Nested-Parallelism PageRank on RISC-V Vector Multi-Processors

Alon Amid, Albert Ou, Krste Asanović, Borivoje Nikolić
University of California, Berkeley
{alonamid,aou,krste,bora}@berkeley.edu

ABSTRACT

Graph processing kernels and sparse-representation linear algebra workloads such as PageRank are increasingly used in machine learning and graph analytics contexts. While data-parallel processing and chip-multiprocessors have both been used in recent years as complementary mitigations to the slowing rate of single-thread performance improvements, they have been used together most effectively on dense data-structure representations as opposed to sparse representations. We present nested-parallelism implementations of PageRank for RISC-V multi-processor Rocket chip SoCs with Hwacha vector architecture accelerators. These software implementations are used for hardware and software design-space exploration using FPGA-accelerated simulation with multiple silicon-proven multi-processor SoC configurations. The design space includes a variety of scalar cores, vector accelerator cores, and cache parameters, as well as multiple software implementations with tunable parallelism parameters. This work shows the benefits of the loop-raking vectorizing technique compared to an alternative vectorizing technique, and presents up to a 14x run-time speedup relative to a parallel-scalar implementation running on the same SoC configuration. A 25x speedup is demonstrated in a dual-tile SoC with dual-lanes-per-tile vector accelerators, compared to a minimal scalar implementation, demonstrating the scalability of the proposed nested-parallelism techniques.

1 INTRODUCTION

Graph processing has been a recent topic of interest in high performance computing, systems, and architecture research. While graph abstractions have long been of interest in mathematical and numerical computing communities, the rise of data analytics and the big-data revolution have exposed the various use-cases of graph processing to many additional domains. Computing statistical properties of graphs is required for many scientific, data-analysis, and machine learning applications, including recommendation systems [5], fraud detection [2] and biochemical processes [1, 28].

The Hwacha micro-architecture [20, 29] is a silicon-proven[16, 22, 23, 32] open-source decoupled vector-machine implementation associated with the RISC-V ISA and Rocket-Chip SoC generator [4] infrastructure. It is an evolution of previous decoupled vector-fetch projects such as Maven [19], and uses the Hwacha RISC-V non-standard ISA extension. Hwacha's main micro-architectural features include a decoupled multi-lane design orchestrated by a master sequencer. Each lane consists of an SRAM-based register file with a capacity of 16 KiB, allowing for a maximum vector length of 2048 double-width elements. The SRAM-based register file consists of 4 1R1W banks accessed in a systolic pattern, providing for an aggregate read bandwidth of 4x128 bits and an equal write bandwidth every cycle. Each lane also includes several floating-point and integer functional units allowing for a throughput of approximately 4 ops/cycle, depending on the operation. Hwacha is integrated with the Rocket Chip SoC generator and uses a TileLink-based [8] cache-coherent memory system. The vector memory unit (VMU) in each lane has a bandwidth of 128 bits/cycle to the backing L2 cache.

The Hwacha micro-architecture has been optimized for, and mostly been evaluated on, dense linear algebra kernels such as general matrix multiplication (DGEMM) [21]. Specifically, it has not been designed or optimized for sparse and irregular workloads, and the properties of the Hwacha vector architecture have not yet been explored using sparse linear algebra kernels. An evaluation of the bottlenecks of sparse linear algebra workloads on this micro-architecture can provide additional insight into future design choices.

One particular instance of a common graph processing kernel is PageRank [27]. PageRank is an algorithm originally used by Google to measure the importance of websites, with the purpose of ranking them. Each website is modeled as a node (or vertex) in a graph, and hyperlinks between websites are modeled as edges in the graph. After running the PageRank algorithm, each vertex (representing a website) is assigned a PageRank score, which allows it to be compared and ordered against other websites, hence - creating a ranking. The PageRank score is effectively a probability distribution that represents the likelihood of a random walker (or a random "hyperlink clicker") to arrive at a particular vertex (or web-page). By viewing the PageRank problem as an irreducible Markov chain, this probability distribution can be computed as an eigenvector problem or a homogeneous linear system [10, 18]. Using the power method, this results in an iterative process of sparse matrix-vector multiplication (SpMV) operations. Therefore PageRank SpMV implementations can be representative of a large class of sparse linear algebra workloads.

When exploring parallel implementations, researchers are commonly constrained by rigid assumptions: either a fixed hardware implementation (such as the case in high-performance computing research), or fixed software benchmark suits (such as the case for hardware architecture evaluation). This work attempts to broaden the design space to include both hardware and software parameters for mappings of PageRank kernels to a chip multi-processor with Hwacha vector accelerators.

2 NESTED PARALLELISM PAGERANK

Throughout this study, "nested parallelism" is considered to be the use of multiple parallel execution methods in a hierarchical manner. Nested parallelism is used extensively in various software libraries to maximize the amount of exploited parallelism given a set of execution resources. Furthermore, it allows for tuning and finer-grained load-balancing between parallelizable elements [11].

The ideas of exploiting nested parallelism in graph processing have shown encouraging results in several previous attempts.

Nested parallelism within a single GPU has been studied to efficiently utilize GPU architectures for general data-parallel workloads [12, 26]. Nested parallelism using multiple GPUs has been demonstrated on graph processing algorithms, but requires careful dynamic load balancing due to the high cost of transferring data between host processors and CPUs [13]. Nested parallelism in graph processing has also been explored using packed-SIMD approaches with Intel AVX extensions and multi-core processors [14, 25]. Recent work by research groups in Cornell [17] provides significant contributions in the study and taxonomy of loop-level parallelism through nested parallel hardware elements. Loop-task parallel programs are a major use-case for nested-parallelism implementations. This study explores the nested parallelism of chip multi-processors (CMPs) with decoupled vector-fetch machines integrated into SoCs, based on the silicon-proven Hwacha [20] micro-architecture. This involves both hand-tuned optimization of the internal vector-architecture code for the consideration of the particular sparse data-structures representations, as well as an additional layer of OpenMP for CMP multi-threading and load-balancing management. To our knowledge, this nested-parallelism approach for PageRank has not been previously attempted using vector-fetch architecture instructions sets and vector-machine micro-architectures.

2.1 DCSR/DCSC Sparse Matrix Representation

Static graphs can be represented as sparse adjacency matrices. Sparse matrices are commonly represented using multiple levels of indirection, making the exploitation of parallelism within a sparse matrix for linear algebra operations highly dependent upon the data-structure representation. While some representations such as Coordinate format (COO) may allow embarrassingly-parallel execution at the cost of data-locality, other representations such as Compressed Sparse Row/Column (CSR/CSC) improve data-locality and constant-time accesses at the cost of creating dependencies between different parts of the data-structure (hence, reducing parallelism).

The Double Compressed Sparse Row/Column (DCSR/DCSC) [7] representation adds an additional level of indirection on-top of CSR/CSC representations. This format is useful for the cases of hyper-sparse matrices [7], since it provides an additional level of compression. Nevertheless, this additional level of compression comes at the cost of an explicit indices arrays and an auxiliary pointers array to reduce access time complexity.

For the purposes of nested-parallelism experimentation, DCSR/DCSC representations are a useful data structure, since they expose two levels of indirection, which provide a natural boundary between two levels of parallelism. A simple scalar implementation of a PageRank SpMV kernel using a DCSR format includes 3 nested loops: The outer loop iterates over the auxiliary row partition pointers (each of which points to the beginning of a CSR matrix), the second loop iterates over the row indices and pointers, and finally the inner loop iterates over the non-zero values and column indices.

Hence, the top level of the DCSR data structure (the outermost loop) can be thought of as a coarse-grained parallel layer. This layer can be parallelized in multi-threaded hardware implementations by using parallel hardware threads. Specifically, the implementations

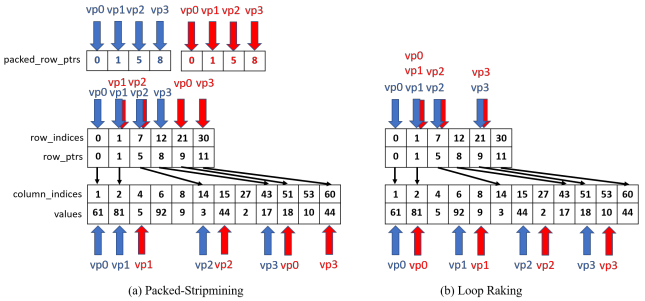

Figure 1: Packed-Stripmining and Loop Raking vectorization methods for a CSR sub-component of a DCSR matrix. The blue arrows represent the actions of virtual processors in the first iteration, while the red arrows represent the actions of virtual processors in the second iteration. In packed-stripmining, the pointer array is accessed contiguously, while in loop raking, the values array is accessed with a constant stride.

in this work use OpenMP threads to parallelize across CMP cores. Note each thread processes a sub-section of the matrix which is represented in CSR format (with the additional row indices array).

The second level of our nested-parallelism scheme therefore processes the internal CSR sub-matrices. We use the data-parallel vector-architecture accelerator in order to parallelize the internal loops of the CSR sub-matrices. Hence, we refer to the coarse-grained single level parallel implementation as the reference scalar implementation.

2.2 Virtual Processors View

A popular way of thinking of the parallel nature of vector machines is as multiple concurrent "virtual processors" [3, 31]. Since a CSR matrix data-structure is composed of two arrays, the virtual processors can operate in-parallel either on the pointers array, or on the values array. In graph-processing terms, these two approaches have been described in [12] as the node-parallel approach and the edge-parallel approach. This work attempts to apply these approaches by comparing two vectorizing techniques: the first technique, nicknamed "packed-stripmining", attempts parallel processing of the pointers array elements (node-centric). The second technique, known as "loop-raking", focuses on parallel processing of the values array (edge-centric). Note that "packed stripmining" and "loop raking" are not complementary approaches used together, but rather alternative approaches to parallelizing the same problem across different parts of the data-structure.

2.3 Packed-Stripmining

Stripmining is a common technique for vectorization of dense loops using vector-length-agnostic code. This means that the code is not aware of the size of the hardware vector registers during compile-time. Therefore, a stripmining loop attempts to configure the maximum possible requested vector length, and treats the accommodated vector length as a variable. The stripmining loop then “strips”

a layer of the actual vector register length, and repeats the process for reminder.

However, stripmining works best on a continuous array of elements in order to exploit parallelism efficiently. In the case of a CSR sparse matrix representation, the imbalance in the number of non-zero elements in each sparse row of a CSR matrix requires additional manipulation for efficient stripmining. The progress of the stripmining loop over the pointers array depends on the number of non-zero elements each pointer in the array is pointing too. This imbalance results in processing elements being idle, waiting for the "worst case" virtual processor to finish. One possible solution to this scenario is to pack only "active" (non-idle) pointers from the pointers array. The "Packed-Stripmining" approach for CSR matrices "packs" an array of imbalanced row pointers into a dense array, at the cost of using control-flow statements within each iteration of the stripmining loop. By re-packing the row-pointers every iteration of the vector-processing loop, this "packed-stripmining" loop can operate on the packed array as it would commonly operate in balanced dense scenarios.

Note, that the packing stage itself cannot be vectorized due to an internal conditional `while` loop which breaks the vectorization (or result in an inefficient implementation), and therefore the packing process is separated from the vectorized segment. However, since the Hwacha vector accelerator is a vector-fetch architecture, there is a possibility of overlap between control flow code, running on the scalar core, and vectorized code, running on the vector accelerator. This overlap should help minimize the additional control-flow overhead, incurred by the packing stage.

Furthermore, while the packing approach is designed to fix the problem of imbalanced sparse matrices (and transitively, imbalanced and power-law graphs), this approach still encounters a difficulty if the imbalance is extreme (for example: one row has more elements than all other rows combined), or if there is significant imbalance towards the last rows/columns of the matrix. We must remember that each virtual-processor handles only one row (or "vertex" in the case of a graph), and therefore, if all the virtual-processors are done working but there is one row with many elements that still need to be processed, this row will only be processed by a single virtual-processor while leaving the remaining virtual processors idle.

2.4 Loop-Raking

The Loop Raking vectorizing approach was originally proposed for sorting algorithms [31]. The raking access pattern is a common vector pattern used for two-dimensional data-structures [3]. It has been commonly used in dense data-structure scenarios such as dense matrix multiplication and data compression. It allows contiguous elements to be processed by the same virtual processor, which can have positive or negative implications (depending on the scenario) regarding spatial data-locality, memory consistency and atomic operations. In the raking access pattern, virtual-processors process array-elements in intervals of $array\_size/vector\_length$.

This approach offers a partial solution to the imbalance problems that appear in the packed-stripmining approach. In loop-raking, all virtual processors can be utilized in every iteration of the loop (with the possible exception of the last iteration). Since the vectorization is performed across the values array rather than the pointers array, performing an PageRank SpMV with loop-raking results in an inherently more load-balanced scheme. Nevertheless, checking row sizes and boundaries is still required in order to have full information about each matrix element for the purposes of linear algebra operations. Therefore, unlike the dense use cases for which loop-raking was originally proposed, a "tracker" vector register is still required in the sparse matrix case in order to maintain information about progress through each row in a CSR structure. This tracker vector somewhat limits the possible load-balancing, since the current implementation under evaluation in this work chooses to define the rake interval as the size of the largest row. Therefore, while loop-raking resolves the utilization problem of processing a large row at the tail-end of the matrix, it does not solve the problem of an extremely large row which composes the majority of elements in the matrix (as may be the case in a power-law graph).

Another way to view the difference between packed-stripmining and loop-raking is by the types of tracking checks that are performed. In the packed-stripmining approach, the "virtual processors" process the packed array, and perform checks to track the progress of elements in the non-zero elements array. This is as opposed to the loop-raking approach, in which the "virtual processors" process the non-zero elements array (both the indices and the values), while performing checks to track the status of the pointers array.

3 EVALUATION METHOD

In order to fully implement and evaluate the proposed parallel nested-parallelism models, a full Linux-based software stack was required. Standard parallel programming libraries such as OpenMP are used for the external coarse-grain parallelism level and to perform pre-processing and data-structure construction on public graph datasets. GraphMat [30] was chosen as the base graph-processing framework infrastructure in this work for several reasons. GraphMat uses DCSC and DCSR data-structures to represent the graph adjacency matrices, which have been previously described to provide a natural boundary between the external parallelism abstraction and the internal parallelism abstraction in nested-parallelism. Since common graph processing benchmark data-sets are usually provided in edge-list format, the use of the GraphMat infrastructure abstracts away the complexities of constructing the efficient DCSC and DCSR graph representations out of these edge-list formats. Additionally, GraphMat uses bit-vectors in-order to help represent sparse vectors. The use of bit-vector is very similar to the use of vector predicate registers in the Hwacha vector accelerator. While the Hwacha architecture is not able to load bit-vectors directly into predicate registers, this implementation is still helpful for it's equivalent representation. GraphMat has been used for a variety of experiments and workloads, including in the architecture research community [9], and has proven to be consistently high-performing while maintaining it's unique abstractions.

Hardware design space exploration is based on the Rocket Chip SoC generator with the Hwacha vector accelerator generator. Rocket Chip is a silicon-proven SoC generator associated with the RISC-V ecosystem. The SoC setup includes configurations with single-core and dual-core Rocket Chip in-order cores, each accompanied by

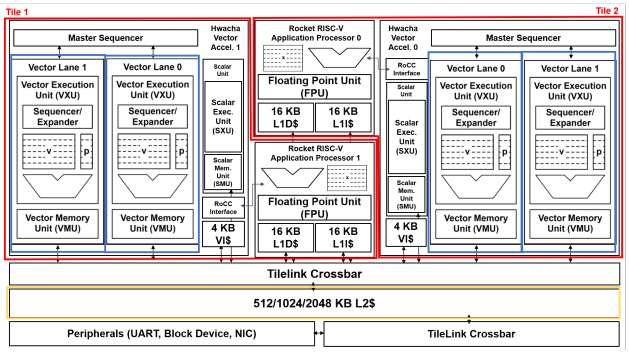

Figure 2: Hardware SoC configurations under evaluation generated by the rocket chip generator with the Hwacha vector accelerator. Different colors mark different hardware configuration parameters under exploration

various configurations of single-lane or dual-lane Hwacha vector accelerators. We group together a single scalar in-order core and a single vector accelerator into a *tile*. The SoC configurations include a memory hierarchy with two levels of cache, of which the vector-accelerator is connected directly to the L2 cache. The size of the L2 cache is also configurable parameter across the test configurations.

Performance evaluation was executed using FPGA-accelerated cycle-exact simulation on the FireSim platform [15]. The FireSim platform allows for FPGA-accelerated cycle-exact simulation on the public cloud using Amazon Web Services (AWS) EC2 FPGA instances. This FPGA-accelerated simulation enables running application benchmarks on top of a fully functional Linux system in a cycle-accurate simulation with only a 500x slow-down compared to real time execution on a real silicon implementation. Similar experiments would require multiple weeks using standard software RTL simulators. Furthermore, the FireSim framework also includes elaborate memory models which can simulate a full DDR3 backing memory system and last level caches (LLC) with high precision timing models, while maintaining the performance level of FPGA-accelerated simulation [6]. It is worth noting that at least two of the SoC configurations under evaluation have been previously taped-out as test-chips. However, since test-chips lack a realistic backing-memory system (which is significant for the evaluation of sparse workloads), we chose to perform the full evaluation using the FireSim platform.

Twelve different SoC hardware configurations were simulated with a simulated SoC frequency of 1033 MHz. The backing-memory model used for the simulations was a DDR3 memory model with speed-grade of 14-14-14. Figure 2 shows block diagrams of the evaluated SoC configurations.

Performance was measured on three sample graphs selected from the Stanford Network Analysis Project [24]. The graphs were selected to present different use-cases and characteristics, while still maintaining a size which allows for testing at reasonable times across the design-space.

Performance was also evaluated using an additional software parameter which controls the number of DCSR partitions in relation to the number of hardware threads. This DCSR partition factor is multiplied by the number of hardware threads to determine the number of overall top-level DCSR partitions. For example, if the DCSR partition factor is 4, and the number of hardware threads is 2 (in a dual-tile configuration), then the graph DCSR representation will have 8 DCSR partitions. Since the coarse-grain OpenMP external parallelization scheme parallelizes across cores using units of DCSR partitions, increasing this factor increases the granularity of the dynamic allocation of partitions between cores. However, while this factor increases the dynamic allocation of kernels to hardware threads, it may also decrease vector lengths used in the vectorized code if a large number of partitions results in smaller graph sections per-partition.

4 DESIGN SPACE ANALYSIS

The measured results present several interesting patterns. We address two different speedups when analyzing the results: the overall-speedup compared to a reference minimal scalar design (single-tile, L2 size of 512 KB), and relative-speedup compared to an equivalent design without a vector accelerator (*i.e.* a dual-tile-single-lane design would be compared against a scalar implementation of a dual-tile design with the same cache size).

4.1 L2 Cache Size

Unsurprisingly, different cache sizes have little to no effect on the performance of PageRank on all graph types and all parallel PageRank implementations in the various hardware configurations. This behavior is consistent both when varying the number of tiles and when varying the number of vector lanes. The wikiVote graph is small enough to fit in all of the evaluated L2 cache configurations, hence, it is not surprising that we do not observe changes in behavior across the evaluated L2 cache sizes for the wikiVote graph. However the roadNet-CA and amazon0302 graphs are larger graphs which cannot fit in any of the L2 cache size configurations, but we also do not observe an improvement across different cache sizes for these larger graphs under any implementation. This behavior is somewhat expected of graph workloads, which have been known to have poor spatial and temporal locality.

4.2 Number of Tiles vs. Vector Lanes

As expected, increasing the number of tiles improves the absolute performance of all graphs and software configurations compared to a minimal single-tile configuration. The coarse-grain OpenMP-based scalar reference implementation obtains near-linear scaling between a single tile to two tiles. However, when comparing the relative-speedup of the vectorized kernels vs. the reference scalar kernel (figure 4), it is noticeable that the loop raking technique obtains a higher relative-speedup in the dual-tile case compared to the single-tile case. On the other hand, the packed-stripmining approach obtains the same, and sometimes even worst, relative-speedup in the dual-tile case compared to the single-tile case. When considering the absolute-speedup between the single-tile and dual-tile case 3), the loop-raking technique obtains near-linear absolute scaling between one-tile to two-tiles, similar to the scaling of the scalar implementation. The packed stripmining approach presents less consistent scaling behavior, especially on the amazon0302 graph. We can conclude from these observations that the

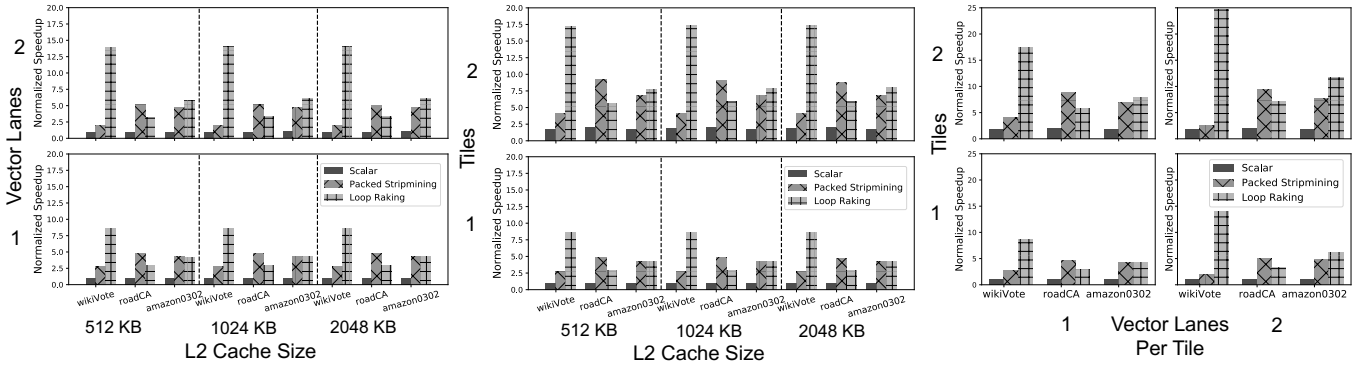

Figure 3: Design space exploration across hardware parameters. Comparison on absolute Speedup compared to a minimal reference design consisting of a single scalar core and 512KB L2 cache. The dual-tile/single-lane configuration provides greater speedup than the single-tile/dual-lane configuration (with similar area overheads). L2 cache size is not a factor for speedups on graphs larger than the cache size.

loop-raking method is more scalable, in relation to the number of tiles, than the packed-stripmining method.

Also as expected, increasing the number vector lanes per tile generally improves the performance of most graph and software configurations, compared to a single-lane or single-scalar-tile configurations. Similarly to the multi-tile case, the loop-raking technique obtains a higher relative-speedup in the dual-lane case compared to a single-lane case. However, this observation is less-informative than in the dual-tile case, since the multi-lane scenario relative-speedup is equivalent to the multi-lane absolute-speedup since there is only a single scalar core in both the single-lane and dual-lane cases. The absolute-speedup scaling between the single-lane configuration to the dual-lane configuration does not scale as well as it did in the multi-tile comparison. For the packed-stripmining method, an additional lane provides a minimal speedup gain. Furthermore, the packed-stripmining implementation actually exhibits a smaller speedup in the dual-lane configuration on the wikiVote graph compared to the single-lane configuration. Nevertheless, it is clear than an additional lane indeed provides additional significant speedup for the loop-raking method.

Given the area and power cost of additional tiles and vector lanes, it is interesting to investigate the trade-off between the two for each parallel implementation. We would like to compare a single-tile-dual-lane design to a dual-tile-single-lane design. Both designs have a total of two vector lanes (albeit, split between two tiles vs. concentrated in one tile), but the latter has an additional scalar control processor controlling the second vector lane. While the area comparison is not exact, we know from previous test-chips which include Rockets scalar processors and Hwacha vector accelerators [16] that the vector lanes are the dominant area component compared to scalar cores. When observing the normalized speedups in figure 3, it is clear that a dual-tile-single-lane design demonstrates a more significant speedup compared to the minimal scalar single-tile scalar design on all of the evaluated graphs. Figure 4 shows that these observations remain consistent across the different software configurations as well. We can therefore conclude that multi-tile-single-lane configurations are likely a better choice for a PageRank workload with nested-parallelism implementations (and perhaps sparse workloads in general) compared to single-tile-multi-lane configurations. Nevertheless, these observations need to be supported by supplemental energy and area measurements from a fabricated SoC.

4.3 Packed-Stripmining vs. Loop-Raking

An initial observation of the measured results in figure 4 indicates that the best performing vectorized kernel depends on the choice of graph and the DCSR partitioning parameters. When observing the results with a DCSR partition factor of 1, we see that loop-raking outperforms packed-stripmining for the wikiVote and amazon0302 graphs, but gets beaten in the roadNet-CA graph. However, further observation demonstrates that the loop-raking speedup (both relative-speedup and absolute-speedup) improves as the DCSR partition factor increases for the roadNet-CA and amazon0302 graphs. Hence, for DCSR partition factors of 4, 8 and 16, loop-raking is able to out-perform packed-stripmining for the roadNet-CA graph as well. Furthermore, the maximum observed speedups obtained by loop-raking (both relative-speedups and absolute-speedups) are significantly higher than the maximum observed speedups obtained by packed-stripmining (5.1x, 4.6x, 2.7x maximum relative speedups for the 3 graphs using packed-stripmining, vs. 7.3x, 9.2x, 13.9x maximum relative-speedups for the 3 graphs respectively using loop-raking). We can therefore conclude that when tuned correctly, loop-raking is generally a better choice of vectorizing kernel for nested-parallelism PageRank implementations. Additional analysis regarding the impact of graph properties on the behavior of each of the implementation is omitted from this report due to space constraints.

4.4 Vector vs. Multi-Core Scalar Processors

Another question of interest when addressing graph-processing and sparse workloads regards the benefit of data-level parallelism vs. task-level parallelism. This question can be projected to the design space under evaluation by comparing the run-time of a scalar-parallel implementation to an equivalent vectorized implementation. It is important to note that while the scalar implementation in this evaluation has a parallel dimension across DCSR

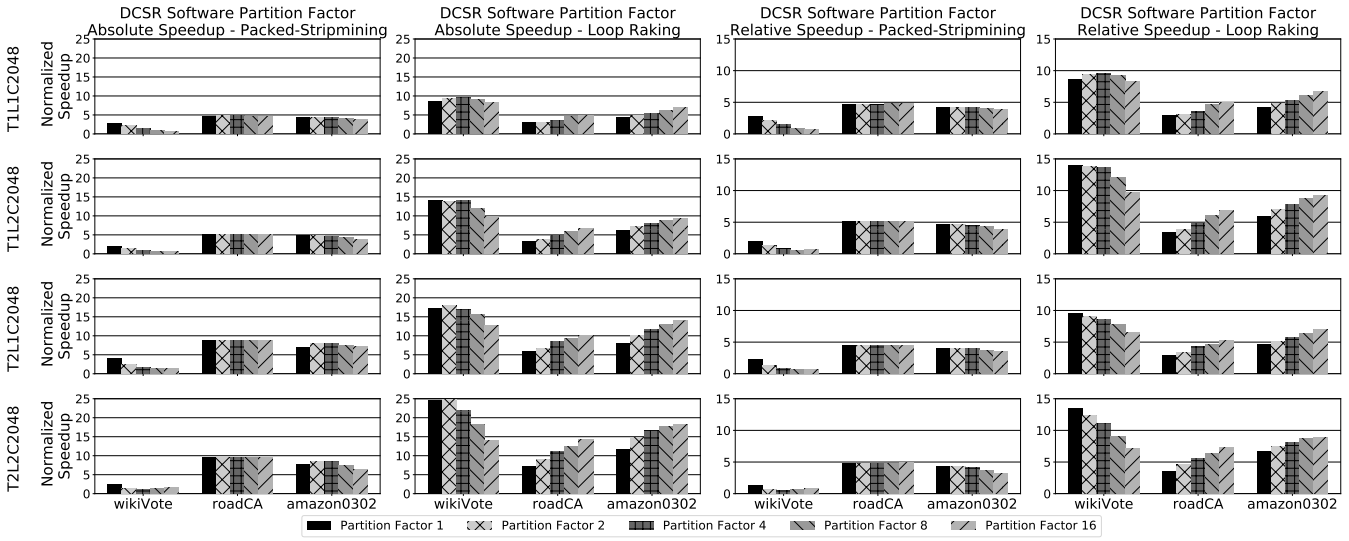

Figure 4: Design space exploration across software parameters. Absolute and relative Speedups for 4 SoC configurations. SoC configurations are labeled with the format T[Number of Tiles]L[Number of Vector Lanes per Tile]C[L2 Cache Size]. The small graph is negatively impacted by the software partition factor due to the reduce vector length. Loop-raking with higher software partition factors allow for better load-balancing, and therefore better speed-ups. Large partition factor allow loop-raking to outperform packed-stripmining on all graphs.

partitions, this is only coarse-grained task-level parallelism. The internal loops of the scalar implementation were not optimized for task-level parallelism. Nevertheless, we can attempt to perform a coarse-estimate by observing the results of figure 3. We observe that the dual-tile configurations obtain a 2x speedup when using the scalar implementations compared to the single-tile scalar implementations. At the same time, we observe that a single-tile-single-lane vector accelerator obtains between 2.75-8.6x speedup compared to the scalar implementation.

When analyzing the benefits of adding a vector accelerator for a sparse workload as opposed to adding additional scalar cores, we must consider the number of addition functional units that are contributed by a vector accelerator vs. a scalar core. The Hwacha vector accelerator has 4 floating-point functional units, while a Rocket scalar core has only 1. In almost all cases, we observe that an implementation consisting of the Hwacha vector accelerator obtains more than a 4x absolute speedup. The only cases where the vector accelerator obtains an absolute speedup lower than 4x are for the wikiVote graph in the packed-stripmining case, and for certain DCSR partition configurations of the CA-roadNet graph in the loop-raking case. Hence, it is reasonable to concluded that with the correct choice of software implementation and optimization, the vector accelerator can potentially achieve the desired speedup (greater than 4x) in all of the evaluated scenarios, and therefore data-parallel vector accelerators remain a valid choice for sparse and graph-processing workloads.

5 CONCLUSION

This work presents SW/HW co-design space exploration and evaluation of nested-parallelism PageRank graph processing kernels on multi-core vector architectures. The design space was evaluated by using a variety of SoC configurations of Rocket multi-processors with Hwacha vector accelerators and multiple software configurations. This work demonstrated the benefits of the loop-raking vectorizing technique compared to the packed-stripmining vectorizing technique for a sparse data-structure representations, attributed to the overhead of additional re-packing in the scalar-processor and longer vector lengths. Furthermore, this work demonstrated that using correct data-structure partitioning, the loop-raking vectorizing technique can achieve up to 14x relative-speedup compared to equivalent coarse-grained parallel scalar implementations, and a 25x absolute speedup compared to a minimal single-tile scalar implementation by using dual-tile SoC with dual-lanes-per-tile vector accelerators. Finally, this work demonstrated that a dual-tile-single-vector-lane approach achieves the best performance for these types of vectorized nested-parallel sparse workloads under evaluation within fixed area constraints, by providing a balanced mix of multi-tile task-level parallelism and vector architecture data-level parallelism.

6 ACKNOWLEDGMENTS

The authors would like to thank Colin Schmidt for his assistance. The information, data, or work presented herein was funded in part by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number DE-AR0000849. Research was partially funded by ADEPT Lab industrial sponsor Intel, and ADEPT Lab affiliates Google, Siemens, and SK Hynix. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

REFERENCES

[1] Tero Aittokallio and Benno Schwikowski. 2006. Graph-based methods for analysing networks in cell biology. *Briefings in Bioinformatics* 7, 3 (2006), 243. https://doi.org/10.1093/bib/bbl022

[2] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph Based Anomaly Detection and Description: A Survey. *Data Min. Knowl. Discov.* 29, 3 (May 2015), 626–688. https://doi.org/10.1007/s10618-014-0365-y

[3] Krste Asanovic. 1998. *Vector microprocessors.* Ph.D. Dissertation. University of California, Berkeley.

[4] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Palmer Dabbelt, John Hauser, Adam M. Izraelevitz, Sagar Karandikar, Benjamin Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moretó, Albert Ou, David Patterson, Brian H Richards, Colin Schmidt, Stephen M. Twigg, Huy Vo, and Andrew Waterman. 2016. The Rocket Chip Generator. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17* (2016).

[5] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. 2010. Fast Incremental and Personalized PageRank. *Proc. VLDB Endow.* 4, 3 (Dec. 2010), 173–184. https://doi.org/10.14778/1929861.1929864

[6] David Biancolin, Sagar Karandikar, Donggyu Kim, Jack Koenig, Andrew Waterman, Jonathan Bachrach, and Krste Asanović. 2019. FASED: FPGA-Accelerated Simulation and Evaluation of DRAM. In *The 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'19) (FPGA '19).* ACM, New York, NY, USA, 10.

[7] Aydin Buluc and John R Gilbert. 2008. On the representation and multiplication of hypersparse matrices. In *2008 IEEE International Symposium on Parallel and Distributed Processing.* 1–11. https://doi.org/10.1109/IPDPS.2008.4536313

[8] Henry M Cook, Andrew S Waterman, and Yunsup Lee. 2015. *TileLink cache coherence protocol implementation.* Technical Report.

[9] Tae Jun Ham, Lisa Wu, Narayanan Sundaram, Nadathur Satish, and Margaret Martonosi. 2016. Graphicionado: A High-performance and Energy-efficient Accelerator for Graph Analytics. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-49).* IEEE Press, Piscataway, NJ, USA, Article 56, 13 pages. http://dl.acm.org/citation.cfm?id=3195638.3195707

[10] Taher Haveliwala. 1999. *Efficient computation of PageRank.* Technical Report. Stanford.

[11] Susan Flynn Hummel and Edith Schonberg. 1991. Low-overhead scheduling of nested parallelism. *IBM Journal of Research and Development* 35, 5.6 (Sep. 1991), 743–765. https://doi.org/10.1147/rd.355.0743

[12] Yuntao Jia, Victor Lu, Jared Hoberock, Michael Garland, and John C Hart. 2011. Edge v. node parallelism for graph centrality metrics. In *GPU Computing Gems Jade Edition.* Elsevier, 15–28.

[13] Zhihao Jia, Yongkee Kwon, Galen Shipman, Pat McCormick, Mattan Erez, and Alex Aiken. 2017. A Distributed multi-GPU System for Fast Graph Processing. *Proc. VLDB Endow.* 11, 3 (Nov. 2017), 297–310. https://doi.org/10.14778/3157794.3157799

[14] Peng Jiang and Gagan Agrawal. 2018. Conflict-free Vectorization of Associative Irregular Applications with Recent SIMD Architectural Advances. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization (CGO 2018).* ACM, New York, NY, USA, 175–187. https://doi.org/10.1145/3168827

[15] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, Q. Huang, Kyle Kovac, Borivoje Niklic, Randy Katz, Jonathan Bachrach, and Krste Asanovic. 2018. FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA).* 29–42. https://doi.org/10.1109/ISCA.2018.00014

[16] Ben Keller, Martin Cochet, Brian Zimmer, Jaehwa Kwak, Alberto Puggelli, Yunsup Lee, Milovan Blagojević, Stevo Bailey, Pi-Feng Chiu, Palmer Dabbelt, Colin Schmidt, Elad Alon, Krste Asanović, and Borivoje Nikolić. 2017. A RISC-V Processor SoC With Integrated Power Management at Submicrosecond Timescales in 28 nm FD-SOI. *IEEE Journal of Solid-State Circuits* 52, 7 (July 2017), 1863–1875. https://doi.org/10.1109/JSSC.2017.2690859

[17] Ji Kim, Shunning Jiang, Christopher Torng, Moyang Wang, Shreesha Srinath, Berkin Ilbeyi, Khalid Al-Hawaj, and Christopher Batten. 2017. Using Intra-core Loop-task Accelerators to Improve the Productivity and Performance of Task-based Parallel Programs. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50 '17).* ACM, New York, NY, USA, 759–773. https://doi.org/10.1145/3123939.3136952

[18] Amy N Langville and Carl D Meyer. 2004. Deeper inside pagerank. *Internet Mathematics* 1, 3 (2004), 335–380.

[19] Yunsup Lee, Rimas Avizienis, Alex Bishara, Richard Xia, Derek Lockhart, Christopher Batten, and Krste Asanović. 2011. Exploring the tradeoffs between programmability and efficiency in data-parallel accelerators. In *2011 38th Annual International Symposium on Computer Architecture (ISCA).* 129–140.

[20] Yunsup Lee, Albert Ou, Colin Schmidt, Sagar Karandikar, Howard Mao, and K Asanovic. 2015. The Hwacha Microarchitecture Manual, Version 3.8. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-263* (2015).

[21] Yunsup Lee, Colin Schmidt, Sagar Karandikar, Daniel Dabbelt, Albert Ou, and K Asanovic. 2015. Hwacha Preliminary Evaluation Results, Version 3.8. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-264* (2015).

[22] Yunsup Lee, Andrew Waterman, Rimas Avizienis, Henry Cook, Chen Sun, Vladimir Stojanović, and Krste Asanović. 2014. A 45nm 1.3GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators. In *ESSCIRC 2014 - 40th European Solid State Circuits Conference (ESSCIRC).* 199–202. https://doi.org/10.1109/ESSCIRC.2014.6942056

[23] Yunsup Lee, Brian Zimmer, Andrew Waterman, Alberto Puggelli, Jaehwa Kwak, Ruzica Jevtic, Ben Keller, Stevo Bailey, Milovan Blagojevic, Pi-Feng Chiu, Henry Cook, Rimas Avizienis, Brian Richards, Elad Alon, Borivoje Nikolic, and Krste Asanovic. 2015. Raven: A 28nm RISC-V vector processor with integrated switched-capacitor DC-DC converters and adaptive clocking. In *2015 IEEE Hot Chips 27 Symposium (HCS).* 1–45. https://doi.org/10.1109/HOTCHIPS.2015.7477469

[24] Jure Leskovec et al. 2010. Stanford network analysis project. *http://snap. stanford. edu* (2010).

[25] Sander Lijbrink. 2015. *Irregular Algorithms on The Xeon Phi.* Master's thesis. Universiteit Van Amsterdam.

[26] Adam McLaughlin and David A. Bader. 2018. Accelerating GPU Betweenness Centrality. *Commun. ACM* 61, 8 (July 2018), 85–92. https://doi.org/10.1145/3230485

[27] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[28] Nataša Pržulj. 2011. Protein-protein interactions: Making sense of networks via graph-theoretic modeling. *Bioessays* 33, 2 (2011), 115–123.

[29] Colin Schmidt and Albert Ou. 2018. Hwacha: A Data-Parallel RISC-V Extension and Implementation. In *Proceedings of the Inaugural RISC-V Summit.* RISC-V Foundation.

[30] Narayanan Sundaram, Nadathur Satish, Md Mostofa Ali Patwary, Subramanya R. Dulloor, Michael J. Anderson, Satya Gautam Vadlamudi, Dipankar Das, and Pradeep Dubey. 2015. GraphMat: High Performance Graph Analytics Made Productive. *Proc. VLDB Endow.* 8, 11 (July 2015), 1214–1225. https://doi.org/10.14778/2809974.2809983

[31] Marco Zagha and Guy E. Blelloch. 1991. Radix Sort for Vector Multiprocessors. In *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing (Supercomputing '91).* ACM, New York, NY, USA, 712–721. https://doi.org/10.1145/125826.126164

[32] Brian Zimmer, Yunsup Lee, Alberto Puggelli, Jaehwa Kwak, Ruzica Jevtić, Ben Keller, Steven Bailey, Milovan Blagojević, Pi-Feng Chiu, Hanh-Phuc Le, Po-Hung Chen, Nicholas Sutardja, Rimas Avizienis, Andrew Waterman, Brian Richards, Philippe Flatresse, Elad Alon, Krste Asanović, and Borivoje Nikolić. 2016. A RISC-V Vector Processor With Simultaneous-Switching Switched-Capacitor DC–DC Converters in 28 nm FDSOI. *IEEE Journal of Solid-State Circuits* 51, 4 (April 2016), 930–942. https://doi.org/10.1109/JSSC.2016.2519386