数理モデル化と問題解決 30－5
（２０００．６．２６）

Collaboration of Parafrase–2 and NaraView for Effective Parallelization Supports

Masayo Haneda * Mariko Sasakura † Umpei Nagashima ◇
Yoshitoshi Kunieda ‡ Kazuki Joe *

* Nara Women's University † Okayama University
◇ National Institute for Advanced Interdisciplinary Research ‡ Wakayama University

To use parallel computer systems effectively, users need to reconstruct their sequential applications into parallel programs, and such parallelization is not easy work for general users. Parallelizing compilers were developed to solve this problem, but it is still quite difficult to parallelize applications correctly, efficiently, and automatically. For this reason, some parallelization support tools are desired. NaraView, as one of the support tool, visualizes a given program by extracting internal information from a parallelizing compiler Parafrase–2. In this paper, we validate the usefulness of NaraView by parallelizing a real application.

NaraView の Parafrase-2 との連携における効果的な並列化支援

羽田 昌代 * 笹倉 万里子 † 長嶋 雲兵 ◇
國枝 義敏 ‡ 城 和貴 *

* 奈良女子大学 † 岡山大学
◇ 産業技術融合領域研究所 ‡ 和歌山大学

並列計算機を効果的に使用するために，アプリケーションの並列化を自動で行う自動並列化コンパイラが開発されているが，これを用いても正確で効果的な並列化を行うことは極めて難しい．このため自動並列化コンパイラには，その使用を支援するツールが開発されていることが多い．我々が，これまでに開発している並列化支援ツール NaraView は並列化コンパイラ Parafrase-2 から得られる中間表現の情報を抽出し，与えられたプログラムの視覚化を行う．本稿では実アプリケーションの並列化を通して NaraView のより有効な使用について述べる．

1 Introduction

Recently, parallel computers have become commonly used and have taken the place of expensive vector-processor based supercomputers, however many numerical calculation users still rely on vector processor computers. It is hard for these users to reconstruct their conventional numerical applications into a form for scalar-parallel computers. This lead to the development of parallelizing compilers that transform sequential programs into a parallel form automatically. Parallelizing compilers analyze programs for efficient parallelization, but the compilers can not yet find optimal transformations. Therefore, the user must specify the transformations to be applied.

We developed NaraView[4] as a parallelization support tool for Parafrase–2[3] which is a parallelizing compiler developed at the Center for Supercomputing Research and Development of the University of Illinois at Urbana-Champaign. NaraView visualizes the hierarchical task graph (HTG)[1], the intermediate representation of Parafrase–2.

In this paper, we parallelize a real application and compare the results of three parallelization methods: with the default passfile, with a passfile generated by using information from NaraView, and with hand optimization on the source code as well as the modified passfile. Then, we will show the efficacy of visualized program information for parallelizing real applications with NaraView.

We use the extended Huckel calculation program that is one of the program for chemical calculation written in Fortran77 as the real application. The application consists of 8 subroutines. We select the subroutine hoqrv2.f that does most of the calculation because the other subroutines are quite simple.

The organization of the paper is as follows. Section 2 introduce to NaraView. In Section 3, we parallelize the application with Parafrase–2 and NaraView. Finally, the result of the parallelization are evaluated in Section 4.

2 NaraView

NaraView[2] is a program visualization system for parallelizing compilers. It visualizes source codes obtained from the Parafrase–2 parallelizing compiler for the support of parallelization. NaraView consists of the four following views:

1. *Program Structure View*
2. *Source Code View*
3. *Hierarchical CFG View*
4. *Data Dependence View*

3 Parallelization with NaraView

In this section, we demonstrate the parallelization of an application program with Parafrase-2 and NaraView.

3.1 Parallelization with the default passfile

First, we use the default passfile that is attached to Parafrase–2 to parallelize the application. The program that is parallelized by Parafrase–2 with the default passfile is visualized by NaraView as shown in Fig.1. In Fig.1, two loops are transformed into a parallel form.

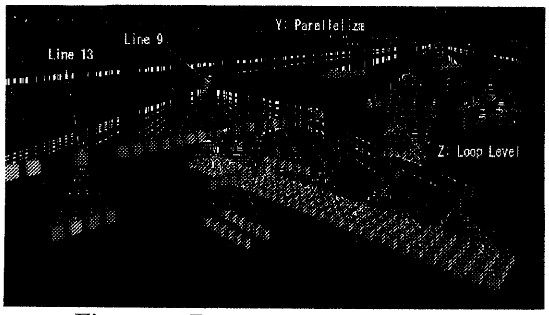


Figure 1: Program structure view

3.2 Parallelization with a Modified Passfile

We modify the default passfile using the information obtained from NaraView. The result of the investigation suggests effective transformations for each loop that is not yet parallelized.

For example, the loop of line–13 is taken from them. The *data dependence view* is shown in Fig.2. The characters shown in the figure correspond to variables of the program. The

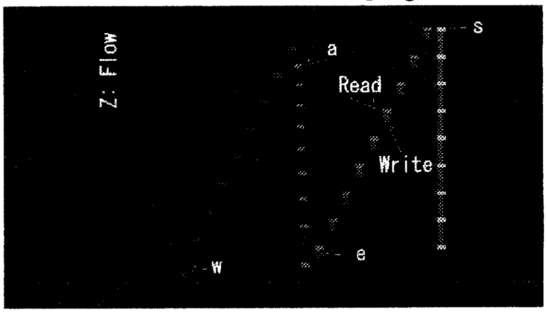


Figure 2: Data dependence view(line–13)

view indicates that there is a loop-carried dependence at variable s. s is a scalar variable, because cubes that express variable s are arranged vertically to the xy-plane. If the shape of the vertical poll is turned into stairs, the statement including s would be parallelized. This transformation is known as **scalar expansion**. In this case, **scalar expansion** can be applied to s. Because the cubes that express variable e are arranged like stairs, it indicates that e is an array. The poles between adjacent cubes represent true dependence in each loop iteration. The two accesses that cause the dependence appear at different statements in the loop. Because the dependence is a true dependence, these statements can be distributed to different loops. After **loop distribution**, they can each be parallelized.

In the same way, we apply such techniques to other loops and add these two transformations into the passfile.

We parallelize the program with the new passfile and visualize the result with NaraView again (Fig.3).

We find that two more loops can be parallelized, the loop of line–13 and the loop of line–69, by using **loop distribution**. In this case, parallelization by **scalar expansion** seems not to be effective. The rest of the non–parallelized loops should be examined with other kinds of transformations to gain more parallelism.

3.3 Parallelization with hand optimizations

We find loops to be parallelized, while it is difficult to parallelize them with Parafrase-2 only. Therefore, we modify the source code directly to eliminate data dependence.

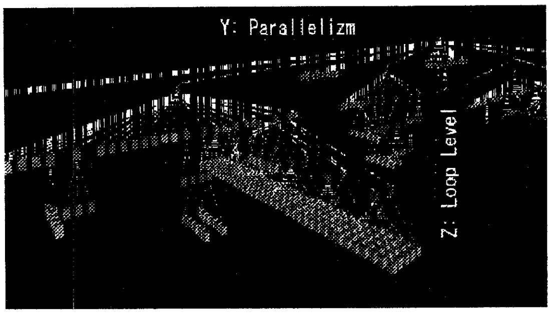


Figure 3: Parallelization with the modified passfile

We next look at the loop at line–21, because this loop is the first loop investigated in Section 3.2. In the section, it was expected all statements in this loop would be parallelized. The data dependence at this part is shown in Fig.4. At first, we recognize that there are true

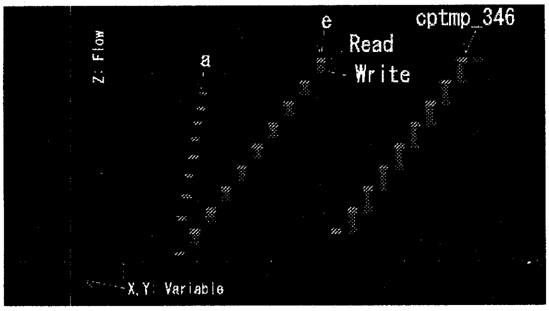
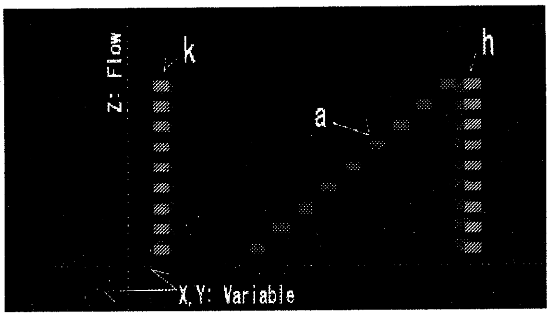


Figure 4: Data dependence view(line–21)

dependencies on array e, and the dependence can be eliminated by re-arranging each statement in the loop to belong to different loops. Although we have added **loop distribution** to the passfile, the **loop distribution** function of Parafrase–2 does not recognize the loop to be fully distributable. Therefore, we modify the source code to distribute the loop and to generate two separated loops. By the modification, one of the loops is parallelized, but the other loop is still not parallelized. The latter loop has a loop-carried dependence, and it requires more modification of the source code to be parallelized.

But the modification produces one more loop for the calculation of the summation. It is difficult to determine whether the modification is effective, because it relies on the execution overhead of the added loop. We did not modify this loop for this reason. The *data dependence view* shown in Fig.5 shows that the loops at line–84,103, and 118 do not have any data dependencies that prevents parallelization. Since Parafrase–2 does not recognize the above loops, we modify these loops to be parallelized. In the same way, we parallelize the rest of non–parallelized loops. Finally, we obtain the parallelized program.



```
    DO 348 i = k + 1,n          84
      a(i,k) = a(k,i) * h       85
348 CONTINUE                    86
```

Figure 5: Data dependence view(line–84)

4 The Effect of Visualization

In this section, we validate the parallelizations applied in Section3.

At first, we compare the *program structure view* of the final parallelized program shown in Fig.6 and the *program structure view* of the program parallelized with the default passfile. The comparison of them with Table1 shows the increase in parallelism. To understand the

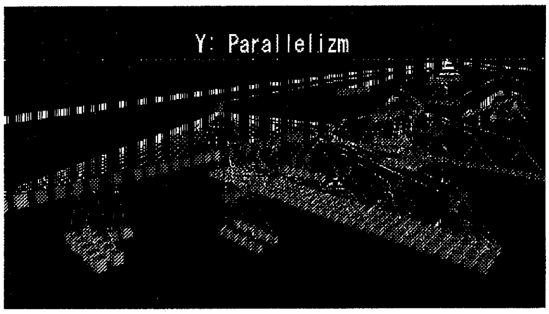
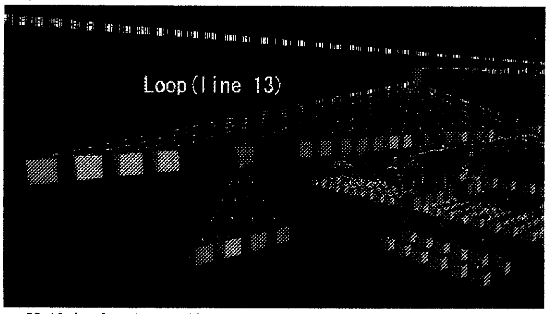


Figure 6: Parallelization with hand optimization

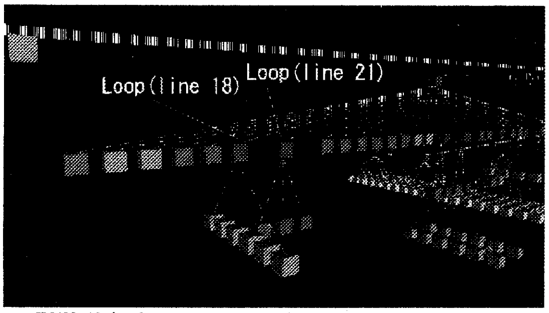
	Section 3.1	Section 3.2	Section 3.3
line-13	$\frac{3n^2-21n-6}{2}$	n^2+2n-8	$5(n-2)$
line-69	n^2-n-1	$\frac{n^2-n+1}{2}$	2

Table 1: The number of loop iterations



```
     DO 10 j = k + 1,n     13
      w(j,1) = 0.d0        14
      e(j) = a(k,j)        15
10    s = e(j) * e(j) + s 16
```

Figure 7: Parallelization with the default pass-file



```
     CDOALL 10 j = k + 1,n               18
      w(j,1) = 0.d0                      19
10   CONTINUE                            20
     DO 347 j = k + 1,n                  21
      e(j) = a(k,j)                      22
      cptmp_346(j) =e(j)*e(j)
     1                 +cptmp_346(j-1) 23
347  CONTINUE                            24
```

Figure 8: Parallelization with a modified pass-file

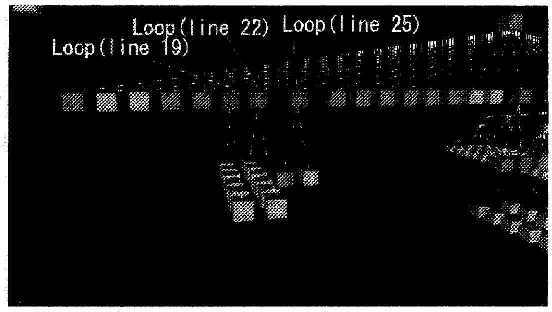
difference between the three results, we zoom in to each parallelized part of the transformed programs in each *program structure view*.

The *program structure views* near the cube of line–13 and the corresponding codes are shown at Fig.7, Fig.8, and Fig.9. They describe the parallelization process of the loop at line–13.

5 Conclusion

In this paper, we described the effectiveness of visualization for parallelization support. We parallelized a real application of the extended Huckel calculation with three parallelization methods to demonstrate the usefulness of the visualization tool, NaraView.

The comparison of the three methods demonstrated the interaction between users and parallelizing compilers improving the parallelism of the program, and it turned out that



```
     CDOALL 10 j = k + 1,n               19
      w(j,1) = 0.d0                      20
10   CONTINUE                            21
     CDOALL 347 j = k + 1,n               22
      e(j) = a(k,j)                      23
347  CONTINUE                            24
     DO 390 j = k + 1,n2                 25
      cptmp_346(j)=e(j)*e(j)
     1                 +cptmp_346(j-1) 26
390  CONTINUE                           27
```

Figure 9: Parallelization with hand optimization

it is efficient to use the visualization of program information as parallelization support.

Additionally, we found some patterns of program information; specifically, data dependencies that match particular transformations. It may predict the existence of individual relation of data dependence patterns and transformation methods. Investigating and formalizing this relation is our future work.

References

[1] M. Girkar and C. D. Polychronopoulos. The hierarchical task graph as a universal intermediate representation. *International Journal of Parallel Programming*, 22(5):519–551, 1994.

[2] M. Haneda, M. Sasakura, U. Nagashima, Y. Kunieda, and K. Joe. Collaboration of parafrase-2 and naraview for effective parallelization supports. In *Proceedings of PDPTA (will appear)*, 2000.

[3] C. D. Polychronopoulos, M. Girkar, M. R. Haghighat, C. L. Lee, B. Leung, and D. Schouten. Parafrase-2: An environment for parallelizing, partitioning, synchronizing, and scheduling programs on multiprocessors. In *International Conference on Parallel Processing, Vol. 2*, pages 39–48, 1989.

[4] M. Sasakura, K. Joe, Y. Kunieda, and K. Araki. NaraView: An interactive 3D visualization system for parallelization of programs. *International Journal of Parallel Programming*, 27(2):111–129, 1999.