

# Language-Extension-Based Vectorizing Compiling Scheme on SDR-DSP

Xiaoqiang Ni<sup>(✉)</sup>, Liu Yang, and Chiyuan Ma

School of Computer, National University of Defense Technology,  
Deya Street 109, Changsha 410073, People's Republic of China  
xiaoqiangni@nudt.edu.cn

**Abstract.** In this paper we propose a Language-Extension-based Vectorizing Compiling Scheme (LEVCS) for a newly developed DSP. The DSP is mainly designed for Software-Defined Radio (SDR) and is called SDR-DSP. The SDR-DSP architecture mixes the styles of VLIW (Very Long Instruction Word) and SIMD (Single Instruction Multiple Data). To explore the potential of SDR-DSP and achieve high performance, vectorization is one of the must equipped critical methods. Because auto-vectorization techniques cannot satisfy the requirements of the typical application, LEVCS is used to direct the vectorization. The C-extending programming language used in LEVCS is called SDR-DSP-C. LEVCS uses flexible data reorganization to make vectorization on SDR-DSP more efficient. We use LEVCS to vectorize five benchmark kernels: Fast Fourier Transform (FFT), Finite Impulse Responsefilter (FIR) and Infinite Impulse Response filter (IIR), Dot product implementation (Dotprod), Sum of vectors (vecsum). Experiment results show that LEVCS is functional correct and can achieve 2.883–8.074 speedups comparing to TI-DSPs.

**Keywords:** SIMD · VLIW · SDR-DSP · Vectorizing compiling scheme

## 1 Introduction

With the development of wireless communication techniques, the performance requirement of DSP becomes higher and higher. Software-defined radio (SDR) is a new communication technique, which implements radio functions in software. SDR becomes popular because it meets the trend for better flexibility and scalability [1]. To meet the requirements of high-throughput and low-power, SDR processor requires more complicated architecture than traditional digital signal processor [2]. SIMD processing becomes one of the main architecture of DSP to meet real-time performance requirements of SDR solutions [4]. We design and develop a new DSP architecture for high performance applications, which is named SDR-DSP. It has a new instruction set and it is based on VLIW [6] and SIMD [7]. It includes two processing units, which are called SU (scalar unit) and VU (vector unit). SDR-DSP also has FPU (floating point units) and it can support single floating-point and double floating-point efficiently.

Today most DSP applications is implemented using a combination of both C code and assembly code. For the critical code which is important to performance, DSP

programmers use highly optimized assembly code [11]. The DSP compilers always supply libraries written in assembly code to support SIMD application.

The CEVA-X C family of DSP cores features a combination of VLIW and Vector engines that enhance typical DSP capabilities with advanced vector processing. The CEVA-XC4000 is the third generation of the CEVA-XC family [9]. Its VLIW architecture shares many similarities with TI (TexasInstrument)' C64x DSP family [8] but only supports fixed-point computation. The lack of FPU implies that the CEVA-dsp cannot efficiently support floating-point applications [10]. CEVA-DSP compiler uses the mode of combining C code with assembly code. It mainly uses assembly intrinsic for SIMD operations [11].

The C6678 DSP is a high-performance fixed/floating-point DSP based on TI's Keystone multi-core DSP architecture C66x. The C66x Digital Signal Processor (DSP) extends the performance of the C64x+ and C674x DSPs through enhancements and new features. Many of the new features target increased performance for vector processing [12]. TI's C66x compiler also uses the mode of combining C code with assembly code. It supports SIMD by using the mode of interfacing C and C++ with assembly language [13].

Writing assembly code is difficult and time consuming. The assembly programmer has to handle time consuming machine-level issues such as registers allocation and instruction scheduling. The work on vectorization must be done manually by assembly programmers. It will be more convenient if these issues can be taken care of by the compiler [11]. If compiler supports high-level vectorized language, programmers can utilize architecture characteristics by using high-level language.

SDR has a large amount of frequently changing radio communication algorithm [3], the high-level language developing environment for SDR-DSP is urgent. The architecture of SDR-DSP requires more complicated compiling techniques to develop data parallelism and instruction parallelism. SDR-DSP has special VU to support SIMD. Making good use of the SIMD architecture characteristics of SDR-DSP is critical to the performance. So the support to vectorization of SDR-DSP compiler is very important.

Today the techniques of autovectorization of compilers are not mature. Saeed Maleki et al. evaluate the vectorization capabilities of today's most popular compilers [14]: GCC (version 4.7.0), ICC (12.0) and XLC (11.1). They use different benchmarks which include a set of synthetic benchmarks, two applications from PACT and the Media Bench applications. The results of the evaluation show today's compilers can at most vectorize 45–71% of the loops in the synthetic benchmark and only 18–30% in the collection of application. Today's popular compilers are not effective in autovectorization and it is difficult in compiling field. It needs long time and large amount of research to find scientific optimized ways to solve this problem. SDR-DSP is designed for wireless communication and the requirement to develop high performance applications is urgent. It is essential for us to find a new scheme of vectorizing compiling for SDR-DSP.

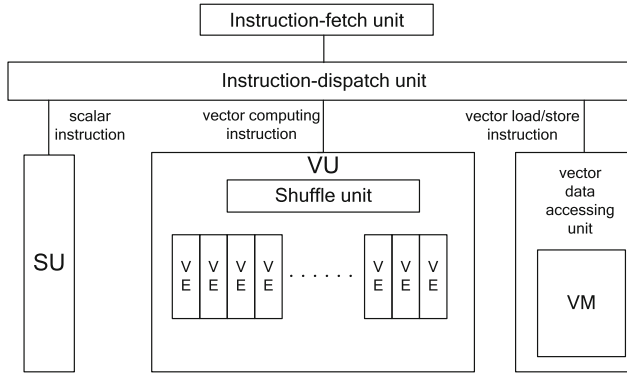
This paper gives a Language-Extension-based Vectorizing Compiling Scheme for SDR-DSP. For convenience, we call it LEVCS. We design a C-extending programming language called SDR-DSP C and develop a vectorizing compiler to support SDR-DSP C. LEVCS can support SDR-DSP C and flexible data reorganization. In this paper, Sect. 2 describes the architecture of SDR-DSP. Section 3 introduces LEVCS, including

language extending for vectorization and Data reorganization for vectorization. Section 4 gives the results of the experiment and performance analysis.

## 2 Architecture of SDR-DSP

As shown in Fig. 1, SDR-DSP consists of two processing units: SU (scalarunit) and VU (vectorunit). SDR-DSP can issue ten instructions per clock cycle. It supports instruction-level parallelism based on VLIW and data-level parallelism based on SIMD.

- SDR-DSP includes unified instruction-fetch unit and instruction-dispatch unit. The dispatch unit issues instructions for SU and VU simultaneously.
- SU performs scalar tasks and controls the flow of the execution of VU. VU performs computation-intensive parallel tasks.
- SDR-DSP has a vector memory (VM) to store vector data. Vector Data Accessing Unit is used to load and store vector data. It supports efficient data supply and transport for wide vector computation.
- VU includes a set of isomorphic VEs and the number of VEs is configurable. Each VE has local register file, accumulators and parallel functional units (MAC, ALU and BP). The parallel functional units support fixed-point and floating-point operations.



**Fig. 1.** Architecture of SDR-DSP

A lot of applications need to reorganize data within VEs in VU. To support shuffle operation, SDR-DSP has data-shuffling unit to exchange data among different VEs. There is a special shuffle-modes memory which is separate from vector memory (VM). It contains various shuffle modes. Shuffle operation permutes data among local registers in various VEs by byte, half word or word. The shuffle operation among VEs can make data exchange more efficient.

### 3 Introduction to LEVCS

Because of the SIMD characteristics of SDR-DSP architecture and the data-intensive characteristics of SDR applications, the efficient vectorizing compiler of SDR-DSP is very important.

**Language Extending for Vectorization.** For some C programs, SDR-DSP compiler uses autovectorization to analyze the parallel parts in the programs. Then the parallel parts which satisfy the conditions to be vectorized will be recognized and transformed to the vectorized code running on VU.

Because of the limitation of autovectorization we have described in Sect. 1, lots of complex C programs cannot be vectorized automatically. Some programs include complex loops which are very difficult to be analyzed and vectorized. Some programs benefit from complex SDR-DSP instructions, such as saturation arithmetic, reduction, shuffle and soon. These instructions cannot be mapped easily from high level language [5] and autovectorization for them is more difficult. To compensate the lack of the autovectorization of compiler, we put forward LEVCS for SDR-DSP. In LEVCS, we design and implement a C-extending programming language for SDR-DSP which is called SDR-DSP C.

SDR-DSP C is designed according to the instruction set and architecture of SDR-DSP. It provides support to vectorization on SDR-DSP and programmers can use SDR-DSP C to write vectorized programs conveniently. SDR-DSP C extends standard C language with some pragmas and intrinsics. SDR-DSP C extends standard C language with vector data types and vector instructions for SDR-DSP. The main vector data types are shown in Table 1.

**Table 1.** Vector data-types.

Data types	Machine mode	Signification
vec double	V8DI	A vector consisted of 8 doubles
vec float	V16SF	A vector consisted of 16 floats
vec int	V16SI	A vector consisted of 16 ints
vec short	V32HI	A vector consisted of 32 shorts
vec char	V64QI	A vector consisted of 64 chars

SDR-DSP has some complex vector instructions, such as multi-mode shuffle, multi-width reduction, complex multiplication and soon. Using these instructions effectively can greatly improve applications' performance. So we design new syntax corresponding to all of these instructions in SDR-DSP C. For example, to implement shuffle instructions, we add `v_vshufw`, `v_vshuff`, `v_vshufh` and `v_vshufb` in SDR-DSP C, to implement reduction instructions, we add `v_reduc2`, `v_reduc4`, `v_reduc8` and `v_reduc16` in SDR-DSP C. SDR-DSP C adds several pragmas to direct compiling optimization, such as: `#pragma vect`, `#pragma novect`, `#pragma unroll(n)`.

In LEVCS, we develop a high-level language developing environment for SDR-DSP. It includes a vectorizing compiler which support SDR-DSP C. It also includes assembler, linker, debugger and simulator. Programmers can use SDR-DSP C to develop

vectorized programs for SDR-DSP and use SDR-DSP compiler to parse and translate these programs into intermediate language and then output the optimized assemble code. Then the code can be assembled and linked. It supplies a friendly and convenient environment for programmers to develop applications for SDR-DSP.

**Data Reorganization for Vectorization.** The data-level parallelism in the applications of wireless communication, video and image processing always include regular data-level parallelism and irregular data-level parallelism. An application with regular data access can be considered to have regular data-level parallelism. If it has irregular data access and complex control flow, such as data-dependent control flow, the data-level parallelism is considered to be irregular. For regular data-level parallelism, vectorization on SIMD architecture always can get good effect. But for irregular data-level parallelism, the result of vectorization is always not satisfied. In such cases, the performance cannot get improved, and sometimes may be reduced. When the SIMD width is more wide, this problem becomes more serious. So based on maintaining the high efficiency of regular data-level parallelism, the supporting to irregular data-level parallelism is very important. Flexible and efficient data reorganization is essential.

Traditional SIMD architecture uses guarded instructions to support data-dependent controlflow [16]. This method uses masks to disable some SIMD lanes. But for complex branches, this method will waste computing resources and becomes inefficient. Woop [17] uses the scheme of branch fusion. SIMD lanes on different branch paths are executed sequentially and are synchronized in the branch joint. The efficient utilization of SIMDlanes are still not good.

Maven VT microarchitecture [18] uses a unique lane buffering equipment. When meeting branch, the simd lanes on various branch paths will be buffered and will be executed sequentially. In the buffering equipment, SIMD lanes with the same executing paths can be merged to improve the utilization of SIMD lanes. In this method, extra hardware is needed and the hardware cost increases.

Vector Thread Architecture [19] configures an instruction cache for each SIMD lanes. Each simd lanes can fetch instruction independently (threadfetch). This method can support data-dependent control flow effectively but the expansibility are not good. Each SIMD lanes need an instruction buffer, the hardware cost will be into lerable when SIMD width becomes more and more wide.

Dynamic Wrap [20, 21] can support branches efficiently in GPUs. It reorganizes the threads executing various branch paths in multiple wraps in to newwraps, each wrap includes threads executing the same branch path. This methods needs to allocate registers for each wrap to implement the reorganization of abundant wraps and it brings heavy register costs.

Instruction shuffle scheme is a new mechanism that can handle control-flow efficiently [22]. It stores instructions on various branch paths into a unified instruction buffer array. Instruction shuffle unit issues corresponding instructions to the SIMD lanes for various branch paths. This mechanism can implement parallel execution of various branch paths, But extra instructions are needed to support instruction shuffle and the hardware cost also increased. The reorder of the output data is still a problem.

Conditional Stream [23] supports irregular data-level parallelism on IMAGINE processor. It classified the data streams and put the data of the same operations together. The original kernel with control flow is departed into multiple kernels without control flow. This method destroys the original order of data. Many applications in communication, video and image processing are data-order dependent so there covery of data order are more complex.

Most applications of wireless communication, video and image processing are written in high-level language and the source codes in these applications are always complex and flexible. Efficient High-level language compiling is essential. It is very difficult and time consuming to do data reorganization manually. It will be more convenient if these issues can be solved by the compiler. If compiler supports data reorganization, programmers can focus on the algorithms by using high-level language.

Data reorganization is critical to vectorization, so it is essential for us to implement flexible and efficient data reorganization in the compiler for wide SIMD Architecture.

Implementing flexible and efficient data reorganization in the compiler of wide SIMD architecture is essential. LEVCS can do data reorganization for wide SIMD architecture. It implements flexible data reorganization for wide SIMD. It mainly has three modules to implement data reorganization for various requirements: Data reorganization based on Multi-Modulo, Data reorganization for wide vector filling and Data reorganization for Branch.

Many algorithms in wireless communication require complex data exchange, such as FFT [15], FIR, IIR, Hartley transform, Discrete Cosine Transform, Viterbi Decoding and soon. In such algorithms, there are many irregular access to vectors which is a problem to performance. There alparts and imaginary parts of complex numbers are always stored continuously. But in some algorithms, the real parts and imaginary parts always participate in different vector computing, or are different operators of one vector computing. In order to utilize the SIMD architecture efficiently, data reorganization is required.

To implement efficient vector data exchange, LEVCS supports data reorganization based on multi-modulo. According to the requirement of the algorithms, when data is loaded from VM to VR (vector reigister) or is stored from VR into VM, data need to be shuffled based on various modulus. The multiple modulus are designed to direct data reorganization. For each kind of modulo, there is a corresponding item in SMT.

The data are loaded from VM into VRs which will participate in vector operations. In some applications, the data loaded are not long enough to fulfill the vector. In some applications, the data loaded into 16 VEs as a vector includes data for various vector operation or various sources vectors of one vector operation. In these cases, if the data are not reorganized, these data cannot be operated in parallel, some VEs will be idle when the vector operation is executed. The wide simd architecture cannot be fully utilized. LEVCS implements Data reorganization for wide vector filling using inner reorganization, horizontal reorganization and vertical reorganization.

LEVCS identifies loops that can be simdized. Firstly, inner reorganization is used. Inner reorganization is used for one vector operation in inner loops. If the effective vector length is less than simdwidth, some VEs will be idle without data. In such case, short vectors will be combined into wide vector to let more VEs to work in parallel. If vector

length still cannot fulfill the requirements of simd width, the loops will be unrolled. After loop unrolling, more data will be reorganized to do vector filling. The compiler should find enough parallel computation and the stride of various short vectors needs to be appropriate. If the data stride is too big, the cost of loading these data will be too higher to be accepted.

If inner reorganization still cannot fulfill the requirements, horizontal reorganization is used. Horizontal reorganization is for multiple irrelevant vector operations in inner loops. Suppose a vector includes several groups of elements. Various groups contain various sources for various vector operations and each group is not wide enough for the vector width. If such vectors are not reorganized, while one group of elements participate in one vector operation, the VEs corresponding to the other groups of elements will be idle. In such case, LEVCS does horizontal reorganization. The horizontal reorganization combines various groups of multiple vectors together to form new vectors.

If data reorganization of inner loops cannot fulfill the requirements, LEVCS will take outer loops into consideration and do vertical reorganization. Vertical reorganization is for multiple irrelevant vector operations in various layers of loops. Multiple irrelevant vector operations are reorganized together to form wide vectors. LEVCS will continue to unroll outer loops to get enough data for vectors when needed.

For the wide SIMD architecture, the problems brought by branches become more serious. In order not to increase the cost of hardware, Data Reorganization for Branch solves this problem in compiling. LEVCS can do flexible data reorganization according to various cases of branches. All VEs in VU can work in parallel and need not to process various branch paths redundantly. The execution efficiency of loops with branches can be improved.

## 4 Results and Discussion

In the experiment, we use LEVCS to vectorize FFT, FIR, IIR, dotprod and vecsum programs. The programs are compiled with the vectorizing compiler of SDR-DSP. After being assembled and linked, we get the executable program and run it on the cycle accurate simulator. We use  $T_{\text{SDR-DSP}}$  to represent the cycle counts of the kernel in the program.

We also execute floating-point FFT program on TMS320C66X simulator in CCS5.1. We can get same result with the result on SDR-DSP. We use optimization level -O3 to compile the C program [24] and get the cycle counts  $T_{\text{C66X}}$ . Comparing the two experiment results, we can get the speedup from Eq. (1)

$$\text{speedup} = T_{\text{C66X}} / T_{\text{SDR-DSP}} \quad (1)$$

From that, we can see the vectorized program using LEVCS can get correct result and achieve higher performance than C program running on TI-DSP. The result of the experiment (Table 2) shows: developing vectorized program with SDR-DSP C and using the vectorizing compiler of SDR-DSP can make good use of the SIMD characteristics of SDR-DSP. We can achieve with TI-DSP. The experiment proves that LEVCS designed in this paper has validity and efficiency.

**Table 2.** Experiment result

	$T_{\text{SDR-DSP}}$	$T_{\text{C66X}}$	Speedup
FFT_float (1024)	14608	117940	8.074
FIR_float (1024 * 16)	29493	85019	2.883
IIR_float (1024)	6159	29722	4.826
vector_dotprod_float (1024)	675	4131	6.828
vector_sum_float (1024)	526	4131	7.854

## 5 Conclusion

In order to explore the performance of digital signal processor SDR-DSP, this paper designs and implements LEVCS, a Language-Extension-based Vectorizing Compiling Scheme for SDR-DSP. This design provides a vectorized programming method with a new C-extending programming language named SDR-DSP C. The corresponding vectorizing compiler is developed for SDR-DSP, including the support for SDR-DSP C and flexible data reorganization. The result of experiment shows that we can implement vectorization on SDR-DSP correctly and can get good speedups by using LEVCS. In practice, LEVCS can be used to vectorize the computing kernels of the applications on SDR-DSP.

## References

1. Harada, H., Kuroda, M., Morikawa, H., Wakana, H., Adachi, F.: The overview of the new generation mobile communication system and the role of software defined RADIO Technology. *IEICE Trans. Commun.* **E86-B**(12), 3374–3384 (2003)
2. Jo, G.-D., Sheen, M.-J., Lee, S.-H., Cho, K.-R.: ADSP-Based reconfigurable SDR platform for 3G systems. *IEICE Trans. Commun.* **E88-B**(2), 678–686 (2005)
3. Wally, H.W.: *Tuttlebee: Software Defined Radio: Enabling Technologies*. Wiley, Chichester (2002)
4. He, X., Jin, X., Wang, M., Zhou, D., Goto, S.: A 98 GMACs/W 32-core vector processor in 65 nm CMOS. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **E94-A**(12), 2609–2618 (2011)
5. Tanaka, H., Takeuchi, Y., Sakanushi, K., Mai, M., Tagawa, H., Ota, Y., Matsumoto, N.: Generation of pack instruction sequence for media processors using multi-valued decision diagram. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **E90-A**(12), 2800–2809 (2007)
6. Fisher, J.: Very long instruction word architectures and the ELI-512. In: *Proceedings of the Tenth Annual International Symposium on Computer Architecture*, pp. 140–150 (1983)
7. Lorenz M., Wehmeyer L., Drager T.: Energy aware compilation for DSPs with SIMD instructions. In: *Proceedings of the Joint Conference on Languages, Compilers and Tools for Embedded Systems, LCTES/SCOPES 2002*, pp. 94–101. ACM Press (2002)
8. Gardner, J.S.: CEVA exposes DSP six pack XC4000 family uses coprocessors to buff up the baseband. The Linley Group, Microprocessor Report, March 2012
9. CEVA-XC4000. CEVA, Inc. (2012). <http://www.ceva-dsp.com/CEVA-XC4000.html>



10. Balaish, E.: Architecture oriented C optimizations, White Paper, CEVA, Inc., January 2010
11. Balaish, E.: Combining C code with assembly code in DSP applications. White Paper, CEVA, Inc., August 2009
12. Texas Instruments: TMS320C6678 Multicore Fixed and Floating-Point Digital Signal Processor Data Manual. SPRS691C, February 2012
13. Texas Instruments: TMS320C6000 optimizing compiler v7.3 user's guide. SPRU187T, July 2011
14. Maleki, S., Gao, Y., Jess Garzarán, M., Wong, T., Padua, D.A.: An evaluation of vectorizing compilers. In: PACT, pp. 372–382 (2011)
15. Jung, Y., Yoon, H., Kim, J.: New efficient FFT algorithm and pipeline implementation results for OFDM/DMT applications. *IEEE Trans. Consum. Electron.* **49**(1), 14–20 (2003)
16. Bouknight, W.J., Denenberg, S.A., McIntyre, D.E., Randall, J.M., Sameh, A.H., Slotnick, D.L.: The Illiac IV system. In: *Proceedings of the IEEE*, vol. 60, no. 4, pp. 369–388, April 1972
17. Swoop, P., Schmittler, J.: RPU: a programmable ray processing unit for realtime ray tracing. *ACM Trans. Graph.* **24**(3), 434–444 (2005)
18. Lee, Y., Avizienis, R., Bishara, A., et al.: Exploring the tradeoffs between programmability and efficiency in data-parallel accelerators. In: *Proceedings of the IEEE International Symposium on Computer Architecture*, San Jose, USA, pp. 129–140, June 2011
19. Krashinsky, R., Hampton, M., Gerding, S., Batten, C.: The vector-thread architecture, In: *Proceedings of the IEEE International Symposium on Computer Architecture*, Saint-Malo, France, pp. 37–48, June 2004
20. Fung, W.W.L., Sham, I., Yuan, G., et al.: Dynamic Warp Formation And Scheduling For efficient GPU control flow. In: *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, Washington, DC, USA, pp. 407–420 (2007)
21. Fung, W.W.L., Sham, I., Yuan, G., et al.: Dynamic warp formation: efficient MIMD control flow on SIMD graphics hardware. *ACM Trans. Archit. Code Optim.* **6**(2), 1544–3566 (2009)
22. Wang, Y., Chen, S., Zhang, K., Wan, J., Xiaowen Chen, H., Chen, H.W.: Instruction shuffle: achieving MIMD-like performance on SIMD architectures. *IEEE Comput. Archit. Lett.* **11**(2), 37–40 (2012)
23. Kapasi, U., Dally, W.J., Rixner, S., et al.: Efficient conditional operations for data-parallel architectures. In: *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, pp. 159–170. ACM, New York (2000)
24. Texas Instruments: TMS320C6000 Optimizing Compiler v7.4 User's Guide, SPRU187 (2012)

Computer Engineering and Technology

20th CCF Conference, NCCET 2016, Xi'an, China, August

10-12, 2016, Revised Selected Papers

Xu, W.; Xiao, L.; Li, J.; Zhang, C.; Zhu, Z. (Eds.)

2016, X, 232 p. 133 illus., Softcover

ISBN: 978-981-10-3158-8