Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 13, no. 7, November 2017, pp. 1087-1093
DOI: 10.23940/ijpe.17.07.p11.10871093

A New Improved Algorithm for SLP

Zhan-Jie Guo[a], Hui Liu[b,c,*]

[a]*Zhengzhou Technical College, Zhengzhou,450000, China*
[b]*PLA Information Engineering University,Zhengzhou,450000, China*
[c]*Henan Normal University, Xinxiang 453007, China*

Abstract

superword level parallel (SLP) algorithm cannot effectively handle the large-scale applications which covered few parallel codes, and the codes which can be vectorized may be adverse to the vectorization. A new improved algorithm for SLP is proposed. First of all, attempt to transform the non-isomorphic statements, which can't be vectorized to isomorphic statements as far as possible. Namely, locate the opportunities of vectorization which SLP has lost, and then build the Max Common Subgraph (MCS) through adding redundant nodes, process some optimization such as redundant deleting to get the supplement diagram of SLP, it can greatly increase the parallelism of program. At last, using the method of cutting, eliminate the codes harmful to the vectorization, and execute them in serial. This vectorizes the revenue codes, improving the efficiency of programs as far as possible. Experimental results show that, compared with the SLP algorithm, its performance in average is better than it 9.1%.

Keywords: isomorphic; cutting; vectorization; superword parallel; the supplement diagram of SLP

(Submitted on July 25, 2017; Revised on August 30, 2017; Accepted on September 15, 2017)

(This paper was presented at the Third International Symposium on System and Software Reliability.)

1. Introduction

Owing to the rapid development of high performance computation, energy consumption, memory access and communication technology is faced with the development bottleneck, and the development of automatic parallel software is the key to solving this problem at present. Since Intel integrated MMX (Multi-Media Extensions) in Pentium processor in 1996, most processors have started to integrate SIMD extension unit [2], so as to fully excavate the computation and graphics capability of processors. SIMD (single instruction multiple data) not only improves the performance of specific application programs, but also increases the efficiency of most programs. Larsen et al. proposed the concept of superword level parallelism (SLP) [4] in 2000. Its basic process is to go upward until load instruction or instruction, not supporting vectorization along the data dependency graph, from store instruction and packscalar instructions turns into vector instructions via greedy algorithm. For large-scale applications, this process has two problems. (1) The premise of conducting vectorization via SLP is statement isomorphism; it has to skip statements with different structures, without processing them. But in large-scale application programs, the proportion of parallel codes, i.e. isomorphic statements, is quite low among numerousprogram codes. (2) Among the codes supporting vectorization in the program, some codes harmful to vectorization might exist. In other words, there might be a need to move these codes from scalar register into vector register, which will increase the cost. In order to solve the above two problems, an improved SLP algorithm is proposed in this paper. The experiment demonstrates that this method can reduce the execution time by 9.1% on average in a group of widely used kernel test sets when compared with SLP.

* Corresponding author.
E-mail address:guojie0616@163.com.

2. Linear SLPand Relevant Researches

2.1 Linear SLP

SIMD extension is one of the major parallel processing technologies at present. The previous multi-media extensions can support data types with comparatively short data length only, while the latest extension can support 128 bits. AVX of intel can support 256 bits, and AVX2 can support 512-bit superword operation. In order to meet the actual demand, Larsen et al. proposed the superword level parallelism (SLP) vectorization method [4] facing basic blocks for the first time in 2000 to process data parallelism types of relatively wide data types. The process of SLP algorithm is as follows. Firstly, adjacent memory access is set as the packing seed; then the packet is extended heuristically through definition–use chain as well as use – definition chain. Finally, the packet is dispatched via the dependency relationship.

2.2 Relevant Researches

In most cases, SLP vectorization needs to apply packing and unpacking operation. Previous researches show that the cost of superword packing / unpacking is so high that it might exceed the performance optimization brought about by SLP. Therefore, it is very important to reduce the cost of packing / unpacking when applying SLP. The packing conflict graph will offer more opportunities of packet reuse, and decide the packet execution sequence and sequence of statement in the packet during dispatching to reduce unpacking times [13]. Multi-element decision graph can be used to effectively denote data in the packet, and reduce unpacking times [3]. SLP algorithm can process isomorphic statements only, so sufficient isomorphic statements in the code is very important for improving the code efficiency. Jinlong XU et al. [15] put forward a segment constrained SLP excavation path optimization algorithm, which can restrain excessive excavation paths caused by loop unrolling via SLP excavation between segments. Shuai WEI et al. [14] proposed a nested loop vectorization method facing SLP. Weiyi SUO et al. [12] added and improved instruction analysis and redundancy optimization algorithms in SLP automatic vectorization, and solved the problem of low vectorization efficiency caused by dependency relationship or data non-alignment. In this paper, improvement is mainly conducted by aiming at the problems such as low occupancy of parallel codes in large-scale applications and possible existence of codes harmful to vectorization in vectorization codes. Firstly, to increase the parallelism of codes, non-isomorphic statements should be made isomorphic as far as possible. Then the program return must be guaranteed as far as possible via throttling method.

3. Overview of the Algorithm

Overview of the algorithm: Firstly, general preprocessing is conducted for the loops needing vectorization, such as loop unrolling. Redundant nodes are added to non-isomorphic statements with similar structures to give them the same structure [9]. Then, return analysis is made on each group of instruction packet supporting vectorization iteratively via throttling method [8]. For each group of packet, if the return is positive after vectorization, then this packet is beneficial to vectorization; otherwise, it is harmful. In order to guarantee the overall code efficiency, vectorization should be stopped for those codes harmful to vectorization, and scalar computation can be conducted. SLP algorithm processes isomorphic instructions only, so statement isomorphism is the premise. But the proportion of isomorphic statements is limited in large-scale application programs. Therefore, in order to improve the program parallelism as far as possible, to increase the possibility of vectorization, and to enhance the program efficiency, non-isomorphic statements should be made isomorphic first [6]. The processing steps are as follows:

- Position the vectorizationopportunity that might be missed by SLP algorithm
- Establish the supplementary dependency graph of SLP by adding redundant nodes as few as possible

3.1 Position the vectorization opportunity that might be missed by SLP

As everyone knows, SLP algorithm processes isomorphic statements. But the codes have experienced many optimization steps before SLP processing, and optimization steps like redundant node removal might transform the dependency graphs that could have become isomorphic into isomeric dependency graphs. Hence, they will be eliminated by SLP, and these vectorization opportunities will be missed. Therefore, statement dependency graph should be established first, and then superword statement groups can be built.

3.2 Establish the supplementary dependency graph of SLP

After the vectorization opportunity missed by SLP is positioned, non-isomorphic statements should be made isomorphic by adding redundant nodes as few as possible into the dependency graph to establish the supplementary dependency graph of

SLP. Generally speaking, the following steps will be involved. 1. The maximum common sub-graph should be built, and in literature [11], fast backtracking algorithm approximate to the optimal algorithm is used to solve this problem. Then the minimum common super-graph is obtained by comparing the differences between the maximum common sub-graph and original graph. For instance, if g1 and g2 are original graphs, and MCS is the maximum common sub-graph of g1 and g2, then the minimum common super-graph*MinS =MCS + dif1 + dif2*, where *dif1 = g1 – MCS* and *dif2=g2 – MCS*. In this way, the supplementary dependency graph of SLP can be gained through steps like insertion of node selection and redundancy deletion. The realization process is shown in Figure 1:

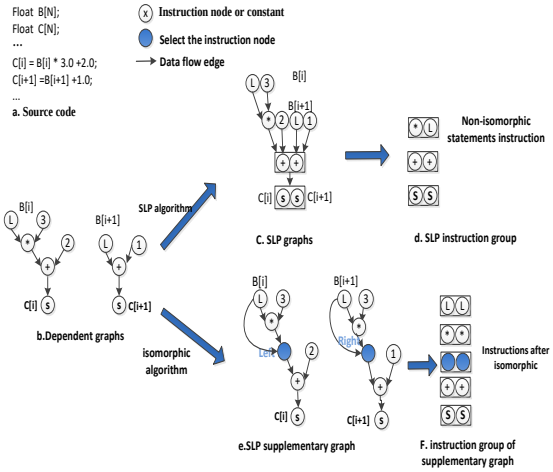

Figure 1. Making non-isomorphic statements isomorphic

For the source code in Figure 1(a), the data dependency graph is established first, as shown in Figure 1(b). If SLP algorithm is adopted, Figure 1(c) is the SLP dependency graph. According to the data dependency graph, 2 groups of instructions supporting vectorization processing can be gained. SLP algorithm searches instruction groups of the same type bottom up starting from store instruction and stopping after meeting load instruction or instruction not supporting vectorization. For instance, the two instruction types in Figure 1(d) are different. Two SLP instruction groups are ultimately gained. But if isomorphicalgorithm is applied, SLP supplementary graph of Figure 1(e) can be obtained. This algorithm realizes isomorphic processing for non-isomorphic statements by adding redundant nodes as few as possible into the dependency graph. Finally, the isomeric dependency graph of Figure 1(b) is supplemented and changed into isomorphic data dependency graph of Figure 1(e). Thus the supplementary dependency graph can get 5 SLP instruction groups, as shown in Figure 1(f). Therefore, isomorphic processing for non-isomorphic statements before SLP processing can greatly improve program parallelism and enhance the program efficiency.

4. Throttling Optimization

When SLP algorithm is used to conduct vectorization for codes, the return is calculated by treating the codes as a whole. It cannot calculate the return of every operation group in each statement respectively, so the parts harmful to vectorization contained in the codes might reduce the vectorization return. In this paper, throttling optimization is adopted. In other words, the vectorization return of each packed instruction set is calculated separately. If the return of vectorization for this instruction set is positive, then vectorization processing will be conducted; otherwise, scalar processing will be adopted.

4.1 Throttling process

The throttling optimization process is as follows. It is the same as SLP algorithm in the previous part, and different treatments are conducted when the instruction cost is calculated later.

- Firstly, conduct vectorization for seed instructions according to the idea of SLP algorithm. Generally speaking, they include: independent memory instruction, to access adjacentmemory address; reduction instruction; independent simple instruction.
- Establish the initial vectorization instruction group according to the data dependency graph by starting from the seed instruction. Search bottom up starting from Store instruction along the data dependency graph until load instruction or instruction does not support vectorization.

- After the data dependency graph is completed, SLP will conduct static evaluation for the code performance by applying the cost model of a specific platform to evaluate the vector cost Vcost and scalar costScost of each instruction. The cost difference is expressed with DfCost, as shown in formula (1).

$$DfCost = V\cos t - S\cos t \tag{1}$$

If the result is negative, it means that vectorization for this instruction will bring about return to this group; otherwise, it is harmful. In order to gain an accurate cost, the algorithm will count extra instructions (collection and diffusion instructions) needed by data conversion between scalar and vector, and is express as SGCost (generally speaking, if one collection or diffusion instruction is used, 1 should be added to the value). Then, the total cost TlCost is worked out that includes the cost of all groups and extra cost, as shown in formula 2:

$$TlCost = \sum_{n=1}^{n} DfCost(n) + SGCost \tag{2}$$

If the value of TlCost is smaller than 0, vectorization can produce returns; otherwise, no return will be generated. If vectorization can be conducted, the equivalent vector code will be used to replace the scalar code; otherwise, the code should maintain the scalar state. Figure 2 presents the realization process of a new improved algorithm of SLP.

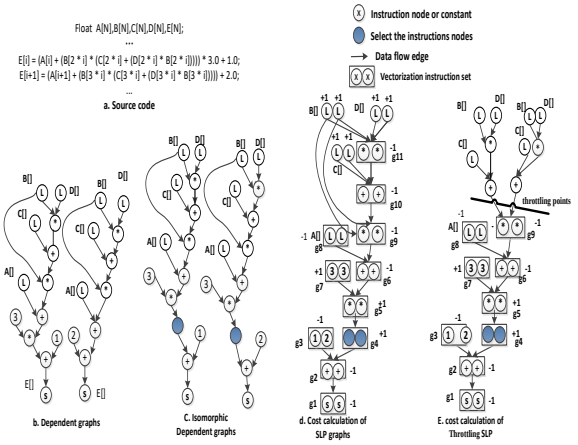

Figure 2. Realization process of the new improved algorithm of SLP

In Figure 2(b), the data dependency graph in Figure 2(a) is displayed. Array E[] will store the calculation results of array A[], B[], C[], and D[]. E[i] and E[i+1] are continuous memory addresses, which are the same load as in A[i] and A[i+1]. Loads from other arrays are discontinuous (this can be known from index 2*i and 3*i of the arrays). The separate circular load in the figure is a discontinuous load, and most vectorization methods cannot realize vectorization processing for these discontinuous loads. Now SLP processing is conducted for these codes. Firstly, it can be seen that the two data dependency graphs in Figure 3(b) are not isomorphic, so isomorphic processing should be conducted for the two graphs in Figure 2(b) via isomorphic method for isomeric statements [9], as shown in Figure 2(c). Hence, SLP algorithm can be used to conduct vectorization processing. Firstly, the seed instruction should be positioned, i.e. adjacent memory access instruction. Stores in E[i] and E[i+1] in the graph form the packet group g1 (i.e. the root of SLP graph in Figure 2(d)). Then the algorithm attempts to constitute more instruction groups with instructions of the same type from bottom to up according to the dependency graph. Other multiply instruction, add instruction, and instructions from A[] also constitute different instruction groups (from g2 to g11) respectively, while load (loads of array B[], C[], and D[]) from discontinuousmemory address access still maintain scalar execution. If the data of every scalar node is used by the vector group, an extra insertion instruction is needed to insert the scalar data into thevector register; thus, the cost is high. As shown in the graph, the vectorization cost of every scalar load will be added by 1. Hence, as presented in the graph, the cost difference of the node on every circular load is marked with +1. Once the data are inserted into the vector register, they can be read out from the vector register at any time when needed, and reused. For example, g9 and g11 in Figure 2(d) can reuse the data of arrayB[].

The vectorization processing performance of the algorithm is evaluated by the cost model of compiler, and the cost

model can assess the cost of every instruction. In the target compiler GCC cost model, the cost of every instruction is displayed as 1, and all nodes in the dependency graph in Figure 2(d) are marked with cost difference. The different cost DfCost of group g1 is -1, indicating that if vectorization is conducted for two stores instructions, the cost is 1. But if scalar processing is still conducted, the cost is 2. The cost difference of instructions maintaining scalar processing is 0, and a positive cost difference exists in any extra instructions supporting vectorization processing, such as insertion instruction, and selection instruction in isomorphic processing. As for the reason, if the cost of scalar processing is maintained, they won't be used. Then the total cost TlCost is calculated. The total cost of this graph is 1, meaning that vectorization processing for this section of code produces no return (vectorization can generate returns only when the total cost is negative).

The reason why vectorization processing of this graph has no return is that sub-graph setting g10 as the root is harmful to vectorization (its total cost is +4). SLP algorithm treats the codes as a whole, so it cannot identify these problems. In order to improve this situation, throttling method is used here to separate those code sub-ranges where the damage is greater than return in vectorization processing, and the cost is calculated separately. As shown in Figure 2(e), segmentation is conducted in group g10. Scalar processing is conducted for sub-graphs setting g10 as the root, while vector processing is carried out for the follow-up parts. At this time, the total cost TlCost is -3, showing that vectorization can produce returns. Performance test is conducted for SLP and SLP algorithm after improvement, and the result show that the improved SLP algorithm is 9.1% faster than SLP on average.

4.2 Calculation of effective throttling points

In order to gain the optimal throttling sub-graph, effective throttling points should first be worked out. The total cost of sub-graph setting every node as the root is calculated, and the calculation results are presented in Table 1.

Table 1. Sub-graph cost of throttling point

RootNode	**g1**	**g2**	**g5**	**g6**	**g9**	**g10**	**g11**
SubTlCost	1	2	3	1	3	4	3
TlCost	0	-1	-2	0	-2	-3	-2

In Table 1, RootNode means the node, SubTlCost represents the total vectorization cost of sub-graph setting this node as the root, and TlCost indicates the total cost of conducting scalar processing for the sub-graph and carrying out vectorization processing for the remaining graphs. The cost is calculated according to the different costs shown in Figure 2(d), and the data in Table 1 are obtained. According to Table 1, g4 and g10 are the optimal throttling points. Owing to the subsequent processing, vectorization should be conducted for codes as many as possible. Therefore, under the same cost, g10 is set as the throttling point. Scalar processing is conducted for g10 in vectorization processing, while vectorization processing is carried out for the remaining parts.

5. Experimental evaluation

5.1 Experimental environment

The dominant frequency of CPU of the experimental platform is 2.10GHz and the memory size is 2GB; it is equipped with 6 processor kernels. The kernel of operating system is Linux 2.6.18, and the version is Redhat Enterprise AS 5.0, which can support 256-bit vector computation.

5.2 Experimental result and analysis

Experiment is conducted on GCC5.1.0 compiler, and the existing SLP algorithm is extended via our algorithm. Besides, experimental evaluation is carried out for our algorithm via SPEC2006 (Standard Performance Evaluation Corporation2006) [10] as well as some standards of NPB (NAS parallel benchmark suite) [5], MediaBench II [1] and Polybench [7]. Table 2 gives detailed information of data used.

In GCC, -O3 is the default option to initiate vectorization, so -O3 is selected as the base option in the test. Based on this, other optimization options are added, such as loop unrolling, loop distribution, etc. In the test, -O3 and -funroll-loops (loop unrolling) are treated as the fundamental compiler options. In this way, three groups of data, including base option, base option + SLP optimization, base option + optimization of this algorithm, are compared. In the experiment, the group

with base option only is called base optimization and is expressed as Base in the graph; the group with SLP optimization is called SLP optimization and is expressed as SLP in the graph. The group with optimization of our algorithm is called new SLP optimization, known as NSLPO (New SLP Optimization) for short. According to the experimental results, the average performance of our algorithm is 17.6% better than that of base optimization and 9.1% better than that of SLP optimization. The results are presented in Figure3.

Table 2. Data information used in the experiment

Example source	**Detailed description**	**Core**
BT	Xi-direction fluxes	compute rhs
433.milc	Su3 matrix by vector mult	mult su3 mat vec sum 4dir
435.gromacs	Kernel from CPU2006	ewaldLRcorrection
453.povray	Triangle bounding box	compute triangle bbox
470.lbm	Kernel from CPU2006	lbmhandleInOutFlow
433.milc	Compute the adjoint of an SU3 matrix	su3-adjoint
cjpeg	Discreet Cosine Transform	jdct-ifast
Polybench	Find shortest paths in a weighted graph	floyd-warshall

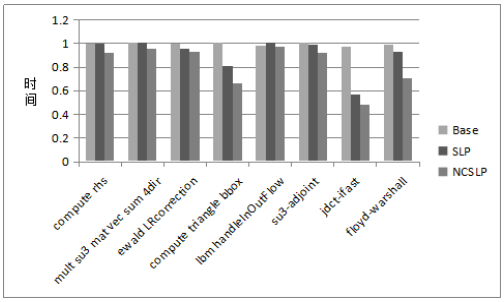

Figure 3. Performance comparison among base optimization, SLP optimization, and NSLPO

Figure3 shows the performance comparison among base option, SLP optimization, and new SLP optimization in GCC compiler on X86 Platform. According to the figure, for most programs, new SLP optimization is superior to SLP optimization and base optimization in performance. The highest performance of new SLP optimization is 49% better than that of base optimization (as shown in jdct-ifast), and its speed is 15% higher than that of SLP optimization (compute triangle bbox). The first two programs (computerhs and multsu3matvecsum4dir) have returns under the new improved algorithm of SLP, but they produce no return under SLP algorithm. SLP dependency graph contains codes harmful to vectorization, so the conclusion that vectorization cost of these codes is greater than the scalar execution cost can be gained during cost model evaluation. The improved algorithm of SLP can remove these codes harmful to vectorization from the whole, conduct vectorization for codes that will produce returns after vectorization, and maintain scalar execution for harmful codes. In this way, the proportion of vectorization codes can be greatly increased, and the program execution efficiency will be improved. As for the follow-up two programs (ewald-LRcorrectionandcomputetriangle-bbox), either SLP optimization or improved algorithmoptimization of SLP is superior to baseoptimization in performance. In these two programs, the vectorization cost evaluated by GCC cost model is smaller than the cost of scalar execution, so program performance is improved. According to Figure3, improved algorithm of SLP has better performance than SLP algorithm.

```
Tp1 = qtv[0]*16384
Tp2 = qtv[1]*22725
Tp3 = qtv[2]*21407
Tp4 = qtv[3]*19266
```

Advanced optimization

```
Tp1 = qtv[0] << 14
Tp2 = qtv[1] * 22725
Tp3 = qtv[2] * 21407
Tp4 = qtv[3] * 19266
```

Figure 4. Non-isomorphism triggered by advanced optimization (jdct-ifast)

The core jdct-ifast includes two different types of codes, both of which can produce returns in NSLPO. The source code of jdct-ifast contains homogeneous computing, which turns into non-isomorphic computing after passing advanced optimization. As shown in Figure4, its advantage is that reduction is realized. The source code of jdct-ifast realizes multiplication for a series of array constants. Some values in the array are square numbers. Multiply instruction reduction of these specific values is mainly attributed to logical left shift, and the isomorphism is broken. Therefore, SLP algorithm cannot realize vectorization processing for them. But our improved algorithm of SLP contains the module of making isomeric statements isomorphic, so it can realize vectorization and improve the program processing efficiency.

6. Conclusions

In this paper, an improved algorithm of SLP algorithm is proposed. Firstly, this algorithm increases the proportion of parallel codes by conducting isomorphic processing for non-isomorphic statements. Then, throttling method is used to process instructions in the isomorphic graph to eliminate those codes that support vectorization but will hinder the improvement of program performance. This algorithm generates various sub-graph sets from the original SLP graph, and finds out sub-graph with the best vectorization performance through cost model. Finally, vectorization is conducted for statements in the sub-graph only, and the mode of serial execution is adopted for other statements harmful to vectorization. SLP algorithm can only process the codes as a whole. If vectorization for some codes can produce returns, while vectorization for other codes is harmful, vectorization for the overall codes might cause damages. Through cost analysis of SLP, vectorization processing cannot be conducted for such codes. But the algorithm proposed in this paper can solve this problem well. It will not only increase the efficiency of code vectorization, but also improve the program performance. According to the experimental results, the performance of this algorithm is improved by 9.1% on average when compared with that of SLP algorithm.

References

1. J. Fritts, F. Steiling, and J. Tucek, "MediaBench II video: Expediting the Next Generation of Video Systems Research," *Microprocessors & Microsystems*, 2005, vol. 33, no. 4, pp. 301-318
2. W. Gao, R. C. Zhao, L Han, "Research on SIMD Auto-vectorization Compiling Optimization," *Journal of Software*, 2015, 26(6):1265−1284 (in Chinese)
3. T. Hiroaki, Y. Akeuchi, K. Sakanushi, et al. "Pack Instruction Generation for Media Processors Using Multi-valued Decision Diagram," in *Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis.* ACM, 2006:154-159
4. S. Larsen, S. Amarasinghe, "Exploiting Superword level Parallelism with Multimedia Instruction Sets," *Acm Sigplan Notices*, 2000, 35(5), 145-156
5. NAS parallel benchmark suite, Avaiklable at http://www.nas.nasa.gov/ Resources/Software/npb.html, Last accessed on June 16, 2014
6. M. Prieto, L. Pinuel, F. Catthoor, et al. "Improving Superword Level Parallelism Support in Modern Compilers," *IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis*, CODES+ISSS 2005, Jersey City, Nj, Usa, September. 2005:303-308
7. L. N. Pouchet, "PolyBench: The polyhedral benchmark suite , " Available at http://www.cs.ucla.edu/˜pouchet/software/ polybench/, Last accessed on November 18, 2012
8. V. Porpodas and T. Jones, "Throttling Automatic Vectorization: When Less is More," *International Conference on Parallel Architecture & Compilation. IEEE*, 2015:432-444
9. V. Porpodas, A. Magni, T. M. JONES, "PSLP: Padded SLP Automatic Vectorization," *Code Generation and Optimization (CGO), 2015 IEEE/ACM International Symposium on. IEEE*, 2015:190-201
10. "Spec cpu2006 , " Available at http://www.spec.org/cpu2006/, Last accessed on August 24, 2015
11. W. M. Joseph, "High Performance Compilers for Parallel Computing," 1996
12. W. Y. Suo, R. C. Zhao, Y. Yao, "Superword Level Parallelism Instruction Analysis and Redundancy Optimization Algorithm on DSP," *Journal of Computer Applications*, 2012, 32(12):3303-3307
13. Z. Gtoumavitis, B. WANG, "Towards a Holistic Approach to Auto-parallelization: Integrating Profile-driven Parallelism Detection and Machine-learning Based Mapping," *Acm Sigplan Notices*, 2009, 44(6):177-187
14. S. Wei, R. C. Zhao, Y. Yao, "Loop-Nest Auto-Vectorization Based on SLP," *Journal of Software*, 2012, 23(07):1717-1728
15. J. L. Xu, R. C. Zhao, L. Han, "Vector Exploring Path Optimization Algorithm of Superworld Level Parallelism with Subsection Constraints," *Journal of Computer Applications*, 2015, 35(04):950-955