TR--90--001

REPRESENTING LARGE CELL MAPS

REPRESENTING LARGE CELL MAPS

Y. S. Kuo

Institute of Information Science, Academia Sinica

Address for Correspondence:

Dr. Y. S. Kuo

Institute of Information Science

Academia Sinica

Nankang, Taipei, Taiwan

R. O. C.

Tel: 886-2-7822002

Fax: 886-2-7824814

Abstract

Cell maps are widely used in layout algorithms to represent objects in the plane. They are traditionally implemented as 2-dimensional (2-dim) arrays, thus are also referred to as cell arrays. The cell map is usually too large to be stored entirely in the main memory, and must be swapped between the main memory and the hard disks by the host computer's paging mechanism. Routing algorithms based on the 2-dim array cell map can generate a large number of page faults and result in poor performance when the size of the cell map exceeds the capacity of the main memory. In this paper, we propose a new data structure for the cell map in order to eliminate the excess of page faults present in the traditional 2-dim array structure. Experiments have shown that this new representation can effectively eliminate 90% of the page faults. Thus the performance of routing algorithms can be greatly improved.

1. Introduction

Computer aided design (CAD) of VLSI circuits is characterized by the large amount of design data it manipulates. The amount of data is so large that they can not be stored entirely in the main memory. Current computer systems support a paging mechanism by which the design data can be swapped transparently between the main memory and the secondary storage such as the hard disks. These input/output operations due to page swapping, known as page faults [5], may contribute as great overhead to the running time of a CAD system. In particular, for low-cost workstations equipped with limited memories and low-performance hard disks, this overhead could become prohibitive.

In the layout of VLSI and printed circuit boards (PCB), geometric objects in the plane must be represented as internal structures in a computer. A natural and conventional way of representing these 2-dimensional (2-dim) geometric data is by using a large 2-dim array such as the cell map or cell array of a maze router [3]. This 2-dim array structure has the advantage of simplicity and ease of neighbor lookups. However, the 2-dim array is usually too large to fit entirely in the main memory, and thus must be subject to page faults. In fact, even if the main memory is large enough to accommodate the entire array, the array still have to compete for the memory with other data structures in an integrated CAD system.

Studies in automatic layout design and verification have focused mainly

on designing data structures and algorithms that are efficient in terms of the CPU time. The underlying assumption has always been that the data structures are stored entirely in the main memory or the I/O time due to page faults is negligible. This is not realistic, however. In practice, the I/O time due to page faults is usually comparable to the consumed CPU time. For a system with limited memory, the I/O time may even far exceed the CPU time. Thus, in the current work, we aim to minimize the page faults that layout algorithms generate in order to reduce the I/O time. In particular, we address the problem of how to structure a large cell map so that the number of page faults generated is minimized.

This page fault problem has been considered by Dion [2]. He recognized the severe consequence a large 2-dim array may result in, and proposed a list structure to avoid the excess of page faults. However, his data structure makes neighbor lookup efficient in one dimension but not in the other. Also, his data structure is complicated in comparision with the 2-dim array.

When a program is executed, the number of page faults it generates is mainly determined by its locality of (memory) reference [5]. Layout algorithms based on large 2-dim arrays exhibit poor locality of reference since adjacent cells in some dimension are far away in memory. In this paper, we will describe a new structure for large cell maps with an improved locality of reference. Experiments have shown that routing algorithms based on the proposed structure generate much less page faults than those based on the

2-dim array. Thus, using this structure, the I/O time due to page faults can be reduced dramatically. On the other hand, the new structure still preserves those desired properties of the 2-dim array. Neighbor lookups in both dimensions are so efficient that the CPU time consumed by routing algorithms using the new structure also reduces. The proposed structure is a 1-dim array. But, the implementation details are largely transparent. Programmers, when coding layout programs, can use the cell map as if it were a natural 2-dim structure.

In the next section, we investigate the requirements of this problem. A small set of operations are identified as necessary and adequate primitives to support the cell map. In Section 3, the cell map is represented as a 1-dim array, and the primitives are implemented as short macros for simplicity and efficiency. Two routing algorithms have been coded to test the 1-dim array structure versus the conventional 2-dim array structure. Test results of these experiments are reported in Section 4. Finally in the last section, we make some concluding remarks.

2. The Requirements

Let us look more closely into the problem. Consider, for instance, a 1024×1024 cell map where each cell contains a single word. When this cell map is declared as a 2-dim array in a conventional programming language, the compiler allocates a consecutive piece of memory space for the 2-dim array in the column-wise or row-wise order. Fig. 1 shows such a typical memory allocation. Suppose that each page of the computer consists of 1024 words. Then each column in Fig. 1 occupies exactly one full page. Two adjacent cells in the same row are always allocated in two different pages. Consider a maze router executing the Lee algorithm [3] on this cell map. On each step, a cell is taken from the wavefront, and its 4 neighbors are scanned. Since the 4 neighbors are located in 3 different pages, the Lee algorithm references 3 pages for each single step. The Lee algorithm thus demonstrates poor locality of reference on 2-dim arrays for it makes many page references by iterating the above steps. This has been manifested by our experimental results to be described in Section 4.

Locality of reference is a term which is very hard to quantify. Thus we propose page reference count as a quantitative indication of locality of reference. Layout algorithms based on cell maps usually execute within individual windows. When a layout algorithm executes in a rectangular window, the majority of the cells inside the window are referenced. Given a representation of the cell map and a rectangular window, we define the page

references as the set of all pages containing some cells inside the window. The page reference count is the number of page references. For example, for the 2-dim array in Fig. 1, the page reference count for a $p \times q$ window is p since the window contains p columns each of which accounts for one page. Given a cell map representation, it is straightforward to determine the page reference count. We consider the page reference count generally applicable to the majority of layout algorithms. The larger the page reference count is, the more page faults a layout algorithm will generate.

Now let us formulate the requirements for an appropriate representation of the cell map.

1. The representation must be able to support fast neighbor address computation. Given a cell's address in memory, the operations East, West, South and North determine the memory addresses of the 4 neighbors adjacent to the cell. In the 2-dim array representation, East(x,y) = (x+1,y), South(x,y) = $(x, y-1)$, etc. Using these 4 operations, one can easily implement operations Southeast, Northeast, etc.

2. Conversion between the representation and the inherent 2-dim array representation must be efficient. This is especially important for fast boundary checks such as $x \leq x_0$, etc.

3. Using the representation, the page reference count associated with a

rectangular window should be relatively small.

3. The 1-dim implementation

Consider a $M \times N$ cell map. For the moment, assume that M and N are powers of 2, say $M = 2^m$ and $N = 2^n$. We use a 1-dim array of size $M \times N$ to represent the 2-dim cell map. Let x and y be the coordinates of an arbitrary cell in the map. x can be separated into its high-order part x_h and its low-order part x_l, $x = (x_h, x_l)$. x_l consists of the w low-order bits of x, and x_h the remaining $m - w$ bits, where w is a constant to be determined later. The address z of the cell in the 1-dim array is formed by concatenating the bit strings x_h, y and x_l, i.e.

$$z = (x_h, y, x_l).$$

We use $x_h(z), y(z)$ *and* $x_l(z)$ to denote the three parts of z. As an example, for w = 3, the cell with coordinates x = 41 and y = 73 has its address z = 4731 in the 1-dim array. (All numbers are octal.)

Conversion between the 1-dim representation and the 2-dim representation is very efficient since this can be done by bit-wise mask and shift operations. Given a cell with address z in the 1-dim array, the addresses of its 4 neighbors can be calculated by the following macros in C language:

```
# define North(z) (z + pw)
# define South(z) (z - pw)
# define East(z) ((x_l(z)! = pw - 1)?(z + 1) : (z + t))
# define West(z) ((x_l(z)! = 0)?(z - 1) : (z - t))
```

where $pw = 2^w$ and $t = N \times pw - pw + 1$.

This 1-dim implementation of the cell map is considered to be highly transparent. Programmers, when coding layout programs, do not have to know the 1-dim implementation details. They simply use these macros (make macro calls).

To illustrate the structure of the 1-dim array, let us consider a small cell map where M = N = 8 and w = 1. Fig. 2 shows the addresses of all cells in the cell map in this 1-dim representation. For example, the cell with coordinates $x = 4(= 100_2)$ and $y = 3(= 011_2)$ has its address $z = 38(= 100110_2)$. Since a page is formed by a consecutive piece of memory, a page consisting of s cells corresponds to a rectangular region in the cell map with width pw and height ph where ph = s/pw. (Usually s is a power of 2.) In Fig. 2, if a page contains 4 cells, then each page corresponds to a 2×2 square region. In general, the cell map can be considered to be partitioned into equal-sized rectangular regions as shown in Fig. 3 where each region is associated with a page.

Given a $p \times q$ window, we can estimate the page reference count as follows: The window must intersect $\lceil p/pw \rceil (resp. \lceil q/ph \rceil)$ or $\lceil p/pw \rceil + 1$ $(resp. \lceil q/ph \rceil + 1)$ regions in the x(resp. y) dimension, where $\lceil a \rceil$ is the smallest integer greater than or equal to a. Thus the page reference count is a number between $\lceil p/pw \rceil \lceil q/ph \rceil$ and $(\lceil p/pw \rceil + 1)(\lceil q/ph \rceil + 1)$. As an example, consider a cell map with M = N = 1024, and the page size is 1024. Let w = 5. Then pw = ph = 32. For a window with width and height 200,

the page reference count is no more than 64. This is apparently smaller than that based on the 2-dim array representation which is 200.

We have made the assumption that M and N are powers of 2. If this is not true, let r and n be such that $(r-1)pw < M \leq r \times pw$ and $2^{n-1} < N \leq 2^n$. The 1-dim array then contains $r \times pw \times 2^n$ cells. Part of the array is never used. Since the paging mechanism loads pages into memory only when they are referenced, only a small fraction of this unused piece of storage will ever be loaded. The amount of main memory wasted is thus insignificant.

The value w should be determined according to the width and height of the window. If the window is square, w should be set so that pw and ph are as close to each other as possible. In a real CAD system, routing algorithms are usually applied to many different windows. Then w should be a system parameter to be tuned after many test runs.

4. Test Results

We have coded two variants of the Lee algorithm in C language in order to test the 1-dim array representation versus the conventional 2-dim array representation. The major data structure in each algorithm is a 2048×2048 cell map. Each cell is a two-byte integer. Thus the cell map occupies 8 MByte memory space. The cell map was implemented as either a 1-dim array or a 2-dim array. So there are 4 programs in total. These programs were tested on a popular workstation SUN 3/60 equipped with 4 MByte of main memory and a 327 MByte hard disk. Since the generation of page faults and the induced I/O time are greatly affected by the system's work load, each program was run under the UNIX single user mode. In the following, we describe what these programs do and report their outcomes.

The first algorithm (coded as two programs) executes the following repeatedly: Starting from a randomly selected unvisited cell, visit k unvisited cells and mark them visited by the Lee expansion where k is a random number generated between 3000 and 5000. The algorithm terminates when a given number of cells have been visited. This algorithm mimics layout programs that execute certain operations inside a peephole and move the peephole around. Test results from the two programs are summarized in Table 1. Note that the I/O time can be calculated by

$$I/O\ time = (wall\ time) - (user\ time) - (system\ time).$$

We have made the following observations: (1) Using the 2-dim array struc-

ture, the consumed I/O time is almost twice the consumed CPU time. (2) The 1-dim implementation effectively eliminates 90% of the page faults present in the 2-dim implementation. (3) Using the 1-dim array structure, the consumed I/O time is only a small fraction of the consumed CPU time. (4) Even in terms of the CPU time (user time + system time), the 1-dim implementation performed slightly better than the 2-dim implementation. The last point is due to the fact that fetching data in a 1-dim array is more efficient than in a 2-dim array.

Within a specified window, the second algorithm generates a path connecting two given cells by using the Lee algorithm. Initially, 30% of the cells are marked as obstacles randomly. Test results of the two implementations of the algorithm are shown in Table 2. Note the tremendous difference between the two methods in the wall time and the number of page faults generated when many cells are visited (The two given cells are far away).

5. Concluding Remarks

Cell maps have been used extensively in CAD applications such as various routing algorithms [4] [6], design rule checking [1], etc. As the dimensions of VLSI circuit elements are getting smaller and smaller, the cell maps require more memory space to accommodate the fine-grid layout. The paging traffic thus rises as a crucial overhead to the system's performance. We have presented a 1-dim array representation of the cell map to eliminate the excess of page faults present in the traditional 2-dim array representation. Using this representation, the I/O time due to page faults can be reduced dramatically.

The proposed structure is very general. The majority, if not all, of layout algorithms based on the 2-dim cell maps can be modified easily to work with this 1-dim structure. Even though this structure has been tested only with some Lee-type algorithms, we consider it highly likely to be effective to other layout algorithms as well. More experience is still to be gained.

The performance of layout algorithms using the proposed structure tends to be less sensitive to the capacity of main memory and the speed of hard disks. Thus, the use of this new structure in layout algorithms may make it possible to support powerful CAD systems on low-cost workstations with limited memories and low-performance hard disks.

References

1. C. M. Baker and C. Terman, "Tools for verifying integrated circuit designs", Lamda, vol. 1, No. 3, June 1980.

2. J. Dion, "Fast printed circuit board routing", Proc. 24th Design Automation Conf., 1987.

3. C. Y. Lee, "An algorithm for path connection and its applications", IRE Trans. on Electronic Computers, Sept. 1961.

4. T. Ohtsuki (edit), Layout Design and Verification, Elsevier Science Publishers B.V. (North Holland), 1986.

5. J. Peterson and A. Silberschatz, Operating System Concepts, Addison-Wesley Publishing Co., 1983.

6. J. Soukup, "Circuit layout", Proc. of the IEEE, Vol. 69, Oct. 1981.

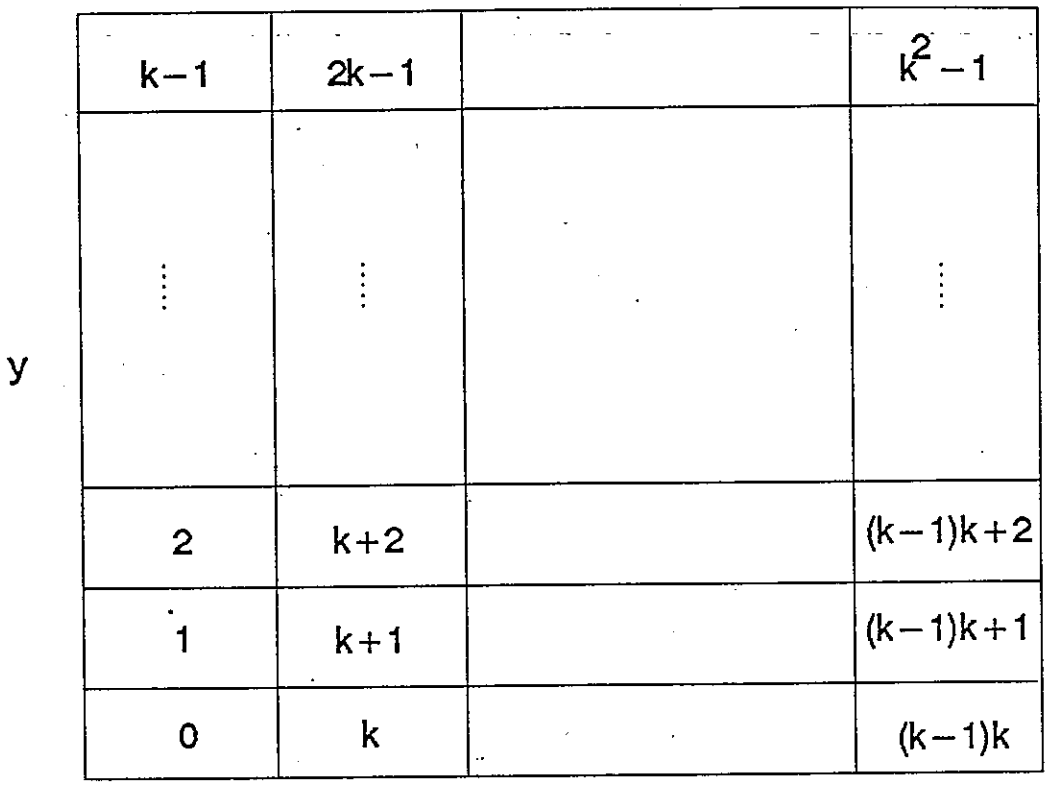

Fig. 1. Cell map as 2 – dim array

14	15	30	31	46	47	62	63
12	13	28	29	44	45	60	61
10	11	26	27	42	43	58	59
8	9	24	25	40	41	56	57
6	7	22	23	38	39	54	55
4	5	20	21	36	37	52	53
2	3	18	19	34	35	50	51
0	1	16	17	32	33	48	49

y

x

Fig. 2. Addresses in 1 – dim array

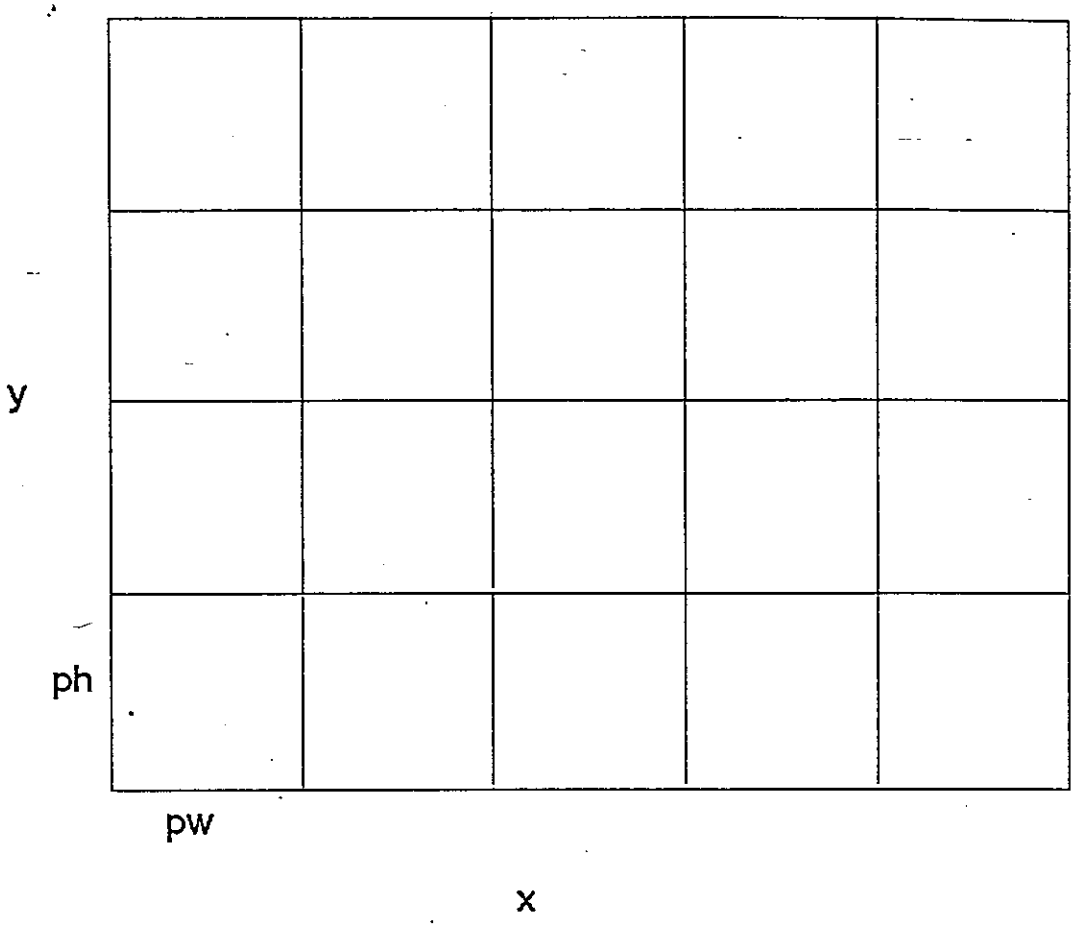

Fig. 3. Rectangular regions associated with pages

Ex.	# of visited cells	user time (sec)		system time (sec)	
		1-dim	2-dim	1-dim	2-dim
1	1,000,000	147.66	162.56	4.72	40.52
2	1,300,000	192.72	211.10	5.62	52.38
3	1,600,000	237.38	261.02	8.00	65.54
4	1,900,000	280.80	311.18	9.76	78.00
5	2,200,000	326.82	358.66	12.28	86.30
6	2,500,000	370.66	409.34	13.96	102.38
7	2,800,000	415.80	456.28	16.02	119.24
8	3,100,000	460.70	507.34	18.84	132.16
9	3,400,000	507.26	559.30	21.04	147.92

Table 1. Test results of algorithm 1.

Ex .	# of visited cells	wall time (hour:min:sec) 1 – dim	2 – dim	# of pagefaults 1 – dim	2 – dim
1	1,000,000	00:02:45	00:09:08	372	8,325
2	1,300,000	00:03:34	00:11:22	487	10,281
3	1,600,000	00:04:33	00:14:17	837	12,934
4	1,900,000	00:05:23	00:16:37	1,031	14,800
5	2,200,000	00:06:25	00:19:15	1,430	17,330
6	2,500,000	00:07:24	00:22:34	1,819	20,538
7	2,800,000	00:08:20	00:25:37	2,155	23,457
8	3,100,000	00:09:28	00:28:29	2,724	26,285
9	3,400,000	00:10:36	00:31:50	3,311	29,522

Table 1. (continued)

Test results of algorithm 1.

Ex .	# of visited cells	user time (sec)		system time (sec)	
		1 – dim	2 – dim	1 – dim	2 – dim
1	27,907	5.42	5.94	0.02	0.46
2	46,469	9.06	9.48	0.22	0.64
3	278,334	55.96	60.16	0.42	1.80
4	383,722	77.08	83.72	0.50	13.60
5	503,444	101.06	111.28	1.18	112.24
6	633,022	127.86	141.60	1.34	260.66
7	777,037	157.96	176.48	1.52	459.38
8	937,641	190.12	218.26	1.48	679.04

o 30% of grid space are occupied.

Table 2. Test results of algorithm 2.

Ex .	# of visited cells	wall time (hour:min:sec) 1-dim	2-dim	# of pagefaults 1-dim	2-dim
1	27,907	00:00:06	00:00:10	21	100
2	46,469	00:00:10	00:00:15	31	148
3	278,334	00:01:00	00:01:12	130	350
4	383,722	00:01:22	00:04:16	166	1,423
5	503,444	00:01:48	00:20:53	214	15,587
6	633,022	00:02:17	00:47:00	267	40,996
7	777,037	00:02:48	01:20:12	329	78,274
8	937,641	00:03:23	01:58:05	395	121,662

o 30% of grid space are occupied.

Table 2. (continued)

Test results of algorithm 2.