Multilevel Cooperative Search: Application to the Circuit/Hypergraph Partitioning Problem

Min Ouyang
Department of CS&E, University of Nebraska-Lincoln, mouyang@cse.unl.edu

Michel Toulouse
Department of Computer Science, University of Manitoba, toulouse@cs.umanitoba.ca

Krishnaiyan Thulasiraman
School of Computer Science, University of Oklahoma, thulasi@cs.ou.edu

Fred Glover
Hearin Center for Entreprise Science, University of Mississippi, fglover@bus.olemiss.edu

Jitender S. Deogun
Department of CS&E, University of Nebraska-Lincoln, deogun@cse.unl.edu

Abstract In this paper, we present an adaptation for hypergraph partitioning of the multilevel cooperative search paradigm first introduced by Toulouse, Thulasiraman, and Glover [13]. We also introduce a new approach for coarsening hypergraphs, and describe a parallel implementation of this algorithm on the SGI O2000 system. Experiments on ISPD98 benchmark suite of circuits show, for 4-way and 8-way partitioning, a reduction of 3% to 15% on hyperedge-cut compared to hMETIS. Bisections of hypergraphs based on our algorithm also outperforms hMETIS, although more modestly.

Published in *Proceedings of the International Symposium on Physical Design*,San Diego, CA, pp. 192-198, ACM Press, 2000.

1. INTRODUCTION

Hypergraph partitioning is an important and well-studied research area in VLSI CAD. Several classes of heuristics have been proposed to address this problem [**?**]. Recently, the multilevel paradigm has been applied to the hypergraph partitioning problem [8]. This approach has established itself as the current state-of-the-art technique both for the computational time and for the quality of the hyperedge-cuts.

In [13], a new parallel cooperative search algorithm for graph partitioning is proposed where the coarsening phase and the projection operator of multilevel algorithms are used. Cooperative search is a paradigm commonly used to design *move-based* search heuristics for parallel and distributed computer systems [3; 6; 12]. Parallel algorithms based on this paradigm rank among the best approaches for some of the most difficult constraint satisfaction problems and global optimization problems, but these algorithms tend to be unstable. The innovation of [13] is to address this issue by adapting the coarsening phase of multilevel algorithms to cooperative algorithms. This provides a general framework for designing cooperative algorithms which are more predictable. Applied to a suite of well known graph partitioning problems, this new algorithm has been able to obtain new best edge cuts for all problems tested.

This paper introduces the design of a cooperative multilevel hypergraph partitioning algorithm (CoMHP). This design is based on a new approach for coarsening hypergraphs which helps to obtain better hyperedge-cuts as well as to improve the run time of the algorithm. This coarsening approach allows for asynchronous parallel computation, but the hypergraphs are not necessarily related level by level as in traditional multilevel algorithms.

The rest of the paper is structured as follows. Section 2 provides a brief introduction to cooperative search. Section 3 addresses a few key issues of multilevel algorithms. Section 4 describes the design of the cooperative multilevel hypergraph partitioning algorithm. Section 5 reports the results of the tests conducted on the ISPD98 benchmark suite of circuits. Finally Section 6 concludes with some suggestions for future work.

2. COOPERATIVE SEARCH

Cooperative search uses move-based heuristics. These heuristics are iterative methods where a solution $x(t)$ is obtained by a simple perturbation of the solution from the previous iteration $t-1$. The sequence of solutions $x(0), x(1),-\ldots, x(\text{last\_iteration})$ visited by move-based heuristics is often pictured as a walk in the solution space. Kernighan-Lin (KL) heuristic [10] as well as meta-heuristics such as tabu search and simulated annealing use move-based heuristics.

The design strategy of cooperative algorithms is bottom-up. A set of different sequential move-based heuristics is first selected. At run time, each heuristic is a process (a walker) exploring the solution space of the same problem instance. The next step consists of defining a *cooperation scheme* among the search processes. The cooperation scheme establishes: 1) which information about the search performed by a process could be relevant to other search processes, 2) when this information should be shared and 3) which processes should be involved in each exchange of information.

Let $f(x_i)$ and $f(x_j)$ be respectively the cost of the best partitionings found by process p_i and p_j. A cooperation scheme could be based on an exchange of partitioning taking place asynchronously between processes p_i and p_j when $|f(x_i) - f(x_j)|$ is greater than some threshold E_P. Assume $f(x_j) < f(x_i)$ so that the best partitioning of p_j is not as good as for process p_i. Information is exchanged by p_i sending its partitioning x_i to p_j. Then p_j moves from $x(t)$ to x_i which becomes its current solution at iteration $t+1$. We call such a move a *cooperation-based transition* to distinguish it from *move-based transition* where transitions from $x(t)$ to $x(t+1)$ are based on the local move-based heuristic. Once a cooperation-based transition is completed, each process resumes its walk in the solution space according to its move-based heuristic.

Cooperation-based transitions interact in a very complex manner with the behavior of move-based heuristics. Once a walker p_j resumes its search from a partitioning provided by a cooperation-based transition, it does so according to a partitioning that p_j may have never reached if it had performed uniquely move-based transitions. As the number of cooperation-based transitions increases, it comes a point where the behavior of the walkers can no longer be related to their local move-based heuristic. In this case the computation performed by the cooperative algorithm becomes "emergent". *Emergent computation* [1] is the collective computation of a system of interacting computational entities. For example, attractor based artificial neural networks such as the Hopfield network [7] get computation out of interactions among very simple computing entities. Similarly, after some point, cooperation-based transitions take precedence on move-based transitions in controlling the behavior of the cooperative algorithms. But search processes are far more complex computing entities than the nonlinear functions of artificial neural networks. Therefore the emergent behavior of cooperative algorithms is highly unstable. To make the performances of cooperative algorithms more reliable, one needs to study the impact of cooperation-based transitions on move-based heuristics in a controlled environment. The levels of multilevel algorithms provide a framework to design more tractable cooperation schemes.

3. MULTILEVEL ALGORITHMS: ISSUES

Let $H_0 = (V_0, E_0)$ be the hypergraph reduction of a given circuit instance such that V_0 is a set of n vertices and E_0 is a set of m hyperedges where each hyperedge is a subset of V_0. The k-way hypergraph partitioning is an optimization problem where V_0 is partitioned into k subsets $s_1, s_2, \ldots s_k$ so as to minimize a cost function $f(x)$ subject to a balance criterion $\frac{|V_0|}{c_1 k} \leq |s_i| \leq \frac{c_1 |V_0|}{k}$ for some constant $c_1 \geq 1.0$. The vector $x \in X_0 \subseteq I^n$ is a partitioning where $x[i] = s_j$, the subset to which vertex v_i is mapped in partitioning x. The set X_0 is the solution space, i.e., the set of possible k-way partitionings that exist for hypergraph H_0. The cost function $f(x)$ is given by $f(x) = \sum_{i=1}^{m} w(e_i)$, where $w(e_i) = 1$ if e_i is a hyperedge that spans multiple elements (subsets) of the partition of V_0 and $w(e_i) = 0$ otherwise.

We identify two phases in multilevel algorithms: a coarsening phase and a partitioning phase. Coarsening is to merge vertices from a given hypergraph H_0 to yield an equivalent hypergraph $H_1 = (V_1, E_1)$ where $v_i \in V_1$ is a cluster of merged vertices from hypergraph H_0. In the coarsening phase, coarsening is applied iteratively to the latest coarsened hypergraph to obtain a sequence $H_1, H_2, \ldots, H_l$

of coarsened hypergraphs. The coarsening ratio $\frac{|V_{i-1}|}{|V_i|}$ varies according to the coarsening algorithm used. For example, some coarsening algorithms are based on a maximal matching strategy, where clusters of H_i result from the merging of two randomly selected vertices from H_{i-1}. In this case the coarsening ratio $\frac{|V_{i-1}|}{|V_i|} \approx 2$, i.e., $|V_i| \approx \frac{n}{2^i}$, $i \leq l$. Let $X_i \subset X_0$ be the *search space* of the partitioning problem for hypergraph H_i, $i > 0$. A ratio such as $\frac{n}{2^i} = n_i$ for the number of vertices in hypergraph H_i yields a search space for H_i of size

$$|X_i| \approx \frac{1}{k!} \left(\begin{array}{c} n_i \\ \frac{n_i}{k} \end{array} \right) \left(\begin{array}{c} n_i - \frac{n_i}{k} \\ \frac{n_i}{k} \end{array} \right) \cdots \left(\begin{array}{c} 2\frac{n_i}{k} \\ \frac{n_i}{k} \end{array} \right) \left(\begin{array}{c} \frac{n_i}{k} \\ \frac{n_i}{k} \end{array} \right)$$

$= \frac{1}{k!} \frac{n_i!}{(\frac{n_i}{k}!)^{\frac{n_i}{k}}}$ which is substantially smaller than the size of solution space $|X_0| \approx \frac{1}{k!} \frac{n!}{(\frac{n}{k}!)^{\frac{n}{k}}}$. Therefore, the coarsening phase of multilevel algorithms is followed by a partitioning phase, where the substantially smaller size of the search space of X_l is used to scan as exhaustively as possible the whole solution space of the hypergraph partitioning problem H_0. The best partitioning found in this way is propagated to the next level as a solution to iteration $t(0)$ of the move-based heuristic used in this level. This process is repeated for hypergraphs $H_{l-2}, H_{l-3}, \ldots, H_0$.

The multilevel partitioning strategy can be defeated in at least two ways. Let $\frac{\sum_{i=1}^{n_i} f(x_i)}{n_i}$ be the *average hyperedge-cuts* of the solution (search) space X_i. Assuming a coarsening strategy based on maximal matching, it is likely that coarsening will have no impact on the distribution of the cost function in coarsened hypergraphs (low and high cost partitionings are removed from search spaces with equal probability during the coarsening phase). Therefore the average hyperedge-cuts of search space X_i, $i \leq l$ will be approximately constant. This in turn implies that maximal matching coarsening smoothes the cost function by driving out low and high cost partitionings. The smoothing action of the coarsening phase can destroy optimal valleys from search spaces, which will then makes impossible for a move-based heuristic to identify good regions of the solution space. There is no easy fix to this problem. Move-based heuristics using multilevel searches, such as k-exchange neighborhood heuristics [2] and simulated annealing, increase the number of levels and spend more time to search each level. This fix is time consuming.

Unlike maximal matching, a coarsening phase may not just aims at reducing the size of hypergraphs, it may also tries to help the partitioning phase by populating the search spaces with good partitionings. In this case the average hyperedge-cuts might not be constant across the search spaces if, for example, the coarsening strategy is biased by the cost function. Low cost partitionings might have proportionally more representatives in search spaces X_i than in the solution space X. But then the partitioning phase could be trapped in local optima. This problem also occurs in genetic algorithms when they are trapped by a premature convergence due to an unfavorable distribution of good but non-optimal solutions in a population.

Reducing the size of the search space is not by itself a sufficient condition to successfully identify optimum regions of the solution space. Populating search space H_i with good but not optimal partitionings may just drag the search into a poor local optimum of the solution space.

4. CoMHP ALGORITHM

From a cooperative search perspective, the explicit generation of search spaces by multilevel algorithms has two advantages over the implicit levels of simulated annealing: it is time efficient and the search spaces are stable. We propose to use the explicit search spaces as defined by the coarsening phase of a multilevel algorithm as a tool to restrict the search performed by move-based heuristics to a small volume of the solution space.

Search spaces are static and may contain only poor local optima. However, the relation between levels provides the opportunity to design cooperation schemes capable to make incremental changes to search spaces. These changes can be used to regain the landscape of the cost function by reducing the average hyperedge-cuts of the search spaces or to escape from a local optimum in the set of walkers. This is one of the main objective of the cooperation scheme that we are using in the cooperative multilevel approach. We now describe the coarsening strategy we use to generate the search spaces and the cooperation scheme of this algorithm.

4.1 CoMHP Coarsening Strategy

We know that in the context of the k-way partitioning problem, coarsening strategies like maximal matching can be used to obtain a k-way partitioning of a hypergraph H_0. One need only to generate a sequence of coarsened hypergraph $H_1, H_2, \ldots H_l$ such that $|V_l| = k$. Each vertex of H_l is then treated as a subset of a partitioning for H_0. Conversely, k-way partitioning algorithms can be used to obtain the sequence $H_1, H_2, \ldots, H_l$ by calling the k-way partitioning algorithm with $k = |V_i|$. This approach to coarsening based on a partitioning algorithm will be referred to as *partition-based coarsening*.

Generally, *matching-based coarsening* algorithms use strategies such as *edge-coarsening*, *hyperedge-coarsening*, and *modified-hypergraph-coarsening* [4; 8] to reduce the size of the hypergraph. In our cooperative multilevel algorithm, the sequence of coarsened hypergraphs is generated using the hMETIS k-way hypergraph partitioning software [9]. For example, the coarsening phase of our algorithm uses a $\frac{n}{2}$-way partitioning to get H_1, $\frac{n}{2^2}$-way partitioning to get H_2, etc. We stop coarsening if $k \leq 200$ vertices are produced or if the number of hypergraphs exceeds 10. This number is purely arbitrary, it was chosen because of the limited capacities of our parallel computer system.

There are two strategies to compute a k-way partitioning using multilevel algorithms: a divide-and-conquer approach which first calculates a 2-way partitioning and computes recursively a 2-way partitioning on each of the original subsets; and a direct approach which calculates directly a k-way partitioning. Partition-based coarsening can use both approaches. If the partitioner is based on recursive bisection, we ask the partitioner to compute a k-way partitioning where $k = \frac{n}{2}$, the number of vertices of the first coarsened hypergraph in the sequence $H_1, H_2, \ldots, H_l$. The hypergraph H_1 corresponds to the $\log_2 n - 1$th bisection of the partitioner. For the other hypergraphs $H_2, H_3, \ldots, H_l$, we use respectively the partitionings of the bisections $\log_2 n - i$, $1 < i \leq l$. In this case all vertices in hypergraph H_i are composition of two vertices from H_{i-1}, hypergraphs are related level by level. This is not necessarily the case when the partitioner uses a direct approach to calculate a k-way partitioning. Our coarsening strategy uses the direct version

of hMETIS, for a multilevel structure with l levels, we call hMETIS l times, each time with the appropriate k value. Here vertices of H_0 that constitute a vertex of a coarsened hypergraph H_i can be distributed among several vertices in any hypergraph $H_j,\ i < j$.

Whether we use a direct or a bisection k-way partitioning algorithm for the coarsening phase, the construction of the search spaces is biased by the cost function of the partitioning problem. We have done some preliminary tests with coarsening strategies not biased by the cost function. We always found better partitionings during the initial partitioning phase as well as during the search phase when hypergraphs are coarsened by a partition-based coarsening strategy (compared to matching-based coarsening strategies). Partition-based coarsening appears to generate coarsened hypergraphs which have fewer hyperedges spanning several vertices compared to those obtain by matching-based coarsening. In this case, the average hyperedge-cuts of search spaces should not be constant and it should be lower compared to a matching-based coarsening. Most likely, the partitioning phase will be initiated from search spaces with several local optima. For standard multilevel algorithms, having fewer hyperedges might be a handicap since it may be trapped during the uncoarsening phase in the local optima of the search spaces. Our multilevel algorithm uses cooperation between levels to implement mechanisms to lower or increase the average hyperedge-cuts of one or several hypergraphs so as to focus or distance the search from local optima of the solution space (as does most metaheuristics).

4.2 CoMHP Cooperation Scheme

Like any cooperative algorithm, our cooperative multilevel algorithm is inherently parallel (distributed). The parallelization is achieved by associating each hypergraph to a process (processor). Exchanges of information can only occur between two adjacent processes, and two processes are adjacent if they compute the partitionings of two adjacent hypergraphs in the sequence $H_0, H_1, \ldots, H_l$. For example, process p_i is adjacent to processes p_{i-1} and p_{i+1}. Exchanges of information are initiated by the internal state of a process, therefore interactions among processes take place asynchronously. There are 3 categories of exchanges of information: *destroy* operator, *create* operator and *interpolation* operator.

In order to define formally these operators, let X_{it} and V_{it} be respectively the search space and the set of vertices of hypergraph H_i at iteration t of process p_i. Let $\overline{t_i}$ be the maximum number of iterations executed by process p_i. Given that the computation is asynchronous and the size of the hypergraphs are different, the maximum number of iterations $\overline{t_i}$ will be different for each process p_i. Therefore we define $T = \max\{\overline{t_0}, \overline{t_1}, \ldots, \overline{t_l}\}$ as the maximum number of iterations executed by any of the processes involved in the parallel computation. Let $P(V_0)$ be the power set of V_0. We define $\mathcal{V} = \sum_{t=0}^{T}\{V_0t \cup V_1t \cup \ldots \cup V_lt\} \subseteq P(V_0)$, the set of vertices in the multilevel structure that are created during the parallel computation ($V_{it} = \emptyset$ for $t > \overline{t_i}$). Similarly, we define $\mathcal{X} = \sum_{t=0}^{T}\{X_{1t} \cup X_{2t} \cup \ldots \cup X_{lt}\} \subseteq X_0$ be the set of partitionings in the search spaces generated during the parallel computation ($X_{it} = \emptyset$ for $t > \overline{t_i}$).

The sets $\mathcal{X}$ and $\mathcal{V}$ are finite, they can be enumerated. Therefore a vertex $v \in V_i$ corresponds to a set that has index v in $\mathcal{V}$, and for any value of t, the set $\mathcal{V}$ reflects the vertices that exist in hypergraphs $H_0, H_1, \ldots, H_l$. Furthermore, two vertices $v_p, v_q \in V_i$ iff $v_p \cap v_q = \emptyset$. Similarly, $x \in X_i$ is the set that has index x in $\mathcal{X}$.

We identify by s_j^i the subset j of partitioning $x_i \in X_0$. Let $X'_{i-1} \subset X_0$ be a set of partitionings of hypergraph H_{i-1} such that the average hyperedge-cuts

$$\frac{\sum_{j=1}^{n'_{i-1}} f(x_j) \in X'_{i-1}}{n'_{i-1}} < c\left(\frac{\sum_{j=1}^{n_{i-1}} f(x_j) \in X_{i-1}}{n_{i-1}}\right)$$

for some real constant $c < 1$, $n'_{i-1} = |X'_{i-1}|$ and as usual $n_{i-1} = |X_{i-1}|$. *Destroy* and *create* operators are based on the subset $X'_{i-1,t} \subset X_{i-1}$ which represents the n'_{i-1} best partitionings that have been found by move-based heuristics in hypergraph H_{i-1} at iteration t of process p_i.

4.2.1 Destroy Operator

The *destroy* operator of process p_i gets good partitionings from hypergraph H_{i-1} and destroys the vertices of hypergraph H_i that overlap subsets of those good partitionings in H_{i-1}. Therefore, this operator first identifies all the vertices $v \in V_i$ that satisfy $v \cap s_p \neq \emptyset, v \cap s_q \neq \emptyset$ for one $x \in X'_{i-1,t}$ for which $s_p, s_q \in x$. Let D_{it} be the set of vertices $v \in V_i$ that overlap more than one subset s of a partitioning in $X'_{i-1,t}$. The objective of the *destroy* operator is to modify the vertices of D_{it} such that they do not overlap the subsets of at least one partitioning in $X'_{i-1,t}$. Consequently, for each $v \in D_{it}$, the operator seeks a $x \in X'_{i-1,t}$ that has at least two subsets overlapped by v. For example, if v overlaps $s_1^1, s_2^1 \in x_1$, then v is replaced by two vertices $v_1, v_2 \in V_i$ such that $v_1 \subset s_1^1$ and $v_2 \subset s_2^1$. The vertices v_1, v_2 are the indexes for V_i of two sets in $\mathcal{V}$ and obviously $v_1 \cap v_2 = \emptyset$ since $v_1 \subset v$ and $v_2 \subset v$.

Usually we limit the set $X'_{i-1,t}$ to only two partitionings from H_{i-1} each time the *destroy* operator is executed. There are no hard arguments to help define the size of $X'_{i-1,t}$. We know that if $|X'_{i-1,t}|$ is too large, we risk to include partitionings in $X'_{i-1,t}$ that are not that good, particularly at the beginning of the computation. The execution of the *destroy* operator if followed by the execution of a local search on the repaired hypergraph H_i.

4.2.2 Create Operator

The *create* operator of process p_i gets good partitionings from hypergraph H_{i-1} and creates new vertices in H_i by merging vertices of H_{i-1} that are often in the same subsets of all partitionings in $X'_{i-1,t}$. For example, let $X'_{i-1,t} = \{x_1, x_2, x_3\}$, *create* looks for a vertex $v \in V_{i-1}$ such that $v \subset s_1^1, v \subset s_1^2, v \subset s_1^3$. Let $C_{it} \subseteq V_{i-1}$ be the set of vertices in V_{i-1} that are in the same subset of all partitionings in $X'_{i-1,t}$. The objective of the *create* operator is to create new vertices in V_i by merging vertices from C_{it}. To do that, *create* follows two criteria: 1) seeks two vertices $v_p, v_q \in C_{it}$ such that v_p, v_q are in the same subset s in all partitionings $x \in X'_{i-1,t}$; 2) the two vertices are in the same hyperedge.

Let v_{new} be a new vertex of V_i following the merger of v_p, v_q. Then $v_{new} \cap v_z \neq \emptyset$ for some vertex $v_z \in V_i$ because $v_p \subseteq v_z$ or $v_p \subseteq v_z$. The *create* operator ended by a remapping of vertex $v_z \in V_i$ to a new set in $\mathcal{V}$ if v_z fails the test $v_{new} \cap v_z = \emptyset$ (in other words, $v_z = v_z \setminus v_{new}$). The *create* operator tends to reduce the number of vertices in a hypergraph, this balance the effect of the *destroy* operator which tends to increase the number of vertices.

4.2.3 Interpolation Operator

The *interpolation* operator of process p_i uses the current best partitioning of H_{i+1} as an initial solution for a move-based search of hypergraph H_i. Let x be the best partitioning of hypergraph H_{i+1} at iteration t of process p_i. Because of the coarsening strategy used in CoMHP, it is possible that vertices in H_i will overlap several subsets of partitioning x. A split of these vertices in H_i need to be performed. Let $s_1, s_2, \ldots, s_k$ be the subsets of partitioning x. The *interpolation* operator of process p_i looks for every vertex $v \in V_i$ that overlaps more than one subset of the partitioning x. Let $v \in V_i$ be such a vertex overlapping subsets s_p, s_q of x. Then two vertices $v_p, v_q \in V_i$ are created in the following manner: $v_p = v \cap s_p$ and $v_q = v \cap s_q$. Following the split, a search procedure is applied to hypergraph H_i using an initial partitioning that reflects the partitioning obtained from H_{i+1}.

4.2.4 The main loop of CoMHP

The main loop of each process p_i consists of 4 major steps in the following sequence: *destroy* operator, *interpolation* operator, *create* operator and global search.

```
CoMHP( ); /* process p_i */
Compute an initial partitioning;
While not terminated {
  Apply Destroy operator
  Execute move-based searches (FMS and PFM [5]) on H_i;
  Apply Interpolation operator
  Execute move-based searches based on good partitionings of H_{i+1};
  Apply Create operator
  Execute move-based searches on H_i;
  GlobalSearch( ) {
    If number of vertices < 500
      do random search;
    else execute hMETIS; }
  Save p_i's good local and global partitioning results;
} (end outer loop)
End CoMHP
```

We use different local and global search algorithms. Local search algorithms start a search based on an existing partitioning, while global search algorithms first generate a partitioning and then perform a local search. We use two local search algorithms in CoMHP. One is the Sanchis algorithm (FMS) [11], the other is the multiway partitioning by free moves (PFM) proposed by A. Dasdan and C. Aykanat [5]. We also used two algorithms for the global search. One is a random search algorithm, where an initial partitioning is generated randomly, followed by the execution of a local search to refine this partitioning. The random search algorithm is used for high levels, for coarsened hypergraphs having less than 500 vertices. We use the multilevel k-way hypergraph partitioning algorithm [9] as another global search algorithm in CoMHP.

4.2.5 How the Cooperation Scheme Works

In traditional multilevel partitioning, the multilevel hierarchy $H_0, ..., H_l$ is computed and then remains static as iterative refinement takes place. The primary difference between this approach and ours is that the hypergraphs $H_1, ..., H_l$ change dynamically during the optimization. The operators *create*, *destroy*, *interpolate* guide the perturbations of the hypergraphs based on solutions generated so far. *Destroy* operator finds those vertices in V_i that overlap subsets of good partitionings in X_{i-1} and destroy them. *Create* operator populates hypergraph H_i with good vertices (in term of permutations). This in turn helps the *interpolation* operator to initiate move-based heuristics in interesting regions of X_{i-1}. One can imagine that only so many iterations are necessary to find an excellent solution for hypergraph H_i, but if the coarsening wasn't fantastic, then H_i might not even be the right hypergraph to optimize. By letting H_i evolves into something else, we in effect, enable more detailed exploration of the solution space at high levels in the hierarchy.

Table 1: Number of vertices in each coarsened level and the percent of vertices of level i that are distributed in more than one vertices in level $i+1$ after coarsening and after partitioning for IBM16.NET

	after coarsening		after partitioning	
Hgraphs	#vertices	%	#vertices	%
H_0	183483	0.00%	183483	0.00%
H_1	91647	33.74%	75941	42.83%
H_2	45823	59.08%	39894	64.12%
H_3	22911	80.08%	21085	78.12%
H_4	11391	90.27%	11242	81.86%
H_5	5631	93.16%	6253	77.55%
H_6	2815	95.35%	3790	69.31%
H_7	1407	97.73%	2227	60.71%
H_8	639	98.12%	1451	45.76%

One can see the evolution of the hypergraphs in Table 1. In this table, column 1 identifies the hypergraphs and column 2 the number of vertices in hypergraph H_i after coarsening. Since our coarsening strategy does not maintained a level by level relationship as in traditional multilevel algorithms, column 3 indicates the percentage of vertices in hypergraph H_i that spreads in more than one vertex in hypergraph H_{i+1} after coarsening. According to the definitions in Section 4.2, $v \in H_i$ is a set in $\mathcal{V}$. So a vertex $v \in H_i$ spreads in more than one vertex of H_{i+1} if $v \cap v_p \neq \emptyset$ for more than one vertex $v_p \in H_{i+1}$. For example, H_0 has no vertex spreading in more than one vertex in H_1 since $v \in H_0$ is a set with a single element in $\mathcal{V}$. Naturally, as we go toward the end of the hierarchy of hypergraphs, vertices correspond to much larger set in $\mathcal{V}$, which increases the probability that they will be spreading in more than one vertex in the hypergraph of the next level.

Columns 4 and 5 provide the same information as for columns 2 and 3 once the partitioning phase has ended. These two columns show the evolution of the hypergraphs due to the cooperation scheme. For example, rows of hypergraphs H_3 to H_8 show a tendency to restore a level by level relationship among the hypergraphs. Less than 46% of vertices of H_8 are spreading in more than one vertex in hypergraph H_9 (the last hypergraph in the hierarchy). This evolution is caused by the *create* operator which merges two vertices of level $i-1$ to form a vertex of level i. This process is more acute in higher levels because hypergraphs are smaller, more iterations of the outer loop are executed, which in turns imply more interactions through the *create* operator among the levels. Not shown in Table 1, is a similar tendency from the partitioners to find better partitionings at all levels as the three operators evolved the hypergraphs.

One can have a glimpse at the emergent behavior of this

cooperative algorithm through this restoration of the level by level relationship. This is considered to be an emergent behavior because nowhere in the code of the processes there are instructions to restore this relationship. The restoration is a consequence of the complex interactions among the hypergraphs created by the cooperation scheme. By combining multilevel algorithms with cooperative search, we have been able to design a cooperation scheme that still allowed emergent behaviors while the combined multilevel search spaces act as an attractor on the dynamics of a cooperative search. The system (the cooperating walkers) just don't have access to states outside what is allowed by the set of levels. We expect the behavior of cooperative algorithms to be more predictable when they are based on this design. The experimental results for the hypergraph partitioning problem show such good and stable performances.

5. EXPERIMENTAL RESULTS

We have evaluated the performance of our CoMHP algorithm on the ISPD98 benchmark suite of netlists [?]. We compare the performance of CoMHP with the latest version of the hMETIS partitioning package. We have implemented our parallel hypergraph partitioning algorithm and hMETIS at the RCF (Research Computing Facility) of the University of Nebraska-Lincoln. RCF possesses a shared memory SGI O2000 system with 16 250Mhz R10k CPUs, 4GB main memory, and runs on the IRIX 6.5 Operating System. For each problem instance, we have executed 10 runs of hMETIS with recursive bisection and 10 runs with hMETIS-Kway (the direct approach) [9]. Our algorithm has been run for 10 iterations of process p_0. Hypergraph H_0 is the largest one in the sequence of hypergraphs, therefore process p_0 takes more time than any other process to complete one iteration of the search phase.

Tables 2 and 3 present the 2,4,8-way hyperedge-cuts for respectively the unit cell area and the non-unit (real) cell area with CoMHP (Co) and hMETIS (hM). Out of the 108 tests executed, hMETIS outperforms or have the same results as CoMHP in 8 instances, CoMHP outperforms hMETIS for 100 instances. For 2-way partitioning, the improvements of CoMHP over hMETIS are not significant. For 4-way and 8-way partitioning, CoMHP can get up to a 15% hyperedge-cuts improvements over hMETIS. For hMETIS, Tables 2 and 3 report the best solution of bisection or hMETIS-Kway. In 102 cases, hMETIS with bisection found the best solution while hMETIS-Kway got the 6 other instances.

Tables 4 and 5 present the runtime of both algorithms. These runtimes are parallel computational time. For CoMHP, the runtime indicates the total time to run 10 iterations of p_0 plus the time to perform the coarsening phase. For hMETIS we report the time to execute 1 run of the bisection approach in order to factor the use of several processors by CoMHP, this biases the results little bit in favor of hMETIS given that CoMHP uses 10 processors only for a few problem instances.

In Tables 4 and 5, on average hMETIS is 20 to 25 time faster than CoMHP for the 108 tests. But currently, the amount of improvement in the hyperedge-cuts of CoMHP is not significant after 3 or 4 iterations of the search phase by process p_0. This is most likely caused by our coarsening algorithm which is strongly biased by the cost function. The search phase starts in good regions of the solution space, causing CoMHP to converge rapidly toward good partition-

Table 2: Min-cut 2,4,8-way partitioning results with up to a 10% deviation from exact partitioning, cells are assigned unit area (Columns "hM" and "Co" stand respectively for hMETIS and CoHMP).

Circuit	2-way		4-way		8-way	
	hM	Co	hM	Co	hM	Co
IBM01	180	180	495	430	750	711
IBM02	262	262	616	560	1841	1483
IBM03	953	950	1682	1619	2402	2219
IBM04	529	530	1689	1597	2778	2507
IBM05	1708	1697	3024	2888	4306	3874
IBM06	889	890	1484	1465	2275	2204
IBM07	849	824	2188	2036	3308	3098
IBM08	1142	1140	2363	2241	3469	3240
IBM09	629	620	1670	1606	2659	2474
IBM10	1256	1249	2283	2164	3761	3305
IBM11	960	960	2321	2196	3433	3160
IBM12	1881	1872	3730	3520	5972	5384
IBM13	840	832	1661	1671	2717	2483
IBM14	1891	1816	3278	3097	5060	4263
IBM15	2598	2619	5019	4591	6623	5960
IBM16	1755	1709	3816	3745	6475	5360
IBM17	2212	2187	5395	5194	8695	7960
IBM18	1525	1521	2881	2810	5169	4435

Table 3: Min-cut 2,4,8-way partitioning results with up to a 10% deviation from exact partitioning, cells are assigned non- unit (actual) area.

Circuit	2-way		4-way		8-way	
	hM	Co	hM	Co	hM	Co
IBM01	217	215	343	340	606	573
IBM02	266	247	470	399	833	762
IBM03	707	608	1348	1220	1981	1879
IBM04	440	438	1321	1209	2408	2241
IBM05	1716	1681	3002	2895	4331	3950
IBM06	367	363	1149	1056	1716	1688
IBM07	716	721	1539	1480	2918	2707
IBM08	1149	1120	2143	1992	3330	3120
IBM09	523	519	1418	1334	2337	2079
IBM10	769	734	1845	1636	3098	2751
IBM11	697	688	1893	1699	2948	2768
IBM12	1975	1970	3577	3402	4957	4762
IBM13	859	832	1698	1568	2439	2298
IBM14	1520	1494	3048	2869	4833	4360
IBM15	1786	1771	4435	4314	6111	5756
IBM16	1681	1639	3562	3149	5580	5146
IBM17	2252	2156	4824	4393	8222	7003
IBM18	1520	1520	3104	2941	4833	4416

Table 4: Run-time performance for min-cut 2,4,8-way partitioning with up to a 10% deviation from exact partitioning, cells are assigned unit area.

Circuit	2-way		4-way		8-way	
	hM	Co	hM	Co	hM	Co
IBM01	0.2	5	0.3	7	0.5	11
IBM02	0.4	10	0.7	12	1.1	21
IBM03	0.4	16	0.8	17	1.1	25
IBM04	0.5	16	1.0	19	1.3	26
IBM05	0.7	18	1.2	24	1.6	30
IBM06	0.6	21	1.2	23	1.7	33
IBM07	1.1	32	2.0	38	2.6	53
IBM08	1.6	36	2.6	51	3.4	59
IBM09	1.0	34	2.0	40	2.6	58
IBM10	2.2	56	3.5	65	5.0	91
IBM11	1.5	50	3.0	59	3.9	78
IBM12	1.9	62	4.6	73	5.1	115
IBM13	2.0	60	3.6	72	5.1	100
IBM14	5.9	79	9.1	141	13.0	169
IBM15	6.6	121	11.0	176	14.1	217
IBM16	7.6	142	13.3	192	19.0	238
IBM17	9.4	219	17.1	196	22.2	374
IBM18	7.7	178	15.1	192	20.4	301

Table 5: Run-time performance for min-cut 2,4,8-way partitioning with up to a 10% deviation from exact partitioning, cells are assigned non-unit (actual) area.

Circuit	2-way		4-way		8-way	
	hM	Co	hM	Co	hM	Co
IBM01	0.2	6	0.3	7	0.5	11
IBM02	0.3	10	0.7	13	1.0	20
IBM03	0.4	11	0.8	19	1.2	26
IBM04	0.5	16	0.9	18	1.3	26
IBM05	0.6	18	1.2	23	1.6	35
IBM06	0.5	15	1.2	22	1.7	35
IBM07	1.0	29	2.0	41	2.7	54
IBM08	1.2	25	2.2	35	3.1	57
IBM09	1.1	40	1.8	45	2.6	65
IBM10	1.7	52	3.4	64	4.9	93
IBM11	1.4	44	2.7	53	4.4	88
IBM12	2.0	58	3.8	75	5.1	113
IBM13	1.9	53	3.7	71	4.9	113
IBM14	6.0	81	9.0	145	13.0	151
IBM15	5.6	111	12.0	160	14.2	197
IBM16	6.7	168	13.1	197	18.0	264
IBM17	11.2	243	18.2	286	23.8	354
IBM18	8.7	189	15.9	235	20.5	296

ings. Consequently, our choice of 10 iterations as stopping criteria does not reflect accurately the current runtime competitiveness of CoMHP. We have used this criteria anticipating that we will be able to improve substantially the long term convergence behavior and the hyperedge-cuts performances of CoMHP in the future. Therefore 10 iterations gives a more stable basis for comparing the runtime of this algorithm with other partitioners. We have also performed tests with hMETIS using 100 restarts and also allowing the same runtime for hMETIS as for CoMHP with 10 iterations. The hyperedge-cuts were not substantially better than with 10 runs, in this regard hMETIS is quite stable.

6. CONCLUSION AND FUTURE WORK

Our work has focused on introducing and testing a new cooperative multilevel hypergraph partitioning algorithm and an associated partition-based coarsening strategy. Our algorithm incorporates cooperative search, which has been a successful paradigm to develop parallel algorithms for constraint satisfaction problems and global optimization problems. We have modified the cooperative search paradigm to integrate the main concepts of multilevel algorithms, a highly successful approach to graph and hypergraph partitioning.

The resulting algorithm produces substantially better 4-way and 8-way partitioning by comparison with hMETIS, but it is slower than this partitioner. Improved runtime performance of our algorithm is possible by modifying local search algorithms used in the search phase. Standard FM like algorithms are developed for refining random partitioning. For already good partitions such those obtained by CoMHP, we do not need to flip all vertices for refinement, but rather stop the search after flipping part (20%, for example) of the vertices. This offers an opportunity to speed-up the search without degrading the quality of partitioning.

We plan to look more closely to the convergence behavior of this algorithm, examining how the current cooperation scheme and coarsening strategy interact with each other to affect convergence. Future research will also examine cooperation schemes involving non-adjacent hypergraphs, for example using an interpolation operator between any hypergraph H_i and hypergraph H_0.

Finally we are planning to apply this cooperative search paradigm to a partitioner capable of partitioning hypergraphs with fixed vertices, to enhance its usefulness in VLSI design.

7. REFERENCES

[1] In Stephanie Forrest, editor, *Emergent Computation.* MIT/North-Holland, 1991.

[2] E.H.L. Aarts and J.K. Lenstra. Introduction. In E. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Oprimization*, pages 1–17. John Wiley & Sons Inc., 1997.

[3] S.H. Clearwater, T. Hogg, and B.A. Huberman. Cooperative Problem Solving. In B.A. Huberman, editor, *Computation: The Micro and the Macro View*, pages 33–70. World Scientific, 1992.

[4] J. Cong and M.L. Smith. A Parallel Bottom-Up Clustering Algorithm with Applications to Circuit Partitioning in VLSI Design. In *Proc. 30th ACM/IEEE Design Automation Conference*, pages 755–760, 1993.

[5] A. Dasdan and C. Aykanat. Two Novel Circuit Partitioning Algorithms Using Relaxed Locking. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 16(2):169–78, Feb. 1997.

[6] T. Hogg and C. Williams. Solving the Really Hard Problems with Cooperative Search. In *Proceedings of*

the 11th National Conference on Artificial intelligence (AAAI93), pages 231–236. AAAI Press, 1993.

[7] J.J. Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79:2554–2558, 1982.

[8] G. Karypis, V. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning: Application in VLSI Domain. *IEEE Transactions on VLSI Systems*, 1998.

[9] G. Karypis and V. Kumar. Multilevel k-way Hypergraph Partitioning. In *Proc. 36th ACM/IEEE Design Automation Conference*. Association for Computing Machinery, 1999.

[10] B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical Journal*, 49:291–307, 1970.

[11] L.A. Sanchis. Multiple-way Network Partitioning. *IEEE Trans. Comput.*, 38(1):62–81, Jan. 1989.

[12] M. Toulouse, T.G. Crainic, and B. Sansó. Self-Organization in Cooperative Tabu Search Algorithms. In *1998 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2379–2385. Omnipress, 1998.

[13] M. Toulouse, K. Thulasiram, and F. Glover. Multi-Level Cooperative Search. In *5th International Euro-Par Parallel Processing Conference, volume 1685 of Lecture notes in Computer Science*, pages 533–542. Springer-Verlag, 1999.