Revisiting Hot Paths in the Context of Machine Learning

Sam Epstein

March 2024

Overview

Recently there has been a explosion of usage of generative AI models on GPUs. For example, OpenAI uses 30,000 GeForce Invidia chips. However the usage of such such chips has been limited to AI applications. This note introduces the question:

Can GPU processors speed up CPUs?

To this end, I propose a hybrid computer, with a CPU connected to a chip that performs pattern matching algorithms. The pattern matching component, or "cache' will identify hot paths in dynamically run code and conditions which allows a jump in computation. Thus the idealized cache will consist of rules of the form:

$$(A, B, C) \xrightarrow{pc_1, pc_2} (D, E, F, G)$$

The above rule jumps from location pc_1 to pc_2. The (A, B, C) condition represents three registers and/or memory locations, along with their values. The (D, E, F, G) represents the consequences of the code. This note introduces the following two machine learning problems. What are the class of pattern matching algorithms that can dynamically perform the following tasks on very large programs?

1. Identify *hot paths*, that is code where most of the execution occurs.

2. Once a hot path has been identified, how to create cache rules.

I now describe two algorithms fulfilling the above tasks.

Rule Creation

Assume that the start location of a hot path is known. The algorithm will continuously combine adjacent rules until a long enough rule is found with small enough side effects and conditions. To do the rule combinations, one needs to know what registers and memory locations need to be used in the future. We propose a change to the gcc compiler, so that when a stack is popped off or memory is freed a new risc-v instruction "FRE" will record the changes for use by the cache. This instruction is sufficient for good performance by the rule-merging algorithm.

Hot Path Detection

This algorithm uses the following assumption: *Hot paths are executed repeated in a small window of time.* The hot path algorithm will use n hash functions: $\{h_i\}$. During execution, for every m instructions run, the hot path algorithm will map the hash h_1 of the m instructions into one of ℓ bins, each initially set to 0. A counter in that bin is incremented. Once a bin b reaches capacity, it is expanded to a new set of ℓ bins. Thus if a set of instructions are mapped to b with h_1, it will then be mapped to a second tier bin using h_2. The increase in counters continues until a tier n bin reaches capacity, and at that point a hot path is detected.

Future Work

If a hot path occurs frequently, it will be predictable. One area of future work is using text prediction recurrent neural networks, which enjoys GPU support.