Hypergraph Decomposition of Sparse Matrices for Vector-Matrix Multiplication

S. Dadizadeh

{*sara88*}*@cs.ubc.ca*
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada

Abstract

We explore a parallel method of performing sparse matrix multiplication, using hypergraph decomposition to obtain a partitioning of the matrix onto processors.

Keywords: hypergraph partitioning, sparse matrix multiplication, matrix-vector multiplication

1 Introduction

Sparse matrix multiplication is one of the most heavily used operations in scientific computing, especially in applications which solve partial differential equations. It is imperative to have scalable matrix operations which perform these computations as efficiently as possible. To achieve the best speed-up possible, we must minimize communication overhead since there are few computations relative to the size of the matrix. Our approach is to model the matrix as a hypergraph and use the MLFM technique to split the hypergraph into partitions, where we assign a partition to each processor [1]. The problem of finding an optimal partition is NP-hard, but because partitioning is critical in several applications, heursitic algorithms with near-linear runtime were developed [2].

2 Strategy

We first need to represent the matrix A as a hypergraph; the model is given in section 2.1. We then use the multilevel Fiduccia-Mattheyses (MLFM) framework to perform the hypergraph partitioning, the algorithm is described in 2.2. In section 3 we delve into implementation details. In 3.1 we give an overview of hMetis, a library that implements the MLFM partitioning algorithm. Finally in 3.2 we describe the implementation details in MPI.

2.1 Hypergraph Representation

Our goal is a sparse-matrix vector product of the form $\vec{y} = A\vec{x}$, where $\vec{y}$ and $\vec{x}$ are dense vectors, and A is a sparse matrix. Matrix A is represented as the hypergraph $H_R(V_R, N_C)$. The vertex and net sets V_R and N_C correspond to the rows and columns of A, respectively. The vertices in a net n_j are called its pins and denoted as $pins[n_j]$. There exist one vertex v_i and one net n_j for each row i and column j, respectively. Net n_j contains the vertices corresponding to the columns which have a nonzero entry on row j ($v_i \in n_j$ if and only if $a_{ij} \neq 0$). Given this representation, we build a hypergraph H, where the k-way partitioning of H assigns vertices of H to k disjoint nonempty partitions. In a partition Π of H, a net that has at least one pin is said to connect that part. The connectivity set Λ_j of a net n_j denotes the number of parts connected by n_j. A net n_j is cut if it connects more than one part ($\Lambda_j > 1$), and uncut otherwise ($\Lambda_j = 1$). The hypergraph partitioning problem is the task of dividing a hypergraph into two or more parts such that the cutsize is minimized, since cuts represent interprocessor communication.

This decomposition scheme, where we represent rows of A as vertices, and columns of A as hyperedges, is called the column-net model for rowwise decomposition. The nets of H_R represent the dependency relations of the atomic tasks on the $\vec{x}$-vector components. Each net n_j incurs the computation $y_i = y_i + a_{ij}x_j$ for each vertex $v_i \in n_j$. This means that each net n_j denotes the set of atomic tasks (vertices) that need x_j. Figure 1 illustrates this dependency relation [2].

2.2 Hypergraph Partitioning Framework

The choice of partitioning algorithm depends on the number of movable objects (i.e. the number of vertices). The multilevel Fiduccia-Mattheyses (MLFM) framework scales well with large numbers of vertices, and creates the best known partitioning results. It consists of three components: clustering, top-level partitioning, and refinement or "uncoarsening". During

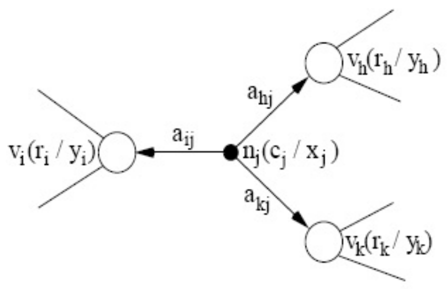


Figure 1: Dependency relation view of the column-net model

the clustering stage, vertices are combined into clusters based on connectivity, leading to a smaller, clustered hypergraph. Then the smallest (i.e. top-level) hypergraph is partitioned with a fast initial solution generator, and iteratively improved, using the Fidduccia-Mattheyses (FM) partitioning framework (which is a single-level version of the algorithm we use here). During refinement, solutions are projected from one level to the next and iteratively improved, once again by the FM algorithm.

3 Details of the Implementation

First we build a hypergraph representation of the matrix A in the format hMetis specifies. We obtain a partitioning from hMetis, and build a structure that contains the data a processor needs to compute its y_i results. Since this data is built dynamically, the root first sends the size, so each process knows how much data to expect. Once the computation is complete, each process sends its y_i results back to the root, where the root aggregates the data to form the result vector $\vec{y}$.

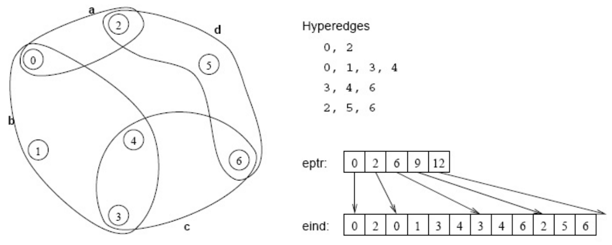


Figure 2: The `eptr` and `eind` arrays which describe the hyperedges of the hypergraph.

3.1 Overview of hMetis

hMetis is a hypergraph partitioning package that produces high quality partitions in reasonable time. The main function we are concerned with is `HMETIS_PartKway (int nvtxs, int nhedges, int *vwgts, int *eptr, int *eind, int *hewgts, int nparts, int ubfactor, int *options, int *part, int *edgecut)`. `nvtxs` and `nhedges` are the number of vertices and hyperedges, respectively. The parameters `eptr` and `eind` are two arrays which describe the hyperedges in the graph. `eptr` is of size `nhedges`+1, and is used to index into the second array `eind` that stores the actual hyperedges. Each hyperedge is stored as a sequence of the vertices that it spans, in consecutive locations in `eind`. `nparts` is the number of desired partitions. Figure 2 illustrates a simple hypergraph with its corresponding `eptr` and `eind` arrays. `part` returns the computed partition as an array of `nvtxs` size. Specifically, `part[i]` contains the partition number in which vertex i belongs to [3] [4].

3.2 Communication and Computation

In the column-net model, a partition Λ of H_R with $v_i \in P_k$ assigns row i, y_i and x_i to processor P_k for rowwise decomposition. A cut net n_j indicates that the processor who contains the corresponding vertex v_j should send its local x_j to all the processors in the connectivity set Λ_j of net n_j except itself. For example, in Figure 3, P_1 should send x_5 to both P_2 and P_3. Each net n_j incurs the computation $y_i = y_i + a_{ij}x_j$ for each vertex $v_i \in n_j$. Each processor P_k is responsible for computing y_i for the vertices (rows) that it has been assigned.

However, for a processor to know that a net is cut, it must have references to the eptr/eind arrays, as well as the partitioning returned from hMetis. Communicating that much data is infeasible, and the hMetis call is nondeterministic so each processor cannot do the computation locally. Instead, when the root process builds the sparse matrix representation of the data that it needs to distribute to each processor, it appends at the end each x_j value that the corresponding processor will need to perform its computation.

3.3 Using this Library

The main function in this library is `matrix_multiply(int argc, char** argv, int ubfactor, FILE* matrix_fp, FILE* vector_fp, FILE* result_fp)`. `argc` and `argv` are passed from the command line, `ubfactor` is the unbalance factor, a value ranging from 1 to 49 denoting the amount of unbalance tolerated in the partition (described in [3]). `matrix_fp` and `vector_fp` are file descriptors, pointing to files which have already been opened for reading. `result_fp` is a file descriptor specifying where to write the result vector $\vec{y}$, and it should already be open for reading. `finalize()` must be called after `matrix_multiply()`, before the program exits.

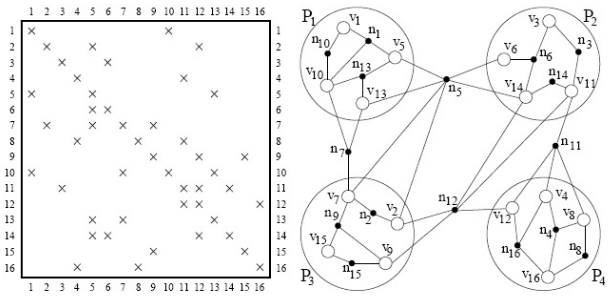
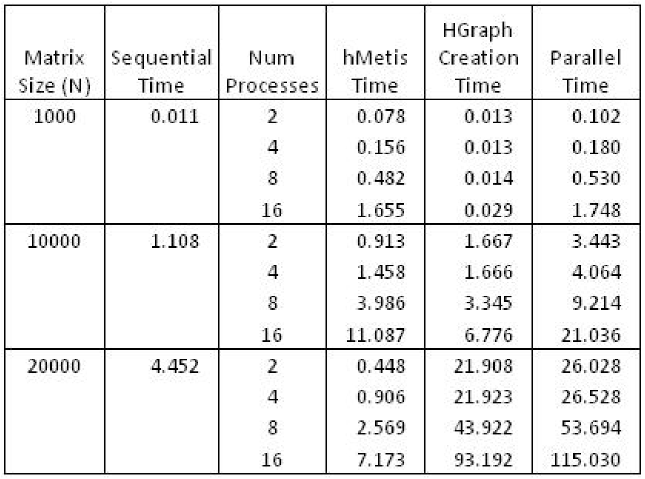


Figure 3: A 16x16 sparse matrix A and its column-net representation in a 4-way partitioning



Matrix Size (N)	Sequential Time	Num Processes	hMetis Time	HGraph Creation Time	Parallel Time
1000	0.011	2	0.078	0.013	0.102
		4	0.156	0.013	0.180
		8	0.482	0.014	0.530
		16	1.655	0.029	1.748
10000	1.108	2	0.913	1.667	3.443
		4	1.458	1.666	4.064
		8	3.986	3.345	9.214
		16	11.087	6.776	21.036
20000	4.452	2	0.448	21.908	26.028
		4	0.906	21.923	26.528
		8	2.569	43.922	53.694
		16	7.173	93.192	115.030

Figure 4: Benchmarks on a 1.60GHz, 2x Xeon duo core with 8GB memory. All timing values are in seconds.

4 Performance

Figure 4 outlines the sequential and parallel run times on varying problem sizes. A matrix size of N denotes a matrix with N*N elements. The parallel version runs much slower than sequential. It is evident that the parallel runtime is dominated by the call to hMetis and the hypergraph creation time. On a problem size of N=10,000, the hMetis call takes 11 seconds, while the sequential version runs in approximately 1 second. Note that the hypergraph creation time can be eliminated by requiring that the matrix be inputted in hypergraph form.

Since we are employing the column-net model, and need to read the matrix in column-wise, we chose to read the matrix into memory instead of performing many seeks through a large file. This was a poor decision, as very large matrices can be held in memory in sparse matrix format, while there is not enough memory to contain the entire matrix. Because of this issue, there were not enough resources to test this program on large enough matrices where there may have been potential speedup.

5 Conclusions

Although this implementation did not yield desirable results, hypergraph decomposition has been shown to find partitions which deliver significantly better performance than simply distributing sets of rows or columns onto processors. Given a few adjustments to the design and implementation of this program, we are confident that this method will result in a high performance sparse matrix multiplier.

References

[1] Scott A. Hutchinson, Ray S. Tuminaro, Ray S. Tuminaro, John N. Shadid, and John N. Shadid. Parallel sparse matrix-vector multiply software for matrices with data locality, 1995.

[2] D.A. Papa and I.L. Markov. *Hypergraph Partitioning and Clustering.* CRC Press, 2007.

[3] Kumar Karypis. hmetis: A hypergraph partitioning package. 1998.

[4] Umit V. Catalyurek and Cevdet Aykanat. Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. on Parallel and Distributed Computing*, 10:673–693.