Proceedings of the of the 17th Conference on Computer Science and Intelligence Systems pp. 475–478
DOI: 10.15439/2022F4
ISSN 2300-5963 ACSIS, Vol. 30

# Automatic code optimization for computing the McCaskill partition functions

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

***Abstract*—In this paper, we present the application of three automatic source-to-source compilers to code implementing McCaskill's bioinformatics algorithm. It computes probabilities of various substructures for RNA prediction. McCaskill's algorithm is compute and data intensive and it is within dynamic programming. A corresponding programming code exposes non-uniform dependences that complicate tiling of that code. The corresponding code is represented within the polyhedral model. Its optimization is still a challenging task for optimizing compilers employing multi-threaded loop tiling. To generate optimized code, we used the popular PLuTo compiler that finds and applies affine transformations, the TRACO compiler based on calculating the transitive closure of loop dependence graphs, and the newest polyhedral tool DAPT implementing space-time tiling. An experimental study fulfilled on two multi-core machines: an AMD Epyc with 64 threads and a 2x Intel Xeon Platinum 9242 with 192 threads demonstrates considerable speedup, high locality, and scalability for various problem sizes and the number of threads of generated codes by means of space-time tiling.**

## I. Introduction

McCASKILL'S algorithm is an efficient dynamic programming one to return the value of the computed partition function $Z = \sum_P exp(-E(P)/RT)$, where $P$ represents all possible nested structures formed by a given RNA sequence $S$, $E(P)$ is the energy of structure $P$, $R$ is the gas constant, and $T$ represents temperature [1].

The approach computes the structure probabilities of each individual base pair in the RNA sequence. These probabilities are used for simultaneous folding and alignment in algorithms to predict an RNA structure with a maximum expected accuracy (MEA) [2].

Each base pair of a structure contributes a fixed energy term $E_{bp}$ independent of its context. Under such an assumption, a partition function for a sub-sequence from position $i$ to position $j$ is represented with table $Q_{i,j}$, while table $Q^{bp}_{i,j}$ holds the values of the partition function of the sub-sequences for a base pair or 0 when base pairing is absent.

The following calculations are used to compute the values of the partition functions and inserted as elements of tables $Q_{i,j}$ and $Q^{bp}_{i,j}$.

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

Listing 1. Code of the McCaskill partition function computation.

```
if (N>5 && 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-1; 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];
}  }
```

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

Listing 1 presents the code implementation computing $Q_{i,j}$ and $Q^{bp}_{i,j}$ filled with random double values (data in arrays do not affect the speed of the code). *ERT* is equal to $\exp(-E_{bp}/RT)$. To simplify target tiled code generation, we replaced $k$ with $k+i$ and the innermost loop boundaries from 0 to $j-i-1$.

Many algorithms in bioinformatics are within dynamic programming (DP). Programming loop nests implementing those algorithms can be represented within the polyhedral model. That model is used in many optimization compilers, which automatically generate efficient parallel tiled code. However, the code implementing McCaskill's algorithm exposes non-uniform dependences that make it difficult effective parallelization and tiling of that code.

A polyhedral optimizer generally improves code locality by means of loop tiling, which groups loop statement instances within smaller blocks (tiles). This allows for reuse provided that the block fits in cache. In parallel tiled code, tiles are enumerated as indivisible macro statements. This increases the granularity of parallel code that often improves the performance of that code executed in parallel systems with shared memory.

Dynamic programming codes expose non-uniform dependences, which limit applying commonly known optimization techniques such as permutation, diamond tiling [3], or index

set splitting [4] very well trained, for example, on stencils.

## II. Opitmizing compilers used for experiments

Modern automatic optimizing compilers, for example, PLuTo [5], demonstrate the success of using the polyhedral model. PLuTo extracts and applies affine functions to parallelize and tile serial code. Target parallel tiled code demonstrates good efficiency on modern multi-core computers with shared memory in particular for stencils.

For a given loop nest statement, compilers based on affine transformations use the relation $[I] \rightarrow [t = C * I + c]$, where $I$ is the loop statement iteration vector; $t$ is the time assigned to execute iteration $I$; $C * I + c$ represents the affine function. When two statement instances get the same execution time, they can be run in parallel. To extract the unknown matrix $C$ and unknown vector $c$, firstly, for each loop nest statement, time-partition constraints are formed by applying extracted dependences. Then the time-partition constraints are resolved for elements of matrix $C$ and elements of vector $c$.

The PLuTo engine is used in other compilers, for example, in Apollo[6], PPCG [7], PTile[8], and Autogen framework [9] as well as in commercial IBM-XL and R-STREAM from the Reservoir Lab [10].

TRACO is based on the slicing framework introduced in paper [11]. It calculates the transitive closure of a dependence graph, which is used to fulfill corrections of original rectangular tiles. The goal of the correction is to eliminate all cycles among target tiles. This allows us to enumerate target tiles in lexicographic order.

After tile correction, the inter-tile dependence graph does not contain any cycle and any technique of loop nest parallelization can be used to generate parallel code, details are presented in paper [12]. TRACO uses the commonly known wive-fronting technique for tiled code parallelization. For its implementation, it applies the ISL library.

PLuTo and TRACO have some limitations. PLuTo may not find the number of linearly independent solutions to time partitions constraints that is equal to the number of the loops surrounding a loop nest statement. This reduces the dimension of target tiles and as a consequence target code locality may be low. For example, it is not able to tile each loop nest for the Nussinov and Knuth algorithms [13], i.e., instead of 3D tiles it generates only 2D tiles. TRACO, in general, may generate irregular tiles that reduce code locality and worsen multiple thread work balance [14].

To generate regular code and increase tile dimension, DAPT implements space-time tiling. First DAPT generates space tiles according to the technique presented in paper [15]. Then it splits each space tile into multiple time slices. Each time slice is represented with a number of time partitions found by means of the ISL scheduler. The number of time partitions within the time slice is defined by the user. As a result, the tile dimension is increased by one. Target code enumerates smaller tiles (time slices) within each space tile. This increases code locality due to increasing the probability of catching all the data associated with each smaller tile in cache provided that the number of time partitions forming the time slice is chosen properly.

However, each of the mentioned above automatic source-to-source compilers is able to generate tiled code for the McCaskill algorithm and we conducted a comparison analysis of the performance of codes generated by them on two modern multi-core machines.

## III. Experimental study

In this section, we present the results of an experimental study with PLuTo, TRACO, and DAPT codes implementing the McCaskill partition function computation. All target parallel tiled codes were compiled using the Intel C++ Compiler (icc) and GNU C++ Compiler (g++) with the -O3 flag of optimization.

To carry out experiments, we used two multi-processor machines: an 2x Intel Xeon Platinum 9242 CPU (2.30GHz, 2x96 threads, 71,5 MB Cache, compiler icc 21.3.0, 2019), and an AMD Epyc 7542 (2.35 GHz, 32 cores, 64 threads, 128MB Cache, compiler g++ 9.3.0, 2019).

The code generated with DAPT is presented in Listing 2, while the codes generated with PLuTo and TRACO can be found at https://github.com/markpal/NPDP_Bench/blob/main/mcc/mcc_dapt.cpp, they are too long to be inserted in this paper.

It is worth noting that tiles generated with TRACO are irregular, they can be fixed or parametric (the size of such tiles is unbounded). PLuTo generates regular fixed tiles except from boundary ones.

Space-time tiling implemented in DAPT generates regular tiles and the tile dimension is one more than that of tiles generates with PLuTo.

In all examined compilers, for parallelism representation, the OpenMP standard is used. For different sizes examined by us, by means of experiments, we discovered that the best tile size for the TRACO target code is 1x128x16. This means that the outermost loop in the loop nest should not be tiled. For the target code generated with PLuTo, the best tile size is 16x16x16. For the DAPT code, the optimal size is 16x16x16 for space slices and the size of the time slice (the number of time partitions within the space tile) is 2.

The McCaskill code can be tiled by all the compilers used for us for experiments. However, only TRACO and DAPT allow us to generate parallel tiled code. The serial code generated with PluTo is very cache-efficient, but PluTo is unable to extract any affine schedule allowing for parallelism of target code. TRACO generates target code applying the transitive closure of the dependence graph for the McCaskill loop nest, then it builds a relation representing inter-tile dependences. Finally, using that relation, it applies the ISL scheduler to extract a valid tile schedule, which is used to generate parallel tiled code. DAPT applies the wave front technique to generate target parallel tiled code.

Tables 1 and 2 hold execution times (in seconds) for the PLuTo, TRACO, and DAPT codes for various RNA sequence

Listing 2. Parallel tiled loop nests of the McCaskill algorithm generated with DAPT .

```
1  if (1 >= 0 && 1 <= 5 && N >= 6) {
2   for(w0 = -1; w0 <= (N - 1) / 8; w0 += 1) {
3   #pragma omp parallel for
4   for(h0 = max(w0 - (N+7) / 8 + 1, -((N+5) / 8)); h0 <= min(0, w0); h0 += 1) {
5     for(i0 = max(max(-N+2, -8*w0 + 8*h0-6), 8*h0); i0 <= min(0, 8*h0+7); i0++) {
6      for(i1 = max(8*w0 - 8*h0, -i0+1); i1 <= min(N-1, 8*w0 - 8*h0 + 7); i1++) {
7       Q[-i0][i1] = Q[-i0][i1 - 1];
8       for (i3 = 0; i3 < -1 + i0 + i1; i3 += 1) {
9        Qbp[-i0+i3][i1] = ((Q[-i0+i3+1][i1-1] * (ERT)) * paired((-i0+i3), (i1-1)));
10       Q[-i0][i1] += (Q[-i0][-i0 + i3] * Qbp[-i0 + i3][i1]);
11 }}}}}
```

TABLE I
EXECUTION TIME OF THE PARALLEL TILED CODES FOR AN INTEL XEON PLATINUM 9242 USING 192 HARDWARE THREADS.

| N | Serial | PLuTo | TRACO | Dapt |
|---|---|---|---|---|
| 1000 | 0,61 | 0,51 | 0,14 | 0,07 |
| 2000 | 8,81 | 4,73 | 0,87 | 0,62 |
| 3000 | 38,63 | 21,73 | 3,43 | 2,22 |
| 4000 | 106,17 | 58,96 | 8,58 | 5,06 |
| 5000 | 227,41 | 116,75 | 17,73 | 9,88 |
| 6000 | 420,26 | 206,48 | 29,43 | 17,45 |
| 7000 | 721,88 | 333,84 | 47,61 | 28,16 |
| 8000 | 1157,84 | 503,13 | 69,81 | 46,41 |
| 9000 | 2575,45 | 713,69 | 98,67 | 71,33 |
| 10000 | 3676,3 | 1005,44 | 135,01 | 105,86 |

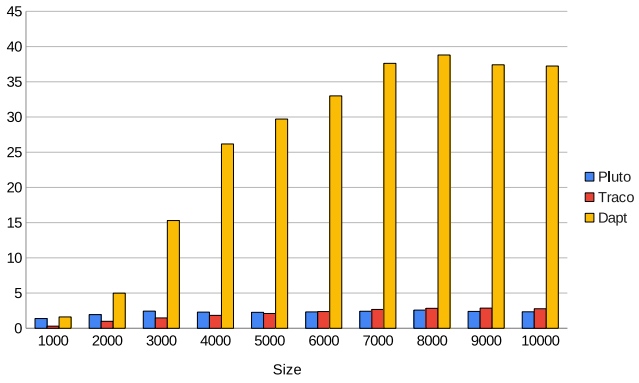


Fig. 1. Speedups of the parallel tiled codes generated by applying TRACO, PLuTo, and Dapt for an Intel Xeon Platinum 9242 and RNA sequence sizes from 1000 to 10000.

TABLE II
EXECUTIONS TIME OF THE PARALLEL TILED CODES FOR AN AMD EPYC 7542 USING 64 HARDWARE THREADS.

| N | Serial | PLuTo | TRACO | Dapt |
|---|---|---|---|---|
| 1000 | 0,87 | 0,63 | 2,89 | 0,54 |
| 2000 | 10,22 | 5,28 | 10,33 | 2,05 |
| 3000 | 46,19 | 18,99 | 31,46 | 3,02 |
| 4000 | 129,29 | 56,33 | 70,56 | 4,94 |
| 5000 | 289,41 | 127,89 | 137,64 | 9,74 |
| 6000 | 572,61 | 246,36 | 241,06 | 17,35 |
| 7000 | 999,91 | 413,36 | 373,89 | 26,58 |
| 8000 | 1567,43 | 607,81 | 553,21 | 40,39 |
| 9000 | 2231,09 | 932,75 | 779,01 | 59,63 |
| 10000 | 3043,21 | 1299,29 | 1096,88 | 81,72 |

lengths on an Intel Xeon Platinum 9242 (192 hardware threads) and an AMD Epyc 7542 (64 hardware threads), respectively. Figures 1 and 3 show the speed-up (a ratio of $T_1$ over $T_n$, where $T_1$ and $T_n$ are execution times for one and $n$ threads used for code running, respectively) of parallel tiled programs for RNA sequence sizes from 1000 to 10000 for an Intel Xeon Platinum 9242 (192 hardware threads) and an AMD Epyc 7542 (64 hardware threads), respectively. Figures 2 and 4 show how target code speed-up depends on the number of threads for an Intel Xeon Platinum 9242 (192 hardware threads) and an AMD Epyc 7542 (64 hardware threads), respectively, for $N = 5000$ (roughly the size of the longest human mRNA).

Analyzing the presented results of experiments, we may state that the DAPT code overcomes considerable those of PLuTo and TRACO ones. The TRACO code overcomes that of PLuTo for eight and more threads. The worse efficiency of the TRACO code for a few thread numbers is caused with the irregularity of the target code (see the previous section).

## IV. CONCLUSION

We presented the results of a comparative performance analysis of three tiled codes generated with optimizing compilers PLuTo, TRACO, and DAPT for the McCaskill partition function calculation. The best performance demonstrates the DAPT code due to the fact that it applies space-time tiling allowing us to increase the tile dimensionality by one in comparison with that of PLUTO. That makes all tiles to be regular and of fixed size. A proper choice of a tile size allows us to hold all the data associated with each tile in cache that increases code locality.

## REFERENCES

[1] 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,

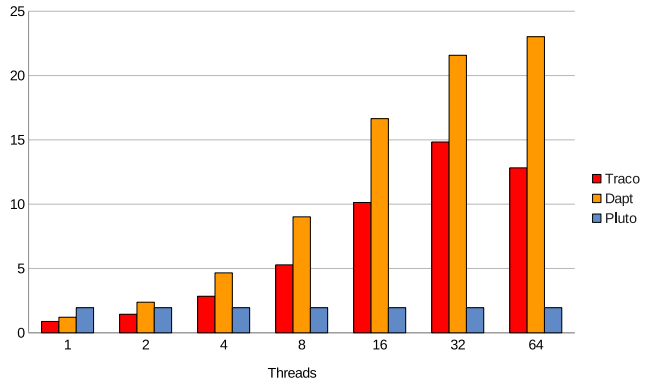


Fig. 4. Speedups of the parallel tiled codes generated by applying TRACO, PLuTo, and Dapt for an AMD Epyc 7542 using various number of hardware threads

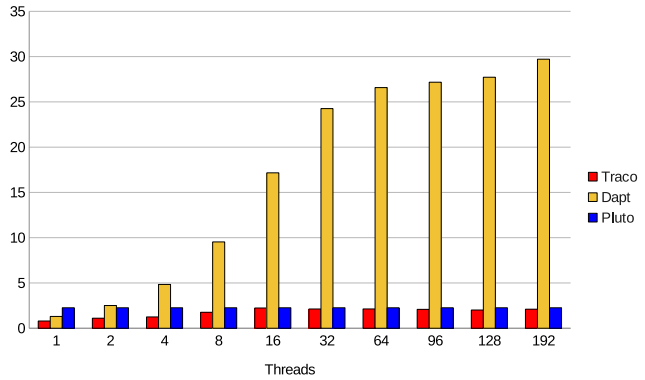


Fig. 2. Speedups of the parallel tiled codes generated by applying TRACO, PLuTo, and Dapt for an Intel Xeon Platinum 9242 using various number of hardware threads.

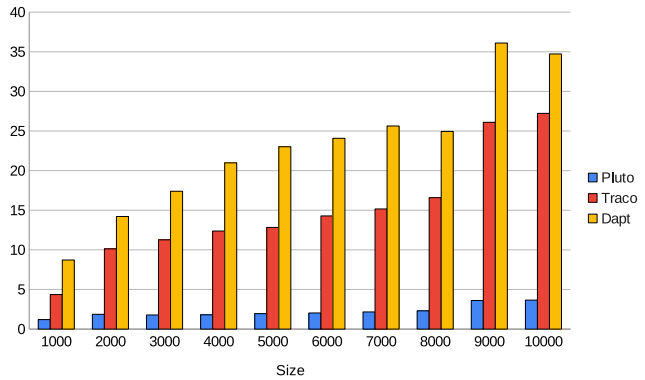


Fig. 3. Speedups of the parallel tiled codes generated by applying TRACO, PLuTo, and Dapt for an AMD Epyc 7542 and RNA sequence sizes from 1000 to 10000.

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

[2] M. Raden, S. M. Ali, O. S. Alkhnbashi, A. Busch, F. Costa, J. A. Davis, F. Eggenhofer, R. Gelhausen, J. Georg, S. Heyne *et al.*, "Freiburg rna tools: a central online resource for rna-focused research and teaching," *Nucleic acids research*, vol. 46, no. W1, pp. W25–W29, 2018.

[3] 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

[4] 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

[5] 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/.

[6] J. M. M. Caamaño, A. Sukumaran-Rajam, A. Baloian, M. Selva, and P. Clauss, "Apollo: Automatic speculative polyhedral loop optimizer," in *IMPACT 2017-7th International Workshop on Polyhedral Compilation Techniques*, 2017, p. 8.

[7] S. Verdoolaege, J. Carlos Juega, A. Cohen, J. Ignacio Gómez, C. Tenllado, and F. Catthoor, "Polyhedral parallel code generation for cuda," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, jan 2013. doi: 10.1145/2400682.2400713. [Online]. Available: https://doi.org/10.1145/2400682.2400713

[8] M. M. Baskaran, A. Hartono, S. Tavarageri, T. Henretty, J. Ramanujam, and P. Sadayappan, "Parameterized tiling revisited," in *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*, ser. CGO '10. New York, NY, USA: ACM, 2010. ISBN 978-1-60558-635-9 pp. 200–209.

[9] R. Chowdhury, P. Ganapathi, J. J. Tithi, C. Bachmeier, B. C. Kuszmaul, C. E. Leiserson, A. Solar-Lezama, and Y. Tang, "Autogen: Automatic discovery of cache-oblivious parallel recursive algorithms for solving dynamic programs," *ACM SIGPLAN Notices*, vol. 51, no. 8, pp. 1–12, 2016.

[10] 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

[11] 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.

[12] 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, pp. 919–939, December 2016. doi: 10.1515/amcs-2016-0065

[13] W. Bielecki and M. Palkowski, "Space-time loop tiling for dynamic programming codes," *Electronics*, vol. 10, no. 18, p. 2233, 2021.

[14] 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

[15] W. Bielecki and M. Poliwoda, "Automatic parallel tiled code generation based on dependence approximation," in *International Conference on Parallel Computing Technologies*. Springer, 2021, pp. 260–275.