aware data structures and algorithms, which are designed to be performance portable for different hardware architectures and memory hierarchies, as they adapt well to emerging novel hardware architectures. DeltaTree, for instance in this case initially designed for traditional HPC platforms, demonstrated optimal performance for customized edge architectures like Movidius Myriad, unlike platform-dependent search trees optimized and fine-tuned for a specific platform (e.g., FAST [9] and PALM [10]).

Our key contribution in this paper is building the case for platform independent, performance portable and energy optimized data structures, which perform well on existing HPC and accelerator platforms, but more importantly at the same time, port well to the latest hardware systems dedicated for computing at the edge. We contribute by demonstrating our

argument through detailed empirical performance evaluation by implementing concurrency-aware vEB based DeltaTree [5]–[7] on the specialized edge computing platform, Movidius Myriad 2, and also contrasting its efficiency and performance on three other popular platforms (cf. Table I). There are not many existing studies evaluating the Movidius Myriad platform in depth, and to the best of our knowledge, our work represents one of the first studies into performance evaluation and benchmarking cache oblivious data structures and concurrent search trees on Movidius Myriad.

The rest of the paper is organized as follows. We discuss the latest trends and developments in energy-efficient edge computing in Section II. We highlight the need for energy-aware data structures, and introduce the concurrent search trees in Section III. We present the experimental evaluation in Section IV, and discuss our findings in Section V. We conclude our work in Section VI.

II. Efficient Compute at the Edge

The computation at the edge is becoming increasingly critical for a number of mobile, personal and industrial applications into machine vision, deep learning, data analytics and IoT, among many other areas. As the mobile and edge devices are resource scarce, the impetus for energy efficiency has driven leading hardware and software players in the systems to focus on hardware dedicated and optimized for computing and deep learning at the edge. One of the leading player is Movidius Myriad platform for machine vision and neural computing [1], [11], which is a forerunner to Intel's recently launched Myriad X system-on-chip (SOC) platform with a dedicated neural compute engine for accelerating deep learning inferences at the edge [12]. Movidius Myriad 2 also features in Google's AIY Vision Kit [13] for running neural network models on-device. Nvidia offers Jetson TX2 module [14] for power-efficient embedded artificial intelligence processing. Arm, with its latest project Trillium [15], develops dedicated Arm Machine Learning (ML) processor and Arm Object Detection (OD) processor, along with ultra-low-power microprocessors such as the Arm Cortex-M processors, to drive machine learning to be ubiquitous at the edge. Google has also leveraged dedicated hardware, termed as TensorFlow Processing Unit (TPU) [16], for speeding-up deep learning.

Smartphone makers are also increasingly leveraging dedicated hardware for deep learning. Apple with its latest iPhone X included a dedicated neural computing processor [17]. Google in its Pixel 2 phone built in custom-designed co-processor, Pixel Visual Core [18], for image processing and machine learning tasks, which is energy-aware to avoid drain on the phone's battery. Samsung, too, with its Galaxy S9 and Galaxy S9 Plus smartphones, introduced Exynos 9 Series 9810 processor [19] with dedicated artificial intelligence processing for improving energy efficiency. This also fits in with the general purpose computing platforms well-suited for the edge like Raspberry Pi [20], Arduino, and Intel Galileo, among other platforms, which can be effectively optimized for IoT applications at the edge.

Other initiatives have focused on optimizing data processing and machine learning algorithms for energy efficient computing at the edge. This aims to enable resource-constrained IoT devices to execute machine learning algorithms locally, instead of uploading data to the cloud for processing. Microsoft, for instance, has developed optimized tree and k-nearest neighbour based algorithms for classification, regression, ranking, and other useful IoT functions [21], [22], which have been shown to perform well on Arduino Uno board with minimal energy footprint.

These developments clearly highlight the trend of moving on from general purpose computational hardware like central processing units (CPU), and graphics processing units (GPU), to dedicated hardware for machine learning and edge processing including field programmable gate arrays (FGPA), and application-specific integrated circuits (ASIC). With the advent of novel memory hardware like storage class memory (SCM), or non-volatile random-access memory (NVRAM), and near-memory and in-memory computing [23], this clearly signals towards optimizing algorithms and data structures to adapt well to these specialized hardware.

III. Energy Aware Data Structures

The different hardware systems require customized implementation of software, algorithms, and data structures, which are carefully tuned and optimized for that specific platform, in order to achieve the best performance and efficiency. Porting to a different platform, or an upgraded architecture for the same platform, can involve significant development effort in optimization. For instance, different hierarchy and capacity of L1/L2 caches and main memory's capacity and bus width can require different trade-offs in tuning the data structures for that particular architecture.

Performance portable algorithms can help in offsetting these transition and development costs. Performance portability is becoming increasingly important for high performance computing [24] in various aspects of compute, memory and I/O bound operations. In this work, we focus on performance portability across different memory hierarchies, aiming to achieve with cache oblivious data structures [25] asymptotically optimal performance across different levels of memory hierarchy.

Achieving high energy efficiency requires fine-grained data locality. Locality-aware cache oblivious model provides performance that is portable, and cache oblivious data structures and algorithms are found to be cache-efficient and disk-efficient [26], making them suitable for improving energy efficiency in modern high performance systems.

A. Optimal Concurrent Search Trees

DeltaTree [5]–[7] is a practical, platform-independent, and lock-based concurrent search tree, which is based on van Emde Boas (vEB) layout trees [27]. The main feature of the cache-oblivious vEB layout is that the cost of any search in this layout is $O(\log_B N)$ data transfers, where N is the tree size and B is the *unknown* memory block size in ideal-cache model [25]. DeltaTree implements *relaxed* cache oblivious model with the

additional restriction that an upper bound UB on the unknown memory block size B is known in advance. As long as an upper bound on all the block sizes of multilevel memory is known, the relaxed cache-oblivious model maintains the key feature of temporal data locality of the original cache-oblivious model [25].

This provides the advantage that the analysis of a cache-oblivious data structure for a simple two-level memory is applicable for an unknown multilevel memory (e.g., registers, L1/L2/L3 caches and memory). So an algorithm that is optimal in terms of data movement for a simple two-level memory is asymptotically optimal for an unknown multilevel memory. DeltaTree adopts the structure and the algorithm that is similar to the B-link tree, a highly concurrent variation of B+Tree [8], and replaces the array nodes in the B-link trees with DeltaNode. Search operations do not need to use locks or wait for any locks, while the update operations only need to lock at most three DeltaNodes [8].

B-trees and its concurrent variants [28], [29] are optimized to reduce I/O. Other concurrent search tree, like FAST [9] and PALM [10], are platform-dependent that are optimized and fine-tuned for a specific platform. BST-TK [30], meanwhile, is a platform-independent and locality-oblivious tree, which is based on the asynchronous concurrency paradigm. Locality-aware concurrent search trees provide better performance and energy efficiency [31]. DeltaTree is locality-aware as well as performance-portable, and our goal here is to explore how well it adapts to the emerging edge computing platforms like Movidius Myriad 2.

B. Movidius Myriad 2 Platform

The Myriad 2 platform [11] developed by Movidius is a highly energy-efficient ultra-low power platform [32], [33] for edge processing, which contains a total of 12 separate specialized SHAVE (Streaming Hybrid Architecture Vector Engine) processors, each of them existing on solitary power islands, which allows very fine-grained power control in software with minimal latency to return to normal operating mode. Each SHAVE has its own Texture Management Unit (TMU), and also contains wide and deep register files coupled with a Variable-Length Long Instruction-Word (VLLIW) for code-size efficiency. Data and instructions reside in a shared Connection MatriX (CMX) memory block which can accommodate different instruction/data mixes depending on the workload. The CMX also includes address-translation logic to allow VLLIW code to be easily relocated to any core in the system.

Such specialized hardware requires porting of existing algorithms and data structures to optimally exploit the strengths of Myriad 2 platform. We implemented the DeltaTree and B-link tree using the Movidius Development Kit (MDK) for Myriad. The development kit, as shown in Figure 1, provides for easier programming interfaces using C/C++, and comes with useful hardware, like cameras, interfaces and ports, for prototyping and developing machine vision and edge processing applications.

Figure 1. Movidius Myriad 2 Development Kit

IV. Experimental Evaluation

To evaluate the conceptual idea of a locality-aware concurrent tree, we compare energy efficiency and throughput of DeltaTree [5]–[7] and *CBTree* [34], an optimized implementation of the fast concurrent B-tree or B-link tree [8]. B-link tree is a highly concurrent B-tree variant, and it is still used as a backend in popular database systems such as PostgreSQL[1]. The experiments were conducted on an Intel high performance computing (HPC) platform, an ARM embedded platform, an accelerator platform based on the Intel Xeon Phi architecture (MIC platform), and a specialized computing platform based on Movidius Myriad 2 (cf. Table I).

A. Benchmark Setup

Energy efficiency metrics (in operations/Joule) were calculated by dividing the number of operations ($rep = 5{,}000{,}000$) with the total CPU and DRAM energy consumption. The ARM and Myriad 2 platforms were equipped with a built-in on-board power measurement system that was able to measure the energy of all CPU cores and memory (DRAM) continuously in real-time. For the Intel HPC platform, the Intel PCM library was used to measure the CPU and DRAM energy. Energy metrics (in operations/Joule) on the MIC platform were collected by polling the */sys/class/micras/power* interface every 50 milliseconds. *Throughput* metrics (in operations/second) were calculated by dividing the number of operations ($rep = 5{,}000{,}000$) with the time needed to finish the whole operations.

All trees were populated with a respective number ($init$) of random keys to simulate search trees that partially fit into

[1]https://github.com/postgres/postgres/blob/master/src/backend/access/nbtree/README

Table I
DELTATREE ENERGY EFFICIENCY AND THROUGHPUT WITH 4 DIFFERENT BENCHMARK PLATFORMS.

Name	HPC	ARM	MIC	Myriad2
System	Intel Haswell-EP	Samsung Exynos5 Octa	Intel Knights Corner	Movidius Myriad2
Processors	2x Intel Xeon E5-2650L v3	1x Samsung Exynos 5410	1x Xeon Phi 31S1P	1x Myriad2 SoC
# cores	24 (without hyperthreading)	– 4x Cortex A15 cores – 4x Cortex A7 cores	57 (without hyperthreading)	– 1x LeonOS core – 1x LeonRT core – 12x Shave cores
Core clock	2.5 GHz	– 1.6 GHz (A15 cores) – 1.2 GHz (A7 cores)	1.1 GHz	600 MHz
L1 cache	32/32 KB I/D	32/32 KB I/D	32/32 KB I/D	– LeonOS (32/32 KB I/D) – LeonRT (4/4 KB I/D) – Shave (2/1 KB I/D)
L2 cache	256 KB (per-core)	– 2 MB (shared, A15 cores) – 512 KB (shared, A7 cores)	512 KB	– 256 KB (LeonOS) – 32 KB (LeonRT) – 256 KB (shared, Shave)
L3 cache	30 MB (shared)	-	-	2MB "CMX" (shared)
Interconnect	8 GT/s Quick Path Interconnect (QPI)	CoreLink Cache Coherent Interconnect (CCI) 400	5 GT/s Ring Bus Interconnect	400 GB/sec Interconnect
Memory	64 GB DDR3	2 GB LPDDR3	6 GB GDDR5	128 MB LPDDR II
OS	Centos 7.1 (3.10.0-229 kernel)	Ubuntu 14.04 (3.4.103 kernel)	Xeon Phi uOS (2.6.38.8 + mpss3.5)	RTEMS (MDK 15.02.0)
Compiler	GNU GCC 4.8.3	GNU GCC 4.8.2	Intel C Compiler 15.0.2	Movidius MDK 15.02.0

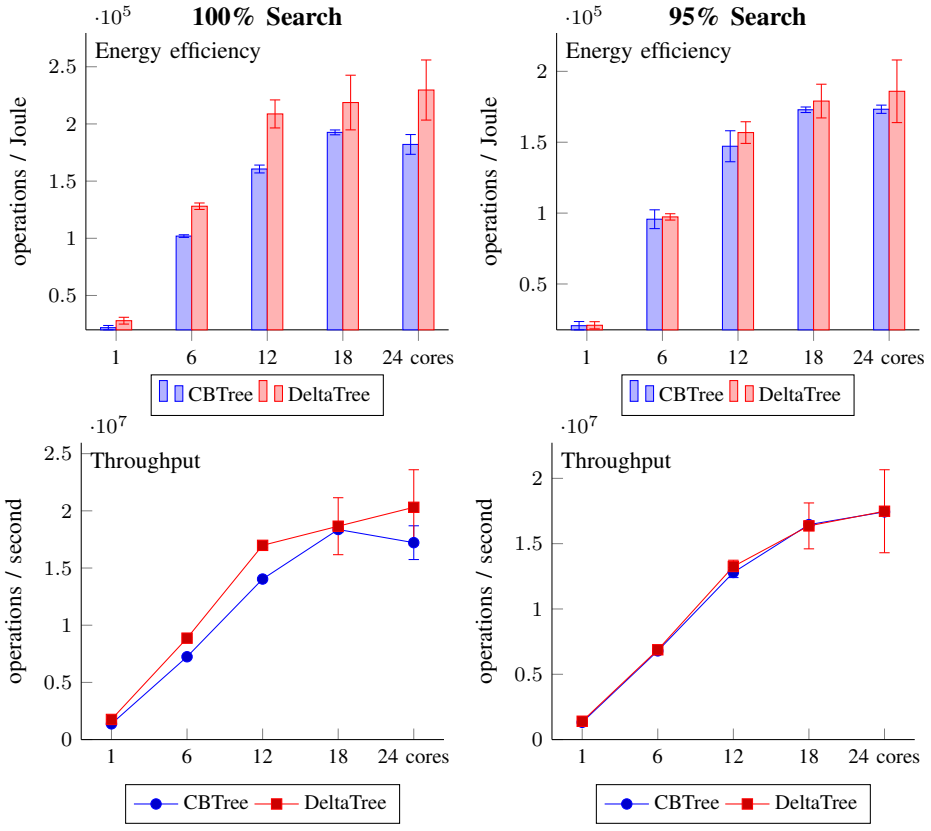

Figure 2. HPC platform. DeltaTree is up to 30% more energy efficient and up to 20% faster than CBTree.

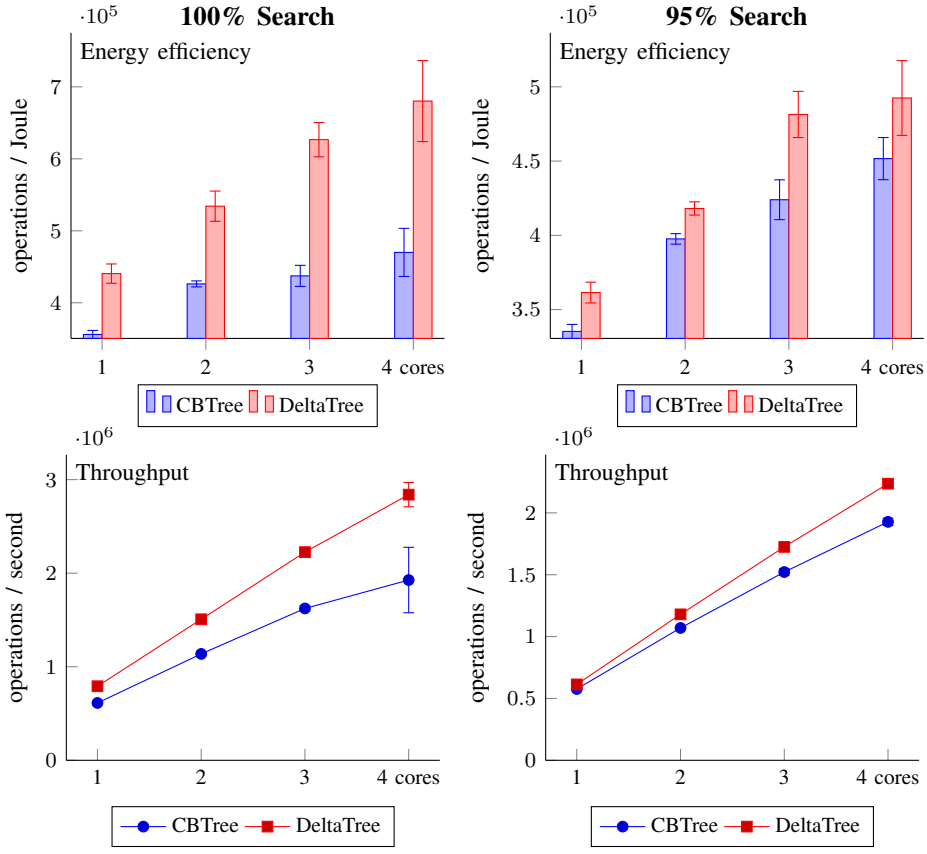

Figure 3. ARM platform. DeltaTree is up to 45% more energy efficient and up to 45% faster than CBTree.

the last level cache. The experiments on the Intel HPC and MIC platforms were using $init = 8{,}388{,}607$ initial keys, while the experiments on the ARM and Myriad 2 platforms were using $init = 4{,}194{,}303$ and $init = 1{,}048{,}575$ initial keys, respectively.

Combination of the update rate $u = \{0, 5\}$ and the number of threads $nr = \{1, \ldots, maximum\_cores\}$ were used for each experiment. An update rate of 0 means 100% search, while an update rate of 5 equals to 5% insert and delete operations out of rep operations (or in other words, 95% of the operations are search). The 95% search choice is inspired by the fact that a 30:1 GET/SET ratio is reported in Facebook Memcached workload [35]. All operations were using randomly generated values of $v \in (0, init \times 2], v \in \mathbb{N}$ as their parameter. Note that the same range was used for populating the initial keys. Pthreads are used for concurrency on HPC, ARM, and MIC platforms, and the running threads are pinned to the available physical cores.

For a fair comparison, we have set the upper bound UB on the unknown memory block size B of DeltaTree and CBTree's order to their respective values so that the page size for DeltaNodes and CBTree was within the system's page size of 4 KB. All experiments are repeated 5 times to ensure consistent results, whereas the respective operations are executed more than 1 million times in each iteration.

B. Energy Evaluation

The energy evaluation results proved that the energy efficiency of DeltaTree is better than CBTree in all of the benchmark scenarios except in the 95% search using 57-cores MIC platform. In the HPC platform (cf. Figure 2), DeltaTree is 30% more energy efficient than CBTree in the 100% search benchmark using 12 cores. For the 95% search benchmark, DeltaTree is 8% more energy efficient than CBTree when using 24 cores.

In the ARM platform (cf. Figure 3), DeltaTree is 45% more energy efficient than CBTree in the 100% search benchmark with 4 cores. DeltaTree is 10% more energy efficient than CBTree in the 95% search benchmark using 4 cores.

In the MIC platform (cf. Figure 4), DeltaTree is 20% more energy efficient than CBTree in the 100% search benchmark with 57 cores. However, in the 95% search benchmark with 57 cores, DeltaTree's energy efficiency is 10% lower than CBTree.

In the Myriad 2 platform (cf. Figure 5), DeltaTree is 4× more energy efficient than CBTree in the 100% search benchmark with 12 shaves. For the 95% search benchmark, DeltaTree is 2.5× more energy efficient than CBTree when using 12 shaves.

C. Throughput Evaluation

In the throughput evaluation results, DeltaTree managed to be faster than CBTree in all of the benchmark cases with

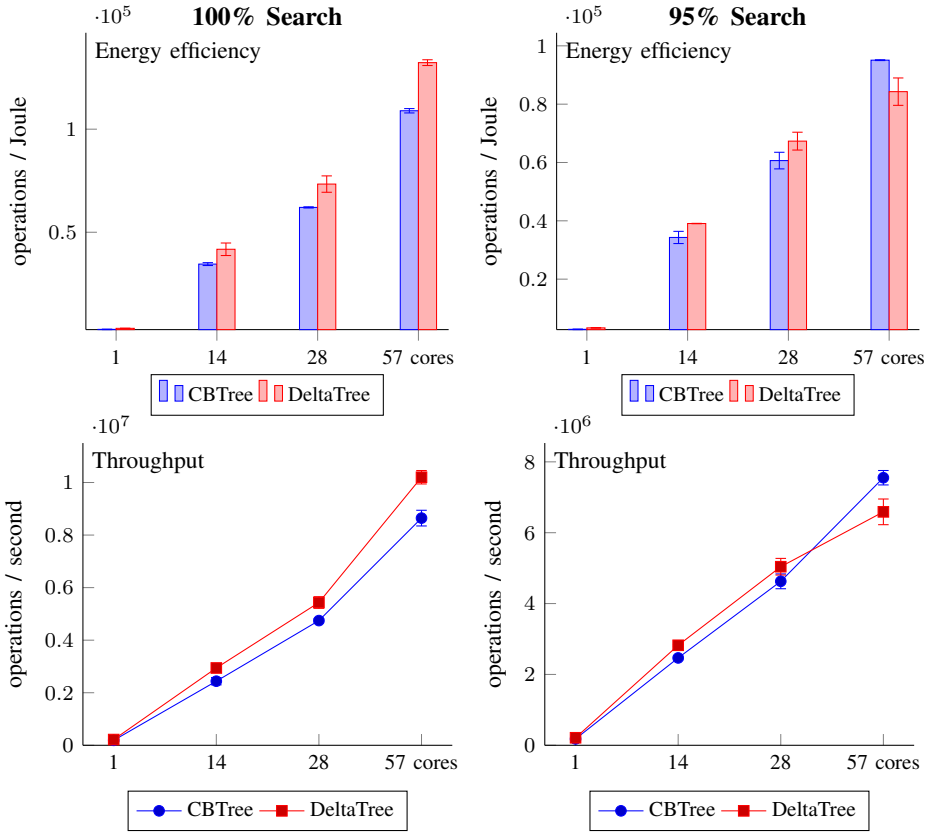

Figure 4. MIC platform. DeltaTree is up to 20% more energy efficient and up to 18% faster than CBTree. However, DeltaTree energy efficiency and throughput is 10% lower than CBTree in the 95% benchmark with 57 cores, which is caused by a false-sharing problem (cf. Section V).

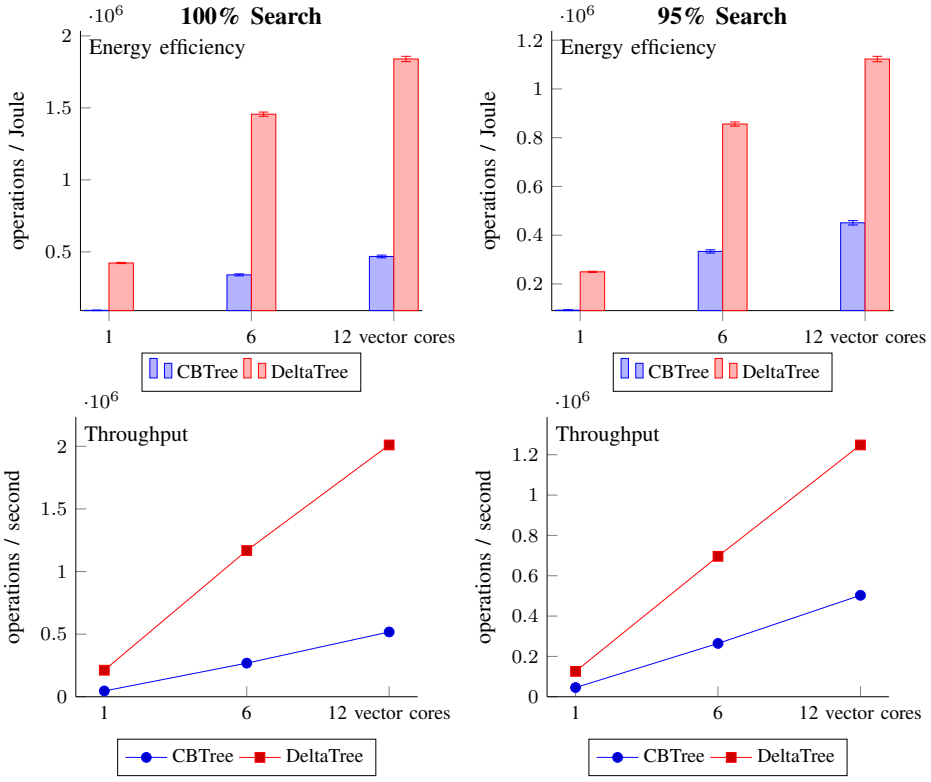

Figure 5. Myriad 2 platform. DeltaTree is up to 4× more energy efficient and 4× faster than CBTree.

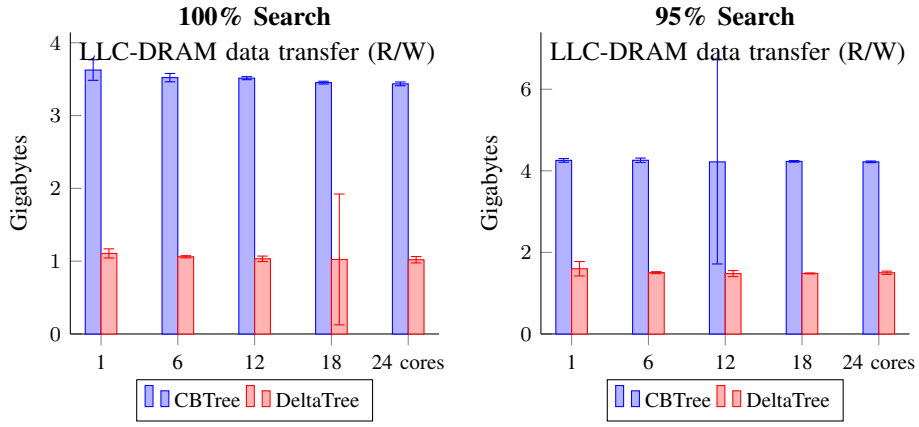

Figure 6. Data movement between CPU's last level cache (LLC) and DRAM on the HPC platform. Collected using Intel PCM library.

one exception in the 57 cores MIC platform's 95% search benchmark. In the HPC platform (cf. Figure 2), DeltaTree is 20% faster than CBTree in the 100% search benchmark with 24 cores. In the 95% search benchmark with 12 cores, DeltaTree is 5% faster than CBTree.

In the ARM platform (cf. Figure 3), DeltaTree is 45% faster than CBTree in the 100% search benchmark with 4 cores. DeltaTree is 15% faster than CBTree in the 95% search benchmark with 4 cores.

In the MIC platform (cf. Figure 4), DeltaTree is 18% faster than CBTree in the 100% search benchmark with 57 cores. Similar to the MIC platform's energy efficiency figure (cf. Figure 4), DeltaTree throughput is 25% lower than CBTree in the 95% search benchmark with 57 cores.

In the Myriad 2 platform (cf. Figure 5), DeltaTree throughput is 4× higher than CBTree in the 100% search benchmark with 12 shaves. For the 95% search benchmark, DeltaTree is 2.5× faster than CBTree when using 12 shaves.

V. Discussion

The DeltaTree energy efficiency and throughput results highlighted the improvement that the locality-aware vEB layout can bring for the concurrent search-intensive workloads in a search tree. In the 100% search benchmarks on all the available platforms, DeltaTree energy efficiency and throughput outperformed those of the CBTree's.

The reason behind DeltaTree's good energy efficiency and throughput can be explained by looking at its data transfer complexity. Figure 6 depicts the amount of data transferred between DRAM and the processors' last level cache (LLC) on the HPC platform. Based on our measurements, DeltaTree's average memory footprint on the HPC platform was 1.5× of CBTree (or 0.6 GB and 0.4 GB, respectively). Interestingly, Figure 6 shows that DeltaTree managed to lower its data traffic between DRAM and LLC up to a third compared to CBTree. Since DeltaTree and CBTree are memory-bound programs (i.e., the program's progress is bound to the speed of memory access), data transfer reduction is the primary cause of the better energy efficiency in the case of DeltaTree's operations.

Nevertheless, DeltaTree has one drawback, namely it can be inefficient and slow when doing concurrent updates using many-core CPUs (cf. Figure 4). Based on the cache profiles that we have examined from the MIC platform, false sharing is found to be the cause of this problem. This finding highlights the weakness of the cache-oblivious data structures, namely they are likely to cause false sharing in the highly concurrent environments where data updates are involved.

For Movidius Myriad 2, DeltaTree adapts very well to the constraints and the design of this edge platform for machine vision and neural computing. The improvement in energy efficiency and throughput for DeltaTree in relation to CBTree is quite significant as compared to the other platforms. The throughput and energy efficiency were 4× better than CBTree in the 100% search benchmark, and 2.5× better in the 95% search benchmark. We can build on these promising findings to evaluate the performance of DeltaTree on other platforms like GPUs (mobile and desktop grade) and FPGAs, as well as on the latest versions of Intel Xeon, Intel Xeon Phi, and Samsung Exynos platforms in the future.

VI. Conclusion

Fine-grained data locality-aware algorithms and data structures can provider better energy efficiency and performance on emerging edge hardware. As concurrent search trees underpin many important systems, any improvements in their throughput yield significant gains for big data, IoT, and edge systems. We analyzed DeltaTree, a concurrency-aware platform-independent cache-oblivious search tree, which proved performance portable and energy efficient for a variety of platforms, including HPC, embedded, accelerator, and specialized edge computing systems. Our experimental evaluation shows DeltaTree performing significantly better than state-of-the-art B-link tree for Movidius Myriad 2 platform, and provides a strong basis to explore in future energy-efficient data structures and algorithms that exploit fine-grained data locality provided by cache-oblivious models for emerging edge hardware platforms.

ACKNOWLEDGMENT

This work was supported by the European Union Seventh Framework Programme under EXCESS project (grant n° 611183, http://excess-project.eu), and by the Research Council of Norway under PREAPP project (grant n° 231746/F20, http://preapp.eu) and eX3 project (grant n° 270053). The authors would like to thank Sigma2 for giving us access to the national e-infrastructure.

REFERENCES

[1] M. H. Ionica and D. Gregg, "The Movidius Myriad Architecture's Potential for Scientific Computing," *IEEE Micro*, vol. 35, no. 1, pp. 6–14, Jan. 2015.

[2] G. Ananthanarayanan, P. Bahl, P. Bodik, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-Time Video Analytics: The Killer App for Edge Computing," *Computer*, vol. 50, no. 10, pp. 58–67, Oct. 2017.

[3] A. M. Khan and F. Freitag, "On Edge Cloud Service Provision with Distributed Home Servers," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. Hong Kong: IEEE, Dec. 2017, pp. 223–226.

[4] A. M. Khan, F. Freitag, V. Vlassov, and P. H. Ha, "Towards IoT Service Deployments on Edge Community Network Microclouds," in *IEEE International Conference on Computer Communications (INFOCOM 2018)*. Honolulu, HI, USA: IEEE, Apr. 2018.

[5] I. Umar, O. Anshus, and P. Ha, "DeltaTree: A Practical Locality-aware Concurrent Search Tree," IFI-UIT TR 2013-74, Tech. Rep., Dec. 2013, arXiv:1312.2628 [cs.DC].

[6] I. Umar, O. J. Anshus, and P. H. Ha, "DeltaTree: A Locality-aware Concurrent Search Tree," in *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '15)*, vol. 43. Portland, Oregon, USA: ACM Press, Jun. 2015, pp. 457–458.

[7] I. Umar, O. J. Anshus, and P. H. Ha, "Effect of portable fine-grained locality on energy efficiency and performance in concurrent search trees," in *21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '16)*. Barcelona, Spain: ACM Press, Mar. 2016.

[8] P. L. Lehman and s. B. Yao, "Efficient Locking for Concurrent Operations on B-Trees," *ACM Transactions on Database Systems*, vol. 6, no. 4, pp. 650–670, Dec. 1981.

[9] C. Kim, J. Chhugani, N. Satish, E. Sedlar, A. D. Nguyen, T. Kaldewey, V. W. Lee, S. A. Brandt, and P. Dubey, "FAST: Fast Architecture Sensitive Tree Search on Modern CPUs and GPUs," in *ACM SIGMOD International Conference on Management of Data (SIGMOD '10)*, Indianapolis, IN, USA, Jun. 2010, pp. 339–350.

[10] J. Sewall, J. Chhugani, C. Kim, N. R. Satish, and P. Dubey, "PALM: Parallel Architecture-Friendly Latch-Free Modifications to B+ Trees on Many-Core Processors," *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 795–806, 2011.

[11] D. Moloney, B. Barry, R. Richmond, F. Connor, C. Brick, and D. Donohoe, "Myriad 2: Eye of the Computational Vision Storm," in *26th IEEE Hot Chips Symposium (HCS 2014)*. Cupertino, CA, USA: IEEE, Aug. 2014.

[12] J. Russell, "Intel Debuts Myriad X Vision Processing Unit for Neural Net Inferencing," *HPCwire*, Aug. 2017. [Online]. Available: https://www.hpcwire.com/2017/08/28/intel-debuts-myriad-x-vision-processing-unit-neural-net-inferencing/

[13] B. Rutledge, "Introducing AIY Vision Kit: Make devices that see," Nov. 2017. [Online]. Available: https://www.blog.google/topics/machine-learning/introducing-aiy-vision-kit-make-devices-see/

[14] D. Franklin, "NVIDIA Jetson TX2 Delivers Twice the Intelligence to the Edge," 2017. [Online]. Available: https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/

[15] J. Davies, "Arm is changing machine learning experiences: Project Trillium," Feb. 2018. [Online]. Available: https://community.arm.com/processors/b/blog/posts/ai-project-trillium

[16] N. Jouppi, "Google supercharges machine learning tasks with TPU custom chip," May 2016. [Online]. Available: https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html

[17] T. Simonite, "Apple's 'Neural Engine' Infuses the iPhone With AI Smarts," *Wired*, Mar. 2017. [Online]. Available: https://www.wired.com/story/apples-neural-engine-infuses-the-iphone-with-ai-smarts/

[18] O. Shacham and M. Reynders, "Pixel Visual Core: Image Processing and Machine Learning on Pixel 2," Oct. 2017. [Online]. Available: https://www.blog.google/products/pixel/pixel-visual-core-image-processing-and-machine-learning-pixel-2/

[19] "Samsung Optimizes Premium Exynos 9 Series 9810 for AI Applications and Richer Multimedia Content," Jan. 2018. [Online]. Available: https://news.samsung.com/global/samsung-optimizes-premium-exynos-9-series-9810-for-ai-applications-and-richer-multimedia-content/

[20] M. Richardson and S. Wallace, *Getting Started with Raspberry Pi*. O'Reilly Media, Inc., 2012.

[21] C. Gupta, A. S. Suggala, A. Goyal, H. V. Simhadri, B. Paranjape, A. Kumar, S. Goyal, R. Udupa, M. Varma, and P. Jain, "ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices," in *34th International Conference on Machine Learning (ICML '17)*, vol. 70, Sydney, Australia, Aug. 2017, pp. 1331–1340.

[22] A. Kumar, S. Goyal, and M. Varma, "Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things," in *34th International Conference on Machine Learning (ICML '17)*, vol. 70, Sydney, Australia, Aug. 2017, pp. 1935–1944.

[23] P. Siegl, R. Buchty, and M. Berekovic, "Data-Centric Computing Frontiers: A Survey On Processing-In-Memory," in *2nd International Symposium on Memory Systems (MEMSYS '16)*. Alexandria, VA, USA: ACM Press, Oct. 2016, pp. 295–308.

[24] A. Dubey, P. H. J. Kelly, B. Mohr, and J. S. Vetter, "Performance Portability in Extreme Scale Computing: Metrics, Challenges, Solutions," *Dagstuhl Reports*, vol. 7, no. 10, pp. 84–110, 2018.

[25] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran, "Cache-Oblivious Algorithms," in *40th Annual Symposium on Foundations of Computer Science (FOCS '99)*, New York City, NY, USA, Oct. 1999, pp. 285–297.

[26] G. Brodal, "Cache-Oblivious Algorithms and Data Structures," in *9th Scandinavian Workshop on Algorithm Theory (SWAT '04)*, Humlebaek, Denmark, Jul. 2004, pp. 3–13.

[27] P. van Emde Boas, "Preserving order in a forest in less than logarithmic time," in *16th Annual Symposium on Foundations of Computer Science (SFCS '75)*, Oct. 1975, pp. 75–84.

[28] A. Braginsky and E. Petrank, "A Lock-free B+Tree," in *24th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '12))*. Pittsburgh, PA, USA: ACM, 2012, pp. 58–67.

[29] M. A. Bender, M. Farach-Colton, J. T. Fineman, Y. R. Fogel, B. C. Kuszmaul, and J. Nelson, "Cache-Oblivious Streaming B-Trees," in *19th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '07)*. San Diego, CA, USA: ACM, 2007, pp. 81–92.

[30] T. David, R. Guerraoui, and V. Trigonakis, "Asynchronized Concurrency: The Secret to Scaling Concurrent Search Data Structures," in *20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '15)*, vol. 50, no. 4, Istanbul, Turkey, Mar. 2015, pp. 631–644.

[31] I. Umar, O. Anshus, and P. Ha, "GreenBST: Energy-Efficient Concurrent Search Tree," in *22nd International European Conference on Parallel and Distributed Computing (Euro-Par '16)*, Grenoble, France, Aug. 2016, pp. 502–517.

[32] V. N.-N. Tran, B. Barry, and P. H. Ha, "Power models supporting energy-efficient co-design on ultra-low power embedded systems," in *16th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS '16)*. Samos, Greece: IEEE, Jul. 2016, pp. 39–46.

[33] P. H. Ha, V. N.-N. Tran, I. Umar, P. Tsigas, A. Gidenstam, P. Renaud-Goud, I. Walulya, and A. Atalar, "EXCESS Project: D2.1 Models for Energy Consumption of Data Structures and Algorithms," *arXiv preprint arXiv:1801.09992 [cs.DC]*, 2014.

[34] Y. Afek, H. Kaplan, B. Korenfeld, A. Morrison, and R. E. Tarjan, "CBTree: A Practical Concurrent Self-Adjusting Search Tree," in *26th International Symposium on Distributed Computing (DISC '12)*, Salvador, Brazil, Oct. 2012, pp. 393–417.

[35] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload Analysis of a Large-scale Key-value Store," in *ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '12)*. London, England, UK: ACM, Jun. 2012, pp. 53–64.