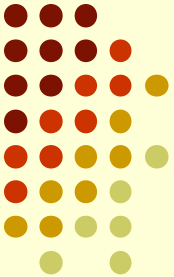


Проектиране и оптимизация на режими на работа на електронни устройства

Владимир Янакиев

Институт по информационни и комуникационни технологии

Българска академия на науките

Публикации:

1.Yanakiev V, M. Lazarova, (2010), Identification and Clusterization of Images with Neuron Networks: Proceedings of the XLV International Scientific Conference on Information, Communication and Energy Systems and Technologies ICEST 2010, Ohrid, Macedonia, Vol.2, ISBN: 978-9989-786-58-7, pp. 587-590.

2.Янакиев.В, Предсказване на микробиологичния състав в геотермални извори, 40 юбилейна конференция МГУ pp 37-41

3.Yanakiev, V., E. Paunova. Static memory optimization by clustering and neural networks in embedded devices. International Conference CompSysTech, 16-17 June 2011, Vienna, Austria, pp 317-322

4.Янакиев, В., Е. Паунова. Обработка на информацията за пътни знаци, предавана по автомобилни мрежи (CAN) с помощта на невронни мрежи. Петнадесети майски четения, Дни на науката 2011, Велико Търново, България, 27 май 2011 г.

5. Yanakiev V. EMBEDED MEMORY AND RUNTIME OPTIMIZATION WITH NEURAL NETWORK AND GENETIC ALGORITHMS, VIII Национална Студентска Научно-техническа Конференция, 23.09.2012 Созопол, pp 289-297

IICT

Положени изпити и резулати от тях:

22.07.2009 - изпит по специалността: 5,25
18.09.2009 - компютърен курс по С: 6,00
30.11.2009 - Интернет технологии за управление: зачита се
1.12.2009 - Проектиране на интернет приложения: зачита се
28.09.2010 - английски език: 5,25

Други активности:
Ръководене на упражнения в ТУ-София
-Програмиране и използване на компютър 2 част: структури данни, списъци, алгоритми на ***C***
-програмни езици:основи на ***C++***

Започване на дисертация: 01.01.2009
Край на дисертацията 31.12.2012

В електрониката се използват множество устройства, свързани посредством Controller Area Network (CAN) , LIN ADC. Автомобилът е типичен пример за множество устройства Electronic Control Units (ECUs). Всяко такова устройство събира информация и изчислява някакъв изход.

Големият брой на тези устройства означава голяма комуникация, поради което локалните изчисления трябва да се извършат максимално бързо.Тези устройства често се произвеждат в голямо количество, затова често цената е съществен фактор. Всяко такова устройство използва процесор с различна флаш и РАМ. Затова и максималната оптимизация на флаш и рам е важна.

Нека изходът на едно устройство да зависи само от входа (или входовете) му или:
Out=F(X),
където X е вектор от входове $X=(x_1,x_2,..x_n)$

Поставяме следните 2 цели пред една подобна задача:
1) да се напише функция на програмен език от високо ниво (в случая C), реализираща F(X) с цел минимален обем на използваните RAM и flash и минимално време за изпълнение. (Оптимизация на прочитането на входните и записването на изходните стойности не се разлежда.). Сравнението да се направи с класически реализации.
Математическо представяне: да се намери множество от функции ($F_1(X)$, $F_2(X)$,…,$F_k(X)$) и множества ($U_1,U_2,...,U_k$), отговарящи на:

$$Out = F(X) = \begin{cases} F_1(X), & \text{за } X \in U_1 \\ F_2(X), & \text{за } X \in U_2 \\ \dots\dots\dots\dots\dots\dots\dots \\ F_k(X), & \text{за } X \in U_k \end{cases}$$

където $U_1 \cup U_2 \cup \dots \cup U_k = U$

(U е множество от всички възможни комбинации на X) $\forall X \in U$

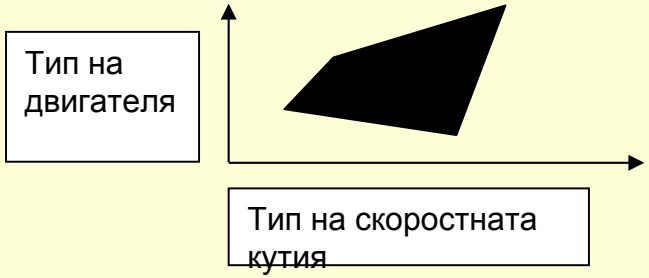
$$U_i \cap U_j \equiv \exists(\text{за } \forall i,j, \quad i \neq j)$$

Като прграмната реализация на F(X) в този вид постига търсените оптимизации

2) при подадени всички възможни входни комбинации и съответни изходни данни автоматично да се генерира кодът на дадена програма с цел постигане на оптимизацията, посочена в 1).

Примерни задачи:

Задача 1: 2 сигнала за вход и един за изход
сигнал 1: тип на двигателя (n на брой стойности)
сигнал 2: тип на скоростната кутия (m на брой стойности)
Изход: тип на резервоара (да приемем, че има 2 възможности 0 и 1)
Както е показано на следващата фигура:
Изхода има стойност 1, там където е обградено в черно, в останалата част 0



Задача 2: един изходен сигнал и един входен
вход: състояние на съединителя: възможни стойности: грешка (стойност 0), отпуснат(1), частично натиснат(2), напълно натиснат(3)
Изход: превключване на скорости: забранено(0), подготовка(1), разрешено(2)
Следната таблица е изходът на базата на входа:

вход	0	1	2	3
изход	0	0	1	2

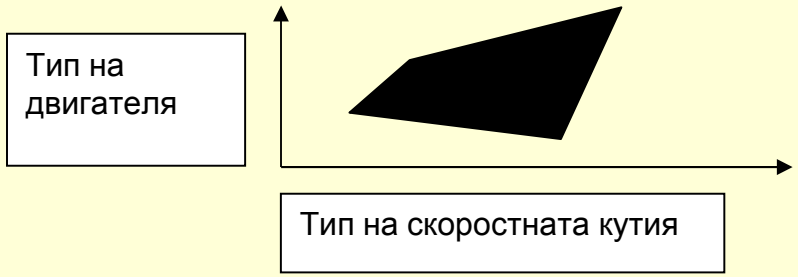
Методи, използвани за оптимизация: групиране на изходни стойности, линейна апроксимация.

Използвани хардуер :
Процесори: V850, PPC

Използва се компилатор Green Hills:
Използван език ***C***
измерванията се разглеждат на две нива: с и без включена оптимизация на компилатора (оптимизация по отношение минимизиране на време за изпълнение и използвана памет от страна компилатора на ниво асемблер).

Групиране на изходните сигнали

Разглеждаме задача тип 1:



Анализ при използване на компилатор с изключена оптимизация

CPU: V850
Compiler: Green Hills
Програмен език: C

В следващата таблица са показани различни измервания на използвана flash и Ram

Instructions in C:	**Използвани PAM и POM**
If((Cond$_1$)&&…&&(Cond$_n$))	2 2 байтови инструкции за всяко условие във флаша
for // cycle	4 2 байтов и 1 4 байтова инстукжия
Инициализация на статичен масив (length n, element size k)	n*k size in RAM and flash (in flash is necessary only for initialization) ининициализацията е преди стартирането на програмата
Condition *(A==B)* or *(A!=B)*	2 instruction of 2 bytes: flash
Condition *(A>B)* or *(A>=B)*	•2-bytes instruction: flash
*//умножение	1 2-bytes and 4-bytes (multiplication) instruction: flash
Basic assignment	1 2-bytes instruction (move): flash
+//събиране	1 2-bytes and 4-bytes (multiplication) instruction: flash

Фигурата може да бъде описана с линии от тип $ax+by+c=0$

$$F_1(x,y)=0,\ F_2(x,y)=0,\ F_3(x,y)=0,\ F_4(x,y)=0$$

При подходящи знаци на тези функции фигурата може да се опише от

$$((F_1(x,y)>0)\ \&\&\ (F_2(x,y)>0)\ \&\&\ (F_3(x,y)>0)\ \&\&\ (F_4(x,y)>0))$$

Или ако се разгледа код реализация

$$if((a_1*x+b_1*y+c_1>0)\&\&(a_2*x+b_2*y+c_2>0)\&\&(a_3*x+b_3*y+c_3>0)\ \&\&(a_4*x+b_4*y+c_4>0))$$

$$Out = 1;$$

$$else$$

$$Out = 0;$$

т.е. за използван флаш имаме за всяка описваща линия:
2 умножения, 2 събирания едно сравнение
И отделно 2 присвоявания или представено като формула:
Flash=2N*multi+2N*Add+N*cond+2* assignments
,където N е броя описващи линии (в случая 4)

В заключение от анализа до тук при използването на групиране на стойности използваната флаш е пропорционална на броя описващи линии

Други възможна реализации:

Вариант 1:

```
Out=0;
if(V0==V) Out=1 ;
else if(V1==V) Out=1;
……
else if(Vm==V) Out=1;
```

Където $V_0=(x_0,y_0),..V_m=(x_m,y_m)$ са комбинациите входове за които изхода е 1, а V=(x,y) е текущия вход.(V_i==V) се реализира като (x_i==x)&&(y_i==y)

Изплзваната флаш:
За всяка кобинация V_i (или обхванатата точка от гледна точка на графиката):
2 условия и едно присвояване
отделно присвояване
или представено математически
Flash=2M*cond+(M+1)* assignments
,където M е броя на обхванатите точки
Анализа показва че спестяването на памет при групиране спрямо вариант 1 е правопропорционално на на броя обхванати точки и обратно пропорционалнало на броя обхващащи линии

Вариант 2 (реализация с допълнителен масив, предварително инициализиран):

```
Out = 0;
for (i=0;i<n;i++)
    if (aV[n]==V) Out = 1;
```

Анализа по отношение на памет сравнен с групиране на изходни стойности е сходен с Вариант 1

Апроксимация чрез линейна интерполация

Апроксимация от една линия

Анализ на задача 2:
Може да се представи апроксимационно с (5*x+1)/8, където x е входът.

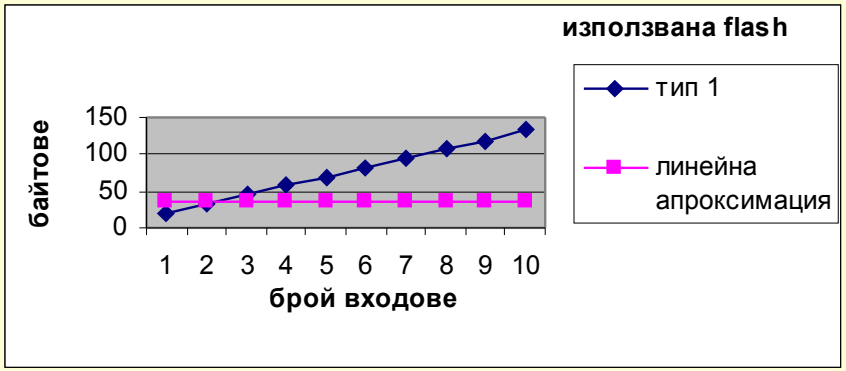
Анализ на използван флаш спрямо реализация 1:

Следните измервания са направени върху процесор PPC с включена оптимизация на компилатора. Разглеждат се случаите, в които f(x) може да се представи от вида (ax+b)/c и се сравнява с имплементация от тип 1 в зависимост от възможните входове на x. Ако се използва линейна интерполация, използваната флаш е 36 байта. В следващата таблица е показано количеството използвана памет в байтове при имплементация тип 1.

имплементация от тип 1

Възможни входове	1	2	3	4	5	6	7	8	9	10
Използван флаш	20	32	46	58	70	82	94	106	118	134

Анализът показва, че има оптимизация по отношение по-малко използвана flash се постига когато когато повече от 3 входа се апроксимират от линейна интерполация.



Анализ на оптимизация от гледна точка минимизиране на времето за изпълнение

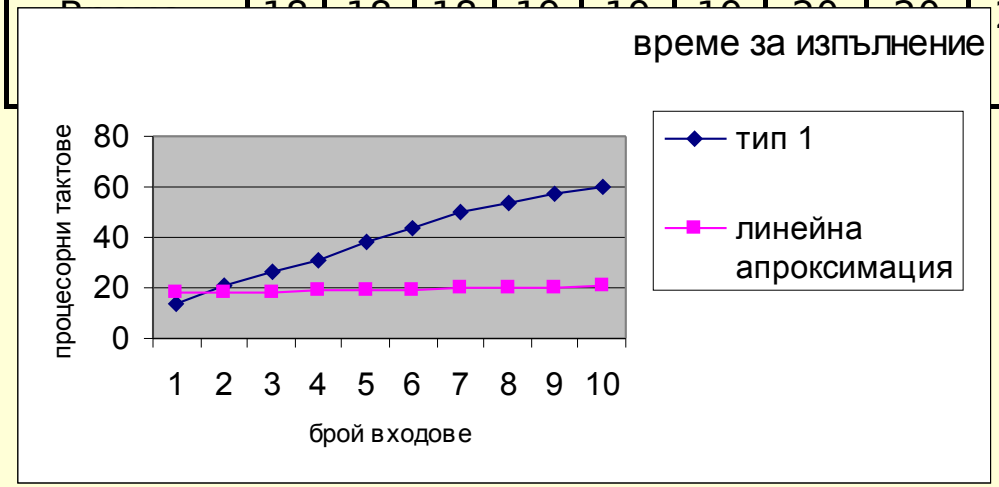
Следните 2 таблици са измерванията на процесорни тактове, необходими за изпълнение в зависимост от възможните стойности на x.

имплементация от тип 1

Брой входни стойности	1	2	3	4	5	6	7	8	9	10
Време за изпълнение	14	21	26	31	38	44	50	54	57	60

имплементация чрез линейна интерполация

Брой входни стойности	1	2	3	4	5	6	7	8	9	10
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	20	21



Анализът показва, че има оптимизация на време, при използването на линейна интерполация, когато 3 или повече входа се апроксимират от линейна интерполация.

Анализ в случай на апроксимация на входно-изходната зависимост от повече от една линии

В общия случай една функция не може да се апроксимира само от една линия. Следните 2 таблици показват използванта флаш и време в случай, че функция, която има 10 възможни входа, бъде разделена на интервали от линейни интерполации.

```
If(x<x1)
    Out=(a1*x+b1)/c1
    else if(x<x2)
Out=(a2*x+b2)/c2
    ……
    else (x<xn)
Out=(an*x+bn)/cn
```

Брой интервали	1	2	3	4	5
Използвана флаш в байтове	26	58	90	122	154

Анализ: при сравнение с таблицата за използвана флаш при имплементация тип 1, се вижда, че се спетява флаш, ако съотношението брой входове към брой интервали в които може да се направи линейна апроксимация е минимум 3:1.

Брой интервали	1	2	3	4	5	6	7	8
Необходимо време в процесорни тактове	18	25	33	40	47	54	62	72

Анализ: при сравнение с таблицата за време за изпълнение при имплементация тип 1, се вижда, че се спетява време, ако съотношението брой входове към брой интервали в които може да се направи линейна апроксимация е минимум 3:1.

Благодаря за Вниманието