Performance Optimization of the AI Feynman Symbolic Regression code

Dmitry Mikushin
Applied Parallel Computing LLC
dmitry@parallel-computing.pro

CONTENTS

Abstract

This report presents the AI Feynman programming code performance optimization.

I. OVERVIEW

AI Feynman [1] is a neural network package for reconstructing original numerical expression (formula) from the resulting dataset.

II. PROGRAM DESIGN

The program code is organized into the preprocessing (*feature extraction*) and neural network *training* phases.

The *feature extraction* is given as a massive bruteforce evaluation of arbitrary basis function combinations. In order to maintain acceptable running times, this phase is implemented as a native code (Fortran). Moreover, each instance of bruteforce evaluation is limited by 30 seconds.

III. PERFORMANCE ANALYSIS

The feature extraction code is highly suboptimal:

- The best fit search over up to 120000 iterations is done in a single thread;
- The outer loop for expression terms permutation is done in a single thread;
- Finally, the main loop for presets read for a data file is serial as well.

All three loops have semi-independent iterations. That is, an adequate synchronization of the *loss criteria* between iterations may improve the processing speed, yet is not mandatory. Technically, this workflow is an ideal fit for a *parallel reduction* (*map-reduce*). The volume of parallel iterations is large enough to fed massively mutithreaded processors (e.g. GPUs) and clusters.

IV. OPTIMIZATION

The Python package build system has been refactored to embed a separate native module extension. The native module incorporates the Fortran code, and is operated by CMake. This design simplifies compiler options tuning, debugging and further development, e.g. towards GPU support.

The multi-indexing *multiloop* iterator has been changed to stateless design. This enabled independent processing of permutation loop iterations.

The Fortran code has been reorganized along the parallel reduction construct. The actual multithreaded implementation is based on Thrust and TBB backend.

V. FURTHER WORK

Within this project, we have performed initial code refactoring and reduction streamlining for efficient high performance processing.

The following additional improvements may significantly contribute to the overall speedup:

- Deploy the prepared Thrust-based reduction on GPUs. This needs to refactor the Fortran code into CUDA Fortran or CUDA C.
- Optimize the function evaluator $f()$. Currently, this part has excessive *if-else* branching, which could be replaced with tables and predicates.

REFERENCES

[1] Udrescu, S.M., and Tegmark, M. 2020. AI Feynman: A physics-inspired method for symbolic regression. Science Advances, 6(16), p.eaay2631.