Edge Importance Identification for Energy Efficient Graph Processing

S M Faisal*, G. Tziantzioulis†, A. M. Gok†, N. Hardavellas†, S. Ogrenci-Memik† and S. Parthasarathy*
*Dept. of CSE, The Ohio State University, Columbus, OH, USA
Email: {faisal, srini}@cse.ohio-state.edu
†Dept of EECS, Northwestern University, Evanston, IL, USA
Email: {getziadz, amg}@u.northwestern.edu,{nikos, seda}@northwestern.edu

***Abstract*—Modern graphs are large, often containing *billions of nodes and edges* that demand huge amount of processing for analysis purposes. The algorithms processing these graphs often run for long time and consume substantial amount of energy. However, not all edges in the graphs are equally important. Some edges play critical role in maintaining the community and other interesting structures in the graph, while the rest are less important for analysis. Identifying edges as important and unimportant allows one to apply *elastic fidelity* computing when processing edges of low importance, hence saving significant amount of energy while processing large graphs. In this paper we propose a novel technique for identifying important edges in a graph using a fast method that exploits *locality sensitive hashing*. We then propose a framework for energy-efficient computing that applies elastic fidelity computing when processing edges of low importance and applies full fidelity computing when processing important edges. This allows the framework to deliver good results while saving energy when processing a large number of low-importance edges. Our proposed technique reduces the power consumption by 3–30% while still producing results that are within acceptable range of the full-accuracy results.**

***Keywords*-Graph; Energy Efficient Computing; Algorithm;**

I. Introduction

Power is now a first-class constrain on the design process for large scale computing solutions as well as mobile systems [1]. This is also due to the increasing power density in modern computers and the *Dark Silicon* [2] problem—limitations imposed by power on device scaling. Traditionally computers guarantee the correctness of computation by maintaining a conservative, large voltage guard band as well as timing slack [3] that result in high energy consumption.

Various techniques have been proposed by the research community to improve energy efficiency [4], [5], [6] that withstand occasional circuit errors while providing correct execution to the end user. More aggressive approaches propose *approximate computing* that rely on voltage scaling, i.e., reducing the operational voltage; leading to a narrow voltage guard band and random bit errors in the computation. These approaches champion the idea of allowing (not correcting) bit errors in computations and letting the applications take care of them as necessary. The idea is to trade off accuracy for energy savings [7], [8], [9].

With increasing popularity of social networks, large graphs with *millions of nodes and edges* have become commonplace. Various techniques have been proposed for processing large graphs including ranking [10] and clustering [11]. Most of the graph processing algorithms are complex and consume significant amount of power when applied on large graphs. However, not all edges in a graph are equally important for preserving the structural properties of the graph [12]. Hence, the effect of voltage-scaled error-prone computing on structurally important edges is more severe than that on the remaining edges of the graph. This enables the variation of computational fidelity across edges according to their importance. We apply this technique by first categorizing the edges of the graph into two categories, *important* and *unimportant*, using a fast, efficient algorithm. Once the edges are categorized, we apply elastic fidelity computing to process the unimportant edges and full fidelity computing on edges that are important. The number of edges categorized as important is controlled by a parameter set by the user, enabling customization based on knowledge of the graphs. Our proposed techniques can be easily extended to specialized graph processing frameworks like GraphLab [13] and Global Graphs [14]. To the best of our knowledge, we are the first to propose categorizing graph edges into *important* and *unimportant* for the purpose of applying energy-efficient computing when processing large graphs.

Natural candidate techniques for selecting important edges are various edge sampling techniques found in current literature. However, sampling or sparsification lose structural information about the graph, while marking edges as important (unimportant) allows us to preserve the structural information intact while applying smart techniques to save energy when processing less important edges. Our key contributions are:

- We propose a fast and effective framework for identifying important edges in a graph.
- We analyze *local* and *global* approaches for identifying edge importance.
- We apply elastic fidelity computing in graph processing on the basis of edge importance to save energy while preserving output quality.
- We experimentally evaluate our techniques with various graph processing algorithms.

II. Related Work

Considerable work has been done in the area of detecting and correcting timing errors that stem from under-volting or over-clocking of functional units. Razor [4], [5] is a register transfer level (RTL) technique that provides a mechanism for detecting and correcting timing errors while operating in sub-nominal voltage. CAD techniques have been proposed in recent years that provide graceful degradation of modules under voltage scaling or over-clocking along with correction for the occasional timing errors [15], [7].

Recent research has explored the idea of approximate computing that allows errors to occur in certain regions in a program via aggressive voltage scaling. A programming framework and architecture were proposed recently [8], [9] for marking individual instructions as imprecise and executing them in voltage-scaled functional units. Work in this space has relied on basic fault injection error models that perform single bit flips at the output with uniform distribution [9], or select random or previously seen values for the entire output [9]. Subsequent works [8] rely on uniform bit-error models with per-component probabilities. Most of these works have explored various scientific and multimedia applications.

A large body of research exists in the area of graph sparsification. Approaches proposed based on sampling [16]. Problems of *edge filtering* [17] and *complex network backbone detection* [18] have been studied where the focus is to construct a spanning tree-like network backbone rather than to preserve community structure in a graph. Work on *Graph Sampling* samples nodes along with edges of the graph. Maiya et al. [19] proposed to find representative subgraphs preserving community structure based on algorithms for building expander-like subgraphs of the original graph.

None of these techniques, however, considers the importance of edges for maintaining community structure in the graph. To the best of our knowledge, we are the first to explore the idea of identifying important edges and subsequently applying low-energy computation when processing unimportant edges to save energy.

III. Methods

Graphs in contemporary applications have become massive in size, and usually contain billions of edges. However, their edges have varying importance with respect to maintaining crucial and interesting properties of the graph. For example, when it comes to capturing community structures in the graph, not all edges have an equally important role in maintaining the community structures. We propose techniques that efficiently mark edges as important (unimportant) based on vertex similarity. Satuluri & Parthasarathy [12] proposed a graph sparsification method that takes vertex similarity into account and sparsifies graphs for scalable clustering. In a similar fashion, we exploit similarity between vertex neighborhoods to identify edges that are important for preserving the community structure in the graph.

Key intuition behind the method is: if two vertices of an edge share a lot of common neighbors, it is very likely that the edge belongs to a community structure in the graph. On the other hand, if the two endpoints of an edge have no common neighbors, it is more likely that the edge in question is a bridge between two communities, hence playing a less important role for maintaining communities in the graph. Based on this intuition, we propose our method which computes vertex similarities between the endpoints of each edge in order to assign an *importance score* to the edge. After this step, we can mark low scoring edges as less important for maintaining the graph structure.

A. Requirements and Intuition

We propose to use a similarity-based categorization of edges. The reason behind using a similarity-based method is to ensure the following properties:

- Edges that are important for preserving the clusters in the graph should be marked as *important*.
- Applying the proposed computing techniques should produce results close to ground truth (obtained by executing the respective algorithm on the original graph).
- The identification process itself should be extremely fast in order to be practically applicable to massively large networks and graphs as a preprocessing step.

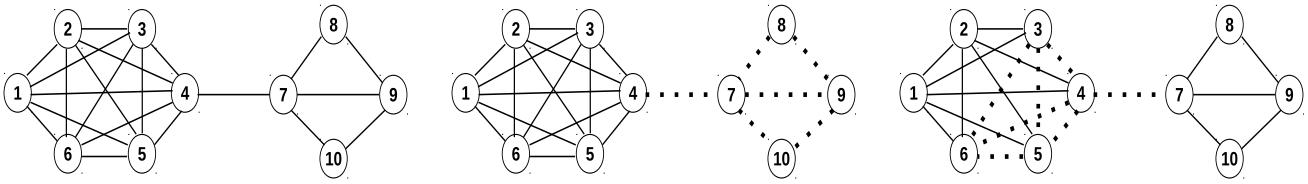
As our goal for identifying important edges is based on the idea of preserving cluster structure in the graph, we prefer to retain intra-cluster edges to inter-cluster edges.

Various edge centrality measures have been used previously to identify edges in sparse parts of the graph. For example, the edge betweenness centrality [20] of an edge. By definition, inter-cluster edges happen to have a high betweenness centrality compared to intra-cluster edges. This seems to be a logical way of identifying important edges in the graph. However, edge betweenness centrality is prohibitively expensive, requiring $O(mn)$ time [20] where m is the number of edges and n is the number of nodes in the graph. We use a simple heuristic for identifying the important edges in the graph.

B. Similarity based identification heuristic

An edge (i, j) is likely to be intra- (inter-) cluster if vertices i and j have neighborhoods with high (low) overlap. To measure the overlap between neighborhoods of nodes, we use *Jaccard* similarity. Let $Adj(i)$ and $Adj(j)$ be the adjacency list of nodes i and j, respectively. Then the similarity between $Adj(i)$ and $(Adj(j)$, referred to as similarity between i and j henceforth, is:

$$Sim(i,j) = \frac{|Adj(i) \cap Adj(j)|}{|Adj(i) \cup Adj(j)|} \tag{1}$$



(a) Original Graph

(b) Important edges (solid) identified using Global Method

(c) Important edges (solid) identified using Local Method

Figure 1. Identification of Important edges using Global and Local approaches

C. Global vs. Local Identification

Once the edges of the graph have been ranked based on their similarity scores, there are two ways *importance* can be determined for each edge. A simple method is to mark the top ranked edges as important. The number of edges marked as *important* depends on a user defined parameter. We call this method *Global Identification* because the approach does not take into account the degree

This approach performs poorly when different clusters have diverse densities – marking more edges of large, dense clusters as important while marking all edges of smaller clusters as unimportant [12]. The alternative to the *Global* method is the *Local* strategy that is robust to the varying densities of clusters in the graph. *Local identification* method does not employ a global threshold; Rather, for each node i with degree d_i, the top $f(d_i) = d_i^e$ edges incident to i are marked as important. Here $e(e < 1)$ is a local identification exponent that adapts to the varying densities across varying parts of the graph and affects the global identification ratio. Smaller values of e results in a graph with fewer edges marked as important and vice versa. In this approach sorting and thresholding is applied to each vertex separately, allowing for the robustness necessary to accommodate varying densities across clusters in the graph. Note that in the *local* approach, at least one edge incident on each node is marked as important. The algorithm for *local* identification is given in Algorithm 1.

Figures 1(b) and 1(c) show the effect of applying the *global* and *local* approaches, respectively, on the example graph in Figure 1(a). The *local* approach does a much better job while the *global* approach marks all edges in the low density cluster as unimportant.

D. Fast Similarity Computation

The algorithms proposed above, under certain assumptions, have complexity $O(n.d_{avg}^2)$ where d_{avg} is the average degree of the graph [12], making it a prohibitively expensive process. Hence, a fast similarity computation is desirable. One popular method for efficiently computing approximate *Jaccard* similarity between two sets is minwise hashing [21].

Local Identification using Minwise Hashing

In this approach, k linear permutations are generated by generating k triplets (a, b, P). Then a length k signature for each node is computed by minhashing each node k times, resulting in a hash table of size $n*k$ for a graph with n nodes. For an edge (i, j), we compare the minhash signatures of nodes i and j, counting the number of matching minhashes. Note that the similarity of an edge is proportional to the number of matches. Thus, we can sort edges incident to a node i by the number of matches of each edge. Moreover, because the number of matches is limited to k, we can use a *linear time* counting sort algorithm to sort the edges. After sorting, we mark the top d_i^e edges as important for node i with degree d_i.

This reduces the time complexity of similarity based identification to $O(km)$, linear in the number of edges [12].

Algorithm 1 Local Importance Identification

Input: Graph $G = (V, E)$, Local Identification exponent e
Output: Edge Identified Graph, $G_{identified}$

```
G_identified ← G
for each vertex i ∈ V do
    Let d_i be the degree of i
    Let E_i be the set of edges incident to i
    for each edge e = (i, j) in E_i do
        e.sim = Sim(i, j) according to Eq. 1
    end for
    Sort all edges e ∈ E_i by e.sim
    Mark the top d_i^e % edges incident to i in G_identified as important
end for
Mark all remaining edges as unimportant
return G_identified
```

IV. Computing Model

Labeling the edges of a graph as important versus unimportant allows us to apply energy-efficient computation in subsequent steps of graph processing. We describe our proposed model for energy efficient computing in the following secgtion.

A. Energy-Efficient Computing Architecture

Voltage Over-Scaling (VOS) has been a popular technique for reducing power consumption, where the supply voltage

of a component or the entire system is reduced in order to gain quadratic power savings. However, the savings come at the cost of loss of accuracy due to *timing errors* in the targeted component. Initial proposals of VOS were bundled with error detection and correction schemes [22] in order to guarantee zero error, limiting scaling and the potential power benefits. Recently, researchers embraced errors allowing them to propagate to the end result of computations for algorithms that exhibit inherent fault-tolerance [8], [9],thereby enabling aggressive voltage scaling.

We propose to use an approximate computing architecture that resembles the one proposed by Esmaeilzadeh et al. [8]. The key idea is to double the set of functional units (FUs) so that one set operates on *precise* instructions while the other operates on *imprecise* instructions. Note that using only one set of FUs and varying operational voltage is not a feasible option for the following reasons:

- Precise and imprecise computations are usually finely interleaved in real algorithms.
- There is a penalty of 75-150 μs for switching supply voltage levels [23]. Frequent switching of voltage levels may extend the computation time and eliminate any potential power savings.

Hence, we assume an architecture that allows low-voltage computation via additional copies of functional units, along with support for the execution of algorithms with specific precision guarantees.

B. Error Models for VOS Architecture

Energy efficient computing via voltage scaling requires thorough assessment with an accurate error model in order to correctly explore the accuracy-power trade-off. Various error models have been used previously to correlate the energy savings to the output quality of applications. The most commonly-used models are simple and based on single bit-flip probabilities, uniform distribution models, or random values [8], [9], [7]. However, these models are not detailed enough and cannot fully capture the error behavior of FUs [24]. Tziantzioulis et al. [24] show that different FUs and even different bit locations of the same operation exhibit significantly different error behaviors. As a result, [24] proposed b-HiVE, a comprehensive and accurate error model that is applicable to a wide-range of FUs and takes the bit location, unit type and value correlations into consideration – increasing the accuracy of the models quite significantly.

In our evaluation we use the C++ software library developed by the authors of b-HiVE [24] to emulate the error behavior of voltage over-scaled FUs that execute the imprecise computations.

C. Graph Processing Model

As described in prior sections, we preprocess graphs in order to classify edges as important and unimportant. Then we execute the graph processing algorithms while apply energy-efficient approximate computing when processing edges that are marked unimportant by our identification algorithm. Computations involving important edges, however, are executed precisely. One benefit of our proposed technique is that the fast identification algorithm is executed once for a graph as a preprocessing step and the information is used for all future executions under the proposed paradigm.

V. Experimental Evaluation

We compare the outputs of our proposed techniques with the outputs produced by the precise execution of the respective algorithms. To obtain energy savings, we apply the identification algorithm as a preprocessing step and then apply modified versions of the graph processing algorithms. The modified algorithms vary from the original versions in that they perform computations on voltage over-scaled functional units when processing an unimportant edge. The voltage over-scaled arithmetic operations are modeled using the b-HiVE library [24]. The output quality of applications is reported using specific measures that are typical for the respective applications.

A. Applications

Regularized Markov Clustering (RMCL): RMCL [25] is an extension of Markov Clustering (MCL) [26] which is based on flow distributions in a stochastic flow matrix. Brief descriptions of MCL and RMCL follow:

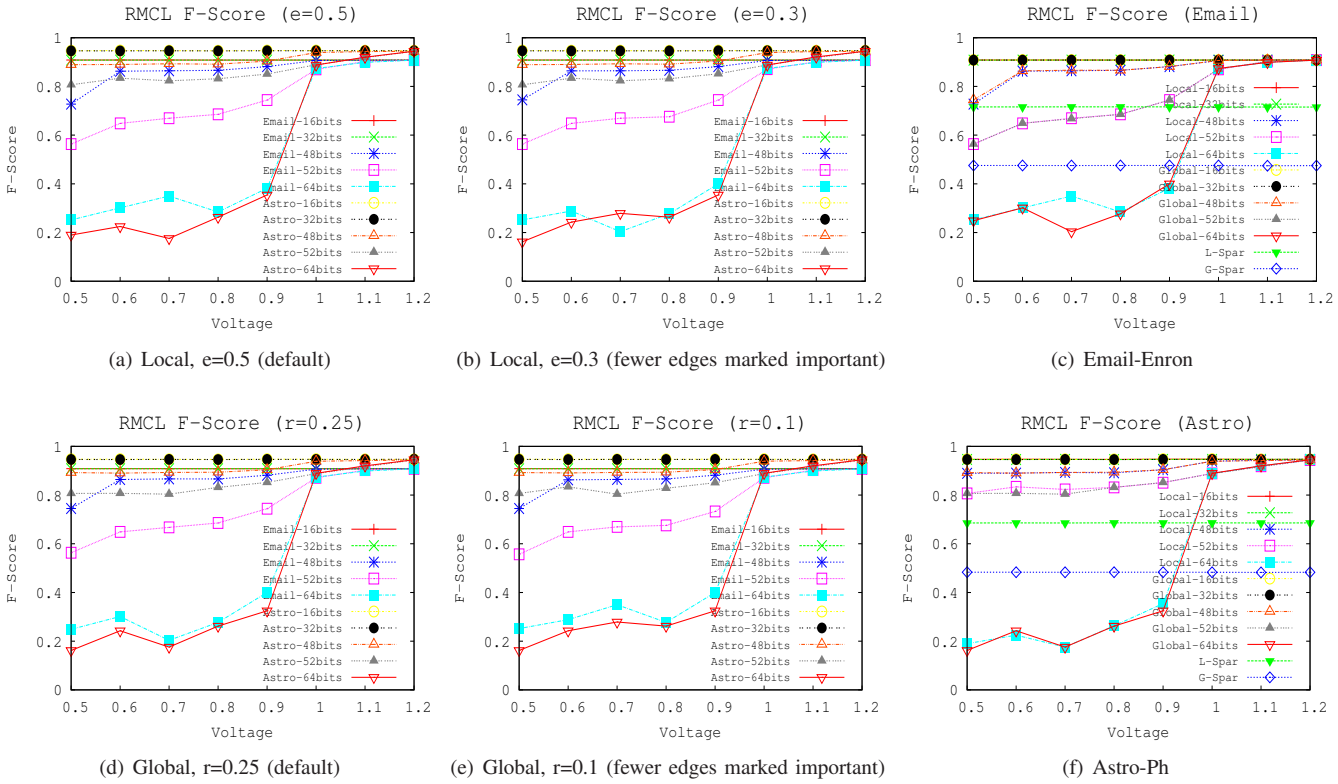
- **Markov Clustering (MCL)** [26] is based on the iterative application of two operations on the transition probability matrix (stochastic flow matrix) M of the graph. The algorithm is as follows:

Algorithm 2 Markov Clustering (MCL)

1: // M is the stochastic flow matrix, prepared using the adjacency matrix A
2: **repeat**
3: M := M_{Expand} := Expand(M)
4: M := Inflate(M, r)
5: M := Prune(M)
6: **until** M Converges

Here *Expand(M)* is $M \times M$. *Inflate(M, r)* corresponds to raising each element of M to its r-th power and normalizing the columns to sum to 1. Here r is the *inflation* parameter ($r > 1$) and is typically set to 2. *Prune(M)* prunes away smaller values (relative to other values in respective columns) in each column of matrix M. The remaining values are re-normalized to maintain the column stochastic property.

- **Regularized MCL (RMCL)** was proposed by Satuluri et al. [25] as an extension of MCL in order to address a major limitation of MCL—*imbalanced* clustering. RMCL does so by *regularizing* (or *smoothing*) the flow



(a) Local, e=0.5 (default) (b) Local, e=0.3 (fewer edges marked important) (c) Email-Enron

(d) Global, r=0.25 (default) (e) Global, r=0.1 (fewer edges marked important) (f) Astro-Ph

Figure 2. RMCL performance

distributions with respect to neighbors by changing the *Expand* ($M := M \times M$) step of MCL to *Regularize* ($M := M \times M_G$) step where M_G is the canonical transition matrix of the graph.

PageRank: is based on a "random surfer" who starts on a random web page and clicks on links and at some point starts on another random page—a phenomenon commonly known as "random walk with restart" [10]. The probability that the random surfer visits a page is its PageRank. For a graph $G = (V, E)$ where V is the set of N vertices and E is the set of *directed* edges, the PageRank vector p is a $(N \times 1)$ vector computed iteratively using the following equation:

$$p^{n+1} = cW^T p^n + (1 - c)p^n \quad (2)$$

where c is a *dumping* factor (usually set to 0.85). W is the *row normalized* matrix of adjacency matrix A.

B. Datasets

We run our experiments on a real-world graph from the SNAP dataset[1]. For all of these results, we keep the identification parameters at their default value, i.e., fast, approximate local identification with 30 hashes for computing approximate similarity and exponent parameter set to 0.5.

[1]http://snap.stanford.edu/data/index.html

Graph	Local Identification	Global Identification
Email-Enron	0.142s	0.137s
Astro-Ph	0.138s	0.125s

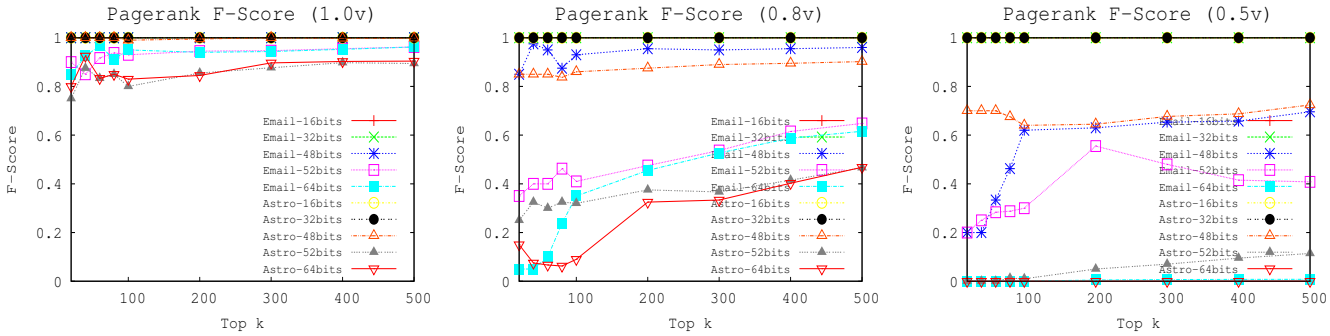
Table I
EDGE IDENTIFICATION TIME

Hence, for each node with degree d, we mark $\sqrt{d}$ of its edges as important and the remaining $d - \sqrt{d}$ edges as unimportant. We also evaluate the performance by limiting the energy-efficient computing to certain ranges of bits: low 16, low 32, low 48, low 52 (mantissa) and all 64 bits. This allows us to evaluate the resilience of low-order bits in computation.

All experiments are run on the Email-Enron dataset with 36692 vertices and 183831 edges, and the Astro-Ph dataset with 17903 vertices and 196972 edges. The time taken for the identification of edges is given in Table I.

C. RMCL Evaluation

We first show the results for the graph clustering algorithm in Figure 2. Figure 2(a) shows the result when the identification parameter e is set to the default value 0.5. Note that the *f-score* is computed with respect to the output produced by the *precise* version of the algorithm. Additionally, during the calculation of the average f-score, we exclude singleton clusters resulting in maximum f-score of 0.9232. As we see in the figures, when errors are limited to the low 32



(a) Voltage 1.0V (b) Voltage 0.8V (c) Voltage 0.5V

Figure 3. Pagerank performance at different operational voltages

bits of the floating point computations, we do not see any difference in the output quality. However, the quality starts dropping when we allow errors beyond the low 32 bits. Yet, as long as errors occur within the low 52 mantissa bits, we see quite graceful degradation as opposed to a significant drop in quality. For all 64 bits, however, going below $1.0V$ significantly affects the results.

Figure 2(a), 2(b), 2(d) and 2(e) show the impact of the identification parameter e (r) on the quality of RMCL output. We see the same trend across all settings, indicating that the identification parameter does a good job in identifying the most important edges so that the quality of the clustering algorithm is stable even when we set the parameter value to $e = 0.3$ ($r = 0.1$), marking most of the edges as *unimportant*. This gives us confidence that the identification algorithm is able to mark the most critical edges as *important* leading to good output quality.

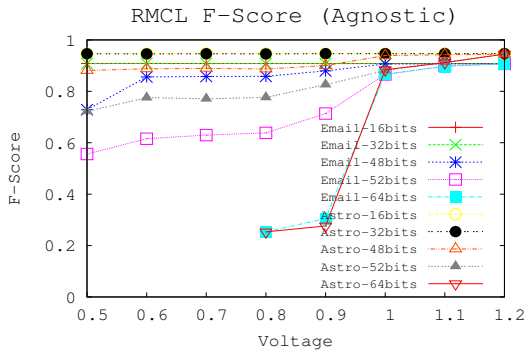


Figure 4. RMCL Performance for agnostic error injection

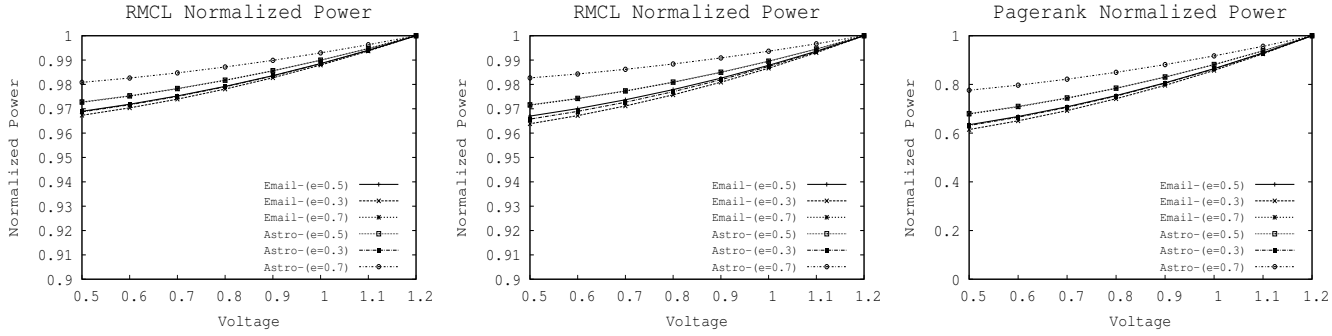
In Figure 4 we show the performance of RMCL when low voltage operation is applied across all edges irrespective of their importance. As shown in the chart, when errors are present in all 64 bits of computation, the agnostic setting crashes below $0.8V$. This result shows the inherent resilience of the application. However, it also shows that an *agnostic* approach is not safe as the algorithm crashes when we go below $0.8V$.

Comparison with Sparsification

In Figure 2(c) we show the performance for identification using *local* and *global* methods as well as results when we use *local* and *global* sparsification approaches to reduce the number of edges from the graph. We first apply *local/global* sparsification on the graph with respective default parameter values and apply RMCL on the sparsified graph. Note that sparsification reduces the number of edges in the graph, hence the processing is faster. However, as it is evident in Figures 2(c) and 2(f), removing edges impacts the quality of clustering. For example, applying *local* sparsification and then RMCL causes the F-Score of RMCL to drop from **0.908** to **0.715** which is less than what our proposed framework can achieve even at $0.5V$ when errors are limited to the low 48 bits, or at $0.9V$ when errors are limited to the *mantissa* bits. The impact is even more severe for the *global* sparsification case where our proposed methods achieve better performance even at $0.5V$ when errors are within the low 52 *mantissa* bits of the FP computation. The effect is more severe on the Astro-Ph dataset as shown in Figure 2(f). The application of *local* and *global* sparsification causes the F-Score to drop from **0.946** to **0.715** and **0.475**, respectively. For this dataset our proposed methods can outperform the *local* sparsification method at $0.5V$ when errors are limited to the low 52 *mantissa* bits of the FP computation.

D. PageRank Evaluation

Figure 3 shows the performance of the PageRank algorithm at different operational voltage levels. To measure the quality of PageRank, we compare the *top k* ranked pages for any given setting with that of the precise *top k* for a given value of k. We report F-Score for performance. We see that performance degrades below $1.0V$ when errors are allowed in all 64 bits of computation. However, when errors are limited to the low 48 bits, even at voltage as low as $0.8V$ we see results with f-score above 0.8. Another interesting observation is when errors due to energy-efficient computing are limited to the low 32 bits, there is virtually no impact on the output quality. This is very promising because the low-



(a) RMCL Normalized Power (Local) (b) RMCL Normalized Power (Global) (c) Pagerank Normalized Power

Figure 5. RMCL & Pagerank power consumption using the proposed framework

order bits show substantial resilience to random bit errors.

E. Power Savings

In this section we provide an estimation of power savings when deploying our proposed framework. In order to get instruction counts based on precise/imprecise modifiers, we use a modified version of Gem5 [27] that supports our marking of instructions and schedules instructions based on their precision level. To compare the power consumption of our proposed scheme against a base architecture we used McPat [28] to calculate the power consumption of the functional units. Simulations on Gem5 are time and resource intensive. Hence, to get the approximate analysis for larger graphs, we use Gem5 simulation along with McPat on a small, synthetic graph and then use the information of respective graphs to extrapolate the power consumption numbers. Thus, these numbers are an approximate analysis of the power consumption. Note that the marking of instructions remains the same for Gem5 simulation. We just extrapolate based on the number of edges and the precise/imprecise instruction ratio.

Figure 5 shows the power consumption for RMCL on our proposed framework. We show the effect of *local* vs *global* identifications in Figures 5(a) and 5(b). We see power savings in the range of **3–6%** for these two graphs. As we use off-the-shelf implementation of RMCL for our experiments, we can mark only **7–8**% of total FP operations as imprecise whereas for PageRank we can mark **70**% as imprecise. Given only **7**% FP operations marked as imprecise, our proposed techniques can still save more than **3**% power consumption. We are confident that a more careful implementation of RMCL would result in more power savings. Figure 5(c) shows the normalized power consumption of the PageRank algorithm for the proposed framework. Our proposed methods can offer significant power savings while providing high-quality results (see voltage levels 0.9 and 0.8 at Figure 3). When errors are limited to the low 48 bits, using our approach at 0.9 and 0.8 volts we can save **16.93%** and **21.5%** of power, respectively, compared to the base architecture. Moreover, when errors are limited to the low 32 bits, we can go all the way down to $0.5V$ with PageRank, saving more than **30%** in power consumption. This can provide significant energy and cost savings in large scale deployments. An important attribute of controlling the level of power savings vs quality is the identification parameter e. A lower e value (i.e., $e = 0.3$) leads to identification of fewer edges as important, thus reduces power consumption, whereas for larger values (i.e., $e = 0.7$) we identify more edges as important, and thus increase power consumption.

VI. Conclusion

In this paper we propose a novel framework for energy-efficient graph processing. We propose fast and efficient techniques to classify graph edges as *important* and *unimportant* with respect to the edge's role in preserving the community structure in the graph. To this end we employ a fast minwise hashing technique to measure the similarity of the vertices at the endpoints of each edge. Based on this similarity we rank edges and then identify a user-defined fraction of them as *important*. We propose and evaluate two approaches: *global* and *local* identification. We show that the *local* approach is more effective for real graphs where the community sizes vary substantially, as the *global* approach identifies globally without taking vertex degrees into account. Our proposed technique allows the user to select the fraction of edges that should be marked as important. Finally, once edges are identified at a preprocessing step, the graphs can be processed using our proposed paradigm where energy-efficient computing is applied across edges that have been marked as *unimportant*. Our experimental evaluation shows that our proposed framework can provide significant power savings while preserving quality of output during large scale graph processing.

Acknowledgment

This work is supported by NSF grants **CCF-1218768**, **CCF-1453853** and **CCF-1217353**.

REFERENCES

[1] T. Mudge, "Power: A first-class architectural design constraint," *Computer*, vol. 34, no. 4, pp. 52–58, Apr. 2001.

[2] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *38th ISCA*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 365–376.

[3] V. J. Reddi and M. S. Gupta, *Resilient Architecture Design for Voltage Variation*, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2013.

[4] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *IEEE/ACM MICRO*, 2003, pp. 7–18.

[5] S. Das, S. Member, D. Roberts, S. Member, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "A self-tuning dvs processor using delay-error detection and correction," *IEEE Journal of Solid-State Circuits*, vol. 41, p. 2006, 2005.

[6] E. Krimer, P. Chiang, and M. Erez, "Lane decoupling for improving the timing-error resiliency of wide-simd architectures," *SIGARCH Comput. Archit. News*, vol. 40, no. 3, pp. 237–248, Jun. 2012.

[7] J. Sartori, J. Sloan, and R. Kumar, "Stochastic computing: Embracing errors in architectureand design of processors and applications," in *14th International Conference on CASES*, ser. CASES '11. New York, NY, USA: ACM, 2011, pp. 135–144.

[8] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," *SIGPLAN Not.*, vol. 47, no. 4, pp. 301–312, Mar. 2012.

[9] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: Approximate data types for safe and general low-power computation," in *PLDI '11*. New York, NY, USA: ACM, 2011, pp. 164–174.

[10] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report, November 1999.

[11] S. Parthasarathy and S. M. Faisal, "Network clustering," in *Data Clustering: Algorithms and Applications*, 2013, pp. 415–456.

[12] V. Satuluri, S. Parthasarathy, and Y. Ruan, "Local graph sparsification for scalable clustering," in *ACM SIGMOD*, ser. SIGMOD '11. New York, NY, USA: ACM, 2011, pp. 721–732.

[13] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in the cloud," *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 716–727, Apr. 2012.

[14] S. M. Faisal, S. Parthasarathy, and P. Sadayappan, "Global graphs: A middleware for large scale graph processing," in *2014 IEEE Big Data 2014, Washington, DC, USA, October 27-30, 2014*, 2014, pp. 33–40.

[15] J. Miao, A. Gerstlauer, and M. Orshansky, "Multi-level approximate logic synthesis under general error constraints," in *ICCAD '14*. Piscataway, NJ, USA: IEEE Press, 2014, pp. 504–510.

[16] D. R. Karger, "Random sampling in cut, flow, and network design problems," in *26th Annual ACM STOC*, ser. STOC '94. New York, NY, USA: ACM, 1994, pp. 648–657.

[17] M. Tumminello, T. Aste, T. Di Matteo, and R. N. Mantegna, "A tool for filtering information in complex systems," *National Academy of Sciences of the USA*, vol. 102, no. 30, pp. 10 421–10 426, Jul. 2005.

[18] J. B. Glattfelder and S. Battiston, "Backbone of complex networks of corporations: The flow of control," *Physical Review E*, vol. 80, no. 3, pp. 036 104+, Sep. 2009.

[19] A. S. Maiya and T. Y. Berger-Wolf, "Sampling community structure," in *WWW'10*. New York, NY, USA: ACM, 2010, pp. 701–710.

[20] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review*, vol. E 69, no. 026113, 2004.

[21] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations (extended abstract)," in *STOC '98*. New York, NY, USA: ACM, 1998, pp. 327–336.

[22] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *MICRO 36*, Washington, DC, USA, 2003.

[23] B. Childers, H. Tang, and R. Melhem, "Adapting processor supply voltage to instruction-level parallelism," in *Kool Chips 2000 Workshop*, 2000.

[24] G. Tziantzioulis, A. M. Gok, S. M. Faisal, N. Hardavellas, S. O. Memik, and S. Parthasarathy, "b-hive: a bit-level history-based error model with value correlation for voltage-scaled integer and floating point units," in *52nd Annual DAC, San Francisco, CA, USA, June 7-11, 2015*, 2015, p. 105.

[25] V. Satuluri and S. Parthasarathy, "Scalable graph clustering using stochastic flows: applications to community discovery," in *KDD '09*. ACM, 2009.

[26] S. M. Van Dongen, "Graph Clustering by Flow Simulation," Ph.D. dissertation, University of Utrecht, The Netherlands, 2000.

[27] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, and Others, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[28] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO-42, 2009*, Dec 2009, pp. 469–480.