

PhD Completion Seminar
Algorithms for the Study of RNA and Protein
Structure

Alex Stivala
University of Melbourne, CSSE
Supervisors: Prof. Peter Stuckey, Dr Tony Wirth

August 27, 2010

Overview

- ▶ RNA structural alignment
- ▶ Parallelism 1: Parallel dynamic programming
- ▶ Protein substructure searching
- ▶ Parallelism 2: Faster protein substructure searching
- ▶ Protein structure cartoons

RNA structural alignment

Introduction: Dynamic Programming

- ▶ Optimization problems where optimum can be computed efficiently from optimal solutions to subproblems (Bellman 1957)
- ▶ Bottom-up
- ▶ Top-down and memoization

Introduction: RNA secondary structure

- ▶ RNA has a secondary structure determined by basepairing interactions within the RNA molecule.
- ▶ This can be predicted from sequence by a D.P. algorithm such as that implemented in `RNAfold` in the Vienna RNA package (Zuker & Stiegler (1981); Hofacker *et al.* (1994)).
- ▶ Rather than one single “best” structure, a base pairing probability matrix (McCaskill (1990)) can be computed, showing probabilities of bases pairing with each other.

RNA structural alignment

D.P. from Hofacker et al. (2004):

$$S(i, j, k, l) = \max \begin{cases} S(i+1, j, k, l) + \gamma, \\ S(i, j, k+1, l) + \gamma, \\ S(i+1, j, k+1, l) + \sigma(A_i, B_k), \\ \max_{h \leq j, q \leq l} S^M(i, h, k, q) + S(h+1, j, q+1, l) \end{cases}$$

$$S^M(i, j, k, l) = S(i+1, j-1, k+1, l-1) + \psi_{ij}^A + \psi_{kl}^B + \tau(A_i, A_j, B_k, B_l)$$

Parallel dynamic programming

Previous work on parallelizing D.P.

- ▶ Used the bottom-up approach
- ▶ Analyze data dependencies and compute independent cells in parallel
- ▶ So only works on problems where it is feasible to apply bottom-up, i.e. compute every single subproblem
- ▶ And requires careful analysis of data dependencies in each particular D.P. for parallelization

Overview of our approach

Parallelize any D.P. on a multicore processor with shared memory.

- ▶ Each thread computes the entire problem.
- ▶ The results are shared via a lock-free hash table in shared memory.
- ▶ Randomization of subproblem ordering causes divergence of thread computations.

General form of a D.P.

$$f(\bar{x}) = \begin{array}{l} \text{if } b(\bar{x}) \text{ then } g(\bar{x}) \\ \text{else } F(f(\bar{x}_1), \dots, f(\bar{x}_n)) \end{array}$$

where

- ▶ $b(\bar{x})$ holds for the base cases
- ▶ $g(\bar{x})$ is the result for base cases,
- ▶ F is a function combining the optimal answers to a number of sub-problems $\bar{x}_1, \dots, \bar{x}_n$.

Computing the D.P.: serial version

```
f( $\bar{x}$ )  
   $v \leftarrow \text{lookup}(\bar{x})$   
  if  $v \neq \text{KEY\_NOT\_FOUND}$   
    return  $v$   
  if  $b(\bar{x})$  then  $v \leftarrow g(\bar{x})$   
  else  
    for  $i \in 1..n$   
       $v[i] \leftarrow f(\bar{x}_i)$   
     $v \leftarrow F(v[1], \dots, v[n])$   
   $\text{insert}(\bar{x}, v)$   
  return  $v$ 
```

Computing the D.P.: parallel version

```
f( $\bar{x}$ )  
   $v \leftarrow \text{par\_lookup}(\bar{x})$   
  if  $v \neq \text{KEY\_NOT\_FOUND}$   
    return  $v$   
  if  $b(\bar{x})$  then  $v \leftarrow g(\bar{x})$   
  else  
    for  $i \in 1..n$  in random order  
       $v[i] \leftarrow f(\bar{x}_i)$   
     $v \leftarrow F(v[1], \dots, v[n])$   
  par_insert( $\bar{x}, v$ )  
  return  $v$ 
```

The D.P.s we use to demonstrate our method

- ▶ knapsack
- ▶ shortest paths
- ▶ RNA structural alignment (we are familiar with this already)
- ▶ open stacks (skipping this here)

For the D.P. experiments, we used the open addressing hash table.

Application to knapsack

$$k(i, w) = \begin{cases} 0 & \text{if } i = 0 \\ k(i-1, w) & \text{if } w < w_i \\ \max\{k(i-1, w), \\ \quad k(i-1, w - w_i) + p_i\} & \text{otherwise} \end{cases}$$

There are two possible orderings of the two subproblems in last case. Each thread chooses one of the two, with equal probability.

Shortest paths

Floyd-Warshall algorithm: $s(i, j, k)$ is the length of the shortest path from node i to node j using only $1, \dots, k$ as intermediate nodes.

$$s(i, j, k) = \begin{cases} 0, & \text{if } i = j \\ d_{ij}, & \text{if } k = 0 \\ \min\{s(i, j, k - 1), & \text{otherwise} \\ \quad s(i, k, k - 1) + s(k, j, k - 1)\} \end{cases}$$

where d_{ij} is the weight (distance) of the directed edge from i .
The randomization chooses, with equal probability, between the $3! = 6$ orderings of the subproblems.

RNA structural alignment (revisited)

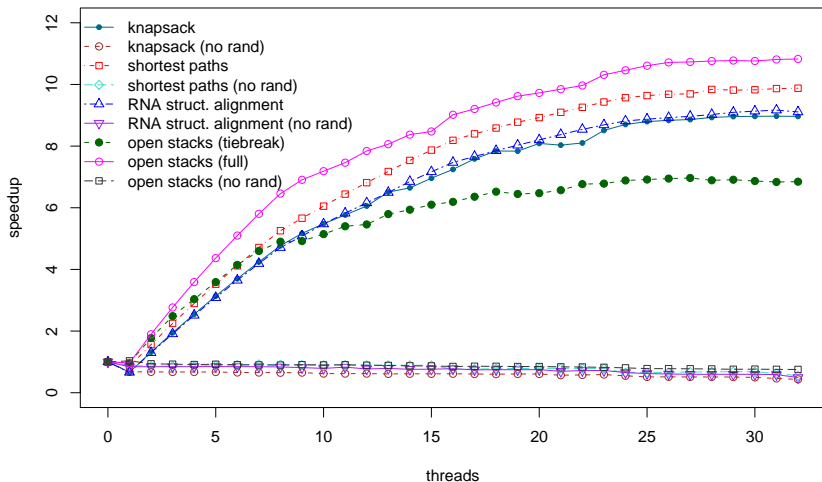
D.P. from Hofacker et al. (2004):

$$S(i, j, k, l) = \max \begin{cases} S(i+1, j, k, l) + \gamma, \\ S(i, j, k+1, l) + \gamma, \\ S(i+1, j, k+1, l) + \sigma(A_i, B_k), \\ \max_{h \leq j, q \leq l} S^M(i, h, k, q) + S(h+1, j, q+1, l) \end{cases}$$

$$S^M(i, j, k, l) = S(i+1, j-1, k+1, l-1) + \Psi_{ij}^A + \Psi_{kl}^B + \tau(A_i, A_j, B_k, B_l)$$

To randomize, we take a random permutation of the list of all subproblems in the $S(\cdot)$ and $S^M(\cdot)$ computation.

UltraSPARC T1 results (1)



UltraSPARC T1 results (2)

problem	speedup	threads
knapsack	8.97	31
shortest paths	9.88	32
RNA struct. alignment	9.17	31
open stacks (tiebreak)	6.96	27
open stacks (full)	10.83	32

Conclusions (parallel dynamic programming)

- ▶ We demonstrated a 10x speedup (for 32 threads) on the open stacks D.P.
- ▶ Applicable to any dynamic program
- ▶ Including very large ones where bottom-up impractical
- ▶ Can also be applied to smaller problems using array not hash table
- ▶ No need to analyze data dependencies etc. for vectorization
- ▶ Only need to consider how to randomize subproblem ordering: generally trivial but “more random” is better

Protein substructure searching

(the short version — no equations)

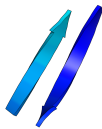
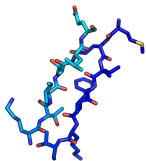
Introduction

- ▶ We want to search for occurrence of substructures in a database of structures.
- ▶ This is important to understand protein structure and function.
- ▶ Structural matching is a computationally challenging problem.

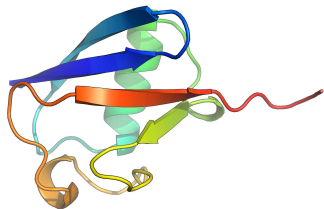
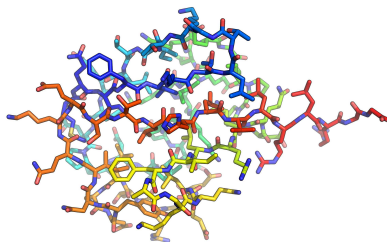
Protein structure in 3 easy steps (1): Primary

```
>1UBI : A | PDBID | CHAIN | SEQUENCE  
MQIFVKTTLTGKTITLEVEPSDTIENVKAKIQDKEGIPPDQQRL  
IFAGKQLEDGRTLSDYNIQKESTLHLVLRRLGG
```

Protein structure in 3 easy steps (2): Secondary



Protein structure in 3 easy steps (3): Tertiary

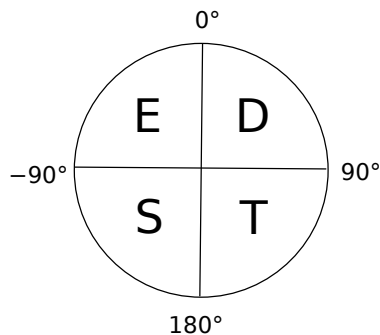
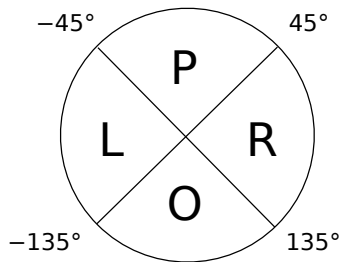


Existing methods

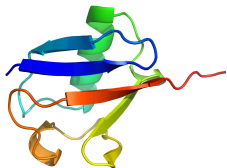
- ▶ There are lots; I'm not going to go into details here.
- ▶ Basically two classes:
 1. Residue or atom level structural alignment;
 2. SSE matching
- ▶ Class 1 methods are slow as they work at a detailed level so have large amounts of data;
- ▶ Class 2 methods are slow as they tend to require solving computationally hard problems (but on much smaller amounts of data than class 1).
- ▶ Lots of heuristic methods to get faster approximations.

- ▶ Konagurthu, Stuckey, Lesk (2008) *Bioinformatics* 24(5):645–651.
- ▶ This work is an extension of the work cited above.

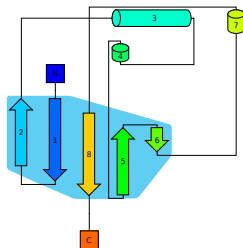
Tableau encoding



β -grasp query structure and tableau



(a)



(b)

0.000								
2.654	0.000							
-1.173	2.147	1.000						
0.389	-2.757	1.352	3.000					
2.040	-1.438	2.079	-1.651	0.000				
-1.258	1.556	-1.108	-1.647	2.985	0.000			
1.693	-1.808	-1.851	1.309	-0.370	-2.913	3.000		
-0.590	2.102	-1.231	0.959	2.566	-0.721	2.264	0.000	

(c)

e
OT e
LE RT xa
PD OS RD xg
RT LE RT LS e
LE RD LE LS OT e
RT LS LS RD PE OS xg
PE RT LE RD OT PE RT e

(d)

Maximally-similar subtableaux

- ▶ Finding a substructure within a structure is to find maximally-similar subtableaux in the two tableaux.
- ▶ This problem can be formulated as a quadratic integer program (QIP).
- ▶ This problem is equivalent to the quadratic assignment problem and thus NP-hard.

TableauSearch

- ▶ TableauSearch is a method of approximating the optimum matching score.
- ▶ It uses dynamic programming (DP) in an alignment-like approach.
- ▶ It is extremely fast; e.g. searching 75632 ASTRAL domains in 66s (Konagurthu *et al.* (2008)).
- ▶ But it is only an approximation to the score, and can miss matches.
- ▶ More importantly, it is a global alignment, giving high scores to similar structures - we cannot use it to search for a substructure in a larger structure.
- ▶ It also depends on SSE ordering (sequence) being maintained
- ▶ IR Tableau (Zhang *et al.* 2010) is an even faster method.

MNAligner

- ▶ Li *et al.* 2007 “Alignment of molecular networks by integer quadratic programming” *Bioinformatics* 23(13):1631-1639
- ▶ The formulation of this problem as a QIP in this paper is strikingly similar to the tableau matching QIP.
- ▶ But the authors note that the constraints are totally unimodular.
- ▶ This means that the QIP can be relaxed to a QP which will have an integer solution (under certain conditions)

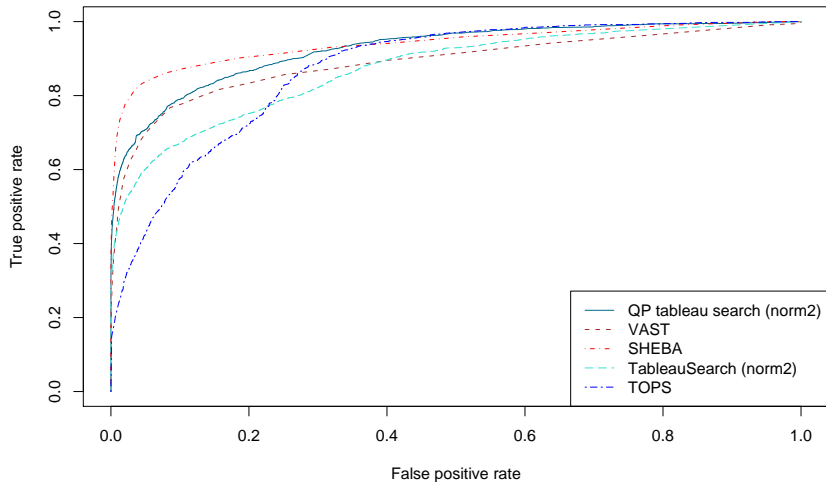
QP tableau search

- ▶ So by relaxing the QIP to a QP as in MNAligner, we can much more efficiently solve it. This is essentially what our QP tableau search method does.
- ▶ We also need to relax some of the constraints and penalize their violation in the objective function instead.
- ▶ I am leaving out the details of the formulation and solution by interior point method for this short talk.

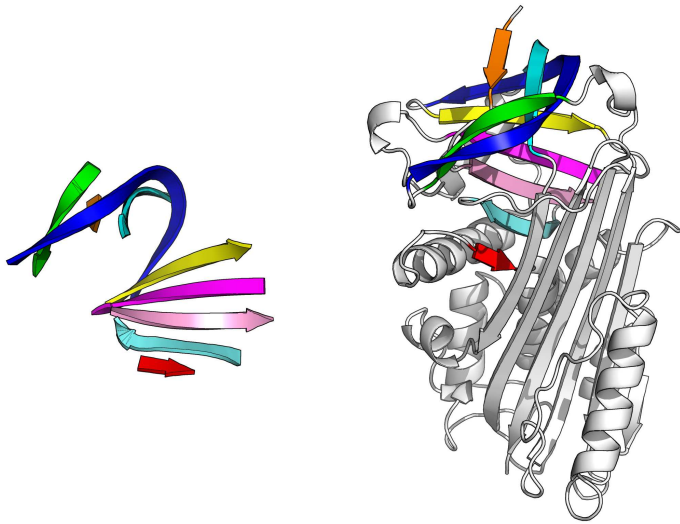
Evaluation

- ▶ We ran queries on the ASTRAL SCOP 1.73 95% sequence identity non-redundant subset (15273 domains).
- ▶ For assessing whole structure matches, we used a randomly chosen set of 200 query structures.
- ▶ For substructure matches, we must make some more manual comparisons with other methods. We will just show an example here.
- ▶ Similarly for non-linear matches.

ROC curves in 200 query set in ASTRAL SCOP 95% subset



Serpin B/C sheet substructure (left) found in 1MTP (right)



Conclusions (QP Tableau Search)

- ▶ Relaxing the QIP tableau matching algorithm to real QP and solving by interior point method is a sensitive and efficient method to locate occurrences of a query substructure in a structure database.
- ▶ Although slower than the DP TableauSearch or IR Tableau, this technique is much faster than solving the QIP or ILP directly with CPLEX and (in common with the latter methods):
 - ▶ can handle substructure queries;
 - ▶ can remove the ordering constraint to find non-sequential alignments;
 - ▶ can return the SSE matches, not just a score.
- ▶ If only we could do it even faster...

Faster protein substructure searching

To try to maximize the same objective function as before, but much faster, we will:

- ▶ use simulated annealing
- ▶ create a parallel implementation on a graphics processing unit (GPU)

State of the system

Pairwise comparison of structure A with N_A SSEs and structure B with N_B SSEs, represented by tableaux T_A and T_B

The state of the matching is vector v , where

$$v_i = j, 1 \leq i \leq N_A, 0 \leq j \leq N_B$$

means that the i th SSE in structure A is matched with the j th SSE in structure B, or with no SSE if $v_i = 0$.

Initial state

Each SSE in structure A is matched with the first SSE of the same type (helix, strand) in structure B, with probability p_m .

Move to neighbor state

- ▶ An SSE i is chosen uniformly at random in structure A and its mapping changed to a random SSE j of the same type in structure B, (optionally) without violating the non-crossing constraint.
- ▶ If there is no such j , then set $v_i = 0$ i.e. that SSE is no longer matched.

Simulated annealing schedule

- ▶ a move is accepted if objective function is better, or if $\exp\left(\frac{g(v')-g(v)}{T}\right) > p$ where p is a random number in $[0, 1]$
- ▶ $T \leftarrow \alpha T$ for the next iteration
- ▶ repeat until iteration limit (100 iterations) reached

Slightly different from “vanilla” simulated annealing:

- ▶ The best state ever found is remembered, not necessarily the final state.
- ▶ The whole schedule is run M times, we call each a *restart*.
- ▶ The answer is the best objective value (and its state) ever found in any of the restarts.

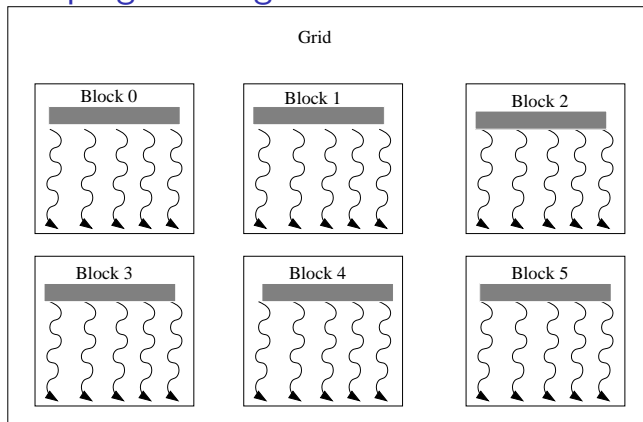
Graphics Programming Units

- ▶ Originally for graphics rendering, now “GPGPU” (general purpose GPU) also used for other parallel algorithms.
- ▶ NVIDIA proprietary CUDA: extensions to C for GPGPU programming
- ▶ OpenCL is a standard for doing the same thing
- ▶ Data-parallelism: basically SIMD - very many threads running the same code simultaneously on different data locations.
- ▶ “co-processing” — host CPU has to send data to GPU and then get results back from GPU.
- ▶ Need to use a large number (thousands) of threads to get good results

GPU Memory hierarchy


- ▶ Large, high latency global memory
- ▶ Small (e.g. 16 KB) fast shared memory
- ▶ Small fast constant (readonly) memory
- ▶ Thread local memory: this “local” memory is slow like global!
- ▶ No cache: the programmer manages memory hierarchy.

CUDA programming model



Key:

 Shared memory

 Thread

Parallelization using CUDA (1)

- ▶ The query data is stored in the constant memory, all threads can read it quickly
- ▶ The whole database of structures is stored in the large (slow) global memory
- ▶ Two levels of parallelism:
 - ▶ Each *block* of threads does a single pairwise comparison of the query to a structure in the database.
 - ▶ Each thread in the block runs the simulated annealing schedule, so we do the restarts in parallel.
- ▶ Each block of threads first *parallel copies* the database structure it is going to use into its shared memory for fast access.

Parallelization using CUDA (2)

- ▶ The only synchronization necessary is:
 - ▶ a barrier after loading structures into shared memory, so all threads in a block can safely use it
 - ▶ a barrier at the end of the restart to do the MAX reduction operation to find the best value of objective function over all threads in the block.
- ▶ If the number of restarts is larger than the number of threads in a block, threads in a block will have to run more restarts until done.
- ▶ If the number of database structures is larger than the number of blocks in the grid, the blocks will have to load and process more db structures until they are done.

Parallelization using CUDA (3)

- ▶ Structures too large for the shared or constant memory can still be handled, by leaving them in the global memory, but this is slower.
- ▶ The tableau representation is compact — means we don't have a problem with overhead communicating data to GPU (and individual structures are usually small enough for fast shared/constant memory).

NVIDIA hardware used in experiments

Tesla C1060 4 GB global memory, 30 multiprocessors, 240 cores, 1.30 GHz.

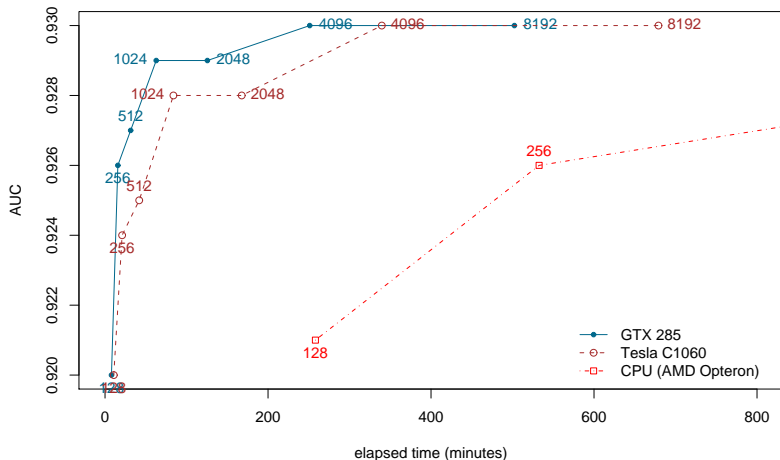
GTX 285 1 GB global memory, 30 multiprocessors, 240 cores, 1.48 GHz.

We used 128 threads/block and 128 blocks/grid (16384 threads total) for all experiments.

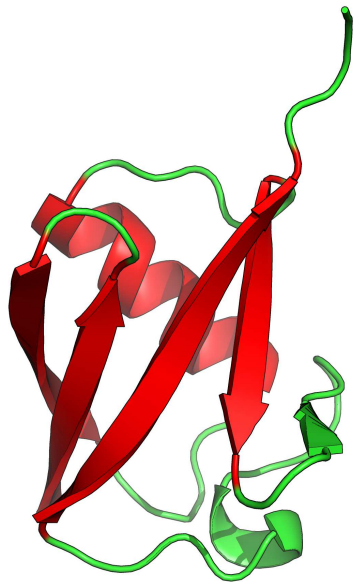
Results for ASTRAL 95% 200 query set

Method	Platform	Restarts	Elapsed time	AUC	standard	95% C. I.	
					error	lower	upper
SHEBA	CPU	-	25 h 22 m	0.931	0.004	0.924	0.938
SA Tableau Search	CPU	4096	142 h 42 m	0.930	0.004	0.923	0.937
SA Tableau Search	GTX 285	4096	4 h 11 m	0.930	0.004	0.923	0.937
SA Tableau Search	Tesla C1060	4096	5 h 40 m	0.930	0.004	0.923	0.937
DaliLite	CPU	-	620 h 03 m	0.929	0.004	0.922	0.936
SA Tableau Search	CPU	128	4 h 18 m	0.921	0.004	0.913	0.929
SA Tableau Search	GTX 285	128	0 h 08 m	0.920	0.004	0.912	0.927
SA Tableau Search	Tesla C1060	128	0 h 11 m	0.920	0.004	0.912	0.927
QP Tableau Search	CPU	-	157 h 51 m	0.914	0.004	0.906	0.922
LOCK2	CPU	-	208 h 09 m	0.912	0.004	0.904	0.920
TableauSearch	CPU	-	1 h 11 m	0.858	0.005	0.848	0.867
TOPS	CPU	-	1 h 11 m	0.855	0.005	0.845	0.864
VAST	CPU	-	14 h 26 m	0.855	0.005	0.846	0.865
IR Tableau	CPU	-	0 h 01 m	0.854	0.005	0.844	0.864
YAKUSA	CPU	-	4 h 14 m	0.827	0.005	0.816	0.837

AUC versus elapsed time on CPU and GPUs



β -grasp motif query



Motif query results

Query	Method	Platform	Restarts	Elapsed time	AUC	standard	95% C. I.	
						error	lower	upper
d1ubia_	SA	GTX 285	128	00 m 03 s	0.918	0.011	0.896	0.940
d1ubia_	SA	CPU	128	01 m 17 s	0.912	0.011	0.890	0.935
d1ubia_	QP	CPU	-	05 m 11 s	0.902	0.012	0.879	0.926
d1ubia_	TOPS	CPU	-	00 m 10 s	0.894	0.012	0.870	0.918
β -grasp	SA	CPU	128	01 m 11 s	0.939	0.010	0.920	0.958
β -grasp	QP	CPU	-	02 m 01 s	0.938	0.010	0.918	0.957
β -grasp	SA	GTX 285	128	00 m 03 s	0.934	0.010	0.914	0.954
β -grasp	TOPS	CPU	-	00 m 09 s	0.847	0.014	0.819	0.875
serpin B/C sheet	SA	GTX 285	128	00 m 03 s	0.993	0.013	0.968	1.019
serpin B/C sheet	SA	CPU	128	01 m 19 s	0.991	0.015	0.962	1.021
serpin B/C sheet	QP	CPU	-	08 m 16 s	0.986	0.019	0.949	1.023
serpin B/C sheet	TOPS	CPU	-	00 m 24 s	0.491	0.054	0.385	0.597

Conclusions (SA Tableau Search)

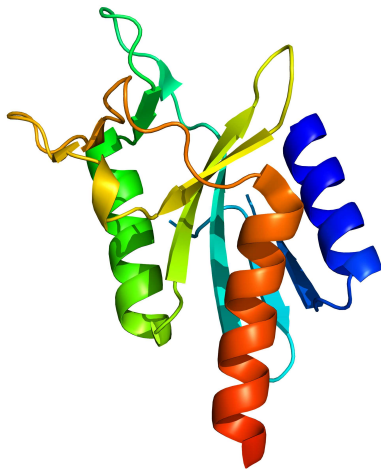
- ▶ Comparable in speed and accuracy with widely used methods
- ▶ Up to $34\times$ speedup on GPU over CPU
- ▶ Making it one of the fastest available methods
- ▶ The first use of a GPU for protein structure search
- ▶ If only there were a graphical interface for building motif queries...

Protein structure cartoons

Cartoons

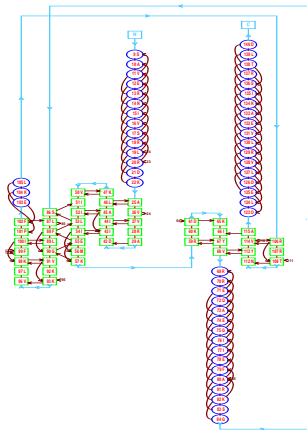
- ▶ hand-drawn
- ▶ HERA / ProMotif (Hutchinson and Thornton 1990, 1996)
- ▶ TOPS (Flores *et al.* 1994, Westhead *et al.* 1999)
- ▶ hand-drawn with TopDraw (Bond, 2003)
- ▶ PDBSum topology cartoons (Laskowski *et al.* 2006), based on HERA.

Example - 1HH1



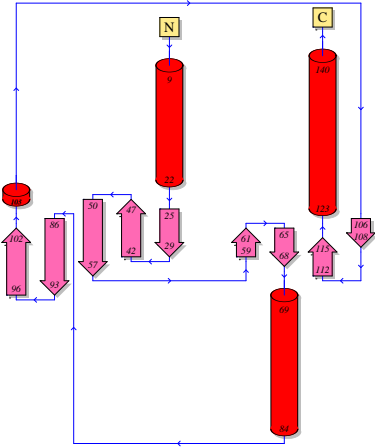
Holliday junction resolvase, PDB id 1HH1, Bond *et al.* 2001, image generated with PyMol

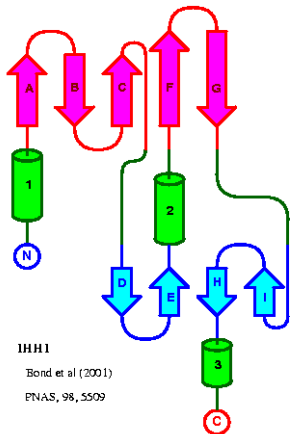
HERA / ProMotif (Hutchinson and Thornton 1996)



1HH1

PDBSum





IHH1
 Bond et al (2001)
 PNAS, 98, 5509

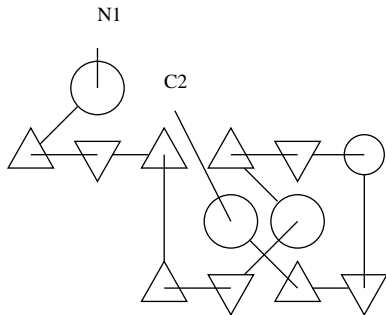


Figure: TOPS vs manually drawn with TopDraw

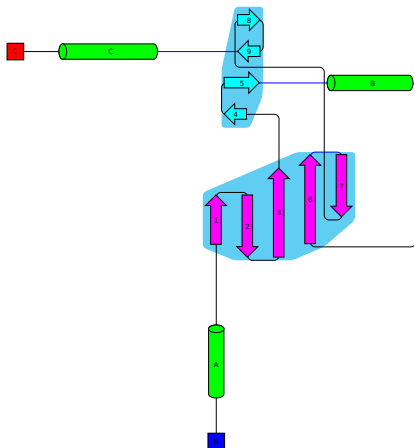
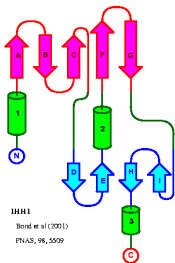
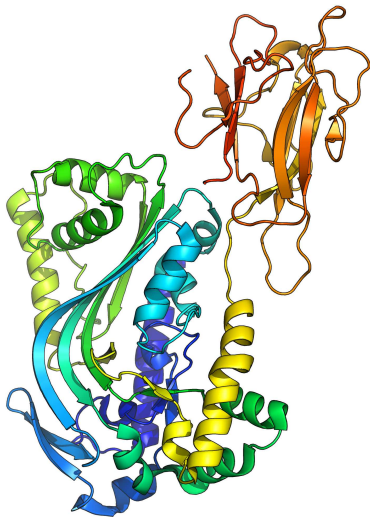


Figure: Pro-origami vs manually drawn with TopDraw

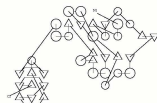
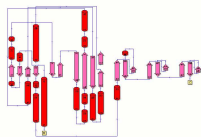
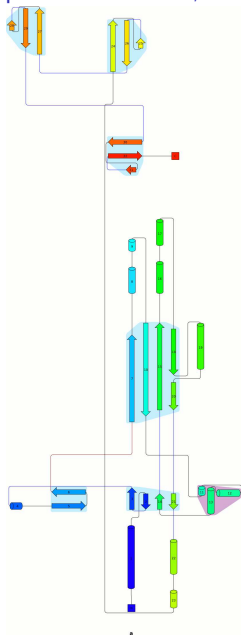
The essence of our approach

- ▶ Generate specifications of cartoon elements (SSEs)
- ▶ Generate connections and relationships between the cartoon elements as constraints
- ▶ Give this information to the constraint-based diagram editor Dunnart (*Wybrow et al.* 2006; *Dwyer et al.* 2009)
- ▶ Dunnart will lay out the diagram consistent with the constraints and can be used interactively to edit it, maintaining the constraints.

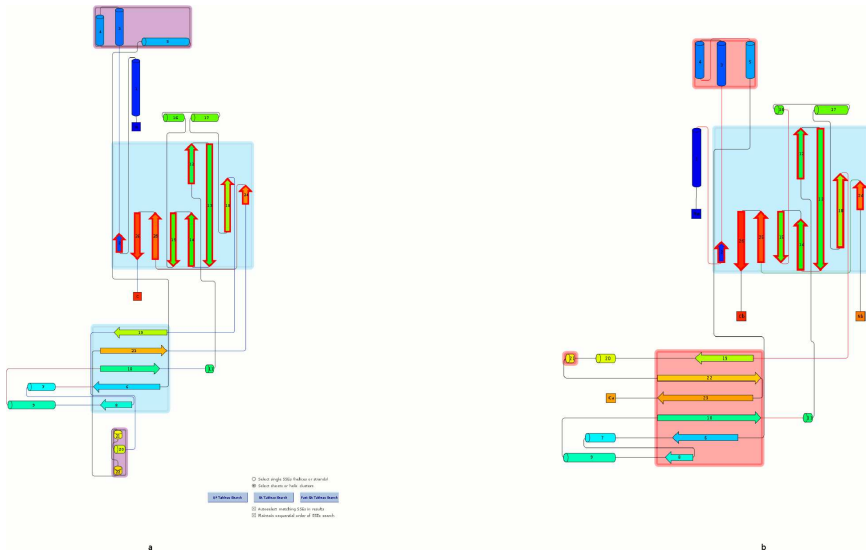
Plu-MACPF, PDB id 2QP2 (Rosado *et al.* 2007)



Comparison of TOPS, PDBSum and Pro-origami for Plu-MACPF



Using Pro-origami to make structural motif query



Conclusions

- ▶ We found that bounding for the RNA structural alignment d.p. was not successful
- ▶ But we demonstrated a $9\times$ speedup (32 threads) by parallelizing that d.p.,
- ▶ as a specific example of a completely general method for parallelizing *any* d.p., without regard to its specific properties.
- ▶ We developed two new algorithms for tableau-based protein structure/motif search,
- ▶ and a parallel implementation of one on a GPU — the first use of a GPU for the protein structure search problem, and one of the fastest available methods.
- ▶ We developed a system for automatically generating protein structure cartoons by means of constraint-based diagrams,
- ▶ and its use as a substructure query building interface to our protein substructure search algorithms.

Acknowledgments

- ▶ Prof. Peter Stuckey
- ▶ Dr Tony Wirth
- ▶ Dr Linda Stern
- ▶ A/Prof. Maria Garcia de la Banda
- ▶ Prof. Manuel Hermenegildo
- ▶ Dr Michael Wybrow
- ▶ Prof. James Whisstock
- ▶ Dr Arun Konagurthu
- ▶ VPAC
- ▶ Tech. services.

Publications

- ▶ A. Stivala, A. Wirth, P.J. Stuckey, “Tableau-based protein substructure search using quadratic programming”, *BMC Bioinformatics* (2009), 10:153
- ▶ A. Stivala, P.J. Stuckey, M. Garcia de la Banda, M. Hermenegildo, A. Wirth, “Lock-free parallel dynamic programming”, *J. Parallel Distrib. Comput.* (2010), 70:839-848
- ▶ A. Stivala, P.J. Stuckey, A. Wirth, “Fast and accurate protein substructure searching with simulated annealing and GPUs”, submitted to *BMC Bioinformatics* (under review)
- ▶ A. Stivala, M.J. Wybrow, A. Wirth, J.C. Whisstock, P.J. Stuckey, “Automatic generation of protein structure cartoons with Pro-origami”, submitted to *Bioinformatics* (under review)

Source code, data sets, web server, etc.

- ▶ QP Tableau Search:
<http://www.csse.unimelb.edu.au/~astivala/qpprotein>
- ▶ SA Tableau Search:
<http://www.csse.unimelb.edu.au/~astivala/satabsearch>
- ▶ Pro-origami server:
<http://munk.csse.unimelb.edu.au/pro-origami>