I.J. Engineering and Manufacturing **2011, 4, 59-64**
Published Online August 2011 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijem.2011.04.09

Available online at http://www.mecs-press.net/ijem

Parallelization of Needleman-Wunsch Algorithm Based on Software Pipelining

Hanwen Hu, Zhenzhou Ji

Department of Computer Science and Engineering, Harbin Institute of Technology, 150001 Harbin, China

Abstract

Sequence alignment is one of the most important algorithms that analyzing massive biological information. In modern bioinformatics, it plays an important role in field of serching for similar sequences, predicting sequence information of unkown sequence, looking for specific position of sequence, predicting protein structure and so on. Needleman-Wunsch algorithm is the earliest global alignment algorithm, it gets widely application with its accuracy, however, it has a high time complexity and its speed is slower. This paper adopts software pipelining technique to optimize Needleman-Wunsch algorithm with parallelization, and OpenMP which is the industrialized standard of shared memory programming is used to parallelize it. The performance of Needlelman-Wunsch algorithm can get a great improvement with the optimization.

Index Terms: Bioinformatics; sequence alignment; Needleman-Wunsch; software pipelining; OpenMP

1. Introduction

Sequence alignment[1][2] is a process of lining up two or more sequences for the purpose of assessing the degree of similarity and the possibility of homology based on specific scoring rules. It is the main content in the early formation of bioinformatics. Sequence alignment can implement serching for similar sequences, predicting sequence information of unkown sequence, predicting genome, looking for specific position of sequence, predicting protein structure and so on. It is of great significance to find information in the function, structure and evolution of biological sequences.

Sequence alignment provides an efficient way to determine whether two sequences have sufficient similarity[3], we can make a further judgement that the two sequences probably share an evolutionary ancestor with the calculated similarity, in other words, they have a homology[3]. The so-called similarity is a quantitative result that is obtained according to certain algorithm, while the homology is a sense of evolution, that is to say, whether two sequences are homologous needs verification of evolutionary facts. The pair-wise sequence

alignment may reflect a common evolutionary ancestor, or represent the same structure, but there is no inevitable relationship in evolution probably[6].

Sequence alignment contains global alignment[3] and local alignment[3]. Global alignment is that all symbols of sequence which is to be studied take part in comparison, as well as all symbols of sequence carry on arranging and scoring, and the results alignmented with the same length of each sequence, such as Needleman-Wunsch algorithm[4][8]. Global alignment is that all symbols of sequence take part in comparison, but only symbols of the fragments with high score in sequence carry on arranging and scoring, such as Smith-Waterman algorithm[5].

2. **Needleman-Wunsch algorithm**

2.1. Dynamic programming

Dynamic programming solves the whole problem according to combining solution of sub-problem and often applies to optimization problems, that is to say, it makes a set of choices to achieve an optimal solution. The main idea of sequence alignment with dynamic programming is that if a path ends at a point on the optimal path, it is the optimal path of starting point to this point, that is to say, any sub-path that terminate on one point of the optional path must be the optional path itself that end at this point[7]. Thus, the optional path can be made by linking up with each optional sub-path.

2.2. Needleman-Wunsch algorithm

Needleman and Wunsch proposed a global alignment algorithm based on dynamic programming in 1970. It is the earliest pair-wise sequence alignment algorithm, and gets widely application with its accuracy, both its time complexity and space complexity are $O(n^2)$. The main idea of the algorithm to solve sequence alignment problem is that using iterative method to calculate similarity score of two sequences and store them in a score matrix, and then backing to find out the optional alignment sequence based on the matrix.

We use M to represent score matrix, $M_{i,j}$ to represent the element in the i-row and j-colum of M, S(a, b) to represent the comparison score between character a and character b. The value of S(a, b) is computed with (1), and each value of element in score matrix is calculated according to (2), and then backing to find out the optional alignment sequence.

$$S(a,\ b) = \begin{cases} 1 & a = b \quad a \text{ and } b \text{ are not space} \\ -1 & a \neq b \quad a \text{ and } b \text{ are not space} \\ -2 & \text{one is space between } a \text{ and } b \end{cases} \tag{1}$$

$$M_{i,\ j} = \begin{cases} M_{i-1,\ j} + S(s_i,\ \_) \\ M_{i-1,\ j-1} + S(s_i,\ t_j) \\ M_{i,\ j-1} + S(\_,\ t_j) \end{cases} \tag{2}$$

3. Openmp

In 1997, computer hardware and software manufacturers defined a industry standard protocol with shared memory programming application interface, that is OpenMP, it overcomes poor portability and extendibility that is a disadvantage of parallel problem for a long time. OpenMP is a parallel programming model shared memory. Parallelization with shared memory is easy and flexible to program, Its development cycle is shorter, and it has a higher parallel efficiency.

OpenMP contains a set of compiler guided statements, runtime library routines and environment variables. It is mainly in parallelization of loop and works with standard C, C++ and Fortran, its instructions extend the model of ordered structure of original program with single program multiple data structure, shared task structure, synchronized structure and surport of shared or private data. In OpenMP, each compiler guided instruction begins with "#pragma omp", "parallel" instruction defines a parallel domain, it will creates some other thread to execute the code in parallel, and "parallel for" can implement that each iteration of for loops is executed in parallel. "Private" clause indicates each thread has its own copy of "i", specific usage of the sample code with C is as follows.

```
#pragma parallel for private(i)
for(i=0; i<n; i++)
printf("Hello World!\n");
```

4. Parallelization of Needleman-Wunsch algorithm

4.1. Software pipelining

Software pipelining is a traditional optimal technique that reforms the loop so that a faster execution rate is realized[9]. Iterations are executed in overlapped fashion to increase parallelism. This technique is a good method to solve data dependences. Its main idea is that removing data dependences between iterations of a loop by adjusting the execution order of operations of each iteration, making the different iterations can be executed in parallel by reconstructing the loop body to improve the performance of program.

4.2. Optimized Needleman-Wunsch algorithm with software pipelining

The time consumption of Needleman-Wunsch algorithm is mainly in calculating of its score matrix. The value of each element in score matrix is calculated based on its left adjacent element, above adjacent element and diagonal element, it has a significant data dependences between each iteration, the pseudo-code of its calculation is as follows.

```
for(i=1; i<=n; i++)
    for(j=1; j<=m; j++)
        Mi,j = Max{ Mi-1,j + S(si, _), Mi-1,j-1 + S(si, tj),
                    Mi,j-1  + S(_, tj)};
```

We take internal loop as the operations of outer loop to unroll the outer loop as Fig. 1, we use each row to represent a time quantum and each column to represent the iteration. In score matrix, the elements under the same line as in Fig. 2 can be calculated at the same time. That is to say, in Fig. 1, after unrolling the loop, synchronous execution of the operations of each iteration can be implemented at the same time quantum. With this, we can improve the processing rate of score matrix. Optimized code by software pipelining is as below. We assume the value of m is smaller than n.

```
for(i=1; i<=n+m-1; i++){
  if(i<=m)
  {
  for(j=1; j<i+1; j++)
      M_{i+1-j,j} = Max{M_{i-j,j} +S(s_{i+1-j}, _), M_{i-j,j-1} + S(s_{i+1-j}, t_j),
                   M_{i+1-j,j-1}+S(_, t_j)};
  }
  else
  {
  for(j=m; j>=i-n+1; j--)
      M_{i+1-j,j} = Max{M_{i-j,j} +S(s_{i+1-j}, _), M_{i-j,j-1} + S(s_{i+1-j}, t_j),
                   M_{i+1-j,j-1}+S(_, t_j)};
  }
}
```

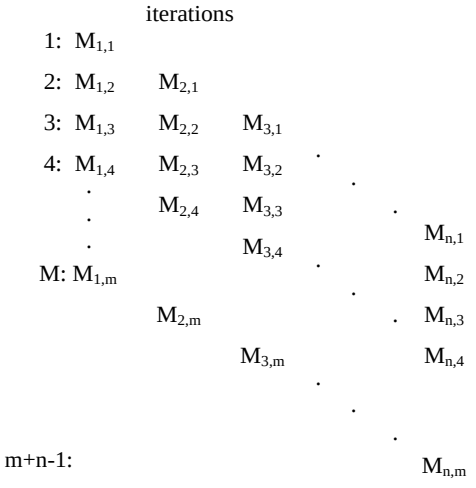

Fig. 1 Unrolled loop

$$\begin{array}{cccccc}
M_{1,1} & M_{2,1} & M_{3,1} & \cdots & M_{n,1} \\
M_{1,2} & M_{2,2} & M_{3,2} & \cdots & M_{n,2} \\
M_{1,3} & M_{2,3} & M_{3,3} & & M_{n,3} \\
M_{1,4} & M_{2,4} & M_{3,4} & & M_{n,4} \\
\vdots & \vdots & & & \vdots \\
M_{1,m} & M_{2,m} & M_{3,m} & \cdots & M_{n,m}
\end{array}$$

Fig. 2 Score matrix

4.3. Parallelization of optimized Needleman-Wunsch algorithm with OpenMP

There are four task schedulings in OpenMP, static scheduling, dynamic scheduling, guided scheduling and runtime scheduling. Static scheduling assigns the same chunk size to threads as a poll way statically, we can adopt it to reduce overhead for tasks of load balancing. Dynamic scheduling is that thread can get another chunk as long as it implements its task, it has a large overhead. Compiler can optimize static scheduling, however, there is no optimized function for dynamic scheduling. Guided scheduling is a mixed strategy, each thread obtains a larger chunk in the first, and then reduces the chunk size gradually in the succedent run until reducing to the limit chunk size, and chunk size is calculated dynamically as needed. Runtime Scheduling based on the value of environment variable OMP_SCHEDULE to decide selection of task scheduling. This paper uses static scheduling and the main code is as follows.

```
for(i=1; i<=n+m-1; i++)
{
  if(i<=m)
  {
   #pragma omp parallel for private(j) schedule (static,1)
   for(j=1; j<i+1; j++)
        M_{i+1-j,j} = Max{M_{i-j,j} +S(s_{i+1-j}, _), M_{i-j,j-1} + S(s_{i+1-j}, tj),
                  M_{i+1-j,j-1}+S(_, t_j)};
   }
   else
   {
    #pragma omp parallel for private(j) schedule (static,1)
    for(j=m; j>=i-n+1; j--)
        M_{i+1-j,j} = Max{M_{i-j,j} +S(s_{i+1-j}, _), M_{i-j,j-1} + S(s_{i+1-j}, t_j), M_{i+1-j,j-1}+S(_, t_j)};
    }
}
```

5. Conclusion

In this paper, we study the Needleman-Wunsch global alignment algorithm in bioinformatics, and optimize it with software pipelining technique and OpenMP programming so that it can be processed in parallel. The optimized Needleman-Wunsch algorithm obtains a high performance. It is perfect for processing alignment of long sequences, if the sequence is fairly short, the performance improvement of program will be not obvious, this is because OpenMP programming uses fork and join fashion when it runs, so parallel processing will bring corresponding overhead of thread creation and destruction, however, in general, the performance of Needleman-Wunsch algorithm gets a great improvement.

References

[1] Zhongneng Xu, Bioinformatics, Tsinghua University Press, Sep 2008, pp. 134-164, (in Chinese).
[2] David W Mount, Bioinformatics:sequence and genome analysis, USA: Cold Spring Harbor Laboratory Press, 2002.
[3] T K Attwood, D J Parry—Smith, Introdution to Bioinformatics, Prentice Hall, 1st ed., Mar 1999.
[4] S Needleman, C Wunsch, “Ageneral method applicable to the search for similarities in theamino acid sequences oftwo Proteins, ”Journal of Molecular Biology, l 970, 48, pp.443-453.
[5] T Smith, M Waterman, “Identification of common molecular sequence,” Journal of Molecular Biology, 1981, 147, pp.195-197.

[6] Maoyi Li, "Bioinformatics problem of sequence alignment," pp.8-25, unpublished, (in Chinese).
[7] Fuxiang Zhang, Jinling Zhou, "The Parallelization Research and the application of the Sequence Compares to Algorithm," vol.8. No.4. Journal of Weifang University, Jul 2008, (in Chinese).
[8] Min Lin, "Sequence alignment algorithm of protein," No.2. Journal of Fujian Computer, 2010, (in Chinese).
[9] Vicki H. Allan, Reese B. Jones, Randall M. Lee, Stephen J. Allan, "Software Pipelining," vol.27. No.4. Journal of ACM Computing Surveys, Sep 1995, pp.367-432.