Implementation of Stereophonic Acoustic Echo Canceller on Intel IA-32 Processors with SIMD Capability

第22回 信号処理シンポジウム
2007年11月7日〜9日（仙台）

ステレオ音響エコーキャンセラの Intel IA-32 CPUによる実現

Implementation of Stereophonic Acoustic Echo Canceller on Intel IA-32 Processors with SIMD Capability

平野晃宏　　中山謙二
金沢大学 大学院 自然科学研究科 電子情報科学専攻

Akihiro HIRANO　　Kenji NAKAYAMA
Division of Electrical Engineering and Computer Science
Graduate School of Natural Science and Technology, Kanazawa University
E-mail: {hirano,nakayama}@t.kanazawa-u.ac.jp

アブストラクト　本論文では、ステレオ音響エコーキャンセラの Intel IA-32 プロセッサによる効率的な実現方法を検討する。SIMD 実現でしばしば遭遇するデータ配置問題を、データ領域の増大なしで解決している。スカラ演算による実現と比較して 4 倍以上の高速化を達成している。

Abstract　This paper presents an efficient implementation of a stereophonic acoustic echo canceller on Intel IA-32 processors with SIMD (single-instruction multiple-data) capability. An efficient data and task allocation overcomes a data alignment problem frequently encountered in a SIMD implementation without additional storage space. Compared with a scalar implementation, this implementation achieves more than four times faster execution speed.

1 Introduction

Echo cancellers are used to reduce echoes in a wide range of applications, such as teleconference systems and hands-free telephones. To realistic teleconferencing, multi-channel audio, at least stereophonic, is essential. For stereophonic teleconferencing, stereophonic acoustic echo cancellers (SAEC's) [1–3] have been studied.

Recent years, PC-based communication systems such as Skype and Messenger becomes very popular. PC-based systems are useful not only for personal communications, but also for business systems such as teleconferencing. For realistic and comfortable teleconferencing, SAEC's should be implemented on PC-based systems.

Modern processors for PC's have powerful instruction set for multimedia processing. Intel IA-32 architectures [4] have MMX (Multi Media eXtension) and also SSE (Streaming Single instruction multiple data Extension, Streaming SIMD Extention). Four-way vector operations are supported for 32-bit floating-point (FP) data.

In this paper, an efficient implementation of SAEC's on Intel IA-32 processors is discussed. Section 2 describes SAEC's. IA-32 processor is briefly described in Sec. 3, followed by some implementation issues. The proposed implementation is shown by Sec. 5. Section 6 compares the performance.

2 Stereophonic Acoustic Echo Canceller

Figure 1 shows a teleconferencing using an SAEC. This echo canceller consists of four adaptive filters corresponding to four echo paths from two loudspeakers to two microphones. Each adaptive filter estimates the corresponding echo path.

The far-end signal $\boldsymbol{x}_i(n)$ in the i-th channel at

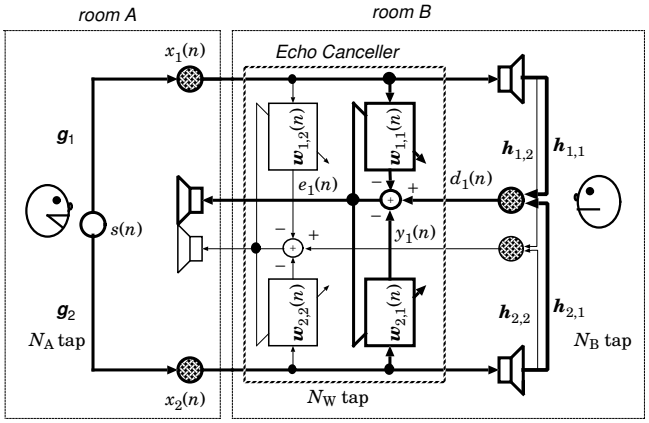


Figure 1: Teleconferencing using SAEC

time index n is generated from a talker speech $s(n)$ by passing room A impulse response $\boldsymbol{g}_i$ from the talker to the i-th microphone. $\boldsymbol{x}_i(n)$ passes an echo path $\boldsymbol{h}_{i,j}$ from the i-th loudspeaker to the j-th microphone and become an echo $d_j(n)$. Similarly, adaptive filters $\boldsymbol{w}_{i,j}(n)$ generates an echo replica $y_j(n)$. $\boldsymbol{w}_{i,j}(n)$ is so updated as to reduce the residual echo $e_j(n)$.

The SAEC generates the echo replica $y_j(n)$ by

$$y_j(n) = \boldsymbol{w}_{1,j}^T(n)\boldsymbol{x}_1(n) + \boldsymbol{w}_{2,j}^T(n)\boldsymbol{x}_2(n). \quad (1)$$

the residual echo $e_j(n)$ is calculated by

$$e_j(n) = d_j(n) - y_j(n). \quad (2)$$

Assuming the Normalized Least Mean Squares (NLMS) algorithm [5], the filter coefficient vector $\boldsymbol{w}_{i,j}(n)$ is updated by

$$\boldsymbol{w}_{i,j}(n+1) = \boldsymbol{w}_{i,j}(n) + \frac{\mu e_j(n)\boldsymbol{x}_i(n)}{|\boldsymbol{x}_i(n)|^2} \quad (3)$$

where a positive constant μ is a step-size parameter.

3 Intel IA-32 Processors [4]

In this implementation, Intel Core microarchitecture, e.g. Core2 Duo, is assumed. Main features are listed below.

- Five execution pipelines, up to fourteen stages
  - ALU, FP/MMX/SSE Move, Branch
  - ALU, FP/MMX/SSE Add
  - ALU, FP/MMX/SSE Multiply
  - Load
  - Store
- Executes up to five instructions per cycle
  - Up to three ALU operations per cycle
  - Up to three SSE operations per cycle
- Branch prediction
- 32kB instruction + 32kB data L1 cache
- 2MB or 4MB L2 cache
- Hardware prefetchers, which predict data access sequence and automatically load data from external memory into cache
- Eight general-purpose integer registers
- Eight FP registers
- Eight SSE registers

The MMX and SSE instructions [6, 7] provide some vector operations. For 32-bit floating-point data, simultaneous calculations on four independent data sets can be carried out. Therefore, up to four-times speed-up might be possible if data bandwidth allowed.

Intel NetBurst microarchitecture, e.g. Pentium4, is also compared. A large difference between Core2 and Pentium4 is a throughput for SSE instructions. Core2 processors can initiate most SSE instructions one at a cycle, while Pentium4 can initiate one at two cycles [8].

4 Considerations on Implementation

There are many considerations on implementation using general-purpose processors with SIMD capability. Examples are listed below.

- Efficient Vectorization
- Data alignment for SIMD load/store operations
- Implementation of tapped delay lines

- Memory hierarchy, especially slow external memory
- Long latency for memory load: Core2 processor requires additional six cycles even for L1 cache
- Few data registers: Eight for IA-32

The vectorization and the data alignment are common to implementation on digital signal processors (DSP's), while the others might be specific for general-purpose processors. Most DSP's are equipped with the address generators for the tapped delay lines, multiple data memories with no-wait access. Few data registers are specific for IA-32 processors.

In the four-way vectorization for FIR filtering, simultaneous calculation for k-th tap through $k+3$-th tap are common way. This causes misalignment problem on the tapped delay lines. Problems related to the data alignments and their solutions are discussed in [9]. In order to overcome misalignments, using multiple copies with different alignments has been proposed. However, four times larger memory is required for four-way SIMD processing.

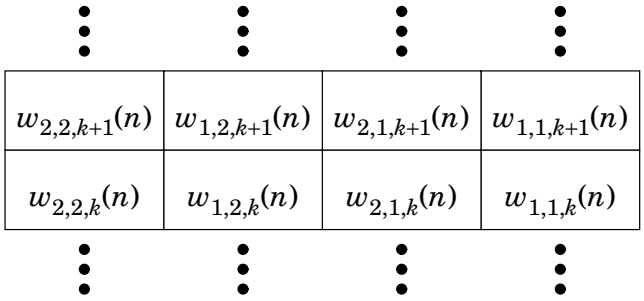
For long latency, the loop unrolling [10] is widely used. This technique requires a large number of data registers; n registers are needed to overcome n-cycle latency. Eight registers might not be sufficient for some applications.

5 Implementation of SAEC

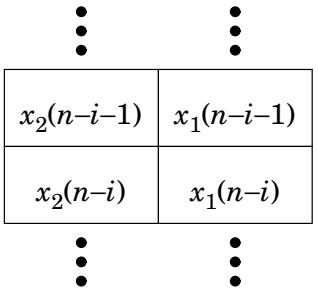
In this implementation, vectorization is carried out on the channel index rather than the time index. The k-th tap from four adaptive filters are calculated simultaneously. Figure 2 depicts the data alignments. No misalignments are occurred for this vectorization.

In order to cope with a long latency for memory access, reducing the number of data loading is a simple solution. It is also efficient for DSP's with load-store architecture [11]. The data load can be reduced by changing the order of (1) and (3). Calculating

$$w_{i,j,k}(n) = w_{i,j,k}(n-1) + \delta_j(n-1)x_i(n-k-1) \quad (4)$$



(a) Filter coefficients



(b) Tapped delay lines

Figure 2: Data alignment

and

$$sum_j(n) = sum_j(n) + w_{i,j,k}(n)x_i(n-k) \quad (5)$$

in the descending order of the tap index k could reduce the number of load operation for both $w_{i,j,k}(n)$ and $x_i(n-k)$. In (4) , $\delta_j(n-1)$ is defined by

$$\delta_j(n-1) = \frac{\mu e_j(n-1)}{|\boldsymbol{x}_i(n-1)|^2}. \quad (6)$$

$w_{i,j,k}(n)$ is a k-th element of $\boldsymbol{w}_{i,j}(n)$. The number of load operation for $x_i(n-k)$ can be reduced because $x_i(n-k)$ in (eq:convolution2) can be reused in (4) for the next $k = k-1$.

Figures 3 and 4 demonstrate the data-flow for the coefficient update and the convolution, respectively. $x_i(n-k-1)$ in Fig. 3 is a re-use data from $x_i(n-k)$ in Fig. 4 for $k = k+1$. $w_{i,j,k}(n)$ in Fig. 3 will be used in Fig. 4. Therefore, N_W load operations and N_W store operations for $w_{i,j,k}(n)$, N_W load operations for $x_i(n-k)$ is required.

To cope with the latency, two vectors are treated in a single loop. The number of vectors in a loop is restricted by the number of data

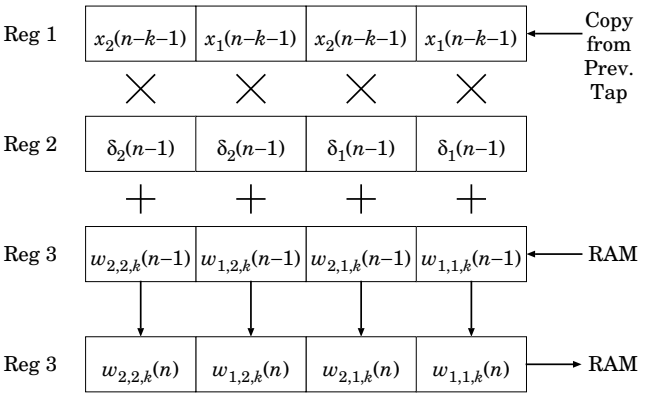


Figure 3: Data-flow for coefficient update

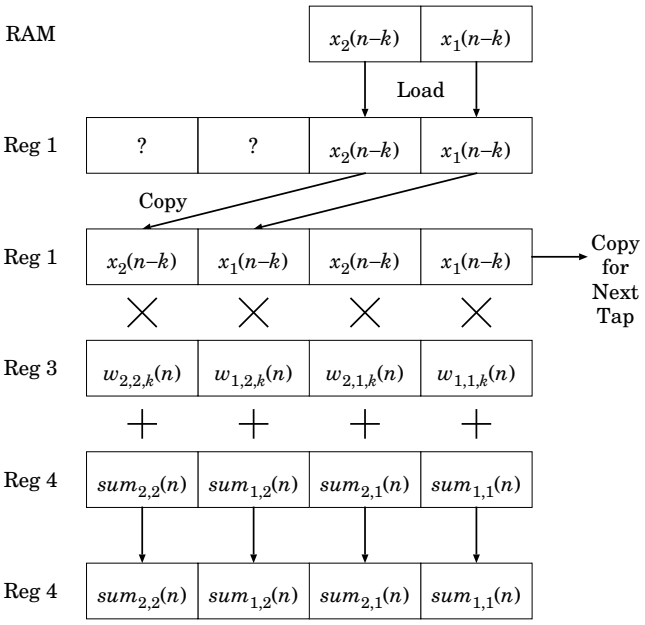


Figure 4: Data-flow for convolution

registers, rather than the efficiency. Since the longest latency in this implementation is seven cycles, eight vectors per loop might be good choice for the efficiency. However, it might require four times larger number of data registers.

In the implementation of the tapped delay lines, the circular buffer operation is required. Without a dedicated address unit, additional operations including a conditional branch per tap or extra storage space might be necessary. Though conditional branches generally degrades the pipeline performance, the branch prediction capability and the branch unit operating in parallel with ALU's and SSE units would reduce the performance degradation. The effect of the conditional branch will be examined through the evaluation of the implemented SAEC's.

Table 1: Specifications of SAEC

Sampling frequency	16kHz
Number of taps	3200
Reverberation time	200ms
Adaptation	NLMS
Precision	32-bit floating point

Table 2: Specifications of Platform

	Core2 Duo	Pentium4
Type	E6600	2.8GHz
Core Clock	2.4GHz	2.8GHz
FSB Clock	1033MHz	800MHz
L1 Data Cache	32kB	8kB
L1 Inst. Cache	32kB	12kB
L2 Cache	4MB	512kB
Chipset	Intel G965	Intel 865G

6 Performance Comparison

The standard SAEC has been implemented and tested on two different platforms. The specifications of the implemented SAEC is shown in Tab. 1. Table 2 depicts the specifications of the platforms. The critical part of the SAEC, a loop containing (4) and (5), has been programmed in an assembly language. Instructions have been scheduled for highest speed on Core2 processors. For performance comparison, a program in C language with FPU is used. SSE-based programs with and without extra delay storage are also compared.

Table 3 compares the performance. The calculation times for 120seconds of input data have been measured. By using SSE instructions, more than four times speed-up is achieved compared with the FPU version. The implementation of the tapped delay lines has almost no difference on the execution speed. This suggests the effective work of the branch prediction.

Table 3: Performance Comparison

CPU	Unit	Delay Ex	Branch	Time
Core2	FPU	Yes	No	77.50
	SSE	Yes	No	15.65
	SSE	No	Yes	15.84
Pentium4	FPU	Yes	No	98.71
	SSE	Yes	No	20.08
	SSE	No	Yes	20.28

The comparison between Core2 Duo and Pentium4 suggests that the performance is determined neither by the core clock speed nor by the throughput of the SSE instructions. Even after optimization of the load operations, the data access speed might be the bottleneck.

7 Conclusion

This paper presents an efficient implementation of SAEC's on Intel IA-32 processors with SSE. The vectorization for the channel index results in an efficient data allocation. No additional storage space is required in order to overcome both the data alignment problem and the circular buffer operations. More than four times faster execution speed compared with the FPU version is achieved.

8 Acknowledgments

The authors would like to express special thanks to Drs. Akihiko Sugiyama, Toshiyuki Nomura and Akira Inoue of Common Platform Software Research Laboratories, NEC Corporation for collaboration on and support to this research, and also for valuable comments on this paper.

References

[1] M. M. Sondhi and D. R. Morgan, "Stereophonic acoustic echo cancellation — an overview of the fundamental problem," *IEEE SP Letters*, vol. 2, no. 8, pp. 148–151, Aug. 1995.

[2] A. Sugiyama, Y. Joncour, and A. Hirano, "A stereo echo canceler with correct echo-path identification based on an input-sliding technique," *IEEE Trans. SP*, vol. 49, no. 11, pp. 2577–2587, Nov. 2001.

[3] A. Hirano, K. Nakayama, and K. Watanabe, "Convergence analysis of stereophonic echo canceller with pre-processing — relation between pre-processing and convergence —," *Proc. of ICASSP '99*, pp. 861–864, Mar. 1999.

[4] "Intel 64 and IA-32 architectures software developer's manual volume 1: Basic architecture," May 2007.

[5] J. Nagumo and A. Noda, "A learning method for system identification," *IEEE Trans. AC*, vol. 12, no. 3, pp. 282–287, Mar 1967.

[6] "Intel 64 and IA-32 architectures software developer's manual volume 2a: Instruction set reference, a-m," May 2007.

[7] "Intel 64 and IA-32 architectures software developer's manual volume 2b: Instruction set reference, n-z," May 2007.

[8] "Intel 64 and IA-32 architectures optimization reference manual," May 2007.

[9] B. Juurlink A. Shahbahrami and S. Vassiliadis, "Performance impact of misaligned accesses in SIMD extensions," *Proc. of ProRISC 2006*, pp. 334–342, 2006.

[10] David A. Patterson and John L. Hennessy, "Computer organization and design," .

[11] A. Sugiyama A. Hirano and S. Ikeda, "DSP implementation and performance evaluation of sparse-tap adaptive fir filters with tap-position control," *Proc. of ICASSP '96*, vol. 3, pp. 1295–1298, May 1996.