Development of Compiler which Supports High-level Programming Language for Dynamic Reconfigurable Architecture DS-HIE

Yasuhiro NISHINAGA, Takuro UCHIDA, Tetsuya ZUYAMA,
Kazuya TANIGAWA, and Tetsuo HIRONAKA
Graduate School of Information Sciences, Hiroshima City University
3-4-1, Ozuka-higashi, Asaminami-ku, Hiroshima, 731-3194, Japan
E-mail : reconf08@csys.ce.hiroshima-cu.ac.jp

Abstract: We have developed the dynamic reconfigurable processor DS-HIE for the streaming processing in our laboratory. The software development environment for the DS-HIE processor was not developed. So it is difficult to evaluate the performance of the DS-HIE processor by using practical applications. Therefore, this paper explains about the development of the compiler for the DS-HIE processor, which supports high-level programming language. The compiler consists of the Front-end part and the Back-end part. Since it will take time to develop a high quality Front-end from the scratch, so we selected the COINS compiler as the Front-end compiler. The Back-end compiler extracts the parts executed by the DS-HIE processor in the input program, and then maps the operations to the Function Units in the DS-HIE processor. After that, the Back-end compiler routes wires between the Function Units. The applications to evaluate the compiler were one dimensional DCT and row processing of LDCP decoding. As a compilation result, the average usage of the function unit was 83%

1. Introduction

In our laboratory, to realize speedup of the streaming processing for small scale systems such as embedded systems, we have developed a heterogeneous multi-core processor Hy-DiSC, which includes a dynamic reconfigurable processor DS-HIE[1]. The DS-HIE processor adopts digit-serial computation and Beneš Network to keep the chip area size small, while preserving high performance by the dynamic reconfiguration. From our evaluation, the DS-HIE processor can achieve more than 28 times higher performance in two dimensional DCT processing, compared with the MeP processor[2] that is a RISC processor developed by TOSHIBA. However, the DS-HIE processor doesn't currently have the software development environment for high-level programming language such as C, so it is difficult to develop a large scale application. To overcome this problem, we developed a compiler for the DS-HIE processor as a part of the development of the Hy-DiSC compiler. The DS-HIE compiler supports high-level programming language so that it will makes us possible to evaluate the DS-HIE processor for various applications.

2. DS-HIE processor

This section describes the overview and structure of the DS-HIE processor.

2.1 Overview of the DS-HIE processor

Figure 1 shows the block diagram of the Hy-DiSC processor. The DS-HIE processor executes the part of a program which can be accelerated by streaming processing, and the RISC processor executes the left over part. The goal of the DS-HIE processor is to achieve small chip area while preserving high performance. The features of the DS-HIE processor are the following.

- The DS-HIE processor executes a program at streaming manner.
- The digit-serial computation manner is adopted.
- The Beneš network is adopted as a routing resource.

First, the performance of the DS-HIE processor is improved by adopting the streaming manner because the DS-HIE processor can execute an application on deep pipeline structure. Second, the digit-serial computation is the computation manner which divides one data into several data and takes several cycles to calculate the data. Therefore, the latency until the computation is done is large, but there is an advantage that the chip area is smaller than traditional parallel computing manner. Further, the feature of small chip area increases the number of operation units in a chip. Last, the Beneš network is a multistage network and provides multiple paths between the source and the destination nodes. The high flexibility of routing by the Beneš network gives us high utilization of operation units.

2.2 Structure of the DS-HIE processor

As shown in Figure 1, the DS-HIE processor consists of Input Buffer, Reconfigurable part, Data Supply Unit, Context Buffer, and Output Buffer. The Data Supply Unit exchanges data between the DS-HIE processor and the RISC processor. The Context Buffer has two entries of context which is a configuration information for Reconfigurable part. Input Buffer and Output Buffer have double buffer structure. The double buffer allows the simultaneous access by the DS-HIE processor and the RISC processor. The Reconfigurable part includes several Operation Stages which consists of Function Units (FUs) and a Beneš Network. The inputs of an Operation Stage are connected to the outputs of previous Operation Stage and outputs of itself.

2.3 Requirement for the compiler

The following are the requirements of the compiler for the DS-HIE processor.

(a) The DS-HIE compiler should support various configurations of the DS-HIE processor.

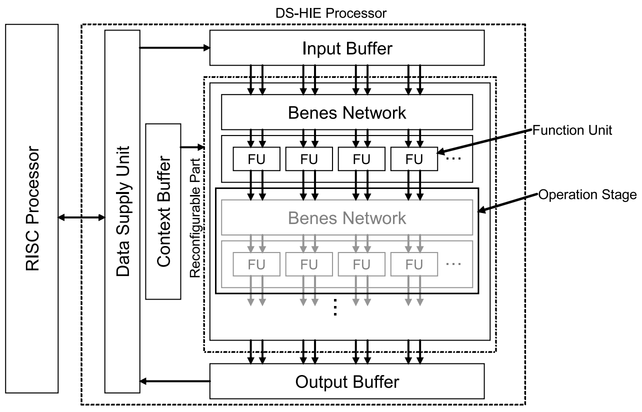


Figure 1. Block diagram of Hy-DiSC processor

(b) The DS-HIE compiler should support high-level languages.
(c) The DS-HIE compiler should support function extensions for optimization facilitate.
Requirement(a) means that the target structure of the DS-HIE processor should be specified by a configuration file which includes the number of the Operation Stage or the number of the Function Units, and so on. Requirement(b) means that to achieve easy development of large scale application. Requirement(c) means to ease the implementation of additional function, like an optimization for the DS-HIE processor.

3. DS-HIE Compiler

In this section, we describe the compilation flow of the DS-HIE compiler and its overviews.

3.1 Modeling of the DS-HIE processor

To give the flexibility over various structures of the DS-HIE processor, we modeled the DS-HIE processor. The modeled structure of the DS-HIE processor is specified by the input file of the DS-HIE compiler. The modeling points are following.

- Number of the Operation Stage
- Number of Function Units in each Operation Stage
- The kind of structure of the routing resource

These three items are specified in the configuration file. The size of the Beneš Network and the number of Input/Output Buffer are automatically calculated by the DS-HIE compiler from the number of Function Units. About the structure of the routing resource, there are possibilities to change the routing resource by several factors like process technologies or evaluations in future. Even if the routing resource was changed, the DS-HIE compiler would be still available because the compiler has been developed to give flexibility on the implementation of the routing algorithm.

3.2 Compilation flow of the DS-HIE compiler

Figure 2 shows the compilation flow of the DS-HIE compiler. The compiler consists of the Front-end part which converts high-level language to intermediate expression, and the Back-end part which generates an executable code for the DS-HIE processor and the RISC processor. The current versions of the DS-HIE compiler lack the module ”Generate RISC code”, so we can only generate the streaming code for the DS-HIE processor.

3.3 Front-end part

The Front-end performs the machine independence optimizations. Basic flow in the Front-end part is following.

- The Front-end part read the code written by a high-level language.
- The Front-end part converts high-level language to intermediate expression.
- The Front-end part optimizes code at machine independent optimization.

Since it will take time to develop a high quality Front-end part from scratch, so we use the part of the developed compiler as the Front-end part for the DS-HIE compiler. The requirement of the target compiler is to support a high-level language and output intermediate expression which are used to describe the result after optimization of machine independence. From these points of view, we selected the COINS compiler[3] as the Front-end part for the DS-HIE compiler. New feature can be added easily to the COINS compiler. The COINS compiler supports high-level language and can output intermediate expression. And, the COINS compiler has been maintained and improved by the Association of COINS Compiler Infrastructure. So, we selected the COINS compiler as the Front-end part for the DS-HIE compiler.

3.4 Back-end part

The Back-end part performs analysis, transformations and optimizations for the DS-HIE processor. The Back-end compiler consists of the following processes.

- Code extraction of the intermediate expression
- Generation of Data-flow Graph
- Mapping
- Routing

In the process ’Code extraction’, the compiler extracts the part of program which can be accelerated by the DS-HIE processor. The part is specified by the prefix of function name which is named by programmers. Then the DS-HIE compiler checks whether the code is correct or not at the Code extraction part.

Next, in the process ’Generation of Data-flow Graph’, the compiler analyzes the operations included in the function which is executed by the DS-HIE processor, and generates the Data-flow Graph.

Third, in the process ’Mapping’, each operation node in the Data-flow Graph is mapped to the Function Unit in the DS-HIE processor. This means the DS-HIE compiler decides which Function Unit executes what kind of calculation.

Last, in the process ’Routing’, to correctly transfer data to each Function Unit, the compiler decides the path on the routing resources.

At next section, we will explain the algorithm of each process in the Back-end part.

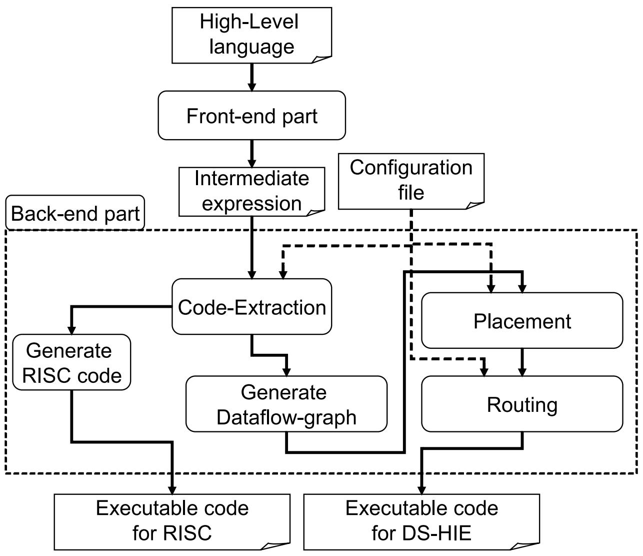
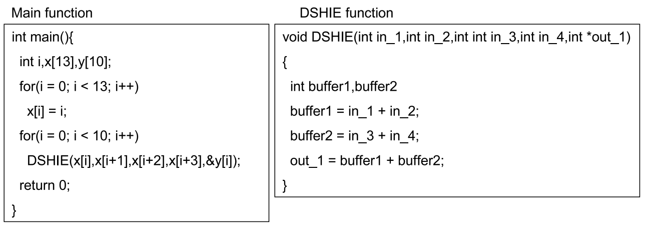


Figure 2. Compilation flow of the Hy-DiSC compiler



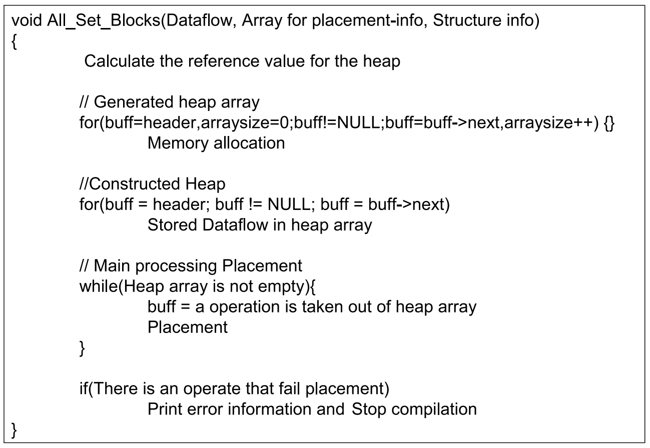
Main function

```
int main(){
  int i,x[13],y[10];
  for(i = 0; i < 13; i++)
    x[i] = i;
  for(i = 0; i < 10; i++)
    DSHIE(x[i],x[i+1],x[i+2],x[i+3],&y[i]);
  return 0;
}
```

DSHIE function

```
void DSHIE(int in_1,int in_2,int int in_3,int in_4,int *out_1)
{
  int buffer1,buffer2
  buffer1 = in_1 + in_2;
  buffer2 = in_3 + in_4;
  out_1 = buffer1 + buffer2;
}
```

Figure 3. Sample C code



```
void All_Set_Blocks(Dataflow, Array for placement-info, Structure info)
{
          Calculate the reference value for the heap

          // Generated heap array
          for(buff=header,arraysize=0;buff!=NULL;buff=buff->next,arraysize++) {}
                    Memory allocation

          //Constructed Heap
          for(buff = header; buff != NULL; buff = buff->next)
                    Stored Dataflow in heap array

          // Main processing Placement
          while(Heap array is not empty){
                    buff = a operation is taken out of heap array
                    Placement
          }

          if(There is an operate that fail placement)
                    Print error information and Stop compilation
}
```

Figure 4. Pseudo code of mapping processing

4. Back-end compiler

In this section, we describe the algorithm of each process in the Back-end part.

4.1 Code extraction of the intermediate expression

To specify the part as a function in the program, the programmer inserts the prefix ”DS-HIE” before its function name. The argument of the function is treated as input/output data which are stored in the Input/Output Buffer of the DS-HIE processor. And the DS-HIE compiler extracts the code which is executable by the DS-HIE processor and checks the code whether the codes is executable by the DS-HIE processor specified by the configuration file. Figure 3 shows the sample program which includes the accelerated part by the DS-HIE processor.

4.2 Generation of Data-flow Graph

At first, we describe the structure of the Data-flow Graph. An operation in the Data-flow graph is described as operation nodes. One operation node has one operation information. The operation nodes of the Data-flow Graph have two inputs and two destinations. So, one operation result can be used as input data for two operations at maximum.

Next, we describe the steps which generates the Data-flow Graph. The DS-HIE compiler finds the part for execution on the DS-HIE processor, and then checks whether the part can be executed on the DS-HIE processor specified by the configuration file. When the DS-HIE compiler extracts an operation from the part, the DS-HIE compiler makes an operation node and adds the node to a list. And the DS-HIE compiler construct the Data-flow Graph based on input and output information of the operation node.

4.3 Mapping

When the Data-flow Graph is allocated to the Function Unit of the DS-HIE processor, following considerations are needed.
• Consideration on selection order of the operation node in the Data-flow graph to mapped.
• Consideration on selection order of the Function Unit to map the selected operation node.
We adopted an ASAP method for the operation selection algorithm which decides the order of operation to map. ASAP method is the algorithm which selects operation node at the order of the distance from the Input data.

The mapping algorithm maps operations to Function Unit, selected in the order of the distance from the Input Buffer. This means that the Function Units with shorter distance from the Input Buffer has the higher priority on the selection. Figure 4 shows pseudo code for the mapping algorithm of the DS-HIE compiler.

4.4 Routing

The routing algorithm of the DS-HIE compiler is looping algorithm as well-know routing algorithm for the Beneš Network. As a result of evaluations and considerations in future, the routing resources in the DS-HIE processor can be changed to another structure. For these changes, we implemented the routing function as an independent function.

5. Evaluation

We compiled several application programs to evaluate the performance of the developed DS-HIE compiler, and analyzed the compilation results.

5.1 Structure of the target DS-HIE processor

Table 1 shows the structure of the target DS-HIE processor in this evaluation. As the Table 1 shows, the number of the Operation Stage is 2, the number of the FUs in each Operation Stage is 32, and the size of the Beneš Network is 64*64.

5.2 Application programs

The application programs to evaluate the performance of the DS-HIE compiler are one dimensional DCT and row processing of LDCP decoding[4]. The DCT is the algorithm to convert a disintegration signal into a frequency domain. The DCT includes many multiply operations and have high parallelism. The LDPC decoding is the decoding algorithm with an error correcting capability near the Shannon limit. The LDPC decoding consists of initialization, row operations, column operations, calculations of code words, and parity check. In the five processing routine, the row operations are one of the heaviest process so we focused on the row operations.

5.3 Evaluation methods

The usage of the FU was evaluated as a compilation result of each application. The purpose is to evaluate whether the DS-HIE compiler can map the operations in the program effectively to the DS-HIE processor. Two kind of the usage of the FUs is evaluated, the usage of the over all FU (Usage_all) and the usage of the FUs in each Operation Stage (Usage_part). The usage of the over all FU is calculated by

$$Usage\_all = \frac{Number\ of\ the\ used\ all\ FU}{Total\ number\ of\ FU} * 100$$

And the usage of the FUs in each Operation Stage is calculated by

$$Usage\_part = \frac{Number\ of\ used\ FUs}{Number\ of\ FUs\ in\ each\ Operation\ Stage} * 100$$

5.4 Evaluation result

Table 2 shows the compilation result of the DCT function and the row processing function of LDCP decoding. First, as the DCT compilation result, the number of the used Function Unit is 59. While the available number of the Function Unit is 64, so the usage of the over all Function Unit is 92%. The usage of the Function Unit in each Operation Stage Number #1 and #2 is 100% and 75% respectively. Next, as the LDPC compilation result, the number of the used Function Unit is 48. While the available number of the Function Unit is 64, so the usage of the over all Function Unit is 75%. The usage of the Function Unit in each Operation Stage Number #1 and #2 is 100% and 50% respectively. From the result, the average usage of the Function Unit is 83%. In both application programs, the DS-HIE compiler could use all Function Units in the first Operation Stage, which is achieved by high flexibility of the Beneš Network. So, we can say that our developed compiler achieves enough high usage of the Function Units.

Further, we can decrease the time of the program development for the DS-HIE processor. In the development of the application, it takes several days to develop it by hand coding, but with the compiler it was reduced to hours.

Table 1. Specification of the DS-HIE processor

Size of the Beneš Network	64 inputs and 64 outputs
Number of the Operation Stages	2
Number of FUs in each Operation Stage	32
Number of total FUs	64

Table 2. Compilation result

Program name	Evaluation item	Operation Stage Number #1	Operation Stage Number #2	Total
DCT	Number of the used FUs	32	27	59
	Usage of the FUs(%)	100	84	92
LDPC	Number of the used FUs	32	16	48
	Usage of the FUs(%)	100	50	75

6. Conclusion

We developed a compiler which supports high-level programming language to compile the program for the dynamic reconfigurable processor DS-HIE. As compilation result of a one dimensional DCT program and row processing of LDCP decoding, the developed compiler can map and route the program with high usage (83%) of the resources in the DS-HIE processor. And we can decrease the time to develop the program for the DS-HIE processor. But, this compiler outputs only the executable code for the DS-HIE processor. So, we plan to develop the output module for the executable code for the RISC processor.

Acknowledgement

This research was partially supported by the Ministry of Education, Culture, Sports Science and Technology (MEXT), Grant-in-Aid for Scientific Research, 18300016, 2006.

References

[1] T. Zuyama, K. Tanigawa, and T. Hironaka, "Development of DS-HIE Architecture," Proceedings of the ITC-CSCC2007, Vol.1,pp.47-48, 2007

[2] Microcomputer MeP(Media embedded Processor), http://www.semicon.toshiba.co.jp/eng/product/micro/mep/index.html

[3] COINS a Compiler Infrastructure Project, http://www.coins-project.org/

[4] Y. Imai, K. Shimizu, N. Togawa, M. Yanagisawa, and T. Ohtsuki,gA multi-rate compatible irregular LDPC decoder enhancing column operation parallelism(in japanese),hin IEICE Tech. Rep., ser. RECONF2007-46, vol.107, no.342, pp.19-24, 2007