FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

› TECHNIK UND MANAGEMENT

DIPLOM-STUDIUM:
› BIOENGINEERING

Parallelization of RNA Folding Algorithms for Multi Core Processors

[1]Daniel Hooker, [1]Michael T. Wolfinger, [2]Ivo L. Hofacker
[1]FH Campus Wien
[2]Institute for Theoretical Chemistry and Structural Biology, University of Vienna, Austria
hook@tbi.univie.ac.at

Abstract

The *Vienna RNA Package* is a state of the art tool to predict and compare RNA secondary structures. The most used function is the secondary structure prediction which uses the minimum free energy algorithm[1].

Since placing transistors on a single chip and raising the clock speed have inherent limitations computer architects have focused on ideas of the 1980s: multiple processors that share memory are combined to a single computer or even a chip.[2]

To gain an overview of the possibilities of parallelization the simplest RNA folding algorithm was used for pre-testing.

Conclusion

Both possible matrix fill strategies were implemented in two small C programmes and were made parallel with OpenMP. The diagonal fill method is straight forward to parallelize, because of data independence. The row by row method needs extra synchronisation, often referred to as pipelined execution. The diagonal fill method is faster and has a lower overhead and therefore it should be used for multi core processors. On the other hand the row by row strategy has a better cache utilization and should therefore be used for single core processors.

Maximum Matching

A single stranded RNA will spontaneously form secondary structure elements to reduce the free energy before forming the tertiary structure (figure 1). These structures are formed by hydrogen bounds between bases.

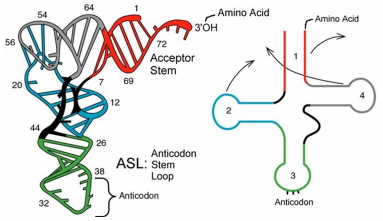

Figure 1: **RNA Structure** – This figure shows how the tertiary structure of a t-RNA is formed from secondary structure elements.

According to the "Nussinov Algorithm"[3] the structure with a maximum number of base pairs is the most stable one (figure 2).

$$X_{i,j} := \max \begin{cases} X_{i+1,j} & (i) \\ X_{i,j-1} & (ii) \\ X_{i+1,j-1} + \delta(r_i, r_j) & (iii) \\ \max_{k,i<k<j}\{X_{i,k} + X_{k+1,j}\} & (iv) \end{cases}$$

Figure 2: **The Nussinov Algorithm** – In case (i) and (ii) we have the probability that either base r_i or r_j is unpaired. Case (iii) includes the probability that r_i and r_j are paired. The last case (iv) includes the probability that we have two substructures.

The maximum number of base pairs for the whole sequence is on the upper right of the matrix. Thus there are two different matrix fill strategies (figure 3) and two different ways of parallelization.

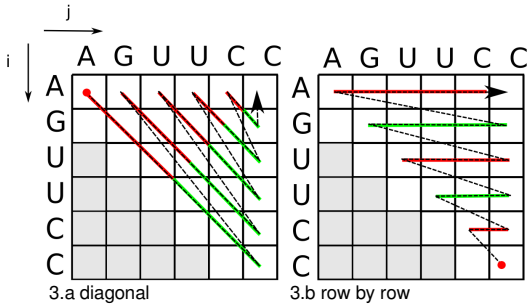

Figure 3: **Matrix Fill Strategies** – The first matrix shows the diagonal strategy and the second matrix shows the row by row strategy. The red colour stands for thread 0 & green signifies thread 1.

Results

The following results were computed with two Intel(R) Xeon(R) CPU E5450 3.00GHz quad core processors and 32 GB RAM main memory.

The threads in figure 4.b have to wait for each other, therefore the speedup of the row by row fill strategy is much lower than of the diagonal one. Also the length of the sequence needs to be longer before a speedup can be achieved.

Generally the longer the sequence the higher is the benefit of parallelization.

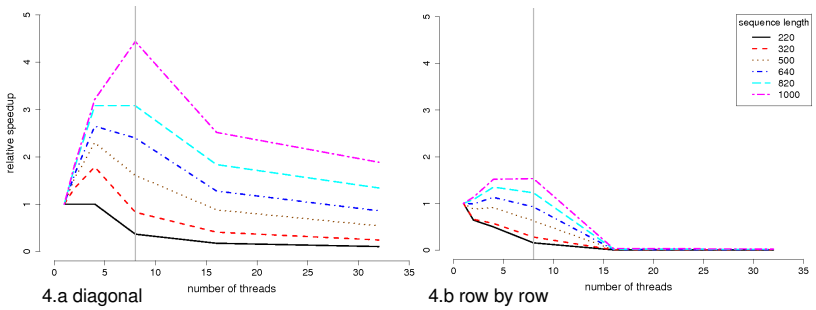

Figure 4: **Relative Speedup** – The relative speedup shows how much a parallel programme runs faster on more processors than it does on just one. The gray line at 8 threads refers to the physical number of processors.

Figure 5 shows that parallelism is not for free. Additional overheads are needed for e.g. creating, starting and stopping threads or waiting for others to finish.

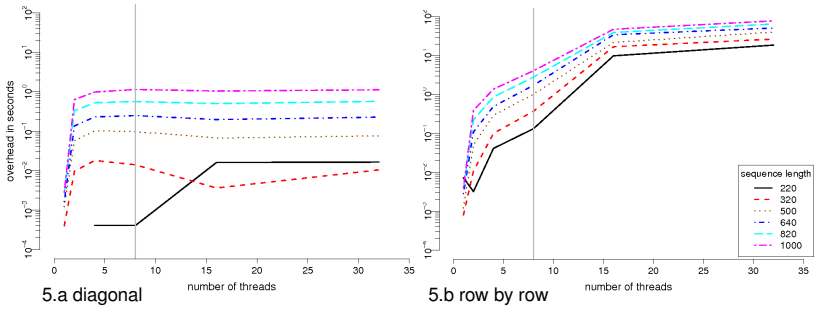

Figure 5: **Overhead** – The overhead is any indirect computation time, I/O or any other resource which is expanded to achieve the goal of the algorithm.

References

[1] Zuker, M. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information.(1981)
[2] Chapman, B. *et al.* Using OpenMP: portable shared memory parallel programming (2007)
[3] Nussinov, R. *et al.* Algorithms for loop matching (1978)