# A Jaccard Weights Kernel Leveraging Independent Thread Scheduling on GPUs

Hartwig Anzt
Karlsruhe Institute of Technology, Germany
University of Tennessee, USA
Email: hanzt@icl.utk.edu

Jack Dongarra
University of Tennessee USA
Oak Ridge National Laboratory, USA
University of Manchester, UK
Email: dongarra@icl.utk.edu

***Abstract*—Jaccard weights are a popular metric for identifying communities in social network analytics. In this paper we present a kernel for efficiently computing the Jaccard weight matrix on GPUs. The kernel design is guided by fine-grained parallelism and the independent thread scheduling supported by NVIDIA's Volta architecture. This technology makes it possible to interleave the execution of divergent branches for enhanced data reuse and a higher instruction per cycle rate for memory-bound algorithms. In a performance evaluation using a set of publicly available social networks, we report the kernel execution time and analyze the built-in hardware counters on different GPU architectures. The findings have implications beyond the specific algorithm and suggest a reformulation of other data-sparse algorithms.**

## I. Introduction

Being able to identify communities and hubs in social networks is one of the key challenges in social network analytics [1]. Identifying communities and conflating them in a more compact representation also helps to summarize huge networks [1], which then facilitates uncovering existing behavioral patterns (user behavior analytics [2]) and predicting emergent properties of the network (predictive analytics [3]). In this regard, community detection can provide functionality similar to clustering, a data mining technique used to partition a data set into disjoint subsets based on the similarity of data points [4].

To detect communities, it is necessary to quantify the relationship between individuals or groups in the dataset. Jaccard weights [5] represent the ratio between the number of individuals shared by two groups and the sum of the individuals that are part of either group or both groups. In a graph-theoretical interpretation of the dataset, individuals correspond to vertices and relationships to edges. The Jaccard weight then extends to the individual level as the ratio between the number of joint neighbors of two vertices (vertices connected via a single edge) and the total number of neighbors. The *Jaccard weight matrix* assembles the connectivity information of all vertex combinations in a single matrix and allows for identifying strongly connected subsets that correspond to communities in the data set.

In this paper we propose a GPU kernel for computing the Jaccard weight matrix for unweighted, undirected graphs in element-wise parallel fashion. Following some brief review of the Jaccard weight concept in Section II, we present in Section III a kernel that computed the Jaccard weight matrix in element-wise parallel fashion. The algorithm design is motivated by the Jaccard weight matrix, deriving as a sparsity-preserving sparse matrix multiplication. The kernel realization is driven by the fine-grained parallelism of modern GPU architectures and the recently introduced independent thread scheduling in NVIDIA's Volta architecture. In Section IV, we use a set of test matrices to show the kernel's competitiveness with previous implementations, and analyze the kernel execution characteristics on different architecture generations. Evaluating the hardware counters, we relate the higher kernel execution performance on NVIDIA's Volta architecture to the increased bandwidth and the improvements in the single instruction, multiple thread (SIMT) [6] execution model. We outline our conclusions in Section V.

## II. Jaccard weights and related work

Originating from set theory, the Jaccard weight [5] is a general measure for the similarity of two sets $S_1$ and $S_2$. In more precise terms, the Jaccard weight quantifies how many elements are shared by the two sets in relation to the cardinality of the union [7]:

$$\mathcal{J}(S_1 S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} \tag{1}$$

with $|\cdot|$ denoting the cardinality. The Jaccard weight is $0$ for disjoint sets, $1$ for identical sets, and takes values in the range $(0,1)$ otherwise.

In graph theory, Jaccard weights are used to measure the connectivity between two vertices. For a graph $G = (V,E)$ with vertex set $V = \{v_1, \dots v_n\}$ and edge set $E = \{(i_1,j_1),(i_2,j_2),\dots(i_m,j_m)\}$, the Jaccard weights quantify how many common direct neighbors two vertices share in relation to the total number of direct neighbors. With $\mathcal{N}(v_i)$ being the neighborhood of $v_i$, which is all vertices that can be accessed from $v_i$ by traversing a single edge, we can define for two vertices $i,j$ the

- intersection weight $w_I(i,j) = |\mathcal{N}(i) \cap \mathcal{N}(j)|$,
- union weight $w_U(i,j) = |\mathcal{N}(i) \cup \mathcal{N}(j)|$.

In this notation, the Jaccard weight becomes

$$\mathcal{J}_{ij} = \mathcal{J}(i,j) = \frac{w_I(i,j)}{w_U(i,j)}. \tag{2}$$

If we represent the undirected graph $G$ in the form of a (symmetric) pattern matrix, the edges connecting vertices correspond to nonzeros in the matrix, and the intersection weight $w_I(i,j)$ corresponds to the vector product $< A(i,:), A(j,:) >$ of the rows $i$ and $j$ of the adjacency matrix $A$, and the union weight $w_U(i,j)$ corresponds to

$$w_U(i,j) = \sum_{:}(|A(i,:)|) + \sum_{:}(|A(j,:)|) - < A(i,:), A(j,:) >$$

where $\sum_{:}(|v(:)|)$ denotes the sum of the absolute values in a vector. The concept of Jaccard weights can easily be extended to weighted graphs. Then, the nonzeros in the adjacency matrix represent the weights of the distinct edges, and the metrics defined above replace the cardinality of neighborhoods with the absolute sum of the edge weights.

If the Jaccard weights for all vertex combinations $(i,j)$ are available, Jaccard clustering can be used to partition the graph into subsets, with vertices of the same cluster being strongly connected in the relative sense.

## III. Fine-grained parallel Jaccard matrix computation

The Jaccard weight matrix kernel we propose is motivated by the observation that the Jaccard weight matrix can be interpreted as a sparsity-preserving sparse general matrix-matrix multiplication (SpGEMM) [7]. More precisely, for an adjacency matrix $A$, the Jaccard matrix $\mathcal{J}$ can be computed as $\mathcal{J} = A \odot_J A^T$ where $\odot_J$ is a algorithm-specific matrix multiplication operator that fulfills $\mathcal{S}(\mathcal{J}) = \mathcal{S}(A)$ for the nonzero pattern $\mathcal{S}$. This implies that the customized matrix multiplication does not need a symbolic multiplication phase–nonzero Jaccard weights $\mathcal{J}_{ij} \in \mathcal{J}$ are only possible in locations $(i,j)$ that are nonzero in $A$. Jaccard weights can become numerically zero, but accepting explicit zeros in the sparse Jaccard weight matrix makes it possible to use the sparsity pattern of $A$ and compute the weights for the nonzero locations in $\mathcal{J}$. For an unweighted graph $G$, the matrix $A$ becomes a pattern matrix, and $\sum_{:}(|A(i,:)| + \sum_{:}(|A(j,:)|$ is the sum of the nonzeros in row $i$ and $j$ – which is readily available if $A$ is stored in compressed sparse row (CSR) format.

For efficiently realizing the Jaccard matrix kernel on GPUs, we propose to parallelize across the nonzero elements in $\mathcal{J}$, see Figure 1. While this ensures workload balance in terms of the number of Jaccard weights each thread processes, the cost of computing a Jaccard weight $\mathcal{J}_{ij}$ heavily depends on the sparsity pattern of the matrix $A$. In particular, the computational cost depends on the cost of the sparse dot product, which again depends on the sparsity pattern of the rows $i$ and $j$. The sparsity pattern also determines the memory access pattern of the kernel. Even for elements adjacent in the Jaccard weight matrix, the computational cost and memory access pattern can differ significantly. For warp-synchronizing GPU architectures, this can easily become a performance problem as the 32 threads of a warp execute simultaneously until all threads of the warp complete the computation of the respectively assigned matrix entry [8]. Furthermore, because the execution

```
__global__ void
jaccardweights_kernel(int num_rows, int num_cols, int nnzJ,
    int *rowidxJ, int *colidxJ, double *valJ,
    int *rowptrA, int *colidxA, double *valA) {
  int i, j, il, iu, jl, ju;
  int k = blockDim.x * gridDim.x * blockIdx.y
      + blockDim.x * blockIdx.x + threadIdx.x;
  double sum_i, sum_j, cap;
  if (k < nnzJ) {
    i = rowidxJ[k]; j = colidxJ[k];
    if( i != j ){
      il = rowptrA[i]; iu = rowptrA[j];
      sum_i = 0.; sum_j = 0.; cap = 0.;
      sum_i = rowptrA[i+1] - rowptrA[i];
      sum_j = rowptrA[j+1] - rowptrA[j];
      while (il < rowptrA[i+1] && iu < rowptrA[j+1]) {
        jl = colidxJ[il]; ju = rowidxJ[iu];
        // if there are actual values:
        // cap = ( jl == ju ) ? valJ[il] * valJ[iu] : sp;
        // else
        cap = ( jl == ju ) ?  cap + one : cap;
        il = ( jl <= ju ) ? il+1 : il;
        iu = ( ju <= jl ) ? iu+1 : iu;
      }

      valJ[k] = cap / ( sum_i + sum_j - cap);
    } else {
      valJ[k] = 1.0;
} } }
```

Fig. 1. CUDA kernel computing the Jaccard weight matrix in CSR format.

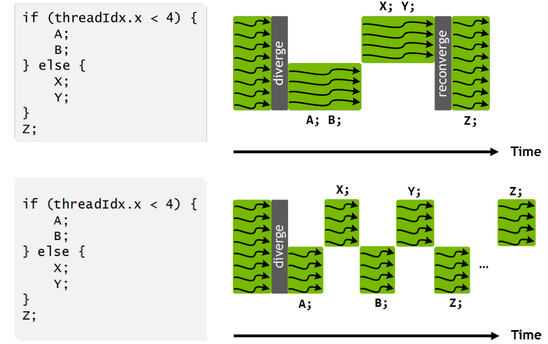



Fig. 2. SIMT execution model. Top: NVIDIA GPU architectures $<$ SM70; Bottom: NVIDIA GPU architecture SM70 (Volta) [9].

of branches cannot be interleaved, thread divergence inevitably results in serialized execution for different portions of the warp, and all other threads of the warp are waiting until the branch is completed (see top in Figure 2).

NVIDIA's Volta technology greatly advances the SIMT [6] execution model. It supports independent thread scheduling, which enables finer-grain synchronization and cooperation between parallel threads in a program [9]. Prior GPU architectures used a single program counter shared amongst all 32 threads of a warp, combined with an active mask that specifies which threads of the warp are active at any given time – leaves threads that are not executing a branch inactive [6]: All threads execute together only after the divergent section is completed. In the Volta architecture, each thread features its own program counter, which allows threads of the same warp to execute different branches of a divergent section simultaneously (see bottom in Figure 2). To maximize parallel efficiency, the Volta

| | K20m | P100 | V100 |
|---|---|---|---|
| Architecture | Kepler | Pascal | Volta |
| DP Performance | 1.2 TFLOPs | 5.3 TFLOPs | 7 TFLOPs |
| SP Performance | 3.5 TFLOPs | 10.6 TFLOPs | 14 TFLOPs |
| HP Performance | – | 21.2 TFLOPs | 112 TFLOPs |
| SMs | 13 | 56 | 80 |
| Operating Freq. | 0.75 GHz | 1.15 GHz | 1.53 GHz |
| Mem. Capacity | 5 GB | 16 GB | 16 GB |
| Mem. Bandwidth | 208 GB/s | 732 GB/s | 900 GB/s |
| Sustained BW | 146 GB/s | 500 GB/s | 742 GB/s |
| L2 Cache Size | 1.5 MB | 4 MB | 6 MB |
| L1 Cache Size | 64 KB | 64 KB | 128 KB |

TABLE I
KEY CHARACTERISTICS OF THE HIGH-END NVIDIA GPUS. THE HALF (HP) PERFORMANCE OF THE V100 IS FOR THE 8 TENSOR CORES. THE SUSTAINED MEMORY BANDWIDTH IS MEASURED USING THE BANDWIDTH TEST SHIPPING WITH THE CUDA SDK.

| | Matrix | rows/cols | nonzeros | memory* |
|---|---|---|---|---|
| SW | smallword | 100,000 | 999,996 | 4.4 MB |
| PA | preferentialAtt. | 100,000 | 499,985 | 2.4 MB |
| CA | caidaRouterLev. | 192,244 | 609,066 | 3.2 MB |
| AD | coAuthorsDBLP | 299,067 | 977,676 | 5.1 MB |
| CI | citationCites. | 268,495 | 1,156,647 | 5.7 MB |
| PD | coPapersDBLP | 540,486 | 15,245,729 | 63.1 MB |
| PC | coPapersCites. | 434,102 | 16,036,720 | 65.9 MB |
| AS | as-Skitter | 1,696,415 | 22,190,596 | 95.5 MB |
| HL | hollywood-2009 | 1,139,905 | 113,891,327 | 460.1 MB |

TABLE II
TEST MATRICES ALONG WITH KEY CHARACTERISTICS. THE MEMORY ESTIMATE IS THE MEMORY REQUIRED TO STORE THE PROBLEM AS PATTERN MATRIX IN CSR FORMAT, USING 32-BIT INTEGERS FOR THE ROW POINTER AND THE COLUMN INDEXES.

| | K20m | P100 | V100 | TITAN X | TITAN Xp* |
|---|---|---|---|---|---|
| SW | 1.61e-3 | 2.45e-4 | 7.75e-5 | 4.43e-4 | – |
| PA | 2.08e-2 | 3.57e-3 | 1.09e-3 | 4.35e-3 | 5.00e-3 |
| CA | 2.09e-2 | 2.92e-3 | 8.31e-4 | 3.59e-3 | 8.00e-3 |
| AD | 1.43e-2 | 1.97e-3 | 5.39e-4 | 2.66e-3 | 4.00e-3 |
| CI | 4.66e-2 | 7.75e-3 | 2.14e-3 | 8.97e-3 | 9.00e-3 |
| PD | 1.55e+0 | 1.57e-1 | 3.52e-2 | 2.02e-1 | 3.08e-1 |
| PC | 2.48e+0 | 2.74e-1 | 5.86e-2 | 3.22e-1 | 5.38e-1 |
| AS | 1.03e+1 | 1.39e+0 | 3.16e-1 | 1.46e+0 | 5.02e-1 |
| HL | 4.01e+1 | 1.14e+1 | 2.48e+0 | 9.57e+0 | 1.28e+1 |

TABLE III
RUNTIME [S] FOR GPU KERNEL TO COMPLETE THE COMPUTATION OF THE JACCARD WEIGHTS MATRIX ON DIFFERENT HARDWARE ARCHITECTURES. THE RUNTIME RESULTS IN THE LAST COLUMN ARE THOSE FROM THE REFERENCE JACCARD KERNEL PRESENTED IN FENDER ET AL. [7].

architecture includes a schedule optimizer which determines how to group active threads from the same warp together into SIMT units. When computing distinct elements in the Jaccard weight matrix, the schedule optimizer can accumulate the cases where the matrix rows have the same nonzero pattern.

While the benefits of interleaving the execution of distinct branches may have a moderate effect on compute-bound kernels, memory-bound algorithms like the Jaccard weight matrix kernel may complete the divergent region faster. This stems from the fact that data read by one thread of the warp may later be used by another thread of the warp executing a different branch. If the branches are executed in order, the data may have to be loaded from the L2 cache attached to the memory controllers, or (even worse) reread from main memory.

## IV. EXPERIMENTAL PERFORMANCE ANALYSIS

### A. Experimental framework

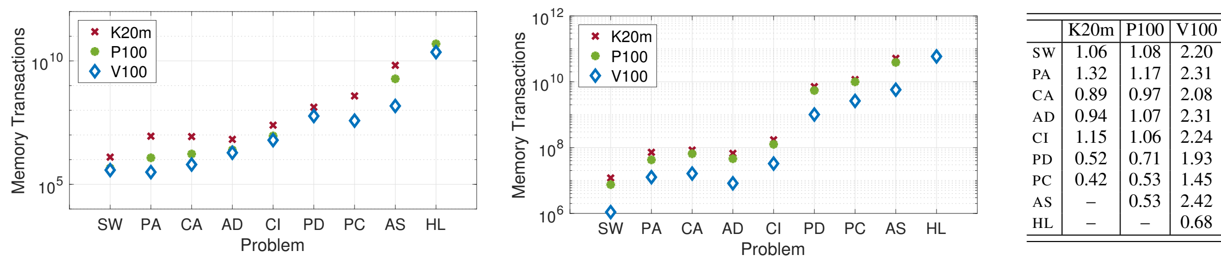
In this section we experimentally assess the performance of the proposed kernel on different GPU architectures. The NVIDIA K20m and the NVIDIA P100 GPUs belong to the Kepler and Pascal generations, respectively. Both architectures feature the traditional SIMT execution model. The NVIDIA V100 is part of the Volta generation where each thread has its own program counter. This allows threads to be scheduled independently. In Table I we list some of the key characteristics of the GPU architectures [10], [9], [11]. All computations are executed on the GPU, the kernel is implemented in the CUDA programming model; CUDA in version 9.0 was used to compile and run the kernels. By default, we use a thread block size of 512. We note that this is not always the best choice for a specific architecture / test matrix combination. However, problem-dependent optimization is costly, and this choice has proven to yield good performance for many settings.

We assess the performance of the new Jaccard weights kernel for the same set of test matrices that was previously used in Fender et al. [7]. The matrices are all available in the SuiteSparse [12] matrix collection and are listed along with some key characteristics in Table II. We consider all matrices as unweighted, undirected graphs. For convenience, we list the amount of memory needed to store the distinct problems as a pattern matrix in CSR format using 32 bit integers (see "memory*"). The combined cache of all multiprocessors is too small to fit the pattern matrix; the cache of a single multiprocessor is only a fraction of the memory volume required.

### B. Experimental results

In Table III we report the total execution time for computing the Jaccard weight matrix kernel. We observe speedup factors between $3\times$ and $10\times$ when moving from the K20m GPU to the P100 GPU, and speedup factors between $3\times$ and $5\times$ when moving from the P100 GPU to the V100 GPU. In Table III we also include the kernel runtime for an NVIDIA TITAN X [13] GPU. This device is part of the consumer line of NVIDIA's Pascal architecture and is the predecessor to the TITAN Xp GPU, which is used in Fender et al. [7] to benchmark an alternative algorithm for computing the Jaccard weight matrix. Compared to the TITAN X GPU, the TITAN Xp GPU features a higher number of CUDA cores (3840 vs. 3584) and a higher main memory bandwidth (548 GB/s vs. 480 GB/s).

An important aspect of the new Jaccard weights kernel's performance is the reuse of data. Data originally read from main memory that resides in the L2 cache can be accessed significantly faster by the processing cores. The size of the L2 cache is increased for the newer architectures. Similarly, the size of the multiprocessor's local cache (L1 cache) is increased from 64 KB per multiprocessor (Pascal generation) to 96 KB per multiprocessor for the Volta architecture (see Table I.




| | K20m | P100 | V100 |
|---|---|---|---|
| SW | 1.06 | 1.08 | 2.20 |
| PA | 1.32 | 1.17 | 2.31 |
| CA | 0.89 | 0.97 | 2.08 |
| AD | 0.94 | 1.07 | 2.31 |
| CI | 1.15 | 1.06 | 2.24 |
| PD | 0.52 | 0.71 | 1.93 |
| PC | 0.42 | 0.53 | 1.45 |
| AS | – | 0.53 | 2.42 |
| HL | – | – | 0.68 |

Fig. 3. Main memory read transactions (DRAM_READ_TRANSACTIONS, left) and L2 cache read transactions (L2_READ_TRANSACTIONS, CENTER, and Executed instructions per cycle (IPC) of the Jaccard weights kernel for the distinct problem/architecture configurations. (Missing data reflects overflow of the hardware counter.)

In Figure 3 we report the data from hardware counters *dram_read_transactions* (left) and L2_READ_TRANSACTIONS (right) available via NVIDIA's *nvprof* profiler. As expected, the smaller L2 cache size on the K20m GPU limits data reuse, and the Jaccard weights kernel requires more main memory reads (see left of Figure 3). The differences in the main memory read volume between the P100 GPU and the V100 GPU are much smaller. With the total main memory transactions volume on the P100 GPU and the V100 GPU being similar, the performance of a memory-bound kernel is expected to strongly correlate with the memory bandwidth. We note the sustained memory bandwidth of the P100 is 3.42× higher than for the K20m, and the sustained memory bandwidth of the V100 is 1.48× higher than for the P100 (see Table I). At the same time, the Jaccard weight matrix kernel executes up to 5× faster on the V100. Thus, the V100 GPU's higher memory bandwidth is insufficient in offsetting the performance differences between the Pascal and the Volta architectures. The L2 cache read volume (Figure 3, left) being significantly larger than the main memory read volume indicates the efficient reuse of cached data. Counterintuitive to the larger L2 cache size, the L2 read volume of the V100 is on average 80% smaller than the the L2 read volume of the K20m and the P100 GPUs. This indicates that the V100 architecture makes more efficient use of data that is read into the multiprocessor memory.

On the right in Figure 3 we report the respectively-achieved instruction per cycle (IPC) rate. Although two generations apart in terms of architecture, the K20m and the P100 achieve very similar IPC rates. This indicates that both architectures use the same SIMT execution model, with one scheduler handling the scheduling of all threads in a warp. The 2× higher IPC rate is likely enabled by each thread having its own scheduler, and the possibility of interleaving the execution of divergent branches if convenient. For memory-bound algorithms reusing some data, the scheduler can introduce the execution of a different branch at the point when required data is present in the multiprocessor cache. This avoids the expensive reloading of data from L2 cache or main memory. Consequently, less threads have to stall waiting for data, and a higher IPC rate can be achieved.

## V. Summary and Future Work

In this paper we proposed a Jaccard weight matrix kernel for GPU architectures. The Jaccard weight is an important tool for identifying communities in big data analytics. The kernel derives as element-parallel sparsity preserving matrix multiplication and makes efficient use of NVIDIA's improved SIMT execution model. For a set of test matrices we reported performance and hardware counters on different architectures.

## Acknowledgment

This material is based upon work supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Number DE-SC0016513. H. Anzt was supported by the "Impuls und Vernetzungsfond of the Helmholtz Association" under grant VH-NG-1241. The authors would like to thank the High Performance Computing & Architectures (HPCA) group at the University of Jaume for granting access to the TITAN X GPU.

## References

[1] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management*, vol. 35, no. 2, pp. 137 – 144, 2015.
[2] K. Yamaguchi, *Leveraging Advertising Data For Behavioral Insights*, Marketing Land, https://marketingland.com/leveraging-advertising-data-for-behavioral-insights-46467, 2013.
[3] D. T. Larose and C. D. Larose, *Data Mining and Predictive Analytics*, 2nd ed. Wiley Publishing, 2015.
[4] C. C. Aggarwal, *Social Network Data Analytics*, 1st ed. Springer Publishing Company, Incorporated, 2011.
[5] M. Lecandowsky and D. Winter, "Distance between Sets," *Nature*, vol. 234, no. 5323, pp. 34–35, 1971.
[6] NVIDIA Corp., *CUDA C Programming Guide*, http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.
[7] A. Fender, N. Emad, S. G. Petiton, J. Eaton, and M. Naumov, "Parallel jaccard and related graph clustering techniques," in *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA@SC 2017, Denver, CO, USA, November 13, 2017*, 2017, pp. 4:1–4:8.
[8] H. Anzt, E. Chow, and J. Dongarra, "On block-asynchronous execution on GPUs," LAPACK Working Note, Tech. Rep. 291, 2016.
[9] NVIDIA Corp., "Whitepaper: NVIDIA TESLA V100 GPU ARCHITECTURE," 2017.
[10] ——, "Whitepaper: Kepler GK110," 2012.
[11] ——, "Whitepaper: NVIDIA Tesla P100," 2016.
[12] "SuiteSparse Matrix Collection," https://sparse.tamu.edu, accessed in April 2018.
[13] NVIDIA Corp., *https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/*.