Document downloaded from:

http://hdl.handle.net/10251/183010

This paper must be cited as:

Gruetzmacher, T.; Cojean, T.; Flegar, G.; Anzt, H.; Quintana-Orti, ES. (2020). Acceleration of PageRank with customized precision based on mantissa segmentation. ACM Transactions on Parallel Computing. 7(1):1-19. https://doi.org/10.1145/3380934

The final publication is available at

https://doi.org/10.1145/3380934

Copyright Association for Computing Machinery

Additional Information

Acceleration of PageRank with Customized Precision based on Mantissa Segmentation

THOMAS GRÜTZMACHER, Karlsruhe Institute of Technology, Germany
TERRY COJEAN, Karlsruhe Institute of Technology, Germany
GORAN FLEGAR, Universitat Jaume I, Spain
HARTWIG ANZT, Karlsruhe Institute of Technology, Germany and University of Tennessee, USA
ENRIQUE S. QUINTANA-ORTÍ, Universitat Politècnica de València, Spain

We describe the application of a communication-reduction technique for the PageRank algorithm that dynamically adapts the precision of the data access to the numerical requirements of the algorithm as the iteration converges. Our variable-precision strategy, using a customized precision format based on mantissa segmentation (CPMS), abandons the IEEE 754 single- and double-precision number representation formats employed in the standard implementation of PageRank, and instead handles the data in memory using a customized floating-point format. The customized format enables fast data access in different accuracy, prevents overflow/underflow by preserving the IEEE 754 double-precision exponent, and efficiently avoids data duplication since all bits of the original IEEE 754 double-precision mantissa are preserved in memory, but re-organized for efficient reduced precision access. With this approach, the truncated values (omitting significand bits), as well as the original IEEE double-precision values can be retrieved, without duplicating the data in different formats.

Our numerical experiments on an NVIDIA V100 GPU (Volta architecture) and a server equipped with two Intel Xeon Platinum 8168 CPUs (48 cores in total) expose that, compared with a standard IEEE double-precision implementation, the CPMS-based PageRank completes about 10% faster if high-accuracy output is needed, and about 30% faster if reduced output accuracy is acceptable.

CCS Concepts: • **Mathematics of computing → Computations on matrices**; • **Computing methodologies → Massively parallel algorithms**;

Additional Key Words and Phrases: PageRank, large-scale irregular graphs, adaptive-precision, high-performance, multi-core processors, GPUs

ACM Reference format:
Thomas Grützmacher, Terry Cojean, Goran Flegar, Hartwig Anzt, and Enrique S. Quintana-Ortí. 2018. Acceleration of PageRank with Customized Precision based on Mantissa Segmentation. *ACM Trans. Parallel Comput.* 0, 0, Article 0 (2018), 20 pages.
https://doi.org/0000001.0000001

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Association for Computing Machinery.
Manuscript submitted to ACM

1 INTRODUCTION

As of 2018, the World Wide Web (or simply the "Web") is estimated to consist of a few dozens of billions of webpages,[1] with its increasing popularity fueled by search engines, that is, virtual machines created by software that inspect virtual file folders to identify relevant documents [Brin and Page 1998; Langville and Meyer 2012]. This huge collection of web pages continues to grow in dimension, irregularity, and number of hyperlinks. Consequently, there is an urgent need for search engines that can provide fast yet accurate results matching the users' queries.

PageRank is a popular algorithm for web information retrieval used by search engines. The mathematics underlying PageRank build upon Markov chain theory, exploiting the principle that "a page is relevant if it is linked by other relevant pages". PageRank applies the classical iterative *power method* [Golub and Loan 1996] to the matrix associated with the adjacency graph of the search space [Langville and Meyer 2012].

The main computational kernel behind PageRank is the *sparse matrix-vector product* (SpMV). This is a *memory-bound* kernel whose performance is constrained by the memory bandwidth on virtually all current architectures. In consequence, PageRank is a memory-bound algorithm. The SpMV kernel has received considerable attention over the past decades due to its widespread use in many scientific and engineering applications (see Section 2). In this paper, we address the memory-bound nature of PageRank by reducing the volume of numerical data retrieved from memory in a high-performance implementation of SpMV. The key in our approach is to exploit the fixed-point nature of the PageRank algorithm. Specifically, the PageRank algorithm allows to successively increase the data accuracy as the iterations converge towards the correct answer.

We previously explored the idea of decoupling the arithmetic precision from the memory precision [Anzt et al. 2019; Grützmacher et al. [n. d.]] and dynamically adapting the memory precision to the numeric requirements of the PageRank algorithm in [Grüztmacher et al. 2018]. There we implemented a particular instance of a customized precision based on mantissa segmentation (CPMS) that splits a 64-bit floating-point number into two or four equally-sized segments of 32 or 16 bits each, respectively. To attain a cache-effective memory access on NVIDIA's K80, P100 and V100 GPUs, our implementation in [Grüztmacher et al. 2018] interleaves the information in banks of 128 bytes (which matches the smallest integer multiple of the cache line size for all three GPU generations). Thus, for example, if we decide to split each number into four segments, say s1, s2, s3 and s4, we store consecutively in memory 128 bytes of information corresponding to segment s1 of the first 64 numbers ($64 \cdot 16$ bits = $64 \cdot 2$ bytes = 128); then 128 bytes for segment s2 of the same first 64 numbers; and when we are done with all the segments for the first 64 numbers, we continue with the next 64 ones; see [Grüztmacher et al. 2018] for details.

In this paper, we extend our prior work in [Grüztmacher et al. 2018] by proposing a more flexible CPMS technique which can operate with arbitrary bank sizes corresponding to any integer multiple of 64 bytes. For this purpose, we employ C++ static polymorphism and pass the bank size as a template parameter. Furthermore, we extend the scope of the CPMS-based PageRank to CPU architectures. This requires not only a more sophisticated kernel design to enable efficient AVX vectorization, but also taking into account the implications of prefetching during memory accesses. Therefore, we additionally consider a CPMS format separating the segments into distinct memory locations. In particular, the deep cache hierarchies and the more complex compilers available for CPUs make it much harder to fully optimize the non-standard memory access routines of CPMS on this type of architectures. Finally, the experimental results presented in this paper differ from those in our previous work in that we do not consider the lower triangular part of the matrix representation of symmetric (undirected) community graphs, but the full matrix.

[1] http://www.worldwidewebsize.com/

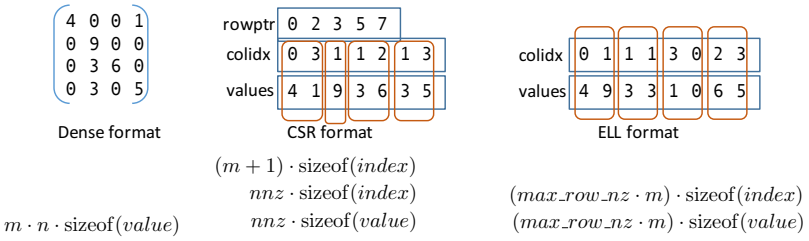


Fig. 1. The CSR and ELLPACK sparse matrix storage format along with memory requirements.

2 RELATED WORK

The performance of PageRank is dictated by the power method which, in turn, is strongly determined by the SpMV kernel.

The performance of a particular SpMV implementation on a specific hardware reflects the complex interactions of different aspects: 1) the volume of matrix sparsity pattern information (in general, indexes); 2) the volume of matrix and vector numeric data (that is, the values); 3) the irregular access to the vector values (determined by the matrix sparsity pattern) and its interaction with the cache configuration and sizes and the code access order; and 4) the workload imbalance in a parallelized setting. All these aspects have to be considered when optimizing an SpMV kernel for a specific hardware architecture [Grossman et al. 2016]. Generally, matrices with an irregular distribution of their nonzero entries are more challenging, since their parallelization usually results in significant workload imbalance and a low cache hit rate. In some cases, an initial preprocessing step, which reorders the matrix rows/columns, can be useful to improve load balancing and data locality [Kreutzer et al. 2014].

Among the most popular general sparse data layouts are the CSR (compressed storage row) and COO (coordinate) formats as those are not biased towards any particular sparsity pattern. The specialized ELLPACK format [Bell and Garland 2009] pads all rows to have the same number of nonzero elements, incurring a certain overhead. While this padding increases the matrix numeric data volume, it yields the benefit of avoiding the storage of row-index information needed in the CSR format, see Figure 1. From the perspective of parallelization, the ELLPACK format is especially appealing for SIMD-parallel architectures, such as GPUs, as it removes branch divergence and ensures uniform memory access. However, it is only attractive for balanced matrices in which the nonzero elements are evenly distributed among the rows so that padding to a uniform nonzero count introduces a moderate overhead.

For multi-core CPUs, Intel's MKL (Math Kernel Library) offers a multi-threaded efficient instance of CSR-SpMV as well as a multi-threaded blocked variant, named BCSR, which views the matrix as a sparse collection of tiny dense blocks (e.g., 2×2 or 4×4). BCSR 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, by considering the matrix as a dense collection of sparse blocks (with each block stored in CSR format). CSB aims to decrease the amount of indexing information and, therefore, reduces communication when accessing the matrix sparsity data information. Additionally, CSB also aims at improving data locality, via a Morton-ordering storage of the block entries.

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

While there exists a long list of SpMV kernels available in vendor libraries such as NVIDIA's cuSPARSE and the community ecosystems MAGMA-sparse [Anzt et al. 2017] and Ginkgo[3], optimizing the SpMV kernel for graphics processors remains an active field of research. Conceptually, for well-balanced sparse matrices, formats such as ELLPACK [Bell and Garland 2009] as well as some sophisticated variants [Kreutzer et al. 2014] increase the numeric data volume by explicitly storing zero elements for SIMD-friendly data access and kernel execution. For irregular and unbalanced matrix sparsity structures, the CSR or COO formats minimize the numeric data volume, and SpMV kernels are tuned to offer better load balance arithmetic work and memory access across the compute resources [Flegar and Quintana-Ortí 2017].

The optimization we pursue in this paper is orthogonal to the previously listed format optimizations as we neither address load balancing, nor explicitly store zero values for SIMD-friendly execution. Instead, we keep the sparse storage formats unchanged, but modify the precision format used for storing the numeric values. Specifically, employing the customized format based on mantissa segmentation allows to retrieve the numeric values of the matrix and the vector with reduced accuracy by accessing only part of the mantissa bits. This will change the SpMV output vector, and can only be used with careful consideration of the numeric effects and their impact on the top-level algorithm.

In this work, we present customized precision solutions for both CPU and GPU architectures, and focus on two sparse matrix formats that are at the extreme ends of sparse matrix properties: The SIMD-friendly ELLPACK format, suitable for efficiently processing balanced matrices; and the CSR format, which exclusively stores the nonzero elements and therefore is attractive for irregular sparsity patterns.

In our work, we employ the volume-reducing customized precision format only for the memory operations while handling all arithmetic in the hardware-supported IEEE 754 double-precision format. A popular approach for mixing different precision formats to reduce the runtime of an algorithm is the concept of mixed-precision with iterative refinement (MPIR). MPIR is a well-known technique that solves with a high level of accuracy while doing most of the computations in reduced precision [Higham 2002]. The central idea is to repeatedly solve an error correction system in lower than working precision, and update the high precision solution approximation until the residual norm drops below an acceptable threshold [Carson and Higham 2017]. This strategy can even be cascaded to solve the error equations in lower precision formats recursively [Carson and Higham 2018]. While also cascaded iterative refinement typically employs the IEEE 754 standard precision formats, *transprecision* pushes the idea of accuracy adaptation even further, utilizing at each intermediate step the minimum precision necessary to produce a satisfactory final solution. What all these approaches share is the strict coupling between arithmetic precision and memory precision. While this strategy may be reasonable for compute-bound algorithms, the PageRank algorithm is heavily memory-bound, and the cost of the memory operations dictates its performance [Grüztmacher et al. 2018]. This motivates us to decouple the memory access format from the arithmetic format, while still using the hardware-supported IEEE standard precision formats in all arithmetic operations [Anzt et al. 2019; Grützmacher et al. [n. d.]]. Some examples illustrating the modular precision approach for algorithms in sparse linear algebra are the adaptive-precision block-Jacobi preconditioner for Krylov subspace methods [Anzt et al. 2018] and a Jacobi iterative solver [Anzt et al. 2015; Grützmacher and Anzt 2019; Grützmacher et al. [n. d.]].

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

3 A BRIEF INTRODUCTION TO PAGERANK

In the setting of web search, PageRank estimates the relevance of a web page by recursively inspecting the relevance of "neighboring" web pages as well as the number of links that point toward the page [Langville and Meyer 2012; Page et al. 1998]. From a practical point of view, PageRank is applied to a directed graph that represents the web pages as nodes and the hyperlinks connecting two pages as directed edges between the corresponding nodes, yielding a score vector that captures the probability that a "random" surfer visits a particular page [Brin and Page 1998]. Furthermore, PageRank only needs to compute the ranking score of each Web page up to a certain precision, as Web surfers are often only interested in the first dozen of documents retrieved by the search engines and usually do not care about the precise score values.

Consider a directed graph $G = (V, E)$ with $n = |V|$ nodes/web pages, connected via a collection of edges/hyperlinks E. Let $A \in \mathbb{R}^{n \times n}$ be a weighted adjacency matrix associated with (V, E), with weights/entries defined so that $A_{ij} = 1/O_i$ if the edge $(i, j) \in E$, or $A_{ij} = 0$, otherwise; here, O_i denotes the total number of hyperlinks leaving from node i. The mathematical representation of PageRank in Algorithm 1 computes a sequence of vectors $p^{\{k\}} \in \mathbb{R}^n$, $k = 0, 1, 2, \ldots$, until convergence [Langville and Meyer 2012]. The damping factor δ and the stopping threshold ε determine the convergence of the procedure and precision of the final result. Furthermore, $e \in \mathbb{R}^n$ is a vector set to ones to initialize $p^{\{0\}}$ to a uniform distribution across all nodes; and $s \in \mathbb{R}$ corresponds to the probability of the random surfer visiting a site which has no outgoing links; see [Langville and Meyer 2012] for further details.

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 indexes of empty rows of A
3: $k := 1$
4: **repeat**
5: $\quad s := \sum_{i \in \mathbb{S}} p_i^{\{k-1\}}$
6: $\quad p^{\{k\}} := \delta A^T p^{\{k-1\}} + (1 - \delta)e/n + (s/n) \cdot e$
7: $\quad \gamma := \|p^{\{k\}} - p^{\{k-1\}}\|_1$
8: $\quad k := k + 1$
9: **until** $(\gamma < \varepsilon)$

The iterative procedure that underlies the realization of PageRank in Algorithm 1 is the classical power method for the computation of the largest eigenvalue of a matrix [Golub and Loan 1996]. When applied to the adjacency matrix representing a collection of Web pages, the main computational kernel appearing in the PageRank algorithm is the SpMV involving the sparse adjacency matrix A. For a matrix with n_z nonzero entries (where n_z is the cardinality of E), this kernel performs $2n_z$ floating-point operations and, at least, $n + n_z$ memory accesses. In consequence, SpMV is a memory-bound operation on virtually all current hardware architectures.

PageRank is usually encoded using the conventional IEEE single-precision or double-precision floating-point data types natively supported by most hardware architectures. A key idea in [Grützmacher and Anzt 2019] is that *"the 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"*. This is particularly interesting for the SpMV kernel as storing (and retrieving) the data values of the sparse matrix and the vector in lower precision paves

the road toward communication-reduction techniques which lower the pressure on the memory bandwidth. For a memory-bound algorithm such as PageRank, reducing the data access volume can accelerate the kernel execution.

4 PAGERANK WITH ADAPTIVE PRECISION

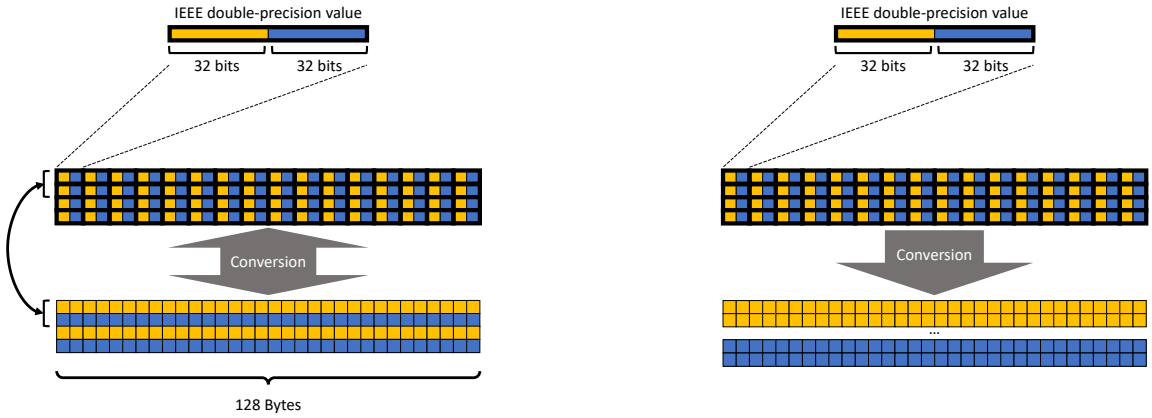


Fig. 2. CPMS-i (left): conversion between the IEEE double-precision format and interleaved 2-segment CPMS; the bank size in this interleaving CPMS is chosen as 128 bytes. CPMS-s (right): CPMS separating the segments in memory.

4.1 Segment-based storage in CPMS

The CPMS format splits a "high precision" floating-point number into several equally-sized *segments* [Grützmacher and Anzt 2019] so that, if lower data access accuracy is acceptable, only a subset of the segments is retrieved from the memory into the processor registers. The remaining bits in the significand of the FP64 value are automatically filled with zeros.

The purpose of this approach is that by retrieving values with lower accuracy, the memory transfer volume is reduced in comparison to FP64 access. As all bits of the high precision value are still available in memory though, the data does not need to be duplicated in memory to enable different precision formats.

The segments can be stored completely separated in memory, such that each segment type has its own contiguous memory space (we name this variant as"CPMS-s") or, alternatively, the memory segments can be interleaved, so that different segment types follow each other in memory in a predefined bank pattern (named as "CPMS-i"). These two alternatives are illustrated in Figure 2. The dimension of the interleaved banks in CPMS-i is a platform-dependent parameter and, in practice, it should be an integer multiple of the cache line size.

The advantage of interleaving the banks in CPMS-i is that the data can be converted in-place in-between the precision formats (for moderate bank sizes). The left-hand side in Figure 2 illustrates this conversion for a 2-segment CPMS-based layout. The in-place conversion between an array of high precision values and the CPMS-i format consists of a local reordering of the bits in memory (see the arrow on the side in Figure 2). For an x-segment CPMS in block format, reordering operates on $x \cdot bs$ byte blocks, where bs is the bank size. This reordering step can exploit data parallelism, which enables efficient conversion on SIMD-type architectures.

In general, architectures with support for prefetching will likely fail to ignore the tail-segments when accessing the data in reduced precision. Hence, we expect that CPMS-s (which separates the segments in memory) can achieve higher performance on CPU architectures. Hence, the conversion between IEEE standard precisions and the CPMS-s format cannot be realized in-place but requires additional workspace. Also, as the conversion requires information about the offset in-between the distinct segment types, the information about the total memory requirements has to be available before the algorithm invocation. This makes the CPMS-s less flexible and more resource-demanding compared to CPMS-i. As memory is a scarce resource on GPU accelerators, on these architectures, we consider only CPMS-i with a bank size of 128 bytes. This setup aims to attain high-performance on older GPU architectures that feature L1 cache lines of 128 bytes (such as NVIDIA's K80) as well as newer GPUs with cache line sizes of 64 and 32 bytes, such as NVIDIA's P100 and V100 [Jia et al. 2018].

4.2 Adaptive precision

The proposed variant of PageRank with adaptive precision access leverages the norm difference $\gamma = \|p^{\{k\}} - p^{\{k-1\}}\|_1$ in Algorithm 1–line 7 to check for residual stagnation. This event indicates either convergence ($\gamma < \epsilon$) or the need for higher access accuracy [Grüztmacher et al. 2018].

Concretely, if γ is close to the current data access precision, the following steps are invoked:

(1) execute a PageRank iteration reading in the current memory precision and writing in the new, extended memory precision (on-the-fly conversion);
(2) normalize the vector in the new memory precision, to preserve the unit-norm of the PageRank-vector $\|p^{\{k\}}\|_1 = 1$; and
(3) set the new memory precision as default data access precision;

The normalization is necessary as the high precision values returned from the arithmetic operations are truncated in the conversion to customized precision by cutting off mantissa bits (rounding towards zero). This can result in $\|p^{\{k\}}\|_1 < 1$.

Once the data access uses 64-bit accuracy, the values stored in CPMS are converted back to the standard IEEE floating-point values. This conversion ensures that the standard 64-bit memory access can be used (without incurring on-the-fly format conversion overhead), and that all data is available in IEEE double-precision after completion of the PageRank algorithm.

4.3 GPU implementation of PageRank with CPMS

The GPU implementation of PageRank in [Grüztmacher et al. 2018] offloads all computationally-intensive tasks to the GPU via CUDA kernel calls. This includes: (1) All norm functions, in particular, the selective norm calculating s in Algorithm 1 (line 5), the calculation of the vector difference norm γ (line 7), and the 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 handle the transformations between the IEEE format and CPMS; and (4) the SpMV kernel potentially adapting the data access precision (line 6).

For the SpMV kernel, we consider the CSR and ELLPACK sparse matrix formats. FOR SIMD-execution, the ELLPACK-SpMV kernel assigns one thread to each row, which ensures coalesced access to the matrix values and column indexes.

The CSR-SpMV kernel also maps one thread to each row. This parallelization strategy is motivated by preliminary experiments indicating that assigning multiple threads to a single row, and complementing the parallel computation of partial sums with a reduction step, results in low performance for the target problems. The reason behind is that the

test matrices contain only a few nonzero elements in most rows, which is characteristic for graph problems like social networks (see Table 1).

The CPU is commissioned with the build-up process (reading matrices, converting to CSR or ELLPACK sparse format), resource management, and GPU kernel invocation. Also the memory access precision is controlled by the CPU, which includes monitoring the values of γ. This avoids branching inside the memory access routines and thread divergence. Instead of implementing several distinct precision-specific memory access routines, we take advantage of C++ template parameters. This way, specific kernels for all required precisions are generated by the compiler, without the need of branching inside the kernel for each load/store operation. The data access and the conversion between the formats is realized using the built-in `reinterpret_cast` function; see Listing 1. In our implementation, we use partial specialization to allow recursive function calls. This enables us to cover all conversions with one read and one write function. A simplified version of reading the head in a 2–segment CPMS is provided in Listing 1. The pseudocode there reflects all central components of the actual implementation, but omits template parameters and other details to improve readability.

```
double reinter_read_head(const int32_t *segments, int index) {
    double result = 0.0;
    int32_t *parts = reinterpret_cast<int32_t *>(&result);
    int headIndex = convertToHeadIndex(index);    // returns the head index of the corresponding value
    parts[1] = segments[headIndex];
    return result;
}
```

Listing 1. Read function using a `reinterpret_cast` for reading the head in a 2-segment CPMS

4.4 CPU implementation of PageRank in CPMS

While the GPU realization of CPMS leverages the built-in `reinterpret_cast` function for splitting and recovering IEEE standard precision formats, the more complex CPU compilers are unable to apply the same optimization steps to the `reinterpret_cast` function that succeed for IEEE standard precision formats. In response, we implement a more sophisticated strategy that enables compiler optimizations, see Listing 2.

```
double union_read_head(const int32_t *segments, int index) {
    union conversion { double dbl; int64_t it;};
    const int headIndex = convertToHeadIndex(index);    // Stores the index of the head of the corresponding value.
    conversion result;
    result.it = segments[headIndex];
    result.it = result.it << 32;    // head must be moved to the proper position due to little endian format
    return result.dbl;
}
```

Listing 2. Read function using a `union` for reading the head in a 2-segment CPMS

To ensure a fair comparison between the PageRank using IEEE-based memory access and the PageRank using CPMS memory access, the code for all kernels is identical except for the memory accesses. We realize this via C++ templates, as shown in the example in Listing 3. There, we compute the 1-norm of a vector `vec` with `n` elements. The vector itself is encapsulated in the class `SplitPointer<TotalNumberSegments>`, which decouples the storage layout from the arithmetic format. `SplitPointer<1>` uses the IEEE format underneath, while `SplitPointer<2>`, for instance, uses a 2-segment CPMS. The template parameter `SegmentsToUse` specifies the precision used in the read access by adjusting the number of segments which are retrieved from memory.

This way, templating all kernels allows us to switch between FP64, CPMS-s, and CPMS-i without modifying any computational kernel. As a side effect, templating the kernels reduces redundancy and thus improves the maintainability of the code stack.

```
1  template<int SegmentsToUse, int TotalNumberSegments>
2  double norm1(int n, const SplitPointer<TotalNumberSegments> vec)
3  {
4      double sum = 0.0;
5      #pragma omp parallel for reduction(+:sum) schedule(static, chunkSize)
6      for (int i = 0; i < n; ++i) {
7          sum += abs(vec.template read<SegmentsToUse>(i));
8      }
9      return sum;
10 }
```

Listing 3. Templated 1-norm calculation, allowing for different storage formats.

The CPU-code is parallelized using OpenMP, as shown in the example Listing 3–line 5. To parallelize "for-loops" we apply a `static` strategy if the workload is balanced across all iterations, or the `guided` strategy for workloads where some workload imbalance may occur, e.g. the row-parallelized SpMV calculations.

To leverage the SIMD core architecture, we vectorize all compute-intensive kernels. We choose the AVX2 instruction set to support a wide variety of architectures while gaining a significant performance boost.

5 PERFORMANCE ASSESSMENT

5.1 Hardware and software environment

The reference implementations for SpMV and PageRank employ IEEE double-precision (hereafter, FP64) in all arithmetic and memory operations. FP64 arithmetic is also used for all the floating-point operations in the CPMS SpMV and the CPMS-based PageRank, but the memory operations (accesses) use the segmentation-oriented customized precision formats. We consider two CPMS realizations, consisting of four 16-bit segments and two 32-bit segments, respectively.

The experimental analysis of CPMS for data-parallel accelerators are conducted on an NVIDIA V100 "Volta" GPU, with support for CUDA compute capability 7.0 [NVIDIA Corp. 2017]. All GPU kernels are encoded and compiled in the CUDA framework, using CUDA version 9.2.

The CPU employed in the experiments is a node from JUWELS (at Jülich supercomputing center), equipped with two Intel(R) Xeon(R) Platinum 8168 sockets (24 cores/socket, 2.70 GHz) and 94 GiB of RAM. The experiments with these CPU map one thread per core, using all 48 cores of the node. The compiler is GCC version 8.2.0.

5.2 Test problems and PageRank convergence

For the experimental evaluation, we select a set of test matrices, taken from the Suite Sparse matrix collection [SuiteSparse 2018], representing social networks and (to increase the experimentation database) distribution infrastructures. The matrix identifiers, along with some key characteristics, are listed in Table 1. We note that, conversely to the results that were reported in the initial paper [Grüztmacher et al. 2018], here we consider the problems as full matrices with elements above and below the main diagonal by correctly handling the "symmetric" flag in the matrix header. In Figure 3, we provide details about the row distribution of nonzeros. Even though the test problems present a highly unbalanced nonzero distribution, we consider both the "irregular-friendly" CSR format and the "GPU-friendly" ELLPACK format. As elaborated in Section 2, the ELLPACK format pads the rows with explicit zeros to enforce that all rows contain

Table 1. Test matrices from the SuiteSparse Matrix Collection along with the number of rows (n)and the number of nonzero elements (n_z).

Name (Abbreviation)	n	n_z	Empty rows
adaptive (Ada)	6,815,744	27,248,640	0
delaunay_n22 (Del)	4,194,304	25,165,738	0
europe_osm (Eur)	50,912,018	108,109,320	0
hugebubbles-00020 (Bub)	21,198,119	63,580,358	0
rgg_n_2_24_s0 (Rgg)	16,777,216	265,114,400	1
road_usa (USA)	23,947,347	57,708,624	0
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

Table 2. 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.

	FP64	2-segment CPMS			4-segment CPMS				
	64bit	32bit	64bit	total	16bit	32bit	48bit	64bit	total
Ada	**65**	7	58	**65**	1	1	81	1	**84**
Del	**64**	15	49	**64**	1	21	64	2	**88**
Eur	**118**	50	68	**118**	1	50	65	3	**119**
Bub	**77**	12	42	**77**	1	1	77	3	**82**
Rgg	**89**	28	61	**89**	1	1	89	2	**93**
USA	**117**	51	66	**117**	1	50	54	2	**117**
Std	**118**	51	67	**118**	1	50	64	3	**118**
edu	**114**	47	67	**114**	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**

the same number of nonzero elements. For some matrices (concretely, edu, Brk and Ggl), this increases the memory footprint beyond the 16 GiB that is available on our V100 GPU or even the memory of the JUWELS compute node. In consequence, these particular problems are not considered in the experiments using the ELLPACK matrix format.

A major goal of the customized precision implementation is to preserve the convergence rate of the reference implementation of PageRank based on IEEE FP64. A few additional iterations may be acceptable, but a significant convergence delay may turn the customized-precision realization unattractive from the performance point of view. In Table 2, we report the convergence details of the PageRank algorithm realized when operating with different configurations: the default implementation using IEEE FP64, a customized precision implementation using a 2-segment splitting, and a customized implementation using a 4-segment splitting. For the customized precision PageRank, we list the number of iterations completed in the distinct accuracy settings and the total iteration count that accumulates from the iterations in the distinct segment configurations. All implementations generate results of the same quality reducing the difference $||p^{\{k\}} - p^{\{k-1\}}||_1$ by at least ten orders of magnitude while starting with $p^{\{0\}} \equiv \frac{1}{n}$.

An initial observation for the 4-segment splitting is that the use of 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

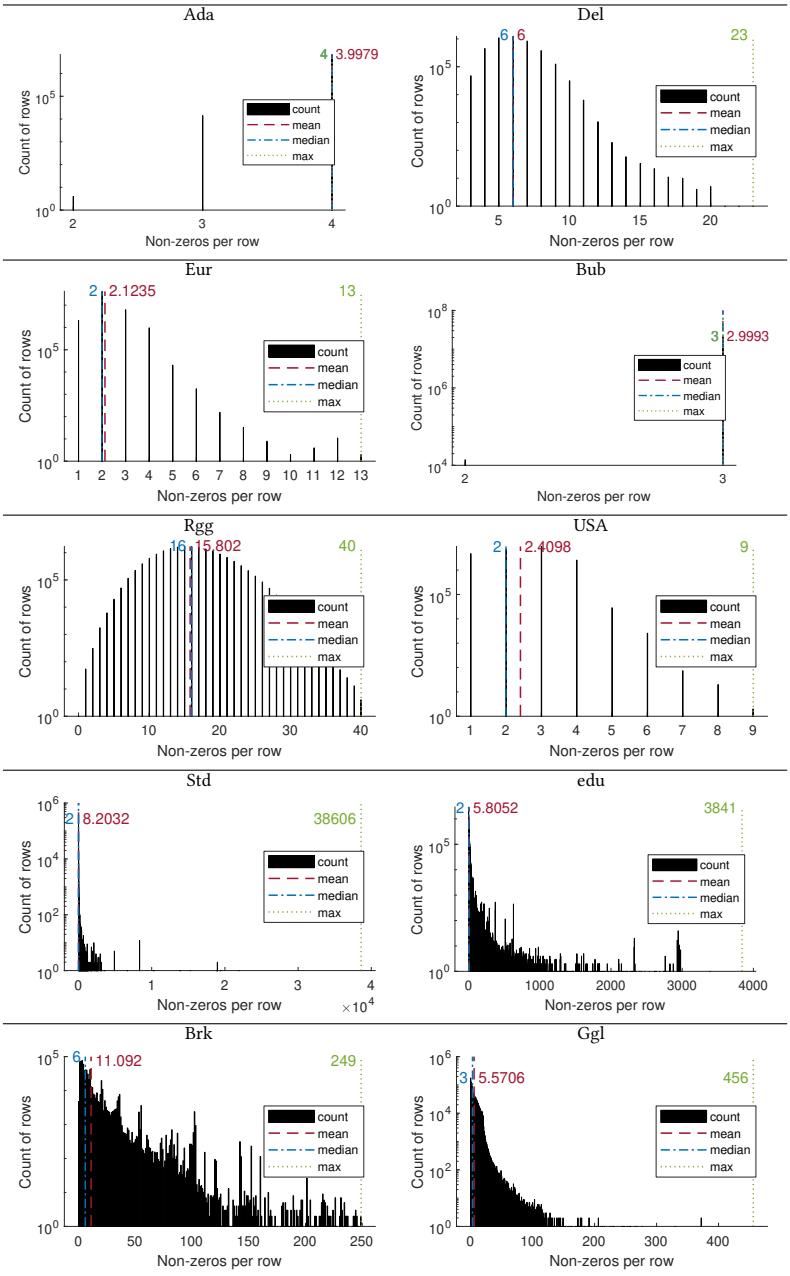


Fig. 3. Nonzero distribution in the test problems.

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).

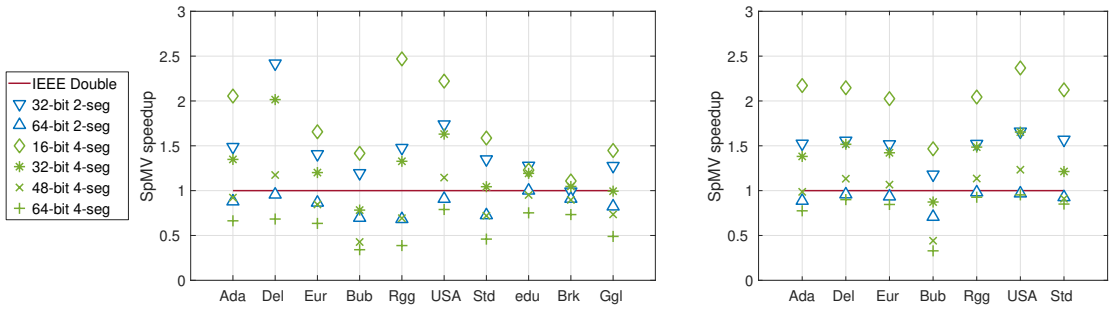


Fig. 4. SpMV speedup, CPMS compared to IEEE double-precision for the CSR format (left) and the ELLPACK format (right).

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 iteration is required. In summary, the convergence rate of the PageRank is generally well-preserved when switching to CPMS-based memory access.

5.3 CPMS-based PageRank on GPUs

We first focus on the CPMS-based implementation of PageRank on GPUs. As previously motivated, we limit the analysis on GPU architectures to the CPMS-i format, interleaving the segments in banks of 128 bytes.

The SpMV embedded in the power iteration is the central and most expensive building block of the PageRank iteration. In Figure 4, we report the speedup of the CPMS SpMV using different segment configurations in the memory access over the reference implementation, which uses IEEE FP64 memory access. We consider both the CSR format (left) and the ELLPACK format (right). In the labels for Figure 4, the first number indicates how many bits of the IEEE FP64 number are retrieved from memory; the second part indicates how many segments the IEEE FP64 numbers are split. Thus, "32-bit 2-seg" means that an IEEE FP64 value was split into 2 segments (with 32 bits each), and only the first segment (32 bits) is accessed. Similarly, "64-bit 2-seg" means that the IEEE FP64 value was split into 2 segments, and both segments (64 bits) are accessed. This is equivalent to IEEE FP64, and can be expected to be slower due to the overhead induced by the operating logic and a higher cache miss rate as in the 2-segment splitting, two distinct memory areas have to be accessed for a single value. The analysis reveals that using 32-bit accuracy in the 2-segment CPMS provides a speedup of about 1.5× on average (the high speedup for Del comes from cache effects). The 4-segment CPMS is about two times faster for 16-bit accuracy. Using 32-bit accuracy, the 4-segment CPMS is about 30% slower 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 as well as the previously mentioned higher cache miss rate. Using more than 32-bit accuracy, the CPMS SpMV is inferior to the IEEE FP64 SpMV in most cases for the CSR format, while it is still slightly faster in most cases for the ELLPACK format.

The next natural question is how these performance advantages of the CPMS SpMV improve the performance of the PageRank algorithm. For this study, we set $\delta = 0.85$ in Algorithm 1, which is a popular choice [Page et al. 1998], and select a relative accuracy stopping criterion $\varepsilon = 10^{-10}$. In Figure 5, we visualize the iteration runtime in the different

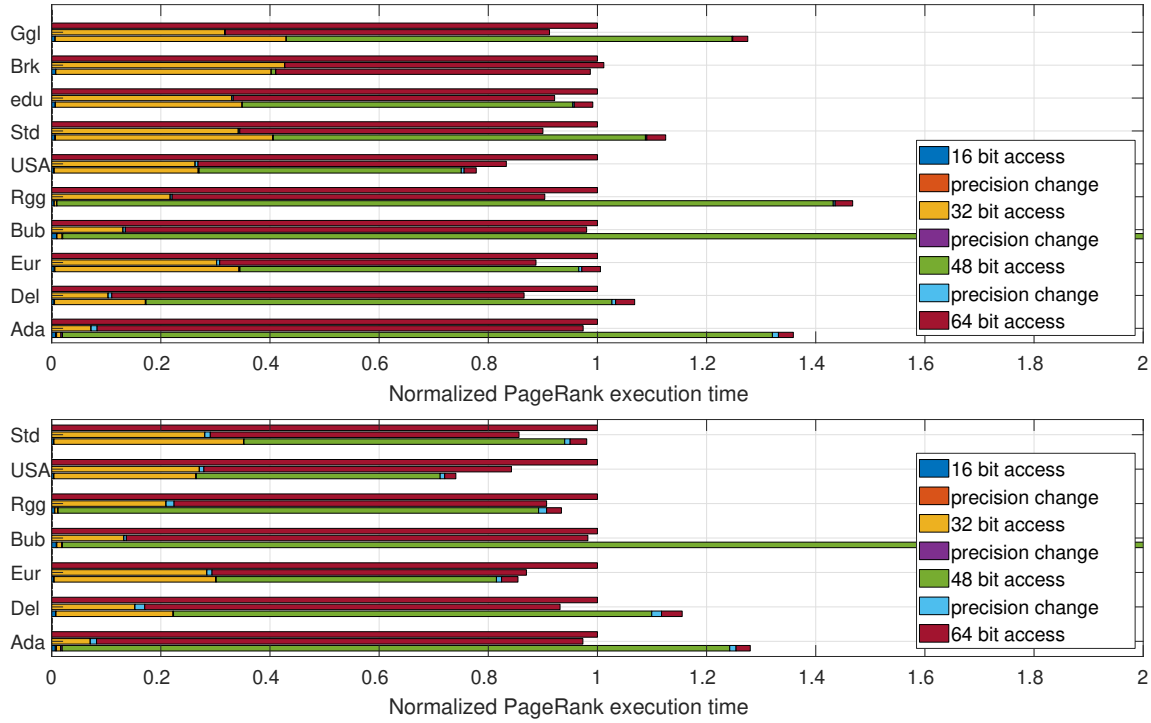


Fig. 5. Normalized PageRank execution time, broken down in time spent at each precision level and the conversion; The "precision change" part contains both, the in-place conversion and the normalization of the iteration vector. The top bar shows the runtime in IEEE double-precision, the middle bar the 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 ELLPACK 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.

format configurations for all target problems. For each case, we normalize the runtime 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 iteration times spent in the distinct accuracy configurations. While the number of iterations in the distinct precision environments correspond to the iteration counts listed in Table 2, the iterations using reduced accuracy memory access are generally faster. The data for CPMS-based PageRank also includes the configuration switching in-between the iteration phases. For some problems (e.g., Std), we notice a difference in the performance trend of the CPMS PageRank using either CSR and ELLPACK (compare top/bottom plot in Figure 5). This is likely related to the interaction of the matrix characteristics in different formats and the architecture cache. The ratio between the execution times spent in different configurations, however, remains the same as it correlated to the iteration counts in Table 2.

Finally, in Figure 6, we show the runtime of the CSR-based PageRank algorithm with respect to the target relative residual accuracy for different memory access strategies – IEEE FP64, 2- and 4-segment CPMS. If a low accuracy solution is acceptable, the PageRank instance using CPMS and mostly accessing values with reduced accuracy is faster than the reference PageRank using FP64 memory accesses.

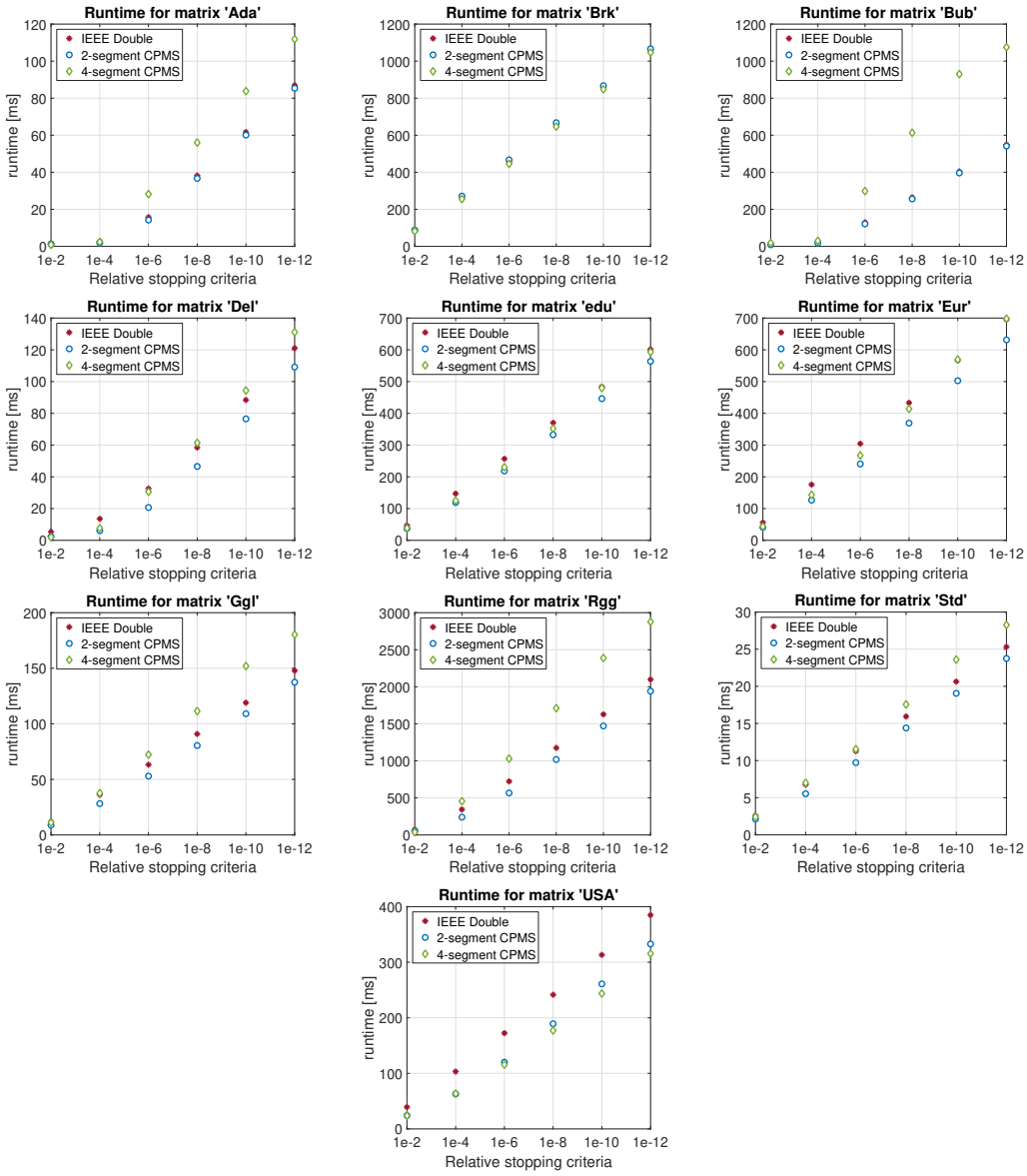
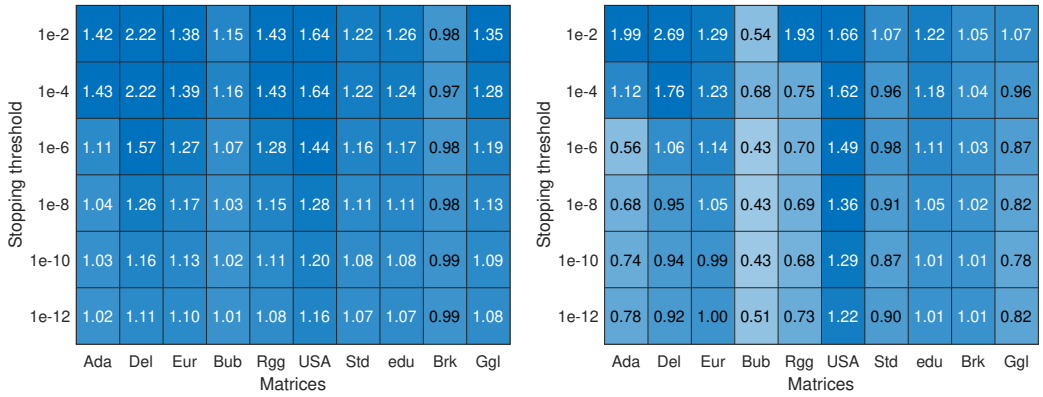


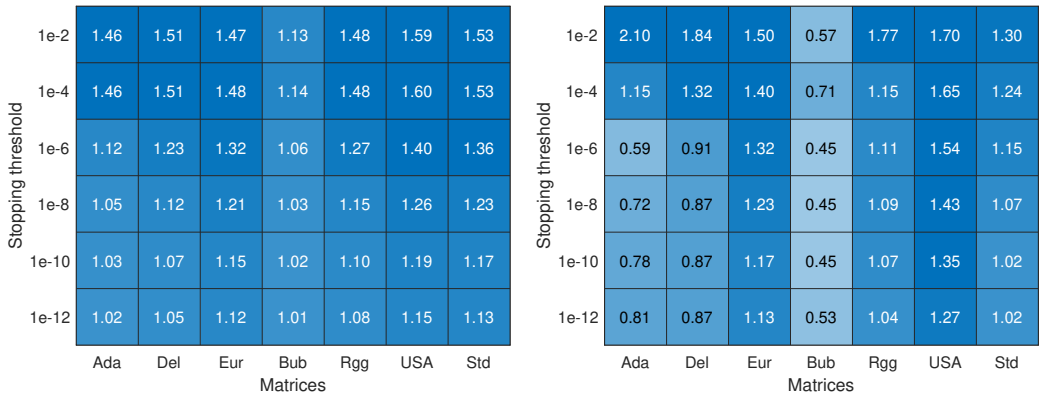
Fig. 6. PageRank runtime related to the stopping criterion. The reported results are for the GPU of PageRank using the CSR format and CPMS or IEEE double-precision memory access.

For easier interpretation, we show in Figure 7 the speedup of the 2-segment realization (left-hand side plot) and the 4-segment realization (right-hand side plot) over the IEEE FP64 implementation. The speedup factors for the 2-segment CPMS (left-hand side in Figure 7) are generally larger than for the 4-segment CPMS (right-hand side in Figure 7). 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



(a) PageRank with CSR and interleaved, 2-segment CPMS (b) PageRank with CSR and interleaved, 4-segment CPMS

Fig. 7. PageRank speedup on the GPU using the CSR format, CPMS compared to IEEE double-precision in a heatmap.



(a) PageRank with ELLPACK and interleaved, 2-segment CPMS (b) PageRank with ELLPACK and interleaved, 4-segment CPMS

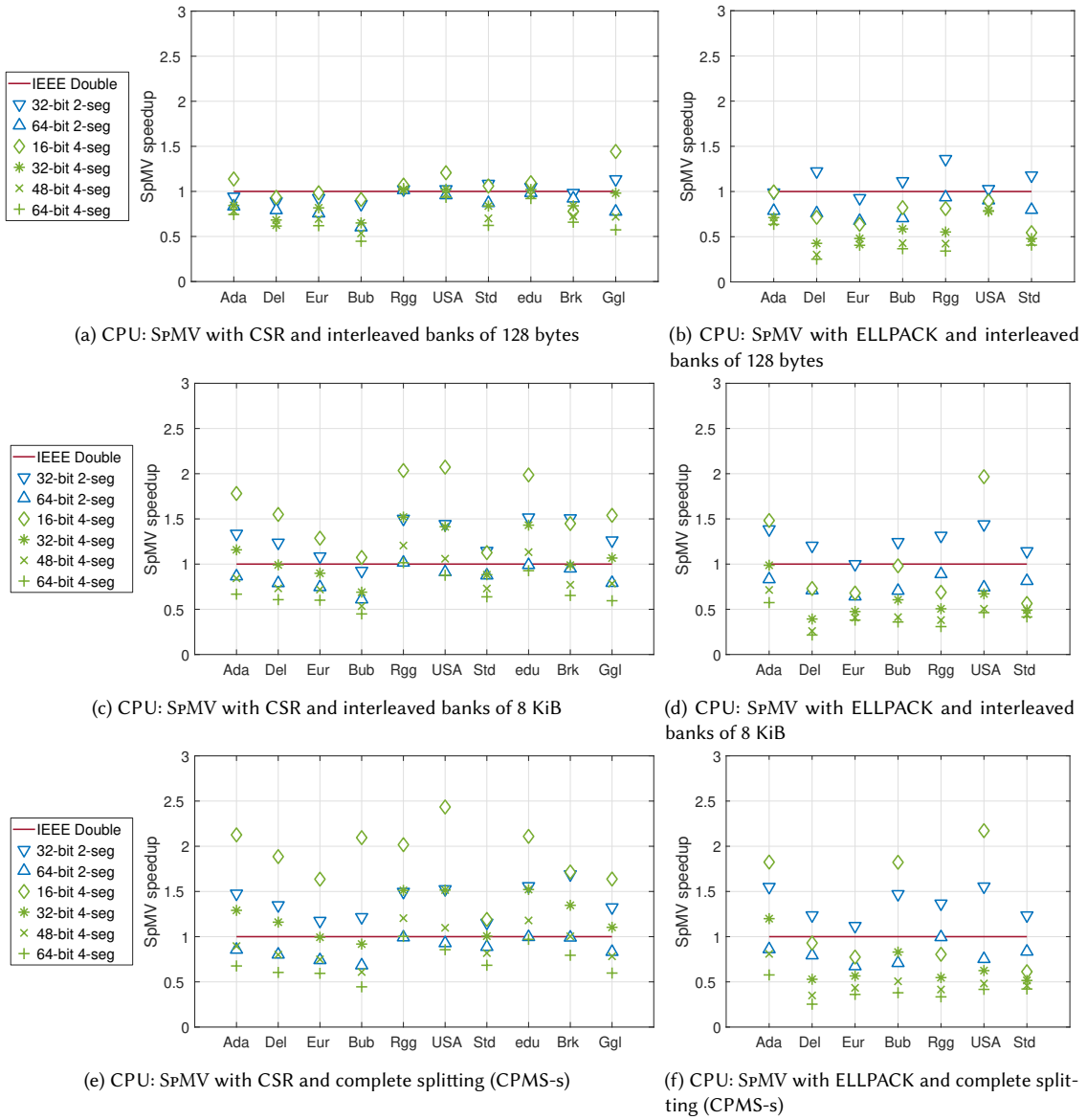
Fig. 8. PageRank speedup on the GPU using the ELLPACK format, CPMS compared to IEEE double-precision in a heatmap.

saturate the memory bandwidth if all threads of a warp read only 16-bit segments. In contrast, the 4-segment CPMS performs better when few iterations are sufficient to fulfill the accuracy requirements.

Figure 8 reports the results for an analogous experimental evaluation using the ELLPACK SpMV kernel, exposing similar runtime benefits when employing the CPMS technique.

5.4 CPMS-based PageRank on CPUs

For the CPU, we consider 12 configurations that arise as combinations of: two different matrix storage formats (CSR and ELLPACK), three interleaving variants (CPMS-i using 128-byte banks and CPMS-i using 8 KiB banks as well as CPMS-s completely separating the segments in memory), and two segmentation strategies (2-segment splitting and 4-segment splitting).



(a) CPU: SpMV with CSR and interleaved banks of 128 bytes

(b) CPU: SpMV with ELLPACK and interleaved banks of 128 bytes

(c) CPU: SpMV with CSR and interleaved banks of 8 KiB

(d) CPU: SpMV with ELLPACK and interleaved banks of 8 KiB

(e) CPU: SpMV with CSR and complete splitting (CPMS-s)

(f) CPU: SpMV with ELLPACK and complete splitting (CPMS-s)

Fig. 9. SpMV speedup on CPU, CPMS compared to IEEE double-precision for the CSR format (left) and the ELLPACK format (right). The results in the top row are using CPMS-i interleaving banks of 128 bytes of segments, in the middle interleaving banks of 8 KiB and the results in the bottom row are using CPMS-s storing the segments in distinct memory blocks, see Figure 2.

First, we again focus on the performance of the CPMS SpMV kernel as this is the key building block. For this kernel, we collect all results in Figure 9. The configuration based on CPMS-i using 128-byte banks performs poorly on the CPU, see the top row in Figure 9. For example, even when reading only half of the data (32-bit access), the throughput (elements/s) is, at most, marginally faster than that of the standard IEEE FP64 version. We believe this is due to the

CPU's prefetching mechanism, which is unable to detect the interleaved pattern of the memory accesses and fails to ignore the un-used tail segments in the low precision data access. By increasing the bank size to 8 KiB, we can alleviate this effect and achieve higher throughput, see Figure 9c and 9d. Compared with the 128-byte banks, the larger interleaving factor enables faster kernel execution for all test problems. In particular, the performance for CPMS-i using 8 KiB banks is competitive with the performance of the CPMS-s strategy in which all segments of the same order are stored consecutively in memory. We can assume that the prefetching behavior for CPMS-s is identical to that of a matrix stored in the IEEE format – while accessing fewer bits in low precision accesses. As elaborated previously, the conversion between CPMS-s and IEEE FP64 cannot be realized in-place, which makes the performance-competitive CPMS-i using 8 KiB banks an attractive solution for CPU architectures with support for prefetching.

For CPMS-i using 8 KiB banks and CPMS-s, the SpMV kernel runs faster on the CPU if reduced precision is acceptable (see middle and bottom row in Figure 9).

Next, we investigate the performance benefits CPMS can render to the PageRank algorithm using the CSR-based SpMV and the ELLPACK-based SpMV. We report the corresponding speedup factors over FP64 memory access in Figures 10 and 11, respectively.

For CSR, CPMS-s consistently outperforms CPMS-i, while the 2-segment CPMS (left-hand side schemes) also outperforms the 4-segment CPMS (right-hand side schemes) in almost all cases, compare top/bottom row in Figure 10. Comparing to the GPU results shown in Figure 7, the CPU implementation of CPMS-i and CPMS-s can unleash even larger speedup factors.

The performance gains for the ELLPACK SpMV are reported in Figure 11. The general trends are similar to those observed for CSR: CPMS-s performs better than CPMS-i, and the 2-segment realization is generally superior to the 4-segment solution. However, the performance gains are smaller than those we reported for the GPU in Figure 8.

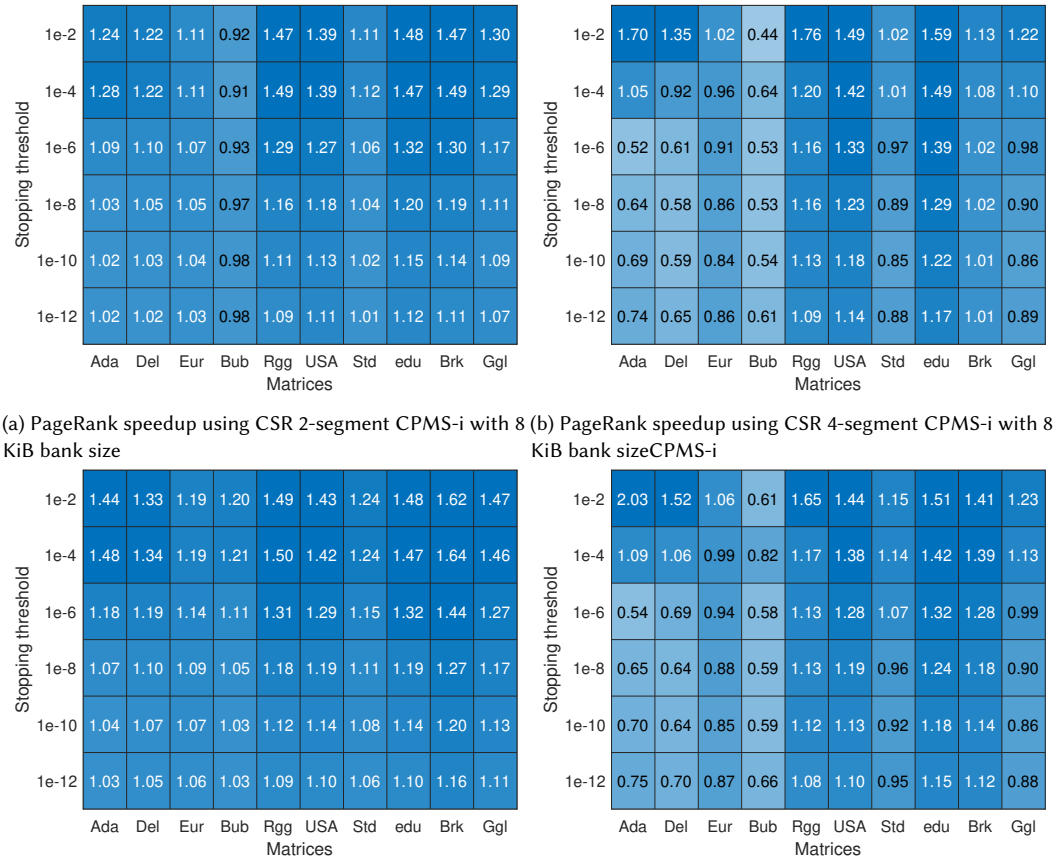
6 SUMMARY AND OUTLOOK

We have demonstrated that the application of a customized precision memory access strategy can unleash attractive performance improvements to the memory-bound PageRank algorithm on multi-core processors and GPU architectures. The keys to these improvements are 1) an adaptive-precision technique that tunes the precision of the data access as the iteration converges; and 2) the selection of a few customized precision formats outside the IEEE 754 standard. The use of customized formats allows modulating the number of the significand bits of the distinct values that are retrieved from memory, while still delivering high memory bandwidth. It also avoids overflow and underflow and efficiently removes the need for data duplication that would occur if employing different IEEE 754 standard precision formats. We used a set of test problems from the SuiteSparse matrix collection to evaluate the performance benefits on a Intel(R) Xeon(R) Platinum 8168 CPU (24 cores per socket) and an NVIDIA V100 GPU. On average, the PageRank variant employing the customized precision memory access technique reduces the time-to-solution by 10% if a highly accurate output is required, and by 30% if lower accuracy is acceptable.

As part of future work, we plan to explore the use of CPMS-based memory access for other memory-bound algorithms.

ACKNOWLEDGMENT

H. Anzt was supported by the "Impuls und Vernetzungsfond" of the Helmholtz Association under grant VH-NG-1241. G. Flegar and E. S. Quintana-Ortí were supported by 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.



(a) PageRank speedup using CSR 2-segment CPMS-i with 8 KiB bank size

(b) PageRank speedup using CSR 4-segment CPMS-i with 8 KiB bank sizeCPMS-i

(c) PageRank speedup using CSR 2-segment CPMS-s

(d) PageRank speedup using CSR 4-segment CPMS-s

Fig. 10. PageRank speedup on the CPU using the CSR format, CPMS compared to IEEE double-precision in a heatmap.

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

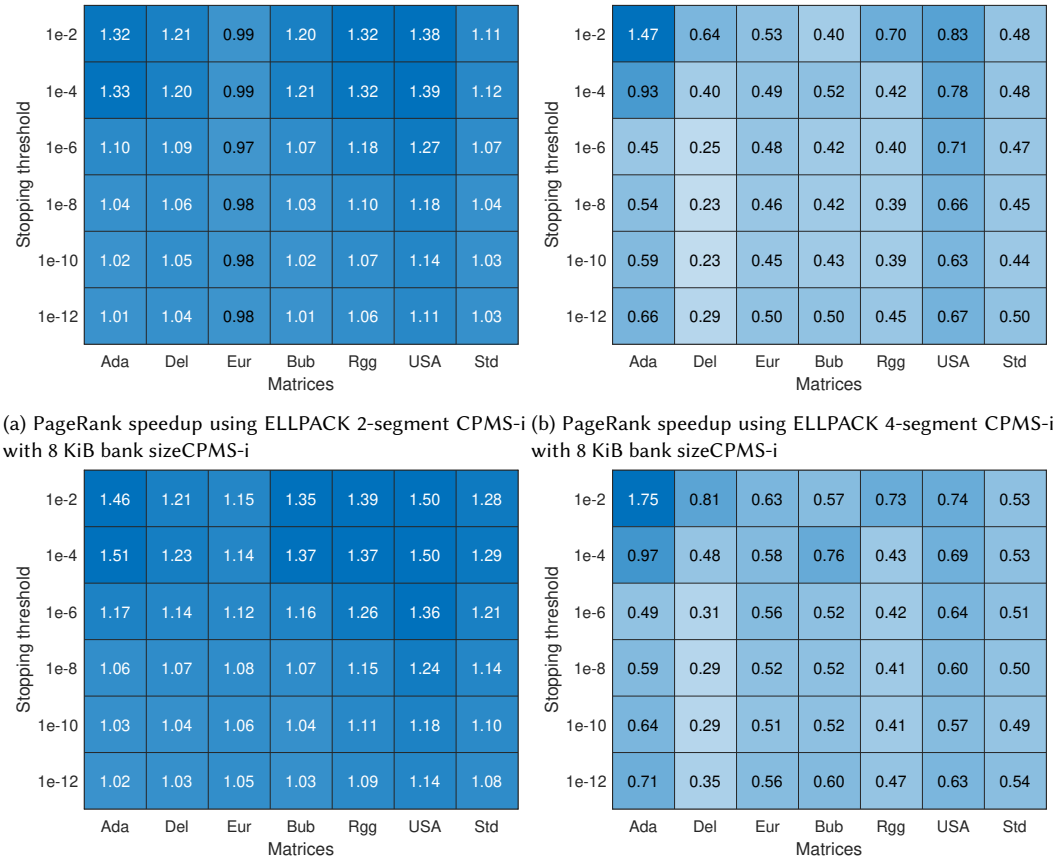
Hartwig Anzt, Jack Dongarra, Goran Flegar, Nicholas J. Higham, and Enrique S. Quintana-Ortí. 2018. Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers. *Concurrency and Computation: Practice and Experience* (2018). https://doi.org/10.1002/cpe.4460

Hartwig Anzt, Jack Dongarra, and Enrique S. Quintana-Ortí. 2015. Adaptive precision solvers for sparse linear systems. In *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing (E2SC '15)*. ACM, New York, NY, USA, Article 2, 10 pages. https://doi.org/10.1145/2834800.2834802

Hartwig Anzt, Goran Flegar, Thomas Grützmacher, and Enrique S Quintana-Ortí. 2019. Toward a modular precision ecosystem for high-performance computing. *The International Journal of High Performance Computing Applications* (May 2019), 109434201984654. https://doi.org/10.1177/1094342019846547

Hartwig Anzt, Mark Gates, Jack Dongarra, Moritz Kreutzer, Gerhard Wellein, and Martin Köhler. 2017. Preconditioned Krylov solvers on GPUs. *Parallel Comput.* 68 (oct 2017), 32–44. https://doi.org/10.1016/J.PARCO.2017.05.006

Nathan Bell and Michael Garland. 2009. Implementing Sparse Matrix-vector Multiplication on Throughput-oriented Processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09)*. ACM, New York, NY, USA, Article 18, 11 pages. https://doi.org/10.1145/1654059.1654078



(a) PageRank speedup using ELLPACK 2-segment CPMS-i with 8 KiB bank sizeCPMS-i

(b) PageRank speedup using ELLPACK 4-segment CPMS-i with 8 KiB bank sizeCPMS-i

(c) PageRank speedup using ELLPACK 2-segment CPMS-s

(d) PageRank speedup using ELLPACK 2-segment CPMS-s

Fig. 11. PageRank speedup on the CPU using the ELLPACK format, CPMS compared to IEEE double-precision in a heatmap.

Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. In *Seventh International World-Wide Web Conference (WWW 1998)*. http://ilpubs.stanford.edu:8090/361/

Erin Carson and Nicholas J. Higham. 2017. A New Analysis of Iterative Refinement and its Application to Accurate Solution of Ill-Conditioned Sparse Linear Systems. *SIAM J. Scientific Computing* 39, 6 (2017), A2834–A2856. https://doi.org/10.1137/17M1122918

Erin Carson and Nicholas J. Higham. 2018. Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions. *SIAM J. Scientific Computing* 40, 2 (2018), A817–A847. https://doi.org/10.1137/17M1140819

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

Gene H. Golub and Charles F. Van Loan. 1996. *Matrix Computations* (3rd ed.). The Johns Hopkins University Press, Baltimore.

Max Grossman, Christopher Thiele, Mauricio Araya-Polo, Florian Frank, Faruk O. Alpak, and Vivek Sarkar. 2016. A survey of sparse matrix-vector multiplication performance on large matrices. *CoRR* abs/1608.00636 (2016). arXiv:1608.00636 http://arxiv.org/abs/1608.00636

Thomas Grützmacher and Hartwig Anzt. 2019. A modular precision format for decoupling arithmetic format and storage format. In *Euro-Par 2018: Parallel Processing Workshops*. Springer International Publishing, 434–443. https://doi.org/10.1007/978-3-030-10549-5_34

Thomas Grützmacher, Terry Cojean, Goran Flegar, Fritz Göbel, and Hartwig Anzt. [n. d.]. A customized precision format based on mantissa segmentation for accelerating sparse linear algebra. *Concurrency and Computation: Practice and Experience* 0, 0 ([n. d.]), e5418. https://doi.org/10.1002/cpe.5418 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5418 e5418 cpe.5418.

Thomas Grüztmacher, Hartwig Anzt, Florian Scheidegger, and Enrique S. Quintana-Ortí. 2018. High-performance GPU implementation of PageRank with reduced precision based on mantissa segmentation. In *Proc. 8th Workshop on Irregular Applications: Architectures and Algorithms (IA3'18)*. 61–68.

https://doi.org/10.1109/IA3.2018.00015

Nicholas J. Higham. 2002. *Accuracy and Stability of Numerical Algorithms* (2 ed.). SIAM.

Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele P. Scarpazza. 2018. *Dissecting the NVIDIA Volta GPU Architecture via microbenchmarking*. Technical Report. arXiv:1804.06826 http://arxiv.org/abs/1804.06826

Moritz Kreutzer, Georg Hager, Gerhard Wellein, Holger Fehske, and Alan R. Bishop. 2014. 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 (2014), C401–C423. https://doi.org/10.1137/130930352 arXiv:http://dx.doi.org/10.1137/130930352

Amy N. Langville and Carl D. Meyer. 2012. *Google's PageRank and beyond: The science of search engine rankings.* Princeton University Press, Princeton, NJ, USA.

NVIDIA Corp. 2017. Whitepaper: NVIDIA Tesla V100 GPU Architecture. (2017).

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1998. The PageRank citation ranking: Bringing order to the web. In *Proceedings of the 7th International World Wide Web Conference*. Brisbane, Australia, 161–172. citeseer.nj.nec.com/page98pagerank.html

SuiteSparse. 2018. SuiteSparse Matrix Collection. https://sparse.tamu.edu. (2018). Accessed in April 2018.