PACE2: AN IMPROVED PARALLEL VLSI EXTRACTOR WITH PARAMETER EXTRACTION

Krishna P. Belkhale and Prith Banerjee

Computer Systems Group
Coordinated Science Laboratory
University of Illinois
Urbana, IL- 61801

ABSTRACT

In *ICCAD88*, we had proposed an efficient parallel algorithm *PACE* to speed up the first phase of VLSI circuit extraction, called the *net list extraction phase*. In this paper, we describe an algorithm *PACE2* targeted to the second phase of extraction, called the *parameter extraction* phase. We have interfaced two models for resistance and capacitance. In this paper, we also propose a different partitioning scheme, namely by equal number of rectangles, so as to balance the load. The parallel algorithm has been implemented on the Intel iPSC2/D4-MX hypercube.

1. INTRODUCTION

Parallel processing has recently become an extremely attractive and cost effective way of achieving orders of performance improvements in VLSI CAD applications [1, 2]. In this paper, we will describe a parallel algorithm for one particular CAD application, namely, VLSI circuit extraction. We have initially implemented the algorithm on an Intel iPSC2 D4/MX hypercube.

Given a set of rectangles in different mask levels for a VLSI layout as the input, a circuit extractor determines the circuit connectivity and estimates various electrical parameters such as the resistances of lines, capacitances of nodes, and dimensions of transistors. There are two phases in circuit extraction. The first phase is the identification of transistors, determination of electrically connected regions (called nets). This phase is called *net list extraction*. The second phase involves the estimation of the electrical parameters of nets and transistors. This phase is called *parameter extraction.* The output of an extractor can be fed into a circuit, switch or logic level simulator. In [3], we had described a parallel algorithm for the net list extraction phase of extraction. In this paper, we describe how two models for parameter extraction *MEXTRA* [4], and *HPEX* [5], are interfaced. The underlying theme of this paper is to look at the general principles for interfacing any parameter model to the output of the net list extraction phase of *PACE*. The main goal is to achieve a considerable speedup in the parameter extraction phase, which is the most time consuming step in extraction. Also, with the increased computational power, we will be able to use more accurate, and hence more time consuming, parameter models. The second contribution of this paper is a better partitioning scheme for load balancing.

2. REVIEW OF PARALLEL NET LIST EXTRACTION

The parallel circuit extractor, *PACE* [3], divides the circuit into 2^d parts, where d is the dimension of the hypercube. This is done by dividing the circuit into 2^k strips in the x-dimension and 2^{d-k} strips in the y dimension, where k is computed based on the aspect ratio of the circuit. The steps in the net list extraction phase in *PACE* are as follows.

This research was supported in part by the National Science Foundation Award under Grant NSF MIP 87-14646, and in part by an equipment grant NSF CCR 87-05240.

P1) Each processor gets all those rectangles that overlap or abut its assigned region through the host.

P2) The overlaps between diffusion and polysilicon rectangles are determined to give transistor channel rectangles. The channel rectangles are grouped into maximal connected groups, which form the local transistors. All rectangles, other than channel rectangles, are grouped into electrically connected sets which form the local nets. The local nets and transistors are called *LCS* sets. These are labeled so that they are unique across processors with the processor number as a part of the label.

P3) For each processor boundary and layer, including the channel layer, a list of maximal boundary segments is obtained. These segments are basically maximal connected segments formed by intersection of rectangles of the layer with the boundary. Each of the segments is associated with an *LCS* set. From the maximal boundary segments, a single list is constructed for each border by combining the segments for all the layers in some particular order. The list constructed for a border will correspond exactly to the one constructed in the processor on the other end of the border.

P4) The lists constructed in P3, and the *LCS* sets that touch a border, are passed to a merge algorithm. The merge algorithm [3, 6] determines for each of the *LCS* sets a unique label which identifies its global connected component (*GCS*). The label is chosen from the labels of the component *LCS* sets. It also determines the number of *LCS* sets that are part of the *GCS* set. The processor that has the *LCS* set with the unique label is said to "own" the *GCS* set. All *LCS* sets at a processor belonging to the same *GCS* set are combined to form a single *LCS* set.

3. IMPROVED CIRCUIT PARTITIONING

The circuit partitioning in *PACE* was done based on area. For non homogeneous circuits this method of partitioning will lead to load imbalance. This is because load on a processor during net list extraction, and in the subsequent parameter extraction, depends primarily on the number of rectangles assigned to a processor. In order to remedy this situation, we perform partitioning in *PACE2* based on the number of rectangles.

There are two partitioning strategies that could be considered. These are the slice and rectangular partitioning, as shown in Fig. 1. For these partitioning methods, we represent a rectangle by a point in two dimensions, for example the center of the rectangle. The objective then is to divide the domain into regions so that each region has approximately the same number of points. In slice partitioning the domain is divided into slices such that each slice has nearly the same number of points. In the rectangular partitioning method the domain is first divided in the x direction into slices with the same number of points, and each of the slices is in turn divided in the y direction into regions carrying nearly the same number of points. Rectangular partitioning scheme is used in *PACE2* because of lower perimeter of the regions assigned to the processors, which implies lower message complexity for the extraction algorithm. The number of parts the

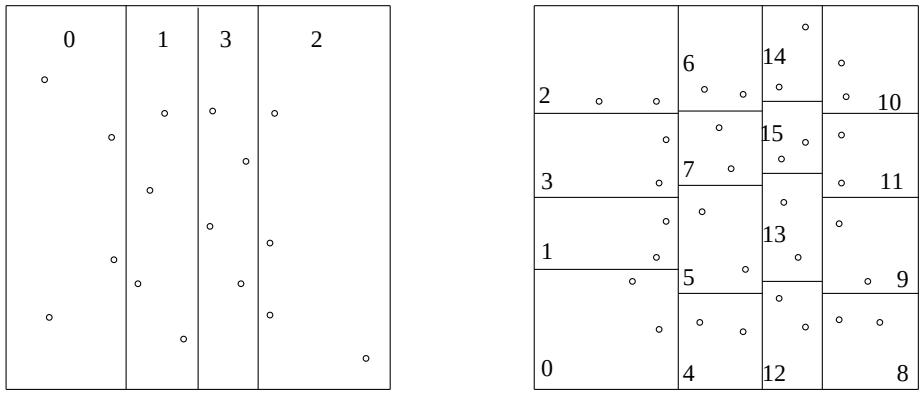


Fig. 1.(a) Slice and (b) Rectangular Partitioning.

region is divided in the x direction and y direction is given by 2^k, 2^{d-k} respectively, where k is computed as before.

The rectangular partitioning method described above could be done by the host or by a node processor after the whole circuit has been read. However, there are two problems with this approach. The first problem is that the host may take a lot of time in doing the partitioning. The second problem is that there may not be enough memory at the host to read in the whole circuit. In *PACE2* this problem is solved by storing the points as a distributed data structure, based on the rectangular partitioning strategy. The host reads the circuit in batches of K rectangles, and issues an add instruction to the distributed data structure for every batch. Addition or deletion of points from the data structure involves restructuring the partitions so that the resulting regions have the same number of points. We have obtained an efficient algorithm for this problem for the case of a general recursive partition. This is described in [7]. In *PACE2* we have applied this algorithm for the special case of rectangular partitioning. In this case, an addition or deletion of w points followed by the restructuring can be done in time $O(dt_\alpha + wdt_\beta + w\log n)$, where t_α, t_β are the start up time and per byte time for communication respectively, and n the number of points in the hypercube. The advantage of this approach is that the load balancing and reading of the circuit can be done in parallel. Since the former is experimentally determined to be faster, the load balancing comes at no extra cost. Once all the rectangles have been read by the host, the nodes go through some post processing steps so that each of them gets all the rectangles that overlap or abut its assigned region. This step is required as rectangles were considered as points during the partitioning step.

4. PARALLEL PARAMETER EXTRACTION

For transistor extraction, an approximate effective length and width is computed in *MEXTRA* [4] and *HPEX* [5]. The effective width is taken as one half of the transistor's perimeter with diffusion. The effective length is then calculated as the area of transistor divided by the effective width. This method is accurate for rectangular transistors, but only approximate for non rectangular transistors.

The rectangles of a net have parasitic capacitance with respect to substrate and also other overlapping or neighboring rectangles. The capacitance of a node (a set of electrically connected rectangles in the same layer) with respect to the substrate can be accurately estimated by $C = \alpha A + \beta P$, where A is the area, P the perimeter of the region, and α and β are constants depending on the layer. In *MEXTRA* only the substrate capacitance is computed for each net by the above formula. However, in *HPEX* both substrate capacitance and resistance is computed.

The most accurate method to determine resistances is to use the Laplace equation for the region of the net. An approximate method is used in *HPEX* for resistance extraction. The first step is to convert the rectangles in the net into a horizontally maximal non overlapping form. The next step is to combine horizontally long rectangles in the vertical direction. In this step, two rectangles, that are sufficiently longer in the x direction than the y direction, are combined vertically if they abut on their horizontal edges. The above two steps derive a set of rectangles for the net which gives an indication of the direction of the current. These two steps are illustrated in Fig. 2.

The next step is to take those rectangles, that are sufficiently longer in one direction than the other, and cut it on its larger side, based on the intersections with other rectangles. Two overlapping rectangles from the intersections are merged into a single one. Subsequently, a rectangle that connects to more than two rectangles, or to only one rectangle, is defined to be a knot. Also, the rectangles associated with the drain, source and gate of any transistor are defined as ports. The rest of the rectangles are defined as branch rectangles. The knots and ports are then assigned unique numbers. A branch of a net is defined to be a path from a knot or port to another knot or port using only branch rectangles. The branches can be determined by a pass of the electrical connectivity graph. The above steps are illustrated for the net of Fig. 2 in Fig. 3.

The next step is to perform the resistance calculations for the branches. Simple analytical formulae are used for this purpose. In this calculation, the resistances of knots and ports are carefully approximated. The calculation for substrate capacitance is done along with the resistance calculation for each branch. The capacitance of a branch is lumped at its endpoints equally. Thus the result of parameter extraction is a distributed RC network for each net. If such a network is directly output, it would result in a large simulation time. In order to avoid this there is a node reduction phase in *HPEX* [8]. This involves finding an approximately equivalent small RC network for each net.

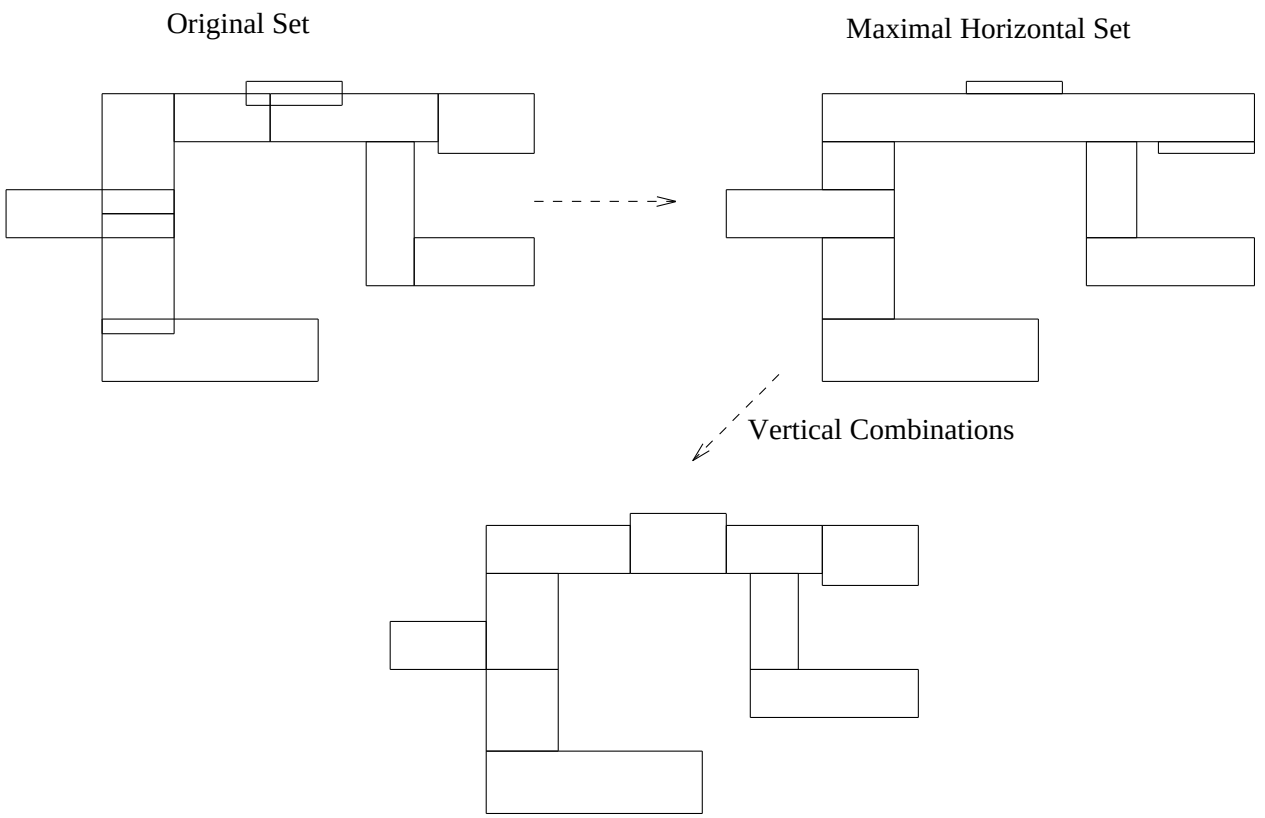
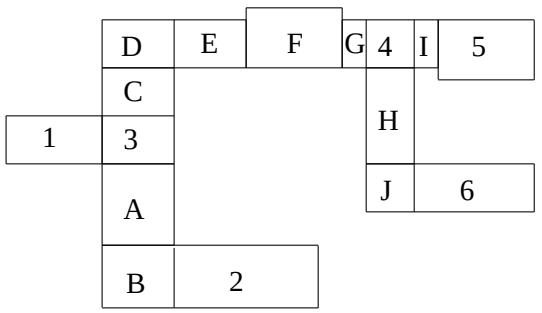


Fig. 2. The first two steps in Resistance Extraction.



1 2 6 5 - Ports or Knots

3 4 - Knots

A B C D E F G H I J - Branch Rects

Branches

1 3 4 H J 6 3 C D E F G 4

3 A B 2 4 I 5

Fig. 3. Rectangle Decomposition and Branch Creation.

Now, we discuss how the above two models are interfaced in *PACE2*. The extraction of transistors is the same in both the approaches. For the local transistors not connected to a boundary, we have all the information for extraction. For the others we do the following. If a local transistor is not owned by the processor, then its channel rectangles, with the connecting diffusion rectangles, are sent to the processor that owns the global transistor. If a local transistor is owned by the processor, it waits for messages to complete the information for the global transistor. Because the number of local transistors that comprise a global transistor is known from the merge algorithm, a processor can determine when it has received all the information for the transistor. The processor can then extract the transistor.

The next step is the extraction of capacitances and resistances of nets. A general approach similar to transistor extraction could be followed. However this approach will lead to lot of load imbalance, as some of the nets like power or ground nets could be very big. In order to solve this problem, the merge algorithm is changed to determine for each *GCS* set, the number of rectangles in its set, similar to the way it determines the number of *LCS* sets in a *GCS* set. A global net with more than *K* rectangles is designated as a long net (*K* can be chosen to be around 100). The other global nets are designated as short nets. For extraction

of short nets we could use an approach similar to transistor extraction. For long nets, we need to design a way to extract them in a split fashion, so that it is as close to the original model as possible. This is the general approach that we have taken for the problem. We will now illustrate this approach for the models of *MEXTRA* and *HPEX*.

The *MEXTRA* model only computes the substrate capacitance for the nets. Now, suppose we have a global net G, comprising of the local nets $L_1, \cdots L_k$. For each of the local nets L_j we can compute its capacitance. This involves converting the rectangles stored in a local net into a non overlapping format, and then finding the contribution to the capacitance from each rectangle. However, the substrate capacitance of G is not just the sum C_{tot} of the substrate capacitances of its local nets, because of the possible overcounting of the perimeter component of the capacitance equation. However, this overcounting can be resolved precisely by sending all the rectangles of L_j that touch the processor border to the processor owning G. The processor owning G can determine all those instances when a border rectangle B_1 of layer l belonging to L_i touches a border rectangle B_2 belonging to L_j of the same layer, where $i \neq j$. For each such instance, we subtract $2\beta\, dist$ from C_{tot}, where β is the perimeter constant of the layer l and $dist$ is the length of the line of contact between B_1 and B_2. With this correction, we get the correct value of the capacitance for the global net G. Because of the accuracy of the method, we extract every global net, even if it is not a long net, in the above split fashion.

The *HPEX* model computes both resistances and capacitances. Let G be a long global net comprising of the local nets $L_1, \cdots L_k$. We now describe how it is extracted in a split fashion. For each of the local nets L_i the two steps of Fig. 2, and the rectangle decomposition step of Fig. 3 is performed. The next step is to determine knot and branch rectangles and assign numbers to the knots. These numbers will contain the processor's address bits concatenated at the end. The rectangles of L_i that touch a border are not classified in this step, though they are counted in the connectivity calculations for other rectangles. The above two steps are illustrated in in Fig. 4(a). The next step is to mark the rectangles in all the branches, that start with a border rectangle, and end with a knot or port or another border rectangle. The marked rectangles are then deleted from L_i and sent to the processor that owns G. This step is illustrated in Fig. 4(b).

The processor that owns G collects all the marked rectangles, some of which are knots, into a set S. The set S is then transformed by the two steps of Fig. 2 and the rectangle decomposition step of Fig. 3. In these steps the rectangles that have been designated as ports or knots are not included. The next step is to determine the new knots in S and number them. This step is illustrated in Fig. 4(c). The complete result is shown in Fig. 4(d). The set S is then sent back to all the processors that contain a part of the global net G. On receiving this, they are included in the local net L_i. Then the branches of L_i are determined. A processor computes the capacitance for each of the knots that are owned by it. It evaluates the resistance and capacitance of a branch if it owns one of the endpoints. The capacitance determined is lumped at the endpoints of the branch owned by the processor appropriately. A processor will report the capacitance of a port or knot only if it owns it. Similarly the resistance of a branch is reported by a processor only if one of the end points is owned by the processor, and the number of the other endpoint is larger than it.

It is easy to interface a node reduction phase, such as [8], at this stage. The node reduction step for short nets or local nets that do not touch a border can proceed as before. However, for long nets node reduction can be done only for nodes that do not connect to a node not owned by the processor. Currently we have not interfaced a node reduction phase, but intend to do it in the future.

5. RESULTS AND DISCUSSION

The parallel circuit extractor *PACE2* has been implemented on the Intel D4/MX hypercube using 8600 lines of C code. In Table 1, we give the performance of the algorithm, ignoring reading and output time, on a number of VLSI custom layouts, for different number of processors. It can be noticed that we have got super linear speedups in some cases. This is mainly because dynamic memory allocation and deallocation takes more time on a single processor when there is not much free memory. Some of the entries in the table are not shown due to memory problems on the hypercube.

In Table 2, we compare the cpu time performance of a version of *PACE2* running on a uniprocessor with *MEXTRA* on Sun 3/280 and Sun4/280 file servers. The Sun4/280 is roughly 2.1 times faster than the single node processor on the hypercube for our application. The sequential implementation of *PACE2* and *MEXTRA* could not run for some of these circuits on Sun 3/280 due to memory limitations. From Table 1 and Table 2, we can estimate the time for the big circuits on a single node of the hypercube. From this calculation, we see that we have obtained speedups in the range 13-14 for the big circuits.

In Fig. 5, we illustrate the effectiveness of the load balancing strategy. We have shown the performance of *PACE2(HPEX)* for Ckt5 with and without load balancing. This graph shows the amount of cpu time spent by each processor. This time excludes the time spent in sending, receiving, and waiting to receive a message. The graph also shows the total time for extraction. It can be seen that the load balancing strategy is very effective. Also, the total time for computation is very nearly equal to the cpu time of the processor that has the maximum cpu time. This implies that the communication overhead is small.

6. CONCLUSIONS

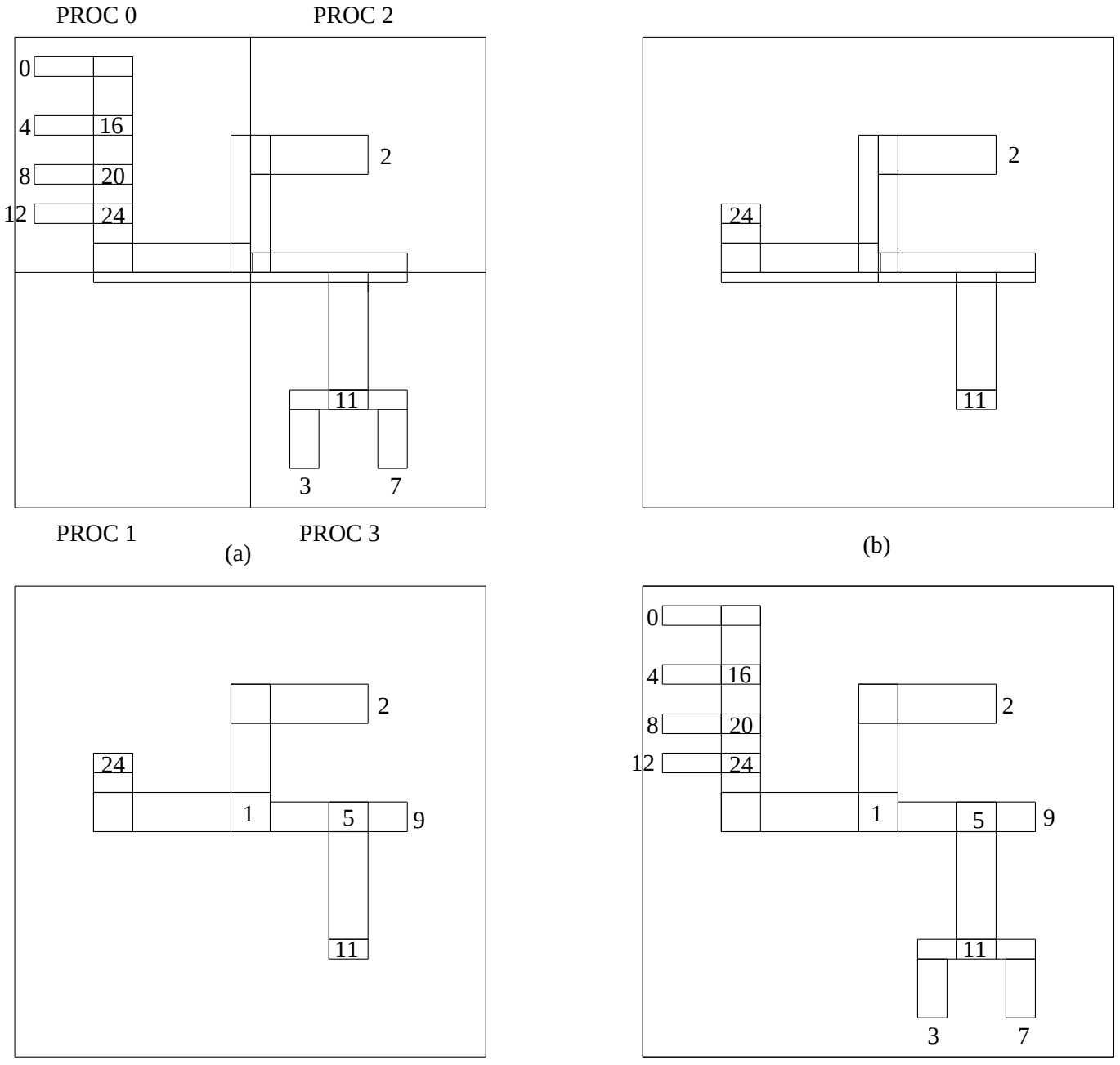
The main conclusion of this research is that with parallel processing we can effectively use more accurate and time consuming extraction methods. This research has provided a general outline of how one would parallelize any such method. Future research for parallel circuit extraction should concentrate on how different extraction models can be interfaced using this general principle.

REFERENCES

[1] Kit Yoke Tham, ''Parallel Processing for CAD Applications,'' *IEEE Design & Test of Computers*, vol. 4, no 5, pp. 13-17, Oct. 1987.

[2] S. Levitin, *A Multiprocessing Approach to Circuit Extraction*. MIT, 1986, Master's Thesis.

[3] K. P. Belkhale and P. Banerjee, ''PACE: A Parallel VLSI Circuit Extractor on the Intel Hypercube Multiprocessor,'' *Int. Conf. on Computer-Aided Design*, Nov. 1988.



(a) Local processing, (b) Rectangles sent to owner (say proc 1), (c) Processing by owner, (d) Final result.

Fig. 4. Interfacing HPEX

[4] D. T. Fitzpatrick, ‘‘MEXTRA: A Manhattan Circuit Extractor,’’ Electronics Research Lab Memo M82/42, UC Berkeley, Jan. 1982.

[5] Shun-Lin Su, Vasant B. Rao, and Timothy N. Trick, ‘‘HPEX: A Hierarchical Parasitic Circuit Extractor,’’ *Proc. 24th Design Automation Conf.*, pp. 566-569, Jun. 1987.

[6] K.P. Belkhale and P. Banerjee, ‘‘Parallel Algorithms for Geometric Connected Component Labeling on a Hypercube Multiprocessor,’’ *IEEE Transactions on Computers*, vol. (submitted).

[7] K. P. Belkhale and P. Banerjee, ‘‘Recursive Partitions on Multiprocessors,’’ *Proc. 5th Distributed Memory Computing Conf.*, pp. 930-938, Apr. 1990.

[8] S. L. Su, V. B. Rao, and T. N. Trick, ‘‘A Simple and Accurate Node Reduction Technique for Interconnect Modeling in Circuit Extraction,’’ *Proc. Int. Conf. on Computer-Aided Design*, pp. 270-273, Nov. 1986.

Circuit	Boxes	Procs	Pace2(Mextra)		Pace2(Hpex)	
			time sec	speedup	time sec	speedup
Ckt1	26000	1	55.4	(1.00)	114.1	(1.00)
		2	27.5	(2.01)	57.3	(1.99)
		4	13.6	(4.07)	29.3	(3.89)
		8	7.6	(7.30)	16.6	(6.87)
		16	5.1	(10.88)	10.3	(11.08)
Ckt2	47000	1	106.1	(1.00)	217.5	(1.00)
		2	53.3	(1.99)	118.3	(1.83)
		4	24.3	(4.37)	53.5	(4.07)
		8	12.3	(8.63)	27.9	(7.80)
		16	8.4	(12.63)	19.5	(11.15)
Ckt3	120000	4	68.6	...	151.1	...
		8	32.3	...	71.4	...
		16	18.1	...	40.5	...
Ckt4	215000	8	58.5	...	128.5	...
		16	29.7	...	71.0	...
Ckt5	320000	8	90.8	...	199.7	...
		16	44.5	...	99.3	...

TABLE 1: Time performance of PACE2 on iPSC2.

Circuit	Sun 3/280			Sun 4/280		
	Mextra sec	Pace2 *Mextra* sec	Pace2 *Hpex* sec	Mextra sec	Pace2 *Mextra* sec	Pace2 *Hpex* sec
Ckt1	68.3	72.4	198.6	33.2	24.6	63.5
Ckt2	135.8	126.6	334.8	63.3	44.3	102.0
Ckt3	345.2	336.9	856.1	165.2	116.3	271.0
Ckt4	...	...	...	305.1	213.8	492.2
Ckt5	...	...	...	434.0	306.7	712.7

TABLE 2: PACE2 sequential implementation.

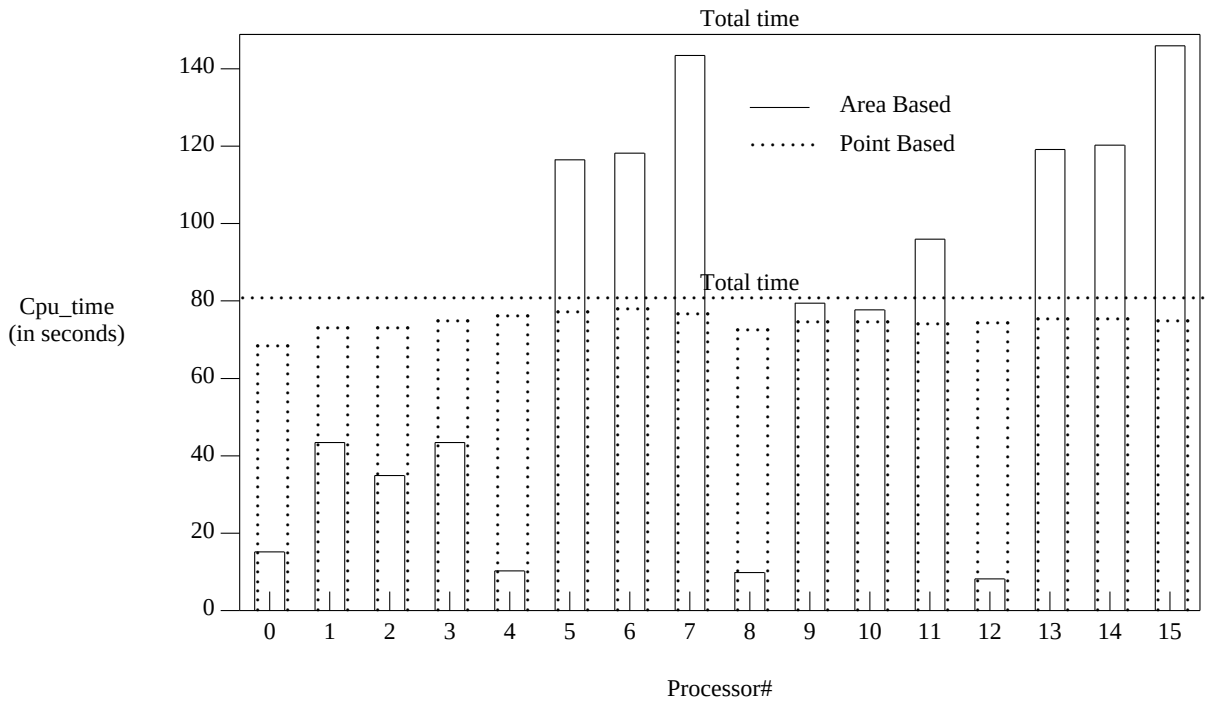


Fig. 5. PACE2(HPEX) with and without load balance