High-Performance GPU Implementation of PageRank with Reduced Precision based on Mantissa Segmentation

Thomas Grützmacher
Karlsruhe Institute of Technology
Karlsruhe, Germany
thogru.kit@gmx.de

Hartwig Anzt
Karlsruhe Institute of Technology
Karlsruhe, Germany
hartwig.anzt@kit.edu

Florian Scheidegger
ETH Zürich,
IBM Research–Zürich
Zürich, Switzerland
eid@ibm.zurich.com

Enrique S. Quintana-Ortí
Universitat Jaume I
Castellón, Spain
quintana@uji.es

***Abstract*—We address the acceleration of the PageRank algorithm for web information retrieval on graphics processing units (GPUs) via a modular precision framework that adapts the data format in memory to the numerical requirements as the iteration converges. In detail, we abandon the IEEE 754 single- and double-precision number representation formats, employed in the standard implementation of PageRank, to instead store the data in memory in some specialized formats. Furthermore, we avoid the data duplication by leveraging a data layout based on mantissa segmentation. Our evaluation on a V100 graphics card from NVIDIA shows acceleration factors of up to 30% with respect to the standard algorithm operating in double-precision.**

***Index Terms*—PageRank, web information retrieval, large-scale irregular graphs, adaptive-precision, high performance**

I. Introduction

PageRank is a popular algorithm for web information retrieval used by search engines, i.e., virtual machines created by software that inspect virtual file folders to identify relevant documents [7], [17]. The source of these documents is the World Wide Web (or simply the Web), which consists of an irregular, dynamic, hyperlinked, and dauntingly large[1] collection of web pages. The World's pastime of surfing requires fast response time (speed) and results matching the users' query. Thus, as the amount of information accessible in the Web continues to grow, the requirements imposed on PageRank become more challenging.

PageRank builds upon the principle that "a page is relevant if it is linked by other relevant pages" and the mathematics behind this search algorithm build upon Markov chain theory. From a practical point of view, PageRank boils down to the classical iterative *power method* [11], applied to a large and sparse matrix which reflects the adjacency graph of the search space [17]. Therefore, even when operating with a subset of the Web, PageRank is a *memory-bound* algorithm, which means that its performance and parallel scalability on (most) current architectures is constrained by the memory bandwidth.

The main computational kernel behind PageRank is the *sparse matrix-vector product* (SpMV), a micro-benchmark that has received considerable attention over the past decades because of its myriad applications (see the related work section next). Optimization strategies for SpMV typically aim to reduce the volume of data retrieved from memory by re-organizing the matrix layout in order to diminish the amount of indexing information, like in case of the well-known compressed sparse row (CSR) format [20] or the compressed sparse blocks (CSB) [8].

In this paper we pursue the same general goal —i.e. reducing the amount of data retrieved from memory— but target the data values themselves (instead of the indexing information) for the particular SpMV application in the PageRank algorithm. In more detail, our paper makes the following contributions:

- We formulate a flexible-precision version of PageRank that adaptively increases the precision employed by the algorithm as the iterative process approaches convergence.
- To allow for more flexibility in the precision format, we abandon the conventional IEEE number representation formats to store the data in memory in more efficient formats. In contrast, all arithmetic is performed using the standard IEEE arithmetic supported by current hardware.
- To accommodate the irregular data structures generally appearing in graph algorithms, and in particular in www applications, we rely on the CSR sparse matrix format [20].
- For reference, we also consider the ELL matrix format [16], which is efficient for balanced matrices as it considerably reduces the row indexing information at the cost of padding.
- To obtain a high-performance implementation of SpMV, while avoiding the duplication of the data representing the adjacency matrix (i.e., the problem graph) in different precisions, we store the structure using the customized

[1]In 2018, the dimension of the Web is estimated to be in the order of a few dozens of billions of webpages; see http://www.worldwidewebsize.com/.

precision format based on mantissa segmentation (CPMS) introduced in [13].
- We evaluate the performance and execution flow of the customized precision PageRank algorithm on an NVIDIA V100 GPU board as well as compare the total execution time with the reference PageRank algorithm.

The rest of the paper is structured as follows. In Section II we review recent efforts on optimizing the SpMV kernel for reduced memory access volume and we motivate the study of the CSR and ELL formats in the customized precision PageRank implementation. In Section III we give an overview about the PageRank algorithm before we provide details about our customized precision PageRank realization in Section IV. In Section V we evaluate the customized precision PageRank performance for benchmark problems on a high-end NVIDIA GPU. We conclude in Section VI with an outlook on future work.

II. Related Work

The performance of the power method underlying PageRank is strongly determined by that of SpMV. Optimizing this particular computational kernel is challenging, especially for irregular large-scale problems such as those representing hyperlinked graphs for the Web [9].

Several aspects determine the performance of a particular SpMV implementation. At an abstract level, these aspects can be grouped into four categories: matrix properties, storage format, software techniques, and target architecture [12]. Adding to the complexity of SpMV's ecosystem, the elements in these four categories are often interconnected. Thus, for example, the efficiency of a storage format is highly dependent on the sparsity pattern of the matrix; there exist data layouts that are specifically designed for data-parallel architectures such as GPUs; and some software optimization techniques may not have a positive effect on certain architectures while being key to others.

CSR and COO (coordinate) are likely the two most general sparse matrix formats (i.e., not biased towards any particular sparsity pattern). Intel's MKL provides an efficient implementation of CSR-SpMV for multicore processors that operates with single and double precision operands. Intel's library also contains a blocked variant of CSR (named BSCR) that views the matrix as a sparse collection of tiny dense blocks (e.g., 2×2 or 4×4). This solution aims to improve locality of reference when accessing the data (and, therefore, reduce cache misses), trading off generality and efficiency for certain applications. CSB[2] takes an opposite direction to BCSR, considering the matrix as a dense collection of sparse blocks (with each block stored in CSR format). A potential advantage of CSB is the reduction of indexing information (and, therefore, communication) when accessing the matrix data. In addition, CSB also aims to improve data locality, via a Morton-ordering storage of the block entries.

[2]https://people.eecs.berkeley.edu/~aydin/csb/html/index.html.

The development of high performance implementations of SpMV on graphics processors is an active area of research, to a large extent due to the more complex implementation but appealing superior memory bandwidth available in these type of architectures compared with that of conventional multicore processors. In general, the ELLPACK format and its family of variants (e.g., ELLR-T and SELL-P; see, [5] and the references therein) are especially alluring when the target system is a GPU. However, for irregular applications resulting in unbalanced sparse matrices, a CSR layout complemented with certain highly CUDA-specific techniques can outperform these [10]. GPU implementations of SpMV based on some of these formats are available in NVIDIA's cuSPARSE, MAGMA-sparse [4] and Ginkgo[3]. In this work, we focus on two sparse matrix formats that are at the extreme ends of sparse matrix properties. On the one end, the ELL format is suitable for balanced matrices as it pads all rows to have the same number of nonzero elements. The associated padding overhead yields the the benefit of avoiding storage of row-index information. As community graphs, such as the adjacency graph representing the links connecting web pages are typically irregular, in addition we consider the CSR format. This format exclusively stores the matrix nonzero values; however it needs to accompany the values with a column index and a pointer pointing to the first element in each row. In Fig. 1 we visualize the storage strategies for the CSR and the ELL format.

Adapting the precision of a computation to the "sought-after" final accuracy is a natural technique to accelerate a computational process. The appropriate precision may be dictated by the final requirements from the user, but can also be determined by the level of accuracy necessary at each intermediate step. A clear example is mixed precision with iterative refinement (MPIR), a well-known technique that produces a solution with a high level of accuracy while doing most of the computations in a reduced precision [14]. To some extent, *transprecision* pursues the generalization of this idea, utilizing at each intermediate step the minimum precision necessary to produce a satisfactory final solution. In linear algebra, some examples illustrating the benefits of flexible precision in the solution of sparse linear systems have been presented for the preconditioning of Krylov subspace methods [2] and a Jacobi-based solver [3], [13]. Our GPU implementation of PageRank with adaptive-precision can be regarded as a transprecision solution, as it employs a variable precision level during the process to still produce a solution with basically the same accuracy as an algorithm that operates in the standard double precision arithmetic during all steps.

III. Overview of PageRank and IEEE floating-point representation

PageRank is defined on a directed graph and computes the relevance of each node in this irregular structure. In the setting of web search, each web page is modeled as a node and the

[3]https://github.com/ginkgo-project/ginkgo

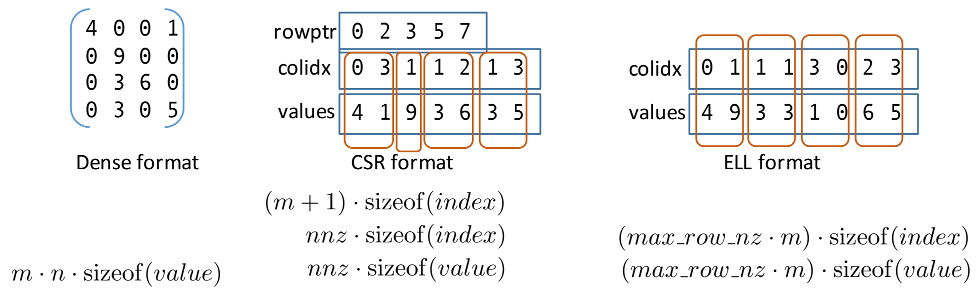
4 0 0 1
0 9 0 0
0 3 6 0
0 3 0 5
Dense format
rowptr 0 2 3 5 7
colidx 0 3 1 1 2 1 3
values 4 1 9 3 6 3 5
CSR format
colidx 0 1 1 1 3 0 2 3
values 4 9 3 3 1 0 6 5
ELL format
$m \cdot n \cdot \text{sizeof}(value)$
$(m+1) \cdot \text{sizeof}(index)$
$nnz \cdot \text{sizeof}(index)$
$nnz \cdot \text{sizeof}(value)$
$(max\_row\_nz \cdot m) \cdot \text{sizeof}(index)$
$(max\_row\_nz \cdot m) \cdot \text{sizeof}(value)$

Fig. 1. The CSR and ELL sparse matrix storage format.

hyperlinks in that web page define directed edges in the model. The relevance is defined recursively across the relevance of neighboring nodes as well as the number of links that point toward a node. The computed ranking captures two important aspects that match well the users' intuition: 1) nodes that are cited often gain relevance; and 2) even though a node might be cited only once, if that citation originates from a node with high relevance, the cited node benefits from the relevance of the origin. PageRank is an iterative method that computes the ranking score of each node up to a certain precision. In a realistic application setting, the users are mainly not interested in the score value itself but rather in the sorted list after the score is computed and mainly in top ranking values. Furthermore, the users rarely look beyond the first dozen of documents retrieved by search engines.

The power method (or power iteration) underlying PageRank is an iterative procedure for the computation of the largest eigenvalue of a matrix [11]. When applied to an adjacency matrix/graph with edges representing hyperlinks and nodes web pages, under mild conditions, this method converges to a score vector representing the probability that a "random" surfer visits each particular page [7].

Let us consider a directed graph $G = (V, E)$ consisting of $n = |V|$ nodes that correspond to web pages, and connected via the $n_z = |E|$ hyperlinks contained in the edge set E. Let $A \in \mathbb{R}^{n\times n}$ be the weighted adjacency matrix with entries defined as follows:

$$A_{ij} = \begin{cases} 1/O_i & \text{if } (i,j) \in E, \\ 0 & \text{otherwise,} \end{cases}$$

where O_i denotes the total number of hyperlinks leaving from node i. Algorithm 1 offers a mathematical formulation of the PageRank algorithm [?]. The iterative scheme produces a sequence of vectors $p^{\{k\}} \in \mathbb{R}^n$, $k = 0, 1, 2, \ldots$, until convergence. Vector $e \in \mathbb{R}^n$ consists of ones only and hence initializes $p^{\{0\}}$ to a uniform distribution across all nodes. The real number δ is the damping factor (usually set to 0.85 [7]) and, together with the stopping threshold ε, directly influence the number of iterations and hence the performance of the algorithm and the precision of the final result.

s corresponds to the probability of the random surfer visiting a site which has no outgoing links, in which case he starts again at a random site. This ensures that the 1-norm of the PageRank vector p equals 1 at every iteration.

Algorithm 1 PageRank(A, ε, δ)

```
1: $p^{\{0\}} := e/n$
2: $\mathbb{S} := \left\{ i \mid \sum_{j=1}^{n} |a_{ij}| = 0 \right\}$    ▷ $\mathbb{S}$ contains all indices of empty rows of A
3: $k := 1$
4: repeat
5:     $s := \sum_{i\in\mathbb{S}} p_i^{\{k-1\}}$
6:     $p^{\{k\}} := \delta A^T p^{\{k-1\}} + (1-\delta)e/n + (s/n) \cdot e$
7:     $\gamma := \|p^{\{k\}} - p^{\{k-1\}}\|_1$
8:     $k := k + 1$
9: until $(\gamma < \varepsilon)$
```

The main computational kernel appearing in the PageRank algorithm is the SpMV involving the transpose of the sparse adjacency matrix A. For a matrix with n_z nonzero entries, this kernel requires $2n_z$ floating-point operations for at least $O(n + n_z)$ memory accesses. (The actual number of memory accesses depends on the sparse data layout chosen for the matrix and the sparsity pattern). This linear relation between flops and memory accesses identifies SpMV as a memory-bound operation, which will proceed at the speed of the memory bandwidth.

High performance implementations of PageRank on current processors operate using the IEEE single precision or double precision floating-point data types supported by the hardware. *A key observation though, is that the native arithmetic realized in floating-point units (FPUs) can be decoupled from the storage format of floating-point numbers, with the latter being a flexible factor under the direct control of the programmer* [13]. In particular, for a memory-bound algorithm such as PageRank, storing the data in memory using a smaller number of bits reduces the pressure on the memory when retrieving the problem data. Prior to the computation, the information can be transformed into one of the natively-supported extended datatypes. The overhead of this transformation can be expected to be amortized by the benefits obtained from the use of high performance, customized realizations of FPUs and also from reduced rounding errors. In any case, compared with the access time, the cost of this transformation is close to negligible, and with some care it can be overlapped with the data transfers from memory.

IV. Design of the CPMS-based PageRank

A. Storage concept in CPMS

In this work, a high precision number is split into equally-sized *segments* [13]. When calculating in lower than full precision, all segments which are not read (from memory) are set to zero (in the processor registers).

Separating the segments is important to enable efficient memory access and avoid data duplication. However, instead of storing each segment in its own dedicated memory space consecutively (as was done in [13]), in our case this is stored in alternating blocks. Each block thus contains values of the same segment and is followed by the block containing values of the next segment. To make memory access cache effective, the size of one bank should have the size of one or multiple cache lines in order to ensure that only the same segment is cached. We choose 128 Byte as the block size to ensure it works for older GPUs like the Kepler K80, which use 128 Byte for L1 cache lines, as well as for new GPUs like the V100 and P100, which have 32 and 64 cache line sizes [15].

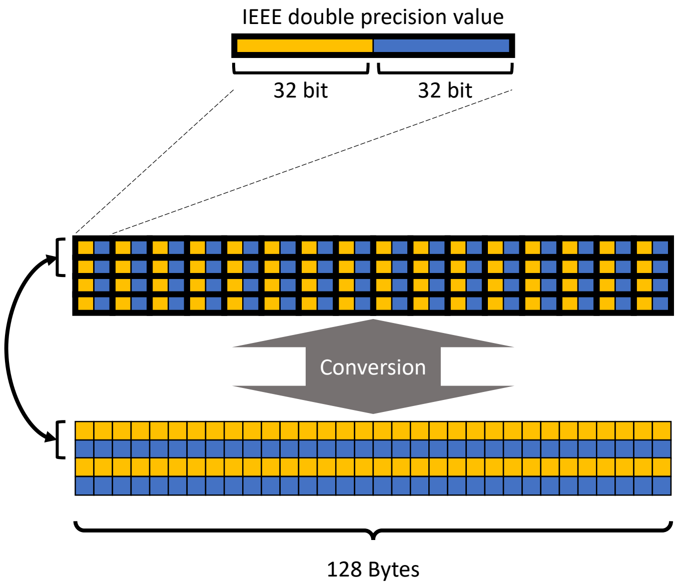

Fig. 2. Conversion example: from an array of IEEE double precision values to an array in CPMS format with 2 segments (and vice versa). The arrow on the side shows the origin of the segments, and where they are located in the different format.

In Fig. 2, we show this conversion for a 2-segment CPMS-based layout. This storage format enables in-place conversion between an array of high precision values and the CPMS format (and vice versa) because it is just a local re-ordering of the data in memory (see the arrow on the side in Fig. 2). For an X-segment CPMS in block format, re-ordering takes place in $X \cdot 128$ Byte blocks, which can be performed in parallel with other re-orderings. This data-parallelism allows the conversion to be performed by the GPU itself.

B. Precision change

Instead of incurring the overhead of using an additional kernel to test if the current precision of the PageRank algorithm needs to be increased, the result of the norm difference in Algorithm 1 line 7 $\gamma = \|p^{\{k\}} - p^{\{k-1\}}\|_1$ can be leveraged for both the convergence check ($\gamma < \epsilon$) and the test for a precision change. Concretely, if γ is close to the current working precision, the following steps are executed:

1) run the PageRank iteration while reading in current precision, and writing in the new, extended precision;
2) normalize the vector in the new precision, so we can ensure that the norm of the PageRank-vector $p^{\{k\}}$ stays at 1; and
3) set the new precision as current working precision;

The normalization is necessary because, while writing back, the floating point representation is "cut", which leads to a rounding towards zero. This, in turn, leads to $\|p^{\{k\}}\| < 1$.

As soon as full precision is reached, the steps above are performed one last time before converting CPMS to an array of IEEE floating-point values with an in-place conversion kernel. This conversion ensures that (1) at the end of the algorithm, all data can be used with any other solver/algorithm that expects an array of IEEE floating point values; (2) penalties that might be caused by the block storage format (e.g., more memory accesses/more cache misses because of scattered sparse values) no longer apply; and (3) the runtime per iteration is the same as the default implementation that uses full precision for the whole solver.

C. Implementation

In our code, all computational intensive tasks are off-loaded to the GPU via kernel calls. These tasks include: (1) All norm functions, in particular, the selective norm calculating s in Algorithm 1 (line 5), the norm of vector differences calculating γ (line 7), and a vector norm used for normalization in precision changes; (2) a vector scaling kernel, used in combination with the vector norm for normalization; (3) two conversion kernels which perform the transformations between IEEE format and CPMS; and (4) one SpMV kernel for each sparse format (line 6). The norm functions are composed of two kernels, which perform a two-step reduction. In the first step, all computational resources of the GPU are used to create an intermediate result vector, which is reduced in the second step to a single value.

The ELL-SpMV kernel uses one thread per row to calculate the result, to ensure consecutive data access for matrix values and column indices. It is similar to the ELL-SpMV kernel proposed in [6], with the additional calculations necessary to compute the new PageRank vector (see Algorithm 1, line 6).

The CSR-SpMV kernel also uses one thread per row to calculate the result. We also explored the possibility of using multiple threads per row, followed by a reduction, to increase performance, as mentioned in [6]. However, augmenting the number of threads processing one matrix row decreases the performance slightly, which might be caused by the fact that the matrices targeted in our experimental evaluation have a low average number of non-zero elements per row (see Table I).

The CPU is commissioned with the build-up (reading matrices, converting to CSR or ELL sparse format), resource management and calling GPU kernels. Because the working precision is decided on the CPU side (dependent on γ), and does not change during a kernel call, we do not need to have conditional branches that decide which segment to load on the GPU side. Instead, we can provide a kernel for each working precision. Rather than writing each kernel by hand, we take advantage of C++ template parameters. As a result, a new kernel is generated by the compiler for each precision, without the need of branching inside the kernel for each load/store operation.

V. Performance Assessment

The experimental analysis was conducted on an NVIDIA V100 "Volta" GPU, with support for CUDA compute capability 7.0 [18]. All GPU kernels were encoded and compiled in the CUDA framework, using CUDA version 9.2. The reference implementation of SpMV and PageRank both employ IEEE double precision (hereafter, flp64). The same type of arithmetic is also used for the floating-point operations of the CPMS-based PageRank, but the memory operations (accesses) and storage use customized precision formats. We consider two CPMS realizations, composed of four 16-bit segments and two 32-bit segments, respectively.

TABLE I
Details of matrices from the SuiteSparse Matrix Collection used in the experimental evaluation.

Name (Abbreviation)	n	n_z	Empty rows
adaptive (Ada)	6,815,744	13,624,320	425,984
delaunay_n22 (Del)	4,194,304	12,582,869	555,807
europe_osm (Eur)	50,912,018	54,054,660	1,616,942
hugebubbles-00020 (Bub)	21,198,119	31,790,179	3,530,509
rgg_n_2_24_s0 (Rgg)	16,777,216	132,557,200	7,121
road_usa (USA)	23,947,347	28,854,312	6,392,288
Stanford (Std)	281,903	2,312,497	20,315
wb-edu (edu)	9,845,725	57,156,537	2,925,419
web-BerkStan (Brk)	685,230	7,600,595	4,744
web-Google (Ggl)	916,428	5,105,039	176,974

For the experimental evaluation we select a set of test matrices, taken from the Suite Sparse matrix collection [1], representing networks. The matrices identifiers along with some key characteristics are listed in Table I. We note that we treat the symmetric test problems as non-symmetric by considering only the lower triangular part. In Fig. 3 we provide details about how the nonzeros are distributed among the rows of the test problems. Even though the test problems have a very unbalanced nonzero distribution, we consider both matrix storage formats, the "irregular-friendly" CSR format and the "GPU-friendly" ELL format. As elaborated in Section II, the

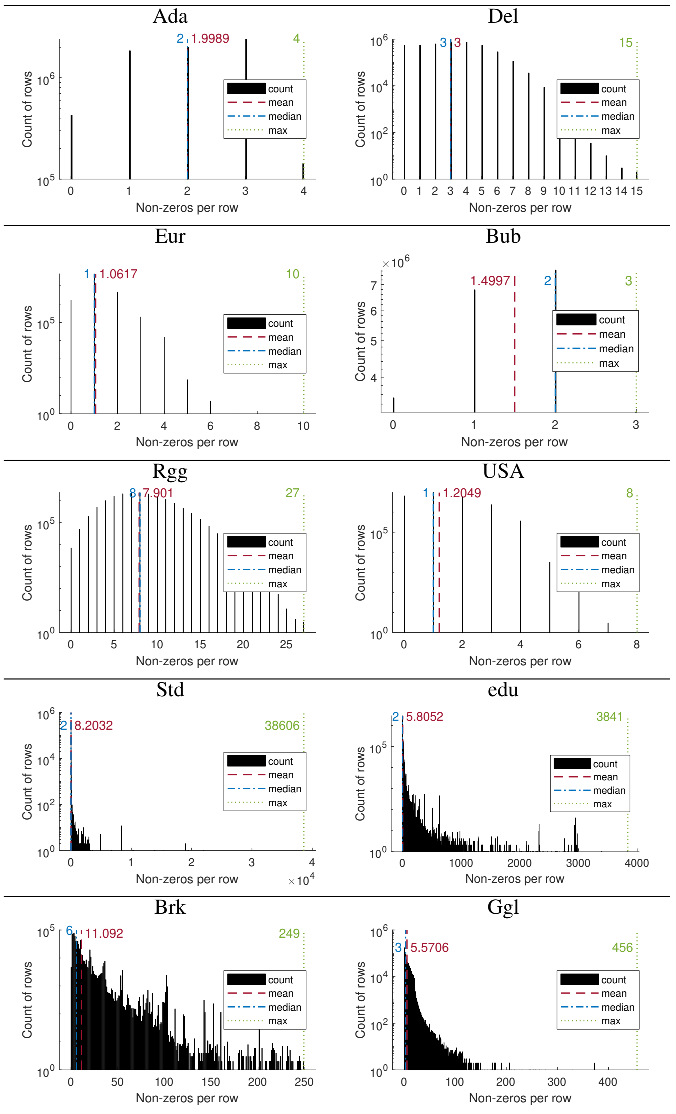

Fig. 3. Nonzero distribution in the test problems.

ELL format pads the rows with explicit zeros to have all rows containing the same number of nonzero elements. For some matrices (concretely, edu, Brk and Ggl), this increases the memory footprint beyond the 16 GBbytes which are available on our V100 GPU. In consequence, these particular problems are not considered in the experiments using the ELL matrix format.

A major goal of the customized precision implementation is to preserve the PageRank convergence of the reference implementation based on IEEE flp64. A few additional iterations may be acceptable, but a significant convergence delay may turn the customized-precision realization unattractive. In Table II we report the convergence details of the PageRank algorithm realized in different environments: the default implementation using IEEE double precision, a customized precision implementation using a 2-segment splitting, and a customized implementation using a 4-segment splitting. For all cases we list the number of iterations completed in the distinct accuracy

TABLE II
ITERATION COUNT OF PAGERANK IN THE DIFFERENT PRECISION FORMATS: FOR THE CPMS, THE TOTAL NUMBER OF ITERATIONS ACCUMULATES FROM THOSE EXECUTED WITH DIFFERENT SEGMENT COUNTS AND STAYS ON PAR WITH THE REFERENCE IMPLEMENTATION.

	fpl64	2-segment CPMS			4-segment CPMS				
	64bit	32bit	64bit	total	16bit	32bit	48bit	64bit	total
Ada	**52**	18	34	**52**	1	16	34	1	**52**
Del	**26**	12	14	**26**	1	11	13	1	**26**
Eur	**112**	48	64	**113**	1	47	62	3	**113**
Bub	**61**	19	42	**61**	1	18	41	1	**61**
Rgg	**106**	41	65	**106**	1	40	63	2	**106**
USA	**31**	14	17	**32**	1	13	16	2	**32**
Std	**118**	51	67	**118**	1	50	64	3	**118**
edu	**114**	47	67	**115**	1	46	65	3	**115**
Brk	**119**	50	69	**119**	1	49	1	68	**119**
Ggl	**116**	47	69	**116**	1	46	67	2	**116**

settings. For the CPMS implementations, we additionally list the total iteration count that accumulates from the iterations in the distinct segment configurations. All implementations generate results of the same quality reducing the residual by at least ten orders of magnitude.

A first observation for the 4-segment splitting is that the first 16-bit segment alone never provides the accuracy necessary to make any progress towards the solution: for all test problems, the algorithm switches to reading 2 segments after the very first iteration. A second observation is that, except for the Brk case, all problems require only a few iterations with 64-bit accuracy, while most iterations in the 4-segment CPMS use 32-bit accuracy (2 segments) or 48-bit accuracy (3 segments).

For the 2-segment CPMS, the number of iterations using full accuracy is slightly higher than the number of iterations using 32-bit accuracy.

The total number of iterations accumulates from the iterations in the distinct segment configurations. For most problems, the CPMS implementations succeed in preserving the convergence of the reference implementations. For some problems, only one additional iterations is required.

The iterations using reduced accuracy may potentially be faster. Generally, the SpMV embedded in the power iteration is the central and most expensive building block of the PageRank iteration. In Fig. 4 we report the speed-up of the SpMV using different segment configurations over the reference implementation using IEEE double precision. We consider both the CSR format (left) and the ELL format (right). The analysis reveals that using 32-bit accuracy in the 2-segment CPMS provides a speed-up of about 1.5× on average (the high speed-up for Rgg comes from cache effects). The 4-segment CPMS is about twice faster for 16-bit accuracy. Using 32-bit accuracy, the 4-segment CPMS are about 30% smaller than the 32-bit accuracy in the 2-segment splitting. Using more than 32-bit accuracy, the CPMS SpMV suffers from the overhead of format conversion and the additional bit used for storing the format information, and is slower than the reference SpMV. Hence, using 64-bit accuracy, the CPMS SpMV is slower than the IEEE756 double precision SpMV.

After analyzing the benefits that CPMS renders to the SpMV, which is the most runtime-intensive part of the PageRank algorithm, we next analyze the execution flow of the complete PageRank algorithm. We choose the algorithm-specific parameters as $\delta = 0.85$, which is a popular choice [19], and a relative accuracy stopping criterion $\varepsilon = 10^{-10}$. In Fig. 5, we show the runtime of the distinct building blocks in the different format configurations for all target problems. For each case, we normalize the times to the total execution time of the reference PageRank (top bar). For the 2-segment CPMS (middle bar) and 4-segment CPMS (bottom bar) we visualize the time spent in the segment configurations. While the iterations in the distinct precision environments correspond to the iteration count analysis in Table II, the CPMS-based realizations of PageRank also include the configuration switching in-between the iteration phases. For some problems (e.g., Std), we notice that the different format and efficiency characteristics of the CSR and the ELL matrix formats result in different performance characteristics of the CPMS PageRank (compare top/bottom plot in Fig. 5). The ratio between the execution times spent in different configurations however remains the same.

Finally, to visualize the benefit of the CPMS PageRank, we show in Fig. 6 the speed-up of the 2-segment realization and the 4-segment realization over the reference implementation. Again, we consider PageRank implementations using the CSR format (left) and the ELL format (right). The speed-up factors for the 2-segment CPMS are generally larger than for the 4-segment CPMS. Interestingly, the 2-segment CPMS realization of PageRank is never slower than the reference PageRank. For Brk, the execution time of the 2-segment CPMS is comparable to that of the reference implementation; for Rgg, it is almost 30% faster. For the remaining test cases, the 2-segment CPMS renders speed-up factors between 1.1× and 1.2×. The speed-up factors for CSR and ELL are similar.

The 4-segment CPMS generally achieves less performance, and in some cases, it is even slower than the reference implementation. One obvious reason is the higher number of precision changes, and the larger overhead in the memory access routines reassembling IEEE double precision numbers from the 16-bit CPMS segments. Also, the memory access routines do not achieve the peak bandwidth when all threads of a warp read 16-bit segments, only. In contrast, the 4-segment

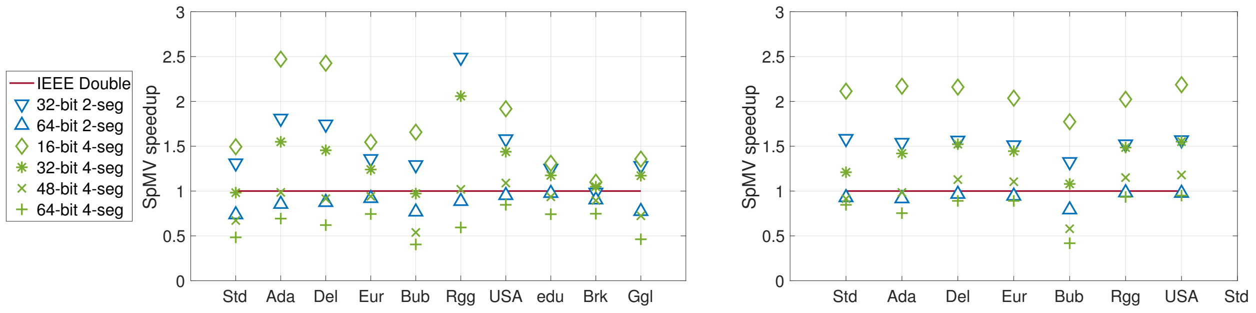

Fig. 4. SPMV Speedup, CPMS compared to IEEE double precision for the CSR format (left) and the ELL format (right).

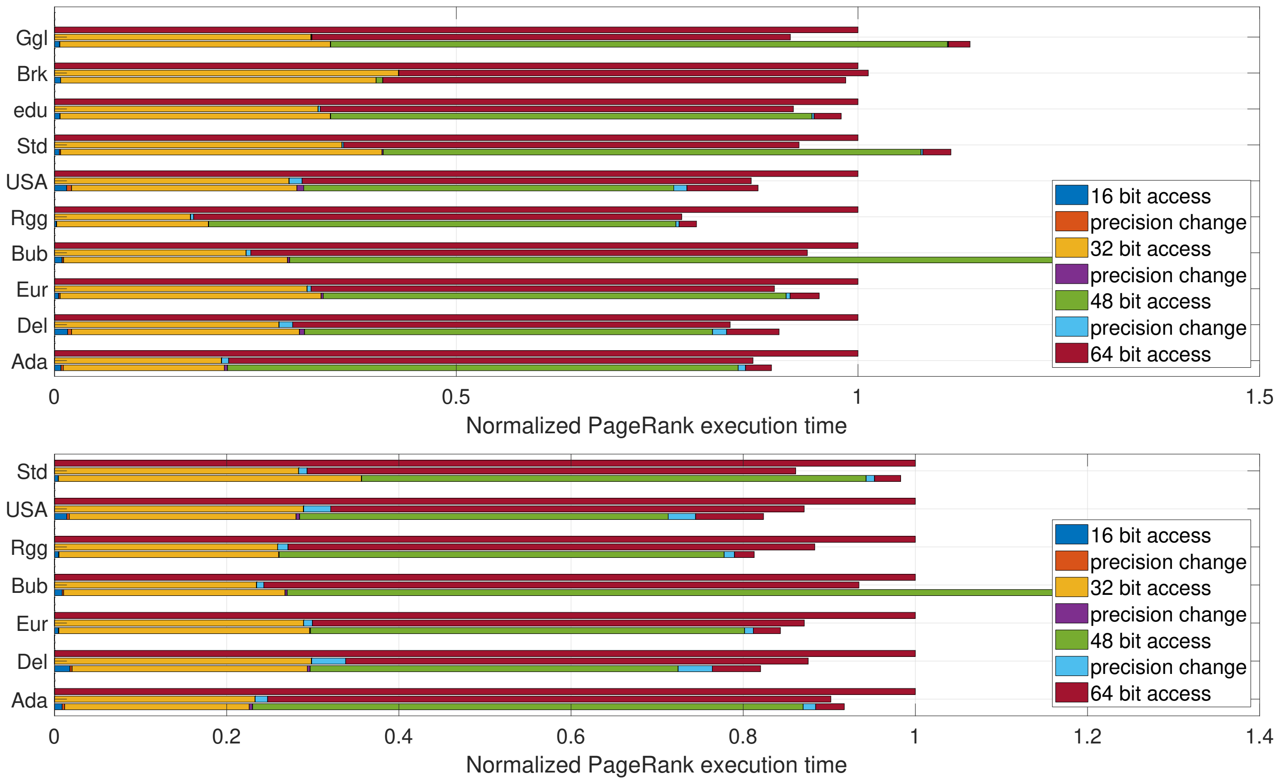

Fig. 5. Normalized PageRank execution time, broken down in time spent at each precision level and the conversion; The "precision change" part contains both the in-place conversion and the normalization of the iteration vector. The top bar shows the runtime in IEEE double precision, the middle bar the runtime with 2-segment and the bottom bar the runtime with 4-segment CPMS. The upper graph uses the CSR format for the sparse system matrix, the lower graph uses the ELL format. For matrices edu, Brk and Ggl the memory requirements exceed the memory capacity of the V100 GPU and, therefore, they are not considered in the experiments with this format.

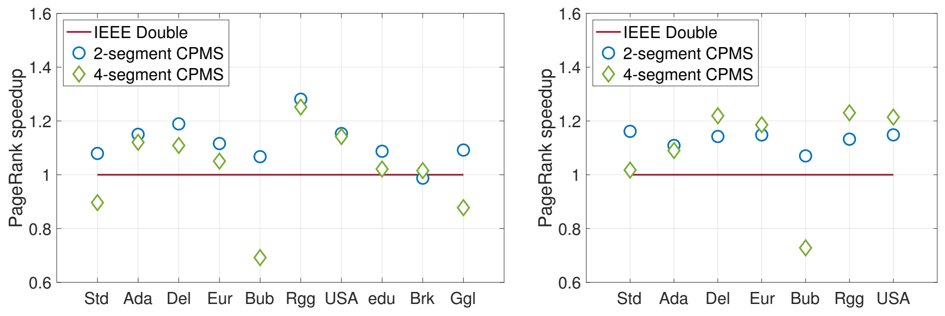

Fig. 6. PageRank Speedup, CPMS vs. IEEE format. CSR sparse format on the left, ELL to the right.

CPMS performs better when using the ELL matrix format.

VI. Summary and Outlook

We have demonstrated that the application of the CPMS-based solution unleashes relevant performance speed-ups for the memory-bound PageRank algorithm, implemented on a GPU, when applied to a set of test problems from the SuiteSparse Matrix Collection. The keys to the acceleration of this algorithm for web information retrieval are 1) an adaptive-precision technique that tunes the precision of the computation as the iteration converges; 2) the selection of a few customized formats different from the IEEE 754 standard; and 3) the elimination of data duplication. Our experimental evaluation on an NVIDIA V100 GPU revealed up to $1.3\times$ speedup with respect to a standard realization that operates in IEEE double-precision.

As part of future work, we plan to explore other sparse matrix layouts for the realization of SpMV on GPUs that can increase performance even further for irregular data structures. In addition, we will pursue the application of CPMS-based solutions to other memory-bound algorithms.

Acknowledgment

H. Anzt was supported by the "Impuls und Vernetzungs-fond" of the Helmholtz Association under grant VH-NG-1241. E. S. Quintana-Ortí was supported by the CICYT project TIN2017-82972-R of the MINECO and FEDER. This work was also supported by the EU H2020 project 732631 "OPRECOMP. Open Transprecision Computing", and the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Awards Number DE-SC0016513 and DE-SC-0010042.

References

[1] Suitesparse matrix collection. https://sparse.tamu.edu, 2018. Accessed in April 2018.

[2] Hartwig Anzt, Jack Dongarra, Goran Flegar, Nicholas J. Higham, and Enrique S. Quintana-Ortí. Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers. *Concurrency and Computation: Practice and Experience*, 2018.

[3] Hartwig Anzt, Jack Dongarra, and Enrique S. Quintana-Ortí. Adaptive precision solvers for sparse linear systems. In *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing*, E2SC '15, pages 2:1–2:10, New York, NY, USA, 2015. ACM.

[4] Hartwig Anzt, Mark Gates, Jack Dongarra, Moritz Kreutzer, Gerhard Wellein, and Martin Köhler. Preconditioned Krylov solvers on GPUs. *Parallel Computing*, 68:32–44, oct 2017.

[5] Hartwig Anzt, Stanimire Tomov, and Jack Dongarra. Implementing a Sparse Matrix Vector Product for the SELL-C/SELL-C-σ formats on NVIDIA GPUs. Technical Report ut-eecs-14-727, University of Tennessee, March 2014.

[6] Nathan Bell and Michael Garland. Efficient sparse matrix-vector multiplication on CUDA, December 2008.

[7] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *Seventh International World-Wide Web Conference (WWW 1998)*, 1998.

[8] Aydın Buluç, Samuel Williams, Leonid Oliker, and James Demmel. Reduced-Bandwidth Multithreaded Algorithms for Sparse Matrix-Vector Multiplication. In *Proc. IPDPS*, pages 721–733, 2011.

[9] Daniele Buono, John A. Gunnels, Xinyu Que, Fabio Checconi, Fabrizio Petrini, Tai-Ching Tuan, and Chris Long. Optimizing sparse linear algebra for large-scale graph analytics. *Computer*, 48(8):26–34, Aug 2015.

[10] Goran Flegar and Enrique S. Quintana-Ortí. Balanced csr sparse matrix-vector product on graphics processors. In Francisco F. Rivera, Tomas F. Pena, and José C. Cabaleiro, editors, *Euro-Par 2017: Parallel Processing*, pages 697–709, Cham, 2017. Springer International Publishing.

[11] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 3rd edition, 1996.

[12] Max Grossman, Christopher Thiele, Mauricio Araya-Polo, Florian Frank, Faruk O. Alpak, and Vivek Sarkar. A survey of sparse matrix-vector multiplication performance on large matrices. *CoRR*, abs/1608.00636, 2016.

[13] Thomas Grützmacher and Hartwig Anzt. A modular precision format for decoupling arithmetic format and storage format. In *Lecture Notes in Computer Science, 16th Int. Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms – HeteroPar'18*. Springer, 2018. To appear.

[14] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2 edition, 2002.

[15] Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele P. Scarpazza. Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking. Technical report, apr 2018.

[16] Moritz Kreutzer, Georg Hager, Gerhard Wellein, Holger Fehske, and Alan R. Bishop. A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units. *SIAM J. Scientific Computing*, 36(5):C401–C423, 2014.

[17] Amy N. Langville and Carl D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, Princeton, NJ, USA, 2012.

[18] NVIDIA Corp. Whitepaper: NVIDIA TESLA V100 GPU ARCHITECTURE, 2017.

[19] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the Web. In *Proceedings of the 7th International World Wide Web Conference*, pages 161–172, Brisbane, Australia, 1998.

[20] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2nd edition, 2003.