RNA Secondary Structure Prediction using AlphaZ

Tanveer Pathan

Department of Electrical and Computer Engineering
Colorado State University, Fort Collins
Committee

- Adviser
  - Dr. Sanjay Rajopadhye

- Members
  - Dr. Wim Bohm
  - Dr. Sudeep Pasricha
Motivation

- **AlphaZ**
  - Research tool
  - Handling real life problems?

- **RNA Secondary Structure Prediction**
  - Evaluation of Internal loops: Takes the most time
  - Time complexity: \( N^4 \)
  - Fast evaluation of internal loops: \( N^3 \) time complexity
    - Proposed by Lyngso et. al, in 1999
    - Implementation: No public release yet
    - Why?
Outline

- Why AlphaZ?
  - What is AlphaZ?

- Bio-informatics
  - RNA Secondary Structure Prediction
  - UNAFold

- C – Alphabets – C
  - Optimizations
  - Results

- Conclusions and Future work
Why AlphaZ?

- Processor Architectures are evolving rapidly
- Speed-up is no longer easy to achieve by following Moore’s law

Result
- Parallel architectures
- Multi-core processors

Programming for these architectures
- Is it easy?
- How steep is the learning curve?
Why AlphaZ?

- Which is the best architecture for the algorithm?
- How many times do I implement the same algorithm?
- Will I have to learn new programming models and language extensions?
  - OpenMP, POSIX threads, MPI
  - CUDA, CTM, SSE, 3DNow!
Why AlphaZ?

- Answer

- Input language for AlphaZ: *Alphabets*

- Alphabets
  - Equational specification of the algorithm
  - Domains associated with the equations

- Alphabets programs
  - Architecture independent
What is AlphaZ?

- **AlphaZ**
  - Functionality of polyhedral model
    - Transformations
    - Code generation
  - Uses COREquations

- **COREquations**
  - Implementation for polyhedral model framework
  - Provides foundation for AlphaZ
    - Basic polyhedral operations
    - Alphabets program parser
What is AlphaZ?

- Polyhedral Model
  - A representation of computations surrounded by affine loops as a mathematical model

\[
\begin{align*}
\text{Equations representing computation} & \quad + \\
\text{Domains representing loops} & \quad \parallel \\
\text{Polyhedral representation of the algorithm}
\end{align*}
\]
What is AlphaZ?

AlphaZ system framework
What is AlphaZ?

- Write equational specifications in alphabets

```plaintext
[External function declarations]
affine systemName {sizeParameters | sizeParameterDomains} given
  [data type] inputVars {domain};
returns
  [data type] outputVars {domain};
using
  [data type] localVars {domain};
through
  [Equations defining outputVars and localVars]
```

*Structure of an alphabets program*
What is AlphaZ?

- AlphaZ will take care of the rest
  - Currently with a little human feedback, which will eventually fade away

- Enables systematic optimizations
  - Transformations
  - Analyses

- Want code to realize your algorithm?
  - Answer: Code generators
  - Can even generate a wrapper program to test your algorithm quickly
Bio-informatics

- Applying statistics and computer science to molecular biology
- A rapidly advancing field
  - RNA folding
  - Protein folding
  - Sequence alignment
- Bio-informatics algorithms
  - New algorithms → developed
  - Old algorithms → improved
  - When will they be implemented?
  - Should a biologist or a mathematician write efficient code?
Bio-informatics

- Is performance programmer a solution?
  - Time to understand
  - Time to implement
  - Best target architecture?
  - Shelf-life of the code?

- Again, the answer is AlphaZ
RNA Secondary Structure Prediction

- RNA (Ribonucleic Acid)
  - A biological molecule consisting a sequence of nucleotide units.

- Four types of Nitrogen bases
  - Adenine (A)
  - Guanine (G)
  - Cytosine (C)
  - Uracil (U)

- Watson–crick base pairing
  - A – U
  - G – C
RNA Secondary Structure Prediction

RNA Sequence

GGGGUACCCCAAGCGGUAAGGCACCGGAAUUCUGAUUCCGGCAUUCGGAGGUUCGAAUCCUCCGUACCCCAACCC

Secondary Structure

Tertiary Structure

SOURCE: mPi Informatik
RNA Secondary Structure showing stacked pairs, hairpin loops, bulges, internal loops and multi-branched loops

RNA Secondary Structure Prediction

- Secondary structure on a RNA can be determined using
  - Experimental methods
  - Computational methods

- UNAFold
  - Computational package
  - Based on thermodynamic energy model
UNAFold equations
- Based on dynamic programming
- Recurrence equations

Three tables Q, Q', QM
For a RNA sequence of length N with $1 \leq i < j \leq N$

$$Q(i, j) = \min \begin{cases} 
  b + Q(i+1, j) \\
  b + Q(i, j-1) \\
  c + E_{ND}(i, j) + Q'(i, j) \\
  QM(i, j) 
\end{cases}$$
UNAFold

\[ Q'(i, j) = \begin{cases} 
E_H(i, j) \\
E_S(i, j) + Q'(i + 1, j - 1) \\
QBI(i, j) \\
\infty \\
a + c + E_ND(j, i) + QM(i + 1, j - 1) 
\end{cases} \]

\[ QM(i, j) = \min_{i+4 \leq k \leq j-5} \{ Q(i, k-1) + Q(k, j) \} \]

QBI represents internal loops
UNAFold

\[ Q_{BI}(i, j) = \min_{4 \leq d \leq j - i - 3, i < i' < j - d} \left\{ E_{BI}(i, j, i', i' + d) + Q'(i', i' + d) \right\} \]

- To evaluate internal loops
  - Computational complexity: \( N^4 \)

Domain of QBI
C – Alphabets – C

UNAFold C Program

profile

Output code

Compute intensive part

transformations

Automatic/ Manual

Alphabets code

Code Generator

AlphaZ

Output code
UNAFold program can be modularized into:

- 3100 lines
- 120 lines
- 60 lines

Algorithm module: 98% of execution time
C – Alphabets – C

Extracting equations

void foo (int N, int *A, int *B, int *C)
{
    int i;
    for (i = 1; i < N; i++)
    {
        if (i < 10)
        {
            C[i] = A[i] * B[i];
        }
        else
        {
            C[i] = A[i] + B[i];
        }
    }
}

C program

affine foo {N | N > 0}
given
    int A {i | 0 < i < N};
    int B {i | 0 < i < N};
returns
    int C {i | 0 < i < N};
through
    C[i] = case
        { | i < 10}: A[i] * B[i];
        { | i >= 10}: A[i] + B[i];
esac;

Alphabets program
Optimizations

- Types of optimizations that can be done in AlphaZ
  - Exploiting re-use in reductions
  - Specifying schedule
  - Specifying processor allocation
  - Specifying memory mapping
  - Tiling

- Since AlphaZ is still in development the scope of all these optimizations is limited
Currently, WriteC is the most functional code generator in AlphaZ

Limitations of WriteC
- Demand-driven code generator
- Since its demand-driven, user cannot give a schedule to WriteC
- WriteC is inherently sequential
  - Hence, processor allocation does not come into the scenario
- Uses its own default memory map
  - User cannot change the memory map when using WriteC
- No tiling, since its demand-driven
Exploit re-use in the QBI term

$N^4$ time complexity

Goal

- $N^3$ time complexity

This result has already been deduced by Lyngso et. al.

AlphaZ approach

- Systematic
Optimizations

- The QBI term

\[ QBI(i, j) = \min_{i'} \{ E_{BI}(i, j, i', i' + d) + Q'(i', i' + d) \} \]

where

\[ E_{BI}(i, j, i', i' + d) = \text{Asym}(i' - i - 1, j - i' - d - 1) \]

\[ + E_S(i', i' + d) \]

- QBI term has double reduction
- To expose the re-use
  - Split the \( E_{BI} \) term
  - Decompose double reduction
  - Factor out the invariant terms
  - Detect the scan

Manually
Reduction Decomposition
Factor Out From Reduction
Simplifying Reductions
Optimizations

- Final equations.

\[ QBI(i, j) = E_S(i, j) + \min_{4 \leq d \leq j - i - 3} \{ X(i, j, d) \} \]

where

\[ X(i, j, d) = S_P(j - i - d - 2) + Y(i, j, d) \]

and

\[ Y(i, j, d) = \min \left\{ \begin{array}{c}
Y(i + 1, j - 1, d) \\
\min_{i + 1 < i' < j - d - 1} \left\{ \begin{array}{c}
Asym(i' - i - 1, j - i' - d - 1) \\
+ E_S(i', i' + d) \\
+ Q'(i', i' + d)
\end{array} \right\}
\end{array} \right\} \]
Execution time comparison between original and fast-i-loops version generated by AlphaZ
Execution time comparison between original and fast-i-loops version generated by AlphaZ for real RNA sequences
Conclusions

- Programming using equations is beneficial
  - AlphaZ tool provides the platform
- AlphaZ can handle real life applications
  - Our work provides justification
- Forms a base for future optimizations
Future Work

- As AlphaZ develops, we get
  - More transformations
  - More code generators including parallel and tiled ones

- With these developments, UNAfold can take advantage of
  - Exploring schedules
  - Exploring memory maps
  - Exploring processor allocations
  - Exploring optimal tile sizes
Optimizations

- **Derivation**
  - Split $E_B$ term into constituents.

  \[
  QBI(i, j) = \min_{4 \leq d \leq j - i - 3, i < i' < j - d} \left\{ \begin{array}{l}
  \text{Asym}(i' - i - 1, j - i' - d - 1) \\
  + S_p(j - i - d - 2) \\
  + E_S(i, j) \\
  + E_S(i', i' + d) \\
  + Q'(i', i' + d)
  \end{array} \right. 
  \]

  - Factor out the $E_S$ term.

  \[
  QBI(i, j) = E_S(i, j) + \min_{4 \leq d \leq j - i - 3, i < i' < j - d} \left\{ \begin{array}{l}
  \text{Asym}(i' - i - 1, j - i' - d - 1) \\
  + S_p(j - i - d - 2) \\
  + E_S(i', i' + d) \\
  + Q'(i', i' + d)
  \end{array} \right. 
  \]
Optimizations

- Decompose the double reduction.

\[ QBI(i, j) = E_S(i, j) + \min_{4 \leq d \leq j-i-3} \{X(i, j, d)\} \]

where

\[ X(i, j, d) = \min_{i < i' < j-d} \left\{ \begin{array}{c}
Asym(i' - i - 1, j - i' - d - 1) \\
+ S_P(j - i - d - 2) \\
+ E_S(i', i' + d) \\
+ Q'(i', i' + d)
\end{array} \right\} \]

- Factor out the \( S_P \) term.

\[ X(i, j, d) = S_P(j - i - d - 2) + \min_{i < i' < j-d} \left\{ \begin{array}{c}
Asym(i' - i - 1, j - i' - d - 1) \\
+ E_S(i', i' + d) \\
+ Q'(i', i' + d)
\end{array} \right\} \]
Optimizations

- Replace reduction term by $Y$.

$$X(i, j, d) = S_P(j - i - d - 2) + Y(i, j, d)$$

where

$$Y(i, j, d) = \min_{i < i' < j - d} \left\{ \begin{array}{l}
\text{Asym}(i' - i - 1, j - i' - d - 1) \\
+ E_S(i', i' + d) \\
+ Q'(i', i' + d)
\end{array} \right\}$$

- Scan in $Y$ along $(i+j)$.

$$Y(i, j, d) = \min \left\{ \begin{array}{l}
Y(i + 1, j - 1, d) \\
\min_{i+1 < i' < j - d - 1} \left\{ \begin{array}{l}
\text{Asym}(i' - i - 1, j - i' - d - 1) \\
+ E_S(i', i' + d) \\
+ Q'(i', i' + d)
\end{array} \right\}
\end{array} \right\}$$
Challenges faced

- Determining exact domains for the equations
- Memory indexing issues
  - Original code: \((i-1) \times (N-1) + (j-1)\)
  - Generated code: \(i \times N + j\)
- No Pre-processor directives
  - Work around: Use external functions to return the values defined using pre-processor directive \#define
- Compilation problems
  - Conflicting macros between the re-use code and generated code