Worst-case optimal joins on modern hardware

Richard Gankema

CWI

While binary join plans have become the de-facto standard of join query processing, recent work has shown that these plans are suboptimal for some classes of queries. A typical example is the query `T(a,b,c) := E(a,b), E(b,c), E(a,c)`, which finds all triangles in an undirected graph. Because there are typically many more paths of length two than there are triangles, this query typically produces substantially larger intermediate results than the final results for any binary join plan, which can significantly hurt performance.

In an attempt to solve this problem, a new class of join algorithms was developed, called the worst-case optimal join (WCOJ). WCOJs are multi-way joins, and do not produce any intermediate results. The defining feature of a WCOJ algorithm is that its asymptotic runtime complexity is bounded by the output size of the query. This is achieved by not joining relation-at-a-time, but variable-at-a-time instead. Examples of WCOJ algorithms are NPRR and Leapfrog Triejoin (LFTJ).

Although WCOJ algorithms have been implemented in a number of systems, they are still nascent compared to heavily optimized join algorithms such as the binary hash join, and there are still numerous open questions with regards to their most efficient implementations. In this research, we take LFTJ as the running example for a WCOJ algorithm, and explore optimizations that can be done to bring its performance closer to what's possible on modern hardware. These optimizations include (1) compressed execution, (2) employing hash table probes instead of binary searches were appropriate, (3) using SIMD instructions to speed up intersections of sorted integers and (4) efficient index creation when none are readily available.

Our current contributions are new compression schemes that work well on sorted data, yet offer random access directly in the compressed data, as well as an optimized implementation of LFTJ employing vectorized hash lookups and SIMD instructions providing speedups of over 5× over its original implementation. Note that this work is still in progress, and all results are preliminary.