Communication Papers of the 18th Conference on Computer Science and Intelligence Systems pp. 37–41
DOI: 10.15439/2023F7645
ISSN 2300-5963 ACSIS, Vol. 37

Polyhedral Tiling Strategies for the Zuker Algorithm Optimization

Wlodzimierz Bielecki, Marek Palkowski, Piotr Blaszynski, Maciej Poliwoda
West Pomeranian University of Technology in Szczecin
ul. Zolnierska 49, 71-210 Szczecin, Poland
Email: mpalkowski@zut.edu.pl

***Abstract*—In this paper, we focus on optimizing the code for computing the Zuker RNA folding algorithm. This bioinformatics task belongs to the class of non-serial polyadic dynamic programming, which involves non-uniform program loop dependencies. However, its dependence pattern can be represented using affine formulas, allowing us to automatically employ tiling strategies based on the polyhedral method. We use three source-to-source compilers Pluto, Traco, and Dapt based on affine transformations, transitive closure of dependence relation graph and space-time tiling, respectively, to automatically generate cache-efficient codes. We evaluate the speed-up and scalability of optimized codes and check their performance employing applying two multi-core machines. We also discuss related approaches and outline future work in the conclusion of the paper.**

I. Introduction

RNA secondary structure prediction is a fundamental and noticeably time-consuming problem in the biological computing. For a given RNA sequence, the secondary non-crossing RNA structure is predicted such that the total amount of free energy is minimized. Smith and Waterman [1], and Nussinov et al. [2] first defined a dynamic programming algorithm for RNA folding. Instead of using the free energy, the algorithms of [1], [2] aim to maximize the number of complementary base pairs.

Zuker et al. [3] first proposed a complex dynamic programming algorithm to predict the most stable secondary structure for a single RNA sequence by computing its minimal free energy. It uses a "nearest neighbor" model. The algorithm estimates of thermodynamic parameters for neighboring interactions. The main idea is that the loop entropies are used to score all possible structures and the secondary structure of an RNA sequence consists of four fundamental independent substructures: stack, hairpin, internal loop, and multi-branched loop. The energy of a secondary structure is assumed to be the sum of the substructure energies.

Zuker's algorithm consists of two steps. The first step, which is the most time-consuming, involves calculating the minimal free energy of the input RNA sequence using recurrence relations as outlined in the provided formulas. The second step involves performing a trace-back to recover the secondary structure with the base pairs. While the second step is not a computationally demanding task, optimization of the energy matrix calculation in the first step is crucial for improving the overall performance of the algorithm [4].

Zuker defines two energy matrices, $W(i,j)$ and $V(i,j)$, with $\mathcal{O}(n^2)$ pairs (i,j) satisfying the constraints $1 \le i \le N$ and $i \le j \le N$, where N is the length of a sequence. $W(i,j)$ represents the total free energy of a sub-sequence defined by indices i and j, while $V(i,j)$ represents the total free energy of a sub-sequence starting at index i and ending at index j if i and j form a pair, otherwise $V(i,j) = \infty$.

The main recursion of Zuker's algorithm for all *i, j* with $1 \le i < j \le N$, where *N* is the length of a sequence, is the following.

$$W(i,j) = \begin{cases} W(i+1,j) & (1) \\ W(i,j-1) & (2) \\ V(i,j) & (3) \\ \min\limits_{i<k<j}\{W(i,k)+W(k+1,j)\} & (4) \end{cases}$$

Below, we present the computation of V.

$$V(i,j) = \begin{cases} eH(i,j) & (5) \\ V(i+1,j-1)+eS(i,j) & (6) \\ \min\limits_{\substack{i\le i'\le j'\le j \\ 2<i'-i+j-j'<d}}\{V(i',j')+eL(i,j,i',j')\} & (7) \\ \min\limits_{i<k<j-1}\{W(i+1,k)+W(k+1,j-1)\} & (8) \end{cases}$$

eH (hairpin loop), *eS* (stacking) and *eL* (internal loop) are the structure elements of energy contributions in the Zuker algorithm.

The computation of Equations 1, 2, 3, 5, 6 takes $\mathcal{O}(n^2)$ steps. Equations 4 and 8 requires $\mathcal{O}(n^3)$ steps. The time complexity of a direct implementation of this algorithm is $\mathcal{O}(n^4)$ because we need $\mathcal{O}(n^4)$ operations to compute Equation 7. This formulation as a computational kernel involves float arrays and operations.

The computation domain and dependencies for Zuker's recurrence cell (i,j) are more complex than those of Nussinov's recurrence. Equations 3, 4, and 8 generate long-range (non-local) dependencies for cell (i,j), while the other equations have short-range (local) dependencies. The computation of the element *V(i',j')* in Equation 3 spans a triangular area of several dozens to hundreds of cells.

Listing 1 shows the affine loop nest for finding the minimums of the V and W energy matrices.

Thematic track: Computer Aspects of Numerical Algorithms

Listing 1. Zuker's recurrence loop nest

```
for (i = N-1; i >= 0; i--){
   for (j = i+1; j < N; j++) {
      for (k = i+1; k < j; k++){
         for(m=k+1; m <j; m++){
            if(k-i + j - m > 2 && k-i + j - m < 30)
               V[i][j] = MIN(V[k][m] + EL(i,j,k,m), V[i][j]);                    // Eq. 3
         }
         W[i][j] = MIN ( MIN(W[i][k], W[k+1][j]), W[i][j]);                     // Eq. 8
         if(k < j-1)
            V[i][j] = MIN(W[i+1][k] + W[k+1][j-1], V[i][j]);                    // Eq. 4
      }
      V[i][j] = MIN( MIN (V[i+1][j-1] + ES(i,j), EH(i,j), V[i][j]);             // Eq. 1,2
      W[i][j] = MIN( MIN ( MIN ( W[i+1][j], W[i][j-1]), V[i][j]), W[i][j]);     // Eq. 5,6,7
   }
}
```

In this paper, we focus on studying the performance of tiled Zuker loop nests codes generated by chosen automatic optimizers based on the polyhedral model.

Loop tiling, also known as loop blocking or loop partitioning, is a program transformation technique used in compiler optimization to enhance cache utilization and improve the performance of loop-based computations [5]. It involves dividing a loop into smaller, tile-sized sub-loops or blocks that fit into the cache effectively. The main idea behind loop tiling is to exploit spatial locality, which refers to accessing data elements that are close together in memory. By dividing a loop into smaller tiles, the loop iterations within each tile can reuse data elements, reducing cache misses and improving memory access patterns.

The polyhedral model represents loop nests as polyhedra with affine loop bounds and schedules. It enables advanced loop transformations and analysis of data dependences. By leveraging this model, compilers can automatically optimize loops, improve performance (especially locality employing loop tiling), and exploit parallelism [6].

II. Related work

The Zuker kernel, as well as the Nussinov RNA folding, involves mathematical operations over affine control loops whose iteration space can be represented by the polyhedral model [7]. However, the Zuker RNA folding acceleration is still a challenging task for optimizing compilers because that code is within non-serial polyadic dynamic programming (NPDP), which is a particular family of dynamic programming with non-uniform data dependences, and it, as mentioned above, is more difficult to be optimized [8]. In addition, the loop structure of Zuker's algorithm is definitely more complicated for automatic tiling strategies than that of Nussinov's algorithm, i.e. the loops are quadruple nested with more instructions which also implies a larger number of data dependencies (including non-uniform ones).

There are other RNA folding numerical approaches which can be presented within the polyhedral model. To enhance the accuracy of structure prediction for a given RNA sequence, the algorithm devised by Zhi J. Lu and colleagues in 2009 incorporates the concept of Maximum Expected Accuracy (MEA), utilizing base pair and unpaired probabilities [9]. This method employs a Nussinov-like recursion, drawing upon the probabilities obtained through John S. McCaskill's algorithm [10]. Numerical sources and aspects of the Nussinov, Zuker, and MEA algorithms can be found in the NPDP Benchmark Suite [11]. It is a collection of NPDP tasks which cannot be effectively optimized using commonly employed tiling strategies, such as diamond tiling [12], [13].

An interesting cache-efficient manual solution for Nussinov's RNA folding algorithm was proposed by Li and colleagues in [14]. Using lower and unused part of Nussinov's, they changed column reading to more efficient row reading. Diagonal scanning exposes parallelism in the output code. The method is known also as *Transpose* technique. In our previous paper [15], we adopted the *Transpose* to optimize Zuker's code. In equations 4 and 8, there are not cache-efficient column reading of the W array, $W[k+1][j]$ and $W[k+1][j-1]$, respectively. The transpose method changes these array accesses to the row reading and adds the following statement $W[col][row] = W[row][col]$ to make a transposed copy of the cells in the lower-left triangle.

Zhao et al. [16] improved the *Transpose* method and performed the experimental study of the energy-efficient codes for Zuker's algorithm. The approach based on the LRU cache model requires about half as much memory as does Li's Transpose. However, the authors did not present parallel codes for the *ByBox* strategy and any automatic optimization was not proposed.

Pluto is a widely-used, advanced tool for optimizing C/C++ programs through the use of polyhedral code generation. It transforms the source code into parallelized, coarse-grained code that is optimized for data locality, primarily using the affine transformation framework (ATF). This state-of-the-art source-to-source compiler is highly regarded in the field for its effectiveness in improving the performance of parallel software. Unfortunately, Pluto fails to achieve maximal code locality and performance for the well-known NPDP problems [8]. It is unable to tile the innermost loop of Nussinov's RNA folding, which is a key to cache locality optimization [7], [17].

It cannot produce parallel code for the McCaskill probabilistic RNA folding kernel [18]. For the Zuker code presented in Listing 1, the approach is unable to tile the 3rd loop nest.

Authors of Pluto, Bondhugula and et al. [7] presented dynamic tiling for the Zuker's optimal RNA secondary structure prediction [7]. 3-d iterative tiling for dynamic scheduling is calculated using reduction chains. Operations along each chain can be reordered to eliminate cycles in an inter-tile dependence graph. Their approach involves dynamic scheduling of tiles, rather than the generation of a static schedule.

Wonnacott et al. introduced 3-d tiling of "mostly-tileable" loop nests of RNA secondary-structure prediction codes in paper [17]. This approach extracts non-problematic statement instances in the loop nest iteration space, i.e., those that can be safely tiled by means of well-known techniques. The reminding statement instances should be run serially to preserve all the dependences available in the loop nest. Unfortunately, the approach is limited to serial codes only. The idea is presented only for simpler Nussinov's RNA folding which maximizes the number of complementary base pairs.

In past, we developed the tiling technique [8] aimed to transform (corrects) original rectangular tiles into target ones, which are valid under lexicographic order. Tile correction is performed by means of the transitive closure of loop dependence graphs. Loop skewing is used to parallelize code. We achieved a higher speed-up of generated tiled code in comparison with that produced with state-of-the-art source-to-source optimizing compilers. However, the transitive closure is a NP-difficult problem and is not always computable in general case.

Tiling correction [8] and Four-Russian RNA Folding [19] were deeply studied by Tchendji and et al. and they proposed a parallel tiled and sparsified four-Russians algorithm for Nussisov's RNA Folding [20]. They claim that this approach computation is more cache-friendly because it applies the blocks of Four-Russians mustered into parallelogram-shaped tiles. The experimental study for CPUs and massively GPUs architectures shows the out-performance in comparison to the results of [8] and [19]. Although, the authors considered manually the Nussinov loop nest only, they promised to study other NPDP problems in future.

The space-time loop tiling approach presented in paper [21] generates target tiles using the intersection operation to sets representing sub-spaces and time slices is applied. Each time partition comprises independent iterations, which can be executed in parallel while time partitions should be enumerated in lexicographical order. The presented approach is a continuation of the work on space-time tiling, which shows promising possibilities in developing new polyhedral optimizing compilers. The codes were generated with the Dapt compiler introduced in paper [22].

III. Experimental study

To carry out experiments, we used a machine with a processor AMD Epyc 7542, 2.35 GHz, 32 cores, 64 threads, 128MB Cache, and machine with a processor Intel Xeon Gold 6240, 2.6GHz (3.9GHz turbo), 18 cores, 36 threads, 25MB Cache. The optimized codes were compiled by means of the GNU C++ compiler version 9.3.0 with the -O3 flag.

TABLE I
Execution times (in seconds) for AMD Epyc 7542 and 64 threads.

Size	Classic	Transpose	Pluto	TileCorr	Space-time
1000	26.90	3.31	3.26	7.88	1.80
1500	132.87	14.08	12.33	25.97	8.22
2000	415.15	42.65	30.99	58.70	22.33
2500	1013.99	100.60	69.19	118.45	53.08
3000	2093.22	202.43	137.49	217.01	109.73
3500	3871.90	370.90	245.93	356.61	201.75
4000	6589.03	626.56	407.87	578.37	342.78
4500	10544.30	998.58	644.83	874.90	550.97
5000	15686.70	1515.55	977.20	1272.33	853.73

TABLE II
Execution times (in seconds) for Intel Xeon Gold 6240 and 36 threads.

Size	Classic	Transpose	Pluto	TileCorr	Space-time
1000	27.31	4.28	2.96	6.87	2.23
1500	135.38	17.16	19.54	29.53	14.29
2000	393.88	43.56	23.87	41.75	18.55
2500	954.87	102.06	50.39	85.09	40.95
3000	1970.48	206.51	97.42	156.89	81.35
3500	3644.10	378.81	180.23	266.01	151.19
4000	6209.09	654.14	300.47	426.64	259.77
4500	9944.50	1048.81	489.30	649.73	416.07
5000	15133.74	1589.30	819.77	968.87	634.46

Tests were conducted using ten randomly generated RNA sequences with lengths ranging from 1000 to 5000. Discussion in papers [8], [14] shows that cache-efficient code performance does not change based on strings themselves, but it depends on the size of a string.

We compared the performance of tiled codes generated with the presented approaches i) *Pluto* parallel tiled code (based on affine transformations) [23], ii) tile code based on the *Space-time* technique [21] generated with Dapt, iii) tiled code based on the correction technique *TileCorr* [8] generated with Traco, iv) Li manual cache-efficient implementation of Zuker's RNA folding *Transpose* [14]. All codes are multithreaded within the OpenMP standard [24].

The tile size 16×16×1×16 for Pluto code was chosen empirically (Pluto does not tile the third loop) as the best among many sizes examined. The tile 16×16×16×16 size for tile correction technique was chosen according to paper [15]. For the space-time tiled code, we chose the same tile sizes. Our preliminary empirical testing did not yield improved tile sizes for this algorithm.

Table 1 presents execution times in seconds for ten sizes of

Fig. 1. Speed-up for AMD Epyc 7542 and 64 threads.

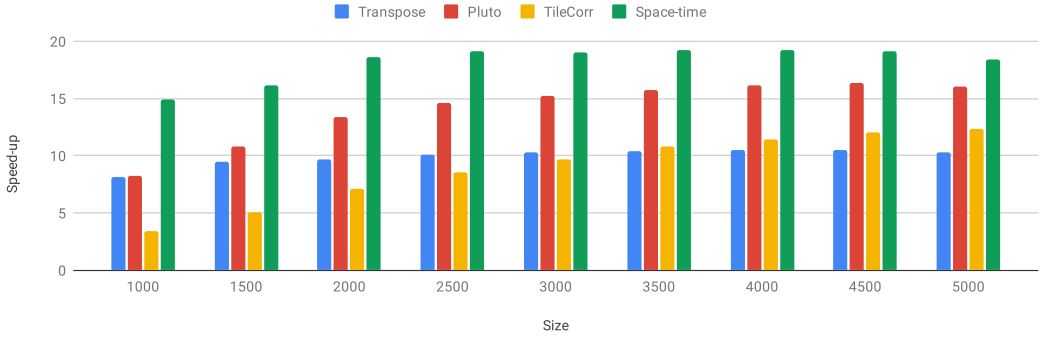
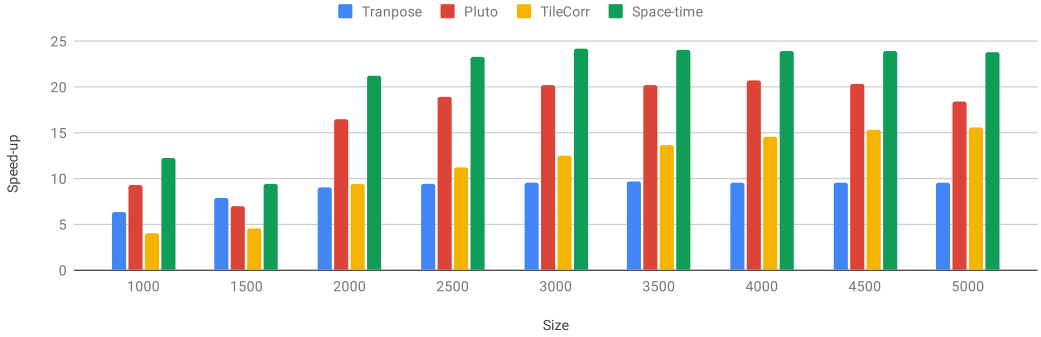


Fig. 2. Speed-up for Intel Xeon Gold 6240 and 36 threads.



RNA sequence using AMD Epyc 7542. Problem sizes from 1000 to 5000 (roughly the size of the longest human mRNA) were chosen to illustrate advantages for smaller and larger instances. Output codes are executed for 64 threads. We can observe that the presented space-time tiling approach allows for obtaining cache-efficient tiled code, which outperforms significantly the other examined implementations for each RNA strands lengths. The second most efficient code is loop tiling produced by the Pluto compiler. Figure 1 depicts the speed-up for times presented in Table I.

Table 2 presents execution times in seconds using two processors Intel Xeon E5-2695 v2 and 48 threads. The presented space-time tiling strategy outperforms strongly the other studied techniques for all RNA strands lengths. Transpose technique allows us to obtain faster code than the ATF tiled code and the tile correction code with this machine. Figure 2 depicts speed-ups for time executions in Table 2.

At the address https://github.com/markpal/zuker, all source codes used in the experimental study are available.

IV. Conclusion

Summing up, the space-time tiled code we introduced allows for improved and scalable performance on both of the multi-core processors, regardless of the number of threads or problem size. The output codes were generated automatically based on the input serial code. The space-time tiling strategy implemented within the polyhedral compiler Dapt appears to be a promising solution for optimizing NPDP tasks, and we plan to examine its use on other NPDP bioinformatics problems.

References

[1] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195 – 197, 1981.

[2] R. Nussinov *et al.*, "Algorithms for loop matchings," *SIAM Journal on Applied mathematics*, vol. 35, no. 1, pp. 68–82, 1978.

[3] M. Zuker and P. Stiegler, "Optimal computer folding of large rna sequences using thermodynamics and auxiliary information." *Nucleic Acids Research*, vol. 9, no. 1, pp. 133–148, 1981.

[4] G. Lei, Y. Dou, W. Wan, F. Xia, R. Li, M. Ma, and D. Zou, "CPU-GPU hybrid accelerating the zuker algorithm for RNA secondary structure prediction applications," *BMC Genomics*, vol. 13, no. Suppl 1, p. S14, 2012. doi: 10.1186/1471-2164-13-s1-s14

[5] J. Xue, *Loop Tiling for Parallelism*. Norwell, MA, USA: Kluwer Academic Publishers, 2000. ISBN 0-7923-7933-0

[6] S. Verdoolaege, "Integer set library - manual," Tech. Rep., 2011. [Online]. Available: www.kotnet.org/~skimo//isl/manual.pdf,

[7] R. T. Mullapudi and U. Bondhugula, "Tiling for dynamic scheduling," in *Proceedings of the 4th International Workshop on Polyhedral Compilation Techniques*, Vienna, Austria, Jan. 2014.

[8] M. Palkowski and W. Bielecki, "Parallel tiled Nussinov RNA folding loop nest generated using both dependence graph transitive closure and loop skewing," *BMC Bioinformatics*, vol. 18, no. 1, p. 290, 2017. doi: 10.1186/s12859-017-1707-8

[9] Z. J. Lu, J. W. Gloor, and D. H. Mathews, "Improved RNA secondary structure prediction by maximizing expected pair accuracy," *RNA*, vol. 15, no. 10, pp. 1805–1813, Aug. 2009. doi: 10.1261/rna.1643609. [Online]. Available: https://doi.org/10.1261/rna.1643609

[10] J. S. McCaskill, "The equilibrium partition function and base pair binding probabilities for RNA secondary structure," *Biopolymers*, vol. 29, no. 6-7, pp. 1105–1119, may 1990. doi: 10.1002/bip.360290621

[11] M. Palkowski and W. Bielecki, "NPDP benchmark suite for loop tiling effectiveness evaluation," in *Parallel Processing and Applied Mathematics*. Springer International Publishing, 2023, pp. 51–62. [Online]. Available: https://doi.org/10.1007/978-3-031-30445-3_5

[12] T. Malas, G. Hager, H. Ltaief, H. Stengel, G. Wellein, and D. Keyes, "Multicore-optimized wavefront diamond blocking for optimizing stencil updates," *SIAM Journal on Scientific Computing*, vol. 37, no. 4, pp. C439–C464, Jan. 2015. doi: 10.1137/140991133. [Online]. Available: https://doi.org/10.1137/140991133

[13] U. Bondhugula, V. Bandishti, and I. Pananilath, "Diamond tiling: Tiling techniques to maximize parallelism for stencil computations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1285–1298, May 2017. doi: 10.1109/tpds.2016.2615094

[14] J. Li, S. Ranka, and S. Sahni, "Multicore and GPU algorithms for Nussinov RNA folding," *BMC Bioinformatics*, vol. 15, no. 8, p. S1, 2014. doi: 10.1186/1471-2105-15-S8-S1. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-15-S8-S1

[15] M. Palkowski and W. Bielecki, "Parallel tiled cache and energy efficient code for zuker's RNA folding," in *Parallel Processing and Applied Mathematics*. Springer International Publishing, 2020, pp. 25–34.

[16] C. Zhao and S. Sahni, "Efficient RNA folding using zuker's method," in *2017 IEEE 7th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*. IEEE, oct 2017. doi: 10.1109/iccabs.2017.8114309

[17] D. Wonnacott, T. Jin, and A. Lake, "Automatic tiling of "mostly-tileable" loop nests," in *5th International Workshop on Polyhedral Compilation Techniques*, Amsterdam, 2015.

[18] M. Palkowski and W. Bielecki, "Parallel cache-efficient code for computing the McCaskill partition functions," vol. 18, pp. 207–210, 2019. doi: 10.15439/2019F8

[19] Y. Frid and D. Gusfield, "An improved Four-Russians method and sparsified Four-Russians algorithm for RNA folding," *Algorithms for Molecular Biology*, vol. 11, no. 1, aug 2016. doi: 10.1186/s13015-016-0081-9

[20] V. K. Tchendji, F. I. K. Youmbi, C. T. Djamegni, and J. L. Zeutouo, "A parallel tiled and sparsified Four-Russians algorithm for Nussinov's RNA folding," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pp. 1–12, 2022. doi: 10.1109/tcbb.2022.3216826

[21] M. Palkowski and W. Bielecki, "Tiling Nussinov's RNA folding loop nest with a space-time approach," *BMC Bioinformatics*, vol. 20, no. 1, apr 2019. doi: 10.1186/s12859-019-2785-6

[22] W. Bielecki, M. Palkowski, and M. Poliwoda, "Automatic code optimization for computing the McCaskill partition functions," in *Annals of Computer Science and Information Systems*. IEEE, sep 2022. doi: 10.15439/2022f4

[23] U. Bondhugula *et al.*, "A practical automatic polyhedral parallelizer and locality optimizer," *SIGPLAN Not.*, vol. 43, no. 6, pp. 101–113, Jun. 2008. [Online]. Available: http://pluto-compiler.sourceforge.net

[24] OpenMP Architecture Review Board, "OpenMP application program interface version 5.2," 2022. [Online]. Available: https://www.openmp.org/specifications/