Parallel Design Pattern for Computational Biology and Scientific Computing Applications

Weiguo Liu and Bertil Schmidt
School of Computer Engineering, Nanyang Technological University, Singapore 639798
liuweiguo@pmail.ntu.edu.sg, asbschmidt@ntu.edu.sg

Abstract

Dynamic programming is an important algorithm design technique in computational biology and scientific computing. Typical applications using this technique are very compute-intensive and suffer from long runtimes on sequential architectures. Parallel program design patterns provide a new tool to semi-automatically generate parallel programs. In this paper we present a new parallel pattern called the "block-cyclic based wavefront" to parallelize typical dynamic programming algorithms in computational biology and scientific computing. We show how this technique leads to significant run-time savings on PC clusters.

1. Introduction

Parallel computers promise to solve many computationally intensive problems much faster than their sequential counterparts. However, because a parallel program is more complex than an equivalent sequential program, to realize this increase in speed some challenges must be overcome first and this daunting task usually falls on a small number of experts [1]. Parallel program design patterns are a promising approach to offer users a convenient way to semi-automatically generate parallel programs. Parallel design patterns are based on sequential program design patterns used in object-oriented languages [2]. The main idea is to separate the communication structure of a parallel program from the sequential application code.

There are many kinds of parallel program design patterns in practice, such as Geometric Decomposition, Divide-and-Conquer, Embarrassingly Parallel and Pipeline Processing [3]. Another important design pattern is the wavefront. It can be used to develop programs for dynamic programming based algorithms in different application domains such as computational biology and scientific computing.

In this paper, we introduce a new parallel pattern called the *block-cyclic based wavefront*. We demonstrate how this pattern can be efficiently applied on distributed memory architectures for several applications amenable to wavefront computations.

2. The Block-Cyclic Based Wavefront Design Pattern

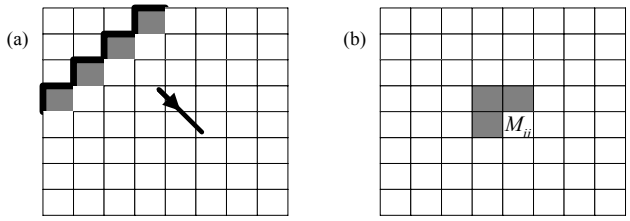

Figure 1. Example of a wavefront computation: (a) shift direction, (b) dependency relationship.

Dynamic programming applications usually exhibit the character of the wavefront computation, that is, each element computes a value that depends on the computation of a set of previous elements. An example is shown in Figure 1. Figure 1b displays the dependency relationship: each matrix element (i, j) is computed from the matrix cells $(i-1, j)$, $(i, j-1)$, $(i-1, j-1)$. The wavefront moves along anti-diagonals as depicted in Figure 1a, that is, the shift direction is from northwest to southeast. Depending on the dependency relationship different wavefront shift directions are possible.

In order to parallelize the wavefront computation on coarse-grained architectures like PC clusters it is more convenient to assign an equal number of adjacent columns to each processor as shown in Figure 2a. In order to reduce communication time further, matrix cells can be grouped into blocks. Processor i then computes all the cells within a block after receiving the required data from processor $i-1$. Figure 2b shows an example of the computation for 4 processors, 8 columns and a block size of 2×2, the numbers 1-7 represent consecutive phases in which the cells are computed.

We call this method *column-based*. It works efficiently for wavefront computations with an even workload across matrix cells, i.e. each matrix cell is computed from the same number of other matrix cells. We call such a wavefront computation *regular*. However, there are many

examples of irregular wavefront computation problems. For these problems the column-based method does not work well, since it leads to an uneven workload.

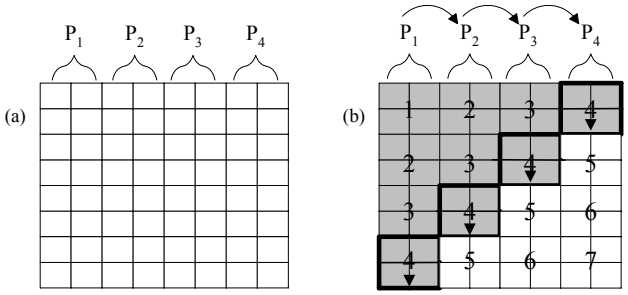

Figure 2. (a) Column-based division of an 8×8 matrix using 4 processors; (b) Wavefront computation for 4 processors, 8 columns and a 2×2 block size. The complete 8×8 matrix can then be computed in 7 iteration steps.

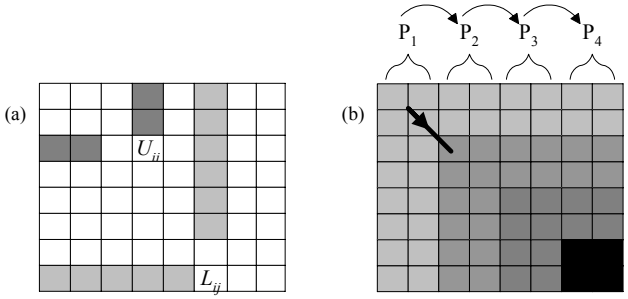

Figure 3. (a) Example of an irregular dependency pattern; (b) Distribution of load computation density.

Figure 3a shows an example of an irregular dependency pattern: matrix element (*i*, *j*) is computed from all matrix cells in row *i* and column *j* from index 1 up to $min(i-1, j-1)$. The load to compute one element in the matrix will then increase along the shift direction of the wavefront. We call this the *load computation density*. Figure 3b shows the change of load computation density along the shift direction by using increasingly blacking shades. We can see that the load computation density at the bottom right-hand corner is much higher than that at the top left-hand corner. The column-based matrix decomposition method as shown in Figure 3b will therefore lead to a poor performance, since the workload on processor P_i is higher than on the processor P_{i-1}. Because the data decomposition largely determines the performance and scalability of a parallel program, a great deal of research has aimed at studying different data decompositions. As a result the block-cyclic based distribution has been suggested as a possible general-purpose basic decomposition for parallel programs because of its scalability, load balancing and communication properties.

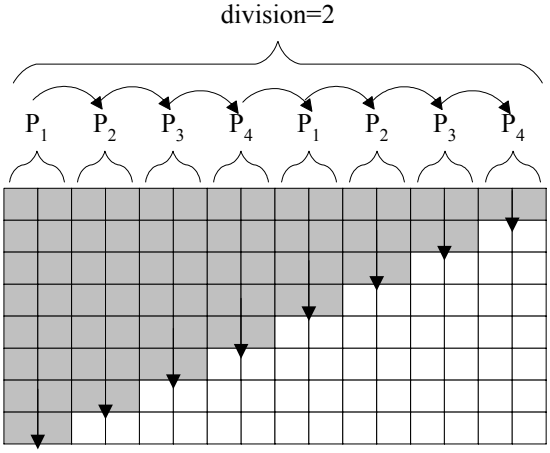

Figure 4. Block-Cyclic based way for distributed memory parallel computers

Using a block-cyclic based distribution of columns can significantly reduce this uneven workload. The concept is illustrated in Figure 4. The parameter *division* is used to implement a cyclic distribution of columns to processors. Increasing the number of divisions leads to a better load balancing. Of course, increasing the number of divisions also increases the communication overhead. Thus, the choice of the *division* parameter is a trade-off between load balancing and communication time.

We call the pattern using this method the *block-cyclic based wavefront design pattern*. In the next section we will study the implementation of this pattern and evaluate its performance.

3. Implementation and Application

A design pattern does not represent a single solution to a given problem, but rather embodies a family of potential solutions. To incorporate the idea of patterns as families of solutions, we use the template technique to construct an intermediate abstract form of a parallel pattern. This template represents the basic structure of the pattern and includes application-dependent frameworks of the basic structure for a set of applications. The framework code defines virtual functions for the sequential application code as well as the corresponding parallel structure. The template is written in terms of hook methods that are called when application–specific operations are needed. By inheriting the basic template class, instantiating the template parameters and overriding the virtual functions, new parallel applications can be implemented conveniently. Using this style of programming, the design

and implementation of the template can be used to write many different programs, reducing the effort needed to build parallel applications.

In this section, we describe five parallel applications using this pattern. We especially concentrate on the performance comparison between the column-based method and the block-cyclic based method. The programs have been implemented using MPI and C++. They are running on a distributed memory cluster which comprises sixteen Intel Pentium IV Xeon 2.6GHz processors, 512MB RAM each. All the processors are connected with each other by a Gbit/sec Myrinet switch.

3.1 Smith-Waterman Algorithm

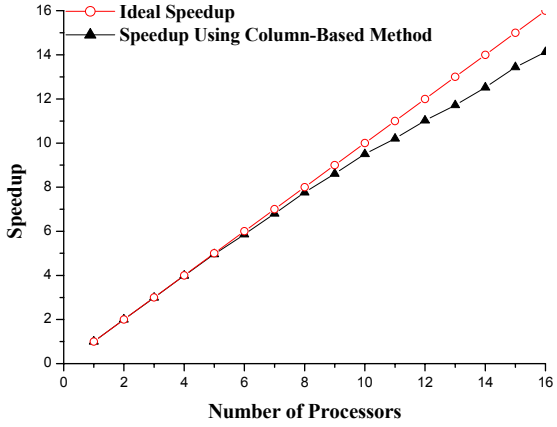

Figure 5. Speedup graph for Smith-Waterman algorithm

The wavefront computation of Smith-Waterman algorithm is shown in Figure 1. From the figure we can find that the computation is regular. So, the column-based method works well in this situation. The performance of the parallel application for two sequences of length 100000 is given in Figure 5. The linear space method is used to reduce the memory space complexity.

3.2 Skyline Matrix Problem

The wavefront computation of skyline matrix problem is shown in Figure 3. We can see that it is an irregular matrix and we use the block-cyclic based method here. The program was run on a 4000 × 4000 matrix of random values. We compared the performance between column-based method and block-cyclic based method. The results are shown in Figure 6. We choose the best performance results using block-cyclic based method when the division is set from 2 to 100.

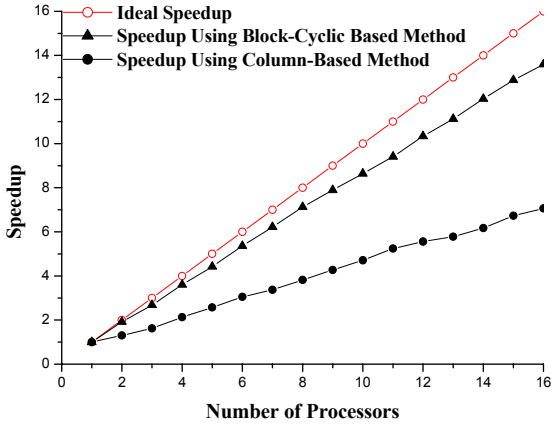

Figure 6. Speedup graph for skyline matrix problem

3.3 Nussinov and Matrix Chain Ordering Algorithm

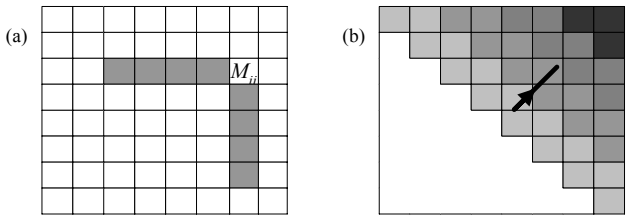

Figure 7. The wavefront computation of Nussinov algorithm and MCOP: (a) dependency relationship; (b) distribution of load computation density

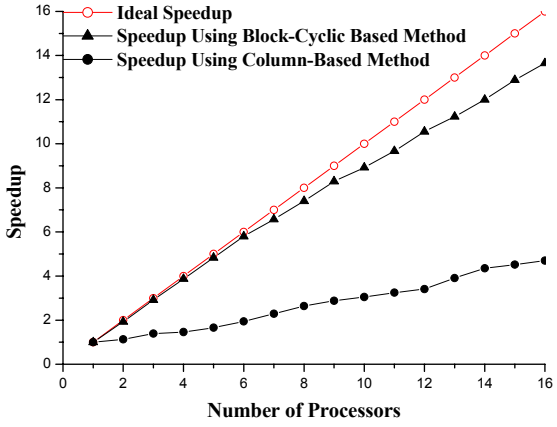

Figure 8. Speedup graph for Nussinov algorithm

Figure 7. shows the wavefront computation for Nussinov and matrix chain ordering algorithm. Obviously, they are irregular computation.

The two programs were run on two 4000 × 4000 matrices of random values respectively. The results for Nussinov algorithm are shown in Figure 8. The speedups for MCOP are very similar to the ones shown in Figure 8. We choose the best performance results using block-cyclic based method when the division is set from 2 to 100. We can see that the block-cyclic based method is significantly faster than the column-based method.

3.4 Arbitrary-Order Viterbi Algorithm

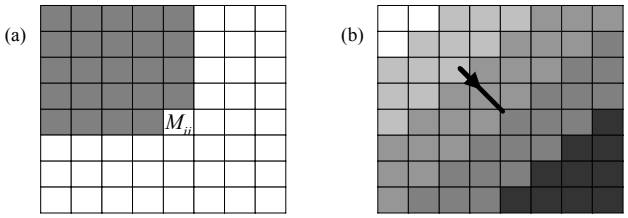

Figure 9. The wavefront computation of arbitrary-order Viterbi algorithm: (a) dependency relationship; (b) distribution of load computation density

We have designed a new kind of Hidden of Markov Model. In this HMM, each state will depend on all the previous states. We call the corresponding modified Viterbi algorithm arbitrary-order Viterbi algorithm. The dependency relationship and the distribution of load computation density for the arbitrary-order Viterbi algorithm are shown in Figure 9.

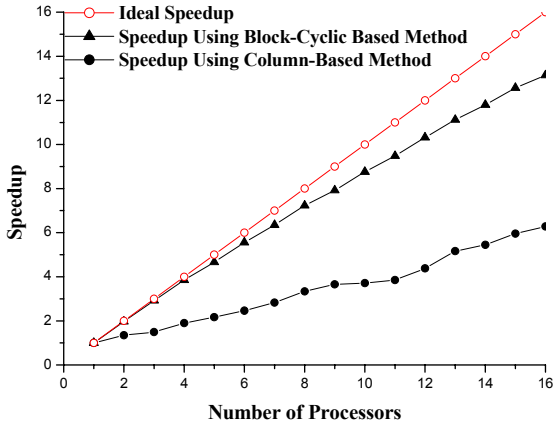

Figure 10. Speedup graph for arbitrary-order Viterbi algorithm

The program was run on a 700 × 700 matrix of random values. The results are shown in Figure 10. We choose the best performance results using the block-cyclic based method when the division is set from 2 to 20.

4. Conclusions and Future Work

In this paper, we have demonstrated how the concept of parallel program design pattern can be efficiently applied to dynamic programming algorithms on distributed memory architectures. By introducing a new pattern called the *block-cyclic based wavefront*, we can achieve semi-automatic generation of parallel programs. The measurements show that the block-cyclic based wavefront pattern can generate parallel programs with substantial performance gains for irregular dynamic programming applications. Our future work in parallel program design patterns will include identifying more applications that benefit from this technique. The results of this study will influence our design of a new API, which will extend the functionality of gMP (the *gRNA message passing interface* – introduced in [5]) to these patterns and thus allow easy parallelization of many compute-intensive genomics applications.

References

[1] J. Anvik, S. MacDonald, D. Szafron, J. Schaeffer, S. Bromling and K. Tan, "Generating Parallel Programs from the Wavefront Design Pattern", *Proceedings of the 7th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS'02)*, Fort Lauderdale, Florida, Apr. 2002.

[2] S. Bromling, S. MacDonald, J. Anvik, J. Schaeffer, D. Szafron, K. Tan, "Pattern-based Parallel Programing", *Proceedings of the 2002 International Conference on Parallel Processing* (*ICPP2002*), Vancouver, British Columbia, August 2002.

[3] Berna L. Massingill, Timothy G. Mattson, and Beverly A. Sanders, "More Patterns for Parallel Application Programs". *Proceedings of the Eighth Pattern Languages of Programming Workshop,* 2001.

[4] L. Rabiner, "A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", *Proceedings of the IEEE,* Vol. 77, No. 2, pp. 257-286, Feb 1989.

[5] B. Schmidt, F. Lin, A. Laud, Y. Santoso, "Parallel Detection of Regulatory Elements with gMP", *Proc. of IPDPS'03*, Nice, France, 2003.

[6] B. Schmidt, H. Schroder, M. Schimmler, "A hybrid architecture for bioinformatics", *Future Generation Computer System*, 18, 855-862, 2002.

[7] T.F. Smith, M.S. Waterman: "Identification of common subsequences", *Journal of Molecular Biology*, pp. 195-197, 1981.

[8] Gregory V. Wilson. "Assessing the Usability of Parallel Programming Systems: The Cowichan Problems", *Proceedings of the IFIP WG10.3 Working Conference on Programming Environments for Massively Parallel Distributed Systems*, pp. 183-193, 1994.