Abstract

論 文 の 内 容 の 要 旨

論文題目	Efficient Exploitation of SIMD Instructions in Non-Numerical Applications (SIMD命令の効率的な活用による 非数値計算アプリケーションの高速化)
氏　　名	井上 拓

To achieve high performance on today's systems, it is critically important to exploit different types of parallelisms available in algorithms by mapping them onto hardware parallelisms. For example, multiple cores and multiple SMT threads in a core can accelerate applications by executing multiple threads simultaneously. Another important processor feature to accelerate compute-intensive workloads is Single Instruction Multiple Data (SIMD) instructions, which can operate on multiple data in one instruction, to exploit data parallelism. Many high-performance processors support the SIMD instructions, such as the SSE and AVX instruction set of the x86 processors or the VMX and VSX instruction set of the PowerPC processors. To fully utilize the huge computing capability of today's processors, the programmers need to identify the thread-level and data parallelism available in the algorithms. Hence, there are many existing research projects to enhance important algorithms for *parallelizing* with multiple threads or *vectorizing* with SIMD instructions. The SIMD instructions have been widely used in many scientific computing workloads (such as matrix computations), image processing workloads (such as movie encoding and decoding), and basic string operations since it is mostly straightforward to vectorize these algorithms. However, there are still many important algorithms and workloads we cannot efficiently exploit SIMD instructions.

In this dissertation, we present new high-performance algorithms for efficiently exploiting SIMD instructions on the following three key operations: sorting for integer values, sorting for structures, and sorted set intersection. These algorithms are important building blocks of many non-numerical applications, such as database management systems and search engines. We showed that our proposed algorithms improve the performance over scalar algorithms and existing SIMD algorithms by efficiently exploiting SIMD instructions.

An obvious advantage of the SIMD instructions is the degree of data parallelism available in one instruction. In addition, they allow programmers to reduce the number of conditional branches in

their programs. For example, a program can select the smaller or larger value from each element's pair of two vectors without conditional branches. For another example, a program can aggregate multiple conditional branches by using vector-comparison-based conditional branches, such as branch-if-all-equal instruction supported in most of SIMD instruction sets. On superscalar processors with long pipeline stages, conditional branches can potentially incur pipeline stalls and thus significantly limit the performance. The benefit of reduction in the number of conditional branches is potentially significant for many non-numerical workloads since the branch misprediction overhead is often larger for the non-numerical workloads compared to typical numerical applications such as matrix computations with regular control flows. For example, it was reported that SIMD instructions can accelerate many database operations, such as scan operations and nested-loop join operations, by removing branch misprediction overhead.

In this dissertation, we study efficient sorting algorithm since sorting has been one of the most important building blocks for many applications, such as database management systems. Hence many sequential and parallel sorting algorithms have been studied in the past. However, SIMD instructions in today's processors have limitations and popular sorting algorithms, such as quicksort, are not suitable to exploit SIMD instructions efficiently due to its scattered memory accesses. We propose a new high-performance sorting algorithm suitable for exploiting both the SIMD instructions and thread-level parallelism available on today's multicore processors. Our main contribution includes a new high-performance sorting algorithm that can effectively exploit SIMD instructions. It consists of two algorithms: a vectorized mergesort and a vectorized combsort. In our vectorized combsort, it is possible to eliminate all unaligned memory accesses from combsort. For the vectorized mergesort, we proposed a novel linear-time merge algorithm that can take advantage of the SIMD instructions. We combine these two algorithms. First we divide data into small blocks and sort them using the vectorized combsort. Then, we merge them using the vectorized (multiway) mergesort. We show that our algorithm using SIMD instructions outperformed other implementations of comparison-based sorting algorithm such as STL's std::sort, which implements a quicksort variant, and a SIMD implementation of the bitonic mergesort when sorting a large array of random 32-bit integers. Comparing against an optimized radix sort, our algorithm achieved almost comparable performance using 128-bit SIMD instructions and better performance using 256-bit SIMD instructions. Also, our new algorithm showed better scalability with increasing number of cores than the radix sort and the bitonic mergesort.

We also extend our new sorting algorithm for sorting an array of structures instead of an array of integers. In real workloads, sorting is mostly used to rearrange structures based on a sorting key included in each structure. Here, we call each structure to be sorted a record. For sorting large records using SIMD instructions, a common approach is to first pack the key and index for each record into an integer value, such as combining each 32-bit integer key and a 32-bit index into one 64-bit integer value. The key-index pairs are then sorted using SIMD instructions, and the records are finally rearranged based on the sorted key-index pairs. This *key-index approach* can efficiently exploit SIMD instructions because it sorts the key-index pairs while packed into integer values, allowing it to use existing high-performance sorting implementations for integers. However, the key-index approach causes frequent cache misses in the final rearranging phase due to its random memory accesses, and this phase limits both single-thread performance and scalability with multiple cores. We developed a new stable sorting algorithm that can take advantage of SIMD instructions while avoiding the frequent cache misses caused by the random memory accesses. The main contribution on this topic is a new approach in the multiway mergesort for sorting an array of structures, which can effectively exploit the SIMD instructions while avoiding the random memory accesses. Avoiding the waste of memory bandwidth due to random memory accesses is quite important with multicore processors because the total computing capability of the cores in a processor has been growing faster than the memory bandwidth to the system memory. Our results showed that our new approach achieved up to 2.1x better single-thread performance than the key-index approach implemented with SIMD instructions when sorting 16-byte records. Our approach also yielded better performance when we used multiple cores.

Set intersection, which selects common elements from two input sets, is another important workload we study in this dissertation. It is a fundamental operation in many applications, including multi-word queries in Web search engines and join operations in database management systems. For example, in Web search engines the set intersection is heavily used for multi-word queries to find documents containing two or more keywords by intersecting the sorted lists of matching document IDs from the individual query words. In such systems, the performance of the sorted set intersection often dominates the overall performance. We describe our new algorithm to improve the performance of the set intersection. Unlike most of the existing advanced techniques, we focus on improving the execution efficiency of the set intersection on the microarchitectures of today's processors by reducing the branch mispredictions. Moreover, we can effectively eliminate many of the comparisons by aggregating multiple comparisons and conditional branches with one branch based on a SIMD comparison. Our algorithm roughly doubled the performance for set intersection for 32-bit and 64-bit integer datasets even without using SIMD instructions compared to the std::set_intersection implementation delivered with gcc. The use of SIMD instructions further doubled the performance on both processors.

In these three new algorithms, the key to achieve high performance is 1) to exploit data parallelism available in the algorithm and 2) to reduce the number of conditional branches 3) while avoiding non-contiguous memory accesses, which increases the memory access overhead. Although the data parallelism available in one instruction is an obvious and straightforward advantage of the SIMD instructions, the reduced branch misprediction overhead also gives non-negligible performance gain; and hence the advantage of the SIMD instructions can surpass the data parallelism of the SIMD instruction. For example, we demonstrated 8.0x to 11.9x performance improvement using 4-wide SIMD instruction (SSE for 32-bit integers) in various sorting algorithms that are suitable for vectorizing with SIMD instructions..

To reduce the number of branch mispredictions, we take two different approaches for sorting and set intersection. For the set intersection, we aggregate multiple conditional branches into one since the direction of most of the conditional branches is same. For sorting, on the other hands, we replace conditional branches by SIMD minimum and maximum instructions. In sorting, especially for random numbers, the directions of conditional branches are mostly unpredictable and the branch directions are divergent; hence it is not effective to aggregate multiple branches. By replacing control flow of the unpredictable conditional branches into a data flow by arithmetic instructions avoid the huge overhead of branch mispredictions and hence very effective to improve the performance. Although we take different approaches for handling conditional branches, some of the optimization techniques are common among our proposed algorithms. For example, using a smaller data type instead of a larger type to increase the data parallelism in one instruction is an important technique to get larger performance gain in sorting of structures and set intersection. Typically, using a small data type does not improve the computation performance with scalar processing, and hence it is unique to SIMD processing.

In addition to the superior performances with SIMD instructions, we have demonstrated that we can improve the energy efficiency (performance per Watt) using SIMD instructions efficiently. The energy efficiency is critically important for computing systems today ranging from super computers to mobile devices. We observed only small increase in energy consumption in trade for huge performance boost for both sorting and set intersection. Using SIMD increases energy consumption in vector ALUs, but it also reduces the execution time. In total, we observed significant improvement in the energy efficiency. Hence, our results show that our new algorithms can contribute wide range of applications and systems.