ACADEMIA | Letters

Parallel prefix scan for the computation of axonal projection patterns in biological neural networks

Sebastian James, Department of Psychology, The University of Sheffield

---

The graphical processing unit (GPU) is a specialized processer designed to carry out thousands of parallel computations. The need to process graphical scenes for computer games and movies has motivated sustained investment in the development of these devices which have contributed to a huge growth in both industries. A highly parallel processor is ideally suited for computer graphics because generating the image for the monitor provides the perfect parallelizable problem; the screen consists of millions of pixels, each of which must have its 3 colors specified to form an image. For a 1080p monitor, that's 6 million parallel tasks. The beauty of the problem is that the final integration of the *meaning* of these 6 million results is performed by the gamer's brain! In contrast, the solution of most mathematical problems requires that 'the pixels talk to one another: the computational elements must transfer information in order to deliver a final result or to update variables at every timestep of a simulated system. Information transfer is a task to which the GPU is, *by design*, less well suited. In spite of this drawback, from about the mid 2000s researchers began to explore the use of GPUs for general-purpose computation, seeking to identify those problems which would benefit most from the GPU's parallel computational power. Neural networks, which consist of many identical neuron models computing an output based on input from their neighbors were an obvious target and artificial neural networks have indeed provided a well-known success, seeding an entire 'deep learning' industry. Here, the GPU speedup of the backpropapagation of error makes possible the *training* of very large networks that can solve difficult problems such as driving a car [4, 5], playing board games such as chess [16, 9] or Go [14] and console games such as Pong [12].

Researchers in the field of computational neuroscience have explored the use of the GPU to simulate neurophysiologically realistic neural networks [2, 17]. These generally involve

Corresponding Author: Sebastian James, seb.james@sheffield.ac.uk
Citation: James, S. (2021). Parallel prefix scan for the computation of axonal projection patterns in biological neural networks. *Academia Letters*, Article 461. https://doi.org/10.20935/AL461.

a complex neuron model in which the cell's membrane voltage, its ion channels, and the release of various neurotransmitters may be modeled to understand the network's behavior. Researchers have often focussed on the compute time required for *simulation of the network* rather than for finding the connection parameters. Connection parameters may be found by a variety of optimization techniques, but the bottleneck is often the network simulation time. For simulation, the benefit which the GPU can offer depends on the complexity of the neuron model and the connectivity of the network; N neurons in a network can be simulated in parallel by N processing threads for only one timestep before their outputs must be transferred according to the network's connectivity. The GPU is wellsuited to the parallel execution of the neuron models, but not for the transfer of signals, which requires that GPU threads coordinate memory accesses. So the GPU is best suited for simulating high complexity neuron models operating in low complexity networks. The best speedups reported to date involve simulation of complex, multicompartment models [15]. Results are less favorable for networks involving the more parsimonious singlecompartment neuron models often employed *for their speed and efficiency* in computational neuroscience studies [13]. Given the significant additional complexity of GPU code development, the adoption of GPU computation to simulate neurophysiological neural networks has been limited.

However, there is one practical task in the development of neuroscience models which *is* amenable to GPU acceleration: the computation of connectivity patterns. It is common to define the connection patterns between populations in a biological neural network model according to the hypothesis or on the basis of experimental observations, rather than by a fromscratch training. Often, parameterised mathematical functions (Gaussians, Gabor functions, etc.) are employed. The computation of connectivity patterns between populations consisting of realistic numbers of neural elements can, however, be computationally demanding. The purpose of this letter is to show that the GPU is well suited to the task and to give a sample implementation in Python. The motivation for this work was the construction, in SpineCreator [7, 8], of a visual attention model in which a number of neural populations serve as image maps. Each population is formed into square grids with a side length, d, of 150. Thus, each population contains 22,500 neurons. Projections from one population to another take the form of Gaussian projections (see Fig. 1A). These projections have the strongest weights where the source neurons and the destination neurons are closest; neurons in the center of the source population project most strongly to those in the destination population. The weights drop off as the destination neurons become more distal from the source neurons, and below some thresholds, the weights are set to zero. Examples can be found in [11].

For narrow Gaussian projections, relatively few neural processes connect a given source neuron to elements in the destination population and a *weight table* can be constructed con-

Corresponding Author: Sebastian James, seb.james@sheffield.ac.uk
Citation: James, S. (2021). Parallel prefix scan for the computation of axonal projection patterns in biological neural networks. *Academia Letters*, Article 461. https://doi.org/10.20935/AL461.

taining source neuron index, destination index, and weight. However, to actually determine which source/destination pairs belong in the table, it is necessary to evaluate the connection weight between *every* source/destination pair. In our model, this results in 22,500 × 22,500 = 506,250,000 $n = d4$ weight pairs. The creation of this table is a *reducing operation*; 506,250,000 possible connections are reduced down to a table consisting of a few million nonnegligible weights.

Code A gives an example of Python code that carries out this computation using a general-purpose CPU. It implements the *SpineCreator connectionFunc API*, which defines arguments and a return data format for a Python function to compute the weight table. The example computes a *widening Gaussian*—the projection pattern widens along one axis of the (square) population, modeling the retinotopic projection from the primate retina (with its high acuity fovea) to the superior colliculus [11]. Complying with the SpineCreator connectionFunc API, it passes arrays of source neuron coordinates (**srclocs**), destination neuron coordinates (**dstlocs**), and a number of connection-specific (and userdefinable) parameters as arguments. It expects a table of connections to be returned. The returned table is used by SpineCreator as the connectivity definition and saved in the SpineML format [6].

Execution of the listing in Code A takes about 100 s on a fast (Intel i9) CPU. Although this is not a long wait, the modeler is likely to need to run the operation many times during model development, to experiment with different parameters in the connection patterns. The visual attention model which forms our example has at least 20 different projections, so changing a common parameter could result in nearly an hour of computation, making the model exceedingly tedious to work with. The majority of the time required for the computation occurs in the inner loop over the destination neurons (line 23). Because each computation in the innerloop is fully independent, the computation would seem to be a great candidate for execution on the GPU. However, at first sight, it appears to be necessary to transfer the 506,250,000 innerloop results from GPU memory to CPU memory, then carry out all 506,250,000 weight *comparisons* in order to sequentially build an ordered, reduced table of connections, effectively losing any performance gain provided by the GPU!

The solution is to make use of the *parallel prefix scan* algorithm[3]. This provides a way of computing the sum of a set of numbers by summing in blocks and propagating the partial sums until a final result is produced. Parallel prefix scan can be used to select out the nonzero weights with a method called *stream compaction* [10]. This involves summing 1 for each nonzero weight and using the cumulative sum as the 'line address' in the output table. This way, the output table (which must be preallocated in memory) can be populated in parallel by the GPU. Code B gives an implementation in Python, using Numba CUDA [1] to enable execution on NVIDIA GPU devices. The example follows the workefficient parallel scan [3]

Corresponding Author: Sebastian James, seb.james@sheffield.ac.uk
Citation: James, S. (2021). Parallel prefix scan for the computation of axonal projection patterns in biological neural networks. *Academia Letters*, Article 461. https://doi.org/10.20935/AL461.

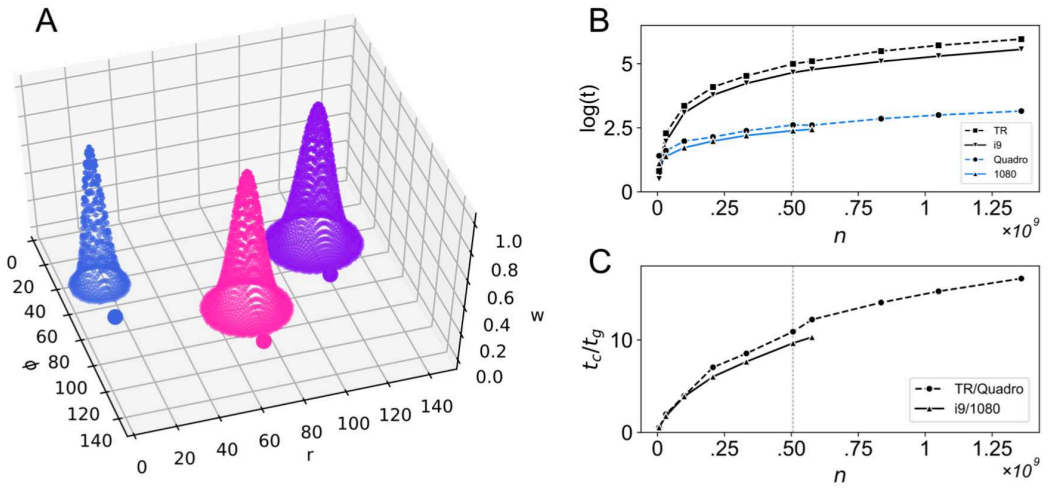

Figure 1: A Projection pattern computed using the widening gaussian defined in the referenced code (Code A and B). Projection weights are plotted for three source neurons, whose locations are shown by large circles. This projection has an offset in the ϕ direction (offsetd0p) of 25. Other parameters: sigma_m=100, W_cut=0.008, fshift=4, sigma_0=3. With these parameters, a weight table of 22,208,750 rows is generated. B Log of computation time (in seconds) plotted versus n, the maximum possible number of weights in a projection from a population of size d2to a second population of size d2. The dotted grey lines indicate n for the population size shown in A. Results are shown for an AMD Threadripper 2990WX CPU (TR), an Intel Core i98950HK CPU (i9), an NVIDIA Quadro P5000 GPU, and a GTX 1080 GPU. For small n, the overhead in the GPU algorithm makes it slower than the CPU algorithm. The crossover occurs at n $\approx$ 1.7 $\times$ 106 (d $\approx$ 64). C Speedup: The time taken on the CPU (tc) divided by the time taken on the GPU (tg) plotted vs. n for two different machines (the Quadro GPU was paired with the Threadripper CPU; 1080 with the i9). The overall speedup exceeds one order of magnitude for populations of the size shown in A. The data for the i9/1080 is truncated because the 1080 GPU did not have enough RAM to compute patterns for d > 155.

given in [10], taking into account features of the GPU's shared memory banks to avoid bank conflicts. The organization of the code may seem unusual, with all the GPU kernel functions being defined *within* the outer **connectionFunc** definition. This ensures that **connectionFunc** can be pasted into the Python script window of SpineCreator. Within **connectionFunc**, there are two main code blocks: i) **dowork** which carries out the parallel computation of the weights,

Corresponding Author: Sebastian James, seb.james@sheffield.ac.uk
Citation: James, S. (2021). Parallel prefix scan for the computation of axonal projection patterns in biological neural networks. *Academia Letters*, Article 461. https://doi.org/10.20935/AL461.

and is similar in structure to the code in Code A; and ii) **reduce_nonzero_gpu**, which carries out the reduction operation, extracting the nonzero weights into the final output table.

Execution of the code takes about 6 seconds on a GTX 1080 GPU for populations of 150×150 neural elements. Computation of the weights by **dowork** takes just 1 millisecond; the reduce operation, **reduce_nonzero_gpu**, requires about 5.8 seconds. When integrated into SpineCreator, this makes the workflow of modifying projection parameters feasible. Fig. 1B shows GPU and CPU times plotted vs. the problem size, n. The graph in Fig. 1C shows that the GPU speedup for larger populations tends towards about 18 times faster than the CPU code.

It is obvious that the GPU implementation code is significantly more complex than that for the CPU in Code A. The implementation consists of roughly ten times as many lines of code. Code A was coded in less than an hour; Code B took weeks of effort. The order of magnitude increase in complexity is justified by the order of magnitude speedup achieved by the GPU code. Additionally, much of the GPU solution code is 'boilerplate': To change the form of the projection, perhaps to implement a new Gabor projection in place of the Gaussian connectivity, only the **dowork** function needs to be reimplemented. Code B thus provides to the computational neuroscience research community a useful template for GPU computation of connection patterns.

Code A refers to: https://github.com/ABRG-Models/VisualAttention/blob/master/ connectionFuncs/offset_retgauss_forpaper.py

Code B re https://github.com/ABRG-Models/VisualAttention/blob/master/connectionFuncs/ offset_retgauss_gpu_forpaper.py.

Acknowledgments

I'd like to thank Paul Richmond at The University of Sheffield for originally suggesting the use of the parallel prefix scan algorithm. This work is supported by a Collaborative Activity Award, Cortical Plasticity Within and Across Lifetimes, from the James S McDonnell Foundation (grant 220020516).

References

1. Anaconda, Inc. Numba for CUDA GPUs documentation, 2012.

2. M. Beyeler, N. Oros, N. Dutt, et al. *Neural Networks*, 72:75–87, Dec. 2015.

Corresponding Author: Sebastian James, seb.james@sheffield.ac.uk
Citation: James, S. (2021). Parallel prefix scan for the computation of axonal projection patterns in biological neural networks. *Academia Letters*, Article 461. https://doi.org/10.20935/AL461.

3. G. E. Blelloch. Prefix Sums and Their Applications. Technical Report CMUCS90190, Carnegie Mellon University, School of Computer Science, 1990.

4. M. Bojarski, D. Del Testa, D. Dworakowski, et al. *arXiv:1604.07316 [cs]*, Apr. 2016. arXiv: 1604.07316.

5. M. Bojarski, P. Yeres, A. Choromanska, et al. *arXiv:1704.07911 [cs]*, Apr. 2017. arXiv: 1704.07911.

6. A. Cope and P. Richmond. SpineML, 2014. RRID: SCR_015641.

7. A. J. Cope, P. Richmond, and S. S. James. SpineCreator, 2015. RRID: SCR_015637.

8. A. J. Cope, P. Richmond, S. S. James, et al. *Neuroinformatics*, Sept. 2016.

9. O. E. David, N. S. Netanyahu, and L. Wolf. DeepChess: EndtoEnd Deep Neural Network for Automatic Learning in Chess. In A. E. Villa, P. Masulli, and A. J. Pons Rivero, editors, *Artificial Neural Networks and Machine Learning – ICANN 2016*, Lecture Notes in Computer Science, pages 88–96, Cham, 2016. Springer International Publishing.

10. M. Harris, S. Sengupta, M. Garland, et al. Chapter 39. Parallel Prefix Sum (Scan) with CUDA, 2010.

11. S. S. James, C. Papapavlou, A. Blenkinsop, et al. *Frontiers in Neuroscience*, 12, 2018.

12. V. Mnih, K. Kavukcuoglu, D. Silver, et al. *arXiv:1312.5602 [cs]*, Dec. 2013. arXiv: 1312.5602.

13. J. M. Nageswaran, N. Dutt, J. L. Krichmar, et al. *Neural Networks*, 22(5):791–800, July 2009.

14. D. Silver, T. Hubert, J. Schrittwieser, et al. *Science*, 362(6419):1140–1144, Dec. 2018. Pub lisher: American Association for the Advancement of Science Section: Report.

15. M. Stimberg, D. F. M. Goodman, and T. Nowotny. *Scientific Reports*, 10(1):410, Jan. 2020. Number: 1 Publisher: Nature Publishing Group.

16. S. Thrun. *Advances in neural information processing systems.*, pages 1069–1076, 1995. 17. E. Yavuz, J. Turner, and T. Nowotny. *Scientific Reports*, 6(1):18854, Jan. 2016. Number: 1 Publisher: Nature Publishing Group.

Corresponding Author: Sebastian James, seb.james@sheffield.ac.uk
Citation: James, S. (2021). Parallel prefix scan for the computation of axonal projection patterns in biological neural networks. *Academia Letters*, Article 461. https://doi.org/10.20935/AL461.