ELSEVIER Microprocessing and Microprogramming 40 (1994) 697–700

Microprocessing and Microprogramming

On the optimal Mapping of Fuzzy Rules on standard Micro–Controllers.

Jos Nijhuis, Herman van Aartsen, Emilija Barakova, Walter Jansen, and Ben Spaanenburg

University of Groningen, Technical Computing Science
P.O. Box 800, 9700 AV Groningen, The Netherlands
(phone: +31 50–637125, email: jos@cs.rug.nl)

– **Abstract** – *Once a fuzzy controller is specified by a rule–set, it can be implemented in dedicated hardware or as a software program. For industrial applications, an inexpensive micro–controller with limited resources is often selected. The implementation (or mapping) issue then leads to a tradeoff between operation speed and memory usage. This paper presents a set of basic transformation rules that allows the designer to optimize such a mapping.*

1. INTRODUCTION

There are several ways to implement fuzzy rule–sets [1]. Special hardware has been developed in order to obtain a high operation speed. Nevertheless most of the currently realized fuzzy logic applications are based on software implementations running on a PC, but most of all on a standard micro–controller. In the latter case, one is confronted with a trade–off between processing speed and storage capacity.

A fuzzy logic controller consists of: a fuzzification inferface, an inference engine, a knowledge base and a defuzzification interface as shown in figure 1.

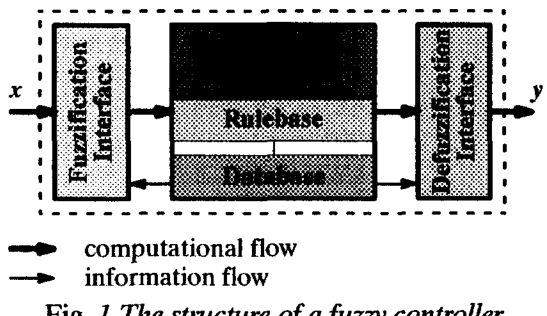

Fig. *1 The structure of a fuzzy controller.*

Consider a very simple fuzzy controller with two inputs, i.e. $x = [x_1,x_2]^T$, one output and two fuzzy rules as follows,

r_1: IF $x_1 = \tilde{A} \wedge x_2 = \tilde{B}$ THEN $y = \tilde{Z}$

r_2: IF $x_1 = \tilde{C} \wedge x_2 = \tilde{D}$ THEN $y = \tilde{X}$.

The required calculations are performed by only five different operators: fuzzification (*fuz*) of the inputs; aggregation (*agr*) of the left-hand side of the fuzzy rules; composition (*comp*) of the right-hand side of the fuzzy rules; combination (*comb*) of the results of the individual rules and defuzzification (*defuz*) of the output values. A straightforward mapping of our simple system would require: 4 *fuz*–operations (one per input term per rule), 2 *agr*–operations (one for each rule), 2 *comp*–operations (one for each output term per rule), 1 *comb*–operation (one for each output) and 1 *defuz*–operation. For an actual mapping the total number of executed operations and the actual realization of the five different operators may be optimized.

Section 2 will define the starting point for the mapping. Next in Section 3 mapping rules for optimizing the number of required operations are defined, whereas in Section 4 mapping rules for increased efficiency of the operators themselves are discussed. Finally some conclusions are drawn from the experience to be reported in the full paper.

2. FUZZY CONTROLLER MAPPING

During the mapping of the fuzzy controller the derived specification is translated into a set of processor instructions (*code*) and data constants (*data*). The memory size of a mapping,

$$mem(<code, data>), \qquad (1)$$

is defined as the total amount of memory in bytes that is needed for storage of the *code*, the *data* and the temporary results. The execution time of a mapping,

$$time(<code, data>), \qquad (2)$$

is defined as the longest (worst–case) period in seconds between the setting of the inputs and the moment newly calculated output values become available. A mapping operation $\mathcal{M}$ for a particular target hardware is given by

$$\mathcal{M}(code, data) \rightarrow < code', data' > \quad (3)$$

The mapping operation changes an initial $< code, data >$–tuple into a new tuple that when executed behaves conform the initial specification. A mapping sequence is a sequence of mapping operations.

$$\mathcal{M}_{seq} = \mathcal{M}_k \circ \mathcal{M}_{k-1} \circ \ldots\ldots \circ \mathcal{M}_1 \quad (4)$$

A mapping sequence is acceptable ($\mathcal{M}_{acc}$) if the resulting *<code,data>*–tuple meets the memory and speed requirements, i.e.,

$$time(< code, data >) \leq t_{req} \quad (5)$$

and

$$mem(< code, data >) \leq m_{req}. \quad (6)$$

The Mapping problem is now defined as finding a sequence of mapping operations that transforms an inital *<code,data>*–tuple into an acceptable one as pictured in figure 2.

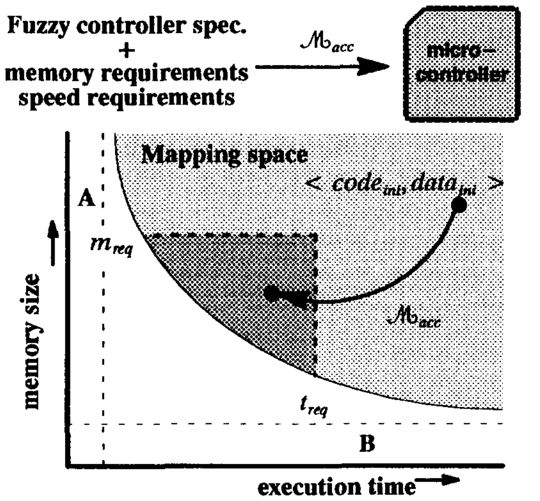

Fig. 2 *Mapping a fuzzy controller onto a micro–processor. Only mappings in the shaded region are possible. Boundary A is determined by maximum processor speed whereas boundary B is determined by the specification of the fuzzy controller itself.*

The *<code,data>*–tuple contains the complete information (in machine code) of the initial specification. Each mapping operation wil transform the *<code,data>*–tuple into a new tuple with different *time* and *mem* values. Depending on our inital position in the mapping space an acceptable sequence of mapping operations may require a decrease in *time* value at the expense of an increased *mem* value. To be able to select a right sequence of operations a basic set of mapping operations have to be defined each having a precise definition of its impact on the *time* and *mem* values.

The remainder of this paper identifies some of these basic mapping operations. A distinction is made between mapping operations that change the number of operations involved by inserting additional control–, branch– and test–instructions and those mapping operations that make a trade–off between the memory usage and evaluation speed of the operators themselves.

3. OPTIMIZATION OPERATIONS

In a fuzzy controller most of the time is used for the evaluation of the rule base, fuzzification and defuzzification. This section will discuss an optimization technique for the evaluation of the rule base by excluding some rules from evaluation. In general when new input is presented a fuzzy controller evaluates the whole rule base using the MIN–MAX or MIN–PROD operation both taking the minimum of the antecedents. If one of the antecedents is zero, both operations evaluate to zero. On the other hand terms of the same set are correlated as far as their values concern. If two terms, say A and B, are disjunct i.e. non–overlapping, and A is activated then B is zero.

Summarizing :

– A production rule evaluates to zero whenever one of the antecedents is zero

– If A and B belong to the same set and are disjunct then $A(x) > 0$ implies $B(x) = 0$.

These two facts give rise to some optimizations, i.e. if two terms A and B of the same set are non–overlapping and $A(x) > 0$, then all rules including the term $B(x)$ can be excluded from computation. So the value of a term determines which rules have to be computed and which rules can be skipped. Take for example the following rule base and fuzzy sets :

```
1: if (x in range(A1)) and (y
in range(B1)) then C1
2: if (x in range(A3)) and (y
in range(B2)) then C2
3: if (x in range(A2)) and (y
in range(B3)) then C3
4: if (x in range(A2)) and (y
in range(B1)) then C4
```

Without optimization all rules are computed. Because A3 is disjunct from A1, x in range(A1) implies x not in range(A3). If x is in the range of A1 only rule 1 has to be computed and rule 2 can be skipped reducing the rules to be computed to three. A solution at hand is to take one of both sets as the main set, for example A, and to order the rules according to this set in ascending order. We can transform the rules above to the following base.

```
1 : if (x in range(A1)) then if
(y in range(B1)) then C1
2 : if (x in range(A2)) then if
(y in range(B3)) then C3
3 : if (x in range(A2)) then if
(y in range(B1)) then C4
4 : if (x in range(A3)) then if
(y in range(B1)) then C2
```

Similarly, we can perform this action on the second set – the B set. The only rules that are affected are 2 and 3, so they are interchanged. This notation gives rise to a direct implementation as follows :

```
        if (x in A1) and (y in
B1) then C1; continue
        if (x in A2) and (y in
B1) then C4; goto label1:
        if (x in A2) and (y in
B3) then C3; goto label2:
label1: if (x in A3) and (y in
B1) then C2; continue
label2: continue
```

In a similar way unneccessary computations of membership functions can be avoided. A membership function is computed only (if it hasn't been already) whenever the whole rule is active (both antecedents are true). The algorithm we used is:

1. Choose one of the sets as the main set. Order the rules in ascending order accordingly. Rules with the same elements of the sets are grouped in this way.
2. If groups are formed choose a second set and perform the same action, till no groups are left.
3. Finally add the jump instructions. Two rules are connected if all constituting antecedents are overlapping.

Using this algorithm the rules are optimized in an easy way. The following section will describe the efficiency and some improvements of the computation of certain membership functions.

4. EFFICIENCY OPERATIONS

In the previous section we described a way to optimize the number of operations that have to be evaluated during each inference of the fuzzy controller. This section will take a closer look at the realization of the operators themselves. One of these operators is the fuzzification of the input values (*fuz*). The most common fuzzy membership functions are stated next, together with their informal definitions :

<u>singleton set</u>

$$fuz(x,\tilde{A}) = \begin{cases} 1 & \text{if } x = a \\ 0 & \textit{else} \end{cases} \tag{7}$$

<u>crisp</u>

$$fuz(x,\tilde{A}) = \begin{cases} 1 & \text{if } a_1 \le x \le a_2 \\ 0 & \textit{else} \end{cases} \tag{8}$$

<u>trapezium</u>

```
fuz(x,A) <- if (x<a_1) then 0 else
 if (x<a2) then (x-a_1)/(a_2-a_1) else
  if (x<a_3) then 1 else
   if (x<a_4) then 1-((x-a_3)/(a_4-a_3))
     else 0
```

Replacement of the division results in :

```
fuz(x,A) <- if (x<a_1) then 0 else
 if (x<a_3) then dummy=(x-a_1)*s_1
   if d>1 then 1 else d
 else dummy=1-(x-a_3)*s_2
 if d<0 then 0 else d
```

<u>bell–shaped</u>

$$fuz(x,\tilde{A}) = \text{f(x) (gaussian funct.)} \tag{9}$$

The fuzzification when using singletons or crisp sets needs only a few comparisons so it can be done very fast. The fuzzification in case of the trapezium and the bell–shaped set implies the use of some ALU–specific operations and is therefore computation–intensive. An alternative is to store the overall membership function as a lookup–table in memory. Access can be done by using only one register load, one addition and one memory load. For example :

compute $fuz(x,\tilde{A})$:

```
load x to r1            —load the value of x.
add  r2,r1,base         —find reference to Ã.
load mem[r2] to r3      —read results
```

Taking, for example, the computation of a trapezium membership function this alternative approach saves about 6 comparisons and two ALU–specific operations. On the other hand, this approach takes much more memory depending on the required accuracy of the membership function. An alternative is to store only those parts of the membership that are difficult to compute as for example the slopes in the trapezium membership function. Storing only these sections results in evaluating the trapezium membership function as follows:

```
if x < a1 then 0 else
if x < a2 then
 mem[x-a1+basepointer_section1] else
if x < a3 then 1 else
if x < a4 then
 mem[x-a3+basepointer_section2] else 0
```

5. DISCUSSION

The main targets for the optimizing fuzzy rule compiler as presented in this paper are both the AMD 29K– and the Hitachi H8–microcontroller family. These families comprise a number of different types each of them with its specific properties. One general mapping would lead to an inefficient use of the various processors. By allowing the designer to tune the mapping of the fuzzy controller, on–chip resources can be fully exploited. On a H8/500 8Mhz microcontroller a 2–input, 1 output, 20–rule fuzzy controller can be evaluated within 0.15 ms for a speed optimized mapping and within 2 ms for a memory optimized mapping indicating the wide range of operational objectives that can be achieved by the proposed method.

ACKNOWLEDGEMENTS

The support of Arcobel b.v. is sincerely appreciated. Furthermore our gratitude goes to R. Tol (RUG), A. deGier (CME) and F. Paternotte (CME) for stimulating discussions.

REFERENCES

[1] J.A.G. Nijhuis and L. Spaanenburg, "ASIC's voor Vage Logica op Weg naar Volwassenheid" (in Dutch), *PolyTechnisch Tijdschrift*, (in Dutch), Vol. 48, No. 6/7, pp. *iA2–iA5*, June/July 1993.

[2] H. Eichfeld, M. Löhner and M. Müller, "Architecture of a CMOS Fuzzy Logic Controller with optimized memory organization and operator design", in: *Proceedings IEEE International Conference on Fuzzy Systems*, San Diego, CA, pp. 1317–1323, 8–12 March, 1992.

[3] H. Watanabe, "RISC Approach to Design of Fuzzy Processor Architecture", in: *Proceedings IEEE International Conference on Fuzzy Systems*, San Diego, CA, pp. 431–441, 8–12 March, 1992.

[4] W.D. Dettloff, K.E. Yount and H. Watanabe, "A Fuzzy Logic Controller with Reconfigurable, Cascadable Architecture", in: *Proceedings 1989 IEEE International Conference on Computer Design: VLSI in Computers & Processors (ICCD'89)*, Cambridge, MA, pp. 474–478, 2–4 October, 1989.

[5] C. Hung and B. Fernández, "Minimizing Rules of Fuzzy Logic System by Using a Systematic Approach", in: *Proceedings Second IEEE International Conference on Fuzzy Systems*, San Francisco, CA, pp. 38–44, 28 March – 1 April, 1993.

[6] D.J. Ostrowski, P.Y.K. Cheung and K. Roubaud, "An Outline of the Intuitive Design of Fuzzy Logic and its Efficient Implementation", in: *Proceedings Second IEEE International Conference on Fuzzy Systems*, San Francisco, CA, pp. 184–189, 28 March – 1 April, 1993.