OpenMP를 이용한 새로운 COG 방식의 고속 병렬 퍼지 제어기

이상구*, 김진일**

요약

본 논문에서는 최근의 쿼드코아 PC 환경에서 OpenMP를 사용하여 고속 퍼지제어기를 위해 후건부의 정수형 픽셀 매핑을 통한 정수형 연산, 고속 비퍼지화를 위한 0 값 제거, 곱셈이 필요없는 새로운 COG 방식을 제안한다. 제안된 방법은 성능검증을 위해 에어컨 제어 모델에서 시뮬레이션하여, 종래의 싱글 프로그래밍에 의한 실수 연산 방식보다 10배 이상 빠른 것으로 확인됐다. 이러한 시스템은 실시간 환경에서의 로보틱 제어와 같은 응용을 위한 강력한 아키텍처로 활용될 수 있다.

A High-Speed Parallel Fuzzy Controller with new COG Method Using OpenMP

Sang-Gu Lee*, Jin-il Kim**

ABSTRACT

In this paper, we propose some of the following points for high-speed fuzzy controller using OpenMP in the recent QuadCore PC environment. (1) integer pixel mapping using only integer operations in the consequent part, (2) eliminating the zero items in the consequent part for fast defuzzification, and (3) novel COG method without multiplications. The proposed method was simulated for the air condition control system to verify its performance as compared to the conventional single programming systems. As the results of simulation, the proposed system is 10 times faster than the conventional method. This system can be applied to build powerful architectures for control applications, such as robotic control, with real-time environment.

Key Words : Fuzzy control, Integer operation, COG, VHDL, OpenMp

* 한남대학교 컴퓨터공학과
** 선문대학교 교양대학
· 제1저자(First Author) : 이상구 · 교신저자(Correspondent Author) : 김진일
· 접수일(2010년 1월 22일), 수정일(1차 : 2010년 2월 12일), 게재확정일(2010년 2월 19일)

Ⅰ. Introduction

After the introduction of fuzzy sets by L. Zadeh[1], fuzzy logic models are recently successfully being used in many applications, including automatic control, image processing, medical diagnosis, data mining, pattern recognition and robotics[2,3]. For them, very large floating-point operations in each fuzzy membership function are performed. In fuzzy computing, computations require the most time in the consequent part and the defuzzification stage rather than in the condition part. To overcome this problem, we use the integer scan line algorithm[4] to map the real values to integer grid. A method of eliminating the unnecessary operations of the continuous leading and trailing zero items in the defuzzification stage is also proposed. This allows a COG method to be implemented with integer additions and without multiplications. The proposed system is applied to an air condition control system for performance evaluation. The proposed system is an order of magnitude faster than the conventional methods using floating-point operations and introduces only a minimal error.

Ⅱ. High-speed Fuzzy System

In this paper, we propose a novel method to use only integer operations in the consequent part and the defuzzification stage. Here, we use an integer pixel grid in the consequent part composed of (M × N) pixels. After computing the α value, the degree of fulfillment, it is multiplied by N and converted to an integer by rounding. Let Round(α × N) be β. The integer β value is transferred to the consequent as a modified degree of fulfillment.

2.1 mapping algorithm

Integer mapping of the fuzzy membership function is the first step in the proposed algorithm[5]. Removing floating-point operations produces significant improvement in speed as compared to conventional methods. We have a 1024 × 64 integer pixel array for easy adapting to hardware implementation. In this grid, a segment of a membership function can be represented by connecting the integer pixels from the starting point pixel to the ending point pixel. Using mid-point technique[4], it is very efficient to compute the next y pixel value at given x pixel using only integer additions. To choose the next pixel, the first midpoint is $(x_p+1, y_p+\frac{1}{2})$. We need only to compute the value of $F(x_p+1, y_p+\frac{1}{2})$, and to test its sign. If the sign is positive, we select point (x + 1, y + 1) as a next pixel. If the sign is negative, we select the point (x + 1, y). If d is the decision variable for this choice, then

$$d_{start} = F(x_p+1, y_p+\frac{1}{2}) = a(x_p+1)+b(y_p+\frac{1}{2})+c$$

$$= a+\frac{b}{2} \rightarrow 2dy-dx \qquad (1)$$

If d < 0, then the next pixel is (xp+1, yp), so we select ‘E’(East). If d ≥ 0, then the next pixel is (xp + 1, yp + 1), so we select ‘NE’(North East).

If 'E' is selected, the next mid-point is incremented by 1 in the x direction. To find the next pixel, we have to compute the amount of variation, $\triangle E$, and update the decision variable d.

$$d_{new} = F(x_p+2, y_p+\frac{1}{2}) = a(x_p+2)+b(y_p+\frac{1}{2})+c$$

$$= 2a+\frac{b}{2} \rightarrow 4dy-dx \qquad (2)$$

$$d_{old} = F(x_p + 1, y_p + \frac{1}{2}) = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$$

$$= a + \frac{b}{2} \rightarrow 2dy - dx \quad (3)$$

$$\Delta E = d_{new} - d_{old} = 2dy \quad (4)$$

If 'NE' is selected, as the similar way,

$$\Delta NE = d_{new} - d_{old} = 2(dy - dx). \quad (5)$$

Fig. 1 shows the algorithm for integer line mapping with start point (x1,y1) and end point (x2,y2). Here, β has an integer value in the range $0 \leq \beta \leq 63$. defuzz(xa) is the value of the membership function in the consequent part when the value of the integer pixel in the x-axis is xa.

As the required part in the defuzzification stage is in trapezoidal form, the part from integer β+1 to 63in the y-axis point does not need mapping. Therefore, we must have mappings until the value of y is β. In order to map the membership function in Fig. 2, we first process line segment (P), then (R), and finally(Q). Following this order has the advantage of calculating the endpoints of segment (Q) before it is processed.

```
dx = x2 - x1; dy = x2 - y1;
d = 2*dy - dx;
xa = x1; ya = y1; a = β;
defuzz[xa] = a;
while (ya < a+1)
{   xa = xa + 1;
   if (d < 0)   {d = d + 2*dy;}
   else   {ya = ya + 1; d = d + 2*(dy-dx); }
   defuzz[xa] = ya;
 }
```

그림 1. 정수 매핑 알고리즘
Fig. 1 Integer mapping algorithm(left_line)

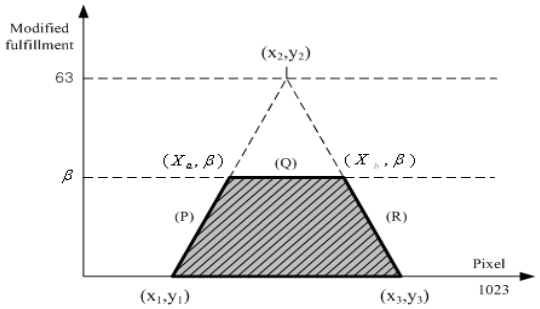

그림 2. 병렬 처리 부분
Fig. 2. Representation of the consequent part

Integer mapping in line (R) is similar to the process used to map line (P), except x is decremented for each iteration.

If d < 0, we select 'NW'. If d ≥ 0, we select 'W'. If 'NW' is selected, the next mid-point is decremented by 1 in the x direction and is incremented by 1 in the y direction. To find the next pixel, we compute the amount of variation, ΔNW, and update the decision variable d.

$$\triangle NW = d_{new} - d_{old} = -2dy - dx \quad (6)$$
$$= -2(dy + dx)$$

If 'W' is selected, as the similar way,

$$\Delta W = d_{new} - d_{old} = -2dy \quad (7)$$

After mapping lines (P) and (R) to the integer pixels, the final task is the mapping of line (Q) to the grid. To map line (Q), we must set y = β for all pixels from point (xa, β) to (xb, β). These two points are the ending points previously computed in lines (P) and (R).

This integer mapping method offers a significant advantage over the methods that use floating-point operations. However, this does require a tradeoff. In this algorithm, a quantization error between the real y value

and its equivalent integer pixel y value exists because this method selects the nearer integer pixel among two neighboring pixels. This quantization error is inversely proportional to the number of y-axis integer pixels. The root mean square value of the quantization error decreases as the number of y-axis integer pixels increases. In practice, the size of the y-axis pixels can be adjusted according to the application domain.

2.2 Defuzzification stage and new COG method

In this paper, we use COG method for defuzzification[5]. The required data structure in the defuzzification stage is an integer array that can store the y-axis integer values corresponding to the 1024 x-axis integer pixels in the consequent part. In this array, defuzz(x), y-axis integer values from 0 to 63 are stored. On computing the consequent part, this array must contain the maximum values in all fuzzy rules. Fig. 3 shows the algorithm for this function. Here, n is the number of the fuzzy rules, and m is the number of the pixels in the x-axis. After computing the consequent part, many values of defuzz(x) have values of zero continuously from either of the array.

```
int defuzz[m], max[m];
 for (i = 1; I <= n; i++)
 {
   for (j = 1; j <= m; j++)
  {
   pixel_mapping();
   // operation of integer pixel mapping;
   // return value ← defuzz[1:m];
   if (max(j) < defuzz(j))
   then max(j) = defuzz (j);
  }
 }
```

그림 3. 최대값 갱신 알고리즘
Fig. 3. Update algorithm of Max value

It is not necessary to compute any operations in that interval less than lower or greater than upper in the defuzzification process. To overcome this problem, we check x1 and x3 for each fuzzy rule, setting lower to the minimum of all x1 and upper to the maximum of all x3 .

```
for (i = upper; I <= lower; i--)
    {
    temp = defuzz( i ) + temp;
    sum = sum + temp;
     }
cog = sum / temp;
cog = cog + (lower - 1);
```

그림 4. COG 동작을 위해 제안된 알고리즘
Fig. 4. The proposed algorithm for COG operations

In this paper, we also propose an algorithm to eliminate the computation times for the multiplications. We use only integer additions to calculate the nominator of the COG operation. One integer division operation is still needed to compute the final COG. Fig.4 shows the algorithm to compute the nominator of the COG function. The final "temp" value in Fig. 4 becomes the denominator. In this method, for n non-zero items, only 2×n additions and 1 division are required.

III. OpenMP

3.1 Introduction to OpenMP

OpenMP (Open Multi-Processing) is an application programming interface for shared-memory parallel programming[6][7]. OpenMP Application Program Interface (API) may be used to explicitly direct multi-threaded, shared memory parallelism. OpenMP is comprised of 3 primary API components: Compiler

Directives, Runtime Library Routines and Environment Variables. The shared-memory model is based upon fork-join parallelism method. We can make the execution of a shared-memory program as periods of sequential execution alternating with periods of parallel execution. A master thread executes all of the sequential code. When it reaches a parallel code segment, it forks other threads. The threads communicate with each other via shared variables. At the end of the parallel code segment, these threads are synchronized, and rejoin the master thread.

OpenMP 3.0 was released in May, 2008[8]. Currently, OpenMP supports C/C++ and Fortran languages. Most major platforms have been implemented including Unix/Linux platforms and Windows.

Table 1 shows the comparisons between OpenMP and MPI(Message Passing Interface) programmings[9]. Both programming environments can be used to program in multiprocessor system environment. MPI is suitable for programming multicomputers especially. Since OpenMP has shared variables, OpenMP is not appropriate for generic multicomputers in which there is no shared memory.

표 1. OpenMP와 MPI의 비교
Table 1. Comparison between OpenMP and MPI

Characteristics	OpenMP	MPI
Suitable for multiprocessors	Yes	Yes
Suitable for multicomputers	No	Yes
Supports incremental parallelization	Yes	No
Minimal extra code	Yes	No
Explicit control of memory hierarchy	No	Yes

MPI also makes it easier for the programmer to take control of the memory hierarchy. On the other hand, OpenMP has the significant advantage of allowing programs to be incrementally parallelized. In addition, unlike programs using MPI, which often are much longer than their sequential counterparts, programs using OpenMP are usually not much longer than the sequential codes they displace.

In this paper, we use OpenMP for parallelizing the computation of fuzzy membership functions. We will compare the performance of the conventional single program based system(using floating-point numbers) and the proposed parallelizing method in OpenMP(using only integer numbers).

3.2 Parallelizing using OpenMP

There are 2 methods in parallelizing techniques in OpenMP. One is parallelizing in fine-grain and another is parallelizing in coarse-grain as shown in Fig. 5, respectively. In (a), two fork-join operations are performed. In the case of coarse-grain, fork-join operation is performed one time. In this paper, we use the parallelizing method in coarse-grain.

Compiler directives in C/C++ is called pragma (pragmatic information). Compiler directives specific to OpenMP in C/C++ are written in codes as like #pragma omp <rest of pragma>.

In the thread creation, omp is used to fork additional threads to carry out the work enclosed in the construct in parallel. The original process will be denoted as master thread with thread ID 0. Work-sharing constructs used to specify how to assign independent work to one or all of the threads.

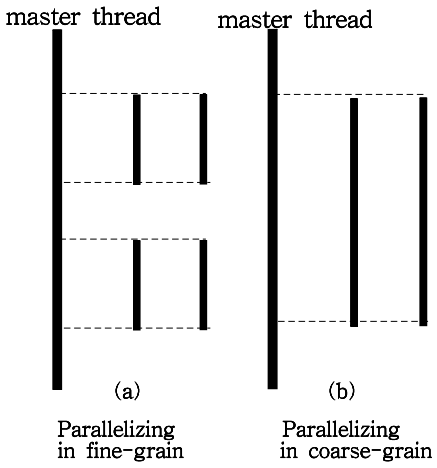

그림 5. 병렬화 방법
Fig. 5. Parallelizing methods

3.3 Proposed parallelizing Method

In OpenMP, the parallel pragma precedes a block of code that should be executed by all of the threads.

In this paper, we can parallelize the (P) part and (R) part in Fig. 2 using syntax of [#pragma omp parallel] in OpenMP. Independent jobs in the parallel region are executed separately in each threads.

```
omp_set_num_threads (2);
#pragma omp parallel
{
#pragma omp sections
#pragma omp section
  C-code for (P) part in Fig. 2
#pragma omp section
  C-code for (R) part in Fig. 2
}
```

그림 6. OpenMP에서 (p)와 (R)의 병렬화
Fig. 6. Parallelizing (P) and (R) parts in OpenMP

Ⅳ. Performance Analysis

We compare and analyze the execution times in one fuzzy inference cycle and the value of COG between the proposed method in OpenMP and the floating-number based single program algorithm among the conventional methods. The target object is an air condition control system. In this paper, 25 fuzzy rules and 7 linguistics of each membership function are used. We used OpenMP version 3.0. Simulation environment is on the recent QuadCore Q8400 PC system under Window 7 operating system.

표 2. 실행시간과 COG 비교
Table 2. Comparison of execution time and COGs at 20°C and 35% relative humidity (Environment: QuadCore Q8400 PC system, OpenMP ver. 3.0)

	(500×30) pixels			(1000×30) pixels		
	exec. time (μs)	COG	error (%)	exec. time (μs)	COG	error (%)
theoretic value	-	26.81	-	-	26.81	-
single program	1.82	26.84	0.1	2.41	26.83	0.1
proposed method on OpenMP	0.16	26.86	0.2	0.23	26.84	0.1

Test condition is for temperature of 20°C and relative humidity of 35%. We have tested our method with grid sizes of 500×30 and 1000×30 pixels, respectively. Results are summarized in the Table 2. As we can see in Table 3, COGs calculated using the conventional method and the proposed method are 26.84 and 26.86 for the 500×30 pixel array, respectively. For the 1000×30 array, the

표 3. 여러 퍼지 시스템과의 비교
Table 3. Comparisons of several fuzzy systems

	Proposed system	**[3]**	**[10]**	**[11]**
Fuzzy membership function shape	Triangular Possible in Trapezoidal	Triangular	Triangular	many
Fuzzy operators	Min/Max	Min/Max	Min/Max	many
Eliminating zero items	yes	no	no	no
Parallel structure	Coarse-grain parallel using OpenMP	yes	yes	no
Fuzzy computing	Integer grid	floating-point numbers	floating-point numbers	floating-point numbers
Defuzzification	No multiplications (COG)	Pipelining	Pipelining (COA)	Parallel processing
Membership function computing	Only Integer Additions	Lookup Table	Lookup Table	Floating point operations

COGs in the conventional method and the proposed method are 26.83 and 26.84, respectively.

The errors in the proposed method are 0.06 (500×30) and 0.04 (1000×30), or about 0.2% and 0.1%, respectively. Error terms mean the difference ratio between theoretic value(Integration method) and simulation result. However, the execution speed of the proposed method is over 10 times faster than that in the conventional method. It is noted that if we increase the pixel size, execution time is increased and the error is reduced even more.

For this implementation, the proposed parallel algorithm reduced the time needed to calculate system outputs by an order of magnitude. Table 3 shows the comparison results of several fuzzy systems. In the computation of each membership function, we note that the existing algorithms use either lookup tables or floating point operations.

V. Conclusion

The proposed method using OpenMP was simulated for the air condition control system to verify its performance as compared to the conventional systems. As the results of the simulation, the proposed system is 10 times faster than the conventional method for the air condition control system and an order of magnitude faster on general fuzzy control problems. This system can also be applied to build powerful architectures for control applications, such as robotic control, with time-critical sensor integration.

Acknowledgement

This work was supported by the research grant of the Hannam University in 2007.

References

[1] L. A. Zadeh, "Fuzzy Sets," *Information and Control 8*, pp. 338-353, 1965.

[2] J. Yen and R. Langari, *Fuzzy Logic: Intelligence*, Prentice Hall, Englewood Cliffs, NJ, USA, 1999.

[3] A. M. Ibrahim, *Fuzzy Logic for Embedded Systems Applications*, Elsevier, 2004.

[4] E. S. Angel, *Interactive Computer Graphics, 4th ed.*,

Addison-Wesley, 2007.

[5] S. G. Lee and John D. Carpinelli, "*Very High-speed Integer Fuzzy Controller Using VHDL*," International Journal of Fuzzy Logic and Ingelligent Systems, Vol. 5, No. 3, pp. 274-279, Sep. 2005.

[6] M. J. Quinn, *Parallel Programming in C with MPI and OPenMP*, McGraw Hil, 2004.

[7] B. Chapman, G. Jost and R. Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*, MIT Press, Oct., 2007.

[8] A. R. Board, OpenMP, 2008. (cited from http://www.openmp.org/)

[9]R. Chandra, Parallel Programming in OpenMP, Academic Press, 2001.

[10] G. Aranguren, et. al., "*Hardware implementation of a pipeline fuzzy controller*," Fuzzy Sets and Systems, Vol, 128, pp. 61-79, 2002.

[11] Y. D. Kim, "*High speed flexible fuzzy hardware for fuzzy information processing*," *IEEE Trans. SMC* - Part A Vol. 27, No.1, pp. 45-56, 1997,

[12] 이수철 외, "구간값 모호집합 사이의 유사도 측정", *한국지식정보기술학회 논문지*, 제4권 제2호, pp.15-22, 2009.

Sang-Gu Lee

Ph.D. degree in the Dept. of Electrical Electronics and Computer Engineering, Waseda University, Japan.

1983.3-Present: Professor, Dept. of Computer Engineering, Hannam University.

Research Interest : Embedded system, Parallel Processing, Image Processing

Jin-Il Kim

2000 : Ph.D. degree in the Dept. of computer engineering, Hannam University.

2006-2009 : Professor, Liberal Arts Education Center, Paichai University.

2009.9-Present: Professor, College of General studies, Sunmoon University.

Research Interest : U-learning, Parallel Processing, Mobile Network