Probabilistic Programming in Multicore OCaml

Arnhav Datar, CS18B003
Dr. KC Sivaramakrishnan

January 2021

Research Problem

Probabilistic programming languages(PPLs) are used for performing Bayesian inference [2] in complex generative models. Probabilistic Programming [10] involves the construction of inference problems and the development of corresponding evaluators, that computationally characterize the denoted conditional distribution.

Probabilistic programming has shown applications [7] in the fields of computer vision, cryptography, quantitative finance, biology and reliability analysis. Given the recent rise of probabilistic programming for academic and industry use, we aim to make a Probabilistic Programming Library in Multicore OCaml.

OCaml is becoming very popular in various applications such as advanced financial management, transparency of global financial markets, phylogenetic inference, social media innovations etc. Absence of concurrent probabilistic programming libraries in OCaml is putting limitations on the usage. Given that the inferences done in probabilistic programming can be computationally heavy, the library will be a valuable resource for the OCaml community.

Until recently, probabilistic programming was limited in scope, and most inference algorithms had to be written manually for each task. Recently with availability of higher computing power and advances in concurrency, the probabilistic programming is becoming ubiquitous. To support the higher computational demand and to improve the performance we would like to explore effect handlers in Multicore OCaml to implement concurrent algorithms.

Literature Review

There have been many probabilistic programming efforts both as libraries and as independent languages. The development of these languages has sped up in the past few years, there being a lot of languages/libraries developed in the last 5 years. Some noteworthy mentions are Stan[5], Pyro[4], HackPPL[1], CuPPL[6], OwlPPL[12] and LF-PPL[15].

To the best of our knowledge the following probabilistic programming works are associated with OCaml and effect handlers. IBAL [11] and Hansei [9] were PPLs which were implemented in OCaml. However the languages weren't universal languages. Universal probabilistic languages can have an unlimited number of random variables. Furthermore they are not implemented using effect handlers and they aren't parallelised.

More recently Anik Roy made a universal PPL using Owl[14](a general-purpose numerical library in OCaml) known as OwlPPL[12], however it did not use effect handlers,

like its predecessors. It used a variety of inference algorithms ranging from simple enumeration to MCMC methods like Metropolis-Hastings. While these algorithms are properly implemented and analysed, it does not include more recent successful inference algorithms like the Hamiltonian Monte Carlo[3] and the No U-Turn Sampler[8].

Effect handlers have been used for developing modern universal PPLs like Hack-PPL[1] (Facebook's PPL) and Pyro[4](Uber's PPL). Pyro was built on Poutine which is a library of Effect handlers. However Hack-PPL is a separate programming language and Pyro is a library in Python/Pytorch. Edward[13] is a modern probabilistic modelling library that supports concurrency, but it does not use effect handlers(uses Tensorflow) and is not universal.

Multicore OCaml currently has support for algebraic effect handlers as a way of supporting concurrency primitives. In this project we propose to modify Anik Roy's OwlPPL to add other recent concurrent inference algorithms.

Timeline

Given below is rough Gantt Timeline for the project. As can be seen we have divided the work in segments of coding, learning, testing, evaluation and report writing. Each section has 2-3 subsections.

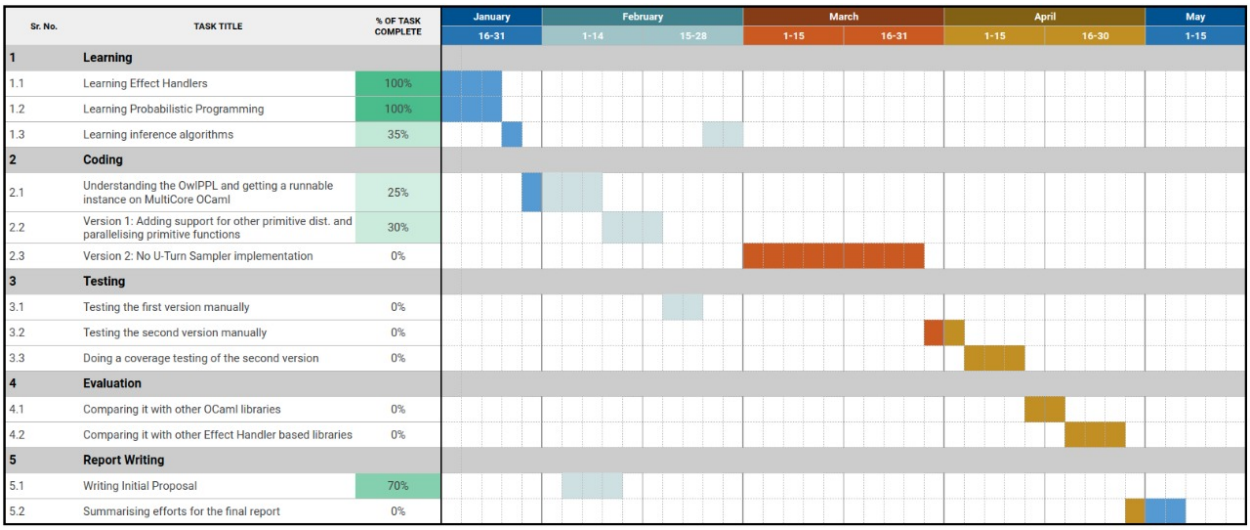

Figure 1: Brief Timeline, A column represents roughly 3 days

Currently OwlPPL has support for only 8 primitive distributions. These are binomial, normal, categorical, uniform(Discrete and Continuous), beta, gamma and Bernoulli. This is significantly lower than most other libraries. We will first remedy this as well as go through the code in the initial phases of the project. For implementing the primitive distributions we intend to use Owl[14], wherever possible. We will also try to make other primitive functions like sample_n(a function which returns n samples from a distribution) concurrent. We will do a manual testing of the code so far at this point.

Following this we will implement the No-U-Turn Sampler[8] with the leap-frog integrator, using Effect Handlers. This is the main portion of the project and will therefore take a significant amount of time. The choice of No-U Turn sampler is based on the fact that it is parallelisable and has shown promising results in converging to the required value/distribution. Furthermore most of the modern PPLs(Stan, Pyro, Hack-PPL et.al.) have used this sampler.

If we seem to be going ahead of schedule, we would try to implement the original inference algorithms in the OwlPPL library using effect handlers. Some of the algorithms like the particle cascade can clearly be parallelised. However algorithms like the Metropolis Hastings is a Markov Chain Monte Carlo, hence making it difficult to be parallelised.

We will post this stage do a manual as well as coverage testing of the algorithms implemented. We will then move on to evaluating the performance of our library to other similar works for some benchmarks. These benchmarks will test performance on single core as well as multiple cores, and will also test the performance based on number of samples.

References

[1] Jessica Ai et al. "HackPPL: A Universal Probabilistic Programming Language". In: *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. MAPL 2019. Phoenix, AZ, USA: Association for Computing Machinery, 2019, pp. 20–28. ISBN: 9781450367196. DOI: 10.1145/3315508.3329974. URL: https://doi.org/10.1145/3315508.3329974.

[2] Atilim Güneş Baydin et al. "Etalumis". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Nov. 2019). DOI: 10.1145/3295500.3356180. URL: http://dx.doi.org/10.1145/3295500.3356180.

[3] Michael Betancourt. *A Conceptual Introduction to Hamiltonian Monte Carlo*. 2018. arXiv: 1701.02434 [stat.ME].

[4] Eli Bingham et al. *Pyro: Deep Universal Probabilistic Programming*. 2018. arXiv: 1810.09538 [cs.LG].

[5] Bob Carpenter et al. "Stan : A Probabilistic Programming Language". In: *Journal of Statistical Software* 76 (Jan. 2017). DOI: 10.18637/jss.v076.i01.

[6] Alexander Collins and Vinod Grover. *Probabilistic Programming with CuPPL*. 2020. arXiv: 2010.08454 [cs.PL].

[7] Andrew D. Gordon et al. "Probabilistic Programming". In: *Future of Software Engineering Proceedings*. FOSE 2014. Hyderabad, India: Association for Computing Machinery, 2014, pp. 167–181. ISBN: 9781450328654. DOI: 10.1145/2593882.2593900. URL: https://doi.org/10.1145/2593882.2593900.

[8] Matthew D. Hoffman and Andrew Gelman. *The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo*. 2011. arXiv: 1111.4246 [stat.CO].

[9] Oleg Kiselyov and Chung-Chieh Shan. "Embedded Probabilistic Programming". In: *Proceedings of the IFIP TC 2 Working Conference on Domain-Specific Languages*. DSL '09. Oxford, UK: Springer-Verlag, 2009, pp. 360–384. ISBN: 9783642030338. DOI: 10.1007/978-3-642-03034-5_17. URL: https://doi.org/10.1007/978-3-642-03034-5_17.

[10] Jan-Willem van de Meent et al. *An Introduction to Probabilistic Programming*. 2018. arXiv: 1809.10756 [stat.ML].

[11] Avi Pfeffer. "The Design and Implementation of IBAL: A General-Purpose Probabilistic Language". In: *Applied Sciences* (Jan. 2000).

[12] Anik Roy. "A probabilistic programming language in OCaml". In: (2020).

[13] Dustin Tran et al. “Deep probabilistic programming”. In: *International Conference on Learning Representations*. 2017.

[14] Liang Wang. *Owl: A General-Purpose Numerical Library in OCaml*. 2018. arXiv: 1707.09616 [cs.MS].

[15] Yuan Zhou et al. *LF-PPL: A Low-Level First Order Probabilistic Programming Language for Non-Differentiable Models*. 2019. arXiv: 1903.02482 [cs.LG].