Alexander Powell

11/09/2016

CSCi 680

**Critique of “Cache-conscious Frequent Pattern Mining on a Modern Processor”**

Summary

The purpose of this paper was to examine the performance of frequent pattern mining algorithms on a modern processor. At a high level, the results show that even with the best FPM implementations and efficient memory management, these implementations still under-utilize modern processors. The authors cited the biggest causes of decreased performance on poor data locality and low instruction level parallelism. The paper presents three contributions to help alleviate the performance bottleneck problem. The first is improving cache performance of FPM algorithms by employing a tile-able cache conscious prefix tree. The second is leveraging hardware cache line prefetching to hide the cache miss latency. Thirdly, new algorithms are presented to take advantage of simultaneous multi-threading (SMT).

Pros and Cons

I believe sufficient resources were devoted to the background and related work sections in the paper, however some of the notation was difficult to follow. As a reader who is unfamiliar with frequent pattern mining algorithms, some of the concepts could have been explained more clearly and with more high level examples. However, there’s no doubt that the ideas presented in the paper were novel and groundbreaking, and can hopefully have a considerable impact on how frequent pattern mining is performed in the future.

Questions

1. In section 3.4, the paper states that since calls to malloc() and free() are computationally expensive, custom memory managers were designed to replace them. Do they mean new methods were written, or only one malloc (and free) were used, and then distributed? If new methods were written, how are they able to beat the performance of the original system calls? If they are simply using one big malloc call instead of a number of smaller ones, is it possible to have memory overflows when working with large prefix trees, since one contiguous block of memory equal to the total size of the prefix tree will be allocated?
2. When discussing the benefits of simultaneous multithreading, they mention that data fetched by one thread can be reused by another thread, which will reduce materialized cache miss latency. The authors state that this strategy was used during the FP-Growth() calls, but that it didn’t provide more than a 3% improvement. How come there was not a larger improvement, and could using more than 2 threads per tile have helped?