



## **Mapping and Partitioning of Task Graphs Using Kernighan-Lin/Fiduccia-Mattheyses Algorithm**

Ashish Mishra<sup>1</sup>, Raja Jimit<sup>2</sup>, Abhijit Rameshwar Asati<sup>3</sup>, Kota Solomon Raju<sup>4</sup>

<sup>1,2,3</sup>Department of Electrical and Electronics Engineering, BITS-Pilani, Pilani, Rajasthan, India

<sup>4</sup>Principal Scientist & Project Leader,

Reconfigurable Computing Systems & Wireless Sensor Network Systems Lab,  
Digital System Group, CSIR-CEERI, Pilani, Rajasthan, India

**Abstract:** Hardware Software partitioning of a task graph refers to the mapping of task nodes to physical components such as processors, Application Specific Integrated Circuits, memory with an optimization of parameters involving execution time, physical area, cost of memory and other factors. This work evaluates the performance of iterative min-cut heuristic Fiduccia-Mattheyses for HW and SW partitioning. The results shows the algorithm can be very effective in giving the bi-partite partition.

**Keywords:** Fiduccia-Mattheyses(FM), Kernighan-Lin, task graph, cell and move, Matlab.

### **I. Introduction**

The system-level partitioning problem refers to the assignment of operations or task nodes of a task graph to hardware or software components. Overall system performance is determined by the effect of hardware-software partition on the utilization of the processor, number of hardware components, bandwidth of the bus between the processor and application-specific hardware, amongst other related factors. Thus a partitioning scheme must attempt to capture and make use of its effect on system performance in making trade-offs between hardware and software implementations of an operation. Typically, the time constraints of the system won't be met if it is all implemented as software on a generic processor, but it would be too expensive to design an application-specific hardware chip for the whole functionality. While pure hardware synthesis tools like Hardware Description Languages allow the implementation of systems on a chip from high-level specification, the resulting cost of implementation is a considerable drawback. On the other hand, off-the-shelf generic processors, and implementation in software, are much cheaper, but seldom suit the hard time constraints imposed on embedded systems. Thus, cost-effective designs should use a mixture of hardware and software to accomplish their goals.

The primary issue with a design-oriented approach, where allocation of tasks to hardware and software precedes the synthesis, is that it is not known whether the final system will meet its requirements. This suggests a synthesis-oriented approach, where constraints on performance and cost of the systems are specified, and a systematic constraint-driven exploration of the design space is done. A partitioning algorithm matches every functional object with a system component object and looks for partitions with constraint optimization. These constraints may be in the form of execution time deadlines, area bounds and number of components to use. There are many approaches to solving the partitioning problem like iterative algorithms, dynamic algorithms, heuristic algorithms and exact algorithms. Partitioning is an NP hard problem, and therefore exact solutions tend to be quite slow for bigger dimensions of the problem. Among the well-known heuristic algorithms, Fiduccia-Mattheyses, Genetic Algorithm (GA), Simulated Annealing (SA), Tabu Search Algorithm, Multiway Greedy Algorithms and Ant Colony Optimization (ACO) are most common. Besides the heuristic algorithms referred to above, sometimes family of heuristics such as hierarchical clustering. Kernighan-Lin heuristics are equally useful for application in the partitioning problem. The Integer Linear Programming (ILP) is an exact algorithm subject to the provision that all system constraints can be expressed as a set of linear inequalities containing independent variables [1][2].

### **II. Fidducia-Mattheyses Algorithm**

Given a circuit with  $n$  elements, we wish to generate a balanced two-way partition of the circuit into sub-circuits of hardware and software. The number of elements in each partition is to be determined dynamically so as to minimize the area i.e., the cost function and a constraint of time are applied so as to ensure good performance.

The definitions used in algorithm are

Cut state of a net: A net is said to be cut if it has cells in both blocks, and is uncut otherwise.

Gain of cell: The gain  $g(i)$  of a cell 'i' is the number of nets by which the cut set would decrease if cell 'i' were to be moved.

Cut set of partition: The cut set of a partition is the number of elements of the set of all nets with cut state equal to cut.

**Base cell:** The cell selected for movement from one block to another is called base cell. It is the cell with maximum gain and the one whose movement will not violate the balance criterion.

Given a partition (A, B) of the cells, the main idea of the algorithm is to move a cell at a time from one block of the partition to the other in an attempt to minimize the cut-set of the final partition. The cell to be moved is called the base cell, is chosen both on the basis of the balance criterion and its effect on the size of the current cut set. Define the gain  $g(i)$  of cell(i) as the number of nets by which the cut set would decrease were cell(i) to be moved from its current block to its complimentary block. Note that a cell's gain may be negative. During each move we must keep in mind the balance criterion to prevent all cells from migrating to one block of the partition.. Thus the balance criterion is used to select the block from which a cell of highest gain is to be moved. It will often be the case that this cell has a non-positive gain. In that case, we still move the cell with the expectation that the move will allow the algorithm to climb out of local minima. After all moves have been made, the best partition encountered during the pass is taken as the output of the pass. To prevent the cell-moving process going into an infinite loop, each base cell is immediately locked in its new block for the remainder of the pass. Thus only free cells are actually allowed to make one move during a pass, until either all cells become locked or the balancing criterion prevents further moves. The best partition encountered during the pass is then returned. Additional passes may then be performed until no further improvements are obtained. In practice this typically occurs quickly, in several passes.

Various steps in the algorithm are:

1. Compute the balance criterion

$$r \cdot \text{area}(V) - \text{areamax}(V) \leq r \cdot \text{area}(V) + \text{areamax}(V)$$

2. Compute gains of all cells.

The gain  $g(i)$  resulting from the movement of cell  $i$  from block A to block B is:

$$g(i) = FS(i) - TE(i).$$

$FS(i)$  = the number of nets connected to cell  $i$  and not connected to any other cell in the block of cell  $i$  and

$TE(i)$  = the number of nets that are connected to cell  $i$  and not crossing the cut.

3. Select base cell that has a maximum gain which satisfies balance criterion. If tie then use Size balance criterion and hierarchy.

4. Lock cell: Fix the base cell Update all cell gains that are connected to critical nets via the base cell.

4. If all cells are fixed, go to Step 5. Otherwise choose next base cell with maximum gain, and move this cell.

Iterate over, going to Step 3

5.:Determine the best move sequence  $c_1, c_2, c_3 \dots (1 < m < i)$ , so that total gain  $G_m$  is maximized.

If  $G_m > 0$ , go to Step 6, Otherwise exit.

6. Make all  $i$  moves permanent. Execute  $m$  moves, reset all fixed nodes. Start with a new pass, go to Step 1.

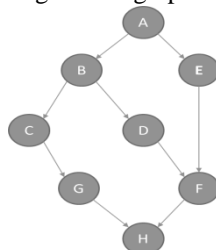
### III. Problem Definition

The task graph shown in the Fig. 1 is to be partitioned into a hardware software mapping subject with a sequential time constraint of 275 units. The objective of the partition is to yield a minimum area mapping. All the tasks have to be run sequentially on the time line. The Area and Time metrics of the different system components are shown in Table I. A maximum of two components are to be used initially with at least one hardware component and at least one software component

Two graphs have been partitioned in this paper and the corresponding results are shown.

Fig. 1 shows the dummy given graph and fig. 2 shows the given architecture for which the mapping has to be carried out. Table 1 shows the parameters given for each node on four given components in the architecture.

Fig.1 Task graph 1



The parameters for a sample input task graph 1 is presented in the table below and the task graph is fig2.

Table1: sample input data

Task	Time				Area			
	CPU1	CPU2	ASIC 1	ASIC 2	CPU1	CPU2	ASIC 1	ASIC 2
A	60	30	20	10	40	60	25	45
B	90	50	30	15	40	60	30	35
C	81	54	27	15	40	60	15	20
D	60	40	20	10	40	60	10	15

E	90	44	30	15	40	60	10	10
F	87	30	27	20	40	60	10	25
G	90	50	40	15	40	60	15	35
H	99	56	33	20	40	60	15	15

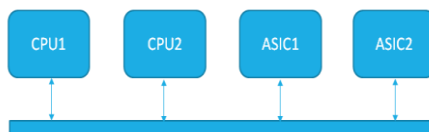


Fig. 2 Testing Architecture

#### IV. Results

The results obtained prove that using FM algorithm the resultant output is very much dependent on the initial partition. We have implemented with a variety of initial partitions and some sample results are shown in the tables below: For task graph 1, Initial Partition1 = {ABCD}, {EFGH}

Table 2 : Partitions with area and time taken

A	B	C	D	E	F	G	H	Area	Time
3	1	1	3	3	3	3	3	125	264
3	3	3	2	3	3	3	3	180	207
4	4	2	4	4	4	2	2	190	185
3	3	4	3	3	3	4	4	155	127

Initial Partition2 = {ABCDEFGH}, {}

Table 3 : Partitions with area and time taken

A	B	C	D	E	F	G	H	AREA	TIME
3	3	3	3	3	3	2	2	160	186
3	3	3	3	3	3	3	2	175	250
4	4	2	2	4	2	2	2	150	255
4	4	4	2	4	2	2	2	170	201
4	4	4	4	4	2	2	2	185	201
3	3	4	3	3	3	4	4	155	147

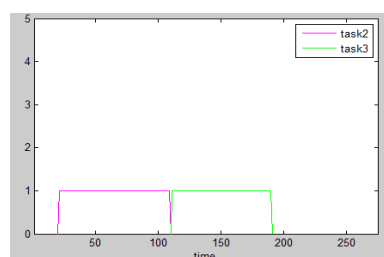
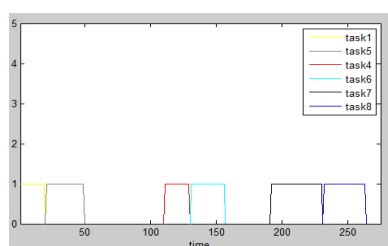


Fig. 3 Scheduling of Graph on ASIC-1 and Scheduling of Graph on CPU-1

Since the results varied according to initial cut\_set. It was better to give better cut\_set with the help of Kernighan-Lin algorithm. Kernighan-Lin is a  $O(n^3)$  heuristic algorithm[2] for solving the graph partitioning problem. The algorithm has important applications in the layout of digital circuits and components in VLSI. Graph partitioning problems arise in a variety of circumstances, particularly in computer science, but also in pure and applied mathematics, physics, and of course in the study of networks themselves. KL algorithm can be found in any CAD tools for VLSI design text book .

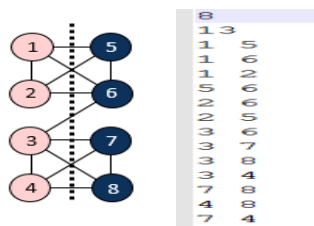


Fig. 4 Input graph and file format

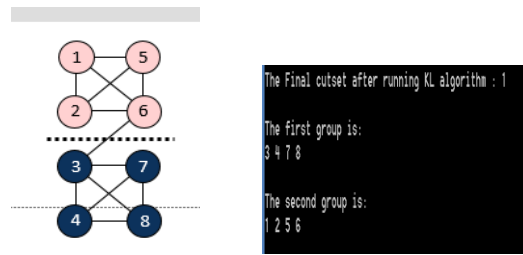


Fig.5 Final output form KL algorithm

```

Since maxTotalGain=0.
The Final cutset after running KL algorithm : 3

The first group is:
1 2 4 5

The second group is:
3 6 7 8
    
```

Fig. 6 Output for the given graph

## V. Conclusion

FM Algorithm was implemented with specific initial cutsets and ratio factors on small task graphs, which give only locally optimum values depending on the provided initial cutsets. The minimum total area value for deadline satisfying partitions found using two initial cutsets was found to be 125 for the partition {BC}, {ADEFHG} using CPU1 and ASIC1. Different initial cutsets may be tried to give a more optimum value of area. Global Heuristic Algorithms have a much better performance than FM and can optimize more than one heuristic, but FM has the added advantage of being simple and less time consuming to get an approximate minimum cost. FM was implemented on directed Task Graphs which have at most one connection with each other. The future work may incorporate hypergraphs too. Also attempts can be made in our work to minimize the time complexity in the implementation of the algorithm.

## References

- [1]. C.M.Fiduccia,R.M.Mattheyses, A linear time heuristic for network partition,19<sup>th</sup> Design Automation Conference.
- [2]. Kernighan, B. W.; Lin, Shen (1970). "An efficient heuristic procedure for partitioning graphs". Bell Systems Technical Journal 49: 291–307.