2003 IEEE International Conference on CLUSTER COMPUTING
HONG KONG

Parallel Design Pattern for Computational Biology and Scientific Computing

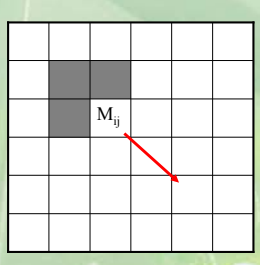
Weiguo Liu and Bertil Schmidt

1. Abstract

Dynamic programming is an important algorithm design technique in computational biology and scientific computing. Typical applications using this technique are very computing-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.

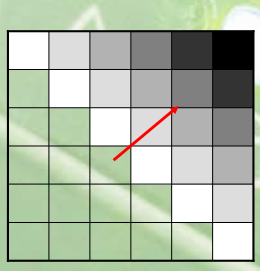
2. Dynamic Programming Algorithm : Regular and Irregular Wavefront Computation

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(Ai, Bj), \\ M(i-1,j)-d, \\ M(i,j-1)-d, \\ 0; \end{cases}$$

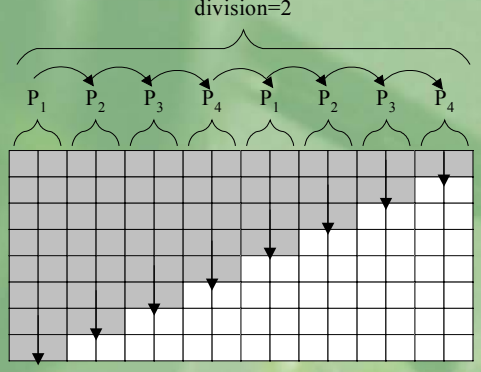


$$M(i,j) = \max \begin{cases} M(i+1,j-1) + \delta(i,j), \\ M(i,j-1), \\ M(i+1,j), \\ \max_{i<k<j}[M(i,k)+M(k+1,j)]; \end{cases}$$

where $i < j$



3. Block-Cyclic Based Partition for Distributed Memory Parallel Computers

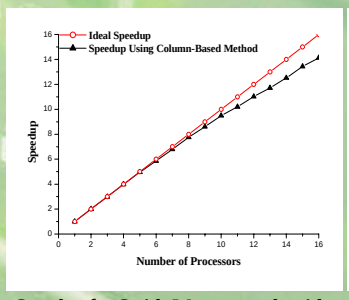


- The parameter *division* is used to implement a cyclic distribution of columns to processors.

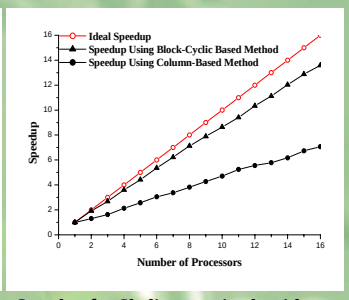
4. Implementation

```
template <class datatype>
class Wavefront{
public:
/* The constructor method uses the user-specified parameters to initialize the framework */
        Wavefront (int my_rank, int  processornumber, int matrixwidth, int matrixheight,
                   int division, int blocksize);
        void launch (){
                preprocess ();
                commonpreprocess();
                while (the division loop is not completed){
                        processor 1{
                                prepare1 ();
                                commonprepare1();
                                operate1 ();
                        }
                        processor 2 to P{
                                prepareP ();
                                commonprepareP();
                                operateP();
                        }
                }
                postprocess ();
        }
/* These three methods are used to implement the block-cyclic distribution scheme */
        void commonpreprocess(){};
        void commonprepare1(){};
        void commonprepareP(){};
/* These four methods allow users to specify customized behavior for a given application */
        virtual void preprocess (){};
        virtual void postprocess (){};
        virtual void prepare1 (){};
        virtual void prepareP (){};
/* These two methods are used to implement the compute-communicate procedure */
        virtual void operate1 (){};
        virtual void operateP (){};
/* This method is provided by the user to compute the node (i, j), it is called by the
   methods operate1() and operateP() */
        virtual void application (){};
};
```

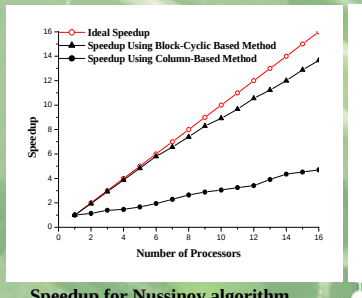
5. Results



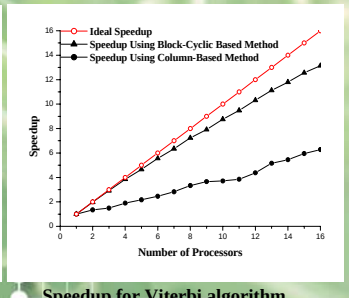
Speedup for Smith-Waterman algorithm



Speedup for Skyline matrix algorithm



Speedup for Nussinov algorithm



Speedup for Viterbi algorithm

6. Conclusions

- Semi-automatic generation of parallel programs
- Substantial performance gains for irregular dynamic programming applications