

# <sup>1</sup> RustSASA: A Rust Crate for Accelerated Solvent Accessible Surface Area Calculations

<sup>3</sup> Maxwell J. Campbell  <sup>1</sup>

<sup>4</sup> 1 University of California, San Francisco, United States

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

## Software

- <sup>5</sup> [Review](#) 
- <sup>6</sup> [Repository](#) 
- <sup>7</sup> [Archive](#) 

---

Editor: [Open Journals](#) 

Reviewers:

- <sup>8</sup> [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

## License

<sup>16</sup> Authors of papers retain copyright  
<sup>17</sup> and release the work under a  
<sup>18</sup> Creative Commons Attribution 4.0  
International License ([CC BY 4.0](#)).

## <sup>5</sup> Summary

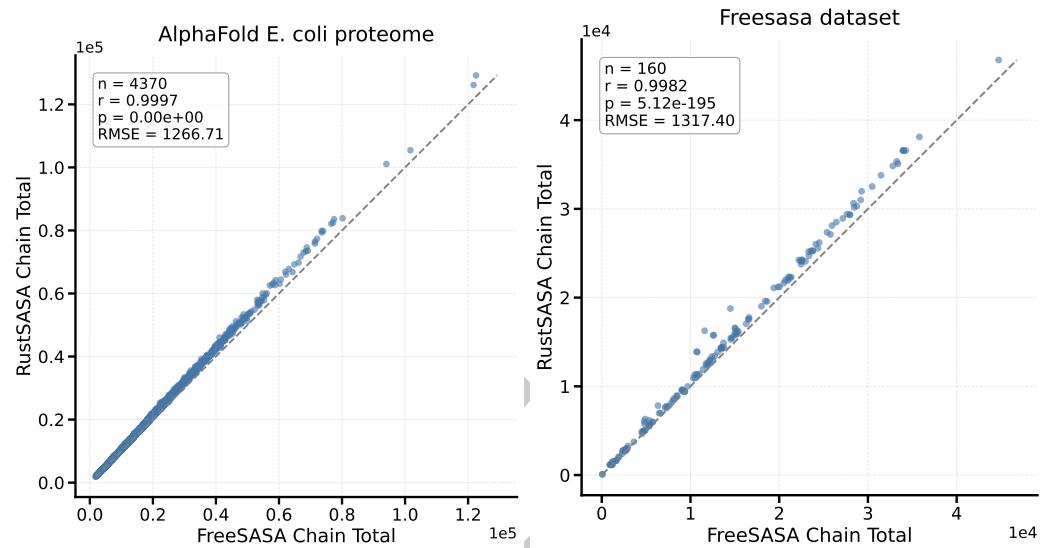
<sup>6</sup> Solvent accessible surface area (SASA) calculations are fundamental for understanding protein  
<sup>7</sup> structure, function, and dynamics in computational biology. These calculations quantify the  
<sup>8</sup> surface area of biomolecules accessible to solvent molecules, providing insights into protein  
<sup>9</sup> folding, stability, and intermolecular interactions. The Shrake-Rupley algorithm has served as  
<sup>10</sup> the standard for SASA calculations since 1973, but existing implementations often become  
<sup>11</sup> computational bottlenecks when analyzing large protein datasets. As proteomics datasets  
<sup>12</sup> continue to grow with initiatives like AlphaFold producing hundreds of millions of predicted  
<sup>13</sup> protein structures, the need for efficient SASA calculation tools has increased dramatically.  
<sup>14</sup> RustSASA addresses this challenge by providing a high-performance implementation of the  
<sup>15</sup> Shrake-Rupley algorithm written in pure Rust, delivering a 5 $\times$  speed improvement over  
<sup>16</sup> Freesasa while maintaining calculation accuracy and providing interfaces for multiple program-  
<sup>17</sup> ming languages and frameworks (i.e. MDAnalysis). The source code is freely available at  
<sup>18</sup> <https://github.com/maxall41/RustSASA>.

## <sup>19</sup> Statement of need

<sup>20</sup> Current SASA calculation tools represent a significant computational bottleneck in structural  
<sup>21</sup> biology workflows, particularly for molecular dynamics simulations and high-throughput analyses.  
<sup>22</sup> Popular implementations such as those in Biopython and Freesasa, while accurate, become  
<sup>23</sup> prohibitively slow when processing large protein datasets. RustSASA addresses this performance  
<sup>24</sup> gap by leveraging Rust's efficient parallelization abstractions (via Rayon) and readily available  
<sup>25</sup> SIMD instructions (via Pulp). These optimizations enable RustSASA's performance advantage  
<sup>26</sup> over the simpler C implementation of the same algorithm in Freesasa.  
  
<sup>27</sup> Benchmarking on representative protein structures demonstrates that RustSASA achieves a  
<sup>28</sup> 5 $\times$  improvement over Freesasa, and a 63 $\times$  performance improvement over Biopython. This  
<sup>29</sup> performance advantage reduces computational costs for high-throughput structural analyses  
<sup>30</sup> and makes large-scale comparative studies feasible. Furthermore, RustSASA's multi-language  
<sup>31</sup> support (Rust and Python), command-line interface, and MDAnalysis package ([Gowers et](#)  
<sup>32</sup> [al., 2016; Michaud-Agrawal et al., 2011](#)) ensure broad accessibility across the computational  
<sup>33</sup> biology community.

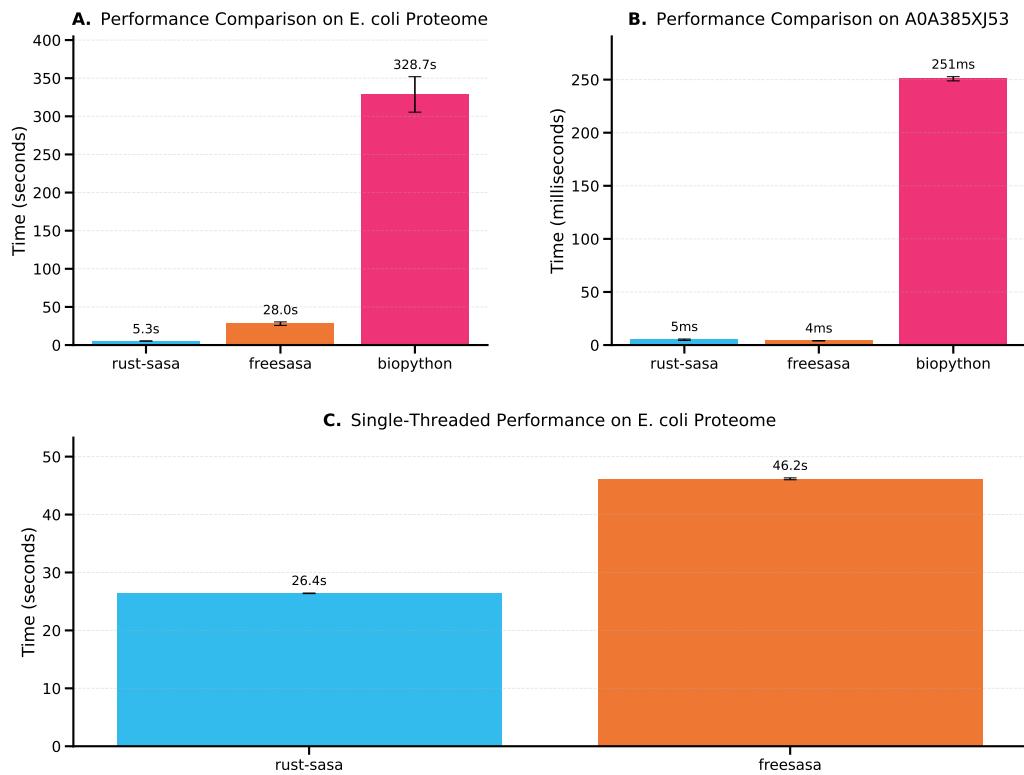
## 34 Results

### 35 Calculation Quality



36 To evaluate the accuracy of RustSASA calculations, we compared results to Freesasa (Mitter-  
 37 nacht, 2016) on both the predicted E. coli proteome from AlphaFold DB (Jumper et al., 2021;  
 38 Varadi et al., 2021) and the Freesasa evaluation dataset. RustSASA produces SASA values  
 39 that closely match those from Freesasa, achieving Pearson correlation coefficients > 0.99 on  
 40 both datasets.  
 41

42    **Performance**



**Figure 1:** **A.** Comparing the multi-threaded performance of RustSASA, Freesasa, and Biopython the full AlphaFold E. coli proteome **B.** Comparing the multi-threaded performance of RustSASA, Freesasa, and Biopython on A0A385XJ53, a protein randomly selected from the AlphaFold E. coli proteome **C.** Comparing the single-threaded performance of RustSASA, and Freesasa on the full AlphaFold E. coli proteome

43    We evaluated the performance of Freesasa, RustSASA, and Biopython (Cock et al., 2009)  
 44    across three evaluations. First, we performed multi-threaded SASA calculations for all proteins  
 45    in the E. coli proteome. Second, we evaluated the performance of these methods on a single  
 46    randomly selected protein (A0A385XJ53) from the AlphaFold E. coli proteome. Third, we  
 47    evaluated the single-threaded performance of RustSASA and Freesasa on the E. coli proteome,  
 48    Biopython was excluded from this benchmark due to its poor performance hindering timely  
 49    evaluation.

50    For the full proteome benchmarks (Fig 1A) we used Hyperfine (Peter, 2023) with 3 runs and 3  
 51    warmup iterations. All methods utilized parallel processing across eight cores. GNU parallel  
 52    (Tange, 2011) was used to parallelize Freesasa and Biopython, while RustSASA utilized its  
 53    internal parallelization. RustSASA processed the entire proteome in ~5 seconds compared to  
 54    ~28 seconds for Freesasa and ~328 seconds for Biopython, representing 5× and 63× speed  
 55    improvements, respectively.

56    For the single protein benchmark (Fig 1B), we used Hyperfine with 3 warmup iterations and  
 57    25 runs. RustSASA processed the protein in 4.3ms ( $\pm 0.5$ ), Freesasa processed the protein in  
 58    4.0ms ( $\pm 0.2$ ), and Biopython processed the protein in 250.8ms ( $\pm 2.0$ ). On the single threaded  
 59    benchmark (Fig 1C), RustSASA processed the proteome in 26.4s compared to 46.2 seconds  
 60    for Freesasa, representing a ~42% performance improvement, demonstrating that RustSASA's  
 61    performance advantage is not solely due to multi-threading.

## 62 Methods

63 SASA calculation RustSASA computes solvent-accessible surface areas (SASA) using the  
64 Shrake–Rupley algorithm ([Shrake & Rupley, 1973](#)). In this algorithm each atom is represented  
65 as a sphere with a radius equal to its atomic van der Waals radius plus the radius of a spherical  
66 solvent probe; the sphere surface is sampled with a dense quasi-uniform distribution of test  
67 points and a point is considered solvent accessible if it is not occluded by any neighboring atom  
68 sphere. For all calculations reported here a solvent probe radius of 1.4 Å (the approximate  
69 radius of a water molecule) was used.

70 Atomic radii were assigned according to the ProtOr parameter set introduced by Tsai et  
71 al. ([Tsai et al., 1999](#)). These radii were applied to all non-hydrogen heavy atoms present  
72 in the structures; hydrogen atoms, when present in input files, were ignored for the SASA  
73 computations to maintain consistency with common practice for protein SASA estimation. Any  
74 non-standard residues or ligands were ignored following the Freesasa methodology ([Mitternacht,](#)  
75 [2016](#)).

76 To ensure a fair comparison of RustSASA and Freesasa in the single-threaded benchmark, we  
77 wrote a C++ script that utilizes the Freesasa C API to compute the SASA for input proteins  
78 in a given folder. This approach ensures that command-line overhead is not responsible for  
79 RustSASA's performance advantage. Furthermore, in all experiments, Freesasa was configured  
80 to use the Shrake-Rupley algorithm over its default algorithm, Lee & Richards, to ensure an  
81 accurate comparison between the methods. Proteome-scale structure models for Escherichia  
82 coli were obtained from the AlphaFold DB (entry UP000000625\_83333\_ECOLI\_v6, available  
83 at <https://alphafold.ebi.ac.uk/download>). All experiments were conducted on a 2024 Apple  
84 MacBook Air with an M3 processor and 24GB of unified memory.

## 85 Conclusion

86 RustSASA provides a significant advancement in SASA calculation performance while main-  
87 taining accuracy, addressing a bottleneck in computational structural biology. The 5× speed  
88 improvement over current standards enables previously intractable analyses of large protein  
89 datasets and molecular dynamics simulations. By providing interfaces for multiple programming  
90 languages alongside a command-line tool and MDAnalysis package, RustSASA ensures broad  
91 accessibility across the research community. As structural biology datasets continue to expand,  
92 efficient computational tools like RustSASA become essential for advancing our understanding  
93 of protein structure and function.

## 94 Acknowledgements

95 We would like to thank Rodrigo Honorato and Niccolò Bruciaferri for their valuable contributions  
96 to this project. We would also like to thank the reviewers for their insightful comments.

## 97 References

- 98 Cock, P. J., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedberg, I.,  
99 Hamelryck, T., Kauff, F., Wilczynski, B., & others. (2009). Biopython: Freely available  
100 python tools for computational molecular biology and bioinformatics. *Bioinformatics*,  
101 25(11), 1422–1423. <https://doi.org/10.1093/bioinformatics/btp163>
- 102 Gowers, R. J., Linke, M., Barnoud, J., Reddy, T. J. E., Melo, M. N., Seyler, S. L., Domański,  
103 J., Dotson, D. L., Buchoux, S., Kenney, I. M., & Beckstein, O. (2016). MDAnalysis: A  
104 python package for the rapid analysis of molecular dynamics simulations. *SciPy 2016*.  
105 <https://doi.org/10.25080/Majora-629e541a-00e>

- 106 Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool,  
107 K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard,  
108 A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., ... Hassabis, D.  
109 (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873),  
110 583–589. <https://doi.org/10.1038/s41586-021-03819-2>
- 111 Michaud-Agrawal, N., Denning, E. J., Woolf, T. B., & Beckstein, O. (2011). MDAnalysis:  
112 A toolkit for the analysis of molecular dynamics simulations. *Journal of Computational  
113 Chemistry*, 32(10), 2319–2327. <https://doi.org/https://doi.org/10.1002/jcc.21787>
- 114 Mitternacht, S. (2016). FreeSASA: An open source c library for solvent accessible surface area  
115 calculations. *F1000Research*, 5, 189. <https://doi.org/10.12688/f1000research.7931.1>
- 116 Peter, D. (2023). *Hyperfine*. <https://github.com/sharkdp/hyperfine>
- 117 Shrake, A., & Rupley, J. A. (1973). Environment and exposure to solvent of protein atoms.  
118 Lysozyme and insulin. *Journal of Molecular Biology*, 79(2), 351–371. [https://doi.org/https://doi.org/10.1016/0022-2836\(73\)90011-9](https://doi.org/https://doi.org/10.1016/0022-2836(73)90011-9)
- 119 Tange, O. (2011). GNU parallel - the command-line power tool; *Login: The USENIX Magazine*,  
120 36(1), 42–47. <http://www.gnu.org/s/parallel>
- 121 Tsai, J., Taylor, R., Chothia, C., & Gerstein, M. (1999). The packing density in proteins:  
122 Standard radii and volumes11Edited by j. M. thornton. *Journal of Molecular Biology*,  
123 290(1), 253–266. <https://doi.org/https://doi.org/10.1006/jmbi.1999.2829>
- 124 Varadi, M., Anyango, S., Deshpande, M., Nair, S., Natassia, C., Yordanova, G., Yuan, D.,  
125 Stroe, O., Wood, G., Laydon, A., Žídek, A., Green, T., Tunyasuvunakool, K., Petersen, S.,  
126 Jumper, J., Clancy, E., Green, R., Vora, A., Lutfi, M., ... Velankar, S. (2021). AlphaFold  
127 protein structure database: Massively expanding the structural coverage of protein-sequence  
128 space with high-accuracy models. *Nucleic Acids Research*, 50(D1), D439–D444. <https://doi.org/10.1093/nar/gkab1061>
- 129
- 130