Acceleration of Mincut Partitioning using Hardware CAD Accelerator TP5000

Masahiro SANO
CAD Group
FUJITSU LTD.
Kawasaki, Japan 211
Tel: +81-44-754-2166
Fax: +81-44-754-2391
e-mail: sano@fd.cad.fujitsu.co.jp

Shintaro SHIMOGORI
CAD Group
FUJITSU LTD.
kawasaki, Japan 211
Tel: +81-44-754-2166
Fax: +81-44-754-2391
e-mail: shinta@fd.cad.fujitsu.co.jp

Fumiyasu HIROSE
CAD Laboratory
FUJITSU LABORATORIES LTD.
Kawasaki, Japan 211
Tel: +81-44-754-2663
Fax: +81-44-754-2664
e-mail: hirose@flab.fujitsu.co.jp

Abstract— This paper presents a new approach of data pipelining for mincut partitioning acceleration using a parallel computer. We choose the hardware CAD accelerator TP5000 to implement our approach. We obtain a speed improvement of 20 to 25 times as fast as a SPARCStation-10 by using 10 processors in the TP5000.

I. Introduction

The placement problem for VLSI design is to decide the position of cells in order to minimize the chip size, power consumption, signal delays, and cycle time of a resulting chip. Many algorithms have been proposed, these include mincut-based placement algorithms [1, 2].

Among them, the Fiduccia-Mattheyses mincut algorithm (FM) is widely used, due to its fast computation characteristics [2]. The total run-time is determined by $O(P)$, where P is the number of pins. However, as VLSI chips become more complicated due to the advances in VLSI fabrication technology, the time for mincut-based placement becomes much larger.

In addition, the FM algorithm has the disadvantage that the process easily falls into a local minimum. In order to overcome this disadvantage, some additions to the FM algorithm have been proposed [3, 4]. We use the SNT-FM algorithm, which adds Stable-Net-Transition (SNT) to the FM algorithm [4]. According to [4], the SNT-FM algorithm achieves 27% to 62% fewer cuts than the FM algorithm for some layout data from 10 K to 100 K gates. However, the SNT addition also lengthens the computation time of the mincut-based placement. Therefore, placement time becomes an even more important factor.

This paper presents a new approach to data pipelining for mincut partitioning acceleration. In our approach, the mincut process is partitioned into many small sub-processes. Each sub-process is then assigned to a processing element (PE), and each data set is processed on PEs successively. We implement the SNT-FM algorithm on the hardware CAD accelerator TP5000 which is suitable for the data-pipelined processing. We obtain the speed improvement of 20 to 25 times faster than a SPARCStation-10 by using 10 processors in the TP5000.

II. SNT-FM algorithm

The FM algorithm uses a simple "hill-climbing" technique for optimization, and is likely to be trapped in a local minimum. This means that the final partition depends greatly on the initial partition. Some reports state that over 80% of the cut nets after executing the FM were already cut in the initial partition, and the FM falls into a local minimum due to these nets [4]. A net which remains cut from the beginning until the end when the FM is executed once, is called a "stable net", or SN.

Fig.1 shows the SNT-FM algorithm, which adds Stable-Net-Transition (SNT) to the FM algorithm. The SNT operation is executed after the FM process in order to escape from a local minimum. The SNT detects stable nets (line 4 in Fig.1), and forces all cells on stable nets into one block and makes their nets "not cut" (line 5 in Fig.1). Then, the FM restarts with the rearranged partition from the SNT operation as the initial partition. This process is repeated in order to optimize the cut size (line 2 in Fig.1).

```
SNT-FM algorithm{
1.  partition initially at random;
2.  for i = 0 to #iteration {
3.      apply FM;
4.      search for stable nets (SN);
5.      move cells connected to stable nets
        and obtain new initial partition;
    }
}
```

Fig. 1. The SNT-FM algorithm.

ASP-DAC '97
0-89791-851-7$5.00

III. Configuration of Data Pipelines

We present a data pipelining approach to mincut partitioning. In our approach, the mincut process is divided into many small sub-processes. Each sub-process consists of a few simple instructions, and the amount of instructions in each sub-process are balanced between the other sub-processes.

Fig.2 shows the sub-process partitioning for the FM algorithm. First, the cell with the maximum cell-gain is selected from the gain-list and moved from its current block to the complimentary block, where the cell-gain of a cell is defined as the decrease of cuts when the cell moves from its current block to the complimentary block, and the gain-list is a data structure to manage cell-gain. Next, it reads out the net IDs connected with the moved cell, and reads out the cell IDs connected to the net, and reads out the block IDs for the cells. Then, it calculates how many cells exist in each block for the net before and after the cell is moved. After that, the difference of net-gain for the net is calculated, where the net-gain of a cell connected by a net is the decrease of cut for the net when the cell moves from its current block to the complimentary block, so the net-gain can be -1, 0, or 1. Then, a new cell-gain can be calculated by adding the difference of net-gain to the old cell-gain. Finally, it updates the gain-list by removing the cell from the old cell-gain position and appending it to the new cell-gain position. After calculating the cell-gains for all cells adjacent to the moved cell and updating the gain-list, the subsequently moved cell is selected. This operation is repeated until there are no more movable cells.

When one cell is moved, the cell-gain of cells adjacent to the moved cells can change. So, the next moved cell cannot be selected until the new cell-gains are calculated. However, since adjacent cells to a moved cell are independent of one another, reading out sub-processes, calculating sub-processes, and updating sub-process can be processed in parallel.

Fig.3 shows how the FM algorithm can be realized using data pipelining. Each pipeline stage has individual local memory (LM). In Fig.3, each square represents a pipeline stage, i.e., each process of a PE. Each block underneath the PE represents the data structure of the local memory. The arrows between the blocks show data flow.

In Fig.3, PE7 starts up this process by sending the block ID which may contain a moved cell to PE6. Then, PE6 selects the cell ID with the maximum cell-gain from the gain-list in its own LM, and feeds it to PE8. PE8 reads out the net IDs connected with the moved cell ID from the cell-to-net table in its own LM, and sends out the net IDs to PE1. PE1 reads out the cell IDs from net-to-cell table in its own LM and sends the cell IDs to PE2. PE2 reads out the block IDs from the cell-to-block table in its own LM and sends the block IDs to PE3. PE3 reads out the number of cells in each block for the net from its own LM, and calculates it after moving the cell. It stores this value into its own LM and sends the number of cells in each block before and after moving the cell to PE4. PE4 reads out the pre-moving net-gain and post-moving net-gain from its own LM and calculates the change in net-gain by subtracting the pre-moving net-gain from the post-moving net-gain, and sends it to PE5. PE5 reads out the old cell-gain from its own LM and adds the net-gain change. It then stores this new cell-gain into its own LM and sends the new and old cell-gain to PE6. PE6 updates the gain-list in its own LM.

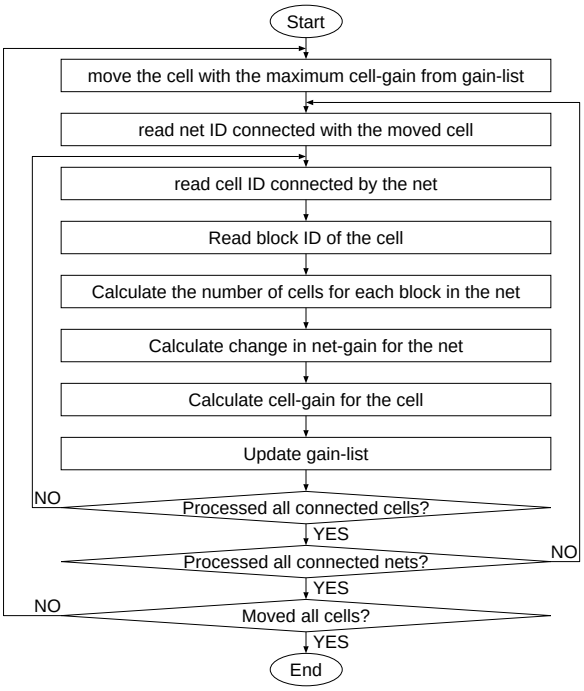

Fig. 2. Partitioning the FM algorithm into sub-processes.

We can also construct pipelines for the rest of the SNT-FM algorithm, which are line 4 and 5 in Fig.1.

IV. Implementation on TP5000

The TP5000 is a hardware CAD accelerator which consists of dedicated Very Long Instruction Word (VLIW) processors with high-speed interconnections [5]. Its data pipelines can be reconfigured. The TP5000 is scalable to 5120 processors. One processor is called a Processing Element (PE), and a group of 10 PEs is called a Processor Group (PG). It takes one PG to make a data pipeline.

The structure of one Processor Group (PG) is shown in Fig.4. The figure shows how the data pipeline of the FM algorithm from Fig.3 is mapped onto one PG. In this figure, bold circles correspond to the PEs, and bold arrows correspond to the data flow in Fig.3.

Fig.5 shows the structure for each PE in the TP5000. The PE consists of two local memories (LM1 and LM2),

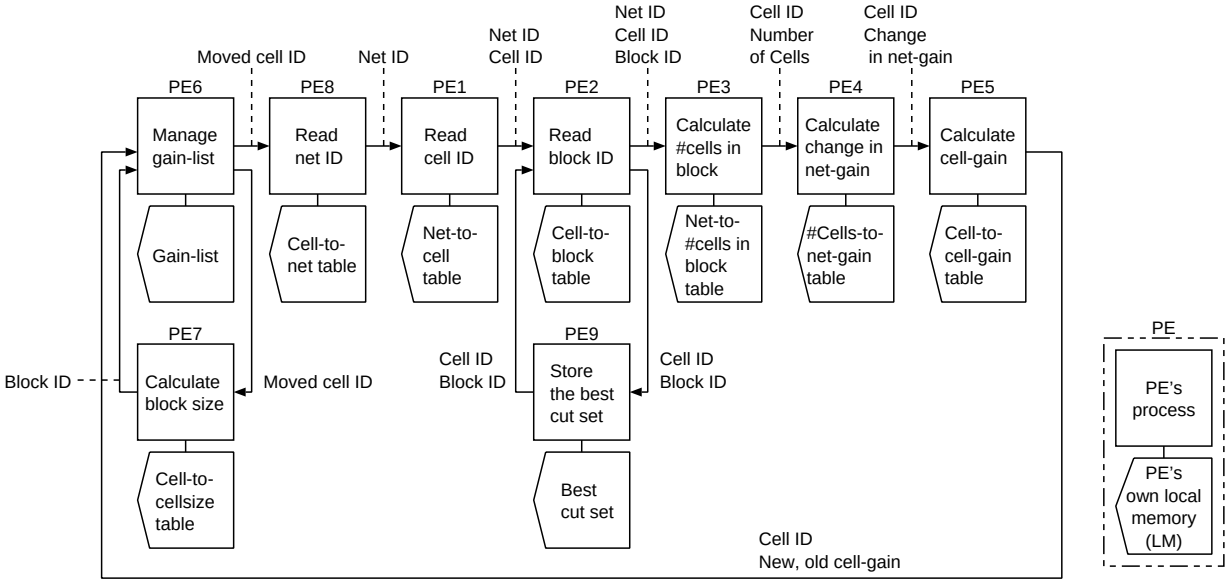

Fig. 3. Data pipelining implementation of the FM algorithm.

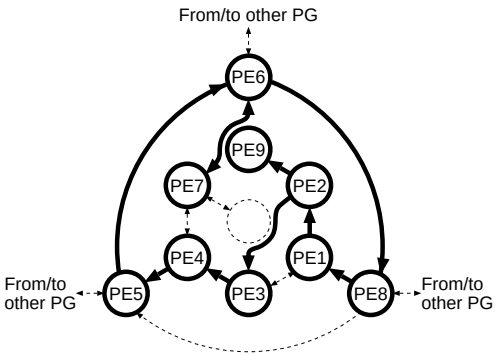

Fig. 4. Mapping onto PG for the FM algorithm.

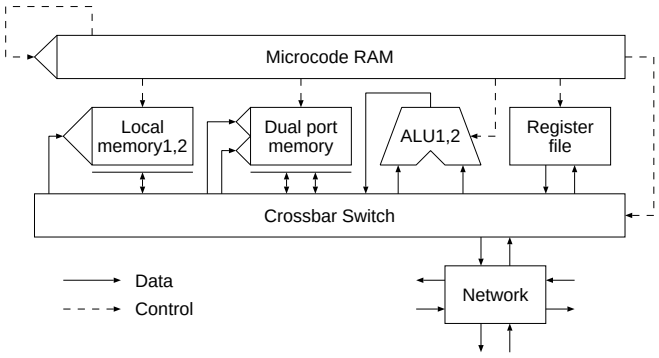

Fig. 5. Processing Element (PE) architecture in TP5000.

one dual-port memory (DPM), two ALUs (ALU1 and ALU2), four register files, a crossbar switch, a network interface, and controlling microcode RAM. The crossbar switch allows for flexible interconnections, and all facilities are controlled in parallel by the VLIW microcode program stored in a microcode RAM. The local memory is high-speed SRAM, so memory access is very fast.

Here, we show the example of the processing in one pipeline stage. The example is the sub-process executed in PE6 in the FM algorithm. PE6 holds the gain-list in its own local memory and dual-port memory.

Fig.6 shows the process of PE6. In Fig.6, the horizontal axis represents facilities, the vertical axis shows the microcode instruction step, and each square represents a process which each facility executes at each microcode instruction step. For example, at microcode step 0 the Network-in, LM1 address control, LM2 address control, condition check, and branch are active, so five processes are executed. Processes of facilities proceed concurrently from microcode step 0 to 4 for one set of data. Fig.6 shows five steps for one set of data. However, at microcode step 4, the processes at microcode step 0 for the next data set are executed simultaneously, shown by the dotted squares. Then the PE executes the microcode of step 1 for the next data set. Therefore, the number of microcode addresses for a data set in the process of PE6 is four. When we compile the program for the process of PE6 using the GNU C compiler (gcc) with the -O3 option on a SPARCStation-10, the entire process is executed with 43 instructions. Thus the process using PE on TP5000 is much faster.

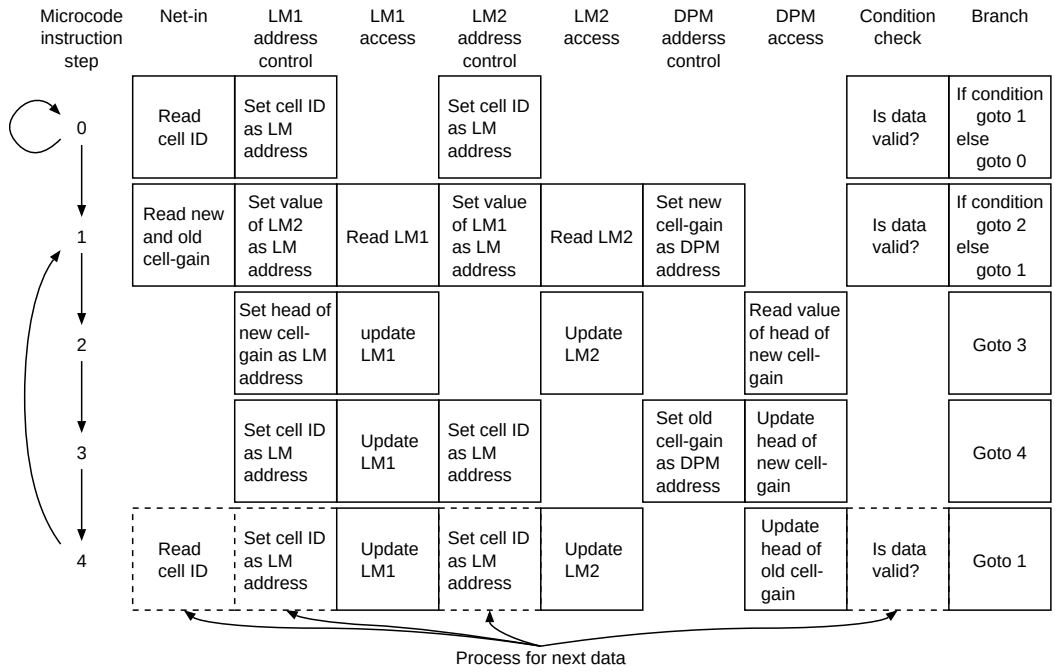

Fig. 6. Process of PE6.

V. Experimental Result

We implement the SNT-FM algorithm using one PG in the hardware CAD accelerator TP5000. The speed of the SNT-FM algorithm was evaluated. We executed our tests on four different circuits. Table I shows the execution time of the SNT-FM algorithm using a SPARCStation-10 (SS-10), and using one PG in the TP5000, and the relative speed improvement (time on SS-10 / time on TP5000).

As shown in Table I, we obtained a speed improvement of 20 to 25 times faster than a SPARCStation-10 by using one PG in the TP5000.

We used SS-10 in this experiment which is fabricated by similar technology level to TP5000 (0.8μ CMOS, 15MHz). Accordingly, Table I shows the advantage of data pipelining approach.

TABLE I
Execution time on SS-10 and on one PG in the TP5000.

Circuit	Cells	Nets	Execution time(sec.)		Improve ratio
			SS-10	TP5000	
A90	7870	11809	466.5	21.5	21.7
B59	9021	12240	609.5	32.2	18.9
C27	14357	17435	1414.7	54.7	25.9
D51	37813	42504	7075.9	325.5	21.7

VI. Conclusion

We presented a data pipelining approach to mincut partitioning and implemented the SNT-FM algorithm on the hardware CAD accelerator TP5000. We improved the execution speed by about 20 to 25 times faster than a SPARCStation-10 by using 10 processors in the TP5000.

The TP5000 is scalable up to 5120 PEs, we plan to further increase the execution speed to more than 100 times faster than a SparcStation-10 by developing an multiple data pipeline mincut approach.

References

[1] M. A. Breuer,"A Class of Min-Cut Placement Algorithms," *Proc. 14th Design Automation Conf.*, pp. 284-290, 1977.

[2] C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," *Proc. 19th Design Automation Conf.*, pp. 175–181, 1982.

[3] M. R. Hartoog, "Analysis of placement procedures for VLSI standard cell layout," *Proc. 23rd Design Automation Conf.*, pp. 314-319, 1986

[4] T. Shibuya, I. Nitta and K. Kawamura, "SMINCUT: A VLSI Placement Tool using Min-Cut," *Fujitsu Sci. Tech. J.*, Vol. 31, No. 2, pp. 197-207, 1995

[5] S. Shimogori, K. Takayama, H. Matsuoka, K. Hirahara and F. Hirose, "Thread Processor TP5000 for CAD Acceleration," *Intl. Symp. on Fifth Generation Computer Systems 1994*, pp. 17-24, Dec. 1994.