Decomposing Linear Programs for Parallel Solution*

Ali Pınar, Ümit V. Çatalyürek, Cevdet Aykanat
Computer Engineering Department
Bilkent University, Ankara, Turkey

Mustafa Pınar
Industrial Engineering Department
Bilkent University, Ankara, Turkey

Abstract. Coarse grain parallelism inherent in the solution of Linear Programming (LP) problems with block angular constraint matrices has been exploited in recent research works. However, these approaches suffer from unscalability and load imbalance since they exploit only the existing block angular structure of the LP constraint matrix. In this paper, we consider decomposing LP constraint matrices to obtain block angular structures with specified number of blocks for scalable parallelization. We propose hypergraph models to represent LP constraint matrices for decomposition. In these models, the decomposition problem reduces to the well-known hypergraph partitioning problem. A Kernighan-Lin based multiway hypergraph partitioning heuristic is implemented for experimenting with the performance of the proposed hypergraph models on the decomposition of the LP problems selected from `NETLIB` suite. Initial results are promising and justify further research on other hypergraph partitioning heuristics for decomposing large LP problems.

1 Introduction

Linear Programming (LP) is currently one of the most popular tools in modeling economic and physical phenomena where performance measures are to be optimized subject to certain requirements. Algorithmic developments along with successful industrial applications and the advent of powerful computers have increased the users' ability to formulate and solve large LP problems. But, the question still remains on how far we can push the limit on the size of large linear programs solvable by today's *parallel processing technology* .

The parallel solution of block angular LP's has been a very active area of research in both operations research and computer science societies. One of the most popular approaches to solve block-angular LP's is the Dantzig–Wolfe decomposition [1]. In this scheme, the block structure of the constraint matrix is exploited for parallel solution in the subproblem phase where each processor solves a smaller LP corresponding to a distinct block. A sequential coordination phase (the master) follows. This cycle is repeated until suitable termination criteria are satisfied. Coarse grain parallelism inherent in these approaches has been exploited in recent research works [5, 8]. However, the success of these approaches depends only on the existing *block angular* structure of the given constraint matrix. The number of processors utilized for parallelization in these

* This work is partially supported by the Commission of the European Communities, Directorate General for Industry under contract ITDC 204–82166

studies is clearly limited by the number of inherent blocks of the constraint matrix. Hence, these approaches suffer from *unscalability* and *load imbalance*.

This paper focuses on the problem of decomposing irregularly sparse constraint matrices of large LP problems to obtain block angular structure with specified number of blocks for scalable parallelization. The literature that addresses this problem is extremely rare and very recent. Ferris and Horn [2] model the constraint matrix as a bipartite graph. In this graph, the bipartition consists of one set of vertices representing rows, and another set of vertices representing columns. There exists an edge between a row vertex and a column vertex if and only if the respective entry in the constraint matrix is nonzero. The objective in the decomposition is to minimize the size of the master problem while maintaining computational load balance among subproblem solutions. Minimizing the size of the master problem corresponds to minimizing the sequential component of the overall parallel scheme. Maintaining computational load balance corresponds to minimizing processors' idle time during each subproblem phase.

In the present paper, we exploit hypergraphs for modeling constraint matrices for decomposition. A hypergraph is defined as a set of vertices and a set of *nets* (hyperedges) between those vertices. Each net is a subset of the vertices of the hypergraph. In this work, we propose two hypergraph models for decomposition. In the first model—referred to here as the *row–net* model—each row is represented by a net, whereas each column is represented by a vertex. The set of vertices connected to a net corresponds to the set of columns which have a nonzero entry in the row represented by this net. In this case, the decomposition problem reduces to the well-known *hypergraph partitioning* problem which is known to be *NP-Hard*.

The second model—referred to here as the *column–net* model—is very similar to the row–net model, only the roles of columns and rows are exchanged. The column–net model is exploited in two distinct approaches. In the first approach, hypergraph partitioning in the column–net model produces the dual LP problem in primal block angular form. In the second approach, dual block angular matrix achieved by hypergraph partitioning is transformed into a primal block angular form by using a technique similar to the one used in stochastic programming to treat non-anticipativity [9].

2 Preliminaries

A hypergraph $\mathcal{H}(\mathcal{V},\mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{N}$ between those vertices. Every net $n \in \mathcal{N}$ is a subset of vertices. The vertices in a net are called *pins* of the net. A graph is a special instance of a hypergraph such that each edge has exactly two pins. $\Pi = (P_1, \ldots, P_k)$ is a *k-way partition* of $\mathcal{H}$ if the following conditions hold: each part $P_\ell, 1 \leq \ell \leq k$ is a nonempty subset of $\mathcal{V}$, parts are pairwise disjoint, and union of k parts is $\mathcal{V}$.

In a partition Π of $\mathcal{H}$, a net that has at least one pin (vertex) in a part is said to *connect* that part. A net is said to be *cut* if it connects more than one part, and *uncut* otherwise. The set of uncut (*internal*) nets and cut (*external*) nets for a partition Π are denoted as $\mathcal{N}_I$ and $\mathcal{N}_E$, respectively. The cost of a partition Π (*cutsize*) is defined by the cardinality of the set of external nets, i.e., $cutsize(\Pi) = |\mathcal{N}_E| = |\mathcal{N}| - |\mathcal{N}_I|$. A partition Π of a hypergraph $\mathcal{H}$ is said to be feasible if it satisfies a given balance criterion $V_{avg}(1-\varepsilon) \leq |P_i| \leq V_{avg}(1+\varepsilon)$ $for \;\; i = 1, \ldots k$. Here, ε represents the predetermined maximum *imbalance ratio* allowed on part sizes, and $V_{avg} = |\mathcal{V}|/k$ represents the part size under perfect balance condition. Hence, we can define the hypergraph partitioning problem as the task of dividing a hypergraph into two or more parts such that the number of cut nets (cutsize) is minimized, while maintaining a given balance criterion among the part sizes.

Hypergraph partitioning is an NP-hard combinatorial optimization problem, hence we should resort to heuristics to obtain a good solution. However, especially in this application, heuristics to be adopted should run in low-order polynomial time. Because, these heuristics will be executed most probably in sequential mode as a preprocessing phase of the overall parallel LP program. Hence, we investigate the fast Kernighan–Lin (KL) based heuristics for hypergraph partitioning in the context of decomposing linear programs. These KL-based heuristics are widely used in VLSI layout design.

The basis of the KL-based heuristics is the seminal paper by Kernighan and Lin [6]. KL algorithm is an iterative improvement heuristic originally proposed for 2–way graph partitioning (bipartitioning). KL algorithm performs a number of passes over the vertices of the circuit until it finds a locally minimum partition. Each pass consists of repeated pairwise vertex swaps. Schweikert and Kernighan [11] adapted KL algorithm to hypergraph partitioning. Fiduccia and Mattheyses [3] introduced vertex move concept instead of vertex swap. The vertex move concept together with proper data structures, e.g., bucket lists, reduced the time complexity of a single pass of KL algorithm to linear in the size of the hypergraph. Here, size refers to the number of pins in a hypergraph. The original KL algorithm is not practical to use for large graphs and hypergraphs because of its high time complexity, and so the partitioning algorithms proposed after Fiduccia-Mattheyses' algorithm (FM algorithm) have utilized all the features of FM algorithm. Krishnamurthy [7] added to FM algorithm a *look-ahead* ability, which helps to break ties better in selecting a vertex to move. Sanchis [10] generalized Krishnamurthy's algorithm to a multiway hypergraph partitioning algorithm so that it could directly handle the partitioning of a hypergraph into more than two parts. All the previous approaches before Sanchis' algorithm (SN algorithm) are originally bipartitioning algorithms.

3 Hypergraph Models for Decomposition

This section describes the hypergraph models proposed for decomposing LP's. In the row–net model, the LP constraint matrix A is represented as the hypergraph $\mathcal{H}_{\mathcal{R}}(\mathcal{V}_{\mathcal{C}}, \mathcal{N}_{\mathcal{R}})$. The vertex and net sets $\mathcal{V}_{\mathcal{C}}$ and $\mathcal{N}_{\mathcal{R}}$ correspond to the columns and rows of the A matrix, respectively. There exist one vertex v_i and one net n_j for each column and row, respectively. Net n_j contains the vertices corresponding to the columns which have a nonzero entry on row j. Formally, $v_i \in n_j$ if and only if $a_{ji} \neq 0$. A k-way partition of $\mathcal{H}_R$ can be considered as inducing a row

$$A_B^p = \begin{pmatrix} B_1 & & & \\ & B_2 & & \\ & & \ddots & \\ & & & B_k \\ R_1 & R_2 & \dots & R_k \end{pmatrix} \qquad A_B^d = \begin{pmatrix} B_1 & & & & C_1 \\ & B_2 & & & C_2 \\ & & \ddots & & \vdots \\ & & & B_k & C_k \end{pmatrix}$$

Fig. 1. Primal (A_B^p) and dual (A_B^d) block angular matrices

and column permutation on A converting it into a primal block angular form A_B^p with k blocks as shown in Fig. 1. Part P_i of $\mathcal{H}_R$ corresponds to block B_i of A_B^p such that vertices and internal nets of part P_i constitute the columns and rows of block B_i, respectively. The set of external nets $\mathcal{N}_E$ corresponds to the rows of the master problem. That is, each cut net corresponds to a row of the submatrix $(R_1, R_2, \dots, R_k)$. Hence, minimizing the cutsize corresponds to minimizing the number of constraints in the master problem.

The proposed column–net model can be considered as the dual of the row–net model. In the column–net model $\mathcal{H}_\mathcal{C}(\mathcal{V}_\mathcal{R}, \mathcal{N}_\mathcal{C})$ of A, there exist one vertex v_i and one net n_j for each row and column of A, respectively. Net n_j contains the vertices corresponding to the rows which have a nonzero entry on column j. That is, $v_i \in n_j$ if and only if $a_{ij} \neq 0$. A k-way partition of $\mathcal{H}_\mathcal{C}$ can be considered as converting A into a dual block angular form A_B^d with k blocks as shown in Fig. 1. Part P_i of $\mathcal{H}_\mathcal{C}$ corresponds to block B_i of A_B^d such that vertices and internal nets of part P_i constitute the rows and columns of block B_i, respectively. Each cut net corresponds to a column of the submatrix $(C_1^t, C_2^t, \dots, C_k^t)^t$.

Dual block angular form of A_B^d leads to two distinct parallel solution schemes. In the first scheme, we exploit the fact that dual block angular constraint matrix of the original LP problem is a primal block angular constraint matrix of the dual LP problem. Hence, minimizing the cutsize corresponds to minimizing the number of constraints in the master problem of the dual LP.

In the second scheme, A_B^d is transformed into a primal block angular matrix for the original LP problem as described in [2, 9]. For each column j of the submatrix $(C_1^t, C_2^t, \dots, C_k^t)^t$, we introduce multiple column copies for the corresponding variable, one copy for each C_i that has at least one nonzero in column j. These multiple copies are used to decouple the corresponding C_i's on the respective variable such that the decoupled column copy of C_i is permuted to be a column of B_i. We then add column-linking row constraints that force these variables all to be equal. The column-linking constraints created during the overall process constitute the master problem of the original LP.

In this work, we select the number of blocks (i.e., k) to be equal to the number of processors. Hence, at each cycle of the parallel solution, each processor will be held responsible for solving a subproblem corresponding to a distinct block. However, a demand-driven scheme can also be adopted by choosing k to be greater than the number of processors. This scheme can be expected to yield better load balance since it is hard to estimate the relative run times of the subproblems according to the respective block sizes prior to execution.

4 Hypergraph Partitioning Heuristic

Sanchis's algorithm (SN) is used for multiway partitioning of hypergraph representations of the constraint matrices. Level 1 SN algorithm is briefly described here for the sake of simplicity of presentation. Details of SN algorithm which adopts multi-level gain concept can be found in [10]. In SN algorithm, each vertex of the hypergraph is associated with $(k-1)$ possible moves. Each move is associated with a *gain*. The *move gain* of a vertex v_i in part s with respect to part t $(t \neq s)$, i.e., the gain of the move of v_i from the home (source) part s to the destination part t, denotes the amount of decrease in the number of cut nets (cutsize) to be obtained by making that move. Positive gain refers to a decrease, whereas negative gain refers to an increase in the cutsize.

Figure 2 illustrates the pseudo-code of the SN based k-way hypergraph partitioning heuristic. In this figure, $nets(v)$ denotes the set of nets incident to vertex v. The algorithm starts from a randomly chosen feasible partition (Step 1), and iterates a number of passes over the vertices of the hypergraph until a locally optimum partition is found (*repeat–loop* at Step 2). At the beginning of each pass, all vertices are *unlocked* (Step 2.1), and initial $k-1$ move gains for each vertex are computed (Step 2.2). At each iteration (*while–loop* at Step 2.4) in a pass, a feasible move with the maximum *gain* is selected, tentatively performed, and the vertex associated with the move is *locked* (Steps 2.4.1–2.4.6). The locking mechanism enforces each vertex to be moved at most *once* per pass. That is, a locked vertex is not selected any more for a move until the end of the pass. After the move, the move gains affected by the selected move should be updated so that they indicate the effect of the move correctly. Move gains of only those unlocked vertices which share nets with the vertex moved should be updated.

---

1 construct a random, initial, feasible partition;
2 **repeat**
2.1 unlock all vertices;
2.2 compute $k-1$ move gains of each vertex $v \in V$
by invoking $computegain(H, v)$;
2.3 $mcnt = 0$;
2.4 **while** there exists a feasible move of an *unlocked* vertex **do**
2.4.1 select a feasible move with max gain g_{max} of an unlocked vertex v
from part s to part t;
2.4.2 $mcnt = mcnt + 1$;
2.4.3 $G[mcnt] = g_{max}$;
2.4.4 $Moves[mcnt] = \{v, s, t\}$;
2.4.5 *tentatively* realize the move of vertex v;
2.4.6 *lock* vertex v;
2.4.7 *recompute* the move gains of *unlocked* vertices $u \in nets(v)$
by invoking $computegain(H, u)$;
2.5 perform *prefix sum* on the array $G[1 \ldots mcnt]$;
2.6 select i^* such that $G_{max} = \max_{1 \leq i^* \leq mcnt} G[i^*]$;
2.7 **if** $G_{max} > 0$ **then**
2.7.1 *permanently* realize the moves in $Moves[1 \ldots i^*]$;
until $G_{max} \leq 0$;

Fig. 2. Level 1 SN hypergraph partitioning heuristic

```
computegain(H, u)
1           s ← part(u);
2           for each part t ≠ s do
2.1             g_u(t) ← 0;
3           for each net n ∈ nets(u) do
3.1             for each part t = 1, ..., k do
3.1.1               σ_n(t) ← 0;
3.2             for each vertex v ∈ n do
3.2.1               p ← part(v);
3.2.2               σ_n(p) ← σ_n(p) + 1;
3.3             for each part t ≠ s do
3.3.1               if σ_n(t) = |n| − 1 then
3.3.1.1                 g_u(t) ← g_u(t) + 1;
```

Fig. 3. Gain computation for a vertex u

Gain re-computation scheme is given here instead of gain update mechanism for the sake of simplicity in the presentation (Step 2.4.7).

At the end of each pass, we have a sequence of tentative vertex moves and their respective gains. We then construct from this sequence the *maximum prefix subsequence* of moves with the *maximum prefix sum* (Steps 2.5 and 2.6). That is, the gains of the moves in the maximum prefix subsequence give the maximum decrease in the cutsize among all prefix subsequences of the moves tentatively performed. Then, we permanently realize the moves in the maximum prefix subsequence and start the next pass if the maximum prefix sum is positive. The partitioning process terminates if the maximum prefix sum is not positive, i.e., no further decrease in the cutsize is possible, and we then have found a locally optimum partitioning. Note that moves with negative gains, i.e., moves which increase the cutsize, might be selected during the iterations in a pass. These moves are tentatively realized in the hope that they will lead to moves with positive gains in the following iterations. This feature together with the maximum prefix subsequence selection brings the *hill–climbing* capability to the KL–based algorithms.

Figure 3 illustrates the pseudo-code of the move gain computation algorithm for a vertex u in the hypergraph. In this algorithm, $part(v)$ for a vertex $v \in \mathcal{V}$ denotes the part which the vertex belongs to, and $\sigma_n(t)$ counts the number of pins of net n in part t. Move of vertex u from part s to part t will decrease the cutsize if and only if one or more nets become internal net(s) of part t by moving vertex u to part t. Therefore, all other pins ($|n|-1$ pins) of net n should be in part t. This check is done in Step 3.3.1.

5 Experimental Results

Level 2 SN hypergraph partitioning heuristic is implemented in *C* language on Sun 1000E (60MHz SuperSparc processor) for experimenting the performance of the proposed hypergraph models on the decomposition of LP problems selected from `NETLIB` suite [4]. Table 1 illustrates the properties of the LP problems used for experimentation. Tables 2–4 illustrate the performance results for the row-net model (RN), column-net model with dual LP approach (CN-D), and

Table 1. Properties of the constraint matrices of the selected NETLIB LP problems

name	M	N	Z	z^r_{max}	z^r_{avg}	z^c_{max}	z^c_{avg}
perold	625	1376	6018	37	9.63	16	4.37
sctap2	1090	1880	6714	24	6.16	6	3.57
ganges	1309	1681	6912	84	5.28	13	4.11
ship12s	1151	2763	8178	49	7.10	6	2.96
sctap3	1480	2480	8874	31	6.00	6	3.58
bnl2	2324	3489	13999	82	6.02	8	4.01
ship12l	1151	5427	16170	75	14.05	6	2.98

column-net model with block transformation (CN-T), respectively. In Table 1, M, N and Z denote the number of rows, columns, and nonzeros in the constraint matrices, respectively. Here, z^r (z^c) represents the number of nonzeros in the rows (columns) of a constraint matrix.

The proposed hypergraph representations of the selected constraint matrices

Table 2. Average decomposition results for the row-net model (RN)

name	k	Master Problem		Sub-Problems						exec. time
		$M\%$ (σ)	$Z\%$ (σ)	min $M\%$	max $M\%$	min $N\%$	max $N\%$	min $Z\%$	max $Z\%$	(secs)
perold	2	19.2 (2.79)	37.7 (7.26)	35.1	45.7	45.4	54.6	25.1	37.1	1.40
	4	47.3 (4.83)	73.6 (3.82)	7.8	18.8	22.4	27.5	4.1	9.5	1.80
	6	59.0 (4.06)	81.8 (2.66)	3.8	10.7	14.9	18.4	1.6	5.0	3.23
	8	68.8 (2.19)	87.8 (1.35)	1.0	7.8	11.1	13.9	0.4	3.2	3.27
sctap2	2	9.7 (2.13)	31.1 (6.58)	41.2	49.1	46.1	53.9	30.6	38.3	1.88
	4	15.6 (0.57)	46.3 (0.72)	19.3	22.9	22.7	27.4	12.1	14.7	3.25
	6	17.0 (0.84)	47.8 (0.75)	12.4	15.2	14.9	18.4	7.7	9.7	5.83
	8	19.0 (1.24)	49.6 (1.00)	9.0	11.3	11.2	13.8	5.5	7.1	8.40
ganges	2	10.0 (1.32)	23.1 (1.72)	41.1	48.8	45.6	54.4	34.6	42.3	1.30
	4	15.2 (1.70)	27.8 (2.00)	18.4	23.7	22.5	27.4	14.6	21.4	3.90
	6	18.1 (1.73)	30.3 (2.30)	11.4	16.0	14.9	18.4	8.4	14.8	6.42
	8	20.7 (2.55)	33.8 (3.86)	7.5	12.1	11.1	13.8	4.9	11.3	9.20
ship12s	2	15.8 (0.52)	71.3 (1.54)	40.4	43.9	45.1	54.9	12.1	16.6	1.38
	4	22.9 (2.05)	80.4 (2.48)	16.3	25.2	22.5	27.5	3.9	6.2	3.35
	6	29.1 (1.70)	87.4 (1.88)	9.4	18.6	14.9	18.4	1.7	2.6	3.52
	8	31.7 (0.56)	90.2 (0.60)	6.4	15.9	11.2	13.7	0.9	1.5	2.75
sctap3	2	8.3 (1.35)	29.7 (4.23)	41.9	49.8	45.9	54.1	31.4	38.9	3.58
	4	15.1 (0.77)	43.1 (0.84)	19.2	23.2	22.6	27.4	12.6	15.9	4.50
	6	17.5 (1.06)	45.7 (1.18)	12.3	15.4	15.0	18.3	7.9	10.4	8.62
	8	19.4 (1.51)	47.5 (1.38)	8.8	11.4	11.2	13.8	5.6	7.7	11.85
bnl2	2	14.0 (1.71)	41.6 (5.04)	38.8	47.2	45.2	54.8	24.6	33.8	5.75
	4	21.9 (0.80)	60.5 (1.31)	15.3	24.5	22.5	27.4	7.8	11.8	9.35
	6	24.6 (1.95)	64.8 (2.22)	7.4	17.1	14.9	18.4	3.7	7.6	15.18
	8	28.5 (2.31)	69.6 (2.53)	4.1	13.2	11.2	13.8	1.8	5.7	20.98
ship12l	2	16.7 (0.14)	70.3 (0.19)	40.1	43.2	45.0	55.0	13.2	16.6	3.67
	4	25.2 (3.79)	74.7 (2.00)	13.6	24.8	22.5	27.3	4.7	7.5	9.62
	6	59.9 (2.76)	92.4 (1.40)	3.0	14.7	15.0	18.4	0.3	2.2	11.90
	8	66.9 (2.54)	95.8 (1.27)	1.9	12.5	11.2	13.8	0.1	1.1	14.32

Table 3. Decomposition results for column-net model with dual LP approach (CN-D)

name	k	Master Problem		Sub-Problems						exec. time (secs)
		$M\%$ (σ)	$Z\%$ (σ)	min $M\%$	max $M\%$	min $N\%$	max $N\%$	min $Z\%$	max $Z\%$	
perold	2	19.5 (2.23)	26.5 (3.46)	33.7	46.8	45.3	54.7	30.2	43.3	0.97
	4	29.5 (2.21)	39.4 (3.16)	13.5	21.4	22.4	27.6	10.8	19.3	1.93
	6	33.4 (2.07)	44.6 (3.07)	7.3	14.9	14.7	18.5	5.2	13.2	3.35
	8	36.2 (1.83)	48.7 (2.43)	5.1	11.1	11.1	13.9	2.9	9.5	4.58
sctap2	2	16.0 (2.34)	21.9 (3.19)	36.8	47.2	45.4	54.6	32.8	45.3	1.02
	4	32.5 (2.52)	44.2 (3.31)	14.0	19.6	22.4	27.5	10.6	17.0	2.25
	6	37.8 (2.62)	51.3 (3.39)	7.7	12.5	14.8	18.4	5.1	10.5	3.42
	8	40.8 (2.14)	55.2 (2.68)	5.3	9.1	11.1	13.8	3.2	7.3	5.03
ganges	2	9.4 (3.11)	13.0 (7.16)	40.4	50.2	45.7	54.3	36.7	50.3	1.50
	4	30.6 (1.63)	58.5 (4.32)	14.8	21.2	22.5	27.4	7.5	16.8	2.42
	6	33.8 (1.62)	63.8 (4.07)	9.2	13.0	14.9	18.3	4.6	10.2	3.88
	8	35.8 (1.27)	66.5 (2.96)	6.5	9.7	11.1	13.8	3.2	7.1	5.85
ship12s	2	9.5 (2.19)	10.1 (2.23)	33.9	56.6	38.1	52.4	33.7	56.1	1.27
	4	16.1 (5.30)	17.0 (5.36)	13.3	28.5	17.2	27.2	13.2	28.1	3.33
	6	17.9 (6.38)	19.0 (6.48)	5.9	20.3	9.8	18.3	5.8	20.0	5.20
	8	19.8 (6.19)	21.0 (6.28)	3.1	15.7	6.8	13.8	3.1	15.5	7.70
sctap3	2	16.7 (2.71)	22.9 (3.70)	36.7	46.6	45.8	54.2	33.4	43.7	1.82
	4	31.9 (1.99)	43.4 (2.65)	13.9	20.0	22.5	27.6	10.7	17.5	3.33
	6	36.9 (2.18)	49.9 (2.84)	8.1	12.4	14.9	18.3	5.8	10.4	4.92
	8	39.6 (1.69)	53.5 (2.16)	5.6	9.3	11.1	13.8	3.5	7.6	7.25
bnl2	2	11.5 (2.85)	13.2 (3.53)	37.8	50.7	44.1	54.0	34.3	52.4	3.75
	4	19.7 (2.71)	23.4 (3.65)	14.8	25.9	21.8	27.3	12.0	26.8	9.07
	6	23.3 (3.26)	27.9 (4.33)	8.4	17.7	14.4	18.3	6.0	18.4	14.38
	8	26.4 (3.48)	32.0 (4.56)	5.3	13.6	10.6	13.8	3.4	14.4	22.70
ship12l	2	1.8 (1.68)	2.0 (1.69)	40.7	57.6	38.8	51.7	40.6	57.4	3.65
	4	8.1 (5.77)	8.5 (5.80)	15.1	29.7	17.6	26.9	15.1	29.6	7.17
	6	8.9 (5.14)	9.4 (5.18)	8.5	20.6	10.7	18.2	8.4	20.5	10.95
	8	12.5 (4.13)	13.0 (4.16)	4.2	16.0	6.7	13.8	4.2	16.0	15.50

are partitioned into $k = 2, 4, 6, 8$ parts by running the level 2 SN algorithm. The maximum imbalance ratio is selected as $\varepsilon = 0.1$. In Tables 2–4, SN heuristic is executed 40 times for each hypergraph partitioning instance starting from different, random, initial partitions. Tables 2–4 display the averages of these runs. In Tables 2–4, $M\%, N\%$, and $Z\%$ denote the percent ratios of the number of rows, columns, and nonzeros of the master problem (subproblems) to the total number of rows, columns and nonzeros of the overall constraint matrix, respectively. Minimum and maximum values of these percent ratios are displayed for the subproblems. In Tables 2 and 3, σ values denote the standard deviations of the respective averages. In Table 4, $+M\%$,$+N\%$ and $+Z\%$ denote the percent increases in the number of rows, columns, and nonzeros, respectively, due to the column-linking rows added during the block transformation. Hence, $M\%, N\%$, and $Z\%$ values in Table 4 correspond to the percent ratios to the respective sizes of the enlarged constraint matrix.

As seen in Table 2, RN model yields promising results for `sctap2`, `ganges`

Table 4. Decomposition results for column-net model with transformation (CN-T)

name	k	Increase in the Problem Size			Master Problem		Sub-Problems						exec. time
		$+M\%$	$+N\%$	$+Z\%$	$M\%$	$Z\%$	min $M\%$	max $M\%$	min $N\%$	max $N\%$	min $Z\%$	max $Z\%$	(secs)
perold	2	43.3	19.7	9.0	30.1	8.2	31.9	38.0	44.5	55.5	40.2	51.6	1.00
	4	84.3	38.3	17.5	45.6	14.9	12.2	15.0	20.9	29.1	17.0	25.9	1.75
	6	109.0	49.5	22.6	52.1	18.5	7.1	8.9	13.6	19.7	9.7	17.4	3.23
	8	127.6	58.0	26.5	56.0	20.9	4.9	6.1	10.0	15.5	6.4	13.7	4.60
sctap2	2	28.4	16.5	9.2	22.1	8.4	35.5	42.4	45.5	54.5	40.6	51.0	1.15
	4	82.9	48.1	26.9	45.2	21.2	12.3	15.1	22.7	27.7	16.5	22.7	2.27
	6	109.8	63.6	35.6	52.2	26.2	7.1	8.8	14.7	18.6	9.7	14.4	3.50
	8	128.0	74.2	41.5	56.1	29.3	4.9	6.1	10.8	14.2	6.5	10.8	5.33
ganges	2	11.4	8.9	4.3	10.2	4.1	41.1	48.7	45.8	54.2	41.9	54.0	1.52
	4	84.6	65.9	32.0	45.8	24.3	12.2	14.8	22.4	27.9	14.1	27.4	2.55
	6	120.0	93.4	45.4	54.5	31.2	6.8	8.3	14.5	18.9	8.2	18.9	3.80
	8	146.9	114.4	55.7	59.5	35.7	4.5	5.6	10.5	14.8	5.6	14.6	6.10
ship12s	2	23.1	9.6	6.5	18.5	6.1	36.9	44.5	41.1	58.9	37.8	56.2	1.45
	4	42.3	17.6	11.9	28.8	10.5	16.0	19.6	16.7	33.4	14.4	30.0	3.33
	6	44.7	18.6	12.6	30.2	11.1	10.3	12.9	8.9	23.6	6.8	21.2	5.12
	8	53.8	22.4	15.1	34.3	13.0	7.3	9.1	5.7	19.4	4.3	16.8	7.97
sctap3	2	27.4	16.3	9.1	21.4	8.4	36.0	42.6	46.1	53.9	41.1	50.5	2.05
	4	77.2	46.1	25.8	43.5	20.5	12.7	15.6	22.3	27.7	16.6	23.0	3.17
	6	100.7	60.1	33.6	50.1	25.1	7.4	9.2	14.8	18.8	10.0	14.8	5.08
	8	116.5	69.5	38.8	53.8	28.0	5.2	6.4	10.8	14.3	6.8	10.9	8.05
bnl2	2	16.8	11.2	5.6	14.3	5.3	39.3	46.5	43.8	56.2	38.4	56.3	3.92
	4	39.7	26.4	13.2	28.3	11.6	16.2	19.7	20.2	29.9	15.6	28.5	9.10
	6	53.4	35.6	17.7	34.7	15.0	9.8	12.0	12.2	21.5	8.1	20.4	14.80
	8	62.1	41.4	20.6	38.1	17.0	6.9	8.5	9.0	16.9	5.5	16.0	22.88
ship12l	2	6.9	1.5	1.0	5.8	1.0	43.5	50.7	42.7	57.3	42.2	56.8	3.58
	4	35.0	7.4	5.0	23.6	4.6	17.2	21.0	17.2	32.2	16.3	30.7	7.40
	6	39.3	8.3	5.6	26.1	5.2	11.0	13.6	11.4	22.2	10.4	21.0	11.28
	8	56.4	12.0	8.0	34.3	7.3	7.2	9.1	6.7	17.1	6.0	15.9	16.00

and `sctap3` problems. In the decomposition of these problems, $M\%$ values for the master problems remain below 21% for all k. As seen in Table 3, CN-D model gives promising results for `ship12s` and `ship12l` problems. In the decomposition of these problems, $M\%$ values for the master problems remain below 20% for all k. As expected, CN-T model produces master problems with large $M\%$ values but small $Z\%$ values in general. The results of CN-T model for `ship12s`, `bnl2` and `ship12l` problems seem to be promising. These experimental results do not favor any model, since the performance of different models vary on different problem instances due to their inherent structures.

A close examination of Tables 1–4 reveals a correlation between the performance of the SN algorithm and the net degrees of the hypergraph models of the constraint matrices besides their inherent structures. Here, degree d_n of a net n is the number of pins (vertices) connected to net n. In Table 1, z^r_{avg} and z^c_{avg} correspond to the average net degrees of the constraint matrices in the RN and CN models, respectively. For example, in the RN model, the average net degrees of `perold` ($d_{avg} = 9.63$) and `ship12l` ($d_{avg} = 14.05$) problems are much larger than those of the other problems displayed in Table 1. As seen in Table 2, the performance of the SN algorithm deteriorates on these two problems. Similarly, in the CN-D model, the average net degrees of `ship12s` ($d_{avg} = 2.96$) and `ship12l` ($d_{avg} = 2.98$) problems are much smaller than those of the other problems displayed in Table 1. As seen in Table 3, SN algorithm shows much better performance on these problems than the other problems.

It is well known that the performance of the KL-based algorithms deteriorates on hypergraphs with large net degrees. In fact, multi-level gain concept in SN algorithm is proposed as a remedy to this problem. In SN algorithm, higher level gains should be used in tie-breaking with increasing net degrees. However, memory requirement of SN algorithm drastically increases with increasing level number. The aim of this paper was an initial experimentation of the proposed hypergraph models for decomposition. We are currently investigating the performance of other hypergraph partitioning heuristics for this application.

6 Conclusion

Decomposition of constraint matrices of LP problems was investigated to obtain block angular structures for scalable parallelization. Hypergraph models proposed to represent LP constraint matrices reduce the decomposition problem to the well-known hypergraph partitioning problem. A Kernighan-Lin based multiway hypergraph partitioning heuristic was implemented for experimenting with the proposed hypergraph models. Promising results were obtained in the decomposition of the LP problems selected from NETLIB.

References

1. G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.
2. M. C. Ferris, and J. D. Horn. Partitioning mathematical programs for parallel solution. Technical report TR1232, Computer Sciences Department, University of Wisconsin Madison, May 1994.
3. C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.
4. D. M. Gay, "Electronic mail distribution of linear programming test problems" *Mathematical Programming Society COAL Newsletter*, 1985.
5. S. K. Gnanendran and J. K. Ho. Load balancing in the parallel optimization of block-angular linear programs. *Mathematical Programming*, 62:41–67, 1993.
6. B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. Technical Report 2, The Bell System Technical Journal, Feb. 1970.
7. B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Transactions on Computers*, 33(5):438–446, 1984.
8. D. Medhi. Bundle-based decomposition for large-scale convex optimization: error estimate and application to block-angular linear programs. *Mathematical Programming*, 66:79–101, 1994.
9. S. S. Nielsen, and S. A. Zenios. A massively parallel algorithm for nonlinear stochastic network problems. *Operations Research*, 41(2):319–337, 1993.
10. L. A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–81, 1989.
11. D. G. Schweikert and B. W. Kernighan. A proper model for the partitioning of electrical circuits. In *Proceedings of the 9th ACM/IEEE Design Automation Conference*, pages 57–62, 1972.

This article was processed using the LaTeX macro package with LLNCS style