Automated System Partitioning for Synthesis of Multi-Chip Modules

Raghava V. Cherabuddi and Magdy A. Bayoumi
Center for Advanced Computer Studies
University of Southwestern Louisiana
Lafayette, LA 70504

Abstract

We present a system-level partitioning technique for the Synthesis of Multi-Chip Modules. It is based on the Stochastic Evolution heuristic, which is an effective heuristic for solving several combinatorial optimization problems. We perform the partitioning at the behavioral level. The advantage of partitioning at the behavioral level is that both area and time constraints can be taken care of at the system level and also that scheduling/allocation can be applied concurrently to system-level partitioning. We formulate the partitioning problem as an extension to the Network-Bisectioning problem for which the Stochastic Evolution heuristic has been shown to provide better results than the Simulated Annealing technique. Preliminary scheduling/allocation and Pin sharing are also performed simultaneously to estimate the area and pincount of each of the partitions. Efficient partitions are obtained for some of the digital signal processing applications in reasonable CPU time.

1. Introduction

Most of the present and future video and multi-media applications are too large to fit in a single chip. The target architecture will be implemented in several separate chips. A major drawback with such approach is the large inter-chip communication delay. Recently, due to advances in packaging technology an efficient solution to this problem is offered by the Multi-Chip Module (MCM) technology. An MCM has several chips bonded to a single substrate. The interconnections among the chips are provided in this substrate. The inter-chip communication delay on this substrate is much less compared to the inter-chip delay at the board level. The major advantages of MCMs are performance, mixed technologies, compactness, reliability and high yield. In addition, MCMs offer a high-density packaging option to meet the requirements of many high-performance digital signal processing applications.

Although many commercial CAD tools have recently been developed for MCMs, they do not include aspects concerning high-level synthesis. Most of the present high-level synthesis systems/techniques do not target the synthesis onto multi-chip modules. The register-level design obtained from the high-level synthesis of a behavioral description may be too large to fit on a single chip. In such cases, it can be partitioned into several chips to be connected on an MCM. System-Level partitioning is performed at the behavioral level. The goal of System-Level partitioning is to synthesize a multi-chip module from the given behavioral description. The objectives of System-Level partitioning are minimizing the total time the application takes on the synthesized multi-chip module and minimizing the number of interconnects (inter-chip communication) on the substrate, subject to the resource/technology constraints such as maximum area per chip, maximum pins per chip (the pincount constraint) and maximum allowable chips per module.

We present a system partitioning technique for the high-level synthesis of multi-chip modules. It is based on the Stochastic Evolution (SE) heuristic[2]. We formulated the System-Level partitioning problem to be an extension of Network-Bisectioning problem for which the SE heuristic has been shown to provide better results than the Simulated Annealing technique. Preliminary scheduling/allocation and pin sharing are also performed simultaneously with partitioning to estimate the area and pincount requirements.

The partitioning problem and the related research is discussed in Section 2. Section 3 describes the Stochastic Evolution heuristic briefly. Our proposed system partitioner is described in detail in Section 4. The implementation and results are presented in Section 5. Finally the conclusion is given in Section 6.

2. The Partitioning Problem

The system partitioning problem is the process of dividing the control and data flow graph (CDFG) into several partitions such that each partition can be synthesized onto a chip. Partitioning can be performed at various levels of the synthesis process. For example, if partitioning is performed at the behavioral level, several design alternatives
can be explored by scheduling and allocation that can be done simultaneously with partitioning. On the other hand if partitioning is performed after the behavioral synthesis, feasible partitions cannot be generated as information about the partitions will not be available earlier in the synthesis process. This happens due to the inter-chip communication delays which take certain number of control steps. Both these approaches have their own advantages and disadvantages.

Several approaches have been considered in order to solve the partitioning problem at various levels of the synthesis process. The first major work in this area was done by McFarland[8]. It used an hierarchical clustering technique to partition behavioral hardware descriptions. Design constraints were not considered while clustering. Gupta and DeMicheli[11] used the Kernighan-Lin algorithm and the Simulated Annealing technique for partitioning functional models of digital systems by satisfying area and time constraints. The binding (operations to functional units) was done prior to partitioning. Lagunese and Thomas[6] performed partitioning at the architectural level (before register-transfer synthesis) using the concept of multistage clustering technique. CHOP[7] is a constraint-driven partitioner that does partitioning at the behavioral level. Both [6] and [7] do not consider scheduling/ allocation while partitioning. The problem of partitioning is interdependent with the problems of scheduling and allocation. Recently Integer Programming was used to develop a system that did partitioning simultaneously with scheduling/ allocation[9]. While Integer Programming (IP) is an elegant method, it is computationally expensive even for moderately sized problems. For larger problems such as the ones we envision as suitable for MCM implementation, IP based solutions are not very practical.

In this paper we present a partitioning technique that is performed at the behavioral level. Preliminary scheduling/ allocation is performed simultaneously with partitioning to estimate the area of the partitions. Our research shows that many DSP architectures can be efficiently partitioned onto an MCM in reasonable CPU time.

The system-level partitioning problem can be formally stated as follows:

**Given a graph G = <V,E>, partition G into a number of subgraphs P1, P2, ..., Pn (n ≤ the maximum allowable number of chips per multi-chip module) such that A_i ≤ AC, P_{Ci} ≤ PCC, 1 ≤ i ≤ n where A_i is the area of partition i, AC is the maximum area of any chip on the MCM, P_{Ci} is the pincount of partition i, and PCC is the maximum pincount of any chip on the module, while minimizing the number of interconnections among the partitions and finally reducing the length of the schedule.**

3. Stochastic Evolution

Stochastic Evolution[2] is a new fast effective heuristic approach to solve several combinatorial optimization problems. The Stochastic Evolution (SE) approach differs from Simulated Annealing in that it does not accept negative gains initially. Hill climbing starts only when a local minima is reached. This allows the SE algorithm to converge faster and give better results than Simulated Annealing.

The main idea of the SE approach is to find a state S that has the minimum Cost(S). A state is defined as a function S : M → L where M is a finite set of movable elements and L is a finite set of locations. The input to the SE heuristic is an initial state S_0, an initial value of the control parameter p_0, and a parameter R used in the stopping criterion. The SE algorithm retains the state of lowest cost among those produced by a function called PERTURB. A new stage S' is generated by moving some elements in state S. A move can be simple or compound. Given a function S : M → L and a movable element m, a simple move from S with respect to m is just a change in the value of S(m), i.e. a simple move generates a new function S' : M → L such that S'(m) = S(m) while S'(m') = S(m') for all m' ≠ m ∈ M. A compound move is a sequence of simple moves. Define Gain(m) = Cost(S) - Cost(S') as the reduction of cost after the move is performed. The function PERTURB now decides whether or not to accept the move associated with the element being scanned currently. The move is accepted if Gain(m) > r where r is randomly generated based on the control parameter p such that p ≤ r ≤ 0. Note that r ≤ 0 always, hence moves with positive gains are always accepted. The algorithm then goes to scan the next element in M. The UPDATE procedure is responsible for updating the value of the control parameter p. The stopping criterion parameter R is the expected number of iterations the SE algorithm needs until an improvement in cost takes place. Each time a state is found which has a lower cost than the best state so far, SE decrements the counter by R, thereby rewarding itself by increasing the number of its iterations before termination. After the termination, the algorithm returns the best state i.e. with the lowest cost function.

4. System Partitioning

We formulated the System-Level partitioning problem as the Network-n-sectioning problem (an extension of Network-Bisectioning) and applied the Stochastic Evolution heuristic to it. The CDFG is considered as a graph G = <V,E>. This graph is to be divided into several partitions such that the cost of each partition is minimized subject to the satisfaction of area and pincount constraints and provided that the number of partitions is less than or equal to
the maximum allowable chips per module. The cost of a partition is the total number of edges cut by that partition. These edges refer to the inter-chip connections in the case of multi-chip module design. For example the cost of the partition in Figure 1 is one because the partition cuts only one edge i.e. the edge (6,8). This edge also refers to the inter-chip connection between chip1 and chip2 if P1 and P2 are synthesized onto chip1 and chip2 respectively.

![Figure 1](image)

The overall flow of the partitioning algorithm is given in Figure 2. The input to our partitioner is a graph $G = (V,E)$. It is assumed that the given input graph is big enough to be synthesized onto at least two chips. The first phase of the partitioner divides the graph into several partitions based on the objective of minimizing the inter-chip communication and the constraint that the number of partitions is less than or equal to the maximum allowable chips per module. Once the partitions are ready, they are given to the next phase called preliminary scheduling/allocation and pinsharing. Details of these are in Section 4.2. The output of the preliminary scheduling/allocation and pinsharing phase are the area and the pincount of the chip that is to be synthesized from each partition. These values are checked against the input maximum area/pincount constraints on each of the chips to be synthesized. If the constraints are not satisfied the partition of the graph violating the constraints is sent back to the partitioner as shown in Figure 2. Otherwise the chip area/pincount and the contents of the chip are given as the output. For example if partition P1 in Figure 1 does not violate the constraints but whereas P2 does, then only partition P2 is sent back to the partitioner. This time P2 will be divided into more number of partitions than before i.e. two.

The partitioner, the preliminary scheduling/allocation and pinsharing are discussed below.

4.1 Mapping Stochastic Evolution to Partitioning

Using the state model described in Section 3, we identify the finite set of movable elements of a state with the nodes of the CDFG, i.e. $M = V$, and the set of locations of states with the variable number of parts, i.e. $L = \{1, \ldots, i\}$, $2 \leq i \leq \text{max}$, where max is the maximum allowable number of chips per module. A partition therefore is simply an onto function $S : V \rightarrow \{1, \ldots, i\}$ where the $i$ parts of the vertex set split by the partitions are the subsets $S^1(1), S^1(2), \ldots, S^1(i)$. If the partitioning process continues due to the area and pincount limitations of a chip, then the subsets are sent back for partitioning.

In the case of network-n-sectioning, the move associated with each vertex $v$ is a transfer of $v$ to the remaining parts. Here the moves may either be simple or compound. Every vertex, one at a time, is moved from its original partition to the remaining $(i - 1)$ partitions, one at a time, and provide us with a new cost function. If the new cost is less than the old one by $r$ units, then the move is accepted. The output of the partitioner is the partition with the lowest cost i.e. minimal inter-chip connections.

To start with, the partitioner takes the value of $i$ to be 2 and divides the graph into two partitions. Both of these partitions are sent to the preliminary scheduling/allocation phase. If any of the partitions violate the area/pincount constraints then the partition of the graph violating the constraints is sent back to the partitioner. The partitioner in the next iteration takes the value of $i$ to be equal to the number of partitions coming back plus one. It first merges
all the partitions (part of the graph violating the constraints) and then divides it into \( i \) parts (with the new value of \( i \)).

The outline of the partitioning algorithm is given below.

```plaintext
modified_graph = empty;
violating_partitions = 0;
total_partitions = 0;
i = 2; /* min. number of partitions to be made */
Partitioner(G,i,max) /* G: graph;
i:# of partitions;
max:max. # of chips */
{
  if (i <= max)
  {
    Pr[i] = Partition(j,i);
    /* Partition divides the graph G into i partitions and these partitions are stored in Pr */
    for j = 1 to i
    {
      if (PSA(Pr[j]) is ok)
      /* testing whether jth partition satisfies the area/pincount constraints */
      total_partitions ++;
    }
    else
    {
      violating_partitions ++;
      modified_graph = merge(modified_graph, Pr[j]);
    }
  }
  /* end of for loop */
  if (violating_partitions == 0)
  exit;
  else
  Partitioner(modified_graph, violating_partitions + 1, max - total_partitions);
  /* end of if (i <= max) */
else exit;
/* end of Partitioner */
```

4.2 Preliminary Scheduling/Allocation and Pin sharing

This is the second phase of the system partitioner. As the binding of hardware resources is not available at the behavioral level, our approach performs a procedure called preliminary scheduling/allocation (PSA) and pin sharing in order to determine the number of functional units (area) and pins required by the chip that is to be synthesized from each of the partitions. The main objective of PSA is to minimize the length of the schedule.

The PSA is performed as follows. There are two cases to be examined here. Firstly, if the number of partitions obtained is less than the maximum allowable number of chips on an MCM, secondly, if the number of partitions is equal to the maximum number of chips on an MCM. The maximum number of chips on an MCM can either be a technology constraint or an user-defined (hardware constraint) constraint. In the first case an adhoc method for PSA is applied because of no limitation on the hardware resources. However in the second case, where there are a fixed number of chips per module (hardware constraint), the graph has to be effectively partitioned onto these chips. A more sophisticated method of PSA is applied in this case. Both these methods are described below.

In the first case the following adhoc method is applied. The PSA works in a way so as to minimize the number of control steps required i.e. to assign hardware to all parallel operations. Thus the PSA procedure assigns hardware to all parallel operations such as nodes 1 and 2 in P1 (Figure 1) and nodes 3 and 4 and 6 and 7 in P2. Therefore P1 requires 2 ALUs and 1 multiplier and P2 requires 2 ALUs and 2 multipliers. The PSA detects all parallel operations of the same type in the given partition and assigns them the functional units of that type. Operations are not parallel if there exists an edge or a path between them. The maximal number of parallel operations of a particular type gives the maximal number of functional units of that type required by that partition.

For the second case the following sophisticated method is applied. Here the number of partitions is equal to the maximum allowable number of chips on an MCM. The resource constraint MCM scheduling problem is also solved using the Stochastic Evolution heuristic. First of all we need to identify the mapping i.e. the set of movable elements, the set of locations, the constraints, and the cost function.

For each of the partitions, the fixed hardware is given as a function of number of different functional units which fit in that area. The objective of the scheduler is to minimize the length of the schedule, which becomes the cost function for the SE heuristic. We identify the finite set of movable elements of a state with the nodes (VP) in the partition (to be synthesized onto a single chip). The set of locations is identified as follows. For each of the partitions, both ASAP (as soon as possible) and ALAP (as late as possible) schedules are performed before the MCM scheduling is applied. Let \( SCS_i \) and \( LCS_i \) be the starting control step and the last control step of node \( i \) respectively. These values provide us with the set of locations (called as freedom or mobility in some researches) \( L = \{ SCSS_1, LCS_1, SCSS_2, LCS_2, ..., SCSS_n, LCS_n \} \) where \( n \) = number of nodes in a partition.

For every partition, a schedule therefore is simply an onto function \( S : VP \rightarrow \)
{SCS1..LCS1,...,SCSn..LCSn}. The moves are performed in a simple or a compound manner just similar to the partitioner. Every vertex i, 1 ≤ i ≤ n, in the partition is moved from its previous control step to a new one in its set of locations (SCSi..LCSI) one at a time and provide us with a new cost function. The moves should not violate any of the precedence relations or the resource constraints. The state with the lowest cost function is therefore the minimum length schedule.

After obtaining the number of functional units of different types, the PSA adds up all their areas (provided as input) and returns the total area estimated for that partition. This total area is compared with the area constraint (also provided as input) of each chip on the MCM.

Pin sharing is an important method of reducing the number of pins required for a chip. Take the case of P2 in Figure 1. Consider only the external edges coming into the partition. Nodes 3 and 4 are executed in the same control step, so their input pins are all required at the same time. Node 7 requires an input pin in the second control step. Actually i9 need not be an additional pin (provided to the chip. It can be shared with either of the 4 pins i5, i6, i7 or i8. For example if i9 is shared with i8, it acts as input to node 4 in control step 1 and acts as input to node 7 in control step 2. Thus pin sharing reduces the number of pins required by a partition (chip). Pin sharing not only considers the input edges i1...i9 but also the internal edges which are cut by a partition. Take the case of P1, the edge from node 6 to node 8 is an input edge to P1, we can see that it can be shared with i1, i2, i3 or i4. Note that operations that are scheduled in the same control step cannot share pins. The pin sharing procedure detects all of the non-parallel operations (opposite of PSA) and finds if those operations have an external pin. If so, they can be shared. Demultiplexers are required in order to provide pin sharing. The area of the partition thus includes the area of the demultiplexers as well. The total pins required by each of the partitions is compared with the pincount constraint of each chip (provided as input) on an MCM.

5. Implementation and Results

The partitioner (including the PSA and pin sharing) was implemented in C on a SUN SPARC station 2. The partitioner was implemented using the network-bisectioning formulation. Tables 1 and 2 show the partitions obtained for some DSP algorithms. For all these implementations we have assumed the maximal area and pincount of a chip to be 2000 sq. units and 100 pins respectively. The maximum allowable number of chips on an MCM is assumed to be 20. Offcourse any linear combination would work fine. The areas of the ALU, multiplier and the demultiplexer were assumed to be 200, 400 and 100 sq. units respectively. The values of p0 and R were taken to be -2 and 5 respectively. These values were selected after the partitioner was run on several examples with various values. Table 1 shows the partitioning for four different examples. Only the internal edges (excluding external input/output pins) were counted and reported in the table. For simplicity the width of all edges is taken to be 5. Like the values of the area the width of the edge can take any value. Column three in Table 1 shows the minimal number of chips that need to be synthesized. Their individual areas, pincount and the components are given in Table 2. The elliptic filter could have been partitioned into only two chips but one of the chips pincount exceeded 100 pins. If the area is more than 2000 sq. units (all other values remaining same) the chips will be even less and the time taken for partitioning will be even better.

Consider the AR filter in Figure 3. The shaded nodes are mapped onto Chip 1 and the remaining onto Chip 2. The partition cost is three — the edges between (16,19), (17,20), and (25,27). The 2 chips were fully utilized in area. Both the chips contained 2 ALUs and 4 multipliers.

6. Conclusion

We presented an efficient system-level partitioning technique which is based on the Stochastic Evolution heuristic. The partitioning problem was formulated as an extension to the Network-Bisectioning problem for which SE works efficiently. Preliminary scheduling/allocation and Pin sharing were also performed simultaneously to estimate the area and pincount for each of the partitions. The partitioning of some DSP applications was performed efficiently in reasonable CPU times. The resource constraint MCM scheduling and partitioning with network-nsectioning is currently being implemented. Extension of this work will involve the complete High-Level synthesis of multi-chip modules especially for pipeline synthesis, conditional operators, loops, etc.

References


| AR Filter | 28 | 30 | 2 | 0.02 |
| Elliptic Filter | 34 | 47 | 3 | 0.05 |
| DCT | 48 | 64 | 7 | 0.16 |
| Circuit # 4 | 120 | 201 | 16 | 1.23 |

Table 1

```
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
  21  22  23  24  25  26
```

Figure 3

| AR Filter | Area=2000 | Pins=65 | 2 ALUs | 4 MULs |
| Elliptic Filter | Area=1900 | Pins=35 | 4 ALUs | 4 MULs |
| DCT | Area=1800 | Pins=40 | 4 ALUs | 2 MULs |
|       | Area=2000 | Pins=65 | 4 ALUs | 3 MULs |
|       | Area=1900 | Pins=45 | 4 ALUs | 1 MUL |
|       | Area=1900 | Pins=45 | 3 ALUs | 4 MULs |
|       | Area=1800 | Pins=50 | 2 ALUs | 2 MULs |
|       | Area=1800 | Pins=20 | 3 ALUs | 3 MULs |
|       | Area=1700 | Pins=20 | 2 MULs | 5 DEMUXs |

Table 2