Proceedings of the Federated Conference on Computer Science and Information Systems pp. 207–210
DOI: 10.15439/2019F8
ISSN 2300-5963 ACSIS, Vol. 18

Parallel cache-efficient code for computing the McCaskill partition functions

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

***Abstract*—We present parallel tiled optimized McCaskill's partition functions computation code. That CPU and memory intensive dynamic programming task is within computational biology. To optimize code, we use the authorial source-to-source TRACO compiler and compare obtained code performance to that generated with the state-of-the-art PluTo compiler based on the affine transformations framework (ATF). Although PLuTo generates tiled code with outstanding locality, it fails to parallelize tiled code. A TRACO tiling strategy uses the transitive closure of a dependence graph to avoid affine function calculation. The ISL scheduler is used to parallelize tiled loop nests. An experimental study carried out on a multi-core computer demonstrates considerable speed-up of generated code for the larger number of threads.**

I. Introduction

DYNAMIC programming (DP) is typically applied to optimization problems in computational biology. Code implementing such computation-intensive tasks include loop nests within the polyhedral model, which allows us to apply optimization compilers to improve code performance. However, the fact that such problems are within non-serial polyadic dynamic programming (NDPD) leads to existence of non-uniform dependences in corresponding loop nests. This limits many commonly known optimization techniques such as permutation, diamond tiling [1], or index set splitting [2] to improve cache efficiency.

One of such NDPD problems is McCaskill's algorithm, which requires computing partition functions, which used to fold an RNA secondary structure and to find the probabilities of various sub-structures. McCaskill's recurrence is quite similar to other NPDP RNA folding algorithms such as Nussinov's and Zucker's ones, which are not trivial to be optimized with automatic optimization compilers.

Today, most popular techniques to automatically generate optimal parallel code are based on affine transformations. For a given loop nest statement, an affine transformation can be presented with the following relation $[I]->[t=C*I+c]$, where I is the iteration vector of the statement; t is the discrete time of the execution of iteration I; $C*I+c$ is the affine expression. If two statement instances have the same execution time, they can be run in parallel.

To find the unknown matrix C and unknown vector c, for each loop nest statement, on the basis of dependence relations time-partition constraints are created and resolved for elements of matrix C and elements of vector c.

State-of-the-art automatic optimizing compilers, such as PLuTo [3], have provided empirical confirmation of the success of polyhedral-based code generation and optimization. PLuTo optimizing compiler is based on the affine transformation framework (ATF), which has demonstrated considerable success in generating high-performance parallel code in particular for stencils.

ATF is also used to generate tiled code. Loop tiling for improving locality groups loop statement instances into smaller blocks (tiles) allowing reuse when the block fits in local memory. In parallel tiled code, tiles are considered as indivisible macro statements. This coarsens the granularity of parallel applications that often leads to improving the performance of an application running in parallel computers with shared memory.

ATF is applied in other compilers such as Apollo and PPCG as well as commercial R-STREAM and IBM-XL. ATF has some drawbacks, papers [4], [5], [6] present its limitations for generation of parallel cache-efficient code for bioinformatics NPDP tasks. Although PLuTo generates outstanding cache-efficient code for McCaskill's algorithm, it is not able to generate any parallel code.

Wonnacott et al. introduced serial 3-D tiling of "mostly-tileable" loop nests of Nussinov's RNA secondary structure prediction in paper [5] to overcome some ATF limitations. But they did not present how to parallelize code generated with a proposed technique.

Mullapudi and Bondhugula [6] have also explored automatic techniques for tiling codes that lie outside the domain of standard tiling techniques. 3-D iterative tiling for dynamic scheduling is calculated by means of reorderable reduction chains to eliminate cycles between tiles for Nussinov's algorithm. Until now, we do not have a precise characterization of the relative domains of those techniques and it is not clear how they can be applied to parallelize McCaskill's algorithm where target arrays are not a result of reorderable functions such as minimum or maximum.

Paper [7] presents a manual implementation of parallel McCaskill's algorithm, but the approach is limited only to message passing architectures and does not consider locality improvement for modern multi-core machines.

Li et al. show how to use array transposition to enable better caching for Nussinov's algorithm [8] with replacing the array reading column order to the row order. The disadvantage of

this approach is the cost of additional memory management, which is overcome by tiling strategies [9]. In paper [10], Li's method was improved, but it allows for generating only serial code.

In this paper, we introduce an alternative approach as a combination of the tile correction algorithm [11] and the ISL scheduler [12] to parallelize tiled loop nests of the McCaskill's algorithm to overcome limitations of the mentioned techniques. This approach is implemented in the TRACO compiler [13].

TRACO does not find and use any affine function to transform the loop nest. It is based on the iteration space slicing framework [14] and applies the transitive closure of a dependence graph to carry out corrections of original rectangular tiles so that all dependences available in the original loop nest are preserved under the lexicographic order of target tiles. The inter-tile dependence graph does not contain any cycle and any technique of loop nest parallelization can be used [11] to generate parallel code. We apply the commonly known loop skewing technique and use the ISL library to implement it and generate parallel tiled code implementing McCaskill's algorithm. We observe high performance and scalability of that code executed on multi-core processors. We compare obtained code performance with that of code generated with PLuTo.

II. McCaskill's algorithm for the partition function computation

John S. McCaskill proposed an efficient dynamic programming algorithm to compute the partition function $Z = \sum_P exp(-E(P)/RT)$ over all possible nested structures P that can be formed by a given RNA sequence S with $E(P)$ = energy of structure P, R = gas constant, and T = temperature [15].

In this paper, we study a simplified version of the approach using a Nussinov-like energy scoring scheme, i.e., each base pair of a structure contributes a fixed energy term E_{bp} independent of its context. Given such an assumption, two dynamic programming tables Q and Q_{bp} are populated. The partition function for a sub-sequence from position i to position j is provided by Q_{ij}. Array Q_{bp} holds the partition function of the sub-sequences, which form a base pair or 0 if base pairing is not possible.

The following recursions are used to compute the partition functions Q and Q^{bp}.

$$Q_{i,j} = Q_{i,j-1} + \sum_{i \le k < (j-l)} Q_{i,k-1} \cdot Q^{bp}_{k,j},$$

$$Q^{bp}_{i,j} = \begin{cases} Q_{i+1,j-1} \cdot \exp(-E_{bp}/RT) & \text{if } S_i, S_j \text{ can form base pair} \\ 0 & \text{otherwise} \end{cases}.$$

Listing 1 presents the implementation of computing Q and Q^{bp}. The input data are RNA sequence S as a chain of nucleotides from the alphabet AUGC (adenine, uracil, guanine, cytosine), minimal loop length l (i.e. minimal number of enclosed positions), energy weight of base pair E_{bp} and normalized temperature RT. The memory complexity of the arrays is $\mathcal{O}(n^2)$, while the time complexity of a direct implementation of this algorithm is $\mathcal{O}(n^3)$ in the sequence of length N.

Listing 1. Serial loop nest implementing the McCaskill partition function computation.

```
if (N>=1 && l>=0 && l<=5)
  for (i=N-1; i>=0; i--)
    for (j=i+1; j<N; j++){
      Q[i][j] = Q[i][j-1];
        for (k=0; k<j-i-l; k++){
          Qbp[k+i][j] = Q[k+i+1][j-1] * ↙
              ↳ ERT * paired(k+i,j-1);
          Q[i][j] += Q[i][k+i] * ↙
              ↳ Qbp[k+i][j];
        }
    }
```

Given these partition function terms, we can find base pair probabilities as well as probabilities that a certain subsequence is unpaired, in the manner discussed in [16].

III. Automatic code optimization

The code presented in Listing 1 was optimized (tiled and parallelized) by means of the TRACO compiler [13]. To tile a loop nest, TRACO forms original rectangular tiles whose size is provided by the user. Then TRACO extracts dependences available in the loop nest applying the Petit tool [17], which returns 19 dependence relations describing all the dependences in the loop nest implementing McCaskill's algorithm. Extracted dependence relations are a mathematical representation of the dependence graph whose nodes are statement instances of the loop nest, while each edge states for a dependence between a pair of nodes.

Using the union of obtained dependence relations, TRACO calculates the transitive dependence of the dependence graph, for this purpose, it uses a function implementing the algorithm presented in paper [18]. The transitive closure of a given graph, G, is a graph, G', such that (i, j) is an edge in G' if there exists a directed path from i to j in G. It is worth noting that in general, the dependence graph is parametric – the number of nodes depends on the upper bounds of loop iterators, which are represented with parameters. So, a special algorithm should be applied to calculate the transitive closure of a parametric dependence graph.

TRACO carries out the following calculations according to the algorithm presented in paper [11]. First, applying the transitive closure of the dependence graph, it checks whether the original (rectangular) tiles are valid. A valid tile with identifier I does not contain any statement instance that is the destination of the dependence whose source belongs to

TABLE I
EXECUTION TIME (IN SECONDS) OF THE ORIGINAL, TRACO AND PLUTO TILED CODES.

N	1 Thread			2 Threads	4 Threads	8 Threads	16 Threads	32 Threads	48 Threads
	Orig.	PLuTo	TRACO	TRACO					
1000	1.6883	0.6096	0.9893	1.0862	0.4055	0.3203	0.1978	0.1500	0.1744
2000	23.1211	5.4271	11.8613	9.9558	6.4051	3.6992	2.1890	1.4105	1.2818
3000	138.5503	17.843	67.6431	48.1494	27.0968	14.9998	9.0412	5.1811	4.7234
4000	391.8773	51.9886	253.2109	169.3396	94.9911	47.8198	27.1342	14.7676	14.0896
5000	874.2910	132.3715	545.7719	378.3082	210.9896	110.7063	54.2998	34.1555	32.9895

the tile whose identifier is lexicographically greater than I. If all original tiles are valid, it directly generates target code, otherwise, it corrects original tiles so that all target tiles are valid under lexicographical order. Such a correction is realized by means of transitive closure.

It is well-known that if all tiles are valid, then the corresponding inter-tile dependence graph describing dependences among tiles is acyclic, so there exists a schedule that assigns a discrete time to the corresponding tile to execute it [19]. If two or more tiles have the same schedule time, they can be run in parallel.

To extract a valid tile schedule, we need a dependence relation, which describes inter-tile dependences. Using obtained valid tiles, TRACO forms such a relation according to the way described in paper [11]. Then TRACO finds a valid tile schedule applying the ISL scheduler [12], which uses the PluTo scheduler with Feautrier's one [19], [20] as fallback. The PluTo scheduler constructs a set of independent affine schedule functions that guarantee a small dependence distance over the schedule constraints. The basic idea of Feautrier's scheduler implemented in the ISL library is to carry as many dependences as possible in each level of a multi-dimensional schedule.

For the examined loop nest, the ISL scheduler returns the following tile schedule for each statement: $[ii, jj, kk] \rightarrow [ii + jj]$, which means that the tile represented with identifier $[ii, jj, kk]$ is mapped to execution time $[ii + jj]$. Such a schedule corresponds to the well-known loop skewing transformation [21]. It is a convenient method to implement the wavefront method of executing a loop nest in parallel, which creates a "wave" that passes through the iteration space. Skewing changes the iteration vectors for each iteration by adding the outer loop index value to the inner one.

To generate parallel code on the tile level, to each loop statement, we apply the skewing transformation $(ii + jj)$ to form the following schedule allowing for parallel code generation.

$SCHED\_PAR := N \rightarrow \{ (i, j, k) \rightarrow$
$(ii + jj, jj, kk, i, j, k) \mid constraints\}$,

where $constraints$ are the constraints of a set representing target tiles for a given loop nest statement.

That schedule maps each instance of a statement to a time partition whose all tiles can be executed in parallel. TRACO passes those schedules to the input of the ISL code generator, which generates target pseudo-code. The TRACO post-processor generates target parallel compilable code in the OpenMP standard [22], which is presented at the repository https://github.com/markpal/hpc_mea. In that code, the first loop is serial, it enumerates time partitions including target tiles. The second loop is parallel, it runs tiles belonging to a given time partition, the reminding loops are serial. Intra-tile dependences (dependences within a tile) are honored because within each target tile, statement instances are executed in lexicographical order (serially).

It is worth noting that TRACO code is less regular than that generated with PLuTo because target tiles generated with TRACO are irregular while PLuTo generates regular tiles except from boundary ones.

IV. EXPERIMENTAL STUDY

This section presents the results of the comparison of the performance of TRACO and PLuTo tiled codes implementing McCaskill's algorithm. To carry out experiments, we have used a computer with the following features: Intel Xeon CPU E5-2699 v2, 3.6GHz, 24 cores, 48 Threads, 45 MB Cache, 16 GB RAM. Programs were compiled with the Intel C Compiler (icc 15.0.2) and optimized at the `-O3` level (more aggressive optimization recommended for loops involving intensive floating point calculations). Parallelism of target code is represented in the OpenMP standard. We discovered empirically that the best tile size for TRACO code is [1x128x16], i.e., the first loop should not be tiled. For tiled code generated with PLuTo, we empirically discover that the best tile size is [16x16x16].

The McCaskill loop nest can be tiled by both PLuTo and TRACO, however, only TRACO allows us to parallelize target code. Although the serial code produced with PluTo is very cache-efficient, the compiler is unable to find any affine schedule allowing for parallel execution of generated tiles. TRACO generates valid tiles applying the transitive closure of the dependence graph built for the McCaskill loop nest, then it forms a relation, which represents inter-tile dependences. Finally, using that relation, it applies the ISL scheduler to get a valid tile schedule to generate parallel code on the tile level.

Table 1 presents execution times (in seconds) for various RNA sequence lengths. Figure 2 depicts the speed-up (a ratio of T_1 over T_n, elapsed times of 1 and n threads) of tiled programs for N = 5000 (roughly the size of the longest human mRNA). Analyzing the obtained results, we may conclude that the TRACO code performance overcomes that of the PLuTo serial one for eight and more threads. The worse performance

of the TRACO code for the few number of threads is caused with target code irregularity (see the previous section). The lack of parallelism limits speed-up and scalability of the PluTo loop nest implementing McCaskill's algorithm on the modern multi-core machine used for experiments.

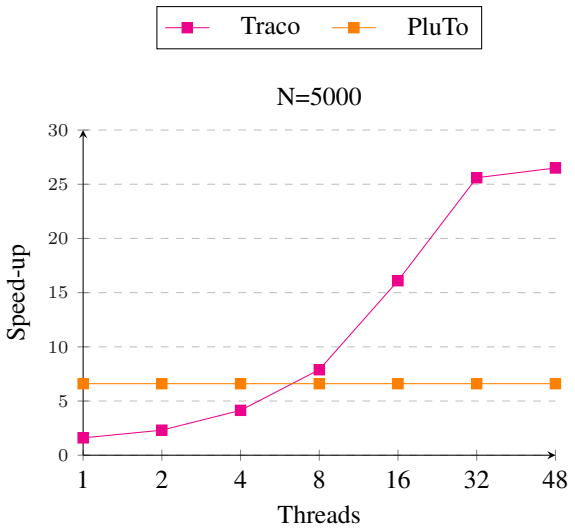

Fig. 1. Speed-up of TRACO and PLuTo codes.

V. Conclusion

In this paper, we presented the usage of the TRACO compiler to optimize the loop nest implementing the McCaskill pattern function calculation. TRACO applies the transitive closure of dependence graphs to generate valid tiles under lexicographical order. Then it forms a relation describing inter-tile dependences and uses the ISL scheduler to obtain a valid tile schedule allowing for generation of parallel tiled code.

Applying optimization techniques based on affine transformations implemented in the PLuTo compiler allows for generation of only serial highly cache efficient code without any parallelism code. The proposed approach outperforms code generated with the PluTo compiler starting up from eight threads.

It is an ongoing task to find cache efficient optimization for NPDP problems in bioinformatics with $\mathcal{O}(n^3)$ and $\mathcal{O}(n^4)$ complexity and non-trivial dependence patterns. In future, we plan to optimize programs implementing base pair probabilities calculation as well as prediction of their structure with maximum expected accuracy (MEA) for a given RNA sequence.

References

[1] 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

[2] U. Bondhugula, A. Acharya, and A. Cohen, "The pluto+ algorithm: A practical approach for parallelization and locality optimization of affine loop nests," *ACM Trans. Program. Lang. Syst.*, vol. 38, no. 3, pp. 12:1–12:32, Apr. 2016. doi: 10.1145/2896389

[3] U. Bondhugula *et al.*, "A practical automatic polyhedral parallelizer and locality optimizer," *SIGPLAN Not.*, vol. 43, no. 6, pp. 101–113, Jun. 2008. doi: 10.1145/1379022.1375595 Http://pluto-compiler.sourceforge.net/.

[4] 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

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

[6] 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.

[7] M. Fekete, I. L. Hofacker, and P. F. Stadler, "Prediction of rna base pairing probabilities on massively parallel computers," *Journal of Computational Biology*, vol. 7, no. 1-2, pp. 171–182, 2000. doi: 10.1089/10665270050081441

[8] 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

[9] M. Palkowski and W. Bielecki, "Tuning iteration space slicing based tiled multi-core code implementing Nussinov's rna folding," *BMC Bioinformatics*, vol. 19, no. 1, p. 12, Jan 2018.

[10] C. Zhao and S. Sahni, "Cache and energy efficient algorithms for nussinov's rna folding," *BMC Bioinformatics*, vol. 18, no. 15, p. 518, Dec 2017.

[11] W.Bielecki and M. Palkowski, "Tiling of arbitrarily nested loops by means of the transitive closure of dependence graphs," *International Journal of Applied Mathematics and Computer Science (AMCS)*, vol. Vol. 26, no. 4, p. 919–939, December 2016. doi: 10.1515/amcs-2016-0065

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

[13] W. Bielecki and M. Palkowski, "A parallelizing and optimizing compiler - traco," 2013. [Online]. Available: http://traco.sourceforge.net

[14] W. Pugh and D. Wonnacott, "An exact method for analysis of value-based array data dependences," in *Sixth Annual Workshop on Programming Languages and Compilers for Parallel Computing*. Springer-Verlag, 1993.

[15] M. Raden, S. M. Ali, O. S. Alkhnbashi, A. Busch, F. Costa, J. A. Davis, F. Eggenhofer, R. Gelhausen, J. Georg, S. Heyne, M. Hiller, K. Kundu, R. Kleinkauf, S. C. Lott, M. M. Mohamed, A. Mattheis, M. Miladi, A. S. Richter, S. Will, J. Wolff, P. R. Wright, and R. Backofen, "Freiburg RNA tools: a central online resource for RNA-focused research and teaching," *Nucleic Acids Research*, vol. 46, no. W1, pp. W25–W29, 2018. doi: 10.1093/nar/gky329

[16] 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. doi: 10.1002/bip.360290621. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/bip.360290621

[17] W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott, *New User Interface for Petit and Other Extensions*, 1996.

[18] W. Bielecki, K. Kraska, and T. Klimek, "Using basis dependence distance vectors in the modified Floyd–Warshall algorithm," *Journal of Combinatorial Optimization*, vol. 30, no. 2, pp. 253–275, 2014.

[19] P. Feautrier, "Some efficient solutions to the affine scheduling problem: I. one-dimensional time," *Int. J. Parallel Program.*, vol. 21, no. 5, pp. 313–348, 1992.

[20] ——, "Some efficient solutions to the affine scheduling problem: II. multidimensional time," *Int. J. Parallel Program.*, vol. 21, no. 5, pp. 389–420, 1992.

[21] M. Wolfe, "Loops skewing: The wavefront method revisited," *International Journal of Parallel Programming*, vol. 15, no. 4, pp. 279–293, 1986.

[22] OpenMP Architecture Review Board, "OpenMP application program interface version 4.0," 2012. [Online]. Available: http://www.openmp.org/mp-documents/OpenMP4.0RC1_final.pdf