

**Real Science**  
**Real Software**  
**Real Performance**

## Computing for Drug Discovery



Algorithm Overview

## Overview

- **Application Space**
- **MTAP architecture**
- **Development Environment**
- **Algorithm Examples**
  - Smith-Waterman Comparison
  - 3-point Pharmacophore Fingerprinting
  - Protein-Ligand Docking
- **Conclusions**

# Bio compute Application Space

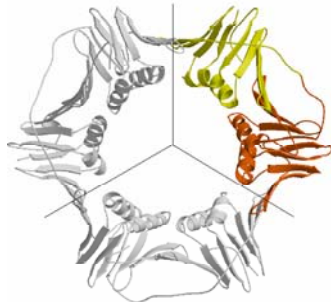
Assembly and analysis of genome sequences

```

ATCTCCGC
CGCTAGCTAA
AATATG
GCTAGCTAATA
TCTCCGCTA
-----
ATCTCCGCTAGCTAATATG
    
```

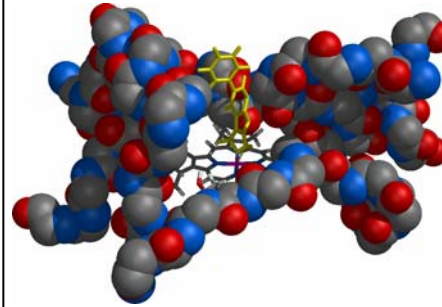
Integer comparison and search

Prediction of protein structure



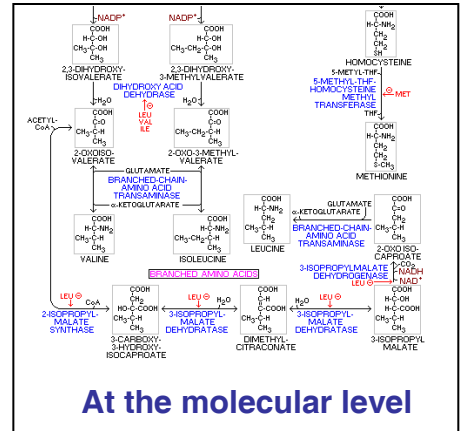
Integer/Float, simplified models for speed

Molecular modeling of biochemical reactions



From Simplified through to MM/QM

Process simulation of biological systems



At the molecular level

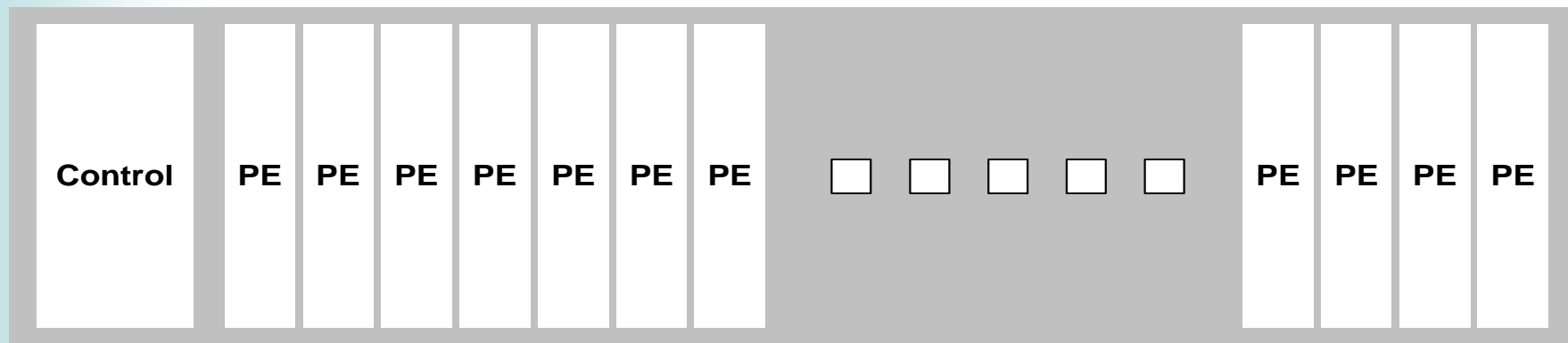
Compute

ratio

Bandwidth

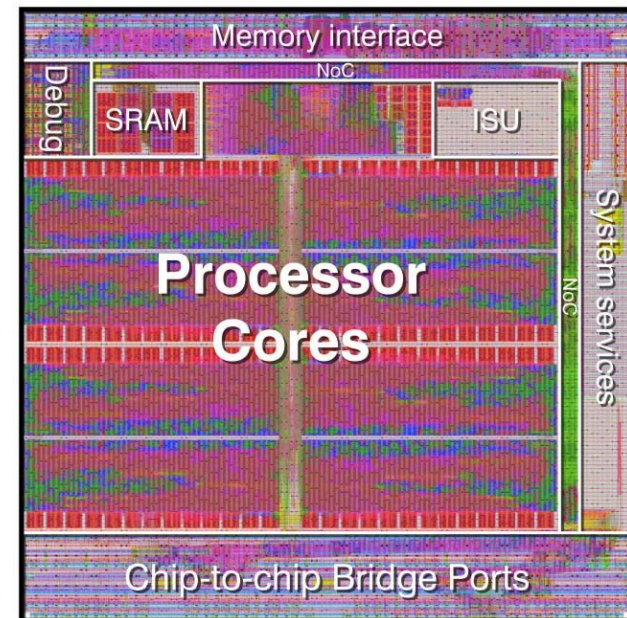
## MTAP Architecture

- **Multi-Threaded Array Processing**
  - An advanced SIMD architecture
  - Array of simple Processing Elements (PEs)
  - Single shared controller
  - Many data items processed in parallel



## Benefits of architecture

- **Scalable and modular**
- **High bandwidth**
  - 10s to 100s Gbytes / second
- **High compute**
  - Billions of operations / second
- **Power efficient**
  - 10 GFLOPS/Watt
    - P4 3.2Ghz = 0.1 GFLOPS/Watt
- **Simple programming model**
  - Programmed in C

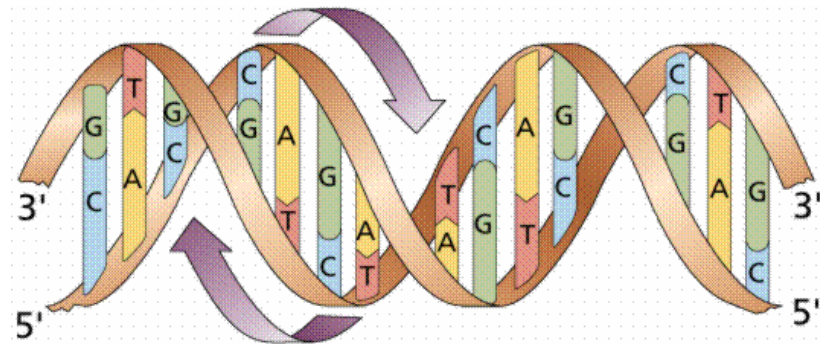




**Real Science**  
**Real Software**  
**Real Performance**

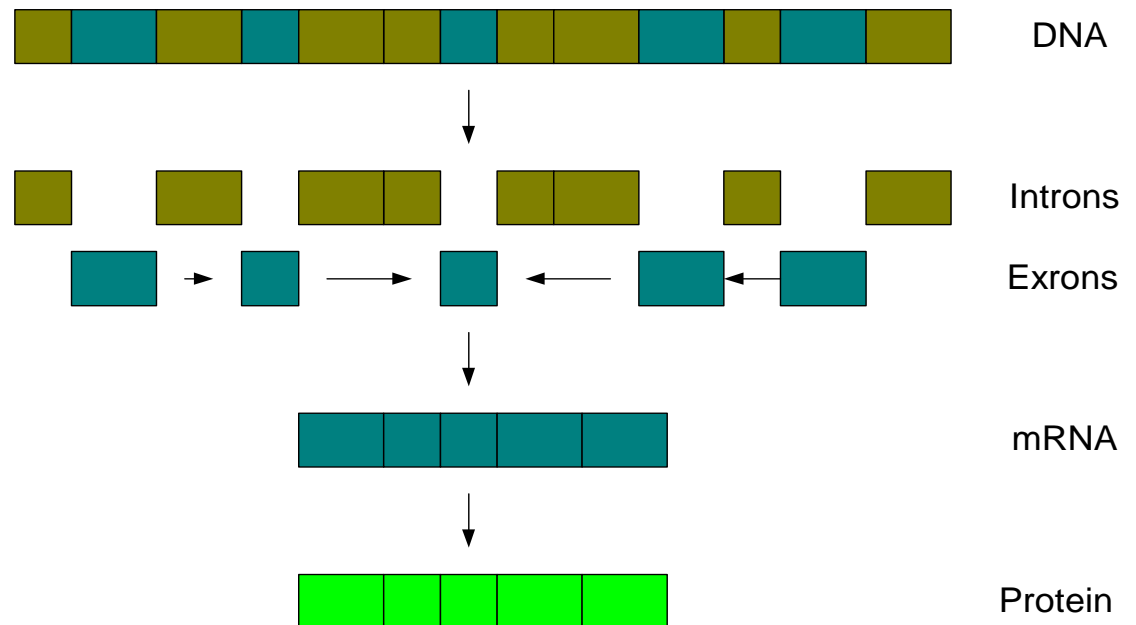
## **Sequence comparison**

- **Proteins and Genes**
- **The Search Problem**
- **Dynamic Programming**
- **Parallel Implementation**



## Proteins and Genes

- DNA contains only four chemical bases: A, T, C, G. Groups of three of these represent amino acids.

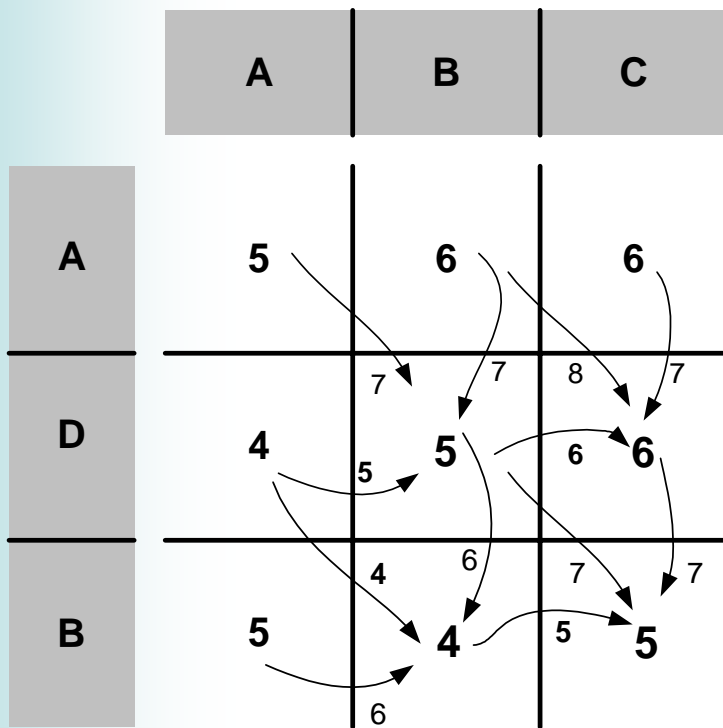


- Proteins are sequences of amino acids.
- DNA contains a mixture of protein creating sequences mixed with 'noise'.
- The transition from DNA to mRNA removes the noise.

## The Search Problem

- **Molecular Biologists have two related search problems to undertake:**
  - Firstly, they wish to locate a sequence of DNA that can generate a known protein.
  - Secondly, they wish to compare two sequences of DNA to locate genes in one strand that appear in another.
- **This comparison problem requires a fuzzy match in order to discard the impact of the ‘noise’ and also small changes in the active part of the gene.**

# Dynamic Programming



- **Initialize Matrix**
  - Each axis labeled with the two sequences to be compares.
  - Top and left edges zero.
- **Each entry is minimum of:**
  - Above +1
  - Left +1
  - Above Left +0 if match
  - Above Left +2 if no match

## Smith-Waterman Algorithm

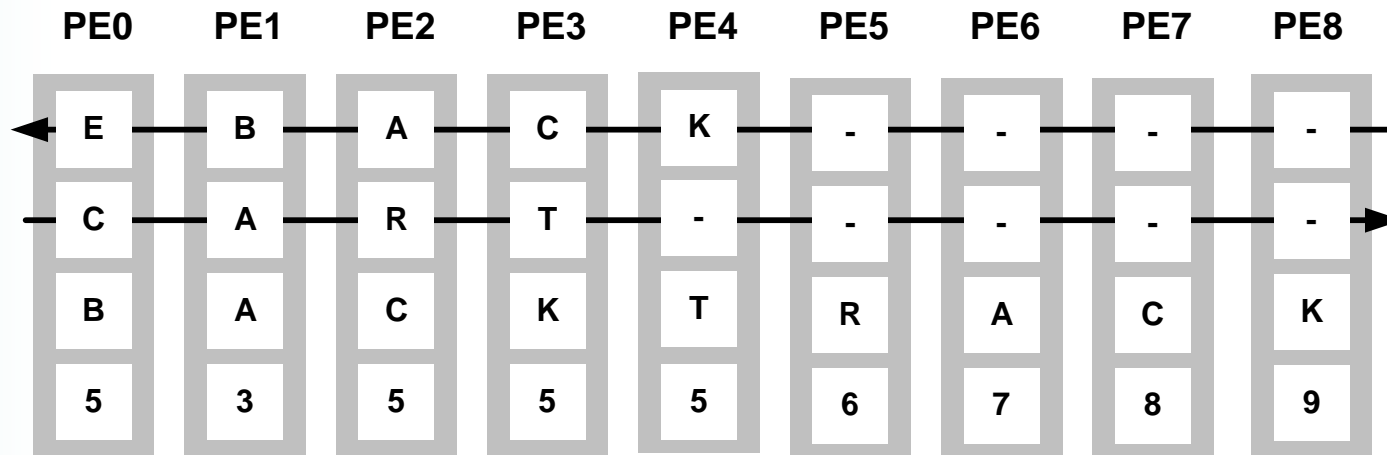
- **Smith-Waterman is ideal for locating target areas for proteins within a large database.**
- **Use a more complex set of rules.**
  - $H(l,j) = \max (0, E(l,j), F(l,j), H(l-1,j-1) + Sbt(S_i,D_i))$
  - $E(l,j) = \max (H(l, j-1) - \alpha, E(l, j-1) - \beta)$
  - $F(l,j) = \max (H(l-1, j) - \alpha, F(l-1, j) - \beta)$
  - $H(l,0) = E(l,0) = 0, H(0,j) = F(0,J) = 0$
  - $Sbt(x,y) = 2$  if  $x=y$ , else  $-1$
- **alpha, beta typically 1.**

## Parallel Implementation (I)

	*	T	R	A	C	E	B	A	C	K
*	0	1	2	3	4	5	6	7	8	9
B	1	2	3	4	5	6	5	6	7	8
A	2	3	4	3	4	5	6	5	6	7
C	3	4	5	4	3	4	5	6	5	6
K	4	5	6	5	4	5	6	7	6	5
T	5	4	5	6	5	6	7	8	7	6
R	6	5	4	5	6	7	8	9	8	7
A	7	6	5	4	5	6	7	8	9	8
C	8	7	6	5	4	5	6	7	8	9
K	9	8	7	6	5	6	7	8	9	8

- Values in diagonals independent.
- Linear array can compute diagonals.
- Only a few previous diagonals required if not backtracking.
- Only local communication needed and low storage overhead.
- Store search string in PEs, shift database past PEs.

## Parallel Implementation (II)



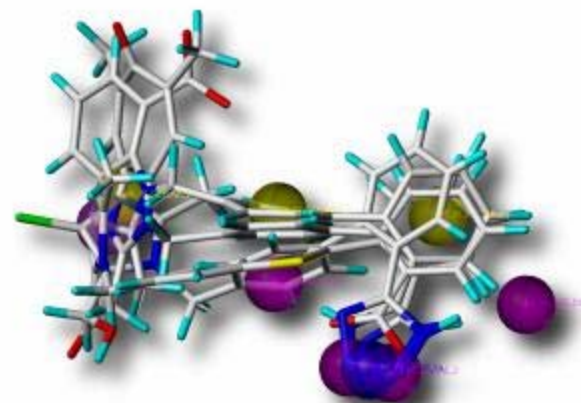
- Each PE holds one element from (short) sequence. Longer sequences can be folded onto array.
- Second (longer) sequence is shifted through PEs.
- Comparisons performed, and high points recorded.
- Second sequence is loaded in pieces, one element per PE.



**Real Science**  
**Real Software**  
**Real Performance**

## **Molecular Fingerprinting**

- **Molecular Fingerprinting**
- **3 Point Pharmacophore**
- **Tanimoto Similarity**
- **Parallel Implementation**

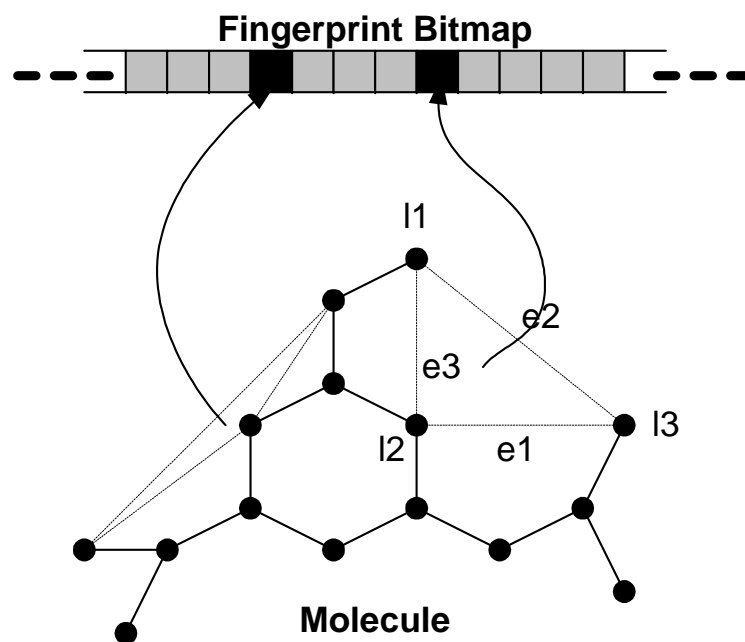


## Molecular Fingerprinting

- **Large database of potential ‘drug-like’ molecules.**
- **Search database for molecules:**
  - like a known molecule
  - or, with geometry that is inverse of a docking site.
- **Use ‘fingerprints’ of molecules. i.e. a simplified description of the molecules properties. Most frequently geometry based.**
- **Include all conformations of molecules.**
- **3 point pharmacophore is an example of a molecular fingerprint.**

## 3-point pharmacophore

Label	Definition
A	Hydrogen bond acceptor
D	Hydrogen bond doner
H	Hydrophobic
R	Aromatic
N	Negatively charged
P	Positively charged
-	All others



Bin	Range
1	$2 \leq d < 4.5$
2	$4.5 \leq d < 7$
3	$7 \leq d < 10$
4	$10 \leq d < 14$
5	$14 \leq d < 19$
6	$19 \leq d < 24$

For every triple of atoms in molecule:

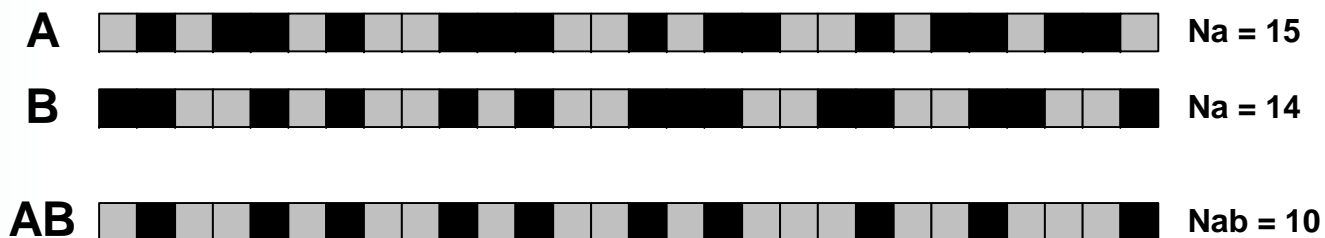
Use atom and edge labels to compute index into bit array.

~6.5K unique triangle labelings possible.

Use index to set bit in finger print.

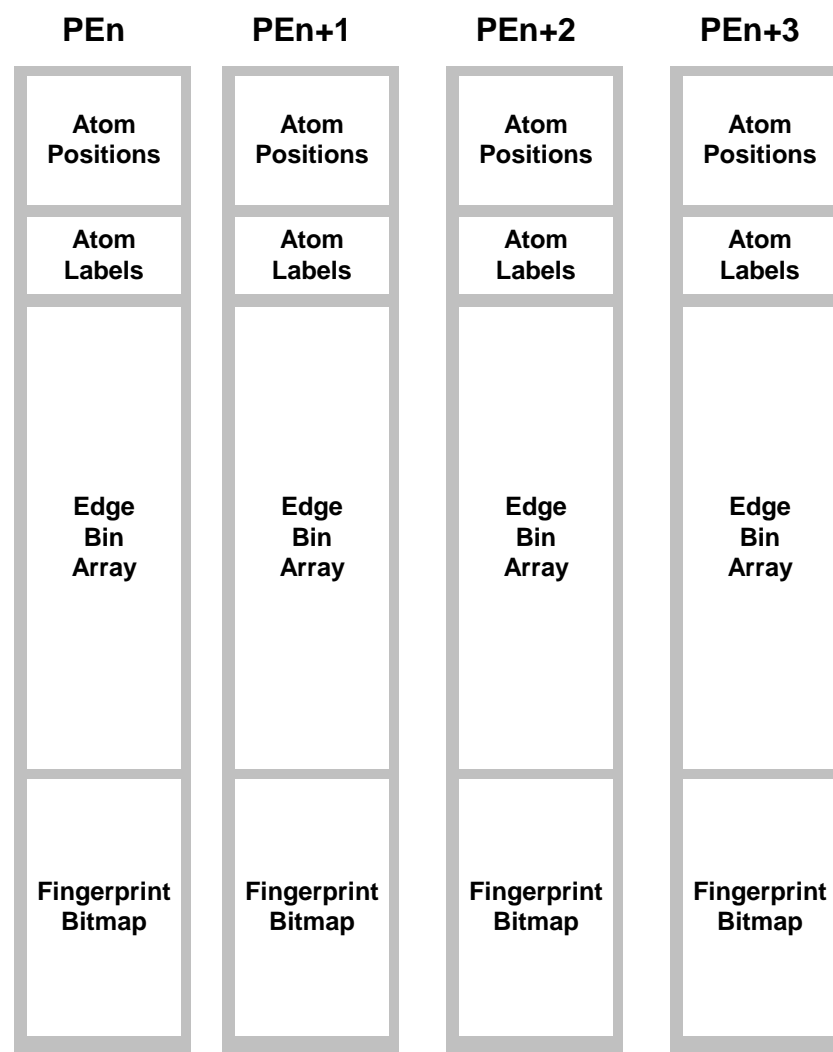
## Tanimoto Similarity


- **Tanimoto bit-sequence similarity**
  - Similarity =  $N_{ab}/(N_a + N_b - N_{ab})$
  - $N_a$  = Number of set bits in A.
  - $N_b$  = Number of set bits in B.
  - $N_{ab}$  = Number of set bits in common.
- **Result is in range 0.0 - 1.0.**
- **1.0 indicates exact match.**



$$\text{Similarity} = N_{ab}/(N_a + N_b - N_{ab}) = 10/(15 + 14 - 10) = 0.53$$

- **One molecule per PE.**
- **Pre-compute:**
  - $n(n-1)/2$  edges
  - Compute bin number.
- **Main loop:**
  - $n(n-1)(n-2)/6$  triangles
  - Combine edge bin numbers and types into bit index, set bit.
- **Storage:**
  - Atom position, 3 x 2 bytes. Atom label, 1 byte each.
  - Bitmap ~6.5Kbit
  - Edge bin array, 4 bits per entry, 4KB PE sufficient for 64 atoms.



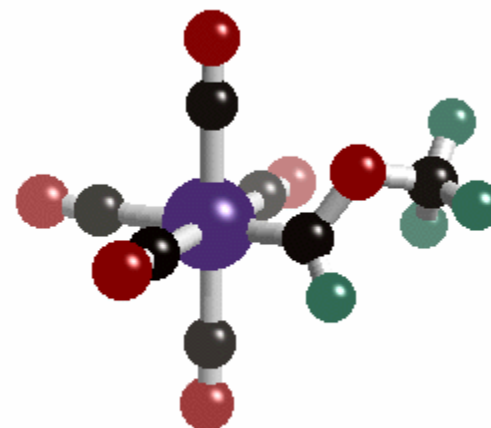


**Real Science**  
**Real Software**  
**Real Performance**

## **Molecular dynamics-based drug docking**

## Algorithm Example III: Docking

- **Protein - Ligand Docking**
- **Algorithm**
- **Parallel Implementation**
- **Performance**

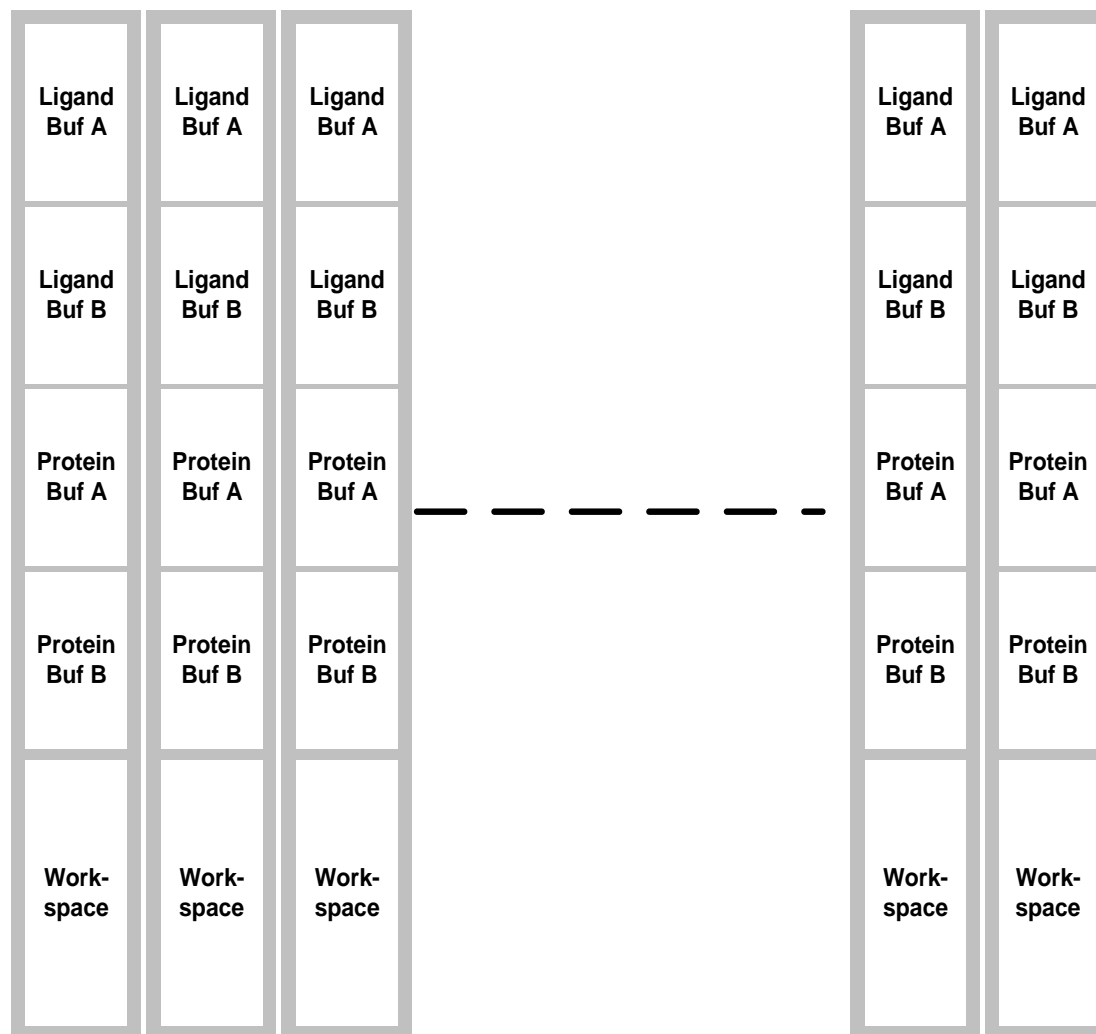


## Method

- **Protein-Ligand Docking**
  - Protein (100s-1000s of atoms)
  - Ligand (10-100s of atoms)
  - Assume protein docking site is known.
  - Identify position and orientation of ligand with minimum protein/ligand interaction energy.
- **Method**
  - For each possible transform:
  - Sum atom-atom interaction energies for L x P atom pairs.
  - Sort energy sums to locate lowest N.
  - Generate further set of transforms from N candidates.
  - Repeat until good enough fit found.

## Parallel Implementation

- One ligand - protein calculation per PE.
- 40 bytes per atom, allows 128 atoms in 5Kbytes. 2 x 32 atoms double buffered.
- Process:
  - Load n ligand atoms
  - Transform ligand atoms
  - Load m protein atoms
  - Compute n x m atom-atom interaction energies and sum.
  - Load next m protein atoms while computing atom-atom.
  - Load next n ligand atoms. Repeat.



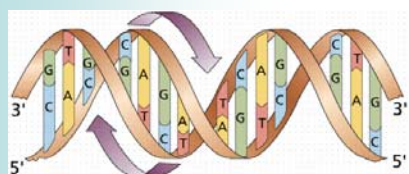
## Atom-Atom Interaction

- **Atom-atom interaction calculations performed in parallel, one per PE.**
- **All data needed is local to PE. Allows choice of computational function.**
- **Example function requires:**
  - 16 adds
  - 9 multiplies
  - 10 compares
  - 2 divides
  - 1 square root

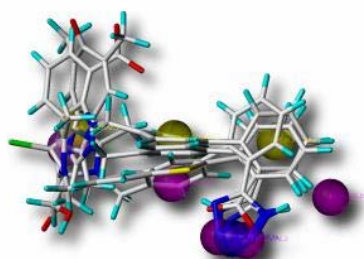
## Performance

- **Compute cost.**
  - ~300 cycles per atom-atom.
- **Bandwidth requirements:**
  - Assume 32 atom subsets.
  - 32 x 32 atom-atom calculations = 1024
  - 32x32x300 cycles per I/O operation = 307,200
  - 32 atoms x 40 bytes = 1,280 bytes (all PEs load same atoms)
  - $1280/307,200 * 210\text{Mhz} = 0.9 \text{ Mbytes/s!}$

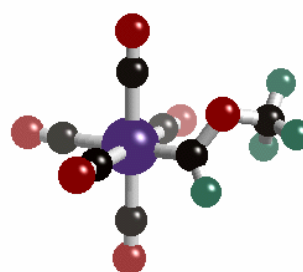
# Summary



**Smith-Waterman  
Sequence Comparison**



**3 Point Pharmacophore  
Molecular Fingerprint**



**Protein-Ligand  
Docking**



**Molecular Mechanics**

Compute

Bandwidth

## Conclusion

- **Architecture designed for *data-intensive* applications**
  - Efficient compute architecture
- **Good fit for most bio-tech problems**
  - Algorithms tend to have inner loops with independent calculation.
  - Well suited to parallel implementation. ~100% PE utilisation.
- **Orders of magnitude performance improvement**
  - Low power, high compute, permits very dense compute solution.