Parallel Computational Biology Tools and Applications for Windows Clusters

Jaroslaw Pillardy

Cornell Theory Center, Cornell University
Ithaca, NY USA
CBSU is set up in the Cornell Theory Center and is a computational and bioinformatics research resource for the Tri-institutional Collaboration among Cornell University, Cornell/Weill Medical College, Rockefeller University, and Memorial Sloan-Kettering Cancer Center. CBSU is also a part of the Cornell Genomics Initiative.

Hardware base:

- 64 node Dell PC cluster (queue name: cbweb), with 128 Pentium III 1.0GHz processors total.
- 192 node Dell PC cluster (queue name: cbsu), with 256 Pentium 4 Xeon 2.4GHz processors total.
- 2 general purpose Dell PC servers for web applications and software development.
- 1 Dell PowerEdge 4600 server running GeneTraffic microarray data analysis software.
- Full access to CTC SQL servers
**Research**
We collaborate on specific research projects that require expertise in genomics and structural biology like protein structure prediction and modeling, data mining.

**Web Computing**
One of the major obstacles faced by many biologists is the difficulty of accessing and using state-of-the-art computational biology tools, especially on high performance computing platforms that are required for many tasks. We design simple-to-use, web-based interfaces that allow easy access to our dedicated computational resources.

**Software Development**
We are developing (or modifying) software tools for customized solutions. Since many applications are computationally intensive, we also port selected programs to the massively parallel environment available at the Theory Center.
At present the cluster (running MS Windows 2000) is used in a batch mode:

– the job is submitted to the queue by the script started from the web page
– user is notified by e-mail when the job finishes
– results are accessible through web link or attached to the e-mail

Batch mode is best for long, massively parallel applications

For short, serial tasks we are now developing a system based on web services (SOAP).
Welcome to the Computational Biology Service Unit at the Cornell Theory Center

<table>
<thead>
<tr>
<th>Web computing tools developed/hosted at CBSU</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOFP</td>
</tr>
<tr>
<td>URMS</td>
</tr>
<tr>
<td>P-CLUSTALW</td>
</tr>
</tbody>
</table>

New! Workshop slides (including the latest microarray workshop)

CBSU is a computational resource for the Tri-institutional Collaboration among Cornell University, Cornell/Weill Medical College, Rockefeller University, and Memorial Sloan-Kettering Cancer Center. CBSU is also a part of the Cornell Genomics Initiative.

The Computational Biology Service Unit group provides software and hardware support for computational biology applications. Assistance in the design and implementation of computational solutions is also provided. The CBSU responsibilities cover a wide variety of topics and tasks, including:

- **Web Computing**
  One of the major obstacles faced by many biologists is the difficulty of accessing and using state-of-the-art computational biology tools, especially on high performance computing platforms that are required for many tasks. We design simple-to-use, web-based interfaces that allow easy access to our dedicated computational resources.

- **Web Resources**
  We maintain a portal of computational biology resources available on the Internet. The resources are screened and rated by us and (potentially) by the users.
Typical problems and tasks

**Sequence analysis and annotation**
- Function annotation (BLAST, HMMER, LOOOPP → protein structure prediction, analysis and modeling)
- Unigene clustering (BLAST)
- Cross-species genome comparison (BLAST, CLUSTALW)
- Genome sequence analysis (RepeatFinder, BLAST)

**Protein structure prediction**
- Homology modeling (BLAST, MODELLER)
- Threading (LOOOPP)
- Ab-initio (UNRES)

**Protein structure modeling**
Computational tools available on our cluster (Parallel)

- **P-BLAST**: parallel driver for BLAST, accessible through web interface. Allows for genome scale sequence comparison and data mining.

- **P-HMMER**: parallel driver for HMMER program, an Hidden Markov Model program for sequence analysis, accessible through web interface. Much slower than BLAST and thus requiring large computational power for genomic scale calculations.

- **LOOPP**: parallel program for threading-based protein structure prediction, accessible through web interface. Requires significant computational power even for a single sequence prediction (~hours on a single CPU).

- **P-LOOPP**: version of LOOPP for large scale threading-based gene annotation.

- **UNRES**: parallel ab-initio program for protein structure prediction.

- **CRYSTALG**: parallel program for predicting structure of crystals composed of small organic molecules.
Computational tools available on our cluster (Serial)

- **G-BLAST**, designed for aligning a collection of small sequences against a large genome sequences. It can also be used to visualize any sequence alignment.

- **I-BLAST**, designed for large query sequence, this program provides a mask-and-reiterate option so the blast targets can be evenly distributed throughout the whole sequence range. Output is hierarchical in html format.

- **RepeatFinder**, this program identifies small scattered repetitive sequences in the genome. Provides spreadsheet data and html-based visualization.

- **URMS**, the algorithm compares the shapes of two proteins based on their alpha-carbon backbones. In this version, a single three-dimensional (3D) rigid motion is used to orient the proteins. Currently in web services version.

- **MDIV**, a program that will simultaneously estimate divergence times and migration rates between two populations under the infinite sites model or under a finite sites model.

BLAST

Output

- Spreadsheet, multiple sequences (*P-BLAST*)
- HTML, big sequences (*I-BLAST*)
- HTML graphical-oriented, interactive genome alignment (*G-BLAST*)

Computations

- Multiple (large scale) small or medium size sequences (*P-BLAST* – sequence-based parallelization)
- Several big sequences (*I-BLAST, mpiBLAST* – database-based parallelization)
- Few small or medium size sequences (*web services* – serial)
P-BLAST: general scheme

Web server

File server

Cluster

User

web page

network disk

queue submission script

e-mail
P-BLAST: MPI scheme

trivial parallelization

Master (node 0):
- Initialize workers (read and n sequences to workers)
  - Wait for a worker to finish
  - Receive results from worker, append to results file
  - Read sequence from input file
  - Send sequence to worker
- Finalize workers (receive all remaining results and send TERM)

Worker
- Wait for a sequence from master
- BLAST it (iteratively with masking if required)
- Send results back to master
P-BLAST: Stability and performance

- BLAST program engine is vulnerable to some types of invalid input causing it to stall. Examples: too short sequence (shorter than seed) after removing N or X, low complexity sequences with blastx.

- BLAST program should be run on worker node in parallel to the MPI driver program and its execution time should be measured against average.

- When the engine stalls it should be restarted and the offending sequence reported to user (usual rate is 1-10 out of 30,000).

- Input sequences should be tested for known problems on worker node (stalling is expensive).
P-BLAST: Stability and performance

• BLAST databases should be installed locally on nodes in order to minimize network usage
  *We are using our own MPI copying program for database propagation*

• BLAST engine should remain in memory as long as the queried database is the same, loading database is a significant part of execution time for short sequences

• P-BLAST is can be restarted
RepeatFinder – another BLAST-based tool

This program identifies small repetitive sequences scattered in the genome. Possible repeat sizes are from about 15 to few hundred bp.

• It is highly configurable and flexible
• It is capable of dealing with large sequences
• It is not only identifying repeats, but it also clusters them and estimates statistical significance
• The resulting repeats are also clustered together based on partial sequence overlaps and their statistical significance
• It provides easy-to-read, HTML-based output, as well as spreadsheet data files
RepeatFinder: algorithm

- The input sequence is iteratively blasted against itself after it is divided into multiple overlapping blocks.
- Matrix of relationships for all putative repeats of appropriate length is calculated.
- The repeats are clustered into groups by analyzing the relationship trees; minimal tree algorithm with end-point deviation metric is used.
- Optimal multiple sequence alignment in each group is calculated and variable ends are trimmed.
- E-value and variability in each group are calculated; too small or too variable groups are eliminated and most probable sequence is calculated.
- Most probable sequence from each group is blasted against the input sequence to check for missed matches.
- The groups are clustered together based on statistical significance or inter-group overlaps.
Information is presented as plots as well as numbers.

All the plots are clickable; a message box will present numerical value(s) of the field being clicked. In the case of distribution plot it will show the boundaries of the segment being clicked (size of the segment is reported above the plot).

RMSD of the repeat’s length is calculated as root-mean-square deviation from the average length.

Composition of a group is shown in a shaded table. Shade of the element is proportional to its numerical value. The first four rows show nucleic acid occupancies (A, C, G and T), the fifth one shows gaps. The sixth row 'P' reports the probabilities of a given field to be occupied by any nucleic acid (not being a gap), and the last row 'V' shows variability.

Variability is calculated as a probability of a given position being occupied by something else than the most probable occupant (0.0 - only one kind of occupancy, 1.0 - flat distribution).
• HMMER is a Hidden Markov Model program for profile-based sequence analysis. It can be used in cases when sequence similarity is low and BLAST cannot find matching segments with significantly high scores.

• P-HMMER utilizes the same scheme as P-BLAST, uses different calculation engine (HMMER).

• HMMER needs much longer time to calculate so it is usually a good idea to filter the input first using P-BLAST if possible

• Due to much longer execution time parallelization is essential
Protein structures can share similar folds despite no significant sequence similarity: an example is myoglobin (1mba) and leghemoglobin (1bin:A).

Identifying (predicting) protein structure helps in determining its function.

Threading can be considered as the third step in sequence analysis after BLAST and HMMER.
**Threading (fold recognition)** is an algorithm for finding the best alignment of a given sequence with another sequence for which its corresponding structure is known, when sequence identity is low.

- Evaluates how well a given sequence fits into 3D structure of another protein (scoring)
- Searches the whole database of structures (templates) choosing the best one
- Produces optimal sequence-to-structure alignment

Protein name: 1I4Z_D
MGFPIPDPYV … KGKI
Example: alignment

Structural sites

Target

1 2 3 4 5 6

alignment
Each alignment is scored with energy function

\[ E(R,1) + E(V,2) + E(W,3) + E(-,4) + E(T,5) + E(V,6) = E_1 \]

\[ E(R,1) + E(V,2) + E(W,3) + E(T,4) + E(-,5) + E(V,6) = E_2 \]
Threading algorithm

- Define a potential
- MODEL – define environment
- A-R-W-S-S-H-
- Evaluation
- Align sequence using potential
- Fold library

query
LOOPP: algorithm scheme

LOOPP program is hierarchical: the driver program calculates the global score based on partial scores ("features") calculated by engine program (raw threading, secondary structure prediction).

**INPUT**: Query sequence

- Finding related sequences and predicting secondary structure
- Calculating threading features of a query sequence and each related sequence versus the whole threading database
- Supplementing missing features

**OUTPUT**: Hits
LOOPP: algorithm scheme

Initialization – serial (0.0 %)

Finding related sequences and predicting secondary structure
3 tasks – parallel – (0.8%)

Calculating threading features of a query sequence and each related sequence versus the whole threading database
5 - 50 tasks – parallel – (26.5%)

Supplementing missing features
500 - 10,000 tasks – parallel – (71.9%)

Scoring and formatting output
serial – (0.8%)
Loop:: Parallelization notes

Master-worker approach similar to P-BLAST

Master is following the main algorithm distributing work to nodes

Fortunately the most parallelizable part is also the most computationally expensive
LOOPP: Scaling
P-LOOPP: Multisequence LOOPP

Master coordinator: distributing sequences to node groups

Group coordinator: organizing LOOPP calculations for one sequence
- Worker – calculating features
- Worker – calculating features
- Worker – calculating features
- Worker – calculating features
- Worker – calculating features

Group coordinator: organizing LOOPP calculations for one sequence
- Worker – calculating features
- Worker – calculating features
- Worker – calculating features
- Worker – calculating features
- Worker – calculating features

Group coordinator: organizing LOOPP calculations for one sequence
- Worker – calculating features
- Worker – calculating features
- Worker – calculating features
- Worker – calculating features
- Worker – calculating features
P-LOOPP: Scaling
Ab-initio structure prediction can be considered as the fourth step in sequence analysis after BLAST, HMMER and threading.

It is energy-based approach that doesn’t make any assumptions about investigated protein. It carries out a search of the structure as a global minimum of a designed potential-energy function, following Anfinsen’s thermodynamic hypothesis.

It is most computationally expensive and usually also very complicated method.
Producing 3D structures: a timescale

- Crystallography / NMR: months
- Ab-initio methods: few days – weeks
- Threading: hours – days
- Homology Modeling: minutes – hours
- Refinement and analysis: minutes – forever
Hierarchical Approach to Protein-Structure Prediction

Stage 1: Global optimization of the potential-energy function in a simplified representation of the polypeptide chain. This is the key stage of the algorithm.

Stage 2: Conversion of the lowest-energy structures to the all-atom representation.

Stage 3: Limited energy optimization of the all-atom structures.
Illustration of the hierarchical algorithm for protein structure prediction

lowest in energy at the simplified level
All-atom representation of polypeptide chain in solution (explicit water)

United-residue (UNRES) representation of polypeptide chain
Monte Carlo-Minimization (MCM)

Generate at random set of structures and locally minimize them.

1. Select the lowest-energy one as "generative" structure.
2. Carry out random change (perturbation) of "generative" structure to produce a new conformation.
3. Minimize the energy of the new conformation.
4. Compare its energy to energy of the "generative" structure by means of the Metropolis criterion [accepted with probability of \( \exp(-\Delta E/kT) \)].
5. If accepted in Metropolis criterion new (minimized) structure becomes the "generative" structure, otherwise the "generative" structure remains unchanged.
6. Iterate into point 3.
Conformation-Family Monte Carlo (CFMC)

- Uses the **Metropolis** criterion to move between families
- Uses the **Boltzmann** distribution to choose conformation from a family
- Does not move between structures, but between families
- It is equivalent to smoothed “staircase deformation” of a potential function

Original function | MCM | CFMC
S(i) – structure i
M(i) – Metropolis test against structure i (accepted with probability $\exp[(E-E_i)/kT]$ or when $E<E_i$)
P/M – perturbation followed by local minimization

Simulation time
$C(k,l)$ – comparison with structures $k$ and $l$ (accepted when $E<E_k$ & $E<E_l$)
Massive parallelization

The previous scheme would work fine if the number of workers is proportional to the *acceptance ratio* (in this case up to 10-20 processors). If not the relation between minima is lost and a method becomes merely a random search …

In order to avoid this problem the parallel MCM search is carried out using parallel *threads* searching simultaneously different parts of the configurational space and updating the same structure-family database.
Efficiency of parallelization of the CFMC algorithm
Efficiency of parallelization of the CFMC algorithm

![Graph showing efficiency of parallelization for CFMC algorithm with number of processors on the x-axis and efficiency on the y-axis. Two lines represent efficiency: one excluding the master and another including the master. The graph indicates a decrease in efficiency as the number of processors increases.]
CBSU core:

Jaroslaw Pillardy  
Daniel Ripoll  
Qi Sun
Acknowledgments

LOOPP has been developed in prof. Ron Elber group
Computer Science, Cornell University

UNRES and CRYSTALG have been developed in
prof. Harold A. Scheraga group
Department of Chemistry, Cornell University