OSPREY 3.0: Open-Source Protein Redesign for You, with Powerful New Features

Mark A. Hallen,†[a,c] Jeffrey W. Martin,†[a] Adegoke Ojewole,‡[d] Jonathan D. Jou,‡[a] Anna U. Lowegard,‡[d] Marcel S. Frenkel,[e] Pablo Gainza,[a] Hunter M. Nisonoff,[a] Aditya Mukund,[a] Siyu Wang,[d] Graham T. Holt,[d] David Zhou,[a] Elizabeth Dowd,[a] and Bruce R. Donald†*[a,b,e]

We present OSPREY 3.0, a new and greatly improved release of the OSPREY protein design software. OSPREY 3.0 features a convenient new Python interface, which greatly improves its ease of use. It is over two orders of magnitude faster than previous versions of OSPREY when running the same algorithms on the same hardware. Moreover, OSPREY 3.0 includes several new algorithms, which introduce substantial speedups as well as improved biophysical modeling. It also includes GPU support, which provides an additional speedup of over an order of magnitude. Like previous versions of OSPREY, OSPREY 3.0 offers a unique package of advantages over other design software, including provable design algorithms that account for continuous flexibility during design and model conformational entropy. Finally, we show here empirically that OSPREY 3.0 accurately predicts the effect of mutations on protein–protein binding. OSPREY 3.0 is available at http://www.cs.duke.edu/donaldlab/osprey.php as free and open-source software. © 2018 Wiley Periodicals, Inc.

DOI:10.1002/jcc.25522

Introduction

For over a decade, the OSPREY software package[1–3] has offered the protein design community a unique combination of continuous flexibility modeling, ensemble modeling, and algorithms with provable guarantees.[4,5] Having begun as a software release for the *K** algorithm,[2,6] which approximates binding constants using ensemble modeling, it now boasts a wide array of algorithms found in no other software. OSPREY has been used in many designs that were empirically successful—*in vitro*[6–12] and *in vivo*[7–10] as well as in non-human primates.[7] OSPREY's predictions have been validated by a wide range of experimental methods, including binding assays, enzyme kinetics and activity assays, in cell assays (MICs, fitness) and viral neutralization, *in vivo* studies, and crystal[7,13] and NMR[9] structures.

However, as OSPREY grew to include more algorithms and features (Fig. 1), the code became increasingly complicated and difficult to maintain. The growing complexity of the software also hindered its ease-of-use. OSPREY 3.0 represents a complete refactoring of the code, and presents a simpler and more intuitive interface that makes protein redesign much easier than before. The new, developer-friendly code organization also facilitates adding new features to the free and open-source OSPREY project, both by ourselves and by other contributors. We have introduced a convenient Python scripting interface and added support for GPU acceleration of the bulk of the computation, allowing designs to be completed much more quickly and easily than in previous versions of OSPREY. We believe OSPREY 3.0 will be a very useful tool for both developers and users of provably accurate protein design algorithms.

Past successes of OSPREY

OSPREY has been used for an impressive number of empirically successful designs, ranging from enzyme design to antibody design to prediction of antibiotic resistance mutations. Notably, OSPREY has been successful in many *prospective* experimental studies, that is, studies in which our designed sequences are tested experimentally, thus validating OSPREY through use in practice rather than simply through a retrospective comparison of OSPREY calculations to previous experimental results. OSPREY is most

[a] *M. A. Hallen, J. W. Martin, J. D. Jou², P. Gainza, H. M. Nisonoff, A. Mukund, D. Zhou, E. Dowd, B. R. Donald*
Department of Computer Science, Duke University, Durham, North Carolina, 27708
**E-mail: brd+jcc8@cs.duke.edu*
[b] *B. R. Donald*
Department of Chemistry, Duke University, Durham, North Carolina, 27708
[c] *M. A. Hallen*
Toyota Technological Institute at Chicago, Chicago, Illinois, 60637
[d] *A. Ojewole², A. U. Lowegard², S. Wang, G. T. Holt*
Program in Computational Biology and Bioinformatics, Duke University Medical Center, Durham, North Carolina, 27710
[e] *M. S. Frenkel, B. R. Donald*
Department of Biochemistry, Duke University Medical Center, Durham, North Carolina, 27710
**E-mail: brd+jcc18@cs.duke.edu*
†*These authors contributed equally to this work.*
‡*These authors contributed equally to this work.*

NIH, Grant/Award Numbers: R01 GM-78031 and R01 GM-118543 to B.R.D; NSF (Graduate Research Fellowship to A.O.); PhRMA Foundation (Informatics Predoctoral Fellowships to A.U.L. and M.A.H.); Liebmann Foundation (fellowship to M.A.H.)
Contract Grant sponsor: Dolores Zohrab Liebmann Foundation;Contract Grant number: NIH R01 GM-118543, NIH R01 GM-78031

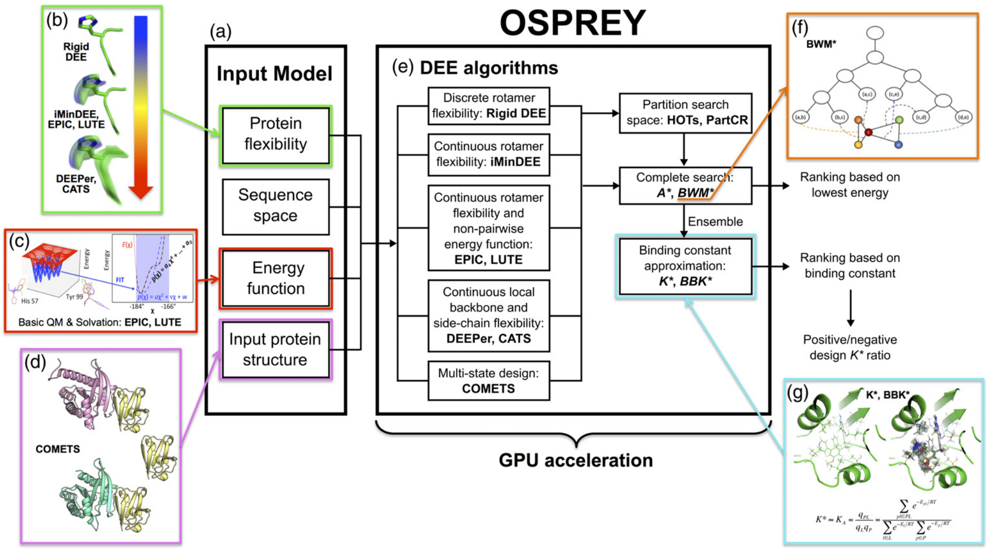
OSPREY
(a)
Input Model
Protein flexibility
Sequence space
Energy function
Input protein structure
(b)
Rigid DEE
iMinDEE, EPIC, LUTE
DEEPer, CATS
(c)
Basic QM & Solvation: EPIC, LUTE
(d)
COMETS
(e) DEE algorithms
Discrete rotamer flexibility: Rigid DEE
Continuous rotamer flexibility: iMinDEE
Continuous rotamer flexibility and non-pairwise energy function: EPIC, LUTE
Continuous local backbone and side-chain flexibility: DEEPer, CATS
Multi-state design: COMETS
Partition search space: HOTs, PartCR
Complete search: A*, BWM*
Ensemble
Binding constant approximation: K*, BBK*
Ranking based on lowest energy
Ranking based on binding constant
Positive/negative design K* ratio
GPU acceleration
(f)
BWM*
(g)
K*, BBK*

Figure 1. The OSPREY protein redesign suite. a) The input model includes a 3D structure of the protein to be redesigned, a definition of the sequence space, the allowed protein flexibility (including the rotamer library), and a pairwise energy function. b) Rigid DEE,[14,15] iMinDEE,[16] EPIC,[17] LUTE,[18] DEEPer,[19] and CATS[20] model different types of protein flexibility. Flexibility ranges in complexity from discrete, rigid rotamers to continuous side chain flexibility to complete flexibility including continuous backbone flexibility. c) The EPIC[17] and LUTE[18] algorithms also expand energy function capability by allowing for nonpairwise, basic quantum chemistry and Poisson–Boltzmann solvation. d) COMETS [21] allows for multistate design by optimizing sequences and conformations for user-specified bound and unbound states. This is accomplished using multiple input structures. e) These algorithms are implemented in OSPREY and improved through the use of GPU acceleration. According to the allowed flexibility, OSPREY runs a specific pruning algorithm followed by a highly optimized descendant of the A^* search algorithm.[22] The A^* output generates a ranking based on either the lowest-energy structure of each sequence, or an ensemble of structures computed by the K^* algorithm. f) The BWM*[23] algorithm exploits sparse residue interaction graphs and branch decomposition to outperform traditional A^*. g) The K^*[2,24] algorithm calculates a K^* score (an approximation of the binding constant, K_a) by provably estimating the partition function for the protein, the ligand, and the protein–ligand complex. The K^* algorithm exploits a thermodynamic ensemble of structures as opposed to a single structure, as illustrated in the panel (PDB ID: 3FQC). K^* can also be used to find sequences that have a high affinity for one ligand (positive design) while having a low affinity for another (negative design) by taking a ratio of K^* scores.[10,13] [Color figure can be viewed at wileyonlinelibrary.com]

applicable to problems that can be posed in terms of biophysical state transitions like binding, allowing the K^* algorithm and its variants to predict the optimal sequences based on an estimate of binding free energy computed using Boltzmann-weighted conformational ensembles. Moreover, most protein design problems can be posed in this way, sometimes in terms of binding to more than one ligand. OSPREY is capable of both *positive design*, in which binding of a designed protein to a target is increased, and *negative design*, in which binding to a target is decreased, as well as more complicated design objectives where specific binding to one target and not to another is required.

For example, we have successfully predicted novel resistance mutations to new inhibitors in MRSA (methicillin-resistant *Staphylococcus aureus*) using multistate design (combining negative and positive design). OSPREY does this by searching for sequences that have impaired drug binding compared to wild-type DHFR, but still form the enzyme-substrate complex as usual, allowing catalysis to proceed.[10,13] Our predictions were validated not only biochemically and structurally, but also at an organismal level.[13,25,26] Similarly, we have successfully changed the preferred substrate of an enzyme—the phenylalanine adenylation domain of gramicidin S synthetase A—from phenylalanine to leucine by modeling the two enzyme-substrate complexes, and searching for sequences with improved binding to leucine and reduced binding to phenylalanine.[6] The resulting designer enzymes exhibited improved catalysis, and designs changing the specificity from phenlyalanine to several charged amino acids were successful as well.[6] The combination of positive and negative design in OSPREY has also successfully designed mutants of the gp120 surface protein of HIV that bind specifically to particular classes of antibodies, enabling their use as probes for detecting and isolating those antibodies from human sera.[12]

These multistate design capabilities, long a mainstay of OSPREY, are accelerated by the modules *BBK** (described below) and COMETS (described in Ref. 21). COMETS provably returns the sequence that minimizes any desired linear combination of the energies of multiple protein states, subject to constraints on other linear combinations. Thus, COMETS can target nearly any combination of affinity (to one or multiple ligands), specificity, and stability (for multiple states if needed). COMETS and *BBK** have been integrated into OSPREY 3.0 and accelerated, and they are currently the only provable multi-state design algorithms that run in time sublinear in the size M of the sequence space. This can be important, since M is exponential in the number of simultaneously mutable residue positions.

Further successes of OSPREY have involved improving positive design, for example, the interaction of the anti-HIV antibody VRC07 with its antigen, gp120. Using this approach, we collaborated with the NIH Vaccine Research Center to design a broadly neutralizing antibody (VRC07-523LS) against HIV with unprecedented breadth and potency that is now in clinical trials (Clinical Trial Identifiers: NCT03015181, NCT02840474, NCT03565315, and NCT03387150[7,27]). We also have designed allosteric inhibitors of the leukemia-associated protein–protein interaction between Runx1 and CBFβ.[9] Similarly, we have used OSPREY to develop peptide inhibitors of CAL, a protein involved in cystic fibrosis.[8] The CBFβ and CAL inhibitors were successful *in vitro* and *in vivo*.[8,9]

In addition, a number of other research groups have successfully used the OSPREY algorithms and software (by themselves) to perform biomedically important protein designs, for example, to design anti-HIV antibodies that are easier to induce[28]; to design a soluble prefusion closed HIV-1-Env trimer with reduced CD4 affinity and improved immunogenicity[29]; to design a transmembrane Zn^{2+}-transporting four-helix bundle[30]; to optimize stability and immunogenicity of therapeutic proteins[31–33]; and to design sequence diversity in a virus panel and predict the epitope specificities of antibody responses to HIV-1 infection.[34]

We believe OSPREY 3.0 will enable an even greater range of successful designs.

Performance Enhancements in OSPREY 3.0

Engineering improvements yield large single-threaded speedups

OSPREY 3.0's code has been heavily optimized to improve single-threaded performance relative to the previous version, OSPREY 2.2.[21] Two main areas have received the most attention and the most improvement in performance so far: *A** search speed, and conformation minimization speed.

OSPREY uses the *A** search algorithm[15] to perform its combinatorial search over sequence and conformational space.[2,16,19] The performance of *A** search in OSPREY depends mostly on the size of the conformation space of the design: the time required for search scales strongly with the number of mutable and flexible residues. Search time is also dependent on the speed at which we can evaluate the energy scoring functions on *A** nodes. Optimizations in OSPREY 3.0 have dramatically increased the *A** node scoring speed, mainly by caching the results of expensive computations and reusing them at different nodes. Many intermediate values used by the *A** scoring functions need only be computed once per design. This reduces the cost of node scoring by roughly an order of magnitude. We can also score child nodes differentially against their parent nodes to speed up node scoring. Caching intermediate values during the parent node scoring and using them to simplify child node scoring yields roughly another order of magnitude speedup in *A** node scoring.

OSPREY 3.0 also includes optimizations to improve the performance of forcefield evaluation and conformation minimization. Conformation minimization is typically the bottleneck in OSPREY calculations with continuous flexibility.[2,16,19,20] The code in OSPREY 3.0 that evaluates forcefield energies for a protein conformation has been heavily optimized, although speed gains here over OSPREY2 are modest (roughly twofold), since the original code was already well-optimized in this area. Much larger performance increases were gained by caching forcefield parameters and lists of atom pairs between different conformations to be minimized, which yielded roughly a 10-fold increase in speed. OSPREY 3.0 also increases performance by only evaluating forcefield terms involving mutable and/or flexible residues in a design, since interaction energies between other residues will be identical across all sequences and conformations. Since most designs only model a minority of the residues in a protein as flexible, this can be a substantial improvement.

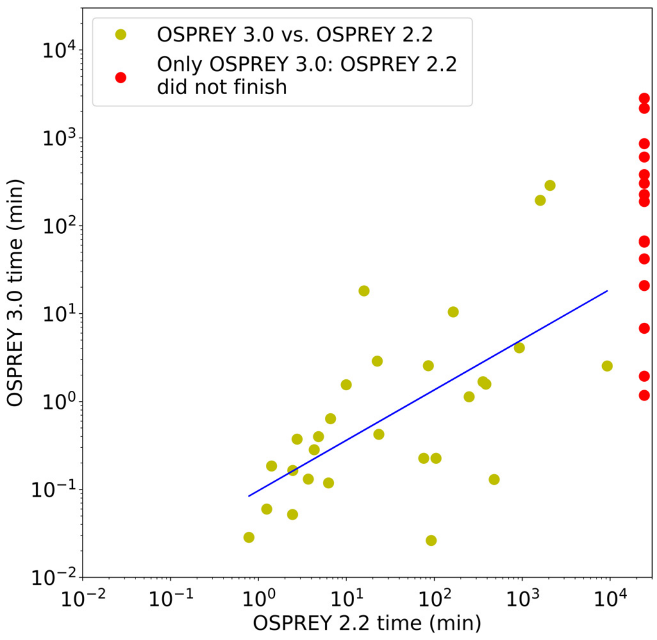

Figure 2. Runtimes of OSPREY 2.2 versus OSPREY 3.0 for 45 protein design test cases (details shown in Table 1), shown on a log scale. Designs that only finished with OSPREY 3.0 (given a 17-day time limit) are shown on the right in red. All test cases involve continuous flexibility[2,16] and minimization-aware DEE[16,17]; 18 involve provably accurate partition function calculations (see Table 1 and Ref. 17 for details). [Color figure can be viewed at wileyonlinelibrary.com]

Performance comparisons are shown for 45 protein design test cases in Figure 2 and Table 1. All these test cases model continuous protein flexibility,[2,16,17] and 18 of them involve provably accurate partition function calculations (see Table 1 and Ref. 17 for details). To summarize, the optimizations to single-threaded performance described above made OSPREY 3.0 on average 461-fold faster than OSPREY 2.2 across 29 protein design test cases, and allowed OSPREY 3.0 to finish the remaining 16 test cases, which OSPREY 2.2 could not finish within a 17-day time limit. For example, OSPREY 2.2 on a Intel Xeon E5–2640 v4 CPU took 49.5 min to run a small (six continuously flexible residues) benchmark sidechain packing problem involving a 114-residue fragment of PDZ3 domain of PSD-95 protein complexed with a 6-residue peptide ligand (PDB ID: 1TP5). But OSPREY 3.0 finished the same design in 7.0 s on the same hardware, which is a 424-fold speedup.

GPU acceleration reduces design runtimes

One of the key challenges in protein design is modeling and searching the many continuous conformational degrees of freedom inherent in proteins and other molecules. The sidechain conformations of each amino-acid type are generally found in clusters, known as rotamers,[35] so it is common practice to approximate protein conformational space as discrete by

Table 1. Details of 45 protein design test cases with continuous flexibility run on both OSPREY 2.2 and OSPREY 3.0.

Protein name	PDB code	PF?[a]	Mutable residue count	OSPREY 3.0 time (min)	OSPREY 2.2 time (min)	Speedup
Scorpion toxin	1AHO	N	7	0.37	2.75	7.38
Scorpion toxin	1AHO	N	9	0.64	6.60	10.35
Scorpion toxin	1AHO	N	12	194.71	1608.16	8.26
Scorpion toxin	1AHO	N	14	287.87	2075.04	7.21
Cytochrome c553	1C75	N	6	0.28	4.30	15.19
Atx1 metallochaperone	1CC8	N	7	2.56	85.41	33.41
Atx1 metallochaperone	1CC8	Y	7	67.12	DNF	>364.72[b]
Bucandin	1F94	N	7	0.40	4.82	12.07
Nonspecific lipid-transfer protein	1FK5	N	6	0.03	0.78	27.34
Transcription factor IIF	1I27	N	7	1.58	385.56	244.4
Ferredoxin	1IQZ	N	9	0.16	2.45	14.92
Trp repressor	1JHG	N	7	2.88	22.50	7.8
Fructose-6-phosphate aldolase	1L6W	N	6	0.23	75.97	336.22
Cephalosporin C deacetylase	1L7A	N	8	4.09	928.27	226.93
PA-I lectin	1L7L	N	6	0.12	6.26	52.85
Phosphoserine phosphatase	1L7M	N	7	1.13	249.33	220.11
alpha-D-glucuronidase	1L8N	N	5	0.13	480.13	3701.36
Dachshund	1L8R	N	8	0.19	1.41	7.62
Granulysin	1L9L	N	7	0.06	1.24	20.8
gamma-glutamyl hydrolase	1L9X	N	5	0.03	92.13	3507.46
Ferritin	1LB3	N	5	0.42	23.42	55.2
Cytochrome c	1M1Q	N	8	1.68	357.59	213.09
Hypothetical protein YciI	1MWQ	N	8	0.13	3.69	28.13
ygfY	1X6I	Y	14	604.71	DNF	>40.48[b]
ADAR1 ZB domain	1XMK	Y	15	2172.23	DNF	>11.27[b]
Histidine triad protein	2CS7	Y	14	2816.56	DNF	>8.69[b]
Transcriptional regulator AhrC	2P5K	Y	11	1.18	DNF	>20811.61[b]
Scytovirin	2QSK	N	10	10.47	164.49	15.71
Scytovirin	2QSK	Y	10	2.54	9267.19	3651.9
Hemolysin	2R2Z	Y	12	42.02	DNF	>582.58[b]
Putative monooxygenase	2RIL	N	8	18.16	15.89	0.87
Putative monooxygenase	2RIL	Y	8	0.23	104.77	463.18
alpha-crystallin	2WJ5	Y	15	226.32	DNF	>108.17[b]
Cytochrome c555	2ZXY	Y	14	381.39	DNF	>64.19[b]
High-potential iron–sulfur protein	3A38	Y	13	65.15	DNF	>375.72[b]
ClpS protease adaptor	3DNJ	Y	12	65.04	DNF	>376.4[b]
Putative monooxygenase	3FGV	Y	10	1.94	DNF	>12591.94[b]
Protein G	3FIL	Y	14	303.81	DNF	>80.58[b]
Viral capsid	3G21	Y	15	188.53	DNF	>129.85[b]
dpy-30-like protein	3G36	N	4	1.55	9.97	6.43
dpy-30-like protein	3G36	Y	4	0.05	2.44	47.07
Hfq protein	3HFO	Y	10	6.81	DNF	>3594.09[b]
Cold shock protein	3I2Z	Y	14	20.84	DNF	>1174.8[b]
HPI integrase	3JTZ	Y	14	859.69	DNF	>28.48[b]
PSD-95 PDZ3 domain	1TP5	N	6	0.12	49.50	424.29

Test cases primarily adapted from Ref. 17.
[a] Y indicates a partition function calculation (a subroutine of the *K** algorithm [2,24]), which analyzes a thermodynamic ensemble of conformations; N indicates calculation of the single global minimum-energy conformation (GMEC). DNF: Did not finish.
[b] OSPREY 2.2 did not finish within the time limit, so we report a lower bound on the speedup: the ratio of the time limit (17 days) to the OSPREY 3.0 runtime

forcing each residue to be in the modal conformation of one of these clusters.[14,15] However, design accuracy is increased significantly when continuous flexibility is taken into account, by allowing the continuous degrees of freedom to move within finite bounds around these modal values.[1,16,19,36] Moreover, this increase in accuracy depends on considering continuous flexibility *during* the conformational search process, rather than simply performing minimization *post hoc* on the top-scoring sequences and conformations output by a discrete search algorithm. Although such a *post hoc* minimization approach would obtain more energetically favorable models of the top sequences, it would still produce the same top sequences as a purely discrete design would, which have been shown to not be truly the top sequences, even if a much finer discrete rotamer subsampling is allowed.[1,16] For example, clashing discrete rotamers can often be converted to favorable conformations by relatively small adjustments in the sidechain conformations.[2,16,19,20] As a result, designs performed with continuous flexibility taken into account *throughout the search* yield significantly different, and more biologically accurate, sequences than the same designs performed using discrete search.[1,16,19]

To address this problem, OSPREY includes several algorithms to design proteins while taking continuous flexibility into account throughout the process of sequence and conformational search.[2,16–20] These algorithms predict optimal protein sequences with provable guarantees of accuracy given a biophysical model that includes continuous flexibility.

This *minimization-aware* design approach requires energy minimization to be performed for a large number of conformations (within the bounds on the continuous degree of freedom that define each conformation). This minimization is a relatively expensive operation, so the bulk of a design's runtime can be spent on energy minimization of conformations. Therefore, improvements to the speed of energy minimization can have a dramatic impact on OSPREY runtimes.

Much work has been done to optimize OSPREY for execution on CPUs, particularly highly multi-core CPUs and even networked clusters of CPU-powered servers.[37,38] However, modern GPU hardware enables high-performance computation for some specific tasks at a fraction of the cost of large CPU clusters, mainly due to the huge video game industry, which propels innovation in hardware design and drives down costs. The widespread adoption of fast and highly programmable GPUs in the past decade has transformed many areas of computational science, including quantum chemistry,[39] computer vision,[40] and cryptography.[41] In particular, GPUs have been found to produce speedups of approximately an order of magnitude in molecular dynamics simulations,[42–44] which, like OSPREY, must sum huge numbers of forcefield energy terms and can use the GPU to parallelize this computation. GPUs have also been used to accelerate the A^* search step of protein design,[45] albeit without addressing the continuous minimization bottleneck.

Thus, to bring the benefit of GPUs to continuously flexible protein design calculations, OSPREY 3.0 includes GPU programs (called *kernels*) built using the CUDA framework[46] that implement the forcefield calculations and local minimization algorithms used in protein redesign.

We present performance results of these GPU kernels on various hardware platforms in Figure 3. A GPU server housing 4 Nvidia Tesla P100 cards can finish minimizations with about 300,000 atom pairs roughly 110-fold faster than a single thread running on an Intel Xeon E5–2640 v4 CPU. With two Intel Xeon E5–2640 v4 CPUs running at full capacity with multiple threads, the four Nvidia Tesla P100 GPUs finish the same minimizations roughly eightfold faster. The speedups of GPUs over CPUs scale with the number of atom pairs in the minimization. For minimizations with fewer (about 30,000) atom pairs, even four Nvidia Tesla P100 GPUs cannot outperform two Intel Xeon E5–2640 v4 CPUs. There is significant overhead to transfer each minimization problem from the CPU to the GPU during designs. Even though GPUs can evaluate the minimizations much faster than CPUs, when there are few atom pairs, this transfer overhead dominates the computation time and causes GPUs to perform merely similarly to CPUs, rather than significantly faster. Nevertheless, the bottleneck in protein design is minimizations with many atom pairs, and for these minimizations OSPREY's speedups on GPUs are on par with the state of the art for GPU speedups of molecular dynamics simulations.

The performance of desktop hardware appears similar to server hardware, except on a smaller scale. A single Nvidia GTX 1070 GPU performs minimizations at roughly half the speed of an Nvidia Tesla P100 GPU. Two Nvidia GTX 1080 GPUs perform similarly to the Nvidia Tesla P100 GPU on the large conformation benchmark (Fig. 3, bottom), but actually perform worse than a single Nvidia GTX 1070 for the small conformation benchmark (Fig. 3, middle)—despite having well over twice the hardware of the single Nvidia GTX 1070 GPU. This anomalous performance suggests the kernel OSPREY 3.0 uses for minimizations is not yet well-optimized for the Nvidia GTX 1080 GPU, and that future engineering efforts could offer significant performance increases for Nvidia GTX 1080 GPUs. The Nvidia GTX 1050, a laptop GPU, does not appear to be powerful enough to offer any advantages over traditional CPU computing in OSPREY 3.0 (Fig. 3, light blue columns).

Modern GPU architectures offer thousands of parallel hardware units for calculations, compared to the tens of parallel hardware units in modern CPU architectures. The performance results of the current generation of OSPREY's GPU kernels indicate that minimization speeds on GPUs have only begun to scratch the surface of what is possible, particularly for minimizations with few atom pairs. Future versions of these GPU kernels will likely offer significantly higher performance on the same hardware—perhaps allowing minimization speeds many times faster than today's GPU kernels. This in turn will make it even more efficient to perform minimization-aware protein design, and allow minimization-aware designs with even more mutable and flexible residues and with more mutation options per residue.

Python Scripting Improves Ease-of-Use

One of the most visible additions to OSPREY 3.0 is the Python application programming interface (API), which allows fine-grained control over design parameters in a streamlined and easy-to-use experience. OSPREY 3.0 still supports a command-line interface with configuration files for backwards compatibility, but new development will be focused mostly on the new Python interface.

The OSPREY 3.0 distribution contains a Python module, which is installed using the popular package manager PIP. Once installed, using OSPREY 3.0 is as easy as writing a Python script. High-performance computations are still performed in the Java virtual machine to give the fastest runtimes, so Java is still required to run OSPREY 3.0, but communication between the Python environment and the Java environment is handled behind-the-scenes, and OSPREY 3.0 still looks and feels like a regular Python application.

See Figure 4 for a complete example of a Python script that performs a very simple design using OSPREY 3.0, and Figure 5 for a slightly more involved design using *BBK**[36] (a new algorithm in OSPREY 3.0, described in its own section below). Figure 6 graphically displays the design setup for the *BBK** design.

New Protein Design Algorithms in OSPREY 3.0

LUTE: Putting advanced modeling into a form suitable for efficient, discrete design calculations

OSPREY 3.0 comes with LUTE,[18] a new algorithm that addresses two issues with previous versions of OSPREY.

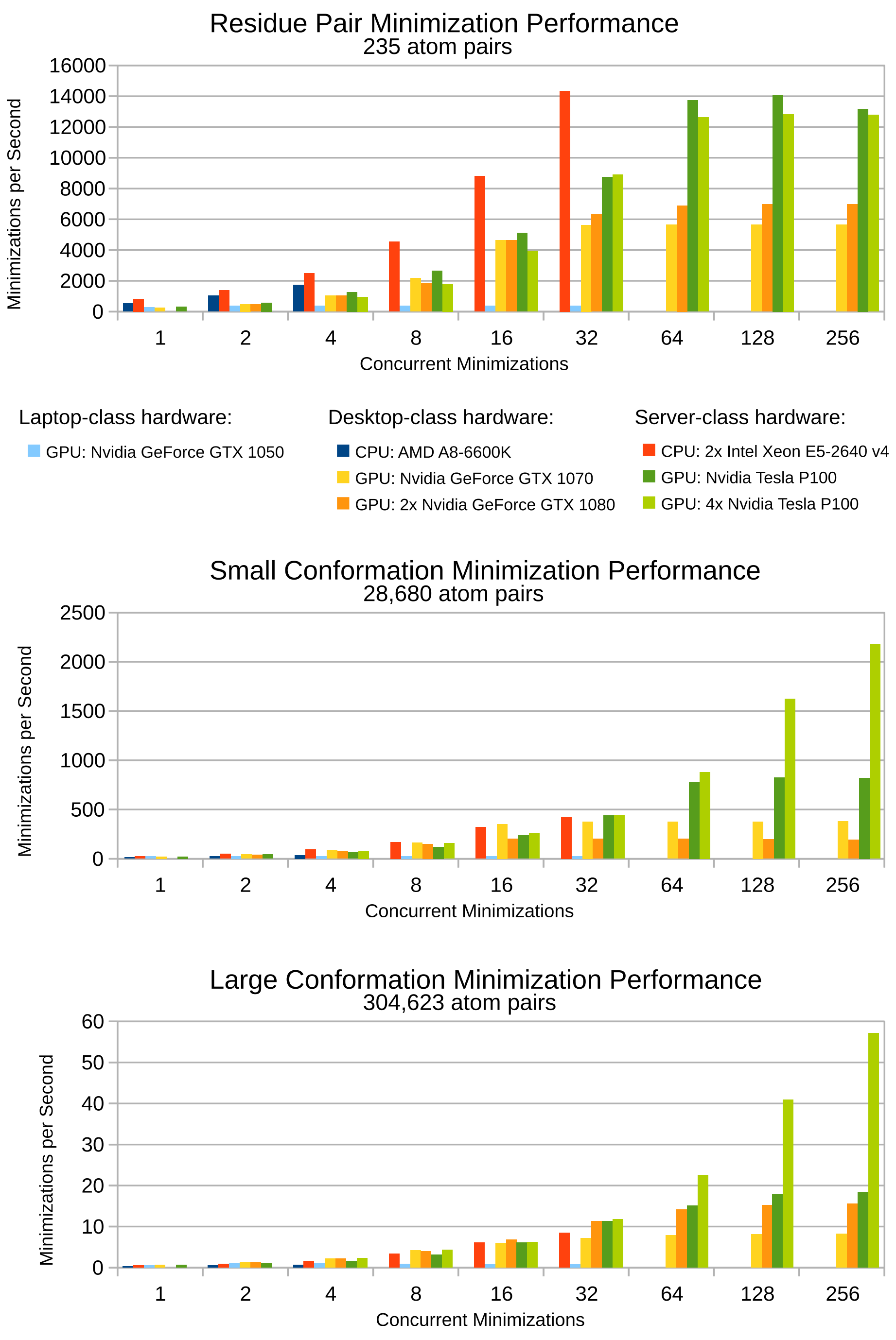

Figure 3. Benchmarks for protein conformation minimization in OSPREY 3.0 for various hardware platforms and for conformations of varying size. From smallest to largest: (top) a single residue pair is the smallest multi-body minimization possible, (middle) a full protein conformation with a single flexible residue represents a small design, (bottom) a full protein conformation with 20 flexible residues represents a large design. For CPU hardware, concurrent minimizations correspond to CPU threads. For GPU hardware, concurrent minimizations correspond to *streams* defined by the CUDA framework. Faster minimization speeds correspond with faster OSPREY runtimes. All minimizations were performed on the Atx1 metallochaperone protein (PDB ID: 1CC8).[47] Flexible residues were modeled with continuous sidechain flexibility, and all other residues remained completely fixed. [Color figure can be viewed at wileyonlinelibrary.com]

First, previous versions modeled continuous flexibility by enumerating conformations in order of a *lower bound* on minimized conformational energy.[2,16] This lower bound can be relative loose, especially for larger systems, and thus a large number of suboptimal conformations—often exponentially many with respect to the size of the system—must be scored by continuous minimization merely because they have favorable lower bounds on their energy. LUTE addresses this problem by enumerating conformations in order of their actual minimized conformational energies instead of simply in order of a lower bound. These energies are estimated using an expansion in low-order tuples of residue conformations. Thus, the burden of modeling continuous flexibility is shifted from the combinatorial optimization (A^*) step, which has unfavorable asymptotic complexity, to a precomputation step (the “LUTE matrix precomputation”[18]) that only scales quadratically with the number of residues. This dramatically reduces the computation time for large designs with continuous flexibility, and has doubled the number of residues that can be treated simultaneously with continuous flexibility.[18]

Second, all previous combinatorial protein design algorithms have relied on an explicit decomposition of the energy as a sum of local (e.g., pairwise) terms. This made design with energy functions that do not have this form difficult. LUTE can straightforwardly support general energy functions, and, as shown in Ref. 18, it can obtain good fits at least in the case of Poisson–Boltzmann

```
import osprey

osprey.start()

# define a strand
strand = osprey.Strand('1CC8.ss.pdb')
strand.flexibility['A2'].setLibraryRotamers('ALA', 'GLY')
strand.flexibility['A3'].setLibraryRotamers(osprey.WILD_TYPE, 'VAL')
strand.flexibility['A4'].setLibraryRotamers(osprey.WILD_TYPE)

# make the conf space
confSpace = osprey.ConfSpace(strand)

# choose a forcefield
ffparams = osprey.ForcefieldParams()

# how should we compute energies of molecules?
ecalc = osprey.EnergyCalculator(confSpace, ffparams)

# how should we define energies of conformations?
confEcalc = osprey.ConfEnergyCalculator(confSpace, ecalc)

# how should confs be ordered and searched?
emat = osprey.EnergyMatrix(confEcalc)
astar = osprey.AStarMPLP(emat, confSpace)

# find the best sequence and rotamers
gmec = osprey.GMECFinder(astar, confEcalc).find()
```

Figure 4. A Python script that performs a very simple design in OSPREY 3.0. The design searches over sequences in which residues A2 and/or A3 of the Atx1 metallochaperone protein (PDB ID: 1CC8) [47] are mutated; residues A2-A4 (i.e., residue 2–4 of chain A) are all modeled with sidechain flexibility, consisting of a discrete search over the Penultimate rotamer library[48]'s rotamers for the specified amino acid types. The mutability, flexibility, and starting crystal structure are all specified in the "define a strand" section of the code. Advanced users can also modify the other sections to specify changes from the default search algorithms, energy function, and other modeling assumptions. This script uses the Max Product Linear Programming (MPLP) algorithm[49] to speed up the A* search[15] employed for sequence and conformational search without compromising accuracy; see Ref. 22 for details.

energies. Moreover, once the LUTE matrix precomputation is completed, the time cost of finding the optimal sequence and conformation does not depend on the energy function used. This is an enormous advantage for more expensive and accurate energy functions like Poisson–Boltzmann, which otherwise would be far too expensive for all but the smallest designs.

OSPREY users can now turn on LUTE for continuously flexible calculations simply by setting a boolean flag (in the **`DEEGMECFinder`** Python constructor). OSPREY 3.0 also supports design with Poisson–Boltzmann solvation energy calculations, which call the DelPhi[51,52] software for the single-point Poisson–Boltzmann calculations (we ask the user to download DelPhi separately for licensing reasons). Such improved modeling is essential to increasing the reliability of and range of feasible uses for computational protein design.

CATS: Local backbone flexibility in all biophysically feasible dimensions

OSPREY pioneered protein design calculations that model local continuous flexibility of sidechains in the vicinity of rotamers in all biophysically feasible dimensions (i.e., the sidechain dihedrals). This continuous flexibility was often critical in correctly predicting energetically favorable sequences,[1,16] especially when those sequences falsely appeared to be sterically clashing when modeled using only rigid rotameric conformations taken from a rotamer library (see section on GPU acceleration above for more details). In OSPREY 3.0, we now extend this ability to the backbone: allowing local continuous backbone flexibility in the vicinity of the native backbone with respect to all biophysically feasible degrees of freedom.

This flexibility is enabled by the CATS algorithm[20] (Fig. 7). CATS uses a new parameterization of backbone conformational space, along with the voxel framework that OSPREY has always included. It is equivalent to searching over all changes in backbone dihedrals (ϕ and ψ) subject to keeping the protein conformation constant outside of a specified flexible region. CATS includes an efficient Taylor series-based algorithm for computing atomic coordinates from its new degrees of freedom, enabling efficient energy minimization. Unlike previous protein design algorithms with backbone flexibility, CATS routinely finds backbone motions on the order of an angstrom (in RMSD

```
import osprey
osprey.start()

# choose a forcefield
ffparams = osprey.ForcefieldParams()

# read a PDB file for molecular info
mol = osprey.readPdb('2RL0.min.reduce.pdb')

# make sure all strands share the same template library (including wild-type rotamers)
templateLib = osprey.TemplateLibrary(ffparams.forcefld, moleculesForWildTypeRotamers=[mol])

# define the protein strand
protein = osprey.Strand(mol, templateLib=templateLib, residues=['G648', 'G654'])
protein.flexibility['G649'].setLibraryRotamers(osprey.WILD_TYPE, 'TYR', 'ALA', 'VAL', 'ILE', 'LEU') \
        .addWildTypeRotamers().setContinuous()
for res in ['G650','G651','G654']:
        protein.flexibility[res].setLibraryRotamers(osprey.WILD_TYPE).addWildTypeRotamers().setContinuous()

# define the ligand strand
ligand = osprey.Strand(mol, templateLib=templateLib, residues=['A155', 'A194'])
for res in ['A156','A172','A192','A193']:
        ligand.flexibility['A156'].setLibraryRotamers(osprey.WILD_TYPE).addWildTypeRotamers().setContinuous()

# make the conf space for the protein, ligand, and complex
proteinConfSpace = osprey.ConfSpace(protein)
ligandConfSpace = osprey.ConfSpace(ligand)
complexConfSpace = osprey.ConfSpace([protein, ligand])

# how should we compute energies of molecules?
# (give the complex conf space to the ecalc since it knows about all the templates and degrees of freedom)
parallelism = osprey.Parallelism(cpuCores=4)
minimizingEcalc = osprey.EnergyCalculator(complexConfSpace, ffparams, parallelism=parallelism, isMinimizing=True)

# BBK* needs a rigid energy calculator too, for multi-sequence bounds on K*
rigidEcalc = osprey.SharedEnergyCalculator(minimizingEcalc, isMinimizing=False)

# how should we define energies of conformations?
def confEcalcFactory(confSpace, ecalc):
        eref = osprey.ReferenceEnergies(confSpace, ecalc)
        return osprey.ConfEnergyCalculator(confSpace, ecalc, referenceEnergies=eref)

# how should confs be ordered and searched?
def astarFactory(emat, rcs):
        return osprey.AStarMPLP(emat, rcs, numIterations=5)

# run K*
bbkstar = osprey.BBKStar(proteinConfSpace, ligandConfSpace, complexConfSpace, rigidEcalc, minimizingEcalc, confEcalcFactory,
        astarFactory, numBestSequences=2, epsilon=0.99, energyMatrixCachePattern='emat.*.dat',
        writeSequencesToConsole=True, writeSequencesToFile='bbkstar.results.tsv'
)
scoredSequences = bbkstar.run()

# use results
for scoredSequence in scoredSequences:
        print("result:")
        print("\tsequence:␣%s" % scoredSequence.sequence)
        print("\tscore:␣%s" % scoredSequence.score)
```

Figure 5. A Python script that performs a simple *BBK** design in **OSPREY 3.0.** This design produces a peptide to bind human fibronectin (the "ligand strand," i.e., chain A) by optimizing a fragment of the protein FnBPA from *Staphylococcus aureus* (the "protein strand," chain G), which has been crystallized in complex with fibronectin domains (PDB ID: 2RL0[50]). As in Figure 4, the script defines the starting crystal structure, mutable residues, and level of mutability and flexibility (here including continuous flexibility) in the form of Python strand objects. Figure 6 represents this design graphically. This design is accelerated by parallelism, running on 4 CPU cores. This example thus shows it is easy to invoke and use parallelism within the OSPREY 3.0 software.

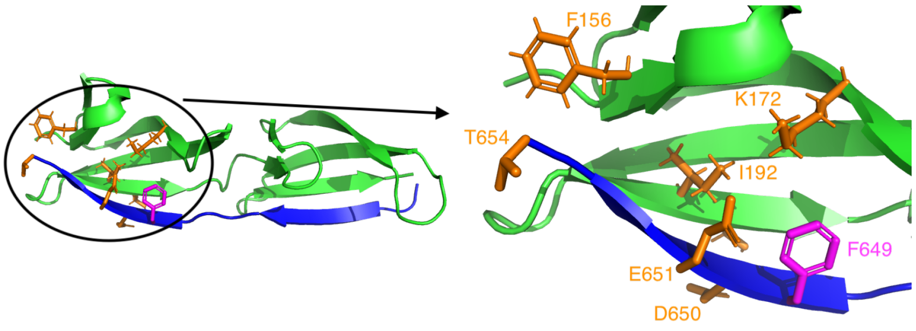

Figure 6. Setup for the Python-scripted *BBK**[36] design described in Figure 5. This design starts with the crystal structure (PDB ID: 2RL0[50]) of a complex between fragments of the protein FnBPA from *Staphylococcus aureus* (blue ribbons) and human fibronectin (green ribbons), and optimizes binding with respect to the amino acid type of FnBPA residue 649 (magenta), while modeling continuous flexibility in several surrounding sidechains (orange). The full complex is shown on the left, while the region surrounding the mutation is shown in detail on the right. See Ref. 36 for background on the FnBPA:fibronectin system. [Color figure can be viewed at wileyonlinelibrary.com]

with respect to the wildtype backbone) while still performing a comprehensive search of its backbone conformation space. In Ref. 20, we have shown that backbone flexibility as modeled by CATS is sometimes critical for avoiding nonphysical steric clashes (Figs. 7B and 7C) and often affects energetics significantly. For example, mutating residue 54 of the antibody VRC07 to tryptophan improves its binding to its antigen (HIV surface protein gp120),[7] but a design to recapitulate this mutation found it to be blocked by a steric clash unless CATS was used to find a backbone motion that escapes the clash.[20] In this design, CATS significantly outperformed a provable search over backrub[53] motions, which are also available in OSPREY.[19,54]

CATS is intended to be run as part of the flexibility model for OSPREY's other algorithms, yielding efficient calculations with continuous flexibility in both the sidechains and the backbone. OSPREY's convenient interface allows a user to add CATS flexibility to a design merely by specifying the start and end points of the backbone segment to be made flexible.

*BBK**: Efficiently computing the tightest binding sequences from a combinatorially large number of binding partners

In previous versions of OSPREY, the *K** algorithm[24] modeled an ensemble of Boltzmann-weighted conformations to approximate the thermodynamic partition function. It combined minimized dead-end elimination pruning[14] with *A**[14,55] gap-free conformation enumeration to compute provable ε-approximations to the partition functions for the protein and ligand states of interest. *K** combined these partition function scores to approximate the association constant, K_a, as the ratio of ε-approximate partition functions between the bound and unbound states of a protein-ligand complex. Notably, each partition function ratio, called a *K* score*, is provably accurate with respect to the biophysical *input model*.[2,16,24]

Although *K** efficiently and provably approximated K_a for a given sequence, it had to compute a *K** score for each sequence of interest. All provable ensemble-based algorithms prior to *BBK**, as well as many heuristic algorithms that optimize binding affinity, are *single-sequence* algorithms which must compute the binding affinity for each possible sequence. The number of sequences, of course, is exponential in the number of simultaneously mutable residue positions. Therefore, designs with many mutable residues rapidly became intractable. OSPREY 3.0 provides a new algorithm, *BBK**, which overcomes this challenge. *BBK**[36] builds on *K**, and is the first provable, ensemble-based protein design algorithm to run in time sublinear in the number of sequences. The key innovation in *BBK** that enables this improvement is the *multisequence (MS) bound*. Rather than compute binding affinity separately for each possible sequence, as single-sequence methods do, *BBK** efficiently computes a

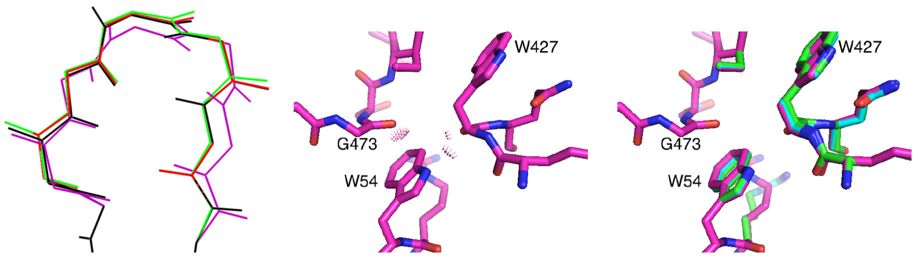

Figure 7. Left: CATS allows systematic search over a voxel of backbone conformations in the vicinity of the wild-type backbone conformation (black). The voxel is specified as box constraints on a novel set of backbone coordinates; conformations with one such coordinate moved to the edge of the voxel are shown in red and green, and a conformation with all such coordinates moved to the edge of the voxel is shown in purple. See figure 1 of Ref. 20 for more details. Middle: Rigid-backbone structural modeling of an experimentally effective mutant of anti-HIV gp120 antibody VRC07 showed unavoidable steric clashes between Trp 54 of VRC07 and Trp 427 and Gly 473 of gp120 (purple). Right: CATS explained the experimentally observed activity by finding a new backbone conformation that resolved these clashes [green; overlaid with clashing rigid-body backbone (purple) and backbone conformation computed with the older DEEPer algorithm (blue)]. DEEPer reduced the clashes somewhat using backrub motions [53], but they were still significant even after the backrubs. See figure 3 of Ref. 20 for more details. Portions of this figure were reprinted with permission from Ref. 20. [Color figure can be viewed at wileyonlinelibrary.com]

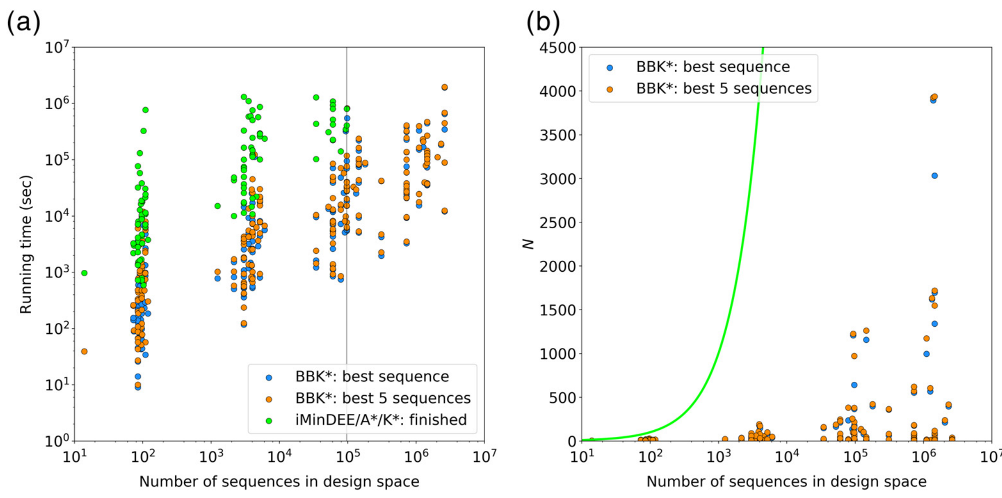

Figure 8. Measuring the contributions of the *BBK** [36] algorithmic improvements to the empirically observed running times of OSPREY. *BBK** calculations were run to predict either the single top sequence or to enumerate the top 5, and were compared to exhaustive computation of *K** scores for each sequence (i.e., iMinDEE/*A**/*K** [2,16] or single-sequence *K**), which was the prior state of the art for Boltzmann-weighted ensemble-based binding affinity computation before *BBK**. a) Running times for *BBK** and single-sequence *K** versus the number of sequences in the search space for 204 protein design test cases, a benchmark set described in Ref. 36. Single-sequence *K** completed only 107 of the test cases within a 30-day time limit (left of the vertical line), and took up to 800 times longer than *BBK** to do so, while *BBK** completed all the designs within the time limit. b) The number *N* of sequences whose energies must be examined or bounded by iMinDEE/*A**/*K** (green line; exponential in the number of mutable residue positions) and by *BBK** (dots). For each data point representing a *BBK** test case, the vertical gap between that data point and the green line (gap on the *y* axis) represents the number of sequences that are pruned without ever having to be examined. Figure adapted with permission from Ref. 36. [Color figure can be viewed at wileyonlinelibrary.com]

single provable *K** score upper bound for a combinatorial number of sequences. *BBK** uses MS bounds to prune a combinatorial number of sequences during the search, entirely avoiding single-sequence computation for all pruned sequences.

Importantly, *BBK** also contains many other powerful algorithmic improvements and implementation optimizations: the parallel architecture of *BBK**, which enables concurrent energy minimization, and a novel two-pass partition function bound, which minimizes far fewer conformations while still computing a provable ε-approximation to the partition function. Combined with the combinatorial pruning power of the MS bound, *BBK** is able to search over much larger sequence spaces than previously possible with single-sequence *K** (Fig. 8). In computational experiments on 204 protein design problems, *BBK** accurately predicted the tightest-binding sequences while only computing *K** scores for as few as one in 10^5 of the sequences in the search space[36]. Moreover, in computational experiments on 51 protein–ligand design problems, *BBK** was up to 1982-fold faster than single-sequence *K**, despite provably producing the same results.[36]

These improvements show that *BBK** not only accelerates protein designs that were possible with previous provable algorithms, it also efficiently performs designs that are too large for previous methods.

BWM*: Exploiting locality of protein energetics to efficiently compute the GMEC

OSPREY 3.0 comes with BWM*,[23] a new algorithm that exploits sparse energy functions to provably compute the GMEC in time exponential in merely the branch-width *w* of a protein design problem's sparse residue interaction graph.

Because energy decreases as a function of distance, many protein design algorithms model protein energetics with energy functions which omit pairwise interactions between sufficiently distant residues. These *sparse energy functions* not only provide a simpler, more efficiently computed model of energy, but also induce *optimal substructure* to the problem: because not all residues interact, the optimal conformation for a given residue can be independent of the conformations at other residues. BWM* exploits this optimal substructure by (1) representing the sparse interactions with a sparse residue interaction graph, and (2) computing a branch-decomposition for use in dynamic programming.

BWM*, unlike treewidth-based methods that also exploit the sparsity of pairwise residue interactions to efficiently compute the GMEC,[56] enumerates a gap-free list of conformations in order of increasing sparse energy. Because this list is gap-free, BWM* not only computes the GMEC of the sparse energy function, but also recovers the GMEC of the full energy function, as shown in Ref. 23. By enumerating all conformations within the provable sparse energy bound between the sparse and full GMEC, BWM* computes a list of conformations that is guaranteed to contain the full GMEC, as well as the sparse GMEC.[57] Moreover, because BWM* can enumerate conformations in gap-free order up to any energy threshold specified by the user, it can be used to accurately compute partition functions, and thus binding free energies that account correctly for entropy, using the *K** algorithm.[2,24]

Thus, in practice, BWM* circumvents the worst-case complexity of traditional methods such as *A** for designs with sparse energy functions, computing the sparse GMEC of an *n*-residue design with at most *q* rotamers per residue in $\mathcal{O}\left(nw^2q^{\frac{3}{2}w}\right)$ time, and also enumerates each additional conformation in merely

Table 2. Comparison of OSPREY predictions to experimental results for mutations in four protein systems

	Mutation(s)	Experimental ranking	Computational ranking
Barnase: Barstar, PDB ID: 1X1U	D39A	1	1
	H102A	2	3
	R87A	3	5
	K27A	4	8
	R59A	5	2
	D35A	6	4
	Y29A	7	7
	E73A	8	12
	E76A	9	6
	W35F	10	11
	E60A	11	10
	Y29F	12	9
		$\rho = 0.755$	
IL-2:IL-2Rα, PDB ID: 2B5I	K38E, S39D	1.5	1
	R35T, R36S	1.5	2
	R35K, R36K	3	4
	E1K, D4K	4	7
	E29R	5	5
	L2A	6	16
	D4K	7.5	9
	S39A, S41A	7.5	12
	E1K	9	11
	H120A	10	10
	E29A	11	6
	L42S, Y43L	12	3
	E1Q	13	14
	N27A	14	15
	K38 T	15	8
	D4N	16	13
		$\rho = 0.554$	
Cyt *c*:Cyt *c* peroxidase, PDB ID: 2PCB	E290N	1	2
	D34N	2	4
	A193F	3	1
	E35Q	4	3
	E32Q	5	5
		$\rho = 0.500$	
IFNα2:ifnar2, PDB ID: 3S9D	R33Q	1	1
	R33A	2	2
	R33K	3	5
	L30A	4	6
	R149A	5	4
	L30V	6	9
	A148A	7	10
	A145G	8	14
	A145M	9	3
	L15A	10	13
	L153A	11	12
	L26A	12	7
	S152A	13	16
	F27A	14	8
	S25A	15	18
	D35A	16	17
	R22A	17	11
	M16A	18	15
	N156A	19	19
			$\rho = 0.795$
		Across All	$\rho = 0.762$

Allowed mutations for each system are listed along with their corresponding rankings from experimental measurements [62–66] versus computational predictions by *K** from OSPREY 3.0. The mutations highlighted in blue are shown in detail in Figure 9. A Spearman's ρ value is calculated for each system and shown here. The "Across All" value is calculated by ranking each system individually and then calculating the Spearman's ρ across all of the designs.

$\mathcal{O}(n\log q)$ time, which is up to three orders of magnitude faster than traditional $A*$ in practice.[23]

Accuracy Benchmarks

We first tested the accuracy of OSPREY 3.0 for the subset of algorithms also available in OSPREY 2.2β, by running both versions of OSPREY on the same test cases and checking that the results matched. Since the accuracy of OSPREY 2.2β using these algorithms has been experimentally confirmed (see Introduction), by transitivity, our tests confirmed OSPREY 3.0's accuracy. In addition, we performed new, retrospective tests, described below.

To evaluate the accuracy of the implementation of the newest optimizations in OSPREY 3.0, we performed a series of designs for a variety of protein–protein interfaces (PPIs) as retrospective validation. We used *K**[24] to computationally predict experimentally measured changes in binding for each PPI. Each protein structure is listed by name and PDB ID in Table 2[58–61] These systems include barnase with its peptide inhibitor barstar,[62,63] the cytochrome *c*:cytochrome *c* peroxidase complex,[64] interferon α-2 (IFNα2) in complex with interferon α/β receptor 2 (IFNAR2),[65] and the interleukin 2 (IL-2):IL-2 receptor α (IL-2Rα) complex.[66]

Our retrospective validation experiments focused on mutations at residues in or proximal to the protein–protein interface that were not limited to alanine scanning. Including some of these tested and reported mutations,[62–66] for each structure we tested anywhere from 5 to 19 designs. In total, we tested 58 mutations using default, out-of-the-box OSPREY 3.0 settings and parameters. Each design included one or two mutable residues along with a set of surrounding flexible residues (See Table 2). Flexible residues were chosen by selecting all residues within 4 Å of the mutable residues and removing those that only have backbone interactions. Two example designs are shown in Figure 9, where OSPREY 3.0 and *K** accurately predict the effect of two point mutations in the interface of the IFNα2: IFNAR2 complex (highlighted in blue in Table 2).

For each system, the *K** scores were ranked in increasing order of reported experimental binding. Spearman's ρ values were subsequently calculated for each system by calculating the statistical dependence between the *K** score rankings and the experimentally measured rankings (See Table 2 and Fig. 10). This is a sound measure because generally the output of a design calculation that is used to decide which mutants to make experimentally is simply the intra-system ranks of the mutants. Looking at the values in Table 2, we see a high correlation in the rankings between experimentally measured binding and binding predicted by OSPREY 3.0 and *K** for each system with values ranging from 0.500 to 0.795. We found that, across the tested systems, the Spearman's ρ value is 0.762. This value is the Pearson correlation of the intra-system ranks of all the mutants. Overall, these correlations are very good for design for affinity in computational protein design.

Discussion

OSPREY has demonstrated its accuracy and utility in practice through many prospective designs that have performed well

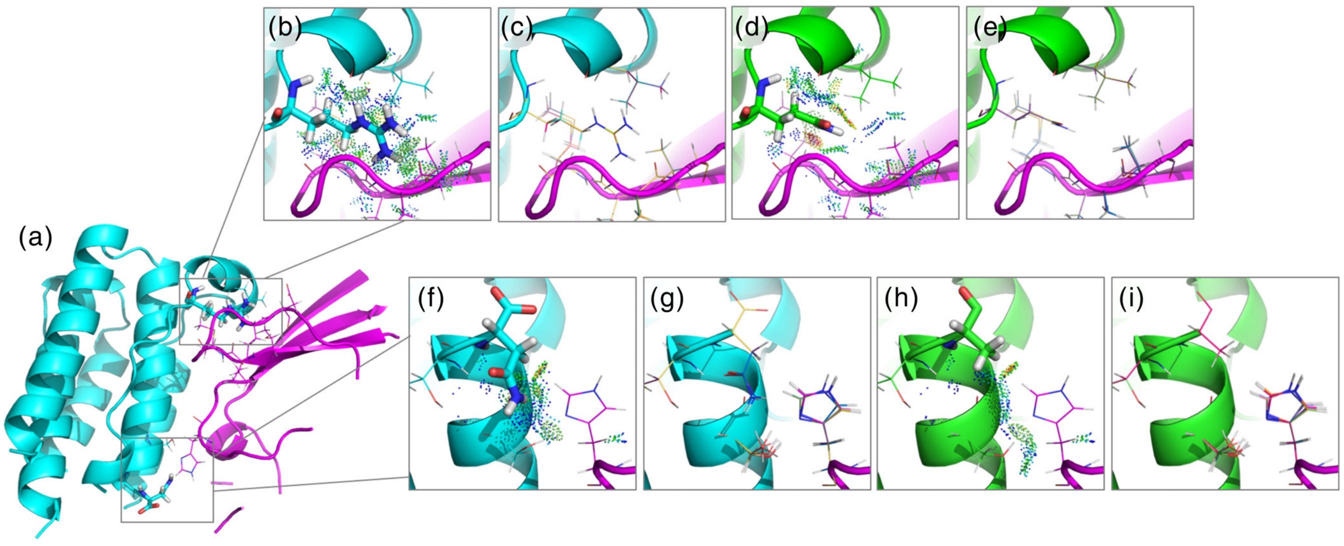

Figure 9. a) The structure of the IFNα2:IFNAR2 complex (PDB ID: 3S9D [60]) with separate chains shown in cyan and magenta and with two example interface design regions shown in boxes. Each box contains a mutable residue shown as sticks and its surrounding flexible residues shown as lines. b) and f) zoom in on each design. (b–e) Design at position R33 for a mutation that OSPREY correctly predicts as decreasing binding: R33Q. b) The wildtype sequence with probe dots [67,68] displaying favorable interactions with surrounding flexible residues (shown as lines). d) The mutant sequence (33Q) with probe dots displaying some favorable as well as unfavorable interactions. Comparing (b) and (d), it is clear there is a loss in favorable interactions and a gain in unfavorable interactions on mutation from R to Q, resulting in an experimentally observed decrease in binding that the *K** algorithm captures accurately (See Table 2). c) and e) show the top 10 conformations in the conformational ensemble used in the *K** calculation for each sequence. (f–i) Design at position N156 for a mutation that OSPREY correctly predicts as increasing binding: N156A. f) The wildtype sequence with probe dots [67,68] displaying some favorable interactions with surrounding flexible residues (shown as lines). h) The mutant sequence (156A) with probe dots displaying some favorable interactions with surrounding flexible residues (shown as lines). There are some gained interactions (shown by an increase in the number of favorable probe dots) in (h) compared to (f), but these are not visually obvious, thus emphasizing the importance of *K**, which successfully picks up these nuanced changes and correctly predicts improved binding (See Table 2). g) and i) show the top 10 conformations in the conformational ensemble used in the *K** calculation for each sequence. Not shown are the ensembles for the unbound states that are also used to calculate the *K** scores. [Color figure can be viewed at wileyonlinelibrary.com]

experimentally.[6–12] OSPREY 3.0 is at least as accurate as the versions of OSPREY used to perform these designs, because it uses the same biophysical model used in those studies, with provable guarantees of accuracy given the biophysical model. We have compared design results using OSPREY 2.2 and OSPREY 3.0 to confirm agreement. However, OSPREY 3.0 performs such designs much more efficiently, due to the engineering improvements described here. Moreover, in this paper we have performed additional comparisons to experimental data to confirm the accuracy of OSPREY 3.0. OSPREY 3.0 also includes methods to improve the biophysical model and thus improve accuracy still further (should the user choose to select OSPREY's newer models).

As our benchmark results here show, we have made substantial progress toward correctly predicting the effect of mutations on protein activity. The high accuracy comes from OSPREY's accurate biophysical model, which accounts for both continuous protein flexibility and conformational entropy, together with algorithms that provably return optimal sequences given that model. In fact, no other software can provide a provable guarantee of accuracy given a model that accounts for continuous flexibility and conformational entropy. Moreover, OSPREY's combinatorial algorithms[4,5] compute optimal sequences efficiently even when searching over a large sequence space.

The large speedups in OSPREY 3.0, together with the easy-to-use Python interface, thus make it much more tractable to perform protein design with such biophysically realistic modeling and with guaranteed accuracy given the model. In particular, OSPREY 3.0 benefits from many sources of speedups that can be used together. Speedups from OSPREY 3.0's optimization of the conformational minimization, forcefield evaluation, and A* routines can exceed two orders of magnitude even compared to OSPREY 2.2[21] running on the same CPU hardware. Together with an additional speedup of over an order of magnitude from GPU's, a design that would take months using OSPREY 2.2 could easily take only a few hours using OSPREY 3.0. Many designs could see even greater speedups, because in addition to these engineering improvements, some of the algorithmic improvements in OSPREY 3.0 provide a dramatic increase in computational efficiency.

The improvements in modeling facilitated by OSPREY 3.0's new algorithms also make protein design with OSPREY more realistic. However, there is still much room for improvement in the biophysical model used by OSPREY, and indeed by all currently available protein design software. Modeling of larger backbone motions, more realistic interactions with water, and electronic polarization, among other phenomena, are all likely to yield substantial improvements in accuracy. The refactored architecture of OSPREY 3.0 will make it easier to experiment with algorithms that facilitate these modeling improvements, and to implement these algorithms within OSPREY's current code base. Moreover, we have released OSPREY 3.0 as open source, to aid the community both in the development and the application of improved models and algorithms for computational protein design.

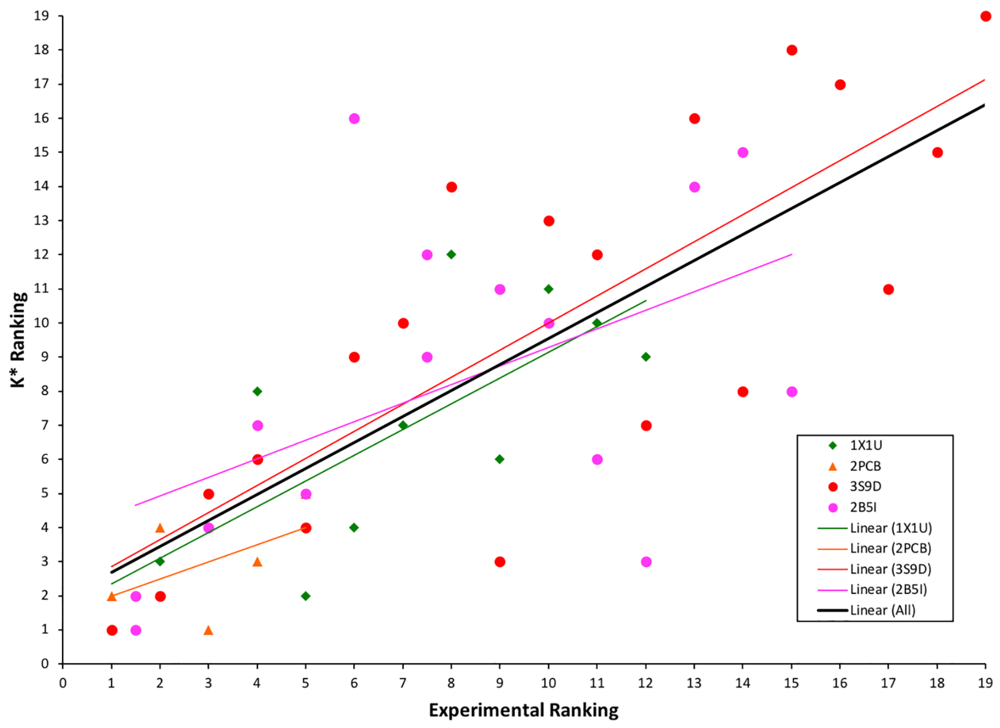

Figure 10. Testing the accuracy of the *K** algorithm in OSPREY 3.0 by comparing *K** rankings to experimentally reported rankings (See Table 2). Each system is represented by its corresponding PDB ID and a linear trendline is shown for each in its corresponding color according to the legend. [Color figure can be viewed at wileyonlinelibrary.com]

Conclusions

OSPREY has long offered unique capabilities to protein designers. In particular, it has always offered a unique combination of provably accurate conformational search, continuous flexibility, efficient search over large sequence spaces, and free energy calculations based on Boltzmann-weighted thermodynamic conformational ensembles. In OSPREY 3.0 we introduced software improvements that will make these algorithms much more practical for the wider design community: performance that is orders of magnitude faster, and a Python interface that makes OSPREY much easier to use. In addition, we expanded the range of biophysical modeling assumptions that OSPREY can accommodate, both in terms of molecular flexibility and energy functions. As with previous versions, we are releasing OSPREY 3.0 as free and open-source software to maximize its benefit to the community. We hope this new version will be of significant utility to designers, whether they have used OSPREY before or are trying it for the first time.

Acknowledgments

The authors would like to thank Dr. Alvin Lebeck for helpful discussions on GPUs, Drs. Kyle Roberts and Swati Jain for helpful discussions on protein design.

Keywords: protein design · drug design · GPU · structural biology · Python

How to cite this article: M. A. Hallen, J. W. Martin, A. Ojewole, J. D. Jou, A. U. Lowegard, M. S. Frenkel, P. Gainza, H. M. Nisonoff, A. Mukund, S. Wang, G. T. Holt, D. Zhou, E. Dowd, B. R. Donald. *J. Comput. Chem.* **2018**, 39, 2494–2507. DOI: 10.1002/jcc.25522

[1] P. Gainza, K. E. Roberts, I. Georgiev, R. H. Lilien, D. A. Keedy, C.-Y. Chen, F. Reza, A. C. Anderson, D. C. Richardson, J. S. Richardson, Bruce R. Donald, Methods Enzymol. **2013**, 523, 87.
[2] I. Georgiev, R. H. Lilien, B. R. Donald, J. Comput. Chem. **2008**, 29, 1527.
[3] I. Georgiev, K. E. Roberts, P. Gainza, M. A. Hallen, B. R. Donald, OSPREY (Open Source Protein Redesign for You) User Manual, **2009**; 94 p. Available at: www.cs.duke.edu/donaldlab/software.php Updated, 2015.
[4] B. R. Donald, Algorithms in Structural Molecular Biology, MIT Press, Cambridge, MA, **2011**.
[5] P. Gainza, H. M. Nisonoff, B. R. Donald, Curr. Opin. Struct. Biol. **2016**, 39, 16.
[6] C.-Y. Chen, I. Georgiev, A. C. Anderson, B. R. Donald, Proc. Natl. Acad. Sci. USA **2009**, 106, 3764.
[7] R. S. Rudicell, Y. D. Kwon, S.-Y. Ko, A. Pegu, M. K. Louder, I. S. Georgiev, X. Wu, J. Zhu, J. C. Boyington, X. Chen, W. Shi, Z. Y. Yang, N. A. Doria-Rose, K. McKee, S. O'Dell, S. D. Schmidt, G. Y. Chuang, A. Druz, C. Soto, Y. Yang, B. Zhang, T. Zhou, J. P. Todd, K. E. Lloyd, J. Eudailey, K. E. Roberts, B. R. Donald, R. T. Bailer, J. Ledgerwood, NISC Comparative Sequencing Program, J. C. Mullikin, L. Shapiro, R. A. Koup, B. S. Graham, M. C. Nason, M. Connors, B. F. Haynes, S. S. Rao, M. Roederer, P. D. Kwong, J. R. Mascola, G. J. Nabel, J. Virol. **2014**, 88, 12669.
[8] K. E. Roberts, P. R. Cushing, P. Boisguerin, D. R. Madden, B. R. Donald, PLoS Comput. Biol. **2012**, 8, e1002477.
[9] M. J. Gorczynski, J. Grembecka, Y. Zhou, Y. Kong, L. Roudaia, M. G. Douvas, M. Newman, I. Bielnicka, G. Baber, T. Corpora, J. Shi, M. Sridharan, R. Lilien, B. R. Donald, N. A. Speck, M. L. Brown, J. H. Bushweller, Chem. Biol. **2007**, 14, 1186.
[10] K. M. Frey, I. Georgiev, B. R. Donald, A. C. Anderson, Proc. Natl. Acad. Sci. USA, Vol. 107, Washington, DC, **2010**, p. 13707.
[11] B. W. Stevens, R. H. Lilien, I. Georgiev, B. R. Donald, A. C. Anderson, Biochemistry **2006**, 45, 15495.
[12] I. Georgiev, P. Acharya, S. Schmidt, Y. Li, D. Wycuff, G. Ofek, N. Doria-Rose, T. Luongo, Y. Yang, T. Zhou, B. R. Donald, J. R. Mascola, P. D. Kwong, Retrovirology **2012**, 9, P50.
[13] S. M. Reeve, P. Gainza, K. M. Frey, I. Georgiev, B. R. Donald, A. C. Anderson, Proc. Natl. Acad. Sci. USA **2015**, 112, 749.
[14] J. Desmet, M. de Maeyer, B. Hazes, I. Lasters, Nature **1992**, 356, 539.
[15] A. R. Leach, A. P. Lemon, Proteins **1998**, 33, 227.

[16] P. Gainza, K. Roberts, B. R. Donald, PLoS Comput. Biol. **2012**, 8, e1002335.

[17] M. A. Hallen, P. Gainza, B. R. Donald, J. Chem. Theory Comput. **2015**, 11, 2292.

[18] M. A. Hallen, J. D. Jou, B. R. Donald, International Conference on Research in Computational Molecular Biology, Springer: Berlin, **2016**, p. 122.

[19] M. A. Hallen, D. A. Keedy, B. R. Donald, Proteins **2013**, 81, 18.

[20] M. A. Hallen, B. R. Donald, Bioinformatics **2017**, 33, i5.

[21] M. A. Hallen, B. R. Donald, J. Comput. Biol. **2016**, 23, 311.

[22] K. E. Roberts, P. Gainza, M. A. Hallen, B. R. Donald, Proteins **2015**, 83, 1859.

[23] J. D. Jou, S. Jain, I. Georgiev, B. R. Donald, J. Comput. Biol. **2016**, 23, 413.

[24] R. H. Lilien, B. W. Stevens, A. C. Anderson, B. R. Donald, J. Comput. Biol. **2005**, 12, 740.

[25] A. Ojewole, A. Lowegard, P. Gainza, S. M. Reeve, I. Georgiev, A. C. Anderson, B. R. Donald, Methods Mol. Biol. **2017**, 1529, 291.

[26] S. M. Reeve, E. W. Scocchera, G. Narendran, S. Keshipeddy, J. Krucinska, B. Hajian, J. Ferreira, M. Nailor, J. Aeschlimann, D. L. Wright, Amy C. Anderson, Cell Chem. Biol. **2016**, 23, 1458.

[27] VRC 605: A Phase 1 Dose-Escalation Study of the Safety and Pharmacokinetics of a Human Monoclonal Antibody, VRC07-523LS, Administered Intravenously or Subcutaneously to Healthy Adults. ClinicalTrials.gov Identifier: NCT03015181. NIAID and National Institutes of Health Clinical Center. January (**2017**). https://clinicaltrials.gov/ct2/show/NCT03015181.

[28] I. S. Georgiev, R. S. Rudicell, K. O. Saunders, W. Shi, T. Kirys, K. McKee, S. O'Dell, G.-Y. Chuang, Z.-Y. Yang, G. Ofek, et al., J.Immunol. **2014**, 192, 1100.

[29] G. Y. Chuang, H. Geng, M. Pancera, K. Xu, C. Cheng, P. Acharya, M. Chambers, A. Druz, Y. Tsybovsky, T. G. Wanninger, Yongping Yang, Nicole A. Doria-Rose, Ivelin S. Georgiev, Jason Gorman, M. Gordon Joyce, Sijy O'Dell, Tongqing Zhou, Adrian B. McDermott, John R. Mascola, Peter D. Kwong, J. Virol. **2017**, 91, e02268.

[30] N. H. Joh, T. Wang, M. P. Bhate, R. Acharya, Y. Wu, M. Grabe, M. Hong, G. Grigoryan, W. F. DeGrado, Science **2014**, 346, 1520.

[31] A. S. Parker, Y. Choi, K. E. Griswold, C. Bailey-Kellogg, J. Comput. Biol. **2013**, 20, 152.

[32] R. S. Salvat, Y. Choi, A. Bishop, C. Bailey-Kellogg, K. E. Griswold, Biotechnol. Bioeng. **2015**, 112, 1306–1318.

[33] H. Zhao, D. Verma, W. Li, Y. Choi, C. Ndong, S. N. Fiering, C. Bailey-Kellogg, K. E. Griswold, Chem. Biol. **2015**, 22, 629.

[34] N. A. Doria-Rose, H. R. Altae-Tran, R. S. Roark, S. D. Schmidt, M. S. Sutton, M. K. Louder, G. Y. Chuang, R. T. Bailer, V. Cortez, R. Kong, Rui Kong, Krisha McKee, Sijy O'Dell, Felicia Wang, Salim S. Abdool Karim, James M. Binley, Mark Connors, Barton F. Haynes, Malcolm A. Martin, David C. Montefiori, Lynn Morris, Julie Overbaugh, Peter D. Kwong, John R. Mascola, Ivelin S. Georgiev, PLoS Pathog. **2017**, 13, e1006148.

[35] J. Janin, S. Wodak, M. Levitt, B. Maigret, J. Mol. Biol. **1978**, 125, 357.

[36] A. A. Ojewole, J. D. Jou, V. G. Fowler, B. R. Donald, J. Comput. Biol. (in press).

[37] I. Georgiev, R. H. Lilien, B. R. Donald, Bioinformatics **2006**, 22, e174.

[38] Y. Pan, Y. Dong, J. Zhou, M. Hallen, B. R. Donald, J. Zeng, W. Xu, J. Comput. Biol. **2016**, 23, 737.

[39] R. C. Walker, A. W. Goetz, Electronic Structure Calculations on Graphics Processing Units: From Quantum Chemistry to Condensed Matter Physics, Wiley, Hoboken, **2016**.

[40] K. He, X. Zhang, S. Ren, J. Sun, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, IEEE, Piscataway, **2016**, p. 770.

[41] R. Szerwinski, T. Gűneysu, International Workshop on Cryptographic Hardware and Embedded Systems, Springer: Berlin, **2008**, p. 79.

[42] J. Glaser, T. D. Nguyen, J. A. Anderson, P. Lui, F. Spiga, J. A. Millan, D. C. Morse, S. C. Glotzer, Comput. Phys. Commun. **2015**, 192, 97.

[43] R. Salomon-Ferrer, A. W. Götz, D. Poole, S. Le Grand, R. C. Walker, J. Chem. Theory Comput. **2013**, 9, 3878.

[44] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, E. Lindahl, SoftwareX 1, Vol. 19, Elsevier, New York, **2015**.

[45] Y. Zhou, W. Xu, B. R. Donald, J. Zeng, Bioinformatics **2014**, 30, i255.

[46] C. Nvidia, Programming Guide, Nvidia: Santa Clara, **2010**.

[47] A. C. Rosenzweig, D. L. Huffman, M. Y. Hou, A. K. Wernimont, R. A. Pufahl, T. V. O'Halloran, Structure **1999**, 7, 605.

[48] S. C. Lovell, M. J. Word, J. S. Richardson, D. C. Richardson, Proteins **2000**, 40, 389.

[49] A. Globerson, T. S. Jaakkola, Advances in Neural Information Processing Systems, Neural Information Processing Systems Foundation: San Diego, **2008**, p. 553.

[50] R. J. Bingham, E. Rudiño-Piñnera, N. A. Meenan, U. Schwarz-Linek, J. P. Turkenburg, M. Höök, E. F. Garman, J. R. Potts, Proc. Natl. Acad. Sci. USA **2008**, 105, 12254.

[51] A. Nicholls, B. Honig, J. Comput. Chem. **1991**, 12, 435.

[52] W. Rochia, S. Sridharan, A. Nicholls, E. Alexov, A. Chiabrera, B. Honig, J. Comput. Chem. **2002**, 23, 128.

[53] I. W. Davis, W. B. Arendall, D. C. Richardson, J. S. Richardson, Structure **2006**, 14, 265.

[54] I. Georgiev, D. Keedy, J. S. Richardson, D. C. Richardson, B. R. Donald, Bioinformatics **2008**, 24, i196.

[55] P. E. Hart, N. J. Nilsson, B. Raphael, IEEE Trans. Syst. Sci. Cybern. **1968**, 4, 100.

[56] J. Xu, B. Berger, J. ACM **2006**, 53, 533.

[57] S. Jain, J. D. Jou, I. S. Georgiev, B. R. Donald, PLoS Comput. Biol. **2017**, 13, e1005346.

[58] T. Ikura, Y. Urakubo, N. Ito, Chem. Phys. **2004**, 307, 111.

[59] H. Pelletier, J. Kraut, Science **1992**, 258, 1748.

[60] C. Thomas, I. Moraga, D. Levin, P. O. Krutzik, Y. Podoplelova, A. Trejo, C. Lee, G. Yarden, S. E. Vleck, J. S. Glenn, Garry P. Nolan, Jacob Piehler, Gideon Schreiber, Christophe Garcia, Cell **2011**, 146, 621.

[61] X. Wang, M. Rickert, K. C. Garcia, Science, 2005, 1159.

[62] G. Schreiber, A. R. Fersht, Biochemistry **1993**, 32, 5145.

[63] C. Frisch, G. Schreiber, C. M. Johnson, A. R. Fersht, J. Mol. Biol. **1997**, 267, 696.

[64] J. E. Erman, G. C. Kresheck, L. B. Vitello, M. A. Miller, Biochemistry **1997**, 36, 4054.

[65] J. Piehler, L. C. Roisman, G. Schreiber, J. Biol. Chem. **2000**, 275, 40425.

[66] R. J. Robb, C. M. Rusk, M. P. Neeper, Proc. Natl. Acad. Sci. **1988**, 85, 5654.

[67] J. M. Word, S. C. Lovell, T. H. LaBean, H. C. Taylor, M. E. Zalis, B. K. Presley, J. S. Richardson, D. C. Richardson, J. Mol. Biol. **1999**, 285, 1711.

[68] K. E. Roberts, B. R. Donald, Protein Interaction Viewer, **2014**. http://www.cs.duke.edu/donaldlab/software/proteinInteractionViewer/

Received: 23 April 2018
Accepted: 14 June 2018
Published online on 14 October 2018