A Back-End Compiler with Fast Compilation for VLIW based Dynamic Reconfigurable processor

Ryuji HADA, Kazuya TANIGAWA, Tetsuo HIRONAKA
Department of Information Sciences
Hiroshima City University
Address 3-4-1 Ozuka-Higashi, Asa-Minami-Ku, Hiroshima, 731-3194, JAPAN
pars@csys.ce.hiroshima-cu.ac.jp

Abstract: - We have developed a compiler for dynamic reconfigurable processor based on VLIW model. VLIW model fetches and executes one configuration data as VLIW instruction. For this model, our compiler schedules mapping elements as operations and live variables in program, with consideration of hardware resources. Next, place-and-route procedure places and routes the mapping elements to hardware resources for several configuration data. However the conventional place-and-route algorithms require much compilation time. The reason is that, for difficulty place-and-route condition, the number of place-and-route iteration is increased to get high code quality. Thus we propose a novel compiler method, which is combining scheduling and place-and-route with fast compilation, on keeping code quality. Our idea is that if a scheduling simplifies the place-and-route condition, small compilation time of place-and-route can realize a reasonable code quality. In scheduling, to balance the number of operations and live variables, and make the place-and-route condition easy, the operations are moved to another step with a fewer operations. In place-and-route, to reduce iteration procedures to get the reasonable result, it limits targets for replace-and-reroute. In this paper, we use PARS as one of target processors based on VLIW model. We evaluate our method and compare it with another method based on Simulated Annealing (SA). From the results, our method achieves that the difference of code quality (the number of configuration data like VLIW instruction) is -3.4% - +1.2%, and compilation time is cut to 1/128 - 1/67, compared with SA base method.

Key-Words: - VLIW, Reconfigurable processor, compiler, scheduling, place and route

1 Introduction

Recently, dynamic reconfigurable processors based on a novel computing model have been attracted[1][2][3]. These processors dynamically configure their circuits by using configuration data, especially in VLIW model which fetches and executes one configuration data as one VLIW instruction. In these processors, since the parallelism included in configuration data are generated by a compiler, the compiler optimization technique is a key for high performance. In our previous researches, we proposed the compiler optimization method for the processor based on VLIW model[4]. In general, to generate the optimized code, much compilation time is required.

In this paper, we propose a novel compiler method, which is combining scheduling and place-and-route with fast compilation for VLIW based dynamic reconfigurable processors. Our idea is that, if a scheduling simplifies place-and-route condition, small compilation time of place-and-route can realize a required code quality.

The compiler has place-and-route procedure. The conventional place-and-route algorithms adopt the heuristic method[5]. In general, the heuristic method requires much compilation time to get high quality code. In the future, because of the following reasons the conventional place-and-route cannot realize reasonable compilation time.

- According to high integration of LSI, processing elements are increased in the processor.
- According to high difficulty of image processing, the number of operations as mapping element is increasing[6]. The image processing is one of main target application of the processor.

According to increase of processing elements and mapping elements, the compilation time of place-and-route iteration procedure for each configuration data is increased. The conventional place-and-route

requires much compilation time for total iterations, on keeping high code quality.

The conventional place-and-route problem is described in the following.

- For difficult place-and-route condition, the number of place-and-route iteration is increased.
- For physically impossible place-and-route condition in one configuration data, the place-and-route requires a certain number of iteration to find it impossible.
- When the place-and-route finds impossible place-and-route condition, it may use several configuration data and replace-and-reroute. In this case, first place-and-route time is waste.

From this problem, to reduce the compilation time, one place-and-route iteration time and the number of the iteration should be reduced. Thus we propose the following two approaches.

- Scheduling reduces the difficulty of place-and-route condition, to avoid the impossible place-and-route condition. Scheduling estimates the difficulty of the place-and-route condition before place-and-route. If it finds the impossible place-and-route condition, it schedules mapping elements to be placed and routed in two or more configuration data. Besides, it adjusts the difficulty not to extend a critical path as possible.
- Place-and-route realizes less the number of iteration, by determining an init placement with consideration of routing condition. Besides, it uses several place-and-route patterns prepared.

Our approaches can be adopted for the processors requiring place-and-route, which is based on our VLIW model, because our approaches do not depend on the processor implementation.

In this paper, we describe our approach and evaluate the compilation time and the number of generated configuration data (like VLIW instructions) as code quality. We compare our method with general heuristic based place-and-route method.

The organization of this paper is the following. In Section 2, we define our VLIW model, and explain a target processor based on our model. Then we explain another compiler method for dynamic reconfigurable processor, to declare the differences between our method and others. In Section 3, we explain why fast compilation time is difficult, and we propose a key idea to resolve this problem. In Section 4, we give an overview of the proposal method, and show the algorithms in Section 5. After that, we evaluate our method in Section 6, and then we conclude in Section 7.

2 Related works

In this section, we define VLIW model as target processor model of our compiler. Then, we introduce our target processor and various compiler approaches which target is VLIW model.

2.1 VLIW model

In this paper, VLIW model reconfigurates and executes configuration data in a program as VLIW instructions. In VLIW model, reconfigurable part has distributed registers and function units (FUs) with B bit width. The number of registers and FUs is N, M respectively. Each FU has two inputs and one output, and these inputs and outputs data can be read from and written to any registers. And the registers in the reconfigurable part are used for data transfer among different configuration data. Data in any registers can be loaded from and stored to memory by load/store unit. Memory architecture has P memory ports. Thus, by generalizing this model and varying the parameters N, A, M, B, P, our work may be extended to compilers for other similar architecture.

In this paper, we select PARS as target processor. Because PARS has simple VLIW model, and this detailed processor architecture is described in [7][8][9][10]. We will describe about this architecture in next sub section.

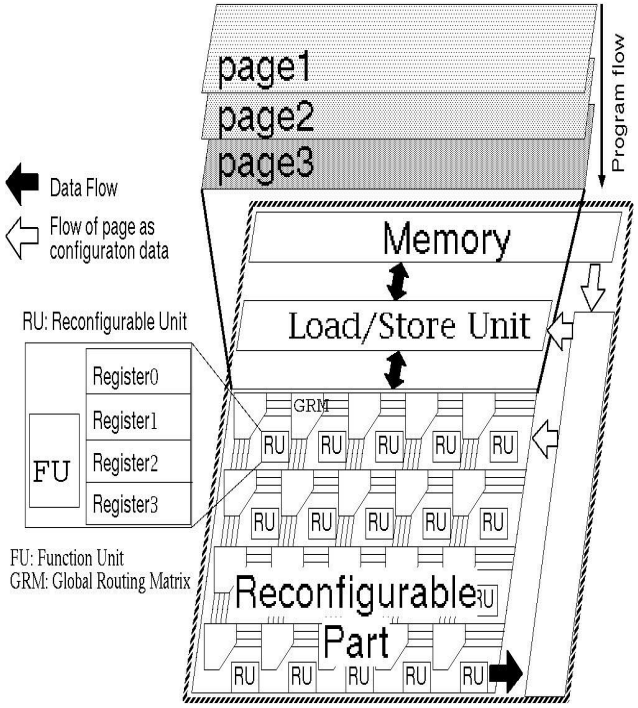

Fig. 1: Structure of TEMPO4x5 Processor

2.2 PARS

In this paper, we use TEMPO4x5 processor which is a prototype of PARS. Fig. 1 shows the structure of TEMPO4x5 processor. Reconfigurable part in the processor has 20 Reconfigurable Units (RUs). Each RU has one Function Unit (FU) with eight-bit ALU and four registers. Global Routing Unit (GRM) forms data transfer among RUs. This reconfigurable part is dynamically configured by configuration data with fixed length. One configuration data is called as *page*. In TEMPO4x5 processor, the configuration and execution of one *page*, which include operations executed at parallel, are done in one cycle. The execution of program on TEMPO4x5 processor is the following.

(1) *Page* fetch
Control unit fetches *page* from memory when it receives the start signal.

(2) Reconfiguration and execution
Operations and data path are configured in reconfigurable part by *page*. The input data for operations are transferred from registers in reconfigurable units.

Next, we explain other dynamic reconfigurable processors based on VLIW model and their compiler methods, to declare the differences between our method and others.

2.3 PipeRench[11][12][13]

PipeRench at Carnegie Mellon University supports execution of virtual pipeline. This virtual pipeline consists of *stripe* which is one virtual pipeline stage, and PipeRench is configured and executed by one *stripe* after the other. If we consider one *stripe* as one VLIW instruction, we can adapt PipeRench model to our VLIW model.

However, PipeRench has one configuration data as each *stripe*, and then deep pipeline stage requires many *stripes*. In such case, the compilation time increases according to the number of *stripes*, so the compiler is designed with consideration of compilation time. To achieve the fast compilation time, their scheduling method shortens the live range of variable allocated to registers to ease place-and-route difficulties. In our approach, we adopt a spill method (spills variables value out/in to/from memory), which adds load/store to original source code.

2.4 ADRES[14][15]

ADRES (Architecture for Dynamically Reconfigurable Embedded System) and DRESC (Dynamically Reconfigurable Architecture System Compiler) were developed by IMEC. If we consider operations in a step as one VLIW instruction, this model is similar to our VLIW model.

In place-and-route, they introduce SA base place-and-route. From this discussion, DRESC is developed to focus on only getting high quality code. There are no descriptions about saving the compilation time in their papers.

3 Problems and solution

We describe about the mapping restriction of the hardware resources in TEMPO4x5 processor, and discuss the problem and our solution.

Fig. 2 shows the correspondence of application program and *pages*. It is desirable that parallel executable operations are mapped to one *page*. However, if it does not meet hardware resource restrictions of one *page*, these operations are mapped to two or more sequential *pages*. For example, when there are four RUs in reconfigurable part and five parallel executable operations, these operations are mapped to two *pages*. In this paper, we call it *page division*.

In routing, data paths which transfer data for operations are realized by two dimensional routing like *page* 2 in Fig. 2. Routing is required to

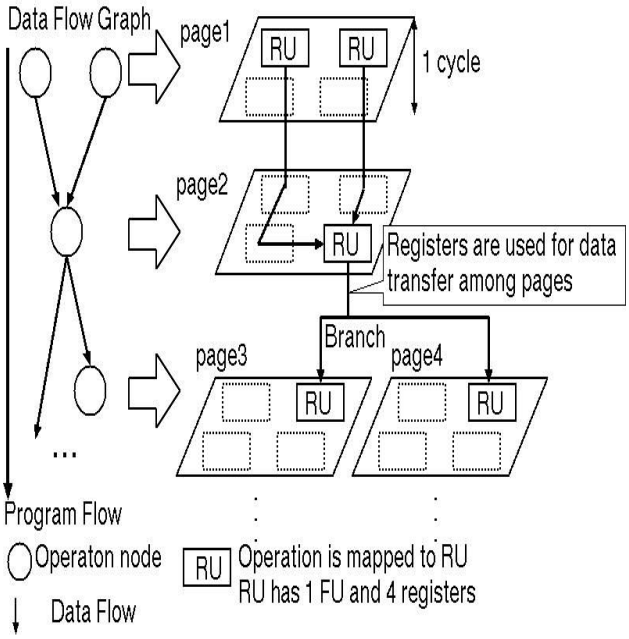

Fig. 2: Correspondence of Application Program and *page*

consider register locations, because data transfers among *pages* are realized by using registers. Dimension of the solution space on place-and-route becomes tree-dimension because of the two dimensional *page* (x and y axes) and the sequence of *pages* (time-axis). Thus, the following two items need to be considered.

(a) Time scheduling decides which operations are executed on which *pages*.

(b) Two-dimensional scheduling decides where operations are placed in each *page*.

If the conventional place-and-route algorithms are used for placement of each *page*, the compilation time is increased by the following reasons.

- If one *page* has many operations and live variables, the solution space becomes complex, which increases the compilation time on the *page*.
- There are many restrictions on place-and-route to realize a fixed frequency in PARS architecture. It may increase iteration procedures to get reasonable results.

To solve the above problems, we propose new approaches in consideration of (a) and (b).

Approach 1

To reduce difficulties caused by *page* which has many operations, we propose code-scheduling which balances the number of operations in each *page* as equally as possible. Its scheduling policy has an effect which shortens the live range of each variable, to reduce the number of live variables in each *page*.

Approach 2

To reduce iteration procedures to get reasonable results, the target operations for replace-and-reroute are limited to only operations which failed to be placed and routed. If the operations cannot be mapped to one *page*, *page division* is applied to these operations.

In the next section, we describe about our method based on these approaches.

4 Overview of our compiler

We overview our method which is comprised of code-scheduling based on the approach 1 and place and-route based on the approach 2. At first, we describe about the components of our compiler and then we describe the details about each component.

4.1 Software development environment

Fig. 3 shows software development environment for PARS. PARS compiler is comprised of the front-end part and the back-end part. In this paper, we describe about the back-end part. The front-end part compiles programs described by high level language into the intermediate code based on quadruples. The back-end part compiles this intermediate code into object code which has a set of *pages*. This back-end part is mainly comprised of code-scheduling and place-and-route.

4.2 Code-scheduling

Code-scheduling is executed before place-and-route. This procedure based on approach 1 assigns operations to *page* temporally. Next, we explain code-scheduling with Fig.4.

Variable rename

To extend the flexibilities of register allocation, this procedure resolves output dependencies and anti-dependencies[17].

Reduction of the live range of variables

At first, all operations are scheduled as soon as possible (ASAP), which extracts parallelism. Then to reduce the live range of variables, operations

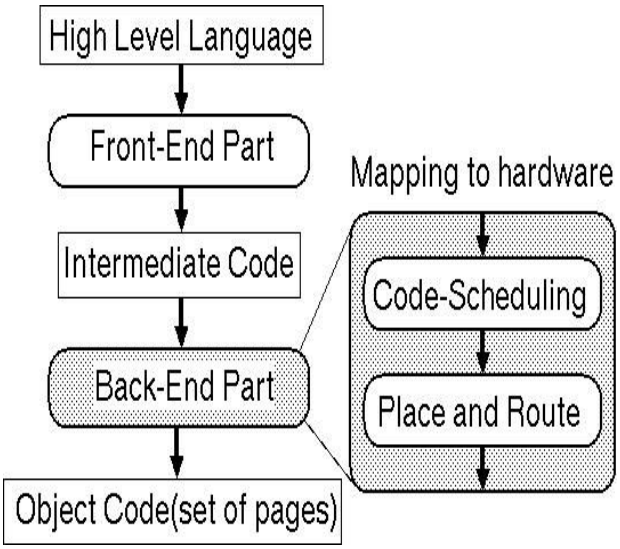

Fig. 3: Software Development Environment

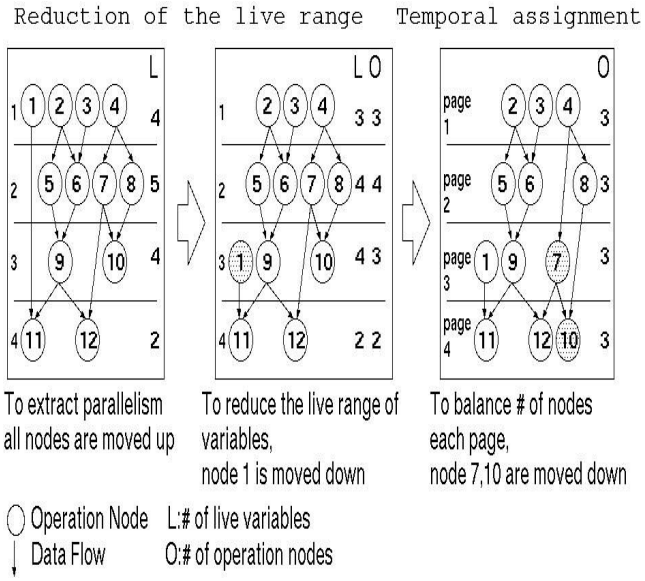

Fig.4: Code-Scheduling Flow

which do not have parent nodes are scheduled as late as possible. The later scheduling is executed not to extend the live range of variables, even if operations are moved to down steps on DFG (Data Flow Graph) like operation 1 in Fig.4. For instance, the number of live variables in the step 1 and the step 2 in Fig.4 is reduced while maintaining the length of critical path.

Temporal assignment

To reduce the number of operations to be placed, this procedure assigns operations to each *page* in consideration of the number of RUs in the processor. At this time, it balances the number of operations in each *page* as equally as possible. In addition, it reduces the number of operations on *page* 2 in Fig.4 by moving operation 7 and operation 10.

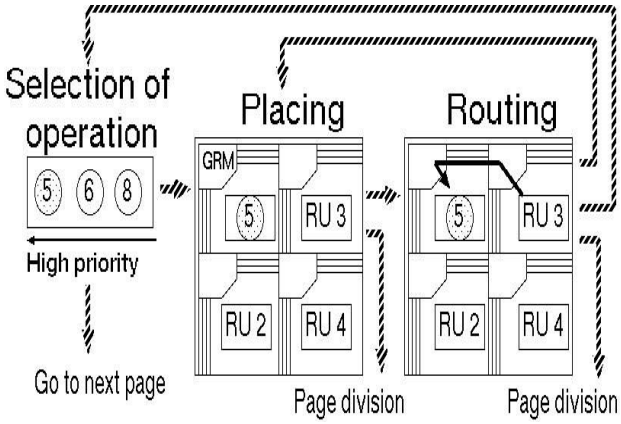

Fig. 5: Place and Route Flow

4.3 Place and route

This procedure is based on approach 2. Place-and-route allocates operations and live variables in each *page*. We explain it by using Fig. 5.

Selection of operation

To achieve an efficient search solution to get reasonable results, this procedure selects an operation according to its priority. Some of operations have to use several RUs simultaneously, which have low flexibility on placement. Because it is difficult to place-and-route them, they are given high priority. It improves the success rate of place-and-route for them, which reduces the possibilities of re-place and re-route. When there are no operations to be selected in the *page*, this procedure works on the next *page*.

Placing

This procedure places the selected operation on RU (the operation is assigned to FU and the destination variable is assigned to its register), according to a given placement order. The order balances the usage rate of resources. For instance, operation 5 is placed on RU 1, operation 6 will be placed on RU 2 and operation 8 will be placed on RU 4. If there are operations which cannot be placed on RU in the *page*, they are placed on RU in another *pages* by *page division*.

Routing

The data path, which transfers source data to the placed operation (operation 5), is routed in this procedure. When the data path can be routed it is done, then this routing procedure is finished, and it goes back to the selection of operation step. When the data path cannot be routed, only the placed operation (operation 5) is re-placed on another RU, and the data path is re-routed. Wherever the operation is re-placed,

if there are no re-routing paths on the *page*, the data path is routed on another new *page* by *page division*.

Our place-and-route method has two major features. One is the selection policy of operations according to the priority. Operations, which placing is difficult, are given high priority. The other is the sequential place-and-route which places and routes operations and live variables one by one. It is iterated until all operations and live variables are placed and routed.

5 Algorithms

In the previous section, we mentioned about overview of our code-scheduling and place-and-route. In this section, we describe these algorithms in detail.

5.1 Code-scheduling algorithm

The input code is represented by the following quadruples:(operator, source1, source2, destination). Fig.6 shows the pseudo-code which represents our code-scheduling described in Subsection 4.2. Our code-scheduling algorithm has two major features. One is the reduced scheduling complexity. Because operations are only moved down the DFG after the ASAP scheduling, the scheduling complexity is reduced. The other is the limited moving range of operations in Temporal assignment. We define mobility value as the number of steps which each operation can be moved. Its initial value is 10 (in the current version of our compiler). When the operation is moved one step down, it consumes 1 from mobility value. If the mobility value becomes 0, the operation cannot be moved. It prevents the live range of variables and scheduling time from increasing too much.

5.2 Place and route algorithm

The input code is the *pages* where the intermediate code is temporarily assigned, and each *page* has operations and live variables. Fig.7 shows the pseudo-code which represents our place-and-route described in Subsection 4.3. Our place-and-route algorithm has one major feature: less usage of random number. Because operations are selected and RUs are searched according to the given order, with this algorithm it obtains higher code quality than average in a short compilation time.

```
/********Variable rename********/
foreach (all operations){
 Rename destination to one time use;
}

/********Reduction of the live range of variables********/
foreach (all operations){
 Schedule operation as soon as possible (ASAP);
 if (operation does not have parent nodes){
  Schedule the operation as late as possible;
 }
}

/********Temporal assignment********/
while ( there are operations to be moved){
 foreach (all pages)
  if (current page has more operations than the next page)
   if ((thre are operations which mobility value is not 0)
  &&(the operation can be moved to the next page)){
/*mobility is # of steps which operation can be moved, initial is 10*/
     Move the operation to the next page;
     Substract 1 from its mobility value;
   }
  }
 }
}

/********Page division********/
foreach (all pages){
 if (not meet hardware resource restrictions in current page){
   Insert new empty page in program as the next page;
   Schedule operations in the two pages to become
   the # of operations as same as possibleas;
 }
}
```

Fig.6: Pseudo-Code of Code-Scheduling Algorithm

6 Evaluation

In this section, to evaluate quality of output *pages* and compilation time of our method (code-scheduling + our place-and-route), we compare it with another method based on Simulated Annealing (code-scheduling + place-and-route based on SA).

6.1 Evaluation conditions

The evaluation conditions are described as the followings.

Evaluation index

We implemented our method and the comparative method with C language. Evaluation index are the number of *pages* which these methods output, and their compilation time.

```
foreach (all pages){
  while (there are unselected operations){

/*******Selection of operation*******/
   Select an unselected operation according to the priority;
   Check the operation as selected;

/*******Placing*******/
   Search untired RU on which the operation can be placed;
   if (there is not RU on which the operation can be placed){
     Continue while;  /*go to Selection of operation*/
   }
   else {  /*find the RU*/
     Place the operation on the RU;
   }

/*******Routing*******/
   Try to route data path to the placed operation;
   if ((no routing path)  /*all RU are tried*/
     || (success)){
     Continue while;  /*go to Selection of operation*/
   }
   else{  /*failed in the RU && there are untried RUs*/
     Go to Placing; /*try to replace the operation on another RU*/
   }
  }

/******Page division********/
  if (there are operations failed to be placed or routed){
    Insert a new page into the code as the next page;
    Move the operations to the new page;
  }
}
```

Fig.7: Pseudo-Code of Place and Route Algorithm

Evaluation method

We measured the compilation time by using the time() function in the C library. When the compilation time is too small, we repeated the compilation and use the average of their compilation time.

Evaluation environment

We used gcc 3.3.2 with -O3 option to compile them.

6.2 Place-and-route restrictions

In this subsection, we show the details of the place-and-route restrictions of TEMPO4x5 processor described in Section 2.

- Twenty eight-bit operations can be placed in one *page* at the maximum.
- An operation, whose data width exceeds eight-bit, is placed on multiple adjacent RUs.
- All operations in one *page* must not have any data dependencies on each other and can be executed in parallel.
- To transfer data in one clock, the routing selects the shortest path in the reconfigurable part.

Table.1: SA parameters

Parameter name	Value
Initial temperature T0	10
Minimum temperature Tmin	1
Number of variation N	10
Coefficient of variation A	0.95
Number of overall iteration M	41

- One *page* can have only either one, a load operation or a store operation, because TEMPO4x5 processor has only one memory port.

6.3 Simulated Annealing

We adopt Simulated Annealing (SA) as the comparative method. Though SA needs a lot of time, it can get a good result[16]. Also, SA has high flexibility and it can be applied in many other fields. We think it is easy to apply SA to each *page* in place-and-route.

SA base place-and-route is executed after our code-scheduling. Table.1 shows the SA parameters used in this evaluation. We set the number of iterations for one *page*, to achieve the maximum code quality in practical compilation time. Because SA base place-and-route uses random placement and placement result of one *page* influences other *pages* at random, so the code quality varies widely. To settle the quality variation, we set the number for overall iterations. The following are the SA base place-and-route algorithm we have used.

1. Initialization:

This procedure initializes temperature T as T0, and sets goal value G as the number of the operations in the current *page*. Place-and-route using random numbers generates to initial state X, and it calculates C as the evaluation value of X. C is the number of operations which succeed in place-and-route.

2. Generation of the next state:

This procedure exchanges two operations and generates the next state next, and calculates C_next as the evaluation value of X_next.

3. State transition:

1. If C_next is equal to G, place-and-route finishes the algorithms for the current *page* and goes to the next *page*.
2. If C_next is improved better than C, the next state (X_next → X, C_next → C) is adopted.
3. If C_next is worse than C, △C is set as the difference between C and C_next. If

½e^(△C/T) ≥ R(random numbers: 0 ≤ R ≤ 1), the next state is adopted.

4. Annealing for one *page*:

Procedures 2.Generation of the next state and 3.State transition are iterated N times.

5. Cooling:

T is multiplied by the coefficient of variation A.

1. If T is larger than the minimum temperature T_min, the procedure goes back to 2.Generation of the next state.
2. If T is smaller than T_min, X is the result of the current *page* and the procedure works on the next *page*. In this time, if there are operations which cannot be placed or routed, *page division* is done for them.

6. Annealing for all pages:

Procedures from 1.Initialization to 5.Cooling are done for all pages, and P[M] is stored for the corresponding output pages.

7. Overall iteration:

Procedure 6.Annealing for all pages is iterated M times, and the best P[M] which has the smallest number of pages is adopted as the result.

6.4 Benchmark applications

The following are the applications used as the benchmark in the evaluation. The line numbers present the number of operations in the intermediate code, which are used as their code-size.

(1) Bubble sort (64 lines)

This algorithm sorts 32 eight-bit data.

(2) Bucket sort (141 lines)

This algorithm sorts 32 eight-bit data, and it is used many load/store operations.

(3) Even-odd sort (352 lines)

This algorithm sorts 16 eight-bit data in parallel. This is benchmark with high parallelism.

(4) FEAL[18] (334 lines)

This algorithm is 64 bit common key cryptosystem. This is benchmark with low parallelism.

(5) DCT (296 lines)

This algorithm is used in JPEG encoder etc. This is benchmark with high parallelism. We use one-dimension DCT for 8x8 pixel blocks.

6.5 Evaluation result

Fig. 8 shows the evaluation result by the number of pages. From the result, compared with the comparative method, our method achieves the code quality very similar, the difference of the number of output *pages* was -3.4% - +1.2%. The reason is that our scheduling to balance the number of operations

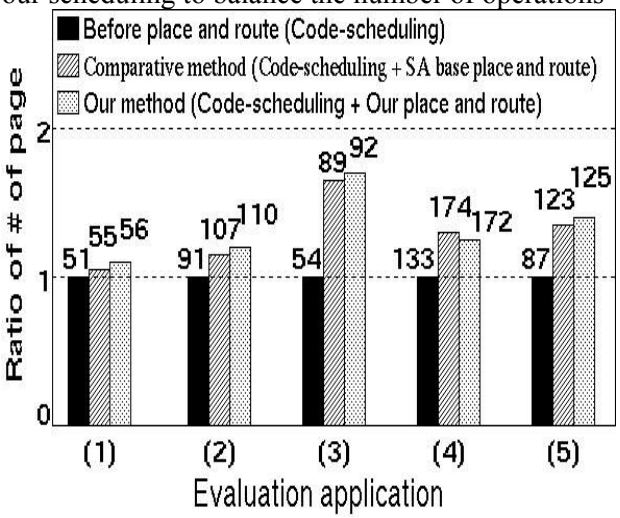

Fig. 8: Result of the Number of pages

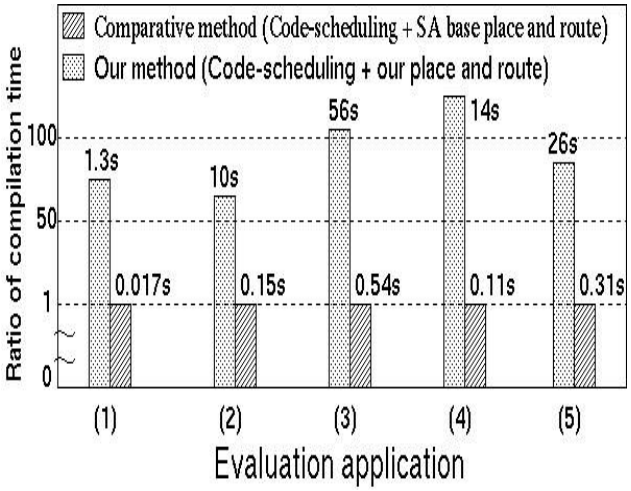

Fig. 9: Result of Compilation Time

and placement policy lead such a good quality in a short compilation time. From this result, we can say that our method achieves equal code quality to SA base place-and-route which achieves the high code quality with the long compilation time.

Fig. 9 shows the evaluation results on compilation time. From the result, compared with comparative method, our method cuts compilation time to 1/128 - 1/67 (in average 1/83). This reason is the followings.

- Our place-and-route using the priority and the given placement order finds its solution faster than SA base place-and-route using random placement.
- Our sequential place-and-route can identify operations which could not be placed and routed. It enables to avoid unnecessary search,

so it can decide to do page division earlier, if the appropriate place-and-route are not found.
- Our place-and-route limits the search space, so it can find solutions in a shorter compile time.

From these results, we can say that our method achieves an efficient mapping with a short compilation time as compared with the conventional method based on SA.

7 Conclusion and future work

In this paper, we show a novel compiler method for dynamic reconfigurable processor based on VLIW model. Our approach has two features. One is code-scheduling which reduce the difficulty place-and-route condition. Another is place-and-route which reduces iteration procedures to get reasonable results. From the evaluation results, our method achieves that the difference of the number of generated configuration data as code quality is -3.4% - +1.2%, and compilation time is cut to 1/128 - 1/67, compared with SA base method. From these results, we can say that our method achieves an efficient mapping in a short compilation time as compared with the conventional method based on SA.

As one of our future works, we plan to develop a compiler for processor having larger hardware resources.

References:

[1] By Michalis, D.Galanis,Gregory.Dimitroulakos, Athanassios. P. Kakarountas, Costas. E. Goutis, Partitioning Applications to Heterogenenous Reconfigurable Hardware, *WSEAS TRANSACTIONS on COMPUTERS*, Issue 10, Volume 4, pp.1289-1296, October 2005.

[2] D.Buell, T.El-Gbazawi, K.Gai, V.Kindratenko, High-Performance Reconfigurable Computing, *IEEE Computer*, Vol.40, No.3, 2007.

[3] DAPDNA, IPFLEX, http://www.ipflex.com/en/.

[4] Ryuji HADA, Kazuya TANIGAWA, Akirra KOJIMA, and Tetsuo HIRONAKA, An Adaptive Compiler method for Scheduling and Place-and-Route for VLIE-based Dynamic Reconfigurable Processor, *Proceeding of the 12th WSEAS International Conference on COMPUTERS*, Part I, pp.61-69, July 2008.

[5] Ahmad Sadegheih, Global Optimisation Using Evolutionary Algorithms, Simulated Annealing and Tabu Search, *WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS*, Issue 6, Volume 1, pp. 1700-1706, December 2004.

[6] Sheeba. V. S, Elizabeth Elias, Design of Two-Dimensional Signal Adapted Filter Banks for Application in Image Processing, *WSEAS TRANSACTIONS on SIGNAL PROCESSING*, Issue 9, Volume 2, pp. 1281-1286, September 2006.

[7] Kazuya Tanigawa, Tetsuo Hironaka, Akira Kojima, and Noriyoshi, Yoshida, A Generalized Execution Model for Programming on Reconfigurable Architectures and an Architecture Supporting the Model, *Conference on Field Programmable Logic and Applications*, Vol.2438, pp.434-443, 2002.

[8] Kazuya TANIGAWA, Tomohiro INOUE, Tetsuo HIRONAKA, Akira KOJIMA, and Noriyoshi YOSHIDA, PARS Architecture Reconfigurable Architecture with Generalized Execution Model---Design and Implementation of its Prototype Processor, *IEICE Trans on Information and System*, Vol.E86-D, No.5, pp.830-840, 2003.

[9] Kazuya TANIGAWA, Tomohiro INOUE, Tetsuo HIRONAKA, and Noriyoshi YOSHIDA, Implementation of the Reconfigurable Processor with Ability of Every-Cycle Reconfiguration and Execution, *COOL Chips V*, Vol.1, pp.162, 2002.

[10] Kazuya TANIGAWA, Takashi KAWASAKI, and Tetsuo HIRONAKA, A coarse-grained reconfigurable architecture with low cost configuration data compression mechanism, *Proceedings of IEEE International Conference on Field-Programmable Technology (FPT'03)*, pp.311--314, December 2003.

[11] Goldstein.S, Schmit.H, Moe.M, Budiu.M, Cadambi.S, Taylor.R and Laufer.R, PipeRench:A Coprocessor for Streaming Multimedia Acceleration, *Proc.26th Annual International symposium on Computer Architecture*, pp.28-39, 1999.

[12] Budiu.M, Cadambi.S, Fast Compilation for pipelined reconfigurable fabrics, *ACM/SIGA 7th International symposium on FPGA '99*, pp. 135-143, 1999.

[13] Goldstein.S, Schmit.H, Budiu.M, Cadambi.S, Moe.M, and Taylor.R, PipeRench:A Reconfigurable Architecture and Compiler, *IEEE Computer*, Vol.33, No.4, pp.70-77, 2000.

[14] Mei.B, Vernalde.S, Verkest.D, Man.H and Lauwerins.R, ADRESS:An architecture with tightly coupled VLIW processor and coarse-

grained reconfigurable matrix, FPL03, pp.61-70,2003
[15] Mei.B, Vernalde.S, Verkest.D, Lauwerins.R, Design Methodology for a Tightly Coupled VLIW/Reconfigurable Matrix Architecture: A Case Study, *DATE04*, Vol.2, pp.1224-1229, 2004.
[16] k.Shahookar, and P.Mazumder, VLSI cell placement techniques, *ACM Computing Surveys*, Vol.23, No.2, pp.143-220, 1991.
[17] A.V.Aho, M.S.Lam, R.Sethi, J.D.Ullman, *Compilers -Principles, Techniques, & Tools-*, Addison Wesley, Second Edition, 2006.
[18] S.Miyaguchi, S.Shiraishi, S.Shimizu, Fast Data Encipherment Algorithm FEAL -8, *Review of the Electrical Communication Laboratories*, Vol.36, No.4, 1988.