Half-Precision Floating-Point Formats for PageRank: Opportunities and Challenges


Published in:
2020 IEEE High Performance Extreme Computing Conference, HPEC 2020

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Publisher rights
Copyright 2020 IEEE.
This work is made available online in accordance with the publisher's policies. Please refer to any applicable terms of use of the publisher.

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Download date:28. May. 2021
Half-Precision Floating-Point Formats for PageRank: Opportunities and Challenges

Amir Sabbagh Molahosseini
EEECS School
Queen’s University Belfast
Belfast, United Kingdom
a.sabbaghmolahosseini@qub.ac.uk

Hans Vandierendonck
EEECS School
Queen’s University Belfast
Belfast, United Kingdom
h.vandierendonck@qub.ac.uk

Abstract—Mixed-precision computation has been proposed as a means to accelerate iterative algorithms as it can reduce the memory bandwidth and cache effectiveness. This paper aims for further memory traffic reduction via introducing new half-precision (16 bit) data formats customized for PageRank. We develop two formats. A first format builds on the observation that the exponents of about 99% of PageRank values are tightly distributed around the exponent of the inverse of the number of vertices. A second format builds on the observation that 6 exponent bits are sufficient to capture the full dynamic range of PageRank values. Our floating-point formats provide less precision compared to standard IEEE 754 formats, but sufficient dynamic range for PageRank. The experimental results on various size graphs show that the proposed formats can achieve an accuracy of 1e-4, which is an improvement over the state of the art. Due to random memory access patterns in the algorithm, performance improvements over our highly tuned baseline are 1.5% at best.

Keywords—Graph Processing, Customized Floating-Point Data Format, PageRank, Transprecision Computing.

I. INTRODUCTION

Graph processing plays an important role in big data analysis such as applied to social networks [1] and web search [2]. Processing of large graphs poses significant performance challenges. Inherent seemingly-random memory access patterns, exacerbated by skewed-degree distributions, cause poor cache utilization and significant exposure to memory bandwidth limits and memory access delay [3, 4]. For instance, over 60 percent of energy consumed in a communication-bound graph algorithm such as PageRank (PR) is due to memory [5]. For iterative algorithms like PageRank, which are based on floating-point computations, reduced-precision and mixed-precision computation has been shown to reduce memory pressure [6, 7].

For many applications, standard floating-point (FP) formats such as the IEEE-754 FP32 and FP64 formats are wider than necessary. The IEEE FP16 format, implemented by NVidia since CUDA7.5 [8] and more recently by Intel [9], provides a suitable format for workloads such as image processing. However, FP16 sacrifices both dynamic range and precision compared to FP32. Moreover, conversion between FP16 and FP32 may incur under/overflow problems. BFloat16 [10] was introduced for deep learning applications [11] and has the same exponent size as the FP32. This eliminates the under/overflow concerns during conversion. The DLFloat16 format with 6 bits exponent and 9 bits mantissa is customized for deep learning applications [12]. Although these 16-bit data formats have been reported to be efficient for deep learning training, and [11] predicted the industry-wide adoption of BFloat16 format for many applications, narrow floating-point formats are highly sensitive to the application characteristics. For instance, [13] proposed a 16-bit format for PageRank that captures the top two bytes of FP64. This makes conversion to/from FP64 efficient. However, they could execute only the first iteration of the power iteration method using this format. We conclude that narrow floating-point formats must be customized to the application.

This paper defines novel half-precision floating-point formats customized to PageRank. This goal involves two aspects. First, we need to understand the requirements of PageRank on dynamic range and precision. Analysis of PageRank values, as well as the PageRank values divided by vertex degree, demonstrates that the exponents of these values are tightly clustered around a specific value that relates to the size of the graph. Based on this observation, we propose a half-precision format where the exponent is encoded tightly in 3 bits, leaving 13 bits for the mantissa (a sign bit is not required for storing PageRank values). This format poses challenges, as a very small fraction of values cannot be encoded in this format and require a wider representation. We present also a second format as we demonstrate that 6 bits of exponent are sufficient to encode all PageRank values, even in very large graphs.

A second concern in the use of custom floating-point formats is the efficient implementation of arithmetic. Floating-point computation is fairly complex to emulate as bit-wise operations on a general-purpose instruction set [14]. It is more efficient to convert compact storage formats to a format for which arithmetic is supported by hardware, typically FP32 or FP64 [7, 13]. Conversions include changes in the bit width of the exponent and rounding of the mantissa. Including bit shifting and masking, conversion operations can easily add 10–16 instructions on a general-purpose instruction set. Our formats are designed to support efficient conversion in as few as 4 x86-64 and AVX assembly instructions. We discuss the challenges that need to be overcome to achieve this.

The remainder of this paper is organized as follows. Section II briefly reviews the PageRank algorithm and mixed-precision PageRank. Section III introduces novel number formats. Section IV presents an evaluation of convergence and performance. Finally, Section V concludes the paper.
II. BACKGROUND

PageRank models the behavior of a random surfer on the internet as a stochastic process whereby links from one webpage to another are followed at random [15]. This formulation results in a linear equation where the eigenvector corresponding to the largest eigenvalue of a column-stochastic matrix describes the relative importance or relevance of each webpage. This eigenvector is the solution of the PageRank problem. It is most commonly computed using the power iteration method. Here, an initial estimate of the eigenvector is iteratively improved by multiplying the column-stochastic matrix with the previous estimate of the eigenvector. This iteration is repeated until the residual error (the norm-1 difference between subsequent estimates) is less than a pre-defined threshold.

Calculating the PageRank vector is highly sensitive to rounding errors due to finite-precision calculations. Gleich [16] presents an account of the challenges that occur. We use his recommended algorithm as our baseline algorithm, depicted as Algorithm 1. This algorithm takes a graph $G=(V,E)$ as input, where $V$ is a set of vertices and $E$ is a set of edges. The parameters $d$ and $Threshold$ represent the damping factor and convergence threshold, respectively. This algorithm uses two normalization steps to compensate rounding errors. Moreover, sums of values accumulated across all vertices (methods SUM and NORMDIFF) use an enhanced precision compared to the precision of PageRank values: FP32 values are accumulated in an FP64 accumulator; FP64 values are accumulated using Kahan’s algorithm for compensated summation [17]. Due to these error compensation techniques, we find that storing PageRank values in FP32 format is sufficient to reach convergence to an accuracy of $10^{-2}$ on all of the graph data sets that we analyze.

Algorithm 1: PageRank with Normalization [16] - Baseline

Inputs: $G(V,E), d, Threshold$

Output: PR values for all vertices $PR[\cdot]$ 

$PR[\cdot] \leftarrow \frac{1}{|V|}$

while ResidualError > Threshold do 

for each vertex $u \in V$ do 

contribution$[u] \leftarrow \frac{PR[u]}{Outlink(u)}$

for each vertex $u \in V$ do 

for each incoming neighbour $v$ of vertex $u$ do 

$PR_{new}[u] \leftarrow PR_{new}[u] + d \times contribution[v]$

$s = SUM(PR_{new}[V])$

for each vertex $u \in V$ do 

$PR_{new}[u] \leftarrow PR_{new}[u] + \frac{1-s}{|V|}$

ResidualError = NORMDIFF($PR, PR_{new}, |V|$)

$norm = SUM(PR_{new}, |V|)$

for each vertex $u \in V$ do 

$PR[u] \leftarrow \frac{PR_{new}[u]}{norm}$

$PR_{new}[u] \leftarrow 0$

Mixed-precision iterative algorithms have been proposed for linear solvers [6, 18], where initial iterations calculate a first estimated solution at reduced precision, then iteratively improve the estimate using increasingly higher precision. Mixed-precision arithmetic could potentially incrementally increase the precision by adding 1 or 2 bits to the mantissa at a time [19].

Mixed-precision arithmetic was previously investigated for PageRank [13]. Efficient arithmetic is ensured through efficient conversion to hardware-supported arithmetic, in this case FP64. Three reduced-precision formats are proposed containing the top 16, 32 or 48 bits of the FP64 number: the sign bit, 11 exponent bits and a slice of the most significant mantissa bits to fill up the format. It is, however, worthwhile to use this 16-bit format only during the first power iteration [13]. After this, the format switches to 32 bits. We will show that 6 exponent bits are sufficient to hold PageRank values for very large graphs. As such, the 32-bit format with 11 exponent bits is fundamentally inferior to FP32 with 8 exponent bits for the PageRank problem. As we find that convergence is possible with FP32 using Gleich’s algorithm, there is no need to consider the 48-bit of FP64 formats for PageRank.

Apart from a few works which have implemented PageRank with applications-specific integrated circuits (ASIC) [20] and field-programmable-gate arrays (FPGA) [21], the existing off-the-shelf hardware’s are the main and widely used platform for implementation of PageRank. However, they support only IEEE standard floating-point formats, typically single precision (32-bit) and double precision (64-bit). There are two main methods for implementing arithmetic for custom floating-point formats using existing modern processors. The first one is emulation of arithmetic on the IEEE supported hardware [22]. Although the emulation can be done via integer arithmetic, the need for multiple instructions makes the emulation slow. The second method is to convert the custom format to IEEE standard formats, and then performing computations [14]. The disadvantage of this method is incurred conversion overheads; while the advantage is the use of optimized and highly efficient floating-point units of processors. Conversion to native formats to perform arithmetic must be efficient in order not to offset the performance gains. Truncated formats as in [13] are easily converted, but do not provide convergence. Moreover, conversion requires a mix of 16- and 32-bit integer operations, after which the result is moved to a floating-point data type without conversion. This is tricky to specify even in low-levels like C and C++ [13, 14]. Type-punning using a union generates load and store instructions to the stack. A reinterpret_cast has the same effect. These additional instructions erode the performance gain.

III. THE PROPOSED HALF-PRECISION FORMATS

The target of this work is to design efficient compact number formats customized according to the specific features of the PageRank algorithm. In order to do this, two important things should be considered to achieve efficient data formats: 1) considering the mantissa size as much as possible in order to provide high-precision for PageRank computations in the starting iterations, and 2) having the same dynamic range as single-precision format, in order to prevent under/overflow problems. In order to achieve such data formats, in contrast to previous work that performed mantissa segmentation [13], our
focus is on using specific features of PageRank in order to reduce the exponent size as much as possible and to grow the space for the mantissa.

A. Analysis of PR values Exponents

A particular difficulty with PageRank is that a low fraction of vertices has a high PageRank value. Due to how vertices are labelled, these values are randomly distributed through the graph and so they will randomly appear in a block of vertices with otherwise similar PageRank values. Although, this removes opportunities for compression or using some shared exponent mechanisms [23], the similarity between exponents of PageRank values and their distribution can show us interesting features and opportunities to optimize data formats. Hence, this subsection describes the theoretical and experimental analysis of exponents of PR values. First, we introduce some basic theorems about PR values.

Theorem 1: The range of PR values is between $\frac{1-d}{|V|}$ and 1.

Proof: The main PR formula is [15, 16]:

$$PR^{t+1}(p_i) = \frac{1-d}{|V|} + d \sum_{j \in \text{links to } p_i} \frac{PR(p_j)}{\text{outlinks}(p_j)}$$

Therefore, according to (1), it is clear that the minimum and maximum possible PR values are $\frac{1-d}{|V|}$ and 1, respectively.

Theorem 2: The number of exponent bits to record PR values is proportional to $[\log_2 \log_2 |V|]$.

Proof: According to Theorem 1, all PR values are less than 1, and the minimum exponent is proportional to $|V|$. On the other hand, the exponent of a floating-point represented real number $X$ is $\log_2 X$, and consequently the minimum exponent of bits to represent it is $[\log_2 \log_2 X]$. Substituting $X$ with the $|V|$ results in $[\log_2 \log_2 |V|]$.

Corollary 1: According to Theorem 2, 6 exponent bits suffice for graphs with up to $2^{26} \equiv 18$ trillion vertices. However, $2^{38} \equiv 4\ billion$, which is not sufficient for very large graphs.

The theoretical limitation of minimum PR value motivated us to also experimentally analyse the PR values, and carefully investigate the pattern of exponents. Due to this, we applied PR in Algorithm 1 to graph datasets [24-29] listed in Table I, and we computed the final PR values after convergence to the residual error of $1e-7$, and analysed the frequency of exponents relative to the exponent of $\frac{1}{|V|}$ as shown in Fig. 1.

Table I. Graph datasets used in this work [24-29]

<table>
<thead>
<tr>
<th>Name</th>
<th>Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>twtr10</td>
<td>21 297 772</td>
<td>265 025 810</td>
</tr>
<tr>
<td>41-twtr</td>
<td>41 652 230</td>
<td>1 468 363 182</td>
</tr>
<tr>
<td>friendster</td>
<td>65 608 366</td>
<td>1 806 067 135</td>
</tr>
<tr>
<td>webcc</td>
<td>89 128 896</td>
<td>2 038 263 947</td>
</tr>
<tr>
<td>dimacs-ud</td>
<td>105 153 952</td>
<td>6 603 753 128</td>
</tr>
<tr>
<td>UK07</td>
<td>109 499 788</td>
<td>3 944 932 432</td>
</tr>
<tr>
<td>UU</td>
<td>115 657 285</td>
<td>1 019 903 159</td>
</tr>
<tr>
<td>clueweb</td>
<td>131 814 559</td>
<td>5 077 679 720</td>
</tr>
<tr>
<td>twtr10</td>
<td>1 684 868 322</td>
<td>7 939 635 651</td>
</tr>
</tbody>
</table>

Fig. 1 shows that exponents of PR values are distributed tightly around $\frac{1}{|V|}$: over 99% of PR values have exponents in a limited range $[E-3,E+4]$ where $E = \text{Exponent}(\frac{1}{|V|})$, and the remaining (less than 1%) vertices have other exponent values. This point is reasonable since in reality many pages have very small PR values, and only a few well-known popular pages have very high values.

The tight distribution of PR exponents can provide a new insight for exponent size reduction. In this case, storing an exponent as an offset against $\text{Exponent}(\frac{1}{|V|})$ is very compact. However, in this regard two important challenges exist. The first one is the larger distribution of the PR values exponents for about 1% of vertices, and the second one is that actually the $\frac{PR[u]}{\text{outlink}(u)}$ values are more important, and their exponent distribution is less tight. This can be theoretically confirmed as follows: The minimum PR value, i.e. $\frac{1-d}{|V|}$ is dependent on two parameters, namely, the damping factor and the total number of vertices. Therefore, the possible range of exponents of all PR values is $[\text{Exponent}(\frac{1-d}{|V|}), 0]$ (since PR values are between 0 and 1, we have not any PR values with exponents larger than zero).

Tables II and III present the exponent of the minimum possible PR value (i.e. $\frac{1-d}{|V|}$) and $\frac{PR[u]}{\text{outlink}(u)}$ value (i.e. $\frac{1-d}{|V|+|U|}$), respectively, together with the required number of bits for unsigned representation of them based on different graph sizes.

Table II. Analysis of exponent of minimum PR value for D=0.85

| Graph Size (|V|) | Minimum Exponent | Bits |
|----------|-----------------|------|
| From     | To              | From | To |
| 2        | 4915            | -4   | -15 | 4  |
| 4916     | 322 122 547     | -16  | -31 | 5  |
| 322 122 548 | 150 000 000 000 | -32  | -40 | 6  |

Table III. Analysis of exponent of minimum $\frac{PR[u]}{\text{outlink}(u)}$ for D=0.85

| Graph Size (|V|) | Minimum Exponent | Bits |
|----------|-----------------|------|
| From     | To              | From | To |
| 2        | 17948           | -16  | -31 | 5  |
| 4 453 116 872 | 150 000 000 000 | -32  | -63 | 6  |
| 4 453 116 873 | 150 000 000 000 | -64  | -67 | 7  |
Corollary 1 and Table II indicate that for available big graphs, just 6 bits are enough for exponents. Therefore, 11 bits exponents in double-precision or even 8 bits in single-precision are redundant for PageRank.

B. Half-Precision Data Formats

The previous subsection exposes the clues needed for designing very compact number formats for PageRank. Here, we introduce two customized half-precision floating-point formats, namely, Accuracy Preserving Half-Precision (APHP) and Time Optimizing Half-Precision (TOHP) formats as shown in Fig. 2. The first one, i.e. APHP, can provide higher precision but has a complex conversion to single-precision; while the second one, i.e. TOHP, can provide simple conversion but with less precision.

Details and motivations on designing TOHP are as follow. First, the sign bit is not required, since all PR values are non-negative numbers. Any arithmetic that may result in negative numbers is performed using native formats such as FP32 and FP64. Second, Table II indicates that the minimum required exponent for most of available big Graph datasets is -63. Therefore, with taking into account that exponents in IEEE standards are storing in the biased notation (the real exponent is incremented by 127 as bias in single-precision standard), the higher two most significant bits (MSBs) for all PR exponents are constant through all PageRank iterations as 01 in IEEE single-precision format. Therefore, the two MSBs of the exponent in single precision can be removed, and the remaining 6 bits are used in the TOHP format for storing the exponent. Therefore, the mantissa will have 10 bits in this format.

On the other hand, Fig. 1 indicates that over 99% of the PageRank values are in a limited range of eight numbers, i.e. \([E - 3, E + 4]\) where \(E = \text{Exponent}(\frac{1}{16})\). This feature has motivated us to further reduce the exponent size, and design the APHP format with just 3-bit exponent as shown in Fig. 2(b). Table IV presents details of decoding/encoding of the 3-bit exponent in APHP to the standard single-precision 8-bit exponent. It can be seen that the flags 0 to 7 are used as the representatives of the most eight frequent exponents, and the case of both exponent and mantissa being zero (zero in full 16-bit) is considered as an exception flag for other possible exponents. In this case, the full value will be stored in a separate array. There are multiple options to store these exception values. A scheme that allows efficient access, but has high space utilization, is to allocate a second array to store 32-bit exception values in single precision format. The array contains one element for each vertex. This retains spatial locality on the array of 16-bit PR values. However, since the number of exceptions are usually less than 1% of the total number of vertices, few of the exception values are actually accessed.

Other schemes, based on for instance hash tables, would reduce storage space but would incur a much higher overhead to retrieve or store the values.

The dynamic range of both of the proposed half-precision floating-point formats is the same as standard single-precision format, since we only change the coding, and the actual exponent value in the computation will be the same as the original without risk of overflow/underflow.

C. Number Format Conversion

The 16-bit floating-point formats are converted to the IEEE standard single-precision format to benefit from the hardware arithmetic units. The single-precision format is sufficient as both its exponent and mantissa fields are wider than the TOHP and APHP formats. Conversion requires two steps: 1) handling special cases (underflow, overflow and NaN), 2) rounding. We resolved each of these difficulties in the proposed data formats as follows: First, our analysis ensures that the 16-bit formats can capture the full dynamic range of the values that need to be stored. This prevents under/overflow problems during conversion. Furthermore, NaN cannot occur in the algorithm. Second, we use round-towards-zero with just truncation of least significant bits (LSBs) of single-precision mantissa in order to reduce conversion-overhead as much as possible. Our experiments demonstrate that the convergence is not specifically affected by this.

The conversion operations are presented in Figs. 3 and 4. Conversion from single-precision to the TOHP requires minimal steps (removing/adding in bits, and shifting). APHP additionally requires to access a separate exception array in order to encode/decode exponents for less than 1% of vertices.

![Diagram](image-url)
Fig. 4. The conversion from IEEE standard single-precision floating-point to the proposed APHP format, and vice-versa (note that exceptions storage access is needed for just less than 1% of vertices, and for 99% of vertices encoding and decoding can be performed one subtraction and addition, respectively).

Fig. 5. The residual error of PageRank with different formats for the 41-twtr dataset (note that Y-axis is in log scale)

Fig. 6. The convergence rate of PageRank with different formats based on 41-twtr dataset

IV. Evaluation

This section evaluates the performance and accuracy of the proposed TOHP and APHP formats. We used a 2-socket machine with 768 GB main memory and each socket has an Intel® Xeon® Gold 6130 with 16 cores, 32KB L1 cache per CPU, 1MB L2 cache per CPU, and 22MB L3 cache shared between all CPUs of the same NUMA node. The machine is configured without hyper threading. The experiments have been performed with at least 500 GB of free memory available to OS. GCC 9.2 and the datasets shown in Table I are used for the evaluation.

A. Accuracy Evaluation

We have introduced two methods: APHP is designed with aim to provide lower memory usage bandwidth while it can conserve the accuracy of double- and single-precision floating-point formats. TOHP however assigns execution time as priority at the expense of fewer mantissa bits. In Fig. 5, we see the convergence of APHP and TOHP for our biggest Twitter dataset, and in Fig. 6, we see the convergence rate of these methods. We have selected 1E-3 as the criteria for residual error.

These figures show APHP can track the accuracy of single and double precision formats. However, TOHP allocates lower number of bits to mantissa and this results in losing accuracy. We have validated that the APHP results are accurate by feeding them into the PageRank algorithm using single-precision floats. This algorithm stopped after one iteration considering 1e-3 as the residual error threshold. This demonstrates that the APHP PR values cannot be distinguished from the single-precision float PR values.

Algorithm 2: Mixed-Precision PageRank

Inputs: $G(V, E), d, \text{Threshold}$

Output: PR values for all vertices $PR[: ]$

$PR[: ] \leftarrow 1 / |V|$

while $\text{ResidualError} > \text{Threshold}$ do

for each vertex $u \in V$ do

$\text{contribution}_{16\_bit}[u] \leftarrow \text{encode}(\frac{\text{decode}(PR_{16\_bit}[u])}{\text{outlink}(u)})$

for each vertex $v \in V$ do

for each incoming neighbour $v$ of vertex $u$ do

$PR_{new}[u] \leftarrow PR_{new}[u] + \text{decode}(\text{contribution}_{16\_bit}[u])$

$s = \text{SUM}(PR_{new}, |V|)$

for each vertex $u \in V$ do

$PR_{new}[u] \leftarrow PR_{new}[u] + \frac{1-s}{|V|}$

$\text{ResidualError} = \text{NORMDIFF}(PR_{16\_bit}, PR_{new}, |V|)$

$\text{norm} = \text{SUM}(PR_{new}, |V|)$

for each vertex $u \in V$ do

$PR_{16\_bit}[u] \leftarrow \text{encode}(\frac{PR_{new}[u]}{\text{norm}})$

$PR_{new}[u] \leftarrow 0$
B. Performance Evaluation

In Fig. 7 we show the normalized speedup of single precision float and TOHP to double precision float. Single-precision float achieves a notable speedup over double, between 5% and 30%. TOHP can improve performance in some data sets (e.g., webB), but we observe a slowdown compared to FP32 in many data sets. To understand why this happens, we have also estimated the maximum gain that can be obtained with 16-bit floating-point formats. These numbers were obtained by loading/storing the 16-bit numbers, but not performing a proper conversion. By consequence, the arithmetic is not meaningful, but the measured time is an upper bound to performance. In the following parts of this section we investigate this issue, first by exploring the maximum achievable gain and then explaining why it has not been possible to reach the maximum gain for all datasets.

1) Why is maximum possible gain small?

A reduction in precision results in a speedup because the data stored for each vertex is reduced, e.g., from 8 to 4 bytes when we change from double-precision to single-precision floating-point. As such, twice as many PR values fit in a cache block and we expect that this will translate in a reduction of main memory accesses. The consequence then is to observe a speedup. A similar situation is expected when compacting the vertex values further to 16 bits, again doubling the number of values stored in a cache line. The results, however, show that there is no strong correspondence between the speedups observed between the double-float and float-16-bit changes. In this instance, we consider the maximum gains achievable. We conclude that the gain depends on the locality of vertices in the graph, and improvements in locality may vary depending on the bit width of the PR values. Moreover, we note that the performance of graph analytics is limited by the random access patterns that are dictated by the structure of the graph. These random access patterns result in poor cache block utilization. When a reduction in bit width of PR values does not enhance locality, the random access patterns will remain exposed to the full main memory latency and the number of main memory accesses is not reduced.

2) Why do not we reach the maximum gain?

Additional instructions increase the execution time. We have crafted the data formats and the conversion code to minimize overhead. However, even then, TOHP incurs a significant overhead. The performance difference between TOHP and the maximum gain is singularly due to overhead incurred in the format conversion.

V. CONCLUSION

This paper showed that irregular applications like PageRank can work with half-precision data to achieve an acceptable accuracy especially for real-time applications where some inaccuracy can be tolerated. In order to put more precision in the 16-bit frame, we performed a comprehensive analysis of exponents of PageRank values, and used the results to decrease the number of exponent’s bits without changing the dynamic range than IEEE standard single-precision. Still lots of challenges exist to fully exploit the potential of half-precision in Graph applications that worth for future research including conversion overhead handling, and finding effective ways to benefit from reduced-size feature of half-precision with random memory access parts of Graph processing applications in order to achieve significant runtime reduction.

ACKNOWLEDGMENT

This work is supported by the European Community’s Horizon2020 Programme (H2020/2004-2020) under the OPRECOMP project, grant agreement no. 732631, and by the United Kingdom EPSRC under grant agreement EP/T022345/1.
REFERENCES


