

## 基于 SIMD 的似然率快速算法

欧建林, 蔡 骏, 林 茜

(厦门大学计算机科学系, 厦门 361005)

**摘要:** 分析基于连续概率密度的隐马尔可夫模型大词汇量连续语音识别系统中的似然率计算方法, 阐述运用并行方式实现似然率计算的可行性, 并在此基础上, 提出一种基于 SIMD 的似然率快速算法, 通过对语音识别工具包 HTK 3.4 中似然率计算模块的改进实现该算法。实验结果表明, 在不降低识别准确率的前提下, 该算法能有效加快似然率计算的速度。

**关键词:** SIMD 技术; 似然率计算; 隐马尔可夫模型; 语音识别

## Fast Algorithm for Likelihood Ratio Based on SIMD

OU Jian-lin, CAI Jun, LIN Qian

(Dept. of Computer Science, Xiamen University, Xiamen 361005)

**【Abstract】** The likelihood ratio computation in Large Vocabulary Continuous Speech Recognition(LVCSR) systems based on continuous density Hidden Markov Model(HMM) is analyzed. The feasibility of using parallel method to implement the likelihood computation is showed. On basis of this, a fast algorithm for likelihood ratio based on SIMD is proposed, which is implemented by improving likelihood computation modules in HTK3.4 toolkit. Experimental results show this algorithm can speed up the likelihood computation without lowering the accuracy rate of recognition of premise.

**【Key words】** SIMD technology; likelihood computation; Hidden Markov Model(HMM); speech recognition

### 1 概述

目前大词汇量连续语音识别(Large Vocabulary Continuous Speech Recognition, LVCSR)系统大多采用统计模型, 其中, 语音的声学建模常用连续密度 HMM(Continuous Density HMM, CDHMM)来实现。在典型的基于 CDHMM 的 LVCSR 系统中, 每个 HMM 状态都表示为一个高斯混合模型(GMM)。为提高识别准确率, 往往要在 GMM 模型中采用较多的高斯分量, 这使得状态似然率的计算量非常大。对于这种基于似然率计算的统计语音识别系统来说, 用于高斯估计的时间在总体识别时间中占有很大的比例, 通常占 30%~70%<sup>[1]</sup>, 是语音识别速度慢的主要原因之一。有些 LVCSR 系统的识别时耗甚至几倍于实时要求。因此, 有必要设计有效的似然率快速算法, 在不降低或不明显降低识别准确率的前提下显著地加快似然率的计算, 进而提高语音识别的实时性能。

当代的新型处理机系统结构(如 Intel 的 SSE 和 AMD 的 Enhanced 3DNow!)大多具有 SIMD 部件, 并提供 SIMD 指令集以支持应用程序中数据层次的并行操作。因而采用 SIMD 技术来加快似然率计算从而提高语音识别系统的实时性能是一项有重要应用价值的研究课题<sup>[2-3]</sup>。

本文研究了在基于 HMM 的统计语音识别系统中采用 SIMD 指令实现似然率并行计算的方法, 并通过对 HTK 3.4 中似然率计算模块进行改造实现了该方法, 同时在 TIMIT 语料库上测试该方法所带来的系统实时性能的提高。

### 2 Intel SIMD 技术

SIMD 作为一种实现数据层次并行操作的计算技术, 其应用主要着眼于加快应用程序的计算速度以满足实时性要求。SIMD 的技术核心简单说就是用单一指令并行地执行

对多个数据的相同操作, 由此提高处理器的数据吞吐量。Intel IA-32 架构的 SIMD 主要由 MMX, SSE, SSE2 技术组成。其中, MMX 技术是伴随着 Pentium 处理器的诞生而出现的, 随后 Pentium III 处理器引入 SSE 技术, 而 Pentium 4 处理器中更进一步采用 SSE2 技术。

为实现 SIMD 运算, Intel 的 MMX 技术将浮点运算单元(FPU)中现有的堆栈寄存器重用为 8 个 64 bit 数据寄存器, 称为 MMX 寄存器(mm0~mm7)。MMX 指令支持 MMX 寄存器上的字节、字、双字的并行数据操作, 从而提高应用程序的性能。但 MMX 技术存在着局限性, 其不支持浮点运算。为克服 MMX 的上述局限性, SSE 技术在 CPU 中新增了 8 个 128 bit 浮点寄存器(xmm0~xmm7), 构成 XMM 浮点寄存器组, 支持单精度浮点数的并行数据处理, 并提供数据预取、数据滞后存储等高速缓存控制指令。相比于 SSE, SSE2 使用 144 个新增指令, 扩展了 MMX 技术和 SSE 技术, 这些指令提高广大应用程序的运行性能。其中的 SIMD 整数指令扩展到 128 bit, 使 SIMD 整数类型操作的有效执行率成倍提高, 双精度浮点 SIMD 指令允许以 SIMD 格式同时执行 2 个双精度浮点操作, 并且扩展数据高速缓存操作等。

### 3 HMM 的似然率计算

在基于 CDHMM 的 LVCSR 系统中, 状态为  $S$  时系统观

**基金项目:** 国家留学基金资助项目(2006104705); 厦门大学“985 工程”二期信息创新平台基金资助项目(0000-X07204); 福建省自然科学基金资助项目(2006J0043)

**作者简介:** 欧建林(1983-), 男, 硕士, 主研方向: 语音识别; 蔡 骏, 副教授、博士; 林 茜, 硕士

**收稿日期:** 2009-01-15 **E-mail:** oujianlin@gmail.com

察向量  $\mathbf{x}_n$  的似然率被视为若干具有高斯分布的概率密度函数的加权和<sup>[4-5]</sup>:

$$p(\mathbf{x}_n|S) = \sum_{k=1}^M \zeta_k p_k(\mathbf{x}_n) \quad (1)$$

其中,  $M$  是状态  $S$  中参与混合的分量的个数;  $\zeta_k$  是各混合分量的权重, 满足  $\zeta_k > 0$  且  $\sum_{k=1}^M \zeta_k = 1$ 。式(1)中的各个混合概率分量通常可以表示为一个多变量高斯概率密度函数:

$$p_k(\mathbf{x}_n) = \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}_k|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_k)\right] \quad (2)$$

其中,  $D$  是观察矢量的维数;  $\boldsymbol{\mu}_k$  和  $\boldsymbol{\Sigma}_k$  分别表示状态  $S$  的第  $k$  个分量的均值矢量和协方差矩阵。依式(2)定义各概率分量, 则式(1)中的状态模型即为具有  $M$  个高斯分量的 GMM。在实际计算中, 由于数据稀疏性, 模型中各状态的协方差矩阵往往采用对角矩阵, 因此式(1)可以表示为如下形式:

$$p(\mathbf{x}_n|S) = \sum_{k=1}^M \zeta_k \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \sum_{k=1}^M \frac{\zeta_k}{(2\pi)^{D/2} \left(\prod_{q=1}^D \sigma_{kq}^2\right)^{1/2}} \exp\left[-\frac{1}{2} \sum_{q=1}^D \frac{(x_{nq} - \mu_{kq})^2}{\sigma_{kq}^2}\right] = \sum_{k=1}^M Z_k \exp\left[-\frac{1}{2} \sum_{q=1}^D \frac{(x_{nq} - \mu_{kq})^2}{\sigma_{kq}^2}\right] \quad (3)$$

其中, 对于每个高斯分量来说,  $Z_k$  是个常数。由于构成  $Z_k$  的各项参数在识别前都是已知的, 因此  $Z_k$  可以预先计算好, 在似然率计算时只需直接从内存中读出即可。为提高计算效率及避免出现下溢, 概率计算通常在对数域中进行。因此, 可以按下式计算对数似然率:

$$\text{lb}[p(\mathbf{x}_n|S)] = \text{lbadd}_{k=1}^M \left[ \text{lb}(Z_k) - \frac{1}{2} \sum_{q=1}^D \frac{(x_{nq} - \mu_{kq})^2}{\sigma_{kq}^2} \right] \quad (4)$$

其中, 函数  $\text{lbadd}_{k=1}^M$  定义如下:

$$\text{lbadd}_{k=1}^M[w_k] = \text{lb}\left[\sum_{k=1}^M \exp(w_k)\right] \quad (5)$$

#### 4 基于SSE的似然率算法

由式(3)、式(4)可知, 似然率计算包括各高斯分量的各个元素上的一系列减法、平方、除法迭代运算, 而每个状态的高斯分量数往往相当大, 取值范围通常为 4~64, 各高斯分量本身的维数又一般高达 39 维<sup>[5]</sup>, 因而似然率计算量很大。通过进一步的分析, 还可以知道各个元素上的运算之间并不存在依存关系, 彼此间是独立的。因此, 似然率计算完全可以采用实现数据层次并行计算的 SIMD 指令并行地载入、运算多个数据, 加快计算速度。

一种最直接的方法就是运用 SIMD 指令来依次进行各个高斯分量  $\mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$  的独立运算, 每次处理一个高斯分量中的  $J$  个元素, 即将  $\mathbf{x}_n$ ,  $\boldsymbol{\mu}_k$  和  $\boldsymbol{\Sigma}_k$  向量中的  $J$  个对应的元素(即集合  $\{x_{nq}\}$ ,  $\{\mu_{kq}\}$  和  $\boldsymbol{\Sigma}_k$ ,  $q = q_0, q_0 + 1, \dots, q_0 + J - 1$ ) 分别装入 SIMD 寄存器, 然后采用 SIMD 指令并行地对寄存器的数据进行向量运算, 这样, 一个高斯分量中的  $J$  个元素就能并行地得到处理。这个并行计算过程在高斯分量的后续元素上重复进行, 直至该高斯分量的运算结束, 转而同样的方式在下一个高斯分量上进行计算。对于常用的 Intel SSE 结构的系统, 算法可具体描述如下:

**输入**  $D$  维观察向量  $\mathbf{x}_n$ ,  $D=39$  以及高斯分量  $\mathcal{N}_k = \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$

**输出** 高斯分量  $\mathcal{N}_k$  的似然率值

BEGIN

(1)对于  $\mathbf{x}_n$ ,  $\boldsymbol{\mu}_k$  和  $\boldsymbol{\Sigma}_k$ , 其内存空间分配时按 16 Byte 边界对齐;

(2)求  $\boldsymbol{\Sigma}_k$  中各元素的倒数, 存储为向量  $\boldsymbol{\Sigma}'_k$ ;

(3)初始化 xmm0 寄存器为 0;

(4)WHILE(算法尚未完成高斯分量上似然率计算)DO

BEGIN

1)将  $\mathbf{x}_n$  向量的前 12 维数据载入 xmm1~xmm3 寄存器中;

2)将  $\boldsymbol{\mu}_k$  向量的前 12 维数据载入 xmm4~xmm6 寄存器中;

3)在 xmm1~xmm6 寄存器上执行减法、乘法运算, 结果存于 xmm1~xmm3 寄存器中;

4)将  $\boldsymbol{\Sigma}'_k$  向量的前 12 维数据载入 xmm4~xmm6 寄存器中;

5)在 xmm1~xmm6 寄存器上执行乘法运算, 结果存于 xmm1~xmm3 寄存器中;

6)依次将 xmm1~xmm3 寄存器中的数据与 xmm0 寄存器中的数据对应相加, 结果保存在 xmm0 寄存器中;

7)对  $\mathbf{x}_n$ ,  $\boldsymbol{\mu}_k$  和  $\boldsymbol{\Sigma}_k$  向量中剩余的元素重复 1)~6)的操作;

END

(5)利用 SSE 中的 SHUFPS 指令对 xmm0 寄存器中的 4 个浮点数进行重排求和;

(6)返回计算结果。

END

由于 SSE 指令在数据是按 16 Byte 边界对齐的情况下才能充分发挥其性能, 因此对于  $\mathbf{x}_n$ ,  $\boldsymbol{\mu}_k$  和  $\boldsymbol{\Sigma}_k$  向量, 在系统初始化时统一按 16 Byte 边界对齐方式为其分配内存空间。这里的  $\mathbf{x}_n$ ,  $\boldsymbol{\mu}_k$  和  $\boldsymbol{\Sigma}_k$  向量其数据类型均为 32 bit 浮点数, 而一条 SSE 指令可以同时执行 4 个 32 bit 浮点数的传输或运算操作, 因此, 通过使用 SSE 中的 MOVAPS 传输指令来完成数据的并行加载以及通过使用 ADDPS, MULPS 等算术运算指令来并行地执行数据之间的算术运算, 从而并行地完成似然率计算。算法中采用将变量中的多维数据连续载入多个寄存器中的做法, 是在最大程度使用 XMM 寄存器的前提下通过连续的数据读取以降低内存访问时间。此外, 在最后要将 xmm0 中的结果从寄存器输出到内存时, 为减少内存访问的时间, 利用 SSE 中的 SHUFPS 重排指令先在寄存器之间完成 xmm0 寄存器的 4 个浮点数求和运算, 并将结果存于 xmm0 寄存器的最低 32 bit, 最后只需输出这个最低 32 bit 浮点数即可, 从而进一步节省时间开销。

#### 5 实验和结果

连续语音识别实验用以评测基于 SSE 似然率计算算法的性能。在实验中, HTK 3.4 被用作为基线系统(Baseline), 通过改造其中的内存管理模块(如 HMem.c 等文件)以实现数据变量的内存地址按 16 Byte 边界对齐; 改造似然率计算模块(如 HModel.c 和 HRec.c 等文件)以实现似然率计算的 SIMD 并行化。

在 TIMIT 语料库上进行音子级别的识别实验。TIMIT 语料库的训练集共有 3 696 个录音句子, 由 462 个说话人每人 8 句构成; 测试集有 1 344 个句子, 由 168 个说话人每人 8 句构成。实验建立 50 个单音子 HMM 声学模型, 包括 TIMIT 音素集合的 49 个单音子模型和 1 个静音模型。所有 HMM 都由 3 个自左向右的状态构成, 每个状态拥有同样的高斯分量个数。语言模型采用的是基于词循环语法。语音文件均被编

(下转第 182 页)