A global, distributed ordering library

Cleve Ashcraft* François-Henry Rouet*

Introduction

Computing a fill-reducing ordering of a sparse matrix is a critical step for sparse direct solvers. In the early days of the field, "bottom-up" or "greedy" methods were popular. These include the Minimum Degree algorithm [7] and its variants, and profile-reducing methods such as the Cuthill-McKee algorithm [2]. More recently, "top-down" or "divide-and-conquer" methods such as Nested Dissection [3] have been favored, in particular for problems arising from the discretization of PDEs on 3D domains. The idea is, given the adjacency graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of the matrix, where $\mathcal{V}$ are vertices and $\mathcal{E}$ are edges, to partition the vertices $\mathcal{V} = \mathcal{B} \sqcup \mathcal{S} \sqcup \mathcal{W}$ where $\mathcal{S}$ separates $\mathcal{B}$ from $\mathcal{W}$; $\mathcal{S}$ is called a *vertex separator*. Then, recursively, $\mathcal{B}$ and $\mathcal{W}$ are partitioned similarly. Most practical implementations, especially in distributed memory environments, rely on the *multilevel* framework, in which the graph is *coarsened* in order to find separators. ParMETIS [6] and PT-Scotch [1] are widely used graph partitioning packages that rely on the multilevel framework.

The approach we propose here is a top-down algorithm, but it is not multilevel. Instead, we perform all the operations on the whole, uncoarsened, graph. We refer to this as a "global" method. In the next sections we describe briefly our algorithm and our distributed-memory implementation.

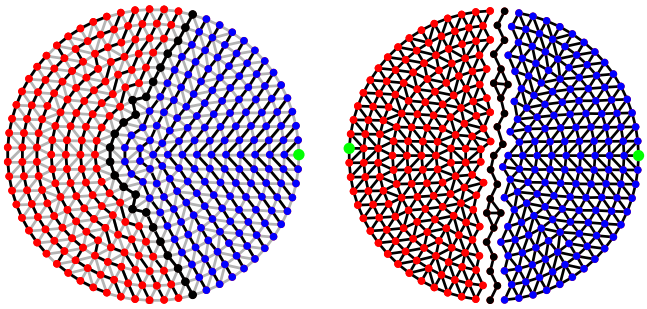
Finding separators

Given a source vertex s, one can compute $\phi(u) = \mathsf{dist}(s, u)$, the distance from s to u, for any vertex u in the graph, by performing a breadth-first search of the graph starting from s. For any $k \in [0, \max(\phi)]$, $\phi^{-1}(k)$, the set of vertices at distance k from u, is called a *level set*. The Cuthill-McKee algorithm reorders the matrix as $\{\phi^{-1}(0), \phi^{-1}(1), \ldots\}$, i.e., it makes the matrix block tridiagonal by ordering level sets one after another. But level sets can also be used to find separators. Indeed, for any k, $S = \phi^{-1}(k)$ is a minimal separator of the graph. This idea was used by George and Liu in their *automatic nested dissection* [4]. Given a source, the best separator induced by that source can be found by evaluating a cost function for all k, such as:

$$\mathsf{cost}(\mathcal{B}, \mathcal{S}, \mathcal{W}) = \begin{cases} +\infty \textbf{ if } \frac{\max(|\mathcal{B}|,|\mathcal{W}|)}{\min(|\mathcal{B}|,|\mathcal{W}|)} > \alpha \\ |S| \left(1 + \beta \frac{||\mathcal{B}|-|\mathcal{W}||}{|\mathcal{B}|+|\mathcal{S}|+|\mathcal{W}|}\right) \textbf{ otherwise} \end{cases}$$

α is a bound on the imbalance we will tolerate, and β is a penalty factor for imbalance. Figure 1a shows the level sets and the best separator built from those level sets for a triangulated circular grid.

In our algorithm, we use *half-level sets*, that are built from two sources instead of one. Consider two vertices s and t, and $\psi(u) = \mathsf{dist}(s, u) - \mathsf{dist}(t, u)$. For any $k \in [0, \max(\psi)[$, $S = \psi^{-1}(k) \sqcup \psi^{-1}(k+1)$, the union of two consecutive $half - levelsets$, is a separator of the graph. This is because the permutation $\{\psi^{-1}(0), \psi^{-1}(1), \ldots\}$ makes the matrix pentadiagonal. Given two sources s and t, we can find the best separator $\psi^{-1}(k) \sqcup \psi^{-1}(k+1)$ by evaluating a cost function such as the one above. We found empirically that the separators induced by half-level sets are more "straight" or "planar" than the curved separators induced by level sets, as shown in Figure 1b.



(a) Level sets. (b) Half-level sets.

Figure 1: Level set and half-level set partitionings.

The most computationally intensive operation in this algorithm is to perform a breadth-first search from the sources s and t. This is not a very scalable operation. However, if we have n_J candidate sources, we can perform the n_J BFS simultaneously, at a cost not significantly higher than the cost of a single BFS, because the number of messages to be sent depends only on the diameter of the graph. Furthermore, once we have n_J sources, the number of *pairs* of sources that we

*Livermore Software Technology Corporation.

can use to build separators is quadratic in n_J, $\frac{(n_J-1)n_J}{2}$. This allows us to evaluate many candidate separators and is a key ingredient for the quality of our ordering. We will show in the presentation how we find a good set of sources.

Separators built using half-level sets are not minimal. We thus need an extra step to smooth the separator into a minimal separator. This can be done with a simple "trimming" method, or by solving a max flow problem to find minimum weight separators. We improve separators by performing multiple cycles in which the separators are first expanded before being trimmed.

Ordering algorithm and parallel implementation

At any step of the recursive bisection:

1. We detect quasi dense rows and we remove them from the subgraph. The corresponding vertices are numbered after the remainder of the subgraph in the output permutation.
2. We check whether the subgraph has multiple connected components. If it does, we redistribute them to different sets of processes (proportionally to their number of vertices). For components shared by multiple processes, return to step 1.
3. We find a separator $\mathcal{S}$ as described above.
4. We split the subgraph into $\mathcal{B}, \mathcal{S}, \mathcal{W}$, we redistribute $\mathcal{B}$ and $\mathcal{W}$ to disjoint sets of processes and we recurse (return to step 1) on $\mathcal{B}$ and $\mathcal{W}$.
5. We stop when no subgraph is shared among two or more processes.

We have a distributed-memory implementation, that we call LS-GPart, of the above algorithm. All the operations that we describe in the previous section are fully parallelized, and the data structures are fully distributed, i.e., there are no global (order-n) vectors. The input graph is distributed by vertices, as for most parallel codes, but we make no assumptions on the distribution. As described above, the graph is dissected until every processes is assigned a subdomain. Subdomains are then ordered using the serial version of Metis.

Experiments

We compare our library against the popular packages ParMETIS 4.0.3 and PT-Scotch 6.0.4. We measure the quality of the ordering by looking at the number of operations for the factorization of the matrix, and the number of nonzeros in the factors. In Table 1, we show results for `bmw7st_1`, a matrix arising from linear static analysis of a car body (U. Florida collection). We use 8 MPI processes; we perform 3 levels of recursive bisection so that every process inherits a subdomain, then subdomains are ordered serially.

Ordering	nnz(LL^T) ($\times 10^6$)	ops(LL^T) ($\times 10^9$)
LS-GPart	27.6	13.2
ParMETIS	27.9	13.2
PT-Scotch	29.6	16.3

Table 1: Matrix `bmw7st_1`, 8 domains.

In Table 2, we consider a regular cubic grid (with 128 points along each side), and this time we use 16 MPI processes. In both cases, one can observe that the quality of our ordering is slightly better than that of ParMETIS and PT-Scotch, in particular for the regular grid. This is interesting because regular problems are usually hard for partitioners; in our case, we perform as well as geometric Nested Dissection.

Ordering	nnz(LL^T) ($\times 10^6$)	ops(LL^T) ($\times 10^9$)
LS-GPart	317.7	3082.1
ParMETIS	333.8	3888.9
PT-Scotch	395.7	5256.2
Geometric	312.8	3129.3

Table 2: 128^3 regular grid, 16 domains.

More detailed results will be shown in the presentation, using matrices from the University of Florida sparse matrix collection, and matrices arising from physical simulations performed by LS-Dyna [5].

References

[1] C. Chevalier and F. Pellegrini, *PT-Scotch: A tool for efficient parallel graph ordering*, Parallel Computing, 34 (2008), pp. 318–331.

[2] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, Proceedings of the ACM 24th national conference, (1969) ,pp. 157–172.

[3] J. A. George, *Nested dissection of a regular finite element mesh*, SIAM Journal on Numerical Analysis, 10 (1973), pp. 345–363.

[4] J. A. George and J. W. H. Liu, *An automatic nested dissection algorithm for irregular finite element problems*, SIAM Journal on Numerical Analysis, 15 (1978), pp. 1053–1069.

[5] J. O. Hallquist et al., *LS-DYNA theory manual*, Livermore Software Technology Corporation (1992).

[6] G. Karypis and V. Kumar, *A parallel algorithm for multilevel graph partitioning and sparse matrix ordering*, Journal of Parallel and Distributed Computing, 48 (1998), pp. 71–95.

[7] W. F. Tinney and J. W. Walker, *Direct solution of sparse network equations by optimally ordered triangular factorization*, Proceedings of IEEE, 55 (1967), pp. 1801–1809.