Contents lists available at ScienceDirect

Nuclear Instruments and Methods in Physics Research A

journal homepage: www.elsevier.com/locate/nima

Technical Notes

Code optimisation in a nested-sampling algorithm

S.J. Lewis [b], D.G. Ireland [b,*], W. Vanderbauwhede [a]

[a] School of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK
[b] SUPA School of Physics and Astronomy, University of Glasgow, Glasgow G12 8QQ, UK

A R T I C L E I N F O

Article history:
Received 2 February 2015
Accepted 2 March 2015
Available online 10 March 2015

Keywords:
Monte Carlo methods
General-Purpose computation on Graphics Processing Units (GPGPU)
Data analysis

A B S T R A C T

The speed-up in program running time is investigated for problems of parameter estimation with nested sampling Monte Carlo methods. The example used in this study is to extract a polarisation observable from event-by-event data from meson photoproduction reactions. Various implementations of the basic algorithm were compared, consisting of combinations of single threaded versus multi-threaded, and CPU versus GPU versions. These were implemented in OpenMP and OpenCL. For the application under study, and with the number of events as used in our work, we find that straightforward multi-threaded CPU OpenMP coding gives the best performance; for larger numbers of events, OpenCL on the CPU performs better. The study also shows that there is a "break-even" point of the number of events where the use of GPUs helps performance. GPUs are not found to be generally helpful for this problem, due to the data transfer times, which more than offset the improvement in computation time.

1. Introduction

Many data analysis tasks in nuclear and particle physics are parameter estimation problems. Values of parameters are found by comparison of a data model function with distributions of measured data. A common, and frequently satisfactory, approach is to maximise a likelihood function, and summarise the information about the parameters by taking the point of maximum likelihood and examining the behaviour in the vicinity of the maximum to estimate the uncertainty in the extracted value of the parameter. This is implemented in gradient-based searches that are the default in packages such as MINUIT [1], and is simply referred to as "fitting".

In some cases, however, such an approach is not adequate to extract all available information from experimental measurements, and a full evaluation of a likelihood function is required. This is particularly true in cases where the measured data are sparse, or where the data model has correlations among parameters that are of higher order than simply linear. By evaluating the full likelihood function, one captures all the available information, but the disadvantage can be that the calculation of likelihood may be extremely demanding.

Modern techniques of Markov Chain Monte Carlo (MCMC) calculations are designed to sample complicated, multidimensional probability density distribution functions efficiently, but whereas a gradient-based optimisation may typically require of order 100 likelihood function evaluations, a typical MCMC calculation needs perhaps of order 10^4 evaluations. Depending on the complexity of the likelihood function this could result in significant computation time.

If measurements produce event-by-event data, a likelihood of each datum can be calculated and combined to give a total likelihood. Since the same calculations need to be performed for each event, this points to the use of parallelisation to help program speed-up. The advent of general purpose graphical processor unit (GPGPU) programming suggests that an implementation of an event-by-event likelihood calculation on a GPU might be the best way forward. Indeed, it has been shown [2] that for event-by-event maximum likelihood calculations in partial wave analysis, speed-ups of two or three orders of magnitude are possible over conventional CPU running.

In this paper we examine the consequences of evaluating a likelihood function of modest complexity, where the measurements consist of event-by-event data of modest numbers. Section 2 introduces the example problem, Section 3 describes the nested sampling algorithm and outlines the various implementations in software and hardware, and Section 4 presents the results.

2. Statement of problem

The example used in this paper is based on a two-body reaction in which a linearly polarised photon interacts with a proton target, producing a pseudoscalar ($J^P = 0^-$) meson and a baryon. We wish

* Corresponding author.
E-mail address: David.Ireland@glasgow.ac.uk (D.G. Ireland).

http://dx.doi.org/10.1016/j.nima.2015.03.006
0168-9002/

to determine the photon beam asymmetry, Σ, which is the difference divided by the sum of cross-sections for the two states of photon linear polarisation. This is achieved by measuring the distribution of mesons as a function of azimuthal angle ϕ, which is the angle between the reaction plane and the direction of the photon's linear polarisation (E-vector). We assume, for simplicity, that there is a fixed photon energy E_γ and centre-of-mass scattering angle $\theta^\star_{CM}$. In a real experiment, data would be sorted into bins with a range of E_γ and $\theta^\star_{CM}$, but the ranges would be minimised to extract the maximum physics information from the variation of observables as functions of E_γ and $\theta^\star_{CM}$. This example could be applied to any reaction in which the photon beam asymmetry is to be determined.

The cross-section as a function of the angle ϕ is given by

$$\sigma = \sigma_0(1 - P_\gamma \Sigma \cos(2\phi)) \tag{1}$$

where σ_0 is the unpolarised cross-section, P_γ is the degree of photon polarisation and Σ is the beam asymmetry we desire to extract.

We assume that it is possible to polarise the photon beam in such a way that the electric vector can be oriented either parallel ($\parallel$) or perpendicular ($\perp$) to a reference plane in the lab frame. An asymmetry between these two settings is then

$$A(\phi) = \frac{\sigma_\parallel(\phi) - \sigma_\perp(\phi)}{\sigma_\parallel(\phi) + \sigma_\perp(\phi)} = P_\gamma \Sigma \cos(2\phi). \tag{2}$$

By measuring the number of mesons as a function of ϕ in both these states, we obtain an estimator of the asymmetry:

$$\hat{A}(\phi) = \frac{N_\parallel(\phi) - N_\perp(\phi)}{N_\parallel(\phi) + N_\perp(\phi)}. \tag{3}$$

A complicating factor of potentially different numbers of incident photons in each of the two states is omitted in this example to aid clarity. The two luminosities are taken to be equal. We also assume that the degree of photon polarisation P_γ is the same for both settings, and is known accurately, so that we do not need to regard it as a nuisance parameter.

With this in mind, we see that the problem is simply a one-parameter problem, where it is knowledge of Σ that we desire to infer from the measured data. The likelihood (probability) of measuring $N_\parallel$ and $N_\perp$ events given a definite asymmetry value a is

$$\mathcal{P}(N_\parallel, N_\perp \mid a) = \frac{1}{Z}(1-a)^{N_\parallel}(1+a)^{N_\perp} \tag{4}$$

where Z is a normalising constant.

For each event, we need the meson azimuthal angle ϕ and the setting ($\parallel$ or $\perp$). For a given value of Σ, an asymmetry is calculated from Eq. (2). Eq. (4) then reduces to

$$\mathcal{P}(N_\parallel = 1, N_\perp = 0 \mid a) = \tfrac{1}{2}(1-a) \tag{5}$$

or

$$\mathcal{P}(N_\parallel = 0, N_\perp = 1 \mid a) = \tfrac{1}{2}(1+a) \tag{6}$$

depending on the setting. For M events, the total likelihood is then the product of the likelihoods of each event:

$$\mathcal{L} = \prod_{i=1}^{M} \mathcal{P}_i. \tag{7}$$

In realistic examples we expect something between 10^3 and 10^4 events, so if code can be parallelised to perform likelihood calculations on several events at once, a speed-up should be possible.

In this study to determine the best code implementation strategy for applications of this type, we have simulated the reaction with known values of Σ to generate events that we know to be free of detector peculiarities. We use a two-body phase-space generator where the azimuthal distributions of the mesons are modulated according to Eq. (1).

3. Implementation

3.1. Nested sampling

Nested sampling [3] is a form of MCMC, a Bayesian approach to inference problems. Whilst nested sampling has been applied to a specific hadron physics problem in this paper, it is a general algorithm that is applied to a wide range of problems.

The primary objective of nested sampling is to provide a sampling of a posterior probability density function, and calculate a value referred to in the literature as the *evidence*. The evidence is a quantity with which different data models can be compared, but in this application we are only interested in parameter estimation, so it is a by-product of the calculation. A *prior* probability density in parameter space is used in conjunction with the event-by-event likelihood function (Eq. (7)) to generate the posterior.

As with other MCMC applications, nested sampling works with a population of points in parameter space. The prior probability density is sampled to give an initial population, and for each point in this sampled prior, a likelihood value is calculated. In nested sampling, the point with the lowest likelihood is recorded and overwritten with a copy of a surviving point. This new point is then altered in an exploration step within parameter space, and its likelihood is calculated. If the resulting likelihood is lower than that of the overwritten point, the new point is moved again in a further exploration step. This process continues until the likelihood of the new point is found to be greater than that of the overwritten point. The algorithm then finds the next point with the lowest likelihood and the process is repeated until a given termination condition is met [4]. The general idea is that the current population will migrate to the regions of greatest probability. For consistency, the termination condition used in this work was a set number of iterations.

3.2. Data parallelisation

Since the clock speed of CPUs has stagnated, parallel programming has become the focus of computing performance development. The use of multicore processors and General-Purpose Graphics Processing Units (GPGPUs) has become mainstream in everything from scientific computing and state-of-the-art gaming technology to standard desktop computers and laptops. There is now a sustainable path to improving computing technologies for the foreseeable future. Although the spotlight is currently on GPGPU computing, it must be remembered that all programs and algorithms will include some amount of sequential code, even if it exists solely to execute kernel functions or perform or some standard initialisations. In most cases, these serial sections of a program create bottlenecks that no amount of parallelisation can avoid. For this reason, heterogeneous platforms – i.e. the combination of highly optimised CPU cores with the massively parallelisable GPU cores – have become increasingly popular. These two components must complement each other – if the CPU is outdated and obsolete, any speed-up obtained from a high-end GPU will be hidden by the slow processing at one of these bottlenecks. In order to make the most of the available hardware, both components must be taken into consideration.

Not all algorithms can be parallelised; recursive and sequential programs, or even serial sections of code, can form bottlenecks that impede the run-time of a program. There are some cases where parallelising data over multiple threads or cores can result in a slower run-time as no speed-up is gained and time is lost