

3

AlphaFold2 transmembrane protein structure prediction shines

Authors: Tamas Hegedus¹; Markus Geisler²; Gergely Lukacs³; Bianka Farkas⁴

¹ *Semmelweis University*

² *University of Fribourg*

³ *McGill University*

⁴ *Semmelweis University, PPKE ITK*

Corresponding Author: hegedus.tamas@hegelab.org

Transmembrane (TM) proteins are major drug targets, indicated by the high percentage of prescription drugs acting on them. For a rational drug design and an understanding of mutational effects on protein function, structural data at atomic resolution are required. However, hydrophobic TM proteins often resist experimental structure determination and in spite of the increasing number of cryo-EM structures, the available TM folds are still limited in the Protein Data Bank. Recently, the DeepMind's AlphaFold2 machine learning method greatly expanded the structural coverage of sequences, with high accuracy. Since the employed algorithm did not take specific properties of TM proteins into account, the validity of the generated TM structures should be assessed. Therefore, we investigated the quality of structures at genome scales, at the level of ABC protein superfamily folds, and also in specific individual cases. We tested template-free structure prediction also with a new TM fold, dimer modeling, and stability in molecular dynamics simulations. Our results strongly suggest that AlphaFold2 performs astoundingly well in the case of TM proteins and that its neural network is not overfitted. We conclude that a careful application of its structural models will advance TM protein associated studies at an unexpected level.

URL: <http://alphafold.hegelab.org>

Acknowledgements: Cystic Fibrosis Foundation: HEGEDU20I0 and NRDIO: K127961(TH); CCF LUKACS20G0, CIHR, CFI and Canada Research Chair Program (GLL) Swiss National Funds 310030_197563 (MG).

Thanks to <https://hpc.kifu.hu>, <https://www.mpibpc.mpg.de/grubmueller>, <http://gpu.wigner.mta.hu>.

4

Accelerating the solution of large number of delay differential equations with GPUs

Author: Dániel Nagy¹

Co-author: Ferenc Hegedűs²

¹ *Budapest University of Technology and Economics, Department of Hydrodynamic Systems*

² *Budapest University of Technology and Economics, Department of Hydrodynamic systems*

Corresponding Author: n.nagyd@gmail.com

Delay differential equations (DDE) appear in several branches of science and engineering. Possible applications include the modelling and forecasting of epidemics, the modelling of stability loss in control systems and many more. The delays in the differential equations can be caused by the incubation time of a virus in epidemic models or by the time which the computer needs to carry out the necessary calculation in case of computer control.

The numerical solution of problems described by DDEs is inevitable in several cases, and sometimes a large number of the same delay differential equations must be solved due to the many possible parameter combinations or initial conditions. **The serial solution of millions of equations is**

usually not viable, thus some kind of parallelization is necessary; however, **general purpose DDE solvers for GPUs do not exist at present**. Compared to ordinary differential equation (ODE) solvers, DDE solvers use values from the past because of the delay in the equation; thus, every timestep must be saved. However, it cannot be guaranteed that the required previous time instance is available in the global memory. Therefore, interpolation between the past values is necessary to maintain the order of the used numerical method.

GPUs were found extremely efficient for the numerical solution of large number of ODEs. **The most efficient method for the parallelization of numerical solution is assigning each equation to one thread** (also called *per-thread* approach). In the present work, the same strategy is applied for the acceleration of DDE solvers. The traditional 4th order fixed-timestep Explicit–Runge–Kutta (ERK) method is implemented with the extension of 3rd order Hermite-interpolation. The code is written in CUDA C++ language. This solver can be applied for a wide range of possible problems on most GPUs. However, an efficient implementation is difficult since it requires the intensive use of global memory, while the goal is to reach the maximum possible FLOP efficiency. The bottleneck of such problems is the memory bandwidth; thus, finding an optimal memory structure is necessary. Furthermore, the global memory size can also be a limiting factor, as each thread saves thousands of states in the global memory. This can lead to several gigabytes of global memory usage limiting the total number of residing threads and throttling the performance.

In the study, the basic idea of the solver is shown. **An efficient memory structure is proposed and tested, with aligned and coalesced memory access pattern**. To minimise the global memory usage, each thread works on a fixed-size array, when this array fills up, the older saved timesteps will be overwritten in a circular fashion. Problem specific codes with the previously described structure are tested on several simple test cases, and the FLOP efficiency, memory load/store efficiency and further important metrics are measured. **It is found that 40% FLOP efficiency can be reached with 95% memory efficiency**. The FLOP efficiency of the problem specific codes can be regarded as the maximal possible efficiency of my proposed solver. The implementation of a general purpose DDE solver is presented in the already existing GPU ODE solver called MPGOS and the metrics of the implementation is measured. A general solver must work for every possible problem, which requires extra computations and memory operations; consequently, its efficiency in my case only reaches half of the problem specific solvers.

Finally, the limitations of a fixed-timestep ERK method is presented, and the possibilities of adaptive ERK solvers for GPUs is discussed. However, for adaptive ERK methods, aligned and coalesced memory access patterns are not possible with the *per-thread* approach, thus a heterogeneous CPU-GPU solver is proposed which may be a viable solution for later implementations.

5

Space-ready FPGA hardware acceleration for .NET software - Hastlayer

Authors: Erno David¹; Dávid El-Saig²; Zoltán Lehóczky²

¹ MTA Wigner FK

² Lombiq Technologies Ltd.

Corresponding Authors: david.erno@wigner.mta.hu, zoltan.lehoczky@lombiq.com

Hastlayer (<https://hastlayer.com/>) by Lombiq Technologies is a .NET software developer-focused, easy-to-use high-level synthesis tool with the aim of accelerating applications. It converts standard .NET Common Intermediate Language (CIL) bytecode into equivalent Very High Speed Integrated Circuit Hardware Description Language (VHDL) constructs which can be implemented in hardware using FPGAs. After cloud-available FPGA platforms, we've made Hastlayer compatible with the Zynq 7000 family of FPGA SoC devices. The primary goal is to be able to utilize onboard computers of satellites built with the same hardware, readily available by NewSpace manufacturers. In this talk, we'll introduce Hastlayer and how it can be used, the challenges and experiences of making it compatible with Zynqs, and our results showing up to 2 orders of magnitude speed and power efficiency increases.