第35卷 第13期 计 算 机 工 程 2009年7月
Vol.35 No.13 Computer Engineering July 2009
·人工智能及识别技术· 文章编号：1000—3428(2009)13—0177—02 文献标识码：A 中图分类号：TP301.6

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 continuou density Hidden Markov Model(HMM) is analyzed. The feasibility of using t 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%[1]，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[2-3]。

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的技术核心简单说来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$\boldsymbol{x}_n$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[4-5]：

$$p(\boldsymbol{x}_n|S)=\sum_{k=1}^{M}\zeta_k\, p_k(\boldsymbol{x}_n) \tag{1}$$

其中，M是状态S中参与混合的分量的个数；ζ_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：

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

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：

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

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：

$$\mathrm{lb}\left[p(\boldsymbol{x}_n|S)\right]=\mathop{\mathrm{lb}\,\mathrm{add}}_{k=1}^{M}\left[\mathrm{lb}(Z_k)-\frac{1}{2}\sum_{q=1}^{D}\frac{(x_{nq}-\mu_{kq})^2}{\sigma_{kq}^2}\right] \tag{4}$$

其中，函数$\mathop{\mathrm{lb}\,\mathrm{add}}_{k=1}^{M}[]$定义如下：

$$\mathop{\mathrm{lb}\,\mathrm{add}}_{k=1}^{M}[w_k]=\mathrm{lb}\left[\sum_{k=1}^{M}\exp(w_k)\right] \tag{5}$$

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[5]，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$\mathcal{N}(\boldsymbol{x}_n;\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)$的独立运算，每次处理一个高斯分量中的$J$个元素，即将$\boldsymbol{x}_n$, $\boldsymbol{\mu}_k$和$\boldsymbol{\Sigma}_k$向量中的J个对应的元素(即集合$\{\boldsymbol{x}_{nq}\}$, $\{\boldsymbol{\mu}_{kq}\}$和$\boldsymbol{\Sigma}_{kq}$, $q=q_0,q_0+1,\cdots,q_0+J-1$)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，转而用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$\boldsymbol{x}_n$, D=39以及高斯分量$\mathcal{N}_k=\mathcal{N}(\boldsymbol{x}_n;\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)$

输出 高斯分量$\mathcal{N}_k$的似然率值

BEGIN

(1)对于x_n, μ_k和Σ_k，其内存空间分配时按16 Byte边界对齐；

(2)求Σ_k中各元素的倒数，存储为向量Σ_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x_n向量的前12维数据载入xmm1~xmm3寄存器中；

2)将μ_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Σ_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x_n, μ_k和Σ_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$\boldsymbol{x}_n$, $\boldsymbol{\mu}_k$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$\boldsymbol{x}_n$, $\boldsymbol{\mu}_k$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)

从表 2 可以看出，对于函数 f_1，由于函数维数和复杂度均不高，SMDE 和 DE 只需 50 次迭代就能收敛到全局最优解，解的精度也很高，但 SMDE 的收敛速度明显比 DE 快很多；SM 受初始解的影响很大，实验中其初始解是在参数空间中随机生成的，当生成的初始解落在全局最优解附近时，SM 能以较高的精度收敛到全局最优解，但绝大多数情况下都不能收敛到全局最优解。f_2 虽然维数不高，但无穷多的局部极大点将全局最优点包围，SM 无法得到最优解；DE 大多数情况下能收敛到全局最优解，但 50 次实验中有 11 次没有收敛到全局最优点；SMDE 在绝大多数情况下能收敛到全局最优解，而且解的精度比 DE 高得多，收敛速度也比 DE 快，但 50 次实验中有 3 次收敛到局部最优解。对于 f_3 和 f_4 这种高维复杂函数，SM 无法收敛到全局最优解；在给定的迭代次数下，对于 f_4，DE 无法收敛到全局最优解，但增加迭代次数至 1 500 可以收敛到全局最优解，而对于 f_3，增加迭代次数也不能收敛到全局最优解，必须调整控制参数；SMDE 对于 f_3 和 f_4 均能以很高的精度和速度收敛到全局最优解，从图中可以清楚地看到 SM 对 DE 的加速作用。对于 f_5，DE 和 SMDE 均能收敛到全局最优点，但从图 6 中可以看出，SMDE 的收敛速度和精度明显高于 DE；SM 对于高维复杂函数 f_5 不能收敛到全局最优解。另外，在进化的初始阶段，由于 DE 向全局最优解逼近较快，没有出现停滞的现象，SMDE 中 SM 没有发挥作用，因此 DE 和 SMDE 在初始阶段有相似的进化曲线。但随着 DE 连续多代(实验中设置为 5 代)没能对当前种群最优解改进，SM 对当前最优解进行局部搜索并将搜索结果替代当前种群中最差解，因此，SMDE 的收敛速度明显加快。同时，SM 的局部搜索使得全局最优解的精度更高。

4 结束语

DE/rand/1/bin 方案具有很强的全局搜索能力，实际中应用得较多，但由于具有随机性，因此收敛速度很慢，在全局最优点附近的精细搜索效果也不好。为加快 DE 算法的收敛速率，提高其精度，本文结合 SM 确定性搜索和 DE 算法全局随机性搜索的优点，提出一种高效易实现的混合优化算法 SMDE。对高维多模态复杂函数的优化表明，SMDE 的搜索精度和收敛速度明显优于 DE，具有很强的初值鲁棒性和很高的优化效率，可广泛应用于各种实际工程优化问题中。

参考文献

[1] Storn R, Price K. Differential Evolution: A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces[Z]. International Computer Science Institute, 1995.

[2] Vesterstrom J, Thomsen R. A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems[C]//Proceedings of CEC'04. Oregon, Portland: IEEE Press, 2004.

[3] 吴亮红, 王耀南, 周少武, 等. 双群体伪并行差分进化算法研究及应用[J]. 控制理论与应用, 2007, 24(3): 453-458.

[4] 何大阔, 李延强, 王福利. 基于单纯形算子的混合遗传算法[J]. 信息与控制, 2001, 30(3): 276-279.

[5] 曹志国, 汪 勇. 基于模拟退火-单纯形法的目标函数的优化[J]. 华中科技大学学报, 2005, 33(6): 67-69.

[6] 陈国初, 俞金寿. 单纯形微粒群优化算法及应用[J]. 系统仿真学报, 2006, 18(4): 862-866.

[7] Nelder J A, Mead R. A Simplex Method for Function Minimization[J]. Computer Journal, 1965, 7(4): 308-313.

编辑 张 帆

（上接第 178 页）

码成 39 维的特征矢量，包括 13 维 MFCC 系数及其一阶、二阶差分，同时训练了不同高斯分量数的 GMM 模型。

实验通过分别计算基线系统、SIMD 改造后的系统在识别过程中所用到的似然率计算时间来评测 SIMD 算法的性能。所有实验结果都是在一台 1 GB 内存、2.8 GHz 主频的 Intel Pentium 4 微机上测得的。其系统平台为 Fedora Core 4，并安装了 Intel C++ Compiler 9.1.047 编译器。实验中测试了高斯分量数从 8~64 的 GMM 模型。表 1 给出了实验结果，图 1 中为不同高斯分量数的情况下 SIMD 算法的加速比。

表 1 似然率计算算法时间对比

高斯分量个数	算法耗时/s	
	SIMD	Baseline
8	190.08	150.33
16	310.77	394.45
32	583.02	745.56
64	1 105.64	1 417.87

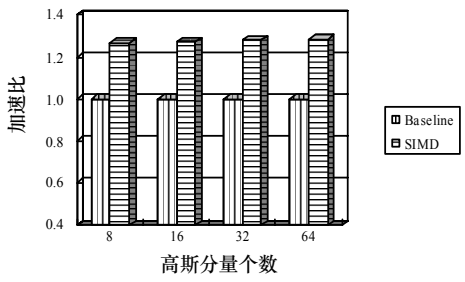

图 1 不同高斯分量的加速比

从图 1 可以看出，SIMD 算法可以提高速度 27%左右，从而节省识别过程所需要的时间开销，提高系统的实时性能。

6 结束语

本文在分析基于 HMM 的连续语音识别系统中似然率计算的基础上，提出一种基于 SSE 的似然率快速计算算法，该算法与其他似然率快速算法[5]不同，它不对似然率计算的数学模型进行修改或简化，而是利用现代微机体系结构中的硬件级并行处理方式来实现似然率计算的并行性，因而是在不影响系统的识别率的前提下加快似然率计算。实验结果表明本算法可以提高似然率计算速度，从而提高系统的实时性能。本算法在高性能微处理机日益普及的背景下具有较大的实际应用价值。

参考文献

[1] Gales M J F. State-based Gaussian Selection in Large Vocabulary Continuous Speech Recognition Using HMM's[J]. IEEE Trans. on Speech and Audio Processing, 1999, 7(2): 152-161.

[2] Kanthac S. Using SIMD Instructions for Fast Likelihood Calculation in LVCSR[C]//Proc. of ICASSP'00. Istanbul, Turkey: [s. n.], 2000.

[3] Lee Y. Mobile CPU-based Optimization of Fast Likelihood Computation for Continuous Speech Recognition[C]//Proc. of ICASSP'07. Honolulu, USA: [s. n.], 2007.

[4] 王炳锡, 屈 丹, 彭 煊. 实用语音识别基础[M]. 北京: 国防工业出版社, 2005.

[5] Pellom B L. Fast Likelihood Computation Techniques in Nearest-Neighbor-based Search for Continuous Speech Recognition[J]. IEEE Signal Processing Letters, 2001, 8(8): 221-224.

编辑 陈 文