Towards a Parallel Disk-Based Algorithm for Multilevel k-way Hypergraph Partitioning

Aleksandar Trifunovic William J. Knottenbelt

Department of Computing, Imperial College London
South Kensington Campus, London SW7 2AZ, UK
email: {at701,wjk}@doc.ic.ac.uk

Abstract

In this paper we present a disk-based parallel formulation of the multilevel k-way hypergraph partitioning algorithm. This algorithm provides the capability to partition very large hypergraphs that hitherto could not be partitioned since the memory required exceeds that available on a single workstation. The algorithm has three main phases: parallel coarsening, sequential partitioning of the coarsest hypergraph and parallel refinement. At each parallel coarsening and refinement step disk is used to minimise memory usage. We apply the algorithm to very large hypergraphs with $\Theta(10^7)$ vertices from the domain of performance modelling and show that the partitioning quality is approximately 20% better in terms of the $(k-1)$ metric than approximate partitionings produced by a state-of-the-art parallel graph partitioning tool.

1. Introduction

Hypergraph partitioning finds application in a number of areas including, but not limited to, VLSI circuit design [2] and decomposition for parallel sparse-matrix vector multiplication [8]. The partitioning problem usually requires the set of vertices of the hypergraph to be partitioned into k disjoint subsets such that the weight of each of the disjoint subsets does not differ from another by more than a prescribed amount. The weight of the set of vertices is expressed as the sum of the constituent vertex weights. The partitioning is carried out in such a way that a certain objective function over the hyperedges is minimized. In the case of the parallel decomposition of sparse matrices the objective function becomes the $(k-1)$ metric [8]. This is the sum, over all hyperedges that span different partitions, of hyperedge weight multiplied by the number of spanned partitions minus one. This is not to be confused with hyperedge *cut* which is the sum, over all hyperedges that span different partitions, of hyperedge weight. The value of the objective function for a particular partitioning is often referred to as the *cutsize* of the partitioning.

Computing the optimal bisection of a hypergraph under the hyperedge cut metric (and hence the $(k-1)$ metric since $k = 2$ for a bisection) is known to be NP-complete [14]. Thus, research has focused on developing polynomial time heuristic algorithms that give good sub-optimal solutions. Much work has been done on sequential algorithms and a survey relating to algorithms in VLSI design is presented in [2]. Recently, the most successful heuristic algorithms (in terms of cutsize and runtime) have been those based on the multilevel paradigm [17, 7]. Multilevel algorithms form a pipeline broadly consisting of three stages. During the coarsening stage a sequence of successively smaller hypergraphs is constructed by merging selected vertices. The initial partitioning stage then computes the partitioning of the smallest (coarsest) hypergraph in the sequence. Finally, during the uncoarsening stage this partitioning is projected through the sequence of hypergraphs constructed during the coarsening stage and is further refined after each projection.

Computing the k-way partitioning (where $k > 2$) can be done either via the recursive bisection method (divide-and-conquer) or by computing the the k-way partitioning directly. In [20] the authors experimentally show that the direct partitioning method can be superior both in terms of runtime and the $(k-1)$ metric to recursive bisection for larger values of k.

In [6, 9] hypergraph partitioning is applied to the parallel computation of response time densities in Markov and Semi-Markov chains. The partitioning is a pre-processing step to the parallel sparse-matrix vector type operations that form the kernel operations in such solvers. A good partition can greatly reduce the amount of interprocessor communication incurred, which is especially important when the ratio of network latency to processor speed is high (such as is the case in commodity workstation clusters). In particular for the response time density calculations the resulting partition is reused many times, making the quality of the partitioning algorithms key to the scalability of these algorithms. The sparse matrices that need to be partitioned may have upwards of $\Theta(10^7)$ rows (vertices) and $\Theta(10^8)$ non-zeros (pins) so that in these cases the problem becomes intractable by sequential computation. By comparison, the largest hypergraph in the current VLSI benchmark suite has just 184 752 vertices and 860 036 pins [1].

In the absence of a parallel hypergraph partitioning algorithm, parallel graph partitioning was used in [6, 9] to distribute the matrix across the processors. While superior to a random partition, graph partitioning schemes do not accurately reflect the actual communication cost incurred during parallel matrix vector multiplication but merely provide (and thus attempt to minimize) an upper bound [8]. In this paper we present a disk-based parallel formulation built upon a sequential multilevel k-way partitioning algorithm for hypergraphs [20]. The target architecture is a cluster of commodity workstations connected by a switched ethernet network. The algorithm is evaluated on a number of sparse hypergraphs arising from response time density computations and is shown to consistently outperform a leading parallel graph partitioning tool ParMeTiS [23] in terms of cutsize by up to 27%.

The rest of this paper is organized as follows. Section 2 describes sequential multilevel hypergraph partitioning in more detail. Section 3 presents the parallel formulation and Section 4 the experimental evaluation. Finally, concluding remarks and ideas on further research directions are presented in Section 5.

2. Sequential Multilevel Hypergraph Partitioning

As shown in Fig. 1, a hypergraph is an extension of a graph data structure in which edges connect arbitrary, non-empty sets of vertices. Formally, we define a hypergraph $H(V, E)$ as follows. V is defined to be the set of vertices and E the set of hyperedges, where each hyperedge $e_i \in E$ is a subset of the vertex set V. With each vertex $v_i \in V$ is associated a weight w_i, which in the case of a row-wise parallel sparse-matrix decomposition is given by the number of hyperedges incident on that vertex. This corresponds to the number of non-zeros in the row (i.e. the computational load that each row induces on a processor). Similarly, with each hyperedge $e_i \in E$ is associated a cost c_i. In the case of the parallel sparse-matrix decomposition, this is unity since it corresponds to the cost of communication a vector element across a processor boundary. The *size* of a hyperedge is defined as its cardinality.

The k-way partitioning problem is to find k disjoint sub-

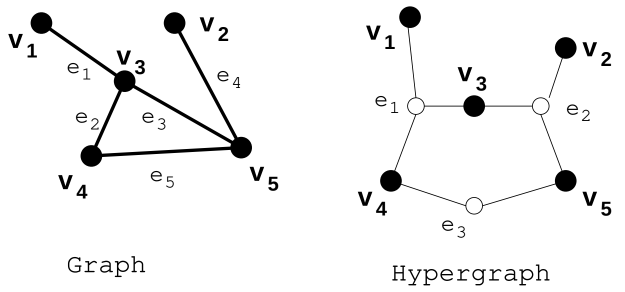

Figure 1. A graph and a hypergraph

sets V_i, $(i = 0, \ldots, k-1)$ of the vertex set V with corresponding weights W_i, $(i = 0, \ldots, k-1)$ such that, given a prescribed balance criterion $0 < \epsilon < 1$,

$$(1-\epsilon)W_{ave} < W_i < (1+\epsilon)W_{ave}$$

holds $\forall i = 0, \ldots, k-1$ and the objective function over the hyperedges is minimized. Here W_{ave} denotes the average partition weight. When the objective function is the $(k-1)$ metric, the partition cost (or cutsize) is given by

$$P_{cost} = \sum_{i=0}^{n-1} (\lambda_i - 1)c_i$$

where λ_i is the number of partitions spanned by hyperedge e_i. This formalizes the intuition of the parallel sparse-matrix decomposition that minimizes the communication cost subject to maintaining a computational load balance.

The multilevel paradigm is preferred to *flat* approaches (i.e. those that do not attempt to coarsen the hypergraph) because it scales well in terms of runtime and solution quality with increasing problem size. Iterative improvement algorithms are more likely to get trapped in poor local minima as problem size increases. Alternative approaches to obtaining good solutions such as spectral methods are reviewed in more detail in [2]. However, for large problem instances these are usually incorporated within the multilevel framework to preserve realistic runtimes [4].

The following subsections describe the multilevel pipeline in more detail.

2.1. The Coarsening Phase

The aim of the coarsening phase is to reduce the original problem instance via a succession of smaller hypergraphs that maintain as far as possible the structure of the original hypergraph.

Coarsening is performed by merging the vertices of the hypergraph together to form vertices of the coarse hypergraph. The corresponding hyperedges of the coarse hypergraph are constructed from the original set of hyperedges via the latter vertex mapping. Single vertex hyperedges in the coarse hypergraph are discarded as they cannot contribute to the cutsize of a partitioning of the coarse hypergraph. If more than one hyperedge maps onto the same hyperedge of the coarse hypergraph only one hyperedge is retained, with its cost set to be the sum of the costs of the hyperedges that mapped onto it.

It is desirable for the coarsening phase to maintain the natural clusters (highly connected vertices) in the original hypergraph as clusters of coarse vertices in the coarsened hypergraphs. In addition, the coarsening should reduce the size and number of hyperedges, as well as reducing the *exposed hyperedge cost*, which is defined as the sum of individual hyperedge costs and which represents the upper bound on the cutsize of a partitioning. The coarsening algorithm has significant impact on the final solution quality since most heuristic algorithms tend to terminate at local minima with respect to the heuristic. Thus, a poor coarsening algorithm may only allow a partitioning algorithm to explore parts of the solution space where the local minima solutions are of poor quality relative to the global minimum.

Coarsening algorithms are discussed in detail in both [2] and [17]. During our experiments we have found that the *FirstChoice* coarsening algorithm [20] and related algorithms [8] result in balanced partitionings and fast runtimes for our case study hypergraphs. The FirstChoice coarsening algorithm proceeds as follows. The vertices of the hypergraph are visited in a random order. For each vertex v_i, all vertices (both those already matched and those unmatched) that are connected via hyperedges incident on v_i are considered for matching with v_i. A connectedness metric is computed between pairs of vertices and the most strongly connected vertex to v_i is chosen for the matching, provided that the resultant cluster does not exceed a prescribed maximum weight. This condition is imposed to prevent a large imbalance in vertex weights in the coarsest hypergraphs. Note that more than two vertices may map to the same cluster in the coarse hypergraph. Another family of algorithms, known as *Hyperedge Coarsening* algorithms [17], seek a maximal independent set of hyperedges. The sets of vertices that belong to each of the hyperedges in the set are collapsed together to form vertices in the coarse hypergraph. In order to find the maximal independent set, the hyperedges are sorted in decreasing order of hyperedge cost. Ties are broken in increasing order of hyperedge size. The hyperedges are now visited in this prescribed order and for each hyperedge that consists of solely unmatched vertices, its vertices are mapped to a single cluster in the coarse hypergraph. The remaining vertices may then be mapped as singleton clusters in the coarse hypergraph or the hyperedges may once again be visited in the above order and groups belonging to the same hyperedges may be mapped to the same clusters. In our experiments, Hyperedge Coarsening often led to less tightly balanced partitionings while not resulting in an improvement in cutsize over FirstChoice coarsening.

An important parameter to the coarsening algorithm is the ratio of the number of vertices in successive coarser hypergraphs. A low ratio implies that many coarsening stages may be required, thus increasing the runtime of the overall algorithm. On the other hand, a larger ratio may result in a poorer quality of coarsening as vertices are matched into sub-standard clusters. In [17] the author reports that a ratio in the range 1.5–1.8 provides a reasonable balance between runtime and solution quality. Our experience is similar with our case study hypergraphs, using ratios in the range 1.5–2.0.

2.2. The Initial Partitioning Phase

The initial partitioning phase provides a partitioning of the coarsest hypergraph. This will be subsequently refined as it is being projected through the sequence of successively finer hypergraphs. Because the coarsest hypergraph tends to be significantly smaller than the original problem instance, the time taken to compute the initial partitioning is usually considerably less than the time taken by the other phases of the multilevel pipeline.

In [20] the authors use recursive bisection to compute the initial k-way partitioning. We also adopt this approach but note that it is possible to use a direct k-way method here to produce a partitioning of similar quality in a similar order of time since the coarsest hypergraph is very small (of the order of a few hundred nodes). Bisection is typically performed using the *Greedy Growing Algorithm* [8, 22]. This algorithm begins with a randomly selected vertex and grows a single partition around it by assigning the most highly connected vertex to the partition from the remaining vertices until the desired partition size is achieved. Because the algorithm is randomized, a number of initial partitionings is computed and the best is retained for the uncoarsening phase.

2.3. The Uncoarsening Phase

Here we propagate the initial partitioning back through the successively finer hypergraphs and at each step further refine the partitioning using heuristic refinement techniques. When the overall k-way partitioning is computed via recursive bisection, the refinement phase consists of a bisection refinement algorithm. Traditionally, iterative improvement algorithms based on the Fiduccia-Mattheyses (FM) algorithm are used. These perform *passes*, during each of which each vertex is moved from its starting partition at most once; the best sequence of moves found by the heuristic is actually performed leading to the refined partitioning. The algorithms operate in $O(p)$ time per pass, where p is the number of pins in the hypergraph, and usually converge within a few passes [13] to local minima with respect to the heuristic used. More sophisticated refinement algorithms have been developed, motivated by the idea of escaping from poor local minima [25, 10, 11, 12].

Extending the FM-algorithm to compute a k-way partitioning at each refinement step (as opposed to using recursive bisection) increases both the time complexity of the algorithm and the likelihood that the algorithm terminates at a relatively poor local minimum [20]. However, good results have been reported with a *greedy refinement* algorithm, especially for increasing values of k [20]. The greedy refinement algorithm performs iterations during which the vertices are visited in a random order and moved to the partition that results in the largest positive gain. Since the hypergraph is sparse, the algorithm avoids calculating the gain to every other of the $(k-1)$ partitions as follows. Vertices are not considered for a move if they are internal to their current partition (i.e. all adjacent vertices are also in the same partition). Otherwise, the gain of a move is only computed for neighbouring partitions (those partitions that contain adjacent vertices) if the move to the neighbouring partition does not violate the balance constraint. Experiments have showed that the algorithm typically converges after a small number of iterations [20].

A more sophisticated refinement scheme repeats the whole coarsening and refinement process to the refined partitioning while preserving properties of the partitioning during the coarsening phase. This type of refinement is called a *V-Cycle* [17, 7] and is a feature of the tool *k*hMeTiS. It attempts to converge to a better solution than would be obtained by simply performing the multilevel pipeline once, but can significantly increase runtime.

3. Parallel Multilevel Hypergraph Partitioning

This section describes the main contribution of this paper, namely the parallel formulation of an existing sequential multilevel k-way partitioning algorithm. The algorithms that make up the multilevel pipeline are inherently sequential in nature, making it difficult to easily find opportunities for concurrency. Moreover, whereas successful parallel formulations for graph partitioning exist [18, 21, 3], none have yet been forthcoming for hypergraph partitioning. It is worth noting the main difference between graphs and hypergraphs: whereas the cardinality of every edge in a graph is two, the cardinality of a hyperedge in a hypergraph can vary from 1 to an upper bound given by the number of vertices. It has been shown that, in general, there does not exist a graph model that correctly represents the cut properties of the corresponding hypergraph [16]. Thus, it does not seem possible to directly apply the parallel graph partitioning algorithms to hypergraphs although they may give

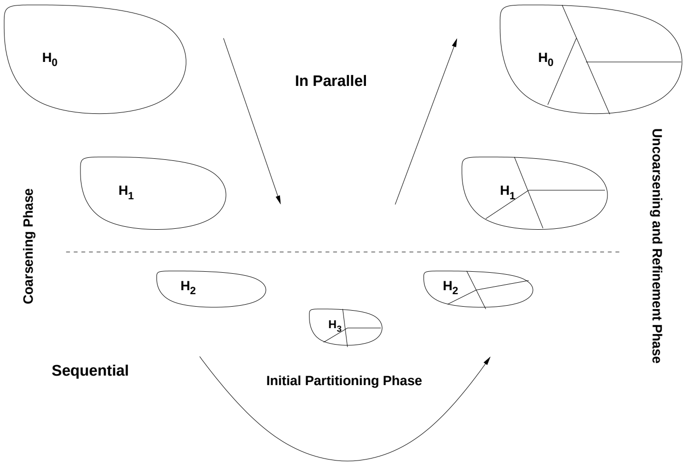

Figure 2. Parallel Multilevel Pipeline

reasonable approximations in some cases.

In the absence of obvious fine-grained parallelism, a coarse-grained formulation is sought. Note that, during the initial partitioning stage (i.e. after coarsening the original hypergraph), the problem instance should be small enough for sequential computation on a single processing unit. Hence, assuming that multiple runs of the sequential algorithm are required, they can be carried out on the available processors in parallel. The parallel multilevel pipeline is illustrated in Fig. 2. The sections below describe the parallel coarsening and refinement phases in more detail.

3.1. Parallel Coarsening Phase

The target architecture for our parallel formulation is a cluster of commodity PC worstations connected by switched 100 Mbps ethernet. Given the relatively high latency of the interconnect, an important objective for the parallel formulation is to minimise the communication costs. This poses the question as to how the data should be distributed across the processors during coarsening.

The natural way to store a hypergraph across p processors would be to store $|E|/p$ hyperedges on each processor. However, this complicates the process of finding a match for a given vertex as required in sequential FirstChoice coarsening. Finding adjacent vertices may now involve communication with every processor since each processor may contain hyperedges incident on this vertex. Alternatively, we may decide to map vertices from the individual hyperedges to single clusters in the coarse hypergraph, as in hyperedge coarsening. But, unless the sets of hyperedges on individual processors are independent, this could not be done concurrently since each time we map vertices from a hyperedge to a single cluster, we need to make sure that one of these vertices is not being mapped by another hyperedge on another processor.

Instead, we exploit the known characteristics of the hypergraph problem instances. It is well known that transition matrices of Markov and semi-Markov chains generated by a breadth first search exhibit a particular structure of non-zero elements [24]. We analyse how this structure translates to the corresponding hypergraph by examining the behaviour of a simple (Edge Coarsening) vertex matching algorithm [20]. The algorithm visits the vertices in a random order and looks for maximal vertex matches subject to the maximum prescribed cluster weight (in the coarse hypergraph) with unmatched vertices. If no suitable matches exist, the vertex is matched as a singleton vertex. The algorithm terminates when the number of vertices in the coarse hypergraph has been reduced by a prescribed amount. The vertex set of the fine hypergraph is partitioned into a number of contiguous blocks such that, when a vertex matches with a vertex from its own block, we denote it as a local match and when it matches with a vertex from another block we denote it a remote match. Vertices that match as singletons are denoted singleton matches. Table 1 shows the aggregate percentages of the above matches over ten runs of the algorithm on hypergraph representations of transition matrices, whose characteristics can be found in the Appendix.

The tendency for vertices to match with their immediate neighbours can be exploited in the way the hypergraph is mapped onto the processors during the coarsening stage. A good coarsening may be achieved by restricting the candidate vertices to immediate neighbours of a vertex. We note that our algorithm may still be correctly applied to general hypergraphs that do not exhibit this tendency, albeit at a performance penalty. Processors are made responsible for a contiguous subset of the vertex set V. In addition, the processors only store locally those hyperedges that contain local vertices. This information is then sufficient to merge the vertices using the FirstChoice coarsening algorithm. Once the vector mapping the vertices to their coarse clusters is computed across the processors, the processors simply contract the local hyperedges using their portion of the mapping vector. The processors then communicate their portions of the mapping vector to the other $p-1$ processors in round-robin fashion to enable every processor to fully contract their locally held hyperedges in the knowledge that the resulting locally-stored coarse hyperedge set will be independent from coarse hyperedge sets stored elsewhere (achieved by using a hash function to assign ownership of hyperedges to particular processors). This completes a single stage of the coarsening phase in parallel. Once the hypergraph has shrunk by a sufficient amount (as measured by its number of pins) a fewer number of processors is used in the subsequent stages until the hypergraph is small enough to fit on a single processor. At this stage, the sequential multilevel pipeline can be used. Each intermediate coarsened hypergraph is stored on disk to make the algorithm memory-efficient. These hypergraphs will subsequently be used during the refinement phase.

3.2. Parallel Refinement Phase

It was noted in [20] that the greedy k-way refinement algorithm is inherently parallel. Our tool currently employs a simple implementation in which processors are responsible for k/p partitions and the associated hyperedges. They concurrently perform local refinement followed by round-robin communication of partitions to the remaining $p-1$ processors in order for refinement to capture possible vertex moves across each partition boundary. The difficulty in efficient parallelisation of the refinement stage is capturing the global nature of k-way refinement in parallel while preserving concurrency. The current implementation is far from ideal in that the opportunity for a vertex to move in the best direction from its original partition is lost if, during one of the local refinement stages, it moves in an inferior direction that nevertheless results in positive gain. Furthermore, the current implementation tends to work better as the desired number of partitions increases for a constant number of processors used. Thus we currently use the fewest number of processors necessary for a particular refinement stage. As during the parallel coarsening phase, disk storage is used in order to make the algorithm memory efficient.

4. Experimental Results

The parallel algorithm was experimentally evaluated on hypergraph representations of transition matrices from the Voting Model with 250 and 300 customers [5]. The main characteristics of these models are given in the Appendix. Both problem instances are too large to be partitioned on a single workstation so a suitable comparison was provided by the parallel graph partitioning tool ParMeTiS [23] that was previously used to compute approximate partitionings for very large transition matrix instances in [6, 9]. To convert the matrix into the appropriate inputs for our parallel hypergraph partitioner and for ParMeTiS, we use the transformations presented in [8]. The computational complexity of the latter transformations are the same, namely $O(m)$, where m is the number of non-zero elements in the matrix. Although the optimisation objectives for the two tools are different (as discussed in Section 3, ParMeTiS can only minimise an approximation to the true communication cost while our tool models it exactly), we compare with ParMeTiS because such state-of-the-art parallel graph partitioners are currently the only feasible way to reduce

Hypergraph	# partitions	% local	% remote	% singleton
Voting 100	2	90.1	0.9	9.0
Voting 100	4	88.1	2.9	9.0
Voting 100	8	84.4	6.7	8.9
Voting 100	16	77.2	13.9	8.9
Voting 100	32	62.8	28.2	9.0
Voting 125	2	90.4	0.7	8.9
Voting 125	4	88.8	2.3	8.9
Voting 125	8	85.9	5.2	8.9
Voting 125	16	79.9	11.2	8.9
Voting 125	32	68.6	22.5	8.9
Voting 150	2	91.5	0.7	7.8
Voting 150	4	90.2	2.0	7.8
Voting 150	8	87.5	4.7	7.8
Voting 150	16	82.4	9.8	7.8
Voting 150	32	72.4	19.8	7.8
Voting 175	2	91.6	0.6	7.8
Voting 175	4	90.5	1.7	7.8
Voting 175	8	88.2	4.0	7.8
Voting 175	16	83.8	8.4	7.8
Voting 175	32	75.1	17.1	7.8

Table 1. Vertex Connectivity Analysis

communication costs for very large matrices (albeit in an indirect manner).

Our experimental pipeline consists of three phases:

1. Parallel coarsening using our parallel formulation of the FirstChoice coarsening algorithm.

2. The coarsening algorithm interfaced with the state-of-the-art sequential multilevel partitioning tool *k*hMeTiS [19].

3. Finally, the output partition vector given by *k*hMeTiS was refined in parallel using the parallel k-way partitioning algorithm.

The parallel algorithm was implemented in the C++ language using the Message Passing Interface (MPI) standard [15]. The architecture used in the experiments consisted of a cluster of commodity PC workstations connected by switched 100 Mbps ethernet network. Each PC was equipped with a 2.8GHz Pentium(4) CPU and 1GB RAM. The results of the experiments are shown in Table 2, in which Par*k*way denotes the results attained by our parallel algorithm. The 250 customer model was partitioned using four processors while the 300 customer model was partitioned using eight processors. A balance constraint of 5% was used for all experiments meaning that no weight of any partition should differ from the average partition weight by more than 5%. A reduction factor of 2.0 was used during the parallel coarsening phase in our implementation while ParMeTiS used the default value for its multilevel algorithm.

The results indicate that our parallel hypergraph partitioning formulation significantly dominates the approximation given by parallel graph partitioning in terms of cutsize. On average, our algorithm produces $(k-1)$ cutsizes 20% lower than those produced by parallel graph partitioning on the voting model with 300 hundred customers and 16% lower on the smaller voting model with 250 customers. In turn, the parallel graph partitioning significantly dominates our disk-based algorithm in terms of runtime. While hypergraph partitioning should inherently take more time than graph partitioning, we believe that a possible reason for the poor runtime exhibited by the parallel hypergraph partitioning algorithm is due to both slow disk access time and disk contention. This may be because we have been using the disk storage on our departmental file server which experiences very high congestion.

5. Conclusion

We have devised a parallel formulation of the multilevel k-way hypergraph partitioning algorithm and have demonstrated its ability to partition very large hypergraphs with $\Theta(10^7)$ vertices by combining the memory and processing power of several workstations. To the best of our knowl-

No. of Partitions	Voting 250 model on 4 processors				Voting 300 model on 8 processors			
	Par*k*way		**ParMeTiS**		**Par*k*way**		**ParMeTiS**	
	(*k*-1) cut	time(s)	(*k*-1) cut	time(s)	(*k*-1) cut	time(s)	(*k*-1) cut	time(s)
8	91 511	1 309	117 354	25	-	-	-	-
16	182 206	1 393	249 415	27	322 737	4 827	442 387	85
32	354 561	1 495	402 681	32	529 763	4 762	687 659	61
64	525 856	1 777	610 597	33	874 652	5,007	1 033 312	80
total:	1 154 134	5 974	1 380 047	117	1 727 152	14 596	2 163 358	246

Table 2. Parallel Partitioning Results

edge this is the first time that hypergraphs of this size have been successfully partitioned, since previously no parallel hypergraph partitioners existed and these hypergraphs are too large to be partitioned in the memory of a single workstation. We have further demonstrated that the quality of the hypergraph partitionings produced by our parallel tool comfortably exceeds the approximate partitionings produced by existing parallel *graph* partitioning tools.

However, there are also some shortcomings in the current implementation which we will address as part of our future work. In particular we note that, while the extensive use of disk has minimised the amount of memory used, as well as reducing the number of communication operations required, it has also resulted in relatively poor runtimes. It is possible to significantly improve the parallel runtime by reducing the number of disk-based operations and by using a better hardware configuration that reduces contention for shared disks. We also note that partitioning runtime will not be a significant factor for many problem instances where a single partitioning may be reused several thousand times (e.g. in the parallel Laplace Transform-based response time analyser described in [6]).

Another focus of future reasearch is the development of a more efficient and flexible parallel formulation of the k-way refinement algorithm. Aside from potentially leading to inferior refinement quality in comparison to the sequential k-way refinement algorithm, the current implementation is restricted in that it can only compute partitionings where k is a non-negative integer power of two that is greater than the number of processors available.

References

[1] C. Alpert. The ISPD98 Circuit Benchmark Suite. In *Proc. International Symposium of Physical Design*, pages 80–85, April 1998.

[2] C. Alpert, J. Huang, and A. Kahng. Recent Directions in Netlist Partitioning. *Integration, the VLSI Journal*, 19(1–2):1–81, 1995.

[3] S. Barnard. PMRSB: Parallel Multilevel Recursive Spectral Bisection. In *Proc. 1995 ACM/IEEE Supercomputing Conference*, 1995.

[4] S. Barnard and H. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Practice and Experience*, 6(2):101–117, April 1994.

[5] J. Bradley, N. Dingle, P. Harrison, and W. Knottenbelt. Performance Queries on semi-Markov Stochastic Petri Nets with an Extended Continuous Stochastic Logic. In *Proc. 10th International Workshop on Petri Nets and Performance Models (PNPM'03)*, pages 62–71, Urbana-Champaign IL, USA, September 2nd–5th 2003.

[6] J. Bradley, N. Dingle, W. Knottenbelt, and H. Wilson. Hypergraph-based Parallel Computation of Passage Time Densities in Large semi-Markov Models. In *Proc. 4th International Conference on the Numerical Solution of Markov Chains (NSMC'03)*, pages 99–120, Urbana-Champaign IL, USA, September 2nd–5th 2003.

[7] A. Caldwell, A. Kahng, and I. Markov. Improved Algorithms for Hypergraph Bipartitioning. In *Proc. 2000 Conference on Asia South Pacific Design Automation*, pages 661–666. ACM/IEEE, January 2000.

[8] U. Catalyurek and C. Aykanat. Hypergraph-Partitioning-Based Decomposition for Parallel Sparse-Matrix Vector Multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):673–693, 1999.

[9] N. Dingle, W. Knottenbelt, and P. Harrison. Uniformization and Hypergraph Partitioning for the Distributed Computation of Response Time Densities in Very Large Markov Models. *Journal of Parallel and Distributed Computing*, 2004. (To appear).

[10] S. Dutt and W. Deng. A Probability-based Approach to VLSI Circuit Partitioning. In *Proc. 33rd Annual Design Automation Conference*, pages 100–105, June 1996.

[11] S. Dutt and W. Deng. VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques. In *Proc. 1996 IEEE/ACM International Conference on Computer-Aided Design*, pages 194–200, Nov 1996.

[12] S. Dutt and H. Theny. Partitioning Around Roadblocks: Tackling Constraints with Intermediate Relaxations. In *Proc. 1997 IEEE/ACM International Conference on Computer-Aided Design*, pages 350–355, Nov 1997.

[13] C. Fiduccia and R. Mattheyses. A Linear Time Heuristic For Improving Network Partitions. In *Proc. 19th IEEE Design Automation Conference*, pages 175–181, 1982.

[14] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., 1979.

Hypergraph	Model Parameters	#vertices	#nets	#pins	Size On Disk (MB)
Voting 100	100/30/4	249 760	249 760	1 391 617	8.2
Voting 125	125/40/4	541 280	541 280	3 044 557	18
Voting 150	150/40/5	778 850	778 850	4 532 947	26
Voting 175	175/45/5	1 140 050	1 140 050	6 657 722	39
Voting 250	250/60/10	5 218 300	5 218 300	32 986 597	186
Voting 300	300/80/10	10 991 040	10 991 040	69 823 797	392

Table 3. Characteristics of hypergraphs used in the paper

[15] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, Cambridge, Massachussets, 2nd edition, 1999.

[16] E. Ihler, D. Wagner, and F. Wagner. Modeling Hypergraphs by Graphs with the same Mincut Properties. *Information Processing Letters*, 45:171–175, March 1993.

[17] G. Karypis. Multilevel Hypergraph Partitioning. Technical Report #02-25, University of Minnesota, 2002.

[18] G. Karypis and V. Kumar. A Coarse-Grain Parallel Formulation of Multilevel k-way Graph Partitioning Algorithm. In *Proc. 8th SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[19] G. Karypis and V. Kumar. *hMeTiS: A Hypergraph Partitioning Package, Version 1.5.3*. University of Minnesota, November 1998.

[20] G. Karypis and V. Kumar. Multilevel k-way Hypergraph Partitioning. Technical Report #98-036, University of Minnesota, 1998.

[21] G. Karypis and V. Kumar. A Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering. *Journal of Parallel and Distributed Computing*, 48:71–95, 1998.

[22] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

[23] G. Karypis, K. Schloegel, and V. Kumar. *ParMeTiS: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 3.0*. University of Minnesota, September 2002.

[24] W. Knottenbelt. *Parallel Performance Analysis of Large Markov Models*. PhD thesis, Imperial College, London, United Kingdom, February 2000.

[25] B. Krishnamurthy. An Improved min-cut Algorithm for Partitioning VLSI Networks. *IEEE Transactions on Computers*, 33(C):438–446, May 1984.

A. Appendix

The hypergraphs used in this paper are derived from a high-level Semi-Markov model of a voting system shown in Fig. 3. A full description of this model can be found in [6]. In this system, voters cast votes through polling units which in turn register votes with all available central voting units. Both polling units and central voting units can suffer breakdowns, from which there is a soft recovery mechanism. If, however, all the polling or voting units fail, then,

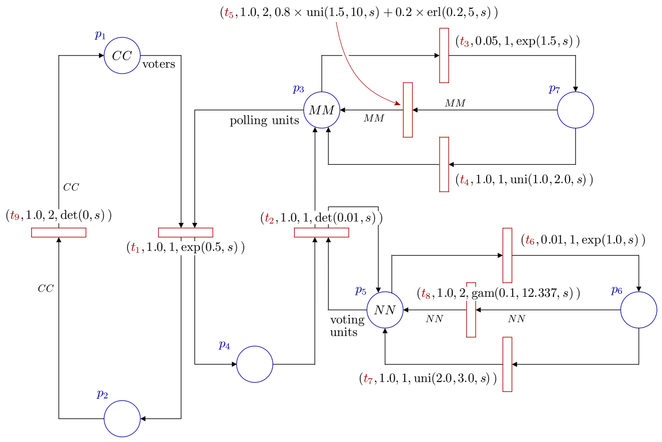

Figure 3. Semi-Markov Stochastic Petri net Voting System Model

with high priority, a failure recovery mode is instituted to restore the system to an operational state. The number of voters, polling units and central voting servers is configurable, and each combination of these parameters results in a sparse transition matrix of a different size, as shown in Table 3. The minimum information required to store a hypergraph are the weights of the vertices and the costs of the hyperedges in addition to the list of constituent vertices from each of the hyperedges (the pins). The sizes of hypergraphs on disk from Table 3 assume 32-bit integer types with no compression.

The aim of the analysis performed on these models is to find the response time density of the time taken for a certain number of voters to successfully register their votes. This requires the numerical inversion of the Laplace Transform of the response-time density, which in turn requires the solution of many thousands of sets of linear equations with the same non-zero sparsity pattern. Hypergraph partitioning is used to reduce the amount of inter-processor communication required to solve each system of linear equations in parallel. Note that the hypergraph partitioning needs only be performed once, but is reused several thousand times.