Extended Abstract: Convex Partitioning of Large-Scale Directed Graphs

Julien Herrmann[1] Bora Uçar[2] Kamer Kaya[3] Ümit V. Çatalyürek[1,4]

1 Introduction

Directed graphs play an important role in combinatorial scientific computing (CSC) due to their modelling power for algorithms, workflow execution and communication patterns. However, when modelling a CSC problem as a graph partitioning problem, the direction information is generally ignored.

There are several problems where the directionality is crucial: for instance, if out-of-core execution is possible within a task-based runtime system, a schedule needs also to minimize the data movement amount from slow storage to faster memory. A similar but finer grained problem arises when the data movement between the memory and cache is considered [1]. This problem is getting more and more important, since the emerging architectures' peak processing rate is increasing significantly faster than their memory bandwidth. Many widely-used scientific kernels today are memory-bound and the only solution for a better performance is reducing the amount of data movement between the memory and cache (hierarchy). Furthermore, in the future, it may be more logical to use the *data movement complexity* of an algorithm instead of its time complexity. Yet, the data movement complexity is not well characterized today: it depends on various kernel transformations and the architectural parameters such as the fast memory (registers/caches) capacity. A thorough understanding is important to reveal the possible performance improvements beyond the current compiler optimizations. Recently, computational directed acyclic graphs (CDAGs) are used to characterize the data movement complexity and dynamically analyze the data locality potential [1].

Definition 1.1. (CDAG) *A CDAG is a directed acyclic graph $G = (V, E)$ with no isolated vertices. V is called the operation set and the edges in E represent the dependencies among the operations.*

2 Preliminaries

A *convex partitioning* of a CDAG into components allows us to schedule and *atomically* execute a component at once without any interleaving by the other components. Let us first define the notion of convexity:

Definition 2.1. (Convex component) Given a directed graph G, a vertex set $S \subseteq G$ is defined as a convex component iff for any pair of vertices $u, v \in S$, the vertices in all the $u \rightsquigarrow v$ and $v \rightsquigarrow u$ paths also belong to S.

Definition 2.2. (Convex k-way partition) Given a directed graph $G = (V, E)$, a convex partition of G is a set of convex components $\{V_1, V_2, ..., V_k\}$ of G such that $\bigcup_{i=1}^{k} V_i = V$ and $V_i \cap V_j = \emptyset$ for any $1 \leq i < j \leq k$.

The state-of-the-art tools designed for partitioning undirected graphs cannot be used to generate convex partitions since the orientations of the edges can damage the convexity as Figure 1 shows. The generalized

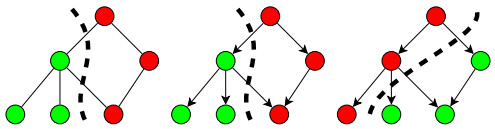

Figure 1: A valid 2-way partitioning of an undirected graph (left), an invalid/non-convex partitioning when edges are oriented (middle), a convex partitioning (right).

CDAG partitioning problem can be defined as follows with possible constraints and objectives in Table 1 where C is the fast memory capacity.

Definition 2.3. (CDAG partitioning problem) *Given a CDAG G, find a convex k-way partition $P = \{V_1, V_2, \cdots, V_k\}$ such that **Constraint**(P) is respected and **Objective**(P) is minimized.*

The partitioning constraints make the components fit to the fast memory. Hence, the operations' values within a component will stay in the fast memory until the operations are completed. Thus, no (extra) data movement is required for the edges inside the parts. Therefore, for CDAGs, Objectives 1 and 2 tend to minimize the total data flow to/from slow memory.

A good partition with a small data movement amount can reveal potential performance improvements for various kernels; for instance, Fauzia et al. used CDAG-based dynamic analysis to improve the Floyd-Warshall all-pairs shortest paths algorithm which was believed to be not possible [1]. Since the analysis

[1]Dept. Biomedical Informatics, The Ohio State University.
[2]CNRS and LIP, ENS Lyon, France.
[3]Sabancı University, Turkey.
[4]Dept. Elec. & Comp. Engineering, The Ohio State University.

needs to be dynamic, we need fast partitioning tools to generate convex partitions with good objective values.

Cons. 1	the total vertex weight is lower than a given C for all the components.
Cons. 2.	the total vertex weight of V_i and the other vertices with a successor in V_i is lower than a given C for every V_i.
Cons. 3	the maximum number of simultaneously live nodes for each component is less than C.
Obj. 1	minimize the edge cut between components.
Obj. 2	minimize the edge cut between components by counting the edges coming from the same node only once (data communication).
Obj. 3	increase the application performance.

Table 1: Practical constraints and objectives.

3 Directed Multilevel Graph Partitioning

We are developing a new directed multilevel graph partitioning tool. Multilevel frameworks became de-facto standard for solving graph and hypergraph partitioning problems efficiently, hence used by almost all of the current state-of-the-art partitioning tools. As all multilevel graph frameworks, our algorithm has three phases:

Coarsening: We obtain smaller acyclic graphs by combining the vertices until a minimum vertex count is reached or reduction on number of vertices is lower than a threshold. However, not all the vertices can be combined: consider a CDAG with three operations a, b, c and three edges (a, b), (b, c), (a, c). Here, the vertices a and c cannot be combined since we form a cycle. To protect convexity, we devised a novel and efficient mechanism to check if an edge is contractible or not based on a precomputed topological ordering of the vertices.

Initial Partitioning: After the graph is coarsened, we generate an initial convex partitioning. To do that, we first topologically order the vertices of this final graph. Then we used Kernighan's algorithm [2] to generate an optimal partition based on this topological ordering. Kernighan's algorithm has a linear execution time in the number of edges (assuming a constant part size). Since the number of edges in the coarsest graph is relatively small, we ran the algorithm multiple times with different topological orders.

Refinement/Uncoarsening: We project the initial solution to the finer graphs and refine it iteratively until a solution for the original graph is obtained. We used an FM-like, move-based direct k-way refinement algorithm, which does multiple passes. In each pass, all vertex gains are computed and current best moves are tentatively realized. Some negative gains moves are allowed to avoid sticking in local optima. In our refinement, to avoid creating cycles among partitions, each vertex can only be moved to one of two "adjacent" parts in the topological order.

4 Experimental Evaluation

We used three kernels for the experiments: multiplication of two 30×30 matrices; 100 iterations of 1D-Jacobi on a vector of size 100; and 30 iterations of 2D-Jacobi on a 30×30 grid. Some properties of the CDAGs for these kernels are summarized in Table 2.

Graphs	2MM	1D-jacobi	2D-jacobi
# vertices	139,500	58,902	236,208
# edges	243,000	98,000	423,360
avg/min/max degree	3.48/1/32	3.33/1/100	3.59/1/30

Table 2: The kernels used in the experiments.

Table 3 shows the partitioning results for the graphs in Table 2. We experimented with $k = \{2, 8, 32\}$ parts, 3% imbalance ratio, and measured the total edge cut (Obj. 1) and total communication volume (Obj. 2) that we try to minimize. dMLGP is our directed multilevel graph partitioner, Kernighan is the algorithm from [2] and Metis is the well-known graph partitioner. dMLGP and Kernighan produces convex k-way partitions whereas Metis does not use the direction information.

k	Kernighan		dMLGP		Metis	
	EdgeCut	TotVol	EdgeCut	TotVol	EdgeCut	TotVol
2MM						
2	94,397	31,784	28,735	3,442	34,083	4,622
16	151,921	82,493	83,595	49,757	81,858	33,107
32	167,458	105,408	100,306	83,760	90,514	43,972
1D-Jacobi						
2	1,128	572	1,131	566	797	375
8	5,132	2,847	5,286	2,895	3,874	2,317
32	18,423	10,549	18,252	10,288	8,616	5,476
2D-Jacobi						
2	14,049	4,523	14,195	4,531	9,841	3,380
8	58,509	23,389	61,950	23,123	36,463	17,098
32	169,570	83,281	175,454	80,569	70,777	35,683

Table 3: The results for 2MM, 1D-Jacobi and 2D-Jacobi. For the 2MM, theoretical lower bounds for the total volume is 205, 578 and 818, respectively for $k = 2$, 8 and 32.

Table 3 shows that the multilevel approach can work much better than Kernighan's algorithm on the 2MM kernel. As the Jacobi-2D results reveal, a better edge cut does not always imply a better communication volume. We are currently working on recursive-bisection, and improving k-way refinement, since we believe k-way refinement is the bottleneck in achieving high quality partitioning.

References

[1] N. Fauzia, V. Elango, M. Ravishankar, J. Ramanujam, F. Rastello, A. Rountev, L.-N. Pouchet, and P. Sadayappan. Beyond reuse distance analysis: Dynamic analysis for characterization of data locality potential. *ACM Trans. Archit. Code Optim.*, 10(4):53:1–53:29, Dec. 2013.

[2] B. W. Kernighan. Optimal sequential partitions of graphs. *J. ACM*, 18(1):34–40, Jan. 1971.