ClearSpeed™

University of BRISTOL

An Accelerated, Computer Assisted Molecular Modeling Method for Drug Design

Dr Richard Sessions, Bristol University

Mr Simon McIntosh-Smith, ClearSpeed Technology

ClearSpeed

University of BRISTOL

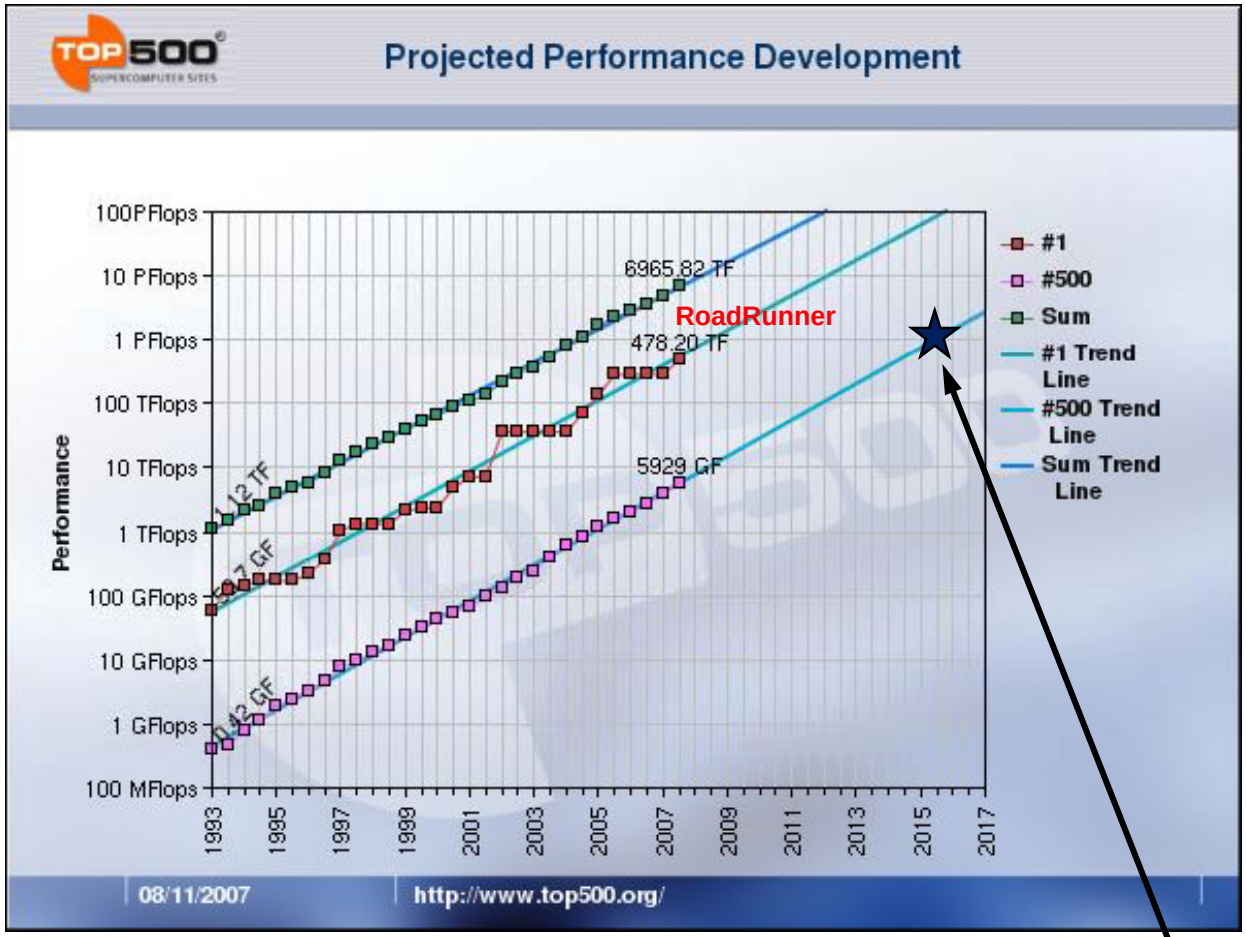
Introduction

- **Next generation protein-ligand docking algorithms require significantly more compute than current approaches**
- **Accelerators such as those from ClearSpeed are one of the most promising ways forward to higher performance systems**
  - "My prediction: High performance computing will soon be dominated by accelerator-based systems." – Michael Wolfe, The Portland Group
- **This work has been investigating mapping such next generation docking algorithms to cutting edge, HPC-optimized accelerators**

ClearSpeed

University of BRISTOL

After all, *everything* will be Petascale soon!



- **Within 7 years *everything* will be Petascale!**

Introduction to accelerated systems

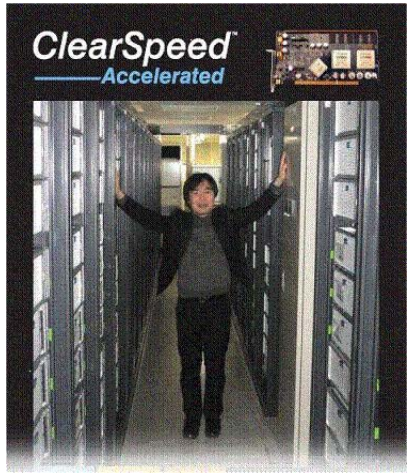
Many systems are already reaching infrastructure limits:

- Data center size
- Power supply
- Cooling

Accelerators emerging to significantly increase performance per (cubic meter, watt)

Tokyo Tech created the first of the new wave of accelerated supercomputers, TSUBAME

- Performance increased from 38 TFLOPS to 56 TFLOPS with 648 ClearSpeed Advance™ accelerators
- An increase in performance of 47%, but for just a 2% increase in power consumption, 0% increase in space
- #9 in the November 2006 Top500



Professor Matsuoka standing beside TSUBAME at Tokyo Tech

June 17th 2008 News

Today ClearSpeed, the only company designing accelerators specifically for HPC introduces a new range of products based on our latest accelerator:

The CSX700 “Callanish” processor

The CSX700 – an accelerator designed for HPC

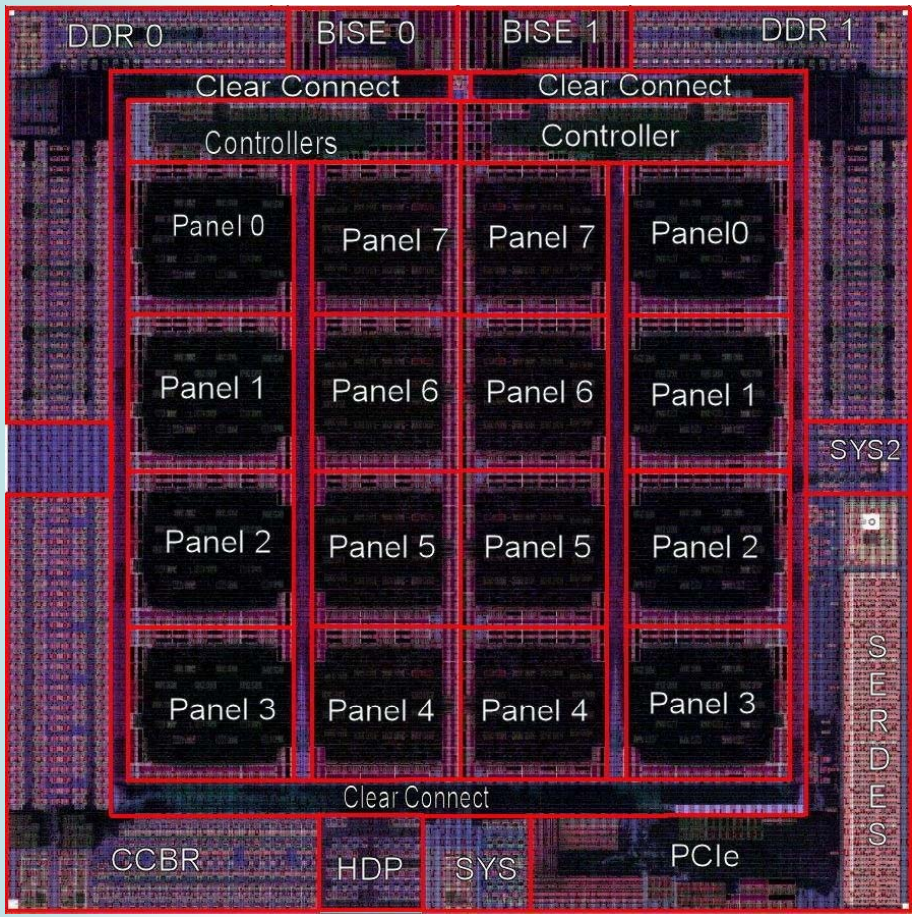
- **ClearSpeed is the only company designing accelerators specifically for HPC:**
  - Focus on 64-bit double precision for high accuracy
  - High reliability features designed in
  - Low power combined with high performance per watt
  - Form factors to fit into the standard blades and servers that populate large datacenters

- **The CSX700 is the latest accelerator from ClearSpeed, delivering big increases in:**
  - Performance,
  - Performance per dollar, and
  - Performance per watt

ClearSpeed™

University of BRISTOL

The CSX700 – "Callanish"

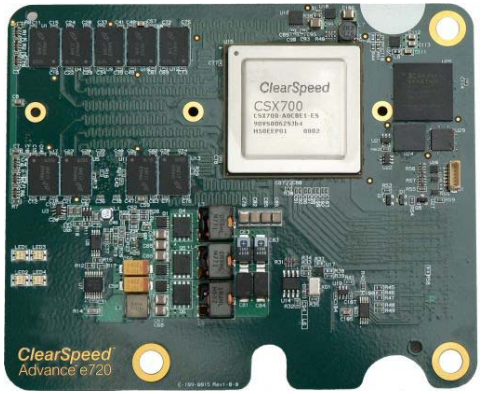


- **Processor Cores:**
  - **192 Processor Elements (2x96)**
  - **96 double precision GFLOPS**
  - **250MHz**
  - **8 redundant PEs**
  - **Error Correction (ECC) on all internal memories**
- **SoC details:**
  - **Integrated PCI Express x16**
  - **2x integrated ECC DDR2 memory controller + scrubber**
  - **2x128 KBytes of SRAM**
- **Design details:**
  - **IBM 90nm process**
  - **256 million transistors**
- **12W Max (Power Managed)**
- **Officially launching at ISC08**

ClearSpeed™

University of BRISTOL

The ClearSpeed Advance™ e710 & e720 accelerators



- ***Enterprise-class* HPC accelerators**
- **The *only* accelerators designed to fit into most standard servers and blades**
  - Low power consumption – 25W max; small, light
- **Designed for high reliability (MTBF)**
  - *All* memory is error protected; no moving parts needed (e.g. fans)
- **96 Double Precision (D.P.) IEEE 754 GFLOPS peak**
  - ~4 GFLOPS per watt double precision
- **Over 2 GBytes/s between accelerator and host – PCIe x8**
- **No extra power connectors, cooling or space/slots required**
- **Under $3000 each in volume, launching at ISC08**

ClearSpeed

University of BRISTOL

The ClearSpeed Accelerated Terascale System

CATS-700 launching at ISC08

- **Enterprise-class reliability:**
  - Error correct/detect on *all* memories
  - Error correct/detect on *all* communications
- **1.152 TFLOPS double precision (64-bit) in 1U**
- **12 Advance™ e710 accelerators**
- **24 GBytes of DDR2 DRAM with SECDED ECC and Scrub**
- **96 GBytes/s of DRAM bandwidth**
- **400 watts typical power consumption**
- **Two PCI Express x8 connections to the host (up to 3m long)**
- **Up to 41 TFLOPS double precision peak in a single rack**
  - From 36 CATS-700 1U nodes
- **10X greater peak performance than the fastest dual socket 3GHz quad-core servers at the *same* power consumption**

ClearSpeed™

University of BRISTOL

A rack of CATS-700

- **Enterprise-class reliability – ECC on *all* memories, both on- and off-chip**
- **From 18 CATS-700:**
  - **20.7 TFLOPS double precision**
  - **432 GBytes of DDR2 with ECC**
  - **1.73 TBytes/s of DRAM bandwidth**
  - **7.2 KW typical power consumption**
- **From 18 3GHz quad core hosts:**
  - **1.8 TFLOPS double precision**
  - **7.2 KW typical power consumption**
- ***22.5 TFLOPS double precision total***
- ***14.4 KW total power consumption***
- **No silent software errors**

ClearSpeed™

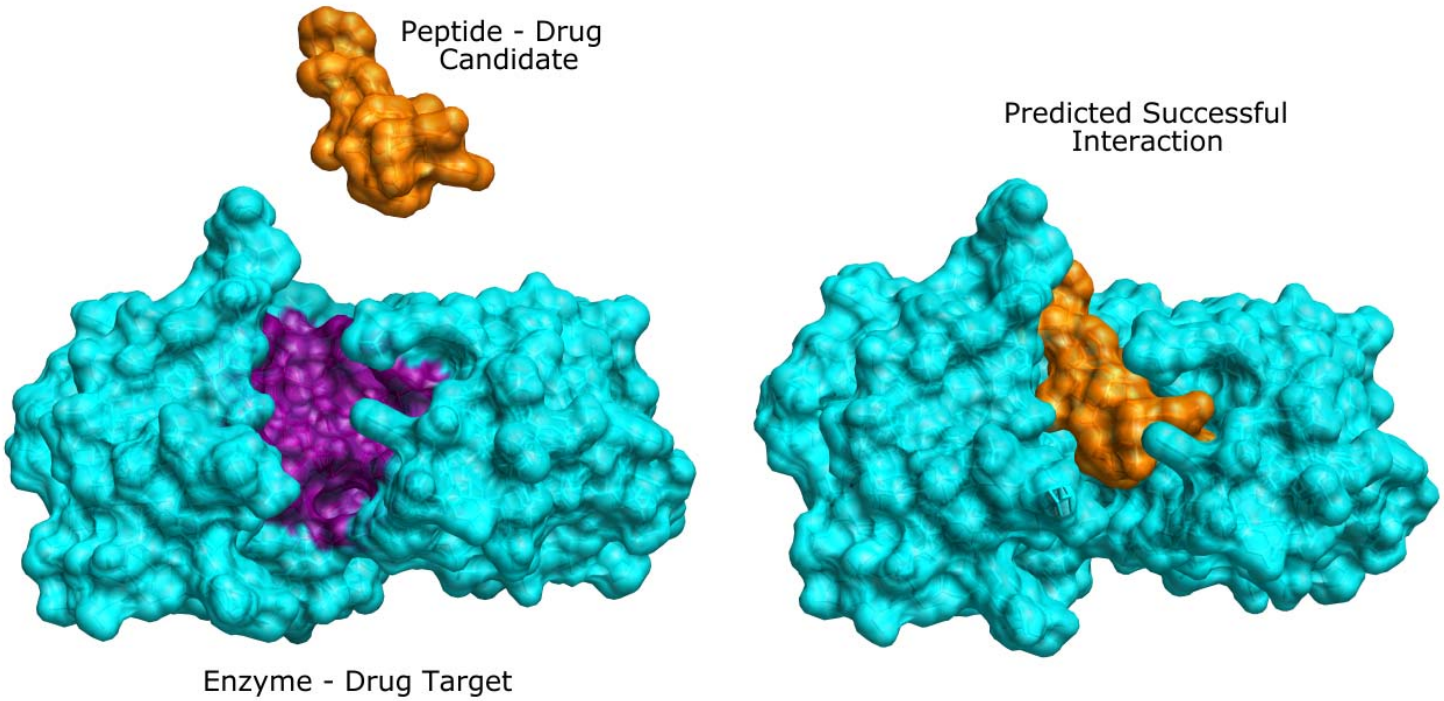
University of BRISTOL

Next-generation drug docking approaches: BUDE

Peptide Based Elastase Inhibitors: A Case Study

Therapy for Emphysema

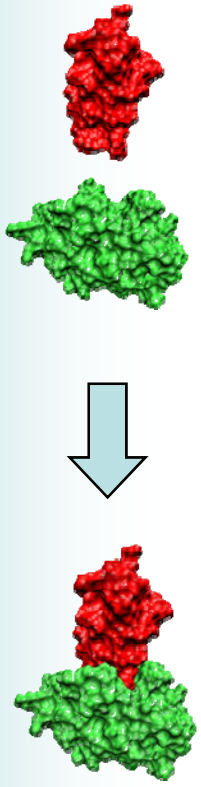
Peptide libraries (based on a Trypsin inhibitor)



Flexible amino acid side-chains in both protein (receptor) and ligand (peptide)

http://www.clearspeed.com/docs/resources/RSBUDE_WhitePaper.pdf

BUDE – molecular docking

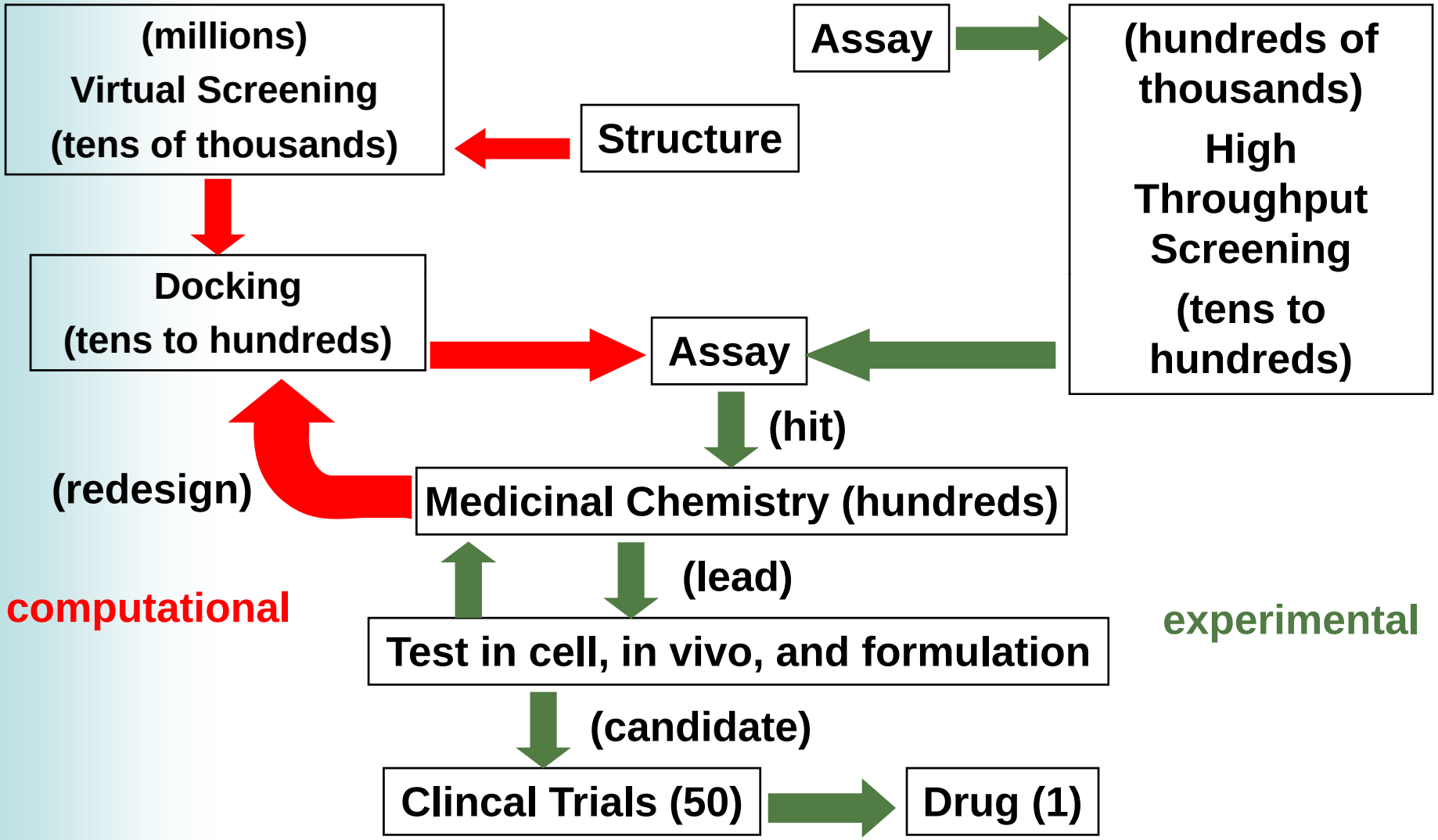


- **Specifically, protein-ligand docking**
  - Macromolecule – protein (receptor)
  - “Other molecule” – the ligand (peptide)
- **Predict the position, orientation and interaction energy of a ligand with the receptor**
- **Used in pharmaceutical research**
  - To follow virtual screening of large chemical databases
  - Select and redesign likely drug candidates

ClearSpeed

University of BRISTOL

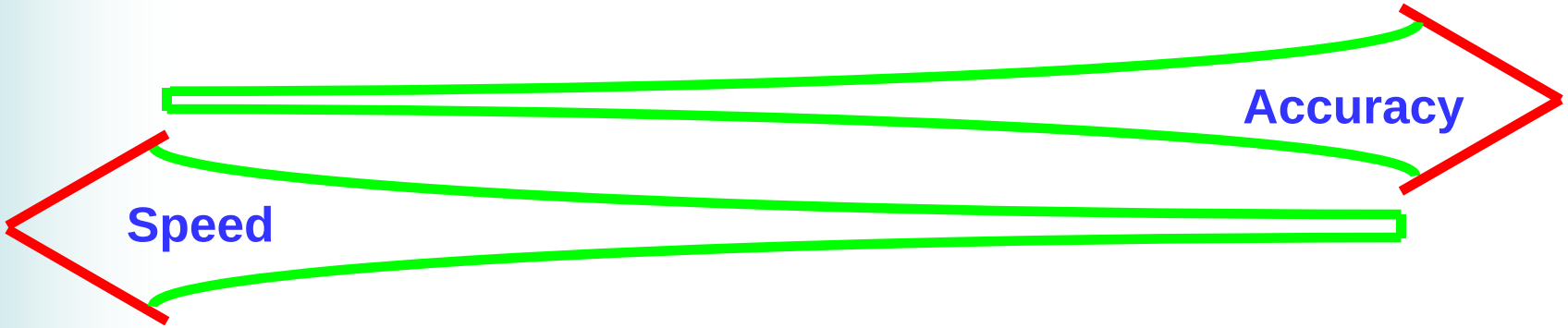
Docking in Drug Discovery: the pipeline



ClearSpeed™

University of BRISTOL

Need for Next Generation Docking Methods



	Typical docking scoring functions	Empirical Free Energy Forcefield BUDE	Free Energy calculations MM[1,2] QM/MM[3]
Entropy:			
solvation	**No**	**Yes**	**Yes**
configurational	**Approx**	**Approx**	**Yes**
Electrostatics	**?**	**Approx**	**Yes**
All atom	**No**	**Yes**	**Yes**
Explicit solvent	**No**	**No**	**Yes**

1. MD Tyka, AR Clarke, RB Sessions, J. Phys. Chem. B 110 17212-20 (2006)
2. MD Tyka, RB Sessions, AR Clarke, J. Phys. Chem. B 111 9571-80 (2007)
3. CJ Woods, FR Manby, AJ Mulholland, J. Chem. Phys. 128 014109 (2008)

Empirical Free Energy Function (atom-atom)

$$\Delta G_{\text{ligand binding}} = \sum_{i=1}^{N_{\text{protein}}} \sum_{j=1}^{N_{\text{ligand}}} f(x_i, x_j)$$

Parameterised using experimental data[4]

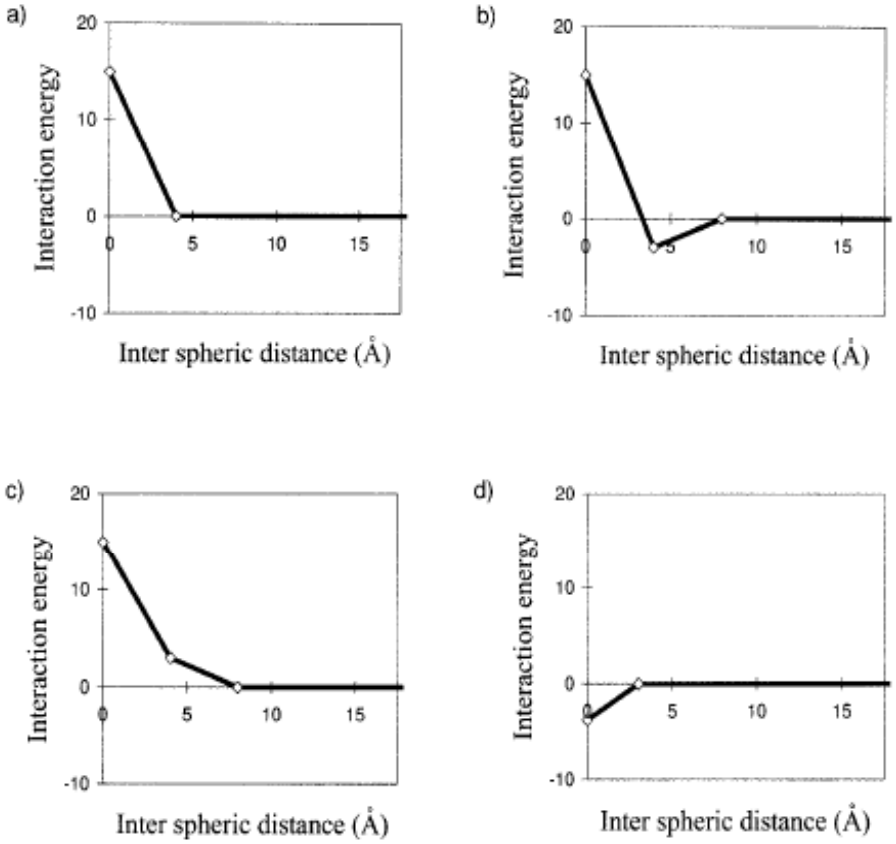


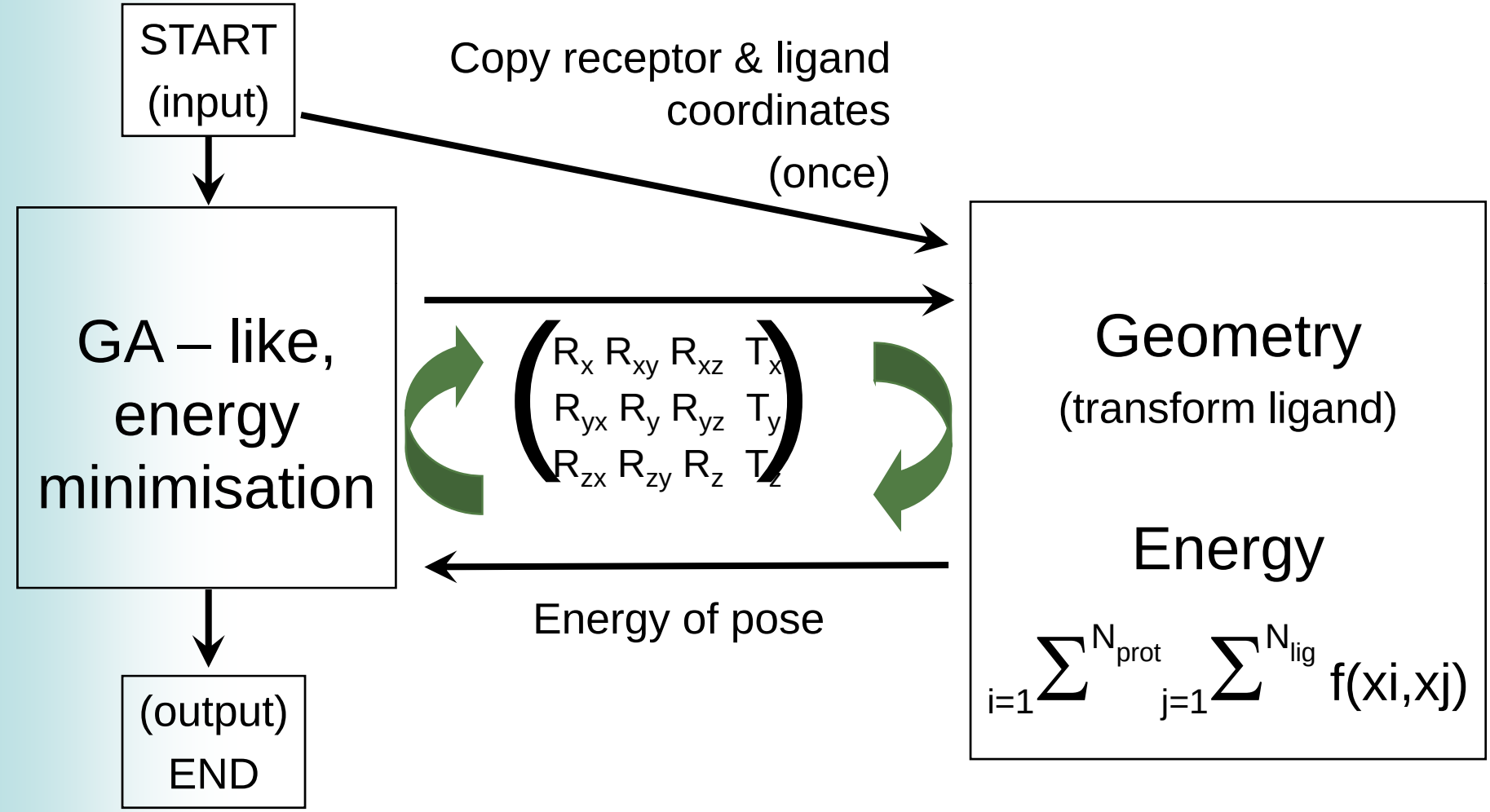
Fig. 1. Inter-residue sphere-sphere interaction energy functions of the force field. a: Between two polar spheres, or between a backbone sphere and any other non hydrogen-bonding sphere. b: Between two non-polar spheres. c: Between a non-polar sphere and a polar sphere. d: Between a hydrogen bond donor sphere and a hydrogen bond acceptor sphere.

[4] N. Gibbs, A.R. Clarke & R.B. Sessions, "Ab-initio Protein Folding using Physicochemical Potentials and a Simplified Off-Lattice Model", Proteins 43:186-202,2001

ClearSpeed

University of BRISTOL

BUDE Acceleration



ClearSpeed™

University of BRISTOL

Code examples: Host code – Advance board initialization

```
if (npes .gt. 0) then
call CS_Init(npes,ifail)
do ipe=0,npes-1
call CS_load_program(ipe,'fasten%p.csx',0,ifail)
if (ifail.ne.0) then
   write(*,'(a,i3)') 'Failed to attach to coprocessor', ipe
   stop
endif
enddo
do ipe=0,npes-1
call CS_find_symbol (ipe,'natpro',         1, 4,       0, ifail)
call CS_find_symbol (ipe,'protein_molecule', 2,
+                                          4*40*natpro,0, ifail)
call CS_find_symbol (ipe,'natlig',         3, 4,       0, ifail)
call CS_find_symbol (ipe,'ligand_molecule', 4,
+                                          4*40*natlig,0, ifail)
call CS_find_symbol (ipe,'ntransforms',    5, 4,       0, ifail)
call CS_find_symbol (ipe,'transforms',     6, 4*99999, 0, ifail)
call CS_find_symbol (ipe,'etotals',        7, 4*99999, 0, ifail)
call CS_find_symbol (ipe,'verbose',        8, 4      , 0, ifail)
call CS_find_symbol (ipe,'cutdis',         9, 4,       0, ifail)
call CS_find_symbol (ipe,'stats',         10, 4,       0, ifail)
enddo
```

Host code – copy receptor & ligand coordinates to board

```
!-- Build then send the Protein and Ligand molecules
!-- Turn the set of arrays that define the protein molecule into one object

          do i=1,natpro
            call packat (xatpro(1,i), rad_p(i), hphb_p(i), hard_p(i),
     +          nndstp(i), npdstp(i), elsc_p(i), hbtypp(i), atom_p(i) )
            protein_molecule(i)= molecule
          enddo

!-- likewise for the ligand
          do i=1,natlig
            call packat (xatlig(1,i), rad_l(i), hphb_l(i), hard_l(i),
     +          nndstl(i), npdstl(i), elsc_l(i), hbtypl(i), atom_l(i) )
           ligand_molecule(i)= molecule
          enddo

          do ipe=0,npes-1
             call CS_putf(ipe, 9, cutdis ,4,0,ifail)
             call CS_puti (ipe, 1, natpro,4,0,ifail)
             call CS_put  (ipe, 2, protein_molecule,4*40*natpro, 0,ifail)
             call CS_puti (ipe, 3, natlig,4,0,ifail)
             call CS_put  (ipe, 4, ligand_molecule,4*40*natlig, 0,ifail)
             call CS_run  (ipe,ifail)
          enddo
        endif
        endif  ! if first pass AND coprocessing
c
```

ClearSpeed™

University of BRISTOL

Host code – Send transformation matrices to board

```
      ibase = 1
      ! next isn't realy npes  - but how many pieces we want the transforms in
      !   this could be one - even on 2 boards if we give 1/4 protein to each csx


      do ipe = 0,npes-1
         ntransforms = nint (ncnf_offload/real(npes))
         ! round up to next mutiple of 96
         nrem = mod(ntransforms,96)
         if (nrem.ne.0) ntransforms = ntransforms+(96-nrem)     ! extend to next 96
         if (ibase+ntransforms.gt.ncnf) ntransforms = ncnf-ibase+1  ! but crop at end-of-list

         if (ntransforms.gt.0) then
            ! calculate ntransforms and store in transform_buffer(ipe) onwards
            ! convert the 3 trans and 3 rotations into a 3x4 TR matrix
            call secnd(secs0)
            call cnftrf(info,kdeb,transf,trsdec,tilttr,rolltr,pantr,
     +           xtratr,ytratr,ztratr,rtcntx,rtcnty,rtcntz,
     +           ibase, ntransforms,     transforms)
            call secnd(secs1)
            times(1,1) = times(1,1) + real(secs1(1)) - real(secs0(1))
            times(2,1) = times(2,1) + real(secs1(2)) - real(secs0(2))
            Write buffer to card and signal it to go
            write (*,'(a,i3,a,i6,a,i6, a,i6,a)')
     +      'coproc',ipe, ' will do from', ibase,' to',
     +      ibase+ntransforms-1,' (',ntransforms,' )'
            call CS_puti (ipe, 5,ntransforms,4,0,ifail)
            call CS_putf (ipe, 6,transforms, 4*12*ntransforms,
     +      SEM_GO,ifail)
         endif
         ibase = ibase + ntransforms
      enddo
```

ClearSpeed™

University of BRISTOL

Host code – copy energies back

```
c-------------------------------------------------------------------------
c wait for the results to come back from the coprocessors, then process
c-------------------------------------------------------------------------
       ibase=1
       do ipe = 0, npes-1
         ntransforms = nint (ncnf_offload/real(npes))
         ! round up to next mutiple of 96
         nrem = mod(ntransforms,96)
         if (nrem.ne.0) ntransforms = ntransforms+(96-nrem)
         if (ibase+ntransforms.gt.ncnf) ntransforms = ncnf-ibase+1

         !print*,ipe, ': ibase,ntransforms=', ibase,ntransforms
         if (ntransforms.gt.0) then
             call CS_get (ipe, 7,  enbuff(ibase),
     +       4*ntransforms, SEM_DONE, ifail)
            ! Optional - fetch the performance timers
            call CS_get (ipe, 10, stats(1,ipe), 4*10, 0, ifail)
         else
            stats(2,ipe)=0.      ! we didn;t use this coproc
         endif
         do i=ibase,ibase + ntransforms-1
            srtval(i) = enbuff(i)   !- take a copy for later ranking
         enddo
        ibase = ibase + ntransforms
      enddo

c optional : Write the perforamnce timings
      write (*,'(a,33f8.3)')
     + 'Timings for concurrent processing host,coproc0,coproc1,..',
     +  (stats(2,ipe),ipe=-1,npes-1)
      !debug : show a selection of results from the beginning, middle and end
      !write (*,'(8E12.4)') (enbuff(i),i=1,6)
      !write (*,'(8E12.4)') (enbuff(i),i=7813-2,7813+2)
      !write (*,'(8E12.4)') (enbuff(i),i=ncnf-6+1,ncnf)
      endif    !- if kroute =1 or 3
```

ClearSpeed™

University of BRISTOL

Advance board code: headers & globals

```
//
// Variables that are exposed to the host to read and write
//
int natlig;
int natpro;
int ntransforms;
int verbose=3;              // set to zero to switch off all printing
int perfprint=0;            // if set to non-zero code will emit performance data
float cutdis=10.;           // tunable distance, that > this we skip all force cals.
float stats[10];            // The number of cases tested and the time taken

// The data structure for one atom - 40 bytes
typedef struct _atom{
    float x,y,z;
    float radius,  hphb, hard, nndst, npdst, elsc;
    char  hbtype, name[3];
} Atom;

#pragma align 32
mono Atom   protein_molecule[80000];
#pragma align 32
mono Atom   ligand_molecule[800];
mono float  transforms[TRANSFORM_BUFFER_SIZE][12];
mono float  etotals[TRANSFORM_BUFFER_SIZE];          // final results

poly Atom   ligand_buffer[LIGAND_SUBSET_SIZE];
poly float  transform[NXB][12];     // n 3x4 transformation matrices per PE

void fasten2 (poly float * mono etotal,
              int natlig, int natpro, float cutdis,
              int xcount, poly float * mono transform);



int main()
{
   //int pbuffer;          // a piece of the protein buffer
   int xbuffer;          // a 4x4 transformation matrix on each PE
   int xcount;           // # of transforms to test at a time (usu = 1, but 4 allows vectorisation)
   int batch;            // counter over the batches of work sent to us from thest
```

ClearSpeed™

University of BRISTOL

Advance Card code Cn: main()

```
// Loop over the set of transformations in chunks of xcount*96
//  where xcount is usualy 1, but can be say 4
   xcount = 1;
   for (ix=0; ix<ntransforms;ix+=(xcount*PE_COUNT))
   {
#ifdef PRINT
      if (verbose>=3) printf("ix=%5d  tr[4]=%7.2f\n", ix, transforms[ix][3]);
#endif
      t0 = get_cycles();

      //  Push the next set of 96*n transformation matrices to the PEs
      // we know async_memcpy is faster than memcpy (2400 MB/s v 300 MB/s)
      dcache_flush();    //make sure no data is in mono cache
      async_memcpym2p(2,&transform[0][0], &transforms[ix+xcount*i_am][0], (short)(xcount*12*sizeof(float)));
      sem_wait(2);

//
// Compute etotal for each transformation
// This involves a loop over every protein atom and every ligand atom
//
      dt=get_cycles();
      fasten2 (etotal, natlig, natpro, cutdis, xcount, &transform[0][0]);
      dt=get_cycles()-dt; t_AA+=dt;
      // Gather the results back to mono
      // no need to block here - could wait until we have a reaonable batch to harvest
      // no need to flush the cache ? data is never touched in mono
      dcache_flush();
      // TODO should be async_ here for better performance. (but alignment ?)
      memcpyp2m (&etotals[ix+xcount*i_am], &etotal[0], (short)(xcount*sizeof(float)));
      t0 = get_cycles()-t0;
      runtime += t0;
    }  // over '960' TRs


    Naa = (float)natlig* (float)natpro * ntransforms * batch;
    stats[0] = Naa;                  // how many AA calcs we did
    stats[1] = runtime/CSCLOCK;      // how long it took
// Signal the host that the results array etotals[] is ready to be collected
// the host will then pull this, send a new set of transformations
// and signals us to process them
    sem_sig(SEM_PROCESS_ATOMS_DONE);
```

ClearSpeed™

University of BRISTOL

Advance Board Code: geometry & energy calculations

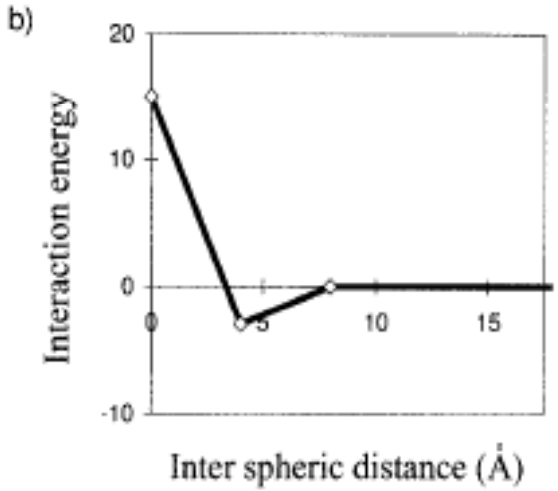
```
void fasten2 (poly float * mono etotal,
   int natlig, int natpro, float cutdis,
   int xcount, poly float * mono transform)
{
  int ilbase,il, ip;                          // ligand and protein index
  //int il_2, ip_2;                          // ligand and protein index
  int ix;                          // Transformation index
  poly Atom * mono ligand_atom;
  Atom * protein_atom;

  poly float etot1, etot2, etot3;      // Total energy - the 3 parts
  poly float strc_e, dslv_e, chrg_e;   // Components of the tot. energy
  poly float radij;                    // sum of 2 atom spheres radii
  poly float distij;                   // distance between 2 atom centres
  poly float distbb;                   // ball:ball dist (=distij - radij)
  poly float elcdst,elcdst1;           // halo distance - 4. or 6. and reciprical

/* ClearSpeed temporary variables */
  int lcount;                          // # of ligand atoms on each PE
  int lcount_close;                    // # of ligand atoms on each PE that are <cutdis away from P a
  poly float distdslv=1.;              // desolvation distance
  poly float const cnstnt=22.5;        // 22.5 factor
  poly char  p_action;                 // =1 if one or both charged/bipolar
  poly char  zone1;                    // which region we are in
  poly float fact;                     // shape function  0.->1.
  poly float p_hphb1,l_hphb1,l_hphb;

  poly const float zero=0., one=1.,half=0.5,four=4.0, six=6.0 ;
  poly float cutdis2;                          // 10*10
// alternative is to keep ligand_buffer local to this routine
//#pragma align 4
//poly Atom   ligand_buffer[LIGAND_SUBSET_SIZE];
  poly float lx[LIGAND_SUBSET_SIZE];   // the xyz of the (16) ligand atoms
  poly float ly[LIGAND_SUBSET_SIZE];   // handled at a time here
  poly float lz[LIGAND_SUBSET_SIZE];
  poly float distij2[LIGAND_SUBSET_SIZE]; // dx^2 + dy^2 + dz^2
  poly int   ligands_close[LIGAND_SUBSET_SIZE];   // flag the 'to do' list

  poly float x,y,z;                      // temporaries
/* the properties of one protein atom */
  poly float p_x, p_y, p_z,              //
             p_radius, p_hphb, p_hard,   // properties of one Protein
             p_nndst, p_npdst, p_elsc;   // atom, in poly memory
  poly char p_hbtype;                    // for speed of access.
```

Advance Board Code: geometry & energy calculations

```
//
// Loop over chunks of ligand molecule that will fit in poly memory (eg 48 atoms)
//
   etot1 = zero; etot2 = zero; etot3 = zero;
   for (ilbase = 0 ; ilbase < natlig; ilbase += LIGAND_SUBSET_SIZE)
   {
      // The final piece of the ligand may contain less that subset number of atoms.
      lcount = natlig - ilbase;
      if (lcount > LIGAND_SUBSET_SIZE) lcount = LIGAND_SUBSET_SIZE;

      // Send the next piece of the ligand
      // TODO replace with
      //   CS_Broadcast(ligand_buffer, ligand_molecule, LIGAND_SUBSET_SIZE*sizeof(struct _atom));
      async_memcpym2p(1,ligand_buffer,&ligand_molecule[ilbase],
                      (short)(LIGAND_SUBSET_SIZE*sizeof(struct _atom)));
      sem_wait(1);

//
// Transformation step
// Here we translate and rotate the ligand atom to its test position
//
      for (il = 0; il < lcount; il++) {
         ligand_atom = &ligand_buffer[il];
         x =   ligand_atom->x;  // or use pointers to save poly memory?
         y =   ligand_atom->y;
         z =   ligand_atom->z;
         // TODO vectorise this transformation
         // Do as 3 loops: x,y,z
         //   get 4 x values, cs_vecMulacc each with tr[0:3]
         lx[il] = x*tr[0] + y*tr[1] + z*tr[2]    + tr[3];
         ly[il] = x*tr[4] + y*tr[5] + z*tr[6]    + tr[7];
         lz[il] = x*tr[8] + y*tr[9] + z*tr[10]   + tr[11];
      }
//
// Loop over all the protein balls
//
   for (ip=0; ip<natpro;ip++) {
      protein_atom = &protein_molecule[ip];

      // Take a copy of this protein atom into poly variables
      p_x      = - protein_atom->x;  // -ve so we can add - might be quicker?
      p_y      = - protein_atom->y;
      p_z      = - protein_atom->z;
// TODO : insert code here to fast reject this atom if it is too far away.
      p_radius = protein_atom->radius;
      p_hphb   = protein_atom->hphb;
      p_hphb_m = protein_atom->hphb;
      p_elsc_m = protein_atom->elsc;
      p_hbtype_m = protein_atom->hbtype;
```

Advance Board Code: geometry & energy calculations

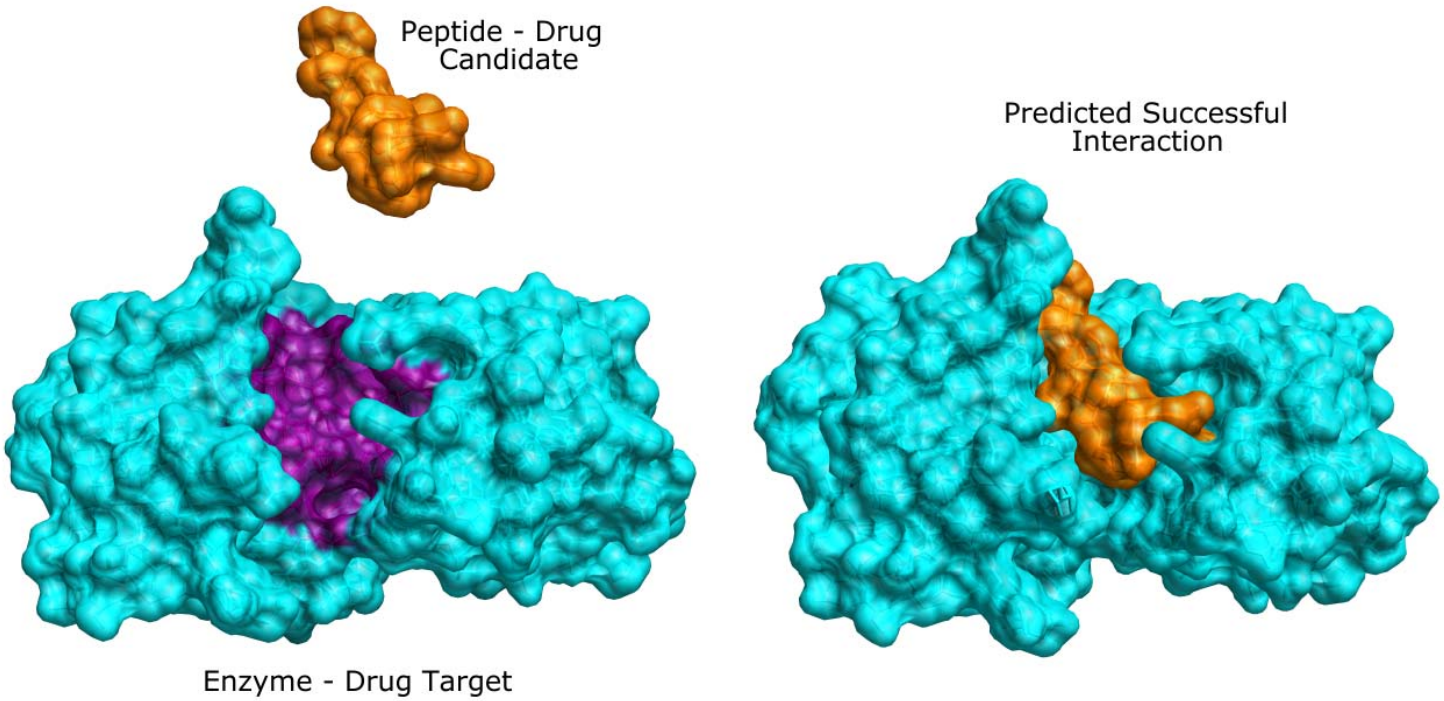
```
      p_action=0;
      l_hphb = ligand_atom->hphb;
      l_hphb1 = l_hphb;
      p_hphb1 = p_hphb;
      if (p_hphb <zero) {
         if (l_hphb<zero ) {
            distdslv = p_nndst;
            p_action=1;
         } else if (l_hphb>zero) {
            distdslv = ligand_atom->npdst;
            p_hphb1 = -p_hphb1;
            p_action=1;
         }
      } else {                              // P must be +ve
         if (l_hphb < zero ) {
            //if (p_hphb > zero) {          // already done this test in mono
               distdslv =  p_npdst;
               l_hphb1 = -l_hphb1;
               p_action=1;
         }
      }

      if (p_action==1) {
         if (distbb < distdslv ) {          // if an interaction
            //dslv_e = half*(p_hphb1+l_hphb1);
            dslv_e = p_hphb1 + l_hphb1;
            if (!zone1) {                   // if in outer zone, scale back
               dslv_e *= (one-distbb/distdslv);
            }
            etot3 += dslv_e;
         }
      }
      } // skip if p_hphb is zero

#endif
            } // skip to point if distij>10
         } // loop ligand atoms subset
      } // loop over protein molecule
   } // loop over ligand moleucle in chunks
   etotal[ix] = half*etot1 +cnstnt*etot2 + half*etot3;

//-----Add this atoms steric, desolvation, and charge energies to the cumulative total 'etot'.
// might want a mask here so we can print each of the 3 values seperately?
      //etotal[ix] += etot1 + etot2 + etot3;

   } // loop trial transformations
   return;
}
```



Peptide Based Elastase Inhibitors: A Case Study

Therapy for Emphysema

Peptide libraries (based on a Trypsin inhibtor)



Flexible amino acid side-chains in both protein (receptor) and ligand (peptide)

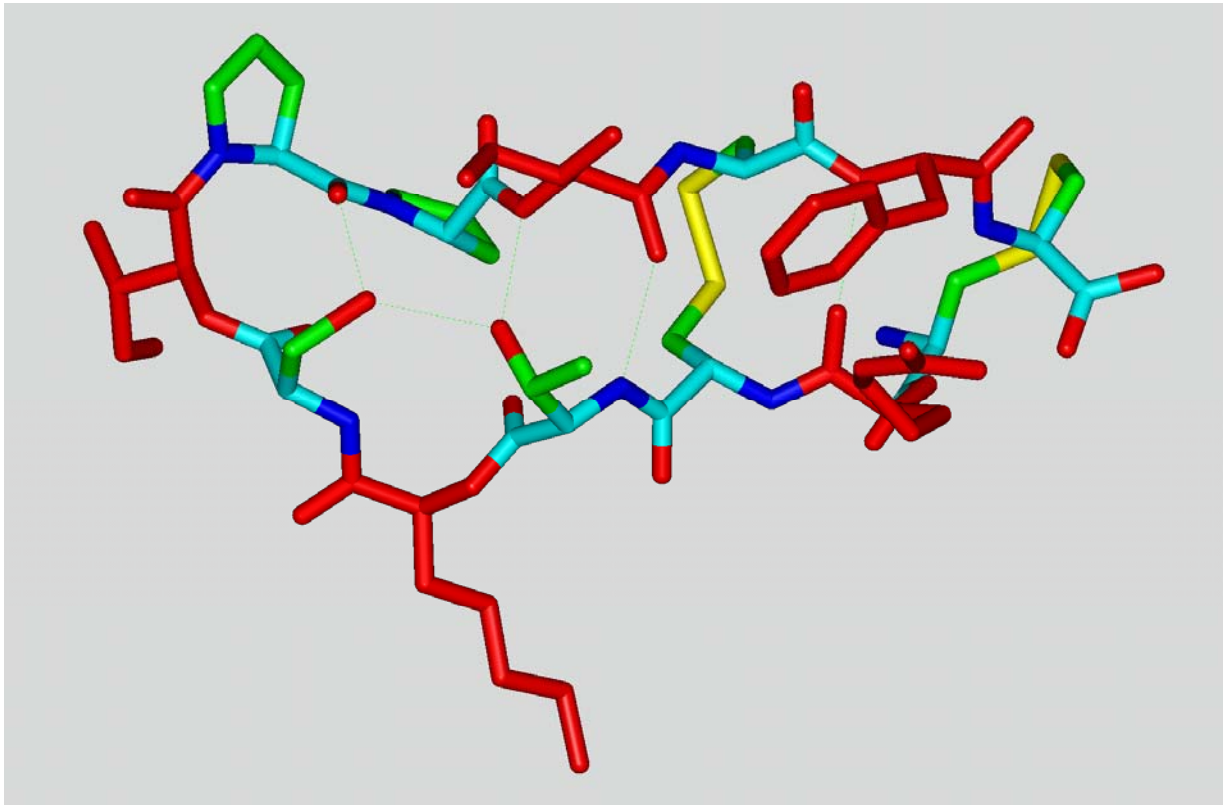
http://www.clearspeed.com/docs/resources/RSBUDE_WhitePaper.pdf

ClearSpeed™

University of BRISTOL

Combinatorial peptide libraries

Putting one of the 20 (natural) amino acids at each of the 5 (red) variable positions gives 20^5 = 3.2 million possible compounds. Chose a library of ~ 1000 peptides



Rotamers – flexible sidechains (protein & peptide)

~2 x10^6 dockings or ~8 x 10^9 pose calculations

ClearSpeed™

University of BRISTOL

Making and testing the peptides

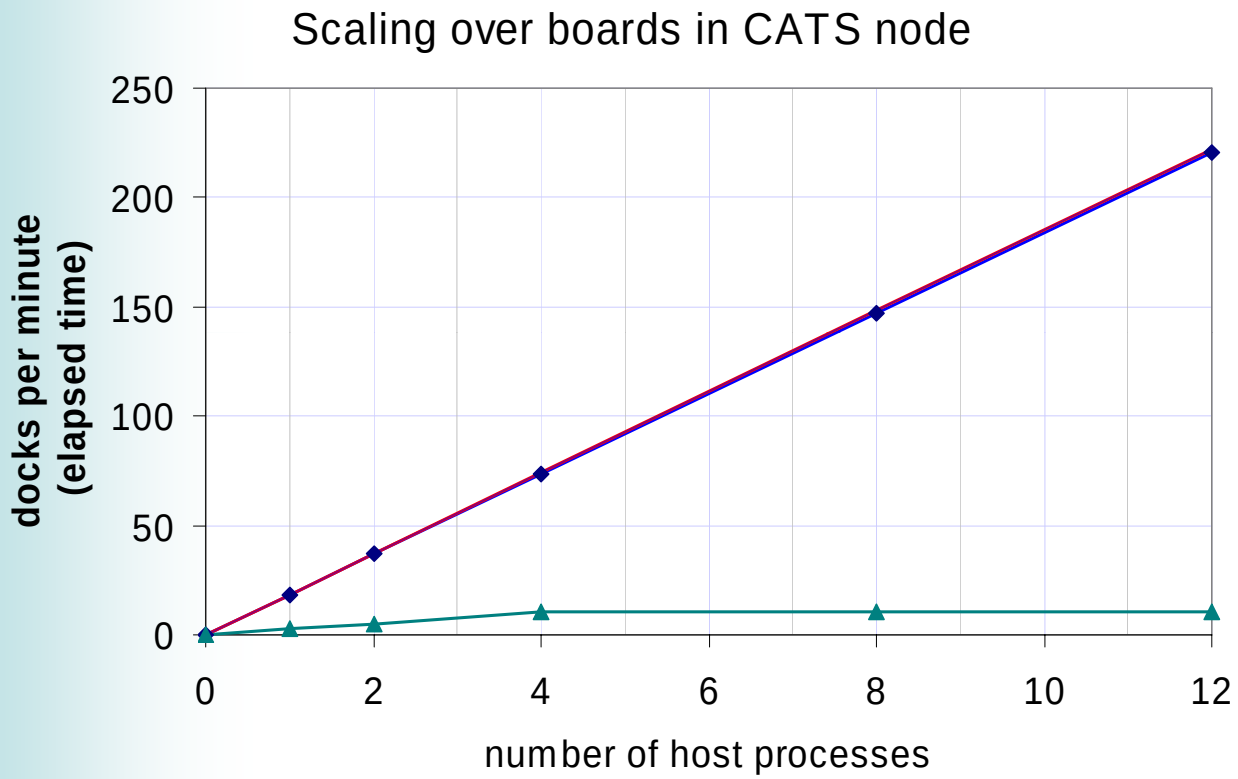
Mixed synthesis of these 40 peptides

Tested for Inhibitory action

Mixture has a K_d 10 μM

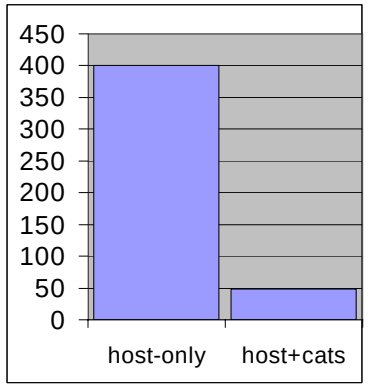
One peptide with K_d between 0.25 μM and 10 μM

Timings and Scaling (multiple jobs)



21 fold speed up over host node

Host load



Accelerated: (1 host process per Advance board): Red – theoretical 100% scaling

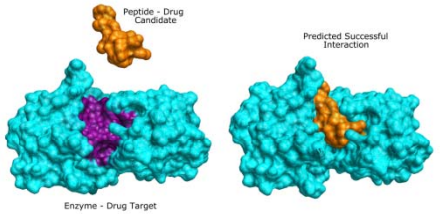
Blue – Actual scaling (99% efficient)

Non-accelerated: (2 x dual core AMD 2.6 GHz) Green

ClearSpeed™

University of BRISTOL

Accelerated BUDE results



- **Scales linearly across multiple CATS nodes**
  - Ran on 10 CATS nodes simultaneously at SC07
- ***21x speedup* per CATS node measured as wallclock time compared to a 2.6 GHz, 2 x dual-core x86 server**
- **A whole Elastase peptide library calculation took 18 hours on ten CATS nodes, compared to the 15 days it would have taken on a 2 x dual-core x86 system of the same size and power consumption**
- **The first set of real peptides based on simulations run on CATS have now been synthesized and show potent elastase inhibition in the laboratory**
- ***This is a real, 64-bit code achieving real-world acceleration and delivering new science***

Acknowledgements

ClearSpeed:

- Ray McConnell
- Simon McIntosh-Smith
- Ken Cameron
- Daniel Kidger
- Tim Lanfear

Bristol University:

- Tony Clarke
- Nick Gibbs
- Emma Compton
- Sophy Smith
- Mike Tyka
- Jon Crisp
- Amaurys Avila Ibarra