Hypergraph Partitioning via Geometric Embeddings

Sepideh Maleki
Department of Computer Science
The University of Texas at Austin
Email: smaleki@cs.utexas.edu

Udit Agarwal
Oden Institute
The University of Texas at Austin
Email: udit@utexas.edu

Keshav Pingali
Department of Computer Science
The University of Texas at Austin
Email: Pingali@cs.utexas.edu

***Abstract*—Hypergraph partitioning has been used in many VLSI domains such as floor-planning, placement, and logic synthesis. Circuits are modeled as hypergraphs in which nodes represent the pins of the circuit and hyperedges represent nets from the output pin of a gate to the input pins of other gates, and the nodes are partitioned into a desired number of clusters so that a metric such as the number of cut hyperedges is minimized. Existing hypergraph partitioning techniques consider only the *topology* of the hypergraph (connectivity between nodes) and ignore its *geometry* (positions of nodes in 2D or 3D space). This can lead to sub-optimal partitioning. In this paper, we describe an *embedding*-based approach for hypergraph partitioning that considers the geometry of circuits, which leads to better quality partitions, while ensuring *strong determinism*.**

I. Introduction

Hypergraph partitioning has applications in many areas including VLSI design [1], data-mining, Boolean satisfiability and numerical linear algebra. A hypergraph H = (V, E) is defined as a set of vertices V , and set of hyperedges E, where each hyperedge is a subset of V . The *hypergraph partitioning* problem is to create k roughly equal partitions of the vertices of a hypergraph such that a given objective function over hyperedges is optimized. A common objective function is to minimize the number of hyperedges connecting vertices in different parts, also known as edge cut. A number of other objective functions have been considered in the literature [1].

Our primary motivation for studying hypergraph partitioning comes from VLSI design. A modern circuit may contain millions of objects and nets. Hypergraphs can be used to represent circuits: vertices represent standard cells and macros, and hyperedges represent nets. Figure 1 shows a gate-level circuit and its hypergraph representation. Hypergraph partitioning is sometimes used in circuit design and synthesis. One algorithm for cell placement is min-cut placement in which cells are partitioned so that the inter-partition connections are minimized. These kinds of techniques are called *topology-based* techniques since they consider only the connectivity of vertices in deciding how to perform partitioning. However objects also have geometric constraints such as boundary and distance constraints. This information is not considered in topology-based partitioning methods, which may result in poor partitioning of the circuit for purposes such as placement.

In this paper we describe the progress we have made in exploiting geometrical information to improve the quality of solutions for VLSI design problems such as placement. Our technique is based on incorporating geometry in a topology-based multilevel hypergraph partitioning algorithm.

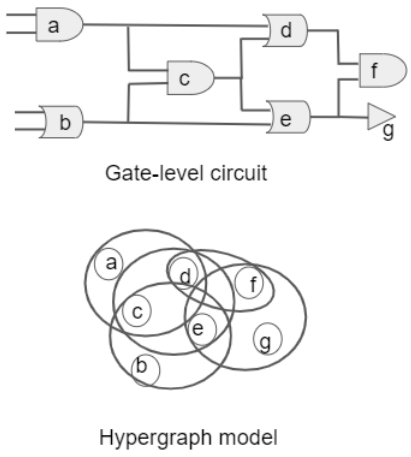

Fig. 1: A gate-level circuit and its hypergraph model

II. Hypergraph Partitioning Methods

A. Topology-based Partitioning

Topology-based partitioning algorithms are based on using the topological information of a hypergraph to obtain the partitions. Breadth-First Search (BFS), Kernighan-Lin (KL) [2], and Fiduccia-Mattheyses [3] algorithms are examples of topology-based partitioners. These algorithms are more effective if they are implemented in *multilevel* hypergraph partitioners. Multilevel partitioning consists of three phases: *coarsening*, *initial partitioning*, and *refinement*. In the coarsening phase, the size of the hypergraph is reduced by repeatedly merging vertices that should be assigned to the same partition. This process is stopped when the size of the coarsened hypergraph is small enough or has reached some other termination criterion. A topology-based partitioning algorithm such as BFS, KL or FM is then run on the coarsest hypergraph. After this initial partitioning, an iterative improvement algorithm such as KL or FM is used to improve the quality of partitioned hypergraph.

In every phase of the multilevel partitioning strategy, we use only the topology of the hypergraph and ignore other information associated with the actual cells in the circuit being partitioned. For example, during the refinement phase, nodes

at the boundary of a partition may be moved to a different partition to improve the edge cut. While several nodes may yield the same improvement in the edge cut, one of them may be preferred over the others for placement since it may permit more effective exploitation of design hierarchies or be more amenable to handling distance constraints.

B. Geometry-based Partitioning

In geometry-based partitioning of hypergraphs, each node has a position in some *d*-dimensional space. This is equivalent to assigning a vector in $\mathcal{R}^d$ to each node. The advantage of having geometry is that we can compute distances between nodes, and use the geometric notion of distance to perform fast partitioning. For example the K-Nearest-Neighbors (KNN) algorithm [4], a standard clustering algorithm in machine learning, can be used to create the partitions. However, techniques like KNN by themselves do not produce partitions with good edge-cuts since they consider only the positions of nodes when performing the partitioning, and do not take the topology of the hypergraph into account.

III. Embedding techniques

When geometry is not available for a hypergraph, *embedding techniques* can often be used to assign positions to nodes. Sorting or geometry-based partitioning techniques can then be used to create the partitions.

The classical examples of embedding techniques are *spectral methods* [5]. In spectral methods, the graph or hypergraph is represented as an adjacency matrix, and some of the eigenvectors of the Laplacian of this matrix are computed. The simplest of these techniques computes the *Fiedler vector*, which is the eigenvector corresponding to the second smallest eigenvalue of this matrix [6]. The signs of the entries in this vector are used to assign nodes to partitions (intuitively, this is a way of using sorting to create partitions once the geometry is available). The Fiedler approach computes an embedding of nodes in $\mathcal{R}^1$. By computing additional eigenvectors, it is possible to create embeddings in higher-dimensional spaces.

While spectral partitioning often produces good quality partitions, they are very expensive and not practical for large hypergraphs. For this reason, they languished for about two decades after an intense period of development in the 80's and 90's.

In the past five years, the machine learning community has invented a new class of embedding techniques that are used heavily in *vector-space models* for text and speech recognition. In particular, techniques for embedding graphs have been developed, and these techniques are faster than spectral methods while producing competitive results. Most of these techniques are based on DeepWalk [7] and node2vec [8]. These techniques use a simple layer neural network to process nodes of a graph. The input to this neural network is a set of random walks on the graph and the output is the embedding of the nodes in the graph. In this paper, we adopt a similar approach to find the embedding of the nodes in the hypergraph.

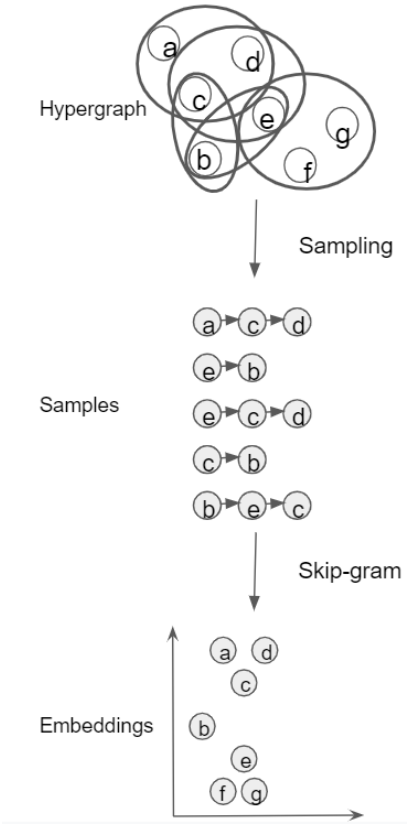

Fig. 2: Embedding process for hypergraphs

A. Mapping hypergraph topology to geometry

Figure 2 shows an illustration of our method. We use a similar approach to graph embedding methods such as node2vec [8]. We compute the vector representation of nodes in a hypergraph using the Skip-Gram neural network architecture. In this model, we first generate samples by doing a random walk on a hypergraph. The purpose of the random walk is to sample one hop and two hop neighborhoods. One hop neighborhood samples hyperedges and nodes that belong to that hyperedges while two hop neighborhood samples neighboring nodes that have a hyperedge in common. Then, we train this model by maximizing its log-likelihood on the training set.

$$J_{M_L} = \log P(v_t|u)$$
$$= score(v_t, u) - log(\sum_{node\ v\prime\ in\ hypergraph} exp\{score(v\prime, u)\})$$

Here, $score(v, u)$ is the dot product of node u and v. After the training is finished, every node in the hypergraph will end up having a vector associated to it that represents the position of that node in the metric space. Next we discuss how to use this geometric representation of nodes to improve the quality of our partitioner.

IV. Embedding-based hypergraph partitioning

To improve the partitioning quality, we use the embedding of nodes along with a multi-level hypergraph partitioning

algorithm. This extra embedding information can be used in all three phases of the multi-level scheme, namely, Coarsening, Initial Partitioning and Refinement. We now give a brief description on how we use the node embeddings in all these phases.

Algorithm 1 Node-Matching Algorithm

Input: Hypergraph $H = (V, E)$, $emb(V)$: embedding vectors for nodes in V;

1: **for all** $v \in V$ *in parallel* **do**
2: **if** v is not *matched* **then**
3: **for all** $u \in Neighbor(v)$ **do**
4: $dist(v, u) \longleftarrow distance(emb(v), emb(u))$
5: **end for**
6: $w \longleftarrow$ neighbor of v with minimum $dist$ value (break ties using id values).
7: Match v to node w.
8: **end if**
9: **end for**

In the Coarsening phase, we first use the embeddings to find node-matchings and then we coarsen/merge the nodes that are matched together. Algorithm 1 describes the pseudocode of our node-matching algorithm. In Steps 3-5, node v computes the distance values to each of its neighbors and then matches itself to the closest node among its neighbors (Step 7). After a matching is obtained, the coarser graph is obtained after coarsening/merging the nodes that are matched together.

Similar approach has been used in [9] to improve the coarsening phase of a multi-level hypergraph partitioning. However, their work does not explore whether the remaining two phases, initial partitioning and refinement, can also be improved using these embeddings. In this work, we explore this avenue by incorporating the embedding information in both initial partitioning and refinement.

For the initial partitioning, we can run a geometry-based partitioning such as spectral partitioning [5], since the initial prtitioning is only done for coarsest graphs. This gives us a bipartition of the coarsest graph.

Algorithm 2 Refinement Algorithm

Input: Hypergraph $H = (V, E)$, $emb(V)$: embedding vectors for nodes in V, Partitions P_0, P_1

1: Compute coordinates of centroids of partitions P_0 and P_1.
2: $L_0 \longleftarrow$ nodes in P_0 that are closer to centroid of P_1
3: $L_1 \longleftarrow$ nodes in P_1 that are closer to centroid of P_0
4: $l_{min} \longleftarrow \min(|L_0|, |L_1|)$
5: Swap l_{min} nodes between partitions P_0 and P_1, with preference given to the nodes that are farther away from the centroid of the current partition.

Algorithm 2 gives the pseudocode of one of the approaches that we would like to explore for the refinement phase. In Step 1 we compute the coordinates of the centroids of both partitions. Steps 2-3 compute sets L_0 and L_1 such that L_0 contains the nodes from partition P_0 that are closer to the centroid of P_1 and L_1 contains the nodes from partition P_1 that are closer to the centroid of P_0. We can then swap l_{min} nodes between P_0 and P_1 such that priority is given to the nodes that are farther away from the centroid of the current partiton (Step 5).

Graph	Nodes	Hedges	Edges
Stanford	281,903	261,588	2,312,497
Xenon	157,464	157,464	7,733,376
IBM18	210,613	201,920	819,697
Webbase	1,000,005	1,000,005	3,105,536
Xyce	1,945,099	1,945,099	9,455,545

TABLE I: Benchmark Characteristics

Graphs	Topology	Topology + Embedding	Zoltan
Stanford	1,746	302	1,885
Xenon	8,157	3,672	3,407
IBM18	2669	2880	2472
Webbase	1,060	881	1,202
Xyce	1,164	2,558	426

TABLE II: Performance comaprison with Zoltan for 2-way partitioning using 14 threads

V. Experiments

To do an initial test of our approach, we implemented a topology based multi-level hypergraph partitioner that can also leverage the obtained embedding information. We ran our paritioner on five hypergraphs listed in Table I. All hypergraphs except Xyce and IBM18 are obtained from the University of Florida Sparse Matrix Collection [10]. IBM18 is a netlist from ISPD98, while VLSI Circuit Benchmark Suite and Xyce are netlists from Sandia Laboratories [11].

In Table II, we compare our results with the state-of-the-art partitioner Zoltan [11]. For all inputs except Xyce, we observe that the embedding information does help in improving the edgecut. In comparison to Zoltan, our embedding implementation performs better on Stanford and Webbase, and worse on the rest of the inputs. These initial experiments show promising results. The initial implementation of this code can be found here: https://github.com/Breakinbad/Galois-1/tree/master/lonestar/experimental/embedding

VI. Conclusion

We showed how embedding techniques can be used to obtain geometry from the topology of a circuit and how to exploit this information in a topology-based hypergraph partitioner. If geometry is available, we can use that information instead of calculating an embedding for the hypergraph. These techniques let us exploit both geometry and topology to partition circuits for placement and other applications.

REFERENCES

[1] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in vlsi domain," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 7, no. 1, pp. 69–79, Mar. 1999.

[2] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, Feb 1970.

[3] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *19th Design Automation Conference*, June 1982, pp. 175–181.

[4] T. Mitchell, *Machine Learning*, ser. McGraw-Hill International Editions. McGraw-Hill, 1997. [Online]. Available: https://books.google.com/books?id=EoYBngEACAAJ

[5] A. Pothen, H. Simon, and K.-P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *SIAM J. Matrix Anal. Appl.*, vol. 11, pp. 430–452, 1990.

[6] M. Fiedler, "Algebraic connectivity of graphs," *Czech. Math. J.*, vol. 23, pp. 298–305, 1973.

[7] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[8] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[9] J. Sybrandt, R. Shaydulin, and I. Safro, "Hypergraph partitioning with embeddings," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[10] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, p. 1, 2011.

[11] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek, "Parallel hypergraph partitioning for scientific computing." IEEE, 2006.