In the past decades, the microprocessor technology has delivered significant improvements in clock speed. Moore’s observation has shown the clock speed was doubled every eighteen months. It means the performance gain also can be doubled every eighteen months. However, several physical limitations such as heat generation and the size constrain of transistor have diluted this trend significantly.

In order to continue the performance gain legend, the many-core processor has been designed. Meanwhile, the demand on parallel algorithms also appeared. Parallel algorithm, on the other hand, uses multiple processing elements simultaneously to solve a problem. This is accomplished by breaking the problem into independent parts so that each processing element can execute a part of the algorithm simultaneously with the others. The processing elements can be diverse and include resources such as a single computer with multiple processors, several networked computers, specialized hardware, or any combination of the above.

Now the many-core processors such as the multicore processors and GPUs have been used for various applications because of their processing power, scalability, and portability of codes. In this dissertation, we have presented several efficient parallel algorithms for the multicore processors and GPUs.

The background with motivation and the introduction of this dissertation are presented in Chapter 1.

In Chapter 2 of the dissertation, we have presented an efficient multicore sorting compatible with qsort. Our multicore sorting is implemented such that its interface is compatible with qsort in C Standard Library and can sort arrays of any size, containing any kind of object and using any kind of comparison predicate. By replacing calls to qsort with our multicore sorting, the sequential sorting is replaced with our parallel sorting easily. Also, we have implemented and evaluated our algorithm in a Linux server with four Intel hexad-core processors (Intel Xeon X7460 2.66GHz). The experimental results have shown that our multicore sorting is 11 times faster than original qsort. Since the speed up factor cannot be more than 24 if we use 24 cores, our algorithm is efficient.

In Chapter 3, we have presented an optimal parallel algorithm for computing Euclidean Distance Map (EDM) of a 2-D binary image. Using proximate points problem as preliminary foundation, we have proposed a simple but efficient parallel EDM algorithm which can achieve $O(n/k)$ time using $k$
processors. To evaluate the performance of the proposed algorithm, we have implemented it in a Linux server with four Intel hexad-core processors (Intel Xeon X7460 2.66GHz) and two different GPU (Graphics Processing Unit) systems, Tesla C1060 and GTX 480, respectively. The experimental results have shown that, for an input binary image with size of 9216×9216, the proposed parallel algorithm can achieve 18 times speedup in the multicore system, comparing with the performance of general sequential algorithm. Meanwhile, for the same input image, the proposed parallel algorithm can achieve 26 times speedup in that of GPU systems.

In the GPU architecture, there are many programming issues, such as coalesced access of global memory and shared memory bank conflicts etc, need to be considered. However, the GPU implementation of the parallel EDM algorithm shown in Chapter 3 is not enough to cope with these programming issues of GPUs. More specifically, in our implementation, 2-dimensional arrays are mainly accessed from/to the global memory four times. Unfortunately, two times of them cannot reap the benefit of the coalesced access.

In Chapter 4, we have shown an improved GPU implementation of the parallel EDM algorithm with more efficient memory access. In our new implementation, we have considered the programming issues of the GPU system such as coalesced access for global memory and shared memory bank conflicts. The new idea of our implementation is that we have improved the access for 2-dimensional arrays that are temporal data stored in the global memory which cannot be done with coalesced access in the previous implementation. To be concrete, transposing the 2-dimensional arrays with the shared memory, the access enables to be performed by coalesced access. We have implemented and evaluated our proposed parallel EDM algorithm in the following three GPU systems, Tesla C1060, GTX 480 and GTX 580, respectively. The experimental results have shown that for an input binary image with size of 9216×9216, our implementation can achieve a speedup factor of 54 over the sequential algorithm implementation. Also, we have presented that the density of black pixels in an input image affects the performance of the proposed GPU implementation.

Finally, this dissertation is concluded in Chapter 5.