GPU Parallelization of Algebraic Dynamic Programming

Peter Steffen[1], Robert Giegerich[1] and Mathieu Giraud[2]

[1]Bielefeld University, Faculty of Technology, Germany
[2]CNRS, LIFL, Université Lille 1, France

April 28, 2010

Topic of this talk

- We have developed a generic approach to Dynamic Programming: *Algebraic Dynamic Programming* (ADP)
- The ADP compiler automatically generates C code for ADP algorithms
- Our new result is the extension of the ADP compiler, such that it generates CUDA code for Nvidia graphic cards

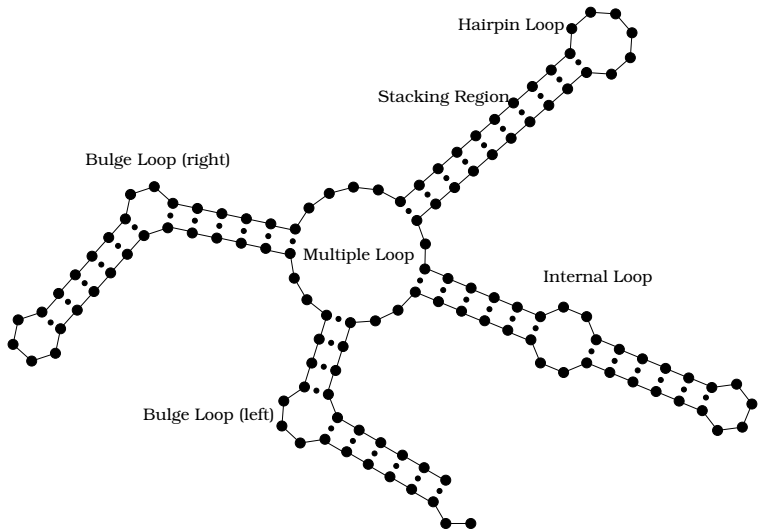
Dynamic Programming (DP)

Dynamic Programming (DP) is useful in

- Sequence comparison

```
da_rling   da_rlin_g
_airline   _airline_
-------------------
DRIRRRRR   DRIRRRRID
```

- RNA secondary structure prediction



- DP evaluates exponential search space in polynomial runtime
- many more applications, also beyond biosequence analysis

DP matrix recurrences

DP matrix recurrence for a local alignment:

$$
\begin{aligned}
&\mathit{alignment}_{i,j} = \max(\\
&\quad [0 | j - i \geq 0] \mathbin{++} \\
&\quad [\text{if } z_{i+1} == z_j \text{ then } \mathit{alignment}_{i+1,j-1} + 4 \\
&\qquad \text{else } \mathit{alignment}_{i+1,j-1} - 3 | j - i \geq 2] \mathbin{++} \\
&\quad [\mathit{xDel}_{i+1,j} - 16 | j - i \geq 1] \mathbin{++} \\
&\quad [\mathit{xIns}_{i,j-1} - 16 | j - i \geq 1])
\end{aligned}
$$

Typical DP recurrences are

- difficult to find and justify
- difficult to re-use
- nearly impossible to debug

Algebraic Dynamic Programming (ADP)

- a declarative method of Dynamic Programming over sequence data
- developed since 2000 by Robert Giegerich, Dirk Evers, Carsten Meyer, Peter Steffen, and others
- used in bioinformatics tools *pknotsRG*(2003), *RNAshapes*(2004), *RNAhybrid*(2004), *RNAcast*(2005), *Locomotif*(2006)

- Giegerich, R. and Meyer, C. and Steffen, P.: A Discipline of Dynamic Programming over Sequence Data in Science of Computer Programming, 51(3) , Pages:215-263, 2004
- Reeder, Jens and Giegerich, Robert: Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics in BMC Bioinformatics, 5(104) , 2004
- Rehmsmeier, M. and Steffen, P. and Höchsmann, M. and Giegerich, R.: Fast and effective prediction of microRNA/target duplexes in RNA, 10, Pages:1507-1517, 2004
- Giegerich, R. and Voss, B. and Rehmsmeier, M.: Abstract Shapes of RNA in Nucleic Acids Res., 32(16), Pages:4843-4851, 2004
- Reeder, Jens and Giegerich, Robert: Consensus shapes: an alternative to the Sankoff algorithm for RNA consensus structure prediction in Bioinformatics, 21(17) , Pages:3516-3523, 2005
- Voss, Björn and Giegerich, Robert and Rehmsmeier, Marc: Complete probabilistic analysis of RNA shapes in BMC Biology, 4(5), 2006
- Steffen, P. and Voss, B. and Rehmsmeier, M. and Reeder, J. and Giegerich, R.: RNAshapes: an integrated RNA analysis package based on abstract shapes in Bioinformatics, 22(4) , Pages:500-503, 2006

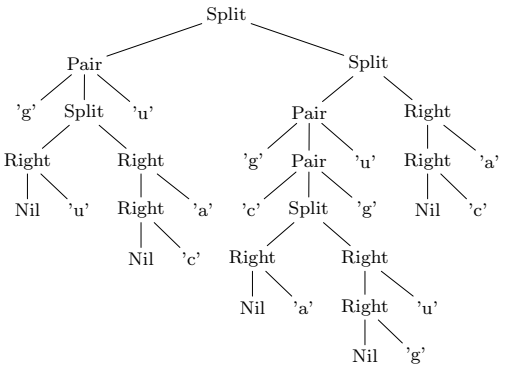
Algebraic Dynamic Programming (ADP)

Example: a Nussinov type RNA secondary structure prediction
The specification of an ADP algorithm consists of four constituents:

- *Alphabet:* The input RNA sequence is a string over the alphabet $\mathcal{A} = \{a, c, g, u\}$.
- *Search space:* Given an input sequence $w \in \mathcal{A}^*$, the search space is the set of all possible secondary structures the sequence w can form.

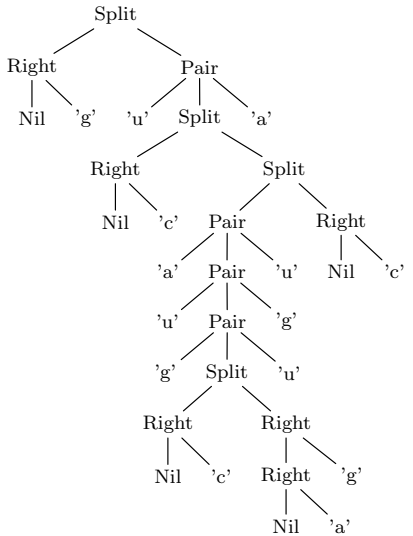
gucaugcaguguca
(...)((...))..

$t_1 =$



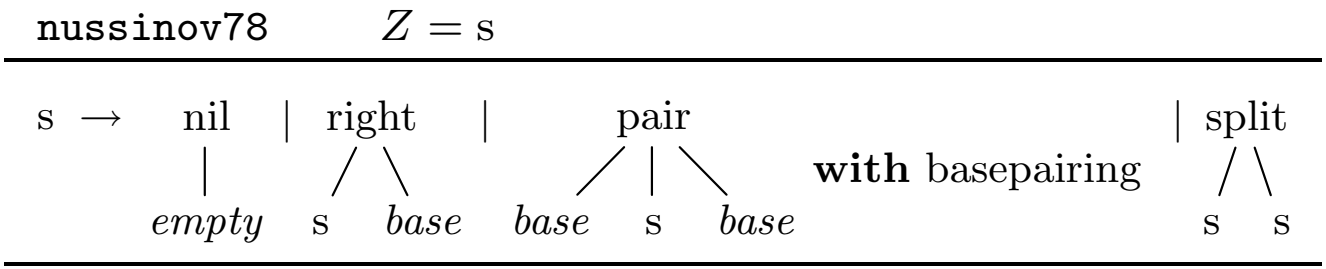
gucaugcaguguca
.(.(((...))).)

$t_2 =$



Algebraic Dynamic Programming (ADP)

The search space is described by a *tree grammar*:



The number of candidates is exponential in the length of the input sequence.

Algebraic Dynamic Programming (ADP)

- *Scoring:* Given an element of the search space as a tree, we need to score this element. Here, we are only interested in counting base pairs. So, we assign a score for every candidate.

```
bpmax = (nil, right, pair, split, h) where
nil(s)          =  0
right(s,b)      =  s
pair(a,s,b)     =  s + 1
split(s,s')     =  s + s'
h([])           =  []
h([s_1,...,s_r]) =  [max_{1≤i≤r} s_i]
```

- *Objective:* We need to choose one or several solutions from the pool of candidates. For this purpose, we add an objective function h which chooses one or more elements from a list of candidate scores.
- Scoring schemes with objective functions are called *evaluation algebras* in ADP.

Algebraic Dynamic Programming (ADP)

- *Scoring:* Given an element of the search space as a tree, we need to score this element. Here, we are only interested in counting base pairs. So, we assign a score for every candidate.

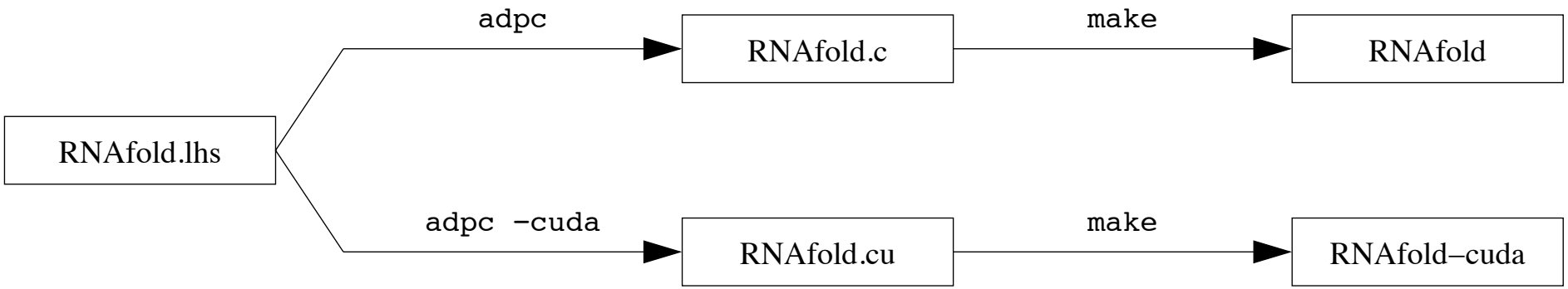
```
bpmax = (nil, right, pair, split, h) where
nil(s)          =  0
right(s,b)      =  s
pair(a,s,b)     =  s + 1
split(s,s')     =  s + s'
h([])           =  []
h([s_1,...,s_r]) =  [max_{1≤i≤r} s_i]
```

- *Objective:* We need to choose one or several solutions from the pool of candidates. For this purpose, we add an objective function h which chooses one or more elements from a list of candidate scores.
- Scoring schemes with objective functions are called *evaluation algebras* in ADP.

RNAfold – Complete grammar

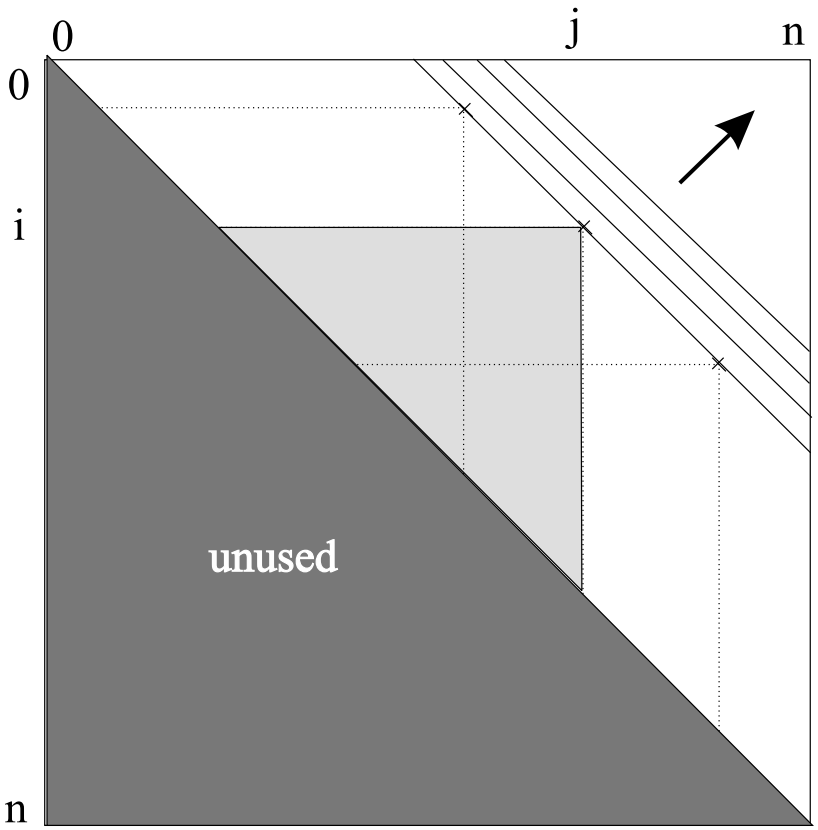
```
rnafold alg f = axiom struct where
   (sadd,cadd,is,sr,hl,bl,br, il, il11, il12, il21, il22,
    dl, dr, dlr, edl, edr, edlr, drem, cons, ul, pul, addss, ssadd, nil, combine, h) = alg

   struct       = tabulated (
                     sadd <<< base    ~~~ struct |||
                     cadd  <<< initstem ~~~ struct |||
                     nil  <<< empty ... h)

   initstem = tabulated (is <<< loc ~~~ closed ~~~ loc ... h)
   closed   = tabulated (
                    stack ||| ((hairpin ||| leftB   ||| rightB ||| iloop  ||| multiloop) `with` stackpairing)

   stack     = (sr   <<< base ~~~ closed ~~~ base)  `with` basepairing ... h
   hairpin   = hl   <<< base ~~~ base ~~~ (region `with` (minsize 3))        ~~~ base ~~~ base ... h
   leftB     = bl   <<< base ~~~ base ~~~ region   ~~~ initstem   ~~~ base ~~~ base ... h
   rightB    = br   <<< base ~~~ base ~~~ initstem ~~~ region     ~~~ base ~~~ base ... h
   iloop     = il   <<< base ~~~ base ~~~ (region  `with` (maxsize 30)) ~~~ closed ~~~
                                          (region  `with` (maxsize 30)) ~~~ base ~~~ base ... h
   comps     = tabulated (
                    cons  <<< block ~~~ comps    |||
                    block                        |||
                    addss <<< block ~~~ region   ... h)

   block    = tabulated (
                   ul      <<<            initstem    |||
                   ssadd   <<< region ~~~ initstem    ... h)
```

ADP Compiler



- The ADP compiler translates ADP algorithms into C
- We have developed an extension to the compiler, that automatically generates CUDA code for NVIDIA graphic cards

RNAfold – Parallelization



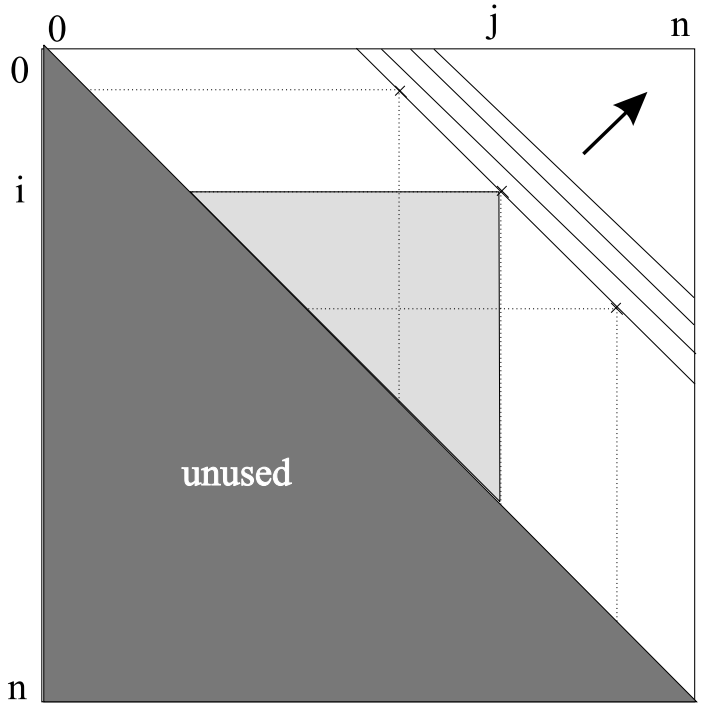
- All elements (i, j) on the same diagonal are independent: one thread per element
- The element (i, j) needs the $O((j - i)^2)$ elements in the underlying triangle.
- This is generic to all ADP programs (results are combined from results of shorter subsequences)

RNAfold – CUDA code

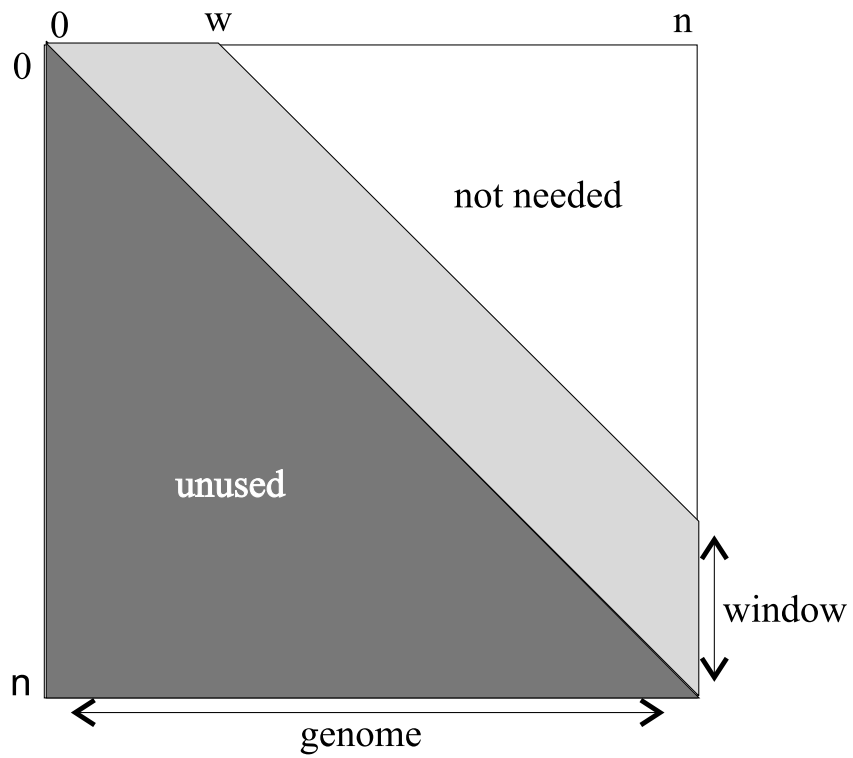
```
__global__ static void calc_all(int diag, int n) {
   int i = blockIdx.x*blockDim.x+threadIdx.x;
   int j = i + diag;
   if ((i <= n) && (j <= n)) {
      calc_closed(i, j);
      calc_initstem(i, j);
      calc_struct(i, j);
      calc_block(i, j);
      calc_comps(i, j);
   }
}

static void mainloop(){
   for (int diag=0; diag<=n; diag++) {
      (...)
      calc_all <<< grid, threads >>> (diag, n);
   }
}
```

RNAfold – Window mode

- n can be very large (genome),
- but RNA folds are only on a few tens/hundred bases.



$O(n^2)$ memory / $O(n^3)$ time



$O(nw)$ memory / $O(nw^2)$ time

Results – ADP + CUDA (2009)

Tests on C. Carsonella ruddii, $n = 160$ kbp (pknotsRG: $n = 20$ kbp)

Grammar, window size, time complexity			Xeon 3.0 GHz (1 core) + Nvidia GTX 280		
			CPU	GPU	speedup
RNAfold-bp.lhs	`-w 80`	$O(w^2 n)$	133.77	5.18	25.8×
RNAfold.lhs	`-w 80`	$O(w^2 n)$	35.57	3.59	9.9×
tRNA-matcher.lhs	`-w 100`	$O(w^2 n)$	43.60	3.01	14.5×
pknotRG.lhs	`-w 80`	$O(w^3 n)$	23.54	3.25	7.2×
pknotRG.lhs	`-w 160`	$O(w^3 n)$	166.27	27.22	6.1×

- RNAfold: divergence (large computations for only 6/16 threads)
  [Rizk, Lavenier 09]: speedup of 17×
- RNAfold-bp: toy computation, no divergence

Results – ADP + CUDA (2009)

Tests on C. Carsonella ruddii, $n = 160$ kbp (pknotsRG: $n = 20$ kbp)

Grammar, window size, time complexity			Xeon 3.0 GHz (1 core) + Nvidia GTX 280		
			CPU	GPU	speedup
RNAfold-bp.lhs	`-w 80`	$O(w^2n)$	133.77	5.18	25.8×
RNAfold.lhs	`-w 80`	$O(w^2n)$	35.57	3.59	9.9×
tRNA-matcher.lhs	`-w 100`	$O(w^2n)$	43.60	3.01	14.5×
pknotRG.lhs	`-w 80`	$O(w^3n)$	23.54	3.25	7.2×
pknotRG.lhs	`-w 160`	$O(w^3n)$	166.27	27.22	6.1×

- RNAfold: divergence (large computations for only 6/16 threads)
  [Rizk, Lavenier 09]: speedup of 17×
- RNAfold-bp: toy computation, no divergence

Results – ADP + CUDA (2009)

Tests on C. Carsonella ruddii, $n = 160$ kbp (pknotsRG: $n = 20$ kbp)

Grammar, window size, time complexity			Xeon 3.0 GHz (1 core) + Nvidia GTX 280		
			CPU	GPU	speedup
RNAfold-bp.lhs	`-w 80`	$O(w^2 n)$	133.77	5.18	25.8×
RNAfold.lhs	`-w 80`	$O(w^2 n)$	35.57	3.59	9.9×
tRNA-matcher.lhs	`-w 100`	$O(w^2 n)$	43.60	3.01	14.5×
pknotRG.lhs	`-w 80`	$O(w^3 n)$	23.54	3.25	7.2×
pknotRG.lhs	`-w 160`	$O(w^3 n)$	166.27	27.22	6.1×

- RNAfold: divergence (large computations for only 6/16 threads)
  [Rizk, Lavenier 09]: speedup of 17×
- RNAfold-bp: toy computation, no divergence

Preliminary Results – ADP + OpenCL (April 2010)

Tests on C. Carsonella ruddii, $n = 160$ kbp

Grammar	Xeon 2.6 GHz				
		+ Nvidia SDK		+ ATI/AMD SDK	
	CPU	CUDA 285 GTX	OpenCL 285 GTX	OpenCL CPU	OpenCL HD 4890
RNAfold-bp	90.85	7.95	10.66	36.24	16.41
RNAfold	35.57	5.30	9.9	12.06	18.67

- same OpenCL code for NVIDIA and ATI/AMD SDKs
- with ATI/AMD SDK: better than regular C code, even without GPU...
- on NVIDIA: OpenCL a little slower than CUDA
- on AMD: we should explore other optimization techniques

Conclusion

- We implemented a parallel GPU CUDA backend for the ADP compiler, which works out-of-the-box for several grammars dealing with RNA sequences
- Our approach is generic and requires few efforts to the user, even if the speedups are not the best ones that could be obtained by manually optimized implementations

Outlook

- *Shared/local memory*.
  - Difficult to automatically deduce from ADP grammar
  - Generate from hints in the grammar?
- *Static evaluation of grammars.*
  - Test other grammars (bioinformatics, other domains)
  - Which grammars are efficient to parallelize, and why?
- *Other targets.*
  - OpenCL, AMD/ATI cards, multicore CPU...
  - ADP: generic methodology, portable solutions

More information

ADP website:

`http://bibiserv.techfak.uni-bielefeld.de/adp`

ADP CUDA website:

`http://bibiserv.techfak.uni-bielefeld.de/adp/cuda.html`