Applied Mathematics & Information Sciences
An International Journal

An Optimization Algorithm of Variable Allocation Based on Block Architecture

XU Chao[1,2], *HE Yan-xiang*[1,3], *CHEN Yong*[1,3], *WU Wei*[1,3], *Zeng Xiao-ling*[1,3]

[1]School of computer of Wuhan University, Wuhan 430072P. R. China
[2] Xuzhou College of Industrial Technology, Jiangsu, Xuzhou 221000, China
[3]State Key Laboratory of Software Engineering of Wuhan University, Wuhan 430072P. R. China

Received: 13 Oct. 2012; Revised 2 Nov. 2012; Accepted 16 Nov. 2012
Published online: 1 Mar. 2013

Abstract: In this paper, an optimization algorithm of selection block instruction (heuristic allocation algorithm) based on 8-bit microprocessor of block memory architecture as experiment platform is proposed to solve the problem of variable allocation and reduce the number of selection instructions. First the RAM space allocation is designed in an optimizational way, then an heuristic allocation algorithm of variable is designed, finally the position inserted by selection block instructions is optimized. This method can reduce the number of selection block instructions effectively and produce a good effect in code compression. To verify the correctness and efficiency of the above algorithm, this paper adopts the actual embedded system as test case to perform in the experiment. The result shows that the method can obviously reduce the number of selection block instructions, save memory space and improve the integral performance of system.

Keywords: 8-bit microprocessor,block architecture,variable allocation,optimization of selection block instruction

1. Introduction

When embedded system is developed, memory space and computing power in target platform is limited, the general compilation tool chain requires large memory and strong computing power, so it is impossible to compile in the same computer. Embedded cross-compiler[13] which can compile executable code for target platform in host platform with strong CPU and enough space is introduced to solve the problem. By contrast with universal system, embedded system puts forward more requirements on cost, power consumption and function for quality of application software. Since cost of ic chip is in proportion to area density of that, the density is higher, the capacity is larger and the cost will be higher. If code generated by embedded compiler isn't optimized fully[3], it is likely to upgrade hardware system and push business cost. So code quantity in embedded system[7] can have immediate impact on hardware cost. In the process of embedded system development, it is crucial to optimize code[4][15]. Only when is code size reduced as far as possible, system efficiency can be improved.

8-bit microprocessor[8] is most widely used accounting for 55% in the embedded platform field. Usually the space of memory address supported by 8-bit microprocessor is limited. 8-bit microprocessor usually adopts block architecture and combines corresponding selection block instructions to access the data space beyond single block for expanding the limited memory space[10]. The selection block instructions can be inserted into source code to select the block the next instruction will reach, but the insertion can increase code size and the cost of execution time[2]. Therefore, how to insert selection block instructions as less as possible is meaningful for embedded platform with limited space. This paper takes 8-bit microprocessor with block architecture as platform, expects to reduce the number of awaiting selection block instructions by analyzing relationship between variables in program beginning with variable allocation to realize better code optimization.

To begin with, the paper introduces a prerequisite of knowledge. In Section 2 block architecture of 8-bit microprocessor is introduced. In Section 3 an heuristic allocation algorithm for variable is introduced including the optimum

* Corresponding author: e-mail: xuch@whu.edu.cn

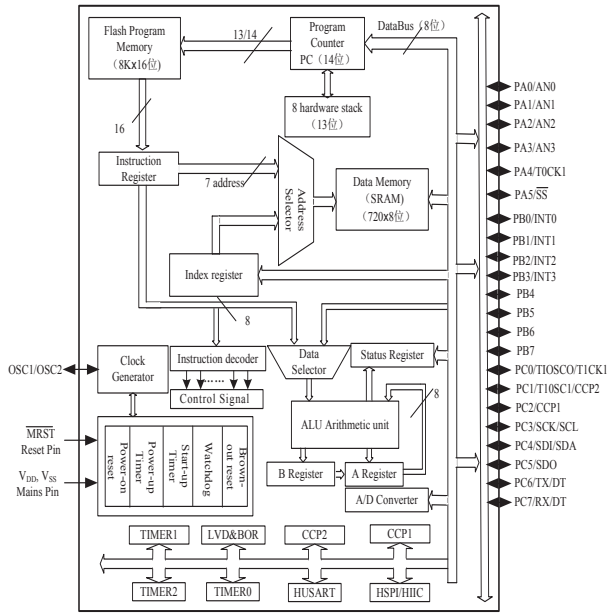

Figure 1 Structure diagram of a chip in HR6Pseries

design of RAM space allocation and insertion position of selection block instructions. In Section 4 results and analysis are demonstrated. In the last part related work and conclusion are presented.

2. An Introduction to 8-bit Microprocessor Platform

In this paper 8-bit microprocessor of HR6P series adopting RISC Harvard structure is embedded with hardware multiplier and encrypted by special users' code, the instruction of which has the characteristics of high confidentiality and efficiency. Buses of program and data access in the structure are mutually independent. Chip instruction set in this series has 48 reduced instructions in it with the characteristics of high coding efficiency and easy extension. Chip integrates many devices including PWM, analog comparing, LCD DRIVERcommunication module, program memory with capacity from 2K16bits to 8K16bits,data memory with capacity from 1288bits to 1K8bits and so on.HR6P series chip is given as an example demonstrating the structure diagram in Figure 1.

3. The Design of Optimization Algorithm of Selection Block Instructions

Block architecture only supports serial data access, which means CPU only can access data space in one block once and which block will be accessed depends on the specific

```
void fun()
{
  Int a,b,c,d,e;
      a=1;        (1)
      b=2;        (2)
      c=a;        (3)
      e=3;        (4)
      d=b;        (5)
      e=d+e;      (6)
      a=3;        (7)
      e=a;        (8)
  }
```

Figure 2 Example Description

selection block instruction. Usually CPU has a special register to store the address of the selection instruction. By inserting the selection instruction into source code the block that the next instruction will reach will be selected. But the insertion of the selection instruction increases code size and pushes the cost of execution time. So how to insert selection instruction as less as possible is meaningful for the embedded platform with limited space.

3.1. The Optimum Design of RAM Space Allocation

Optimization for RAM space allocation is that life period of each local variable is achieved by analyzing control flow and data flow of function and non-life-period-overlap variable can be allocated in same RAM space without influencing the correctness of program. Optimization Algorithm 1 for RAM space allocation is shown as follows:

Algorithm 1

Input:
three address code statements before optimization

Output:
three address code statements after optimization

1: Analyze control flow and data flow and generate ud chain of local variable in function
2: Achieve life period of each variable according to ud chain
3: Calculate life period overlap of each variable to get overlap graph
4: Allocate variable according to overlap graph

To illustrate optimization algorithm for RAM space allocation, a specific function shown in Figure 2 is taken as an example and the concrete implementation procedure is presented.

In Figure 2, (1)to(8)represents life period of variable. Analyze control flow and data flow in function to get ud chain of function and b represents basic block. Division of life period is shown as follows:

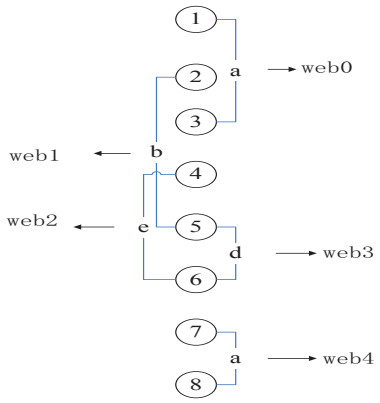

Figure 3 Variable Life Period and Space Allocation without Optimization

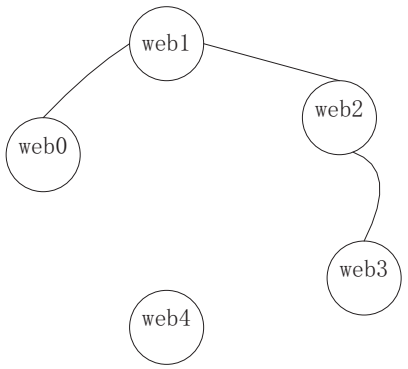

Figure 4 A diagram of Overlap Relationship between Webs in the Example

–There is neither def nor use in b, b is in life period.
–There is no def but use in bif there is a covered block subsequentlyb is in life periodotherwise from the beginning of the block to the last point of use in block is in life period.
–There is def but no use in bfrom the definition point to the finishing of the block is in life period.
–Def is before the first point of useif there is a covered block subsequentlyfrom the definition point to the finishing of the block is in life period otherwise from the definition point to the last point of use in block is in life period.
–Def is between two points of useif there is a covered block subsequentlyb is in life periodotherwise from the beginning of the block to the last point of use in block is in life period.
–Def and use are in the same statementb is in life period.

The corresponding space allocation for variable allocation can be achieved from the analysis. For example, a diagram of variable life period and space allocation without optimization is presented in Figure 3.

In Figure 3 life periods of Variable a,b,d,e are achieved by analyzing control flow and data flow and allocated to their respective space, Variable c isn't analyzed because definition of Variable c isn't used and is optimized before optimization of space allocation. A space is mapped as a Web, according to the sequence of variable's definition, space for variable a is short for Web0, space for variable b is short for Web1 and so on. It is shown in Figure 4 that five Webs are respectively allocated to five different variables. The concrete optimization algorithm for space allocation is as follows:

–Calculate the overlap relationship between webs. If two webs' reachable blocks are not the same, there is no overlap for the two webs. If two webs' reachable blocks are the same, the statements covered by the two webs in the block should be analyzed. If two webs cover the same statement, it illustrates that they overlap, otherwise they don't. The overlap relationship between webs is shown in Figure.3.3 in which line represents the overlap between two webs.
–Arrange all webs in order of space size from large to small and then arrange them in order of use frequency from more to less. In this way it can reduce some subsequent chip selection instructions. Store the arranged webs in chained list which is waiting for allocation. Supposing that the size of Web0 in the above diagram is 3,Web1 is 4,Web2 is 1,Web 3 is 2,Web4 is 2,the order of chained list is: web1,web0,web4,web3 and web2.
–Allocate space for web in chained list in turns. Before allocation, judge if this web overlaps the web in the allocated space. If no overlap, this web can be stored in the space and space size depends on the larger one, otherwise space should be allocated again. After web allocation in above diagram the situation is that space size of Web0,Web2 and Web4 is 3,space size of Web1 and Web3 is 4, Web 4 can be put in either of the two spaces for it has no overlap with the two spaces.

3.2. An Heuristic Allocation Algorithm

3.2.1. Brief Introduction to Heuristic Allocation Algorithm of Variable Class

The set in which all of single variable have become several variables after RAM space allocation optimization is allocated to a mutual space. A set is called a variable class. Block architecture of 8-bit microprocessor requires corresponding selection block to access the data space in different block. It means that when variable class can't be stored in one block after RAM space allocation optimization, how to allocate variable classes to different blocks for reducing the number of selection block instructions should be taken into account.

Two cases are involved: there are two variable classes, if both are in the same block, chip selection instruction

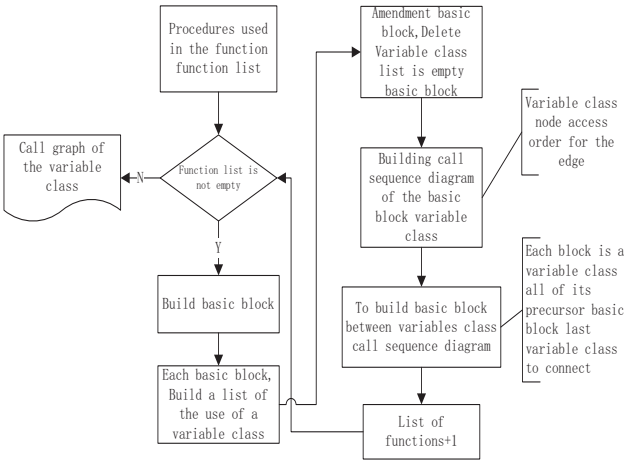

Figure 5 Construction Algorithm of Weighted Digraph

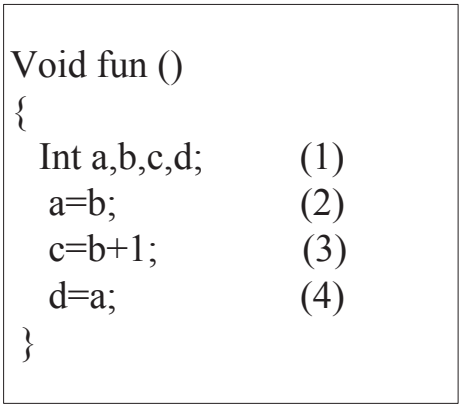

Figure 6 Construction Example of Weighted Digraph

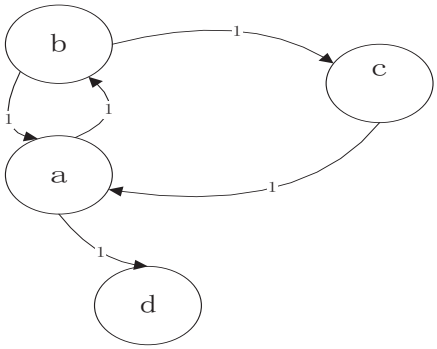

Figure 7 the Corresponding Access Sequence Diagram

isn't needed; if two variable classes are in different blocks, block selection instruction is needed for accessing them. For different blocks the number of selection block instruction is different when switchover between blocks occurs. For example, switchover from Block0 to Block1 needs one selection block instruction, but switchover from Block0 to Block3 needs two selection block instructions. Considering these two cases, this paper proposes heuristic allocation algorithm of variable class which can reduce the number of sequent block selection instructions by allocating the neighboring variable classes to two blocks with less switchover cost.

3.2.2. Description of Heuristic Allocation Algorithm of Variable Class

For solving the above problem about variable class allocation, construct an access relationship diagram in which weighted digraph containing node set and edge set is used to describe the variable class access relationship. Each node represents a variable class and weight of node represents space size occupied by variable class; Each edge represents the access relationship between two variable classes and edge weight represents access sequence between two nodes. In this way the access relationship flow diagram can be expressed as G=¡VE¿. V represents node set or variable class, E represents edge set or access sequence between variable classes. Suppose there are Node a and Node b, directed edges¡a,b¿ represents it accesses Variable b after accessing variable a. Construction flow diagram of weighted digraph is presented in Figure 5.

Now a function can explain the construction progress of the above weighted digraph in Figure 6 and 7.

In Figure 7, it is clear that edge weight of Variable (ab) is 2 and edge weight of Variable(ac) and that of Variable (ad) are both 1.

After weighted digraph of variable class access relationship is constructed, heuristic allocation algorithm of variable class becomes the question inserting the least selection block instruction in the weighted digraph. The number of the inserted selection block instruction is influenced by two factors: (1)if two variable classes are in the same block, chip selection instruction isn't needed. (2)if two variable classes are in different blocks, the neighboring variable classes are allocated to two blocks with less switchover cost. These two factors are the factors of heuristic allocation algorithm. It means that when two variable classes are in the same block, space occupied by variable classes is less than one block, space size occupied is called a heuristic factor; when variable classes are in the different blocks, instruction switchover cost between blocks is another factor, switchover cost is decided by edge weight between blocks and distance between blocks. The concrete description of heuristic allocation algorithm of variable class is shown in Algorithm 2.

Input Variable G as variable class and get an access relationship diagram, where V is variable class, E is access relationship between two variable classes, V and E are parameters of the diagram, input Variable K is the block

Algorithm 2

Input:
Graph G=(V,E),K
Output:
V's K Partition Set P=($P_1, P_2 \ldots\ldots P_k$)

```
list lst = sort edge weight from large to small
while lst is not empty do
    e(u,v) = the first element in lst
    if size(u)+size(v) ≤ m then
        merge u and v into a vertex set
    end if
    eject the first element in lst
end while
while Set(V) > K do
    take out the first two elements in Set(V) to u', v'
    f_min(u', v') ← +∞
    for each u ∈ e, v ∈ Set(V) do
        value(u,v) = weight(e(u,v)) + distance(u,v)
        f(u,v) = α.(size(u)+size(v)) − β.value(u,v)
        if f(u,v) < f_min(u', v') then
            f_min(u', v') ← f(u,v)
            u' ← u
            v' ← v
        end if
    end for
    merge vertex set u', v'
end while
```

number of space division. Edge weight represents relationship between variable classes. e(u,v) represents the edge between point u and point v, size()represents space size occupied by variable, m represents space size of each block, set(V) represents the number of node set. f(u,v) is heuristic function, α, β respectively represents two heuristic factors. Value(u,v) is cost function of block division which is measured by weight(e(u,v)) and distance(u, v). When $f_{\min}(u', v')$, u' and v' merged and put into one block space, the least chip selection instruction can be inserted in the digraph.

3.3. The Optimum Design of Insertion Position of Selection Block Instructions

The above research indicates that variable classes have been allocated to the different blocks in optimizational way, therefore, the corresponding selection block instructions need to be inserted into data of access block and the insertion position of instructions will influence the number of insertion instructions which happens between basic blocks.

In this paper the optimization of insertion position is divided into two parts:(1)select statements are added to the entries of all basic blocks; (2)find out basic blocks' joint with bone pattern branches and put the two merged entry selects at the exit of predecessor block to reduce select statements.

Process of algorithm: input a mapping table varsToBank and an intermediate code representation sequence of a function noBSLCodeList to construct basic block of this function (the first line of code);output intermediate code representation sequence inserted selection block instruction; insert corresponding selection block instructions (from line 2 to line 15 of code) in the basic block and then insert selection instruction between basic blocks (from line 16 to line 30 of code); decomposition blocks generates new intermediate code representation sequence BSLCodeList inserted selection block instruction. Function BtoB used to reduce the number of inserted selection block instructions represents the least inserted instructions for block switchover and BtoB(1,2)represents switchover from Block 1 to Block 2. The optimum design of insertion position of selection block instructions in Algorithm 3.

Algorithm 3

Input:
varsToBank, noBSLCodeList
Output:
BSLCodeList

```
 1: B = construct the basic block of noBSLCodeList
 2: for each b ∈ B do
 3:     currentBank = -1;
 4:     for each intermediate code i in b do
 5:         if i using variable a then
 6:             if currentBank==-1 then
 7:                 b.entryBank = varsToBank[a]
 8:             else if varsToBank[a]!=currentBank then
 9:                 i-> push_front(BtoB(currentBank, varsToBank[a]))
10:             end if
11:             currentBank = varsToBank[a]
12:         end if
13:     end for
14:     b.exitBank = currentBank
15: end for
16: for each b in B do
17:     clear tmpList
18:     for each d in b.preBlocks do
19:         if !tmpList.contain(d.exitBank) then
20:             tmpList.add(d.exitBank)
21:         end if
22:     end for
23:     if tmpList.size==1 then
24:         if tmpList[0] != b.entryBank then
25:             b.push_front(BtoB(tmpList[0], b.entryBank))
26:         end if
27:     else if tmpList.size¿1 then
28:         b.push_front(BtoB(-1, b.entryBank))   //decomposition intermediate code representation sequence BSLCodeList
29:     end if
30: end for
31: BSLCodeList = deconstruct the basic blocks B
32: return BSLCodeList
```

Process of algorithm: input a mapping table varsToBank and an intermediate code representation sequence of a function noBSLCodeList to construct basic block of

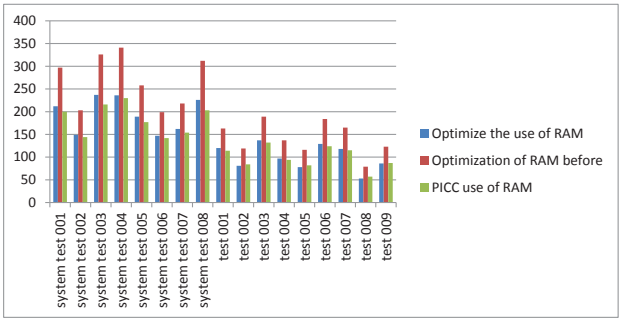

Figure 8 The Corresponding Usage Bar Chart of RAM before and after Optimization

this function (the first line of code);output intermediate code representation sequence inserted selection block instruction; insert corresponding selection block instructions (from line 2 to line 15 of code) in the basic block and then insert selection instruction between basic blocks (from line 16 to line 30 of code); decomposition blocks generates new intermediate code representation sequence BSLCodeList inserted selection block instruction. Function BtoB used to reduce the number of inserted selection block instructions represents the least inserted instructions for block switchover and BtoB(1,2)represents switchover from Block 1 to Block 2.

4. Results and Analysis

The optimum design in this paper is based on embedded cross-compiler, on which the test should rely. Test case set of embedded cross-compiler is experimented with to verify the correctness and effectiveness. Experimental process: input high level C language code to generate ASM file through target compiler, then generate OBJ file through assembler, finally generate HEX file through linker. The optimization effect is measured in terms of three files' correlate comparison.

1The used space size is calculated according to the corresponding information of data segment in the file generated by compiler in order to verify space used by variables after RAM space allocation optimization. Comparisons between the used space before and after optimization and PICC are shown in Table 1 and Figure 8.

In whichoptimization rate =RAM before optimization-RAM after optimization/ RAM used before optimization

Seen from the above chart, effectiveness of the algorithm is verified by saving the space obviously after RAM space allocation optimization.

2. In order to calculate the number of the inserted chip selection instructions, the corresponding HEX codes in chip selection instruction set should be found and searched to get the frequency that the corresponding instructions use, thereby the number of chip selection instructions used in the program is determined to assess the effectiveness of the algorithm. HEX file is a binary code file which can be directly loaded to run in the target machine. A simple HEX file is given as an example shown in Table 2.

Table 1 The Usage Table of RAM before and after Optimization

Test case	RAM after optimiz ation	RAM before optimi zation	Optimiza tion rate of the algorithm	RAM of PICC	Optimiz ation rate of PICC
ST001	212	297	28.62%	200	32.66%
ST002	149	203	26.60%	144	29.06%
ST003	237	326	27.30%	216	33.74%
ST004	236	341	30.79%	230	32.55%
ST005	189	258	26.74%	177	31.40%
ST006	147	199	26.13%	142	28.64%
ST007	162	218	25.69%	154	29.36%
ST008	226	312	27.56%	203	34.94%
T001	120	163	26.38%	114	30.06%
T002	81	119	31.93%	84	29.41%
T003	137	189	27.51%	132	30.16%
T004	97	137	29.20%	94	31.39%
T005	78	116	32.76%	82	29.31%
T006	129	184	29.89%	124	32.61%
T007	118	165	28.48%	115	30.30%
T008	53	79	32.91%	57	27.85%
T009	86	123	30.08%	87	29.27%

Table 2 Some HEX File Example

```
:020000040000FA
:100000000C082B200010826B025824580158055808
:1000100025580158065800588A6D005CA06C206471
:100020000308A16CA1640345086CA0662064030862
:10003000A26CA06620640308A36C2108A1671510B8
:10004000086C2264846CA06620640308806CA2663D
:10005000A3672110086C2464256BA16DA161A66CB7
:0C0060002164A76C8A4900088A4D00103A
:02100000086C7A
:00000001FF
```

In HEX file the first two bytes of each binary code represents the number of the instruction, the next two are flag bits and the last two are check codes. Number calculation of the inserted selection block instruction before and after optimization and comparison with PICC are shown in Table 3 and Figure 9.

In which: optimization rate=(the number of selection block instructions used before optimization - the number of selection block instructions used after optimization) / the number of selection block instructions used before optimization.(The result of the above chart and table shows that after the optimum design of selection block instructions, a reduction in the number of selection block instructions proves the effectiveness of the algorithm. optimization effect of selection block instructions is similar to the

Table 3 Number Calculation Table of the Inserted Selection Block Instruction before and after Optimization

Test case	after optimization instructions	before optimization instructions	Optimization rate of the algorithm	Number of instructions of PICC	Optimization rate of PICC
ST001	693	950	27.05%	665	30.00%
ST002	192	272	29.41%	201	26.10%
ST003	411	605	32.07%	432	28.60%
ST004	1255	1792	29.97%	1198	33.15%
ST005	879	1210	27.36%	834	31.07%
ST006	1054	1420	25.77%	958	32.54%
ST007	1120	1590	29.56%	1019	35.91%
ST008	332	478	30.54%	348	27.20%
T001	73	106	31.13%	75	29.25%
T002	64	95	32.63%	66	30.53%
T003	108	145	25.52%	99	31.72%
T004	146	211	30.81%	154	27.01%
T005	175	243	27.98%	164	32.51%
T006	26	38	31.58%	27	28.95%
T007	182	248	26.61%	170	31.45%
T008	122	162	24.69%	109	32.72%
T009	98	139	29.50%	101	27.34%

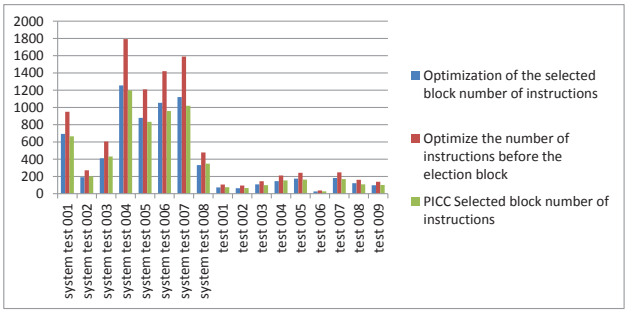

Figure 9 The Corresponding Usage Bar Chart of the Selection Block Instruction before and after Optimization

developed PICC but has better optimization effect for some small test cases.)

3.In order to test the influence over the whole system after selection block instruction optimization,size of HEX file (or size of ROM space) generated after each test case optimization is calculated and compared with the developed industrial compiler PICC for assessing effectiveness of optimum design. Comparison of ROM space is shown in Table 4.

(In which: relative optimization rate = ROM after the algorithm optimization/ ROM of PICC. The result in above table shows that space size is almost kept before 0.9-1.1 times of PICC after optimum design, which means its optimization effect is almost the same as that of PICC. Optimization effect of the algorithm is better for the small test cases.)

Table 4 Number Calculation Table of the Inserted Selection Block Instruction before and after Optimization

Test case	after optimization instructions	before optimization instructions	Optimization rate of the algorithm
ST001	4560	4785	1.05
ST002	3620	3347	0.92
ST003	4344	4106	0.95
ST004	6072	6412	1.06
ST005	5036	5133	1.02
ST006	5177	5244	1.01
ST007	5285	5467	1.03
ST008	3891	3695	0.95
T001	1377	1183	0.86
T002	1293	1369	1.06
T003	1861	1955	1.05
T004	2014	1859	0.92
T005	2345	2580	1.10
T006	1029	935	0.91
T007	2545	2647	1.04
T008	2221	2351	1.06
T009	1980	1726	0.87

Seen from the comparison among RAM, the number of selection block instructions and ROM space, the optimization algorithm saves the space effectively, reduces the number of selection block instructions obviously, has almost the same optimization effect comparing with the developed industrial compiler PICC but better optimization effect for the small test cases. Since PICC pays more attention to global optimization and adopts uniform allocation, each block has been allocated almost the same number of variables. In this way for the situation with less variables, uniform allocation increases the code cost without good effect of code optimization.

5. Related Work and Conclusion

The optimization algorithm study about variable allocation[1][14][9] has two aspects: concurrent access data and serial access data. In the field of concurrent access data[6], the researchers considered that single instruction can increase the number of block space, and thereby increase memory bandwidth to improve the concurrent execution efficiency of program. In the field of serial access data, the researchers at home and abroad also did some research. In the literature[12][11], Scholz et al in Sydney University proposed an optimization technique that can minimize the cost of block switchover by optimizing the position of insertion block instruction. In the literature Bradlee et al proposed a dynamic programming algorithm on the base of block memory architecture and global shared memory.This method can reduce chip selection instruction by allocating

some frequently switching variables to shared memory. In the literature Minming Li et al in City University of H.K proposed a rounding two approximation algorithm by analyzing CFG[5]. The above methods all suppose that variables have been allocated to each block in advance before optimization of chip selection instruction. But if variables can be allocated automatically to put variables with close relationship to the same block through program in terms of relationship between variables before variables allocation, probably better allocation effect can be achieved to reduce chip selection instruction and programming load of programmer. So this paper studies how to reduce the inserted selection block instructions in terms of variables allocation and combines optimum design of instruction insertion position to achieve better optimization effect.

This paper introduces the block architecture of 8-bit microprocessor, the architecture of HR6P series chip used as experiment platform and instruction system. Next, heuristic allocation algorithm is proposed in the paper when the question of variable allocation is mentioned. At the same time, the detailed algorithms about optimum design of RAM space allocation and selection block instruction are presented. Finally correctness and effectiveness are verified through experiment and the experimental result shows that the design saves the space effectively, reduces the number of selection block instruction obviously and achieves effect of space optimization. In the future work, we'll try to find better allocation algorithm of variable class and seek the best way to allocate variable class so that the number of selection block instruction will be reduced further. Meanwhile, test case set will be improved to increase the correctness and comprehensiveness of test.

Acknowledgement

The paper is supported by the State Key Program of National Natural Science Foundation of China (Grant No. 91118003), National Natural Science Foundation of China (Grant No. 61170022), Jiangsu Qing Lan Project and Jiangsu Overseas Research Training Program for University Prominent Young Middle-aged Teachers and Presidents.

References

[1] J. Davidson and C. Fraser. Register allocation and exhaustive peephole optimization. *Software: Practice and Experience*, **14**(9):857–865, 1984.

[2] C. EMBEDDED. Hardware-software co-design of embedded systems.

[3] W. Y. L. Fang L He, G P. A new non-interior-point continuation method for nonlinear complementarity prohlem with po-function. *Acta Mathematica Scientia*, **31A**(1):229–238, 2011.

[4] C. Han, Y. Wang, and G. He. On the convergence of asynchronous parallel algorithm for large-scale linearly constrained minimization problem. *Applied Mathematics and Computation*, **211**(2):434–441, 2009.

[5] M. Li, C. Xue, T. Liu, and Y. Zhao. Analysis and approximation for bank selection instruction minimization on partitioned memory architecture. In *ACM Sigplan Notices*, volume **45**, pages 1–8. ACM, 2010.

[6] L. Liu, L. Xu, and D. Yang. A decentralized resource allocation approach for response-time guarantees in storage system.

[7] Y. Mengting, W. Guoqing, and Y. Chao. Optimizing bank selection instructions by using shared memory. In *Embedded Software and Systems, 2008. ICESS'08. International Conference on*, pages 447–450. IEEE, 2008.

[8] S. Miller and R. John. An interval type-2 fuzzy multiple echelon supply chain model. *Knowledge-Based Systems*, **23** (4):363–368, 2010.

[9] P. Panda, N. Dutt, and A. Nicolau. On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, **5**(3):682–704, 2000.

[10] R. Qin, Y. Liu, and Z. Liu. Methods of critical value reduction for type-2 fuzzy variables and their applications. *Journal of Computational and Applied Mathematics*, **235** (5):1454–1481, 2011.

[11] B. Scholz, B. Burgstaller, and J. Xue. Minimizing bank selection instructions for partitioned memory architecture. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, pages 201–211. ACM, 2006.

[12] B. Scholz, B. Burgstaller, and J. Xue. Minimal placement of bank selection instructions for partitioned memory architectures. *ACM Transactions on Embedded Computing Systems (TECS)*, **7**(2):12, 2008.

[13] Z. Wang and X. Hu. Energy-aware variable partitioning and instruction scheduling for multibank memory architectures. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, **10**(2):369–388, 2005.

[14] C. Xue, T. Liu, Z. Shao, J. Hu, Z. Jia, W. Jia, and E. Sha. Address assignment sensitive variable partitioning and scheduling for dsps with multiple memory banks. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 1453–1456. IEEE, 2008.

[15] X. Zhuang, S. Pande, and J. Greenland. A framework for parallelizing load/stores on embedded processors. In *Parallel Architectures and Compilation Techniques, 2002. Proceedings. 2002 International Conference on*, pages 68–79. IEEE, 2002.

Xu Chao PhD candidate, member of China Computer Federation, the reserch interests include trusted software and embedded system.

He Yanxiang holds PhD in Computer Science from Wuhan University of Hubei, China. He is Professor of Wuhan University, phD supervisor, senior member of China Computer Federation, his research interests include trusted software, distributed parallel processing, and software engineering.