# Parallelization of Loops with Complicated Data Dependency and its Experiment

Kyoko Iwasawa
Computer Science dept.
Takushoku University
Hachioji, Tokyo Japan, 193-0985
E-mail : kiwasawa@cs.takushoku-u.ac.jp

***Abstract*— This study discusses a loop parallelizing method for compilers in a multi-core architecture that enables to detect fine grain parallelism. Our method involves generating parallelized loops from nested loops carrying complicated data dependencies. These loop transformations are formalized by matrix operations. They enable the original loop indexes to be expressed using new loop indexes so that compiler does not need to make any changes in loop body. Our experiments have determined that bubble sort programs can also be parallelized effectively by using our proposed method.**

***Keywords-fine grain parallelism; parallelization; data dependency ; compiler; double- nested loops;***

## I. INTRODUCTION

Multi-core architecture is being widely use; however sometimes multiple central processing unit (CPUs) are not used efficiently for sequential programs. In particular, in some instances, loops with complicated data flow dependency are designed to execute in parallel without any synchronization among compilers (or such types of system software).

To optimize multi-core architecture, developers have been attempting to speed up the execution times of nested loops, which consume a large fraction of execution time, by mean of parallelization.

One of the method to parallelize double-nested loops is the wave-front-line method [1]. This method analyzes not only the inner loop data flow but also the outer loop data flow, in order to identify the line where loop bodies can be executed in parallel. This method uses various synchronization controls (e.g., data passing, lock-unlock, etc.), and the overhead of these synchronization is too high for multi-core and Single Instruction Multiple Data (SIMD) architecture (e.g., packed operation or vector operation).

The characteristic of our study is the restructuring of double-nested loops that may include complicated data dependency constraining loop exchange and splitting. Our method generates a parallel loop by shearing conversion on the double-loop iteration space and then exchanging loops. Our method involves shearing along the inner loop index. This method does not seem to have been discussed previously, in literature and is particularly useful in case of fine grain parallelism. In our study we show how compilers should generate parallelized codes, so that loops with complicated data dependencies can be parallelized and vectorized without any synchronization, this would lead to reduces overheads in multi-core or SIMD architecture.

The rest of this paper is organized as follows: Section II describes parallel conversions. Section III discusses the result of our experiments. Section IV describes the related works and Section V concludes this article.

## II. PARALLELIZING CONVERSION

We first discuss the case of the loop which includes separable data dependence between inner and outer loop. Then, we discuss the parallelizing conversion of the more complicated case; this is due to the inseparable data dependence between inner and outer loop, is shown.

### A. *Separable data dependence between inner and outer loop*

In double-nested loop, when both inner and outer loops carry data dependence, individual loop bodies cannot be executed in parallel neither along the inner loop index nor the outer loop index. In the case where inner loop carried data dependencies and outer loops carried data dependencies are independent, it is easier to identify the wave front line, where loop bodies can execute in parallel. Fig. 1 shows the loop iteration space and separable loop carried dependencies. As is clear from Fig. 1 parallelism takes place on a diagonal line.

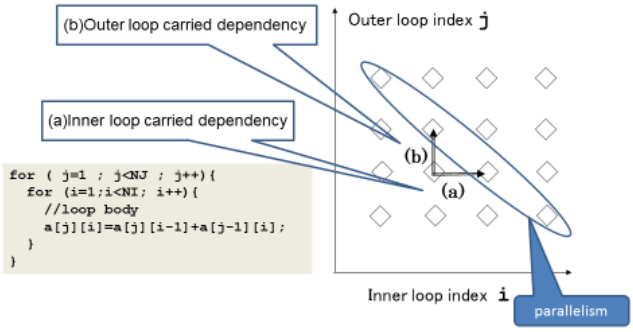


Figure 1 The wave front line on the iteration space

### B. *Inseparable data dependence between inner and outer loop : critical data dependence*

If both inner and outer loops carry critical data dependence, then it bothers neither loop parallelization nor loop exchange. In such a case, loop body on diagonal line cannot execute in parallel.

 ISBN: 978-1-61208-513-5 

To detect parallelism, we generate a new inner parallel loop. The loop bodies are on the line where they can be executed independently, using the following

(1) Find critical data dependencies and calculate the delay by its loop carried iteration number.
(2) Exchange the inner loop and the outer loop
(3) Insert the calculating code of the new loop indexes(I, J) from the original loop indexes(i, j)

$$\begin{pmatrix} J \\ I \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ delay & 1 \end{pmatrix} \begin{pmatrix} j \\ i \end{pmatrix} = \begin{pmatrix} j \\ delay * j + i \end{pmatrix}$$

This result in a parallel inner loop is generated without any of the loop body conversion as can be seen in Fig. 2

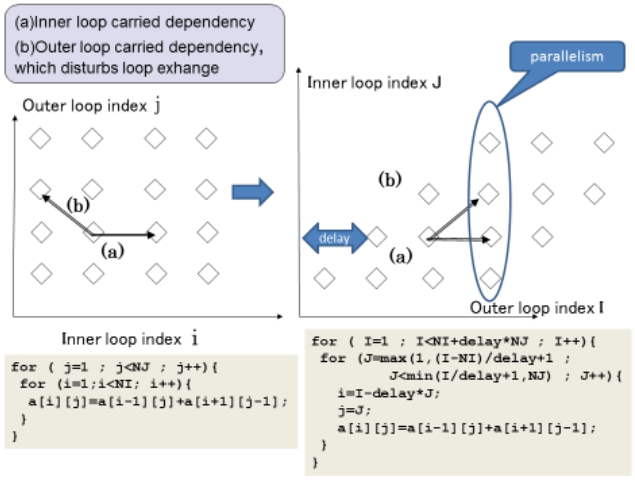


Figure. 2 Shearing conversion along to inner loop index on the iteration space

## III. EXPERIMENT OF BUBBLE SORT PROGRAM

This section discusses the experiment of parallelizing bubble sort program, which is executed serially in general. It accesses one dimensional array in a double-nested loop, therefore there is critical data flow dependency. In addition the wave-front-line (Section II(a)) cannot be detected.

```
(1) Original bubble sort program
void BubbleSort(float A[], int N)  {
 for (int j=0; j<N-1; j++) {
  for (int i=0; i<N-j; i++)
  if (A[i]>[A[i+1])    swap(A[i], A[i+1]);
  }
 }
```

(2) After parallelized bubble sort program delay=2 (distO=1, distI=-2)

```
void BubbleSort(float A[], int N){
 for (int I=0; I<2*(N-1); I++) {
   int initial=max(0 , J-(N-1));
   int last=min(I/2, N-1);
   for (int J=initial;J<=last;J++){  /*  parallel */
     int i=I-2*J;
     if (A[i]>A[i+1])   swap(A[i], A[i+1]);
   }
 }
}
```

Figure. 3 Parallelizing of bubble sort program

The inner loop of Fig. 3(2) can be executed in parallel. Some parameters from the result of data flow analysis are necessary to fill the template (Fig. 2.), and the OpenMP direction is inserted. The converted program was compiled by Intel OpenMP C compiler.

Fig. 4 shows the execution time of Fig. 3(2) program using Intel i7 quad core CPU. When the input data is sufficiently large, it takes half the time on four parallel. The paralle execution time can not be reduced down to a fourth of serial execution time, because outer loop length of parallel program becames two times of the serial program by sheraing conversion.

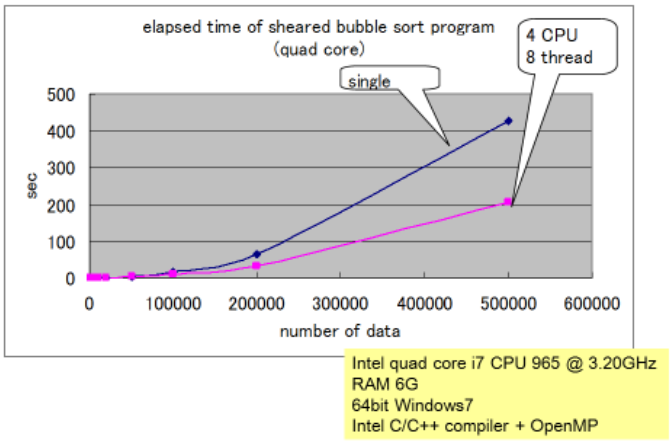


Figure. 4 Execution time of parallelized bubble sort program

## IV. RELATED WORKS

There is a lot of previous work. Array data flow analysis has been studied widely [1][2][4][5][7]. Wolf [1] showed loop skewing by wave front line. Kim [2] showed loop parallelization by using wave front method.

Our study is different from them at the following points. One of them is preparing two shearing methods and choosing suitable one. Shearing along the inner loop index has not been studied, but we notice that it has some advantages [6]. This article shows the result of the implementation by automatic translator of these nested loop conversion [6][7].

## V. CONCLUSION

This study presents a parallelizing method for nested loops for compiler. The compiler makes inner-most parallel and vector loop from nested loop with complicated loop carried data dependency. The new parallel loop enables the expression of original loop indices using new loop indices without requiring the compiler to make any changes in the loop body. The parallelizing translator based on COINS-project [3] has been developing, and it will generate

 ISBN: 978-1-61208-513-5 

parallelized code from programs with more complicated data dependencies automatically, in the future.

## REFERENCES


[1] Wolfe, M., Loop Skewing: The Wavefront Method Revisited, International Journal of Parallel Programming, Springer Netherlands, pp.279-293, (1986).

[2] Kim, K., and Nicolau, A., Parallelizing tightly nested loops, Proceedings of Parallel Processing Symposium, pp.630-633 (1991).

[3] http://coins-compiler.osdn.jp/international/index.html (2016.9.1)

[4] Dulong,C., Krishnaiyer,R., Kulkarni, D. Lavery, W. Li, J. Ng, and D. Sehr, An Overview of the Intel IA-64 Compiler, (2005).

[5] Vasilache, N., Bastoul, C., and Cohen, A., Polyhedral Code Generation in the Real World, proceedings of 15th International Conference CC2006, pp.185-201, (2006).

[6] Iwasawa, K. and Mycroft, A., Choosing Method of the Most Effective Nested Loop Shearing for Parallelism, Proc. Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies, pp.267-276, (2007).

[7] Chakilam, K. C., Representing and Minimizing Multidimensional Dependencies. M.S.C.S. Thesis, Dept. of Computer Science, The University of Akron, (2009).


 ISBN: 978-1-61208-513-5 