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

Aleksandar Trifunovic* and William J. Knottenbelt

Department of Computing,
Imperial College London, United Kingdom
E-mail: {at701,wjk}@doc.ic.ac.uk
*Corresponding author

Abstract:
In this paper we present a high-capacity, application-specific disk-based parallel multilevel k-way hypergraph partitioning algorithm. Our parallel algorithm provides the capability to partition very large hypergraphs that hitherto could not be partitioned because the memory required exceeds that available on a single workstation. The algorithm has three main phases: parallel coarsening, serial partitioning of the coarsest hypergraph and parallel uncoarsening. At each step of the parallel coarsening and parallel uncoarsening phases, 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 quality of partition is approximately 20% better in terms of the $(k-1)$ partitioning objective than that produced by an approximate graph partitioning-based approach using a state-of-the-art parallel graph partitioning tool.

Keywords: hypergraph partitioning; sparse matrix decomposition; load balancing for parallel computing; parallel sparse matrix–vector multiplication.

1 Introduction

A hypergraph is a set system, which (in a set-theoretic sense) means it is a collection of subsets of a given set of objects. The objects in this set are called vertices and the subsets within the collection are called hyperedges. Note that a hypergraph is also a generalisation of a graph, because in a graph each subset in the collection must have cardinality two, whereas in the hypergraph, there is no such restriction. An example of a graph and an example of a hypergraph are shown in Fig. 1.

A hypergraph may also be considered a data structure that represents a set of related objects. The objects are represented by the vertices of the hypergraph. The existence of a relationship between objects is represented by a hyperedge. Because

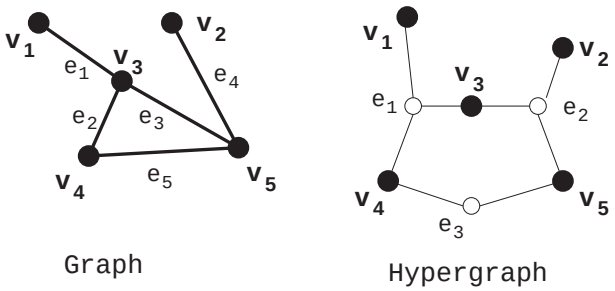

Figure 1 An example of a graph and a hypergraph

the cardinality of a hyperedge is not restricted, hypergraphs are more expressive than graphs. A scalar weight is assigned to each hyperedge to capture the degree of association between the vertices that the hyperedge connects; a scalar weight is also assigned to each vertex to quantify the size, area or computational load of the object that it models.

Graph and hypergraph models are useful in solving partitioning problems with hypergraphs being preferred in many applications of practical interest, such as load balancing for parallel computations (the allocation of work to processors) [12] and VLSI Computer-Aided Design (VLSI CAD) [2]. An important load balancing application is parallel sparse matrix–vector multiplication, where hypergraphs can model the interprocessor communication volume exactly (unlike graphs, which can only provide an approximation). In both load-balancing and VLSI CAD, a system decomposition into a number of subsystems that minimises the subsystem interconnect is sought. This is usually achieved by first partitioning the graph or hypergraph model into parts to minimise a partitioning objective function, subject to a partitioning constraint on the part weights. In the context of hypergraph models for parallel sparse matrix–vector multiplication, the $k-1$ objective function exactly quantifies the interprocessor communication volume [8].

It is known that computing the optimal partitions for many objective functions (including the $k-1$ objective) is NP-complete [18]. Thus, research has focused on developing polynomial time heuristic algorithms that give good sub-optimal solutions. Much work has been done on serial algorithms and a survey of these algorithms in the context of VLSI CAD is presented in [2]. Recently, the most successful heuristic algorithms (in terms of partition quality and runtime) have been those based on the multilevel paradigm [20, 7]. Multilevel algorithms form a pipeline consisting of three phases. During the coarsening phase, a sequence of successive smaller (coarser) hypergraphs is constructed by merging together selected vertices to form a single vertex in the coarser hypergraph. The initial partitioning phase then computes a sub-optimal partition of the smallest (coarsest) hypergraph in the sequence. Finally, during the uncoarsening phase, this partition is projected through the sequence of hypergraphs constructed during the coarsening phase and is further refined at each level.

Computing a k-way partition (where $k > 2$) can be done either via the recursive bisection approach (analogous to divide-and-conquer) or by computing the k-way partition directly. In [24], Karypis and Kumar empirically demonstrate that the direct partitioning approach may be superior both in terms of runtime and the $k-1$

partitioning objective to the recursive bisection approach, for larger values of k.

In [5, 6, 13], hypergraph partitioning is applied to the parallel computation of response time densities in Markov and semi-Markov chains. Hypergraph partitioning is a pre-processing step to the parallel sparse matrix–vector multiplications that are the kernel operations in iterative 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 [5, 6, 13], the computed partition is reused many thousands of 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 and $\Theta(10^8)$ non-zeros, so that in these cases the partitioning problem becomes intractable by sequential computation. By comparison, the largest hypergraph in the current VLSI CAD benchmark suite has just 184 752 vertices and 860 036 pins (non-zeros in its incidence matrix) [1].

In the absence of a parallel hypergraph partitioner, a graph model and a parallel graph partitioner were used in [5, 6, 13] to distribute the matrix across the processors for those cases where the hypergraph model could not be partitioned serially. However, while superior to a random partition, graph partitioning schemes do not accurately represent the actual interprocessor communication volume incurred during parallel matrix–vector multiplication, but merely provide (and thus attempt to minimize) an upper bound [8].

In this paper, we present an application-specific disk-based parallel formulation, built upon a sequential multilevel k-way partitioning algorithm for hypergraphs [24]. The disk-based approach is motivated by the successful use of out-of-core techniques to solve large systems of linear equations [29]. Our target application is parallel sparse matrix–vector multiplication. Specifically, we use the hypergraph model for one-dimensional row-wise sparse matrix decomposition, first described by Catalyurek and Aykanat in [8], to partition a Markov or semi-Markov transition matrix across the processors. We note that hypergraph models for two-dimensional sparse matrix decomposition have also been developed [9, 32].

The target architecture for our algorithm is a cluster of commodity workstations connected by a switched ethernet network. The algorithm is evaluated on a number of sparse hypergraphs modelling transition matrices arising from response time density computations and it is shown to consistently outperform an approximate graph-based method using the leading parallel graph partitioning tool `ParMeTiS` [27] by up to 27% in terms of the $k-1$ objective function.

The remainder of this paper is organized as follows. Section 2 describes serial multilevel hypergraph partitioning in more detail. Section 3 presents our parallel disk-based multilevel hypergraph partitioning algorithm and Section 4 the experimental evaluation. Finally, Section 5 concludes and describes possible directions for further research.

2 Serial Multilevel Hypergraph Partitioning

2.1 Problem Definition

Formally, a hypergraph $H(V, \mathcal{E})$ is a set system on the set of vertices, V. $\mathcal{E}$ is the set of hyperedges, such that for all $e \in \mathcal{E}$, $e \subseteq V$. The incidence matrix of a hypergraph $H(V, \mathcal{E})$, with $V = \{v_1, \ldots, v_n\}$ and $\mathcal{E} = \{e_1, \ldots, e_m\}$, is the $n \times m$ matrix $\mathbf{A} = (a_{ij})$, with entries

$$a_{ij} = \begin{cases} 1 \text{ if } v_i \in e_j \\ 0 \text{ otherwise} \end{cases}$$

The hypergraph model for one-dimensional row-wise sparse matrix decomposition assigns integer weights to the vertices and hyperedges as follows [8]. Each vertex $v \in V$ has a weight $w(v)$ equal to the number of hyperedges incident on v. This corresponds to the number of non-zeros in the row (i.e. the computational load that each row induces on the processor to which it is assigned). Each hyperedge $e \in \mathcal{E}$ has its weight $w(e)$ set to unity since this corresponds to the volume of communication when a vector element is communicated between two processors. We define the *size* of a hyperedge to be its cardinality.

The k-way hypergraph partitioning problem is to find k disjoint subsets (or parts) V_i of the vertex set V with corresponding weights W_i such that, given a prescribed balance criterion $0 < \epsilon \ll 1$,

$$W_i < (1 + \epsilon) W_{avg} \tag{1}$$

holds for all $i = 0, \ldots, k-1$ and the objective function over the hyperedges is minimized. Here, W_{avg} denotes the average part weight. When the $k-1$ partitioning objective function is used, the partition cost is given by

$$P_{cost} = \sum_{\{e \in \mathcal{E} : \lambda_e > 1\}} (\lambda_e - 1) w(e) \tag{2}$$

where λ_e is the number of parts spanned by hyperedge $e \in \mathcal{E}$. This formalizes the intuition of the sparse matrix decomposition that minimizes the volume of interprocessor communication of the parallel sparse matrix–vector multiplication, subject to maintaining a computational load balance.

The multilevel paradigm is preferred to *flat* partitioning approaches (i.e. those that do not attempt to approximate the original hypergraph) because it scales better in terms of runtime and partition quality with increasing problem size. Flat partitioning algorithms are more likely to get trapped in relatively poor local minima as problem size increases [2]. We note that the multilevel paradigm uses flat partitioning algorithms during the initial partitioning and the uncoarsening phases.

The following subsections describe the multilevel partitioning pipeline in more detail.

2.2 The Coarsening Phase

The aim of the coarsening phase is to reduce the original hypergraph problem instance $H(V, \mathcal{E})$ via a succession of smaller hypergraphs $H_i(V_i, \mathcal{E}_i)$, $i = 1, \ldots, c$, that maintain as far as possible the structure of the original hypergraph $H(V, \mathcal{E})$.

Coarsening is performed by merging vertices of hypergraph $H_i(V_i, \mathcal{E}_i)$ together to form a vertex of the successive coarser hypergraph $H_{i+1}(V_{i+1}, \mathcal{E}_{i+1})$. This *clustering* of vertices is represented by a map $g_i : V_i \rightarrow V_{i+1}$. The hyperedge set of the successive coarser hypergraph $\mathcal{E}_{i+1}$ is constructed from $\mathcal{E}_i$ by applying g_i to every vertex in each hyperedge $e \in \mathcal{E}_i$. Single vertex hyperedges in $\mathcal{E}_{i+1}$ are discarded as they cannot contribute to the objective function of a partition of $H_{i+1}(V_{i+1}, \mathcal{E}_{i+1})$. If more than one hyperedge maps onto the same hyperedge of the coarse hypergraph, only one copy of the hyperedge is retained, with its weight set to the sum of the weights 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 successive coarser hypergraphs. In addition, the coarsening should substantially reduce the size and number of hyperedges, because this will make the heuristic partitioning algorithms more effective during the initial partitioning phase.

The coarsening algorithm has a significant impact on the final partition quality since most heuristic algorithms tend to terminate at local minima with respect to the heuristic. 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 [20]. During our experiments, we have found that the *first choice* coarsening algorithm [24] and related algorithms (such as heavy connectivity clustering [8]) yield balanced partitions and fast runtimes for our case study hypergraphs. The first choice coarsening algorithm proceeds as follows. The vertices of the hypergraph $H_i(V_i, \mathcal{E}_i)$ are visited in a random order. For each vertex $v \in V_i$, all vertices $u \in V_i$ (both those already matched and those unmatched) that are connected via hyperedges incident on v are considered for matching with v. A connectivity metric is computed between each pair of vertices u and v, and the most strongly connected vertex to v is chosen for the matching, provided that the resulting 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. The vertex connectivity metric used in [24] is shown in Eqn. 3 below:

$$conn(u, v) = \sum_{\{e \in \mathcal{E}_i : u \in e, v \in e\}} \frac{1}{|e| - 1} \tag{3}$$

Another family of algorithms, known as *hyperedge coarsening* algorithms [20], seek a maximal independent set of hyperedges in $\mathcal{E}_i$. The vertices that belong to each of the hyperedges in the independent 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 weight. 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 partitions while not resulting in improvements in the objective function value over first choice coarsening.

An important parameter of the coarsening algorithm is the rate r at which a hypergraph is reduced in successive coarsening steps. It is given by Eqn. 4 below:

$$r = \frac{|V_i|}{|V_{i+1}|} \tag{4}$$

A low value of r implies that many coarsening steps may be required, thus increasing the runtime of the overall algorithm. On the other hand, a larger value of r may result in a poorer quality of coarsening as vertices are matched into sub-standard clusters in order to reduce the size of the hypergraph. In [20], Karypis reports that values of r in the range 1.5–1.8 provide a reasonable balance between runtime and solution quality. Our experience is similar with our case study hypergraphs, using values of r in the range 1.5–2.0.

2.3 The Initial Partitioning Phase

The initial partitioning phase computes a partition of the coarsest hypergraph $H_c(V_c, \mathcal{E}_c)$. This partition will be subsequently refined during the uncoarsening phase, as it is being projected through the successive finer hypergraphs. Because the coarsest hypergraph tends to be significantly smaller than the original problem instance, the time taken to compute the initial partitioning phase is usually considerably less than the time taken by the other phases of the multilevel pipeline.

In [24], the authors use recursive bisection to compute the initial k-way partition. We also adopt this approach, but note that it is possible to use a direct k-way method to produce an initial partition 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, 26]. This algorithm begins with a randomly selected vertex and grows a single part around it by assigning the most highly connected vertex to the part from the remaining vertices until the desired part size is achieved. The remaining unassigned vertices are allocated to the complement part. Because the algorithm is randomized, a number of initial bipartitions are computed and the best is retained for the uncoarsening phase.

2.4 The Uncoarsening Phase

Here, we propagate the initial partition back up through the successive finer hypergraphs and at each step further refine the partition using a heuristic refinement algorithm (in essence, a flat iterative improvement algorithm [2]). When the overall k-way partition is computed via recursive bisection, the uncoarsening phase requires 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 part at most once; the best sequence of moves found by the heuristic is actually performed leading to the refined partition. The algorithms operate in $O(z)$ time per pass, where z is the number of pins in the hypergraph (or the number of non-zeros in its incidence matrix), and usually converge within a few passes [17] 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 [30, 14, 15, 16].

Extending the FM algorithm to refine a k-way partition directly at each uncoarsening step increases both the time complexity of the algorithm and the likelihood that the algorithm terminates at a relatively poor local minimum [11, 24]. However, good results have been reported with a *greedy refinement* algorithm, especially for increasing values of k [24]. The greedy refinement algorithm performs passes, during each of which the vertices are visited in a random order and moved to the part that yields the largest positive gain in the objective function. Since the hypergraph is sparse, the algorithm avoids calculating the gain to each of the $k-1$ other parts as follows. Vertices are not considered for a move if they are internal to their current part (i.e. all adjacent vertices are also in the same part). Otherwise, the gain of a move is only computed for neighbouring parts (those parts that contain adjacent vertices) if the move to the neighbouring part does not violate the balance constraint. Experiments have showed that the algorithm typically converges after a small number of iterations [24].

A more sophisticated refinement scheme iteratively repeats the coarsening and refinement phases, such that during subsequent applications of the coarsening phase, vertices are matched together only if they have been assigned to the same part in the partition; this preserves the cut properties of the partition during the coarsening phase [21, 7]. Such multi-phase refinement is implemented in the state-of-the-art serial tools `hMeTiS` [23] and `PaToH` [10]. 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 our paper, namely the application-specific disk-based parallel multilevel k-way hypergraph partitioning algorithm.

3.1 Approaches to Parallel Multilevel Hypergraph Partitioning

We note that the algorithms making up the multilevel pipeline are inherently sequential in nature, making it difficult to find opportunities for concurrency. Moreover, whereas parallel multilevel graph partitioning algorithms have been developed [22, 25, 3, 33], none have yet been forthcoming for multilevel hypergraph partitioning.

It has been shown that, in general, there does not exist a graph model that correctly represents the cut properties of the corresponding hypergraph [19]. Thus, it is not possible to directly minimise an objective function on a hypergraph by constructing a graph model and applying a parallel graph partitioning algorithm, although this may give reasonable approximations in some cases.

Here, 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 one to an upper bound given by the number of vertices. This difference is significant in the context of the main obstacle to parallelism in graph and hypergraph partitioning: the presence of adjacent vertices

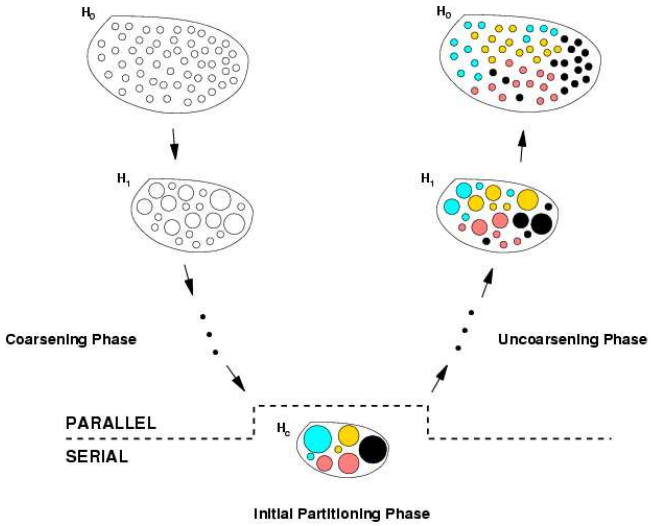

Figure 2 Parallel multilevel pipeline

on different processors.

Firstly, consider a move-based partitioning algorithm. Concurrent movement of adjacent vertices on different processors potentially causes a conflict because the gain of each vertex move, as computed by the respective processor, is conditional upon its adjacent vertices remaining fixed in their respective parts.

Secondly, consider the edge coarsening algorithm [21]. Serially, it matches an unmatched vertex with the most strongly connected neighbouring unmatched vertex that maximises the connectivity metric from Eqn. 3. In parallel, suppose that processors compute vertex matches concurrently. A processor may match an unmatched local vertex with (what it thinks) is another unmatched vertex on another processor; however, this remote vertex may have already been matched by its owner processor.

In [22], in the context of parallel graph partitioning, Karypis and Kumar coloured the vertices of the graph to identify groups of vertices that did not share any common edges. In any given step, the algorithm would operate concurrently only on vertices corresponding to the same colour, avoiding the conflicts outlined above. Identifying groups of vertices that do not share any common hyperedges in the hypergraph may be achieved by first constructing a (graph) clique model by replacing each hyperedge with a vertex clique and then colouring the resulting graph. However, because large hyperedges will induce large cliques in the graph, its chromatic number is also likely to be large[a].

In the absence of obvious fine-grained parallelism, a coarse-grained formulation is sought. We note that only the coarsening and uncoarsening phases need to be parallelised. During the initial partitioning phase, the coarsest hypergraph should be small enough to be partitioned serially on a single processor and the time-complexity of this serial component should be dominated by the time-complexities

[a]The number of colours used is the chromatic number of the graph. Suppose that in a graph $G(V, \mathcal{E})$ the largest vertex clique has d vertices. Then, d is (trivially) a lower bound on the chromatic number $\chi(G)$.

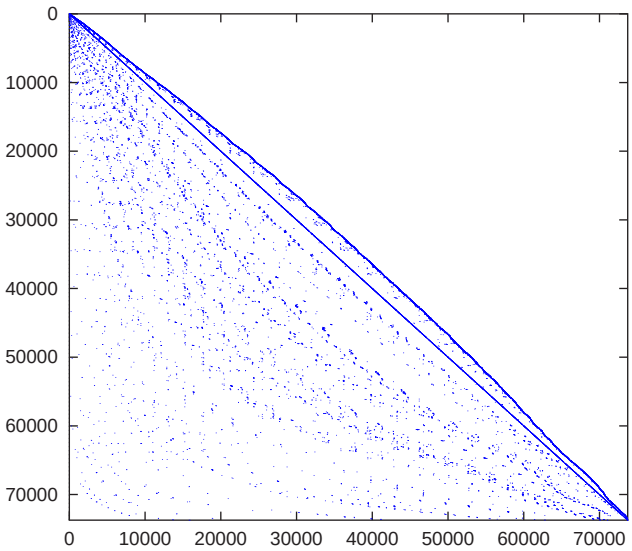

Figure 3 An example of a semi-Markov transition matrix generated by a breadth-first state traversal

of the parallel coarsening and parallel uncoarsening phases. The proposed parallel multilevel pipeline is illustrated in Fig. 2. The sections below describe the data distribution strategy, parallel coarsening and parallel refinement phases of our disk-based algorithm in more detail.

3.2 Data Distribution

The natural way to store a hypergraph $H_i(V_i, \mathcal{E}_i)$ at stage i in a multilevel algorithm across p processors is to store $|V_i|/p$ vertices and $|\mathcal{E}_i|/p$ hyperedges on each processor. The processors are assigned non-overlapping sets of vertices; $V_i^{p_j}$ on processor p_j, such that $\bigcup_j V_i^{p_j} = V_i$.

In order to eliminate the communication overhead that would arise when locating remote vertices that are adjacent to a local vertex $v \in V_i^{p_j}$ on processor p_j, *all* hyperedges incident on vertices in $V_i^{p_j}$ are assigned to $\mathcal{E}_i^{p_j}$ (i.e. allocated to processor p_j).

Note that this hyperedge-to-processor allocation may result in some hyperedges being replicated across several processors, since vertices incident on such hyperedges will be assigned to different processors. We refer to these as *frontier* hyperedges. To increase capacity, the algorithm only stores in memory the hypergraph corresponding to the current stage in the multilevel pipeline; the remaining hypergraphs are stored on disk and loaded into memory as required.

3.3 Parallel Coarsening Phase

The serial coarsening algorithm chosen for parallelisation is the first choice coarsening algorithm. In a naïve parallel formulation, if the processors were to apply the serial first choice algorithm to their local vertex sets concurrently during coars-

Hypergraph	#vertices	#hyperedges	#pins	Size (MB)
voting100	249 760	249 760	1 391 617	8.2
voting125	541 280	541 280	3 044 557	18
voting150	778 850	778 850	4 532 947	26
voting175	1 140 050	1 140 050	6 657 722	39
voting250	5 218 300	5 218 300	32 986 597	186
voting300	10 991 040	10 991 040	69 823 797	392

Table 1 Characteristics of hypergraphs used in the paper

ening stage i, potentially excessive interprocessor communication may result (since adjacent vertices may be located across several processors).

We note that transition matrices of Markov and semi-Markov chains exhibit an approximate lower-triangular structure when generated by a breadth-first search of the state-space [28]. An example of such a matrix, taken from [28], is shown in Fig. 3. In the belief that the approximate lower-triangular structure of the transition matrix can ensure a sufficient number of strongly connected local vertex clusters, our parallel coarsening algorithm only matches together vertices local to a processor. This avoids interprocessor communication during the vertex matching computation.

We first perform an experiment in order to indicate whether restricting the parallel coarsening algorithm in this manner still yields a sufficient number of good vertex matches, when compared to the unrestricted coarsening algorithm. To this end, the edge coarsening algorithm [21] was applied to hypergraphs derived from transition matrices of a semi-Markov model of a voting system [4, 6]. A detailed description of this model can be found in Appendix A while the main characteristics of the voting hypergraphs are shown in Table 1. It was conjectured that two vertices close in terms of their index (corresponding to two rows that are close in the transition matrix) would form good matches. This is because the upper triangular part of the matrix is mostly zero, while the diagonal region is the densest part of the matrix.

We performed the experiment as follows. The vertex set V was partitioned into a number of subsets, each of which contained vertices with contiguous index. During the edge coarsening procedure, whenever unmatched vertices are matched with unmatched vertices within the same subset, the match was called a *local* match. When vertices are matched with others from a different subset, the match was called a *remote* match. Finally, vertices that were simply copied over to the coarser hypergraph were denoted singleton matches. Table 2 shows the average percentages of the above match types over ten runs of the algorithm on the voting hypergraphs.

The results of the experiment indicate that vertices tend to seek matches with their immediate neighbours (defined in terms of their indices) in hypergraphs representing Markov and semi-Markov transition matrices. Our parallel coarsening algorithm exploits this property by allocating vertices of contiguous index to the set $V_i^{p_j}$ on processor p_j (this corresponds to allocating contiguous rows of the transition matrix to p_j).

Once the map g_i has been computed across the processors, we construct the hypergraph $H_{i+1}(V_{i+1}, \mathcal{E}_{i+1})$ using $H_i(V_i, \mathcal{E}_i)$ and g_i. A b–bit hash key is associ-

Hypergraph	partition size	% local	% remote	% singleton
voting100	2	90.1	0.9	9.0
voting100	4	88.1	2.9	9.0
voting100	8	84.4	6.7	8.9
voting100	16	77.2	13.9	8.9
voting100	32	62.8	28.2	9.0
voting125	2	90.4	0.7	8.9
voting125	4	88.8	2.3	8.9
voting125	8	85.9	5.2	8.9
voting125	16	79.9	11.2	8.9
voting125	32	68.6	22.5	8.9
voting150	2	91.5	0.7	7.8
voting150	4	90.2	2.0	7.8
voting150	8	87.5	4.7	7.8
voting150	16	82.4	9.8	7.8
voting150	32	72.4	19.8	7.8
voting175	2	91.6	0.6	7.8
voting175	4	90.5	1.7	7.8
voting175	8	88.2	4.0	7.8
voting175	16	83.8	8.4	7.8
voting175	32	75.1	17.1	7.8

Table 2 Percentages of match types in the edge coarsening algorithm during the vertex connectivity analysis

ated with each hyperedge $e \in \mathcal{E}_i$ and used to assign "ownership" of hyperedges to processors. This hash key is computed using a hash-function $h : \mathbb{N}^a \to \mathbb{N}$, where a is the maximum hyperedge cardinality in $\mathcal{E}_i$. It posesses the desirable property that for an arbitrary set of hyperedges E, $h(e)$ `mod` p, $e \in E$, is near-uniformly distributed [28]. In our experiments, we set $b = 64$.

Each processor only contracts those hyperedges that have been assigned to it by the hash function. This ensures that only one copy of a given hyperedge is contracted (as potentially multiple copies of the given hyperedge exist across processors). Each processor first contracts the local hyperedges using its portion of the map g_i, then communicates this portion of the vector representation of g_i to the other $p-1$ processors. This enables every processor to fully contract its hyperedges. There is no need for explicit removal of duplicate hyperedges following the hyperedge contraction, because this is done when the hyperedges are read in from disk by the processors at the beginning of the subsequent coarsening step.

Motivated by [25], we use a smaller number of processors as the hypergraph is reduced in successive coarsening steps. In our implementation, we only use processor numbers that are powers of two. When the hypergraph is considered to be small enough to fit on a single processor, a serial multilevel algorithm is used; in our implementation, we require that the hypergraph has been reduced by a factor of p before using a serial algorithm.

3.4 Initial Partitioning Phase

During this phase, a partition of the coarse hypergraph is computed serially on a single processor. In principle, any serial algorithm may be used, as this is the least time-critical phase in the parallel multilevel algorithm. For our experiments,

we used the `HMETIS_PartKway()` routine from the `hMeTiS` library [23].

3.5 Parallel Uncoarsening Phase

During the i^{th} level in the uncoarsening phase, each processor is responsible for vertices from k/p parts and the hyperedges incident on these vertices. As in the coarsening phase, some hyperedges are replicated across multiple processors (frontier hyperedges). The parallel refinement algorithm proceeds in a number of steps. During each step, each processor performs local refinement on the parts it currently owns (using serial FM when $k = 2p$, and using the serial greedy k-way refinement [24], if $k > 2p$). If $p = k$, the processors pair-up and only one processor performs serial FM refinement on the two parts.

A round-robin communication of vertices and incident hyperedges is then performed, in order for subsequent steps to consider different directions of vertex move. The partition balance constraint is enforced locally, since each processor refines an independent set of parts. As in the parallel coarsening phase, fewer processors are used at the coarser levels.

4 Implementation and Experimental Evaluation

The disk-based parallel algorithm was implemented in the C++ language using the Message Passing Interface (MPI) standard [31] for interprocessor communication, forming the `Par`*k*`way1.0` tool. This experimental implementation consisted of three phases:

1. Parallel coarsening using the parallel formulation of the first choice coarsening algorithm from Section 3.3. A coarsening reduction ratio of 2.0 (cf. Eqn. 4) was enforced on each processor.
2. Serial initial partitioning performed by the `HMETIS_PartKway()` routine of the `hMeTiS` library. This is called on a single processor when the coarse hypergraph has $O(n/p)$ vertices.
3. Parallel refinement on the partition output by `HMETIS_PartKway()` using the parallel refinement algorithm described in Section 3.5.

The architecture used in the experiments consisted of a cluster of commodity PC workstations, connected by a switched 100 Mbps ethernet network. Each PC was equipped with a 2.8GHz Pentium 4 CPU and 1GB RAM.

The experimental evaluation was carried out on hypergraph representations of transition matrices from the voting model with 250 and 300 voters [4, 5, 6], yielding the voting250 and voting300 hypergraphs, respectively. The voting model is described in more detail in Appendix A. The hypergraphs were constructed from the sparse matrices according to the hypergraph model for one-dimensional row-wise sparse matrix decomposition [8]; their main characteristics can be found in Table 1. The minimum information required to store a hypergraph consists of 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 in Table 1 assume 32-bit integer types with no compression.

Partition size	voting250 results using 4 processors			
	Par$k$way1.0		ParMeTiS	
	$k-1$ objective	time(s)	$k-1$ objective	time(s)
8	91 511	1 309	117 354	25
16	182 206	1 393	249 415	27
32	354 561	1 495	402 681	32
64	525 856	1 777	610 597	33
total:	1 154 134	5 974	1 380 047	117

Table 3 Par$k$way1.0 and ParMeTiS: runtime and partition quality results on the voting250 hypergraph

Partition size	voting300 results using 8 processors			
	Par$k$way1.0		ParMeTiS	
	$k-1$ objective	time(s)	$k-1$ objective	time(s)
16	322 737	4 827	442 387	85
32	529 763	4 762	687 659	61
64	874 652	5 007	1 033 312	80
total:	1 727 152	14 596	2 163 358	246

Table 4 Par$k$way1.0 and ParMeTiS: runtime and partition quality results on the voting300 hypergraph

In order to quantify the communication volume of parallel sparse matrix–vector multiplication exactly, the partitioning objective used in the experiments was the $k-1$ metric (cf. Eqn. 2). A partitioning balance constraint of 5% was imposed, equivalent to setting $\epsilon = 0.05$ in Eqn. 1.

Both problem instances were too large to be partitioned on a single workstation, so a suitable comparison was provided by the state-of-the-art parallel graph partitioning tool ParMeTiS [27]. The transition matrices were converted into appropriate input for ParMeTiS according to the transformations described in [8]. We used default parameter values in ParMeTiS. Note that it is not possible to explicitly enforce the balance constraint on partitions produced by ParMeTiS; however, the vast majority of partitions produced satisfied the 5% balance constraint.

Table 3 presents the experimental results for the voting250 hypergraph and Table 4 results for the voting300 hypergraph. The results indicate that the proposed parallel hypergraph partitioning algorithm significantly dominates the approximation given by parallel graph partitioning in terms of partition quality. On average, the algorithm produces partitions with $k-1$ objective values 20% lower than those produced by ParMeTiS on the voting300 hypergraph and 16% lower on the smaller voting250 hypergraph. In turn, ParMeTiS significantly dominates our disk-based algorithm in terms of runtime.

There are a number of reasons for the large difference in the respective runtimes. Firstly, hypergraph partitioning is an inherently more "difficult" problem than graph partitioning. Secondly, the implementation experienced slow disk access time due to high disk contention; this may partly be due to our use of disk storage on a shared departmental file server. Thirdly, the parallel refinement algorithm explicitly communicated vertices and the corresponding incident hyperedges in order to consider the different directions of vertex move. The volume of this communication was observed to be very large and increased with the number of processors used.

5 Conclusion

We have devised a high-capacity parallel formulation of the multilevel k-way hypergraph partitioning algorithm and have demonstrated its ability to partition very large hypergraphs arising from semi-Markov chain models with $\Theta(10^7)$ vertices by combining the memory and processing power of several workstations. To the best of our knowledge, 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 partitions produced by our parallel tool comfortably exceeds the approximate partitions 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 reduced 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 is not a significant factor for many problem instances where a single partition may be reused several hundred thousand times (e.g. in the parallel Laplace Transform-based response time analyser described in [4, 5, 6]).

References and Notes

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

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

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

4 J.T. Bradley, N.J. Dingle, P.G. Harrison, and W.J. 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.

5 J.T. Bradley, N.J. Dingle, W.J. Knottenbelt, and H.J. 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.

6 J.T. Bradley, N.J. Dingle, W.J. Knottenbelt, and H.J. Wilson. Hypergraph-based Parallel Computation of Passage Time Densities in Large semi-Markov Models. *Linear Algebra and Its Applications*, 386:311–334, July 2004.

7 A.E. Caldwell, A.B. Kahng, and I.L. Markov. Improved Algorithms for Hypergraph Bipartitioning. In *Proc. 2000 Conf. Asia South Pacific Design Automation*, pages 661–666. ACM/IEEE, January 2000.

8 U.V. 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 U.V. Catalyurek and C. Aykanat. A Fine-Grain Hypergraph Model for 2D Decomposition of Sparse Matrices. In *Proc. 15th International Parallel and Distributed Processing Symposium*, San Francisco, USA, April 2001.

10 U.V. Catalyurek and C. Aykanat. *PaToH: Partitioning Tool for Hypergraphs, Version 3.0*, 2001.

11 J. Cong and S.K. Lim. Multiway Partitioning with Pairwise Movement. In *Proc. ACM/IEEE International Conference on Computer Aided Design*, pages 512–516, San Jose, CA, Nov 1998.

12 K.D. Devine, E.G. Boman, R.T. Heaphy, B.A. Hendrickson, J.D. Teresco, J. Faik, J.E. Flaherty, and L.G. Gervasio. New Challenges in Dynamic Load Balancing. *Applied Numerical Mathematics*, 52(2–3):133–152, 2005.

13 N.J. Dingle, W.J. Knottenbelt, and P.G. Harrison. Uniformization and Hypergraph Partitioning for the Distributed Computation of Response Time Densities in Very Large Markov Models. *Journal of Parallel and Distributed Computing*, 64(8):908–920, August 2004. (To appear).

14 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.

15 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.

16 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.

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

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

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

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

21 G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. *IEEE Transactions on VLSI Systems*, 7(1):69–79, 1999.

22 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.

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

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

25 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.

26 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.

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

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

29 W.J. Knottenbelt and P.G. Harrison. Distributed disk-based solution techniques for large Markov models. In *Proceedings of the 3rd International Meeting on the Numerical Solution of Markov Chains (NSMC '99)*, pages 58–75, Zaragoza, Spain, September 1999.

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

31 M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI – The Complete Reference.* MIT Press, Cambridge, Massachussets, 2nd edition, 1998.

32 B. Vastenhouw and R.H. Bisseling. A Two-Dimensional Data Distribution Method for Parallel Sparse Matrix-Vector Multiplication. *SIAM Review*, 47(1):67–95, 2005.

33 C. Walshaw, M. Cross, and M. G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *J. Parallel Distrib. Comput.*, 47(2):102–108, 1997.

A Appendix

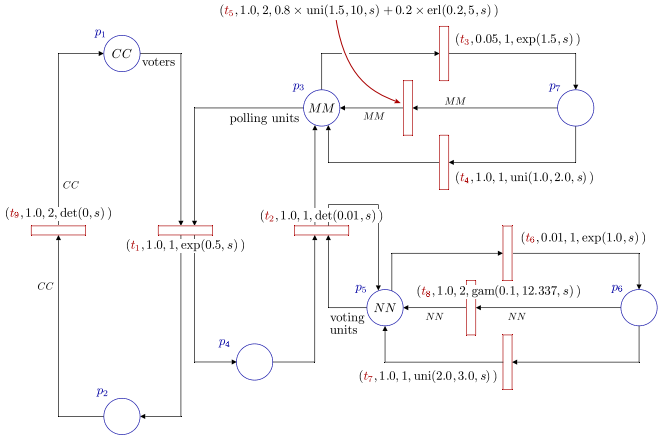

Figure 4 Semi-Markov Stochastic Petri net Voting System Model

The hypergraphs used in this paper are derived from a high-level semi-Markov model of a voting system shown in Fig. 4. A full description of this model can be found in [4, 5, 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, with high priority, a failure recovery mode is instituted to restore the system to an operational state. The numbers of voters, polling units and central voting servers are configurable, and each combination of these parameters results in a sparse transition matrix of a different size, as shown in Table 5.

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

Voting model parameters	Transition matrix/hypergraph
100/30/4	voting100
125/40/4	voting125
150/40/5	voting150
175/45/5	voting175
250/60/10	voting250
300/80/10	voting300

Table 5 Voting model parameters that yield hypergraph representations of sparse transition matrices used in the paper

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. Each set of linear equations requires hundreds of iterations to convergence. The parallel sparse matrix–vector multiplication time dominates the per-iteration time when solving a set of linear equations. Hypergraph partitioning is used to reduce the amount of inter-processor communication and hence, the per-iteration time. Note that hypergraph partitioning need only be performed once and is "reused" several hundred thousand times.