論　文　の　要　旨

題　目　　Efficient Hardware Algorithms for the FPGA
（FPGA 向けの効率的なハードウエアアルゴリズム）

氏　名　周（シュウ）　昕（キン）

Field-Programmable Gate Array (FPGA) is a programmable silicon device designed to be configured by the customer using hardware description language after manufacturing. In the past, FPGAs are used for lower speed and complexity designs due to the lack of internal logic resources and low frequency of the FPGA. Today's FPGAs can easily run at high frequency and have unprecedented logic density. Furthermore, embedded processors, DSP slices, block RAMs are embedded in the FPGA. Also, ability of parallel processing is one of the most important features that separate FPGA from the conventional microprocessor.

In order to improve the processing speed, multicore processors are widely used in many application domains such as general purpose computation, digital signal processing, and image processing. Embedded multicore processors represented by FPGA has lately attracted considerable attention for their potential computation ability and power consumption. By partitioning the algorithm into several independent parts, multicore processors can perform all parts concurrently. If the algorithm is hard to be parallelized, we can also improve the processing speed considerably by employing multicore processor to perform the same algorithm for different data sets.

Hough transform is a technique to find shapes in images such as lines, circles, ellipses, etc. In this dissertation, we have presented implementations of the Hough transform on the FPGA for extracting lines and circles in Chapter 3. The Hough transform defines a mapping from an image into a parameter space represented by an accumulate array. For each edge point of the image, the mapping adds a vote to corresponding elements in the accumulate array. Therefore, the elements that are voted intensively represent associated parameters of detected shapes. The first contribution of this dissertation is to present an efficient implementation of the Hough transform on the FPGA. In our implementation, we partition the parameter space and the voting operation is performed in parallel by an efficient usage of DSP slices and block RAMs. As far as we know, there is no previously published work that fully utilizes DSP slices and block RAMs for the Hough transform. The experimental results show that our FPGA implementation attains a speed-up factor of more than 300 over the sequential implementation on the CPU by using 178 DSP slices and 180 block RAMs. However, this implementation needs to accept the coordinates of edge points as input. Also, since identified lines are obtained just by thresholding after voting, incorrect lines are also detected. Hence, the second contribution of this dissertation is to present an improved

FPGA implementation of the Hough transform. The improved FPGA implementation processes all pixel data given in raster scan order, and the usage of DSP slices reduces. Also, maximum filters are used to obtain the correct lines after voting operation. The improved implementation uses only 90 DSP slices and 181 block RAMs and attains a speed-up factor of more than 38 over the sequential implementation on the CPU. Next, gradient-based Hough transform is one of the efficient improvements to the Hough transform for line detection, where the gradient direction and magnitude of each pixel are used to reduce the number of useless votes for obtaining more precise lines. The third contribution of this dissertation is to present an efficient implementation of the gradient-based Hough transform on the FPGA. This implementation uses only 13 DSP slices and runs 309 times faster over the sequential implementation on the CPU. Furthermore, comparing with other FPGA implementations, the performance of our FPGA implementations is better. On the other hand, the Hough transform can be also used to extract circles. The fourth contribution of this dissertation is to present an efficient implementation of the Hough transform for circles detection on the FPGA, that uses only one-dimensional parameter spaces. Our implementation uses 398 DSP slices and 309 block RAMs and runs in 181.812MHz. According to the experimental results, our implementation attains a speed-up factor of approximately 189 over the sequential implementation on the CPU.

FPGA is also desired hardware device for general purpose computation. The Greatest Common Divisor (GCD) computation is widely used in computer systems for cryptography, data security and other important algorithms. Most of the time of these computer systems is consumed for computing the GCDs of very large integers. In this dissertation, the fifth contribution is to propose an efficient processor core that executes the Euclidean algorithm computing the GCD of two large numbers in an FPGA by using only one DSP slice and one block RAM in Chapter 4. Since the proposed processor core is compactly designed and uses very few resources, we have succeeded in implementing more than one thousand processor cores in an FPGA. The experimental results have shown that our implementation of 1280 GCD processor cores runs 3.8 times faster than the best GPU implementation and 316 times faster than a sequential implementation on the CPU.

Data compression is one of the most important tasks in the area of computer engineering. LZW algorithm is one of the most famous dictionary-based compression and decompression algorithms. Since dictionary tables are created by reading input data one by one, LZW compression and decompression are hard to parallelize. The sixth contribution of this dissertation is to present a hardware architecture of LZW compression and decompression in Chapter 5, respectively. Since the proposed modules of LZW compression and decompression use very few FPGA resources, we have succeeded in implementing 24 modules of LZW compression and 34 modules of LZW decompression in an FPGA, respectively. The experimental results show that, our implementation of 24 LZW compression modules attains a speed-up factor of 23.51 times faster than a sequential implementation on a single CPU, while our implementation of 34 LZW decompression modules attains a speed-up factor of 64.39 times faster than a sequential implementation on the CPU.