文章编号：1001－9081(2009)04－1052－04

基于 HTK的语音识别的并行化研究与实现

刘勇进[1]，史晓东[2]
(1.厦门大学 计算机科学系，福建 厦门 361005；2.厦门大学 智能科学与技术系，福建 厦门 361005)
(liuyj03@163.com)

摘 要：详细地分析了语音识别的过程，给出了相应的算法描述，并分析了语音识别并行化的可能性。将并行计算的思想应用于语音识别的算法中，使用多线程技术，并引入避免竞争条件的机制，在多核计算机上并行地计算 HMM模型节点的似然率，从而得到语音识别的并行化算法。分析了该并行化算法的性能，同时在语音识别工具包 HTK 3.4上实现了这种并行化算法。基于 WSJ0语料库的实验结果表明该并行化算法在不影响识别结果的前提下能够有效地提高语音识别的实时性能。

关键词：多核；HTK；并行计算；语音识别
中图分类号：TP391.42 文献标志码：A

Research and implementation of parallel speech recognition based on HTK

LIU Yong-jin[1], SHI Xiao-dong[2]
(1. Department of Computer Science, Xiamen University, Xiamen Fujian 361005, China;
2. Department of Cognitive Science, Xiamen University, Xiamen Fujian 361005, China)

Abstract: After comprehensively analyzing the process of speech recognition and depicting its corresponding algorithm, the feasibility of the parallelism in the algorithm was analyzed. The parallel computing concept and the multi-threading technology were applied in the algorithm of speech recognition, and a protection mechanism was introduced to avoid the occurring race condition during the computing of likelihood of Hidden Markov Model (HMM) on multi-core computer. Then the parallel algorithm for speech recognition was proposed and its performance was also evaluated. The experiments on WSJ0 corpora demonstrate that the implementation of parallel algorithm which was realized in HTK 3.4 toolkit can greatly improve the real-time performance of speech recognition without affecting the results of recognition.

Key words: multi-core; Hidden Markov Modelling Toolkit (HTK); parallel computing; speech recognition

0 引言

现有的大词汇量连续语音识别(Large Vocabulary Continuous Speech Recognition, LVCSR)系统大都采用连续密度的 HMM(Continuous Density Hidden Markov Model, CDHMM)来实现语音的声学建模，其模型的状态数约在2000到6000之间，每个状态一般由含有 8到 64个高斯分量的高斯混合模型(Gaussian Mixture Model, GMM)来表述。在识别时，需要计算每个语音帧在所有有效状态上的似然率，这是一个非常耗时的过程，其花费的时间通常占总的识别时间的30%到 70%[1]。为了进一步提高识别准确率，模型须采用更多的高斯分量，这使得用于高斯估计的时间在总体识别时间中占有更大的比例，严重地影响了语音识别的实时性能。因此，从语音识别的实时性要求来看，在不降低识别准确率的条件下，降低语音识别的时间具有重要的意义。

目前对提高语音识别实时性能的研究主要集中在算法优化上，而在机器优化方面如并行计算在语音识别上的应用的研究还相对较少。在算法优化中，主要使用加速似然率计算的方法[1－4]来降低语音数据的识别时间，但是该方法有一个缺点，就是导致识别准确率的降低。在机器优化方面，语音识别工具 ATK(An Application Toolkit for HTK)使用了多线程技术，但是其目的在于快速响应用户的请求，并没有真正涉及到语音识别的并行化；另外一种基于 MPI技术的并行计算方法[5－7]被用来降低模型训练的时间，但是当在模型的训练中使用较大的收敛因子(Convergence factor)时，该并行方法训练模型所花费的时间远远超过了原来的非并行化模型训练的时间，这主要是由于消息传递所花费的额外时间占了较大比例，因而使用该并行方法来训练模型不是一个有效可行的方案[6]。然而可以借鉴其并行计算的思想，将并行计算应用到语音识别中来。本文研究了基于多核计算机的语音识别并行计算方法，并在 Linux版的 HTK 3.4上实现了该方法，同时在 WSJ0语料库上测试了该方法在实时性能上的提高。

1 语音识别过程分析

剑桥大学开发的 HTK(Hidden Markov Modelling Toolkit)是语音识别研究中广泛使用的一个工具包，为语音识别的研究提供了一个方便的平台，并且可在 http://htk.eng.cam.ac.uk下载到 HTK源码，因此我们在 HTK源码的基础上分析语音识别的过程。

在语音识别的过程中，一般使用 token传播算法[8－10]来搜索语音数据 $O = O_1, O_2, \cdots, O_T$的最优识别文本，其中 O_t表示 t时刻的语音向量。token的传播过程如图 1所示，$N_t = \{n_i,$

收稿日期：2008－10－10；修回日期：2008－12－08 基金项目：国家自然科学基金资助项目(60573189)；国家 863计划项目(2006AA01Z139, 2006AA010108-3)；福建省重点科技项目(2006H0038)；福建省自然科学基金项目(2006J0043)。

作者简介：刘勇进(1984－)，男，福建泉州人，硕士研究生，主要研究方向：语音识别；史晓东(1966－)，男，江苏江阴人，教授，博士生导师，主要研究方向：自然语言处理、人工智能。

n_t^1, …, n_t^N}表示 O_t对应的所有可能的 HMM模型节点的集合, P_k表示节点之间的转移概率, 每个 HMM模型节点都携带一个 token用于保存该节点的似然率 like路径信息以及其他信息。

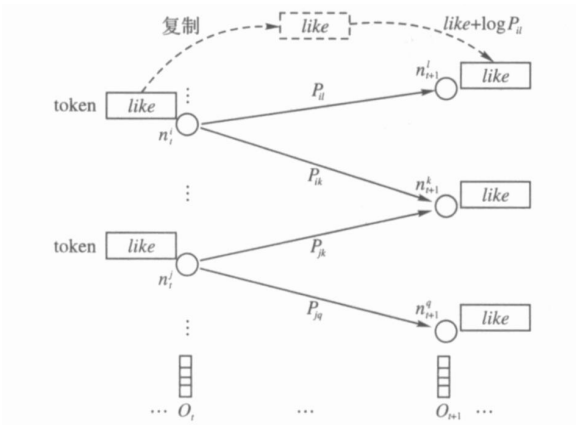

图 1　语音识别中的 token传播

在将节点 $n_t^i \in N_t$的 token传播给其后继节点前需要先计算 O_t的声学模型概率 $P(O_t \mid n_t^i)$及该节点的似然率 like同时重新设置全局最大似然率 GLIKE,其算法描述如下。

1.1　StepInsq1算法

```
INPUT: HMM模型节点 n_t^i;
       t时刻的语音向量 O_t;
BEGIN
1)  计算节点 n_t^i声学模型概率 P← P(O_t | n_t^i);
2)  计算节点 n_t^i的似然率 like← like+ P;
3)  IF like> GLIKE THEN
      GLIKE← like
    END IF
END
```

当 N_t中所有的 HMM模型节点都完成上面的计算过程后, 每个节点所携带的 token将传播给其后继节点, 该传播过程的算法描述如下。

1.2　StepInsq2算法

```
INPUT: HMM模型节点 n_t^i;
       O_{t+1}对应的 HMM模型节点集合 N_{t+1};
BEGIN
  FOR n_t^i的每个后继节点 n_{t+1}^k DO
1)    将 n_t^i的 token复制到 temp;
2)    temp like← temp like+ log P_ik;
3)    IF temp like> n_{t+1}^k当前的似然率 likeTHEN
        ① 将 temp复制给 n_{t+1}^k 的 token;
        ② 更新 n_{t+1}^k的 token的路径信息及其他信息;
      END IF
4)    N_{t+1} = N_{t+1} ∪ 狓n_{t+1}^k狕;
  END FOR
END
```

从算法 StepInsq2中可以看出, 在传播 token的同时也生成了语音向量 O_{t+1}对应的所有可能的 HMM模型节点集合 N_{t+1}。实际上, token传播算法中使用了剪枝技术[8,10]来避免搜索所有可能的节点空间, 因此只有似然率大于某个剪枝阈值如 GThresh= GLIKE-Beam的节点, 其所带的 token才会传播给后继节点。其中 Beam起着限制搜索节点个数的剪枝作用。现在给出语音向量处理过程的算法描述。

1.3　ProcessObservation算法

```
INPUT: t时刻的语音向量 O_t;
       O_t对应的 HMM模型节点集合 N_t;
BEGIN
  初始化 GLIKE和 Beam等;
2)  FOR每个 n_t^i ∈ N_t DO
      StepInsq1(n_t^i, O_t);
    END FOR
3)  设置剪枝阈值 Gthresh← GLIKE- Beam;
4)  初始化 N_{t+1}为空;
5)  FOR每个 n_t^i ∈ N_t DO
      IF n_t^i的似然率大于 GthreshTHEN
        StepInsq2(n_t^i, N_{t+1});
      END IF
    END FOR
END
```

分析 ProcessObservation算法可知, 实际上它是一个两遍(Two Pass)的过程, 分别为算法中的第 2)步和第 5)步, 为了后面描述的方便, 将第 2)步和第 5)步分别称为 Passq1和 Passq2。

2　并行化算法的设计与实现

上一章分析了基于 HTK的语音识别过程并给出了相应的算法描述, 接下来考虑语音识别并行化算法的设计与实现。

在 ProcessObservation算法中, HMM模型节点的数量一般都非常的大, 在实验中出现的个数有几万个, 这意味着 Passq1和 Passq2中可能存在着大量的运算。同时, 计算语音向量的声学模型概率 $P(O_t \mid n_t^i)$也是一个耗时的过程, 其中的用于高斯估计的时间在总体识别时间中占有很大比例, 通常占 30%到 70%[11], 因此降低声学模型概率的计算时间是降低语音识别时间的关键之一。另外, 从第 1章的分析可知, 计算语音向量在每个 HMM模型节点上的声学模型概率是独立进行的, 这就为本文并行计算所有 HMM模型节点上语音向量的声学模型概率提供了可能性。基于以上原因, 考虑 Passq1的并行化。

为了实现语音识别的并行化, 需要考虑几个方面。

1)系统的选择问题。为了充分发挥多核计算机的计算能力, 最大限度地降低语音识别的时间, 不但要求操作系统能够支持多核上的线程调度, 而且要求创建线程的时间开销尽可能的小。目前大多数 Linux系统中使用的是 NPTL[11](Native POSIX Thread Library)。NPTL实现了一对一的线程模式, 并且创建和销毁 1个线程只需大约 20μs的时间, NPTL的这些特性较好地满足了要求。

2)并行粒度的选择。并行粒度的大小将直接影响到并行性能的提高, 因此选择合适的并行粒度是非常重要的一点, 后面对并行粒度的选择作了详细的介绍。

3)语音识别并行化过程中产生的竞争条件的处理问题。重新设计语音识别的代码, 正确处理竞争条件, 从而避免产生错误的识别结果是语音识别并行化中最为困难的一步。

2.1　并行粒度的选择

并行程序中, 并行粒度的选择对效率的提高有很大的影响。并行粒度越小越有利于提高程序的并行度, 但是却增加了计算的通信量和通信次数从而导致更多的额外开销。因此需要选择合适的并行粒度才能最大限度地提高并行的效率。

以大小为 N×N的两个矩阵 A, B相乘的例子来说明在语音识别并行化过程中并行粒度的选择。令 B=(B1, B2), 其中 B1和 B2的大小都为 N×N/2, 则 AB=(AB1, AB2)。如下并行化计算 AB: 在多核计算机上创建两个线程分别计算 AB1、AB2。在 N取不同值的情况下非并行化和并行化计算 AB的性能见表 1, 表中 T1和 Tp分别表示非并行化和并行化计算 AB的总时间, S表示并行化计算 AB的加速比。

表 1 非并行化和并行化矩阵相乘性能比较

N	T_1/s	T_p/s	S
10	0.000013	0.000332	0.04
20	0.000084	0.000349	0.24
50	0.001219	0.001775	0.69
100	0.010835	0.011213	0.97
200	0.085924	0.056525	1.52
500	2.222347	1.127801	1.97
1000	22.149351	11.291114	1.96

从表 1可以看出,在并行化矩阵相乘中,当 N较小即运算量较小时,其所花费的时间远大于非并行化的时间。随着 N的增大,运算量越来越大,并行化计算 AB的加速比也越来越大,当 N达到一定值时,加速比接近 2.0。在 N较小时并行化计算 AB的性能之所以降低其主要原因是矩阵相乘的运算量较小,线程创建和调度所花费的时间在总的计算时间中占有较大的比例。

从上面的分析可知,在语音识别的并行化过程中,为了达到最大的并行性能,需要选择较大的并行粒度以减少线程创建和调度的影响,这也正是选择并行化 $Pass_1$的原因。实际上,在更小的并行粒度上如高斯分量上进行了并行化,最后的结果显示了并行化后的识别时间反而大于非并行化的识别时间。

2.2 并行算法的设计及性能分析

假设计算机的核数为 P,N_t的大小为 m,则并行化 $Pass_1$的算法 1可描述为如下。

2.2.1 $Parallel Pass_1$-1算法

```
BEGIN
1)  将 N_t分成 P份:N_t^1, N_t^2, …, N_t^P,每份的大小为 m/P;
2)  创建 P个线程:t_1, t_2, …, t_P;
3)  PFOR每个线程 t_j DO
      FOR n_t^i ∈ N_t^j DO
        StepIns_1(n_t^i, O_t);
      END FOR
    END PFOR
END
```

其中第 3)步的 PFOR表示所有的线程同时执行。

假设 $StepIns_1$的执行时间为 Δt,则 $Pass_1$的执行时间为:

$$T = m \cdot \Delta t \tag{1}$$

$Parallel Pass_1$-1的执行时间为:

$$T_P = \frac{m}{P} \cdot \Delta t + \Delta t_1 + \Delta t_2 + \Delta t_3 \tag{2}$$

其中 Δt_1表示 $Parallel Pass_1$-1算法中第 1)步所消耗的时间,其值一般是某个很小的常量,相对于 Δt来说可以忽略不计;Δt_2表示为了避免 $StepIns_1$中存在的竞争条件产生影响而对临界资源采取保护措施(如使用互斥锁机制)而引起的时间开销,该值一般比较大;Δt_3表示线程创建和调度的时间开销。

为了进一步降低算法 $Parallel Pass_1$-1的时间,必须先消除 $StepIns_1$中存在的竞争条件。$StepIns_1$中的竞争条件主要由修改全局变量如 GLIKE的值引起的。为了消除竞争条件,使用为每个线程分配一个局部变量 LIKE,并将 LIKE作为 $StepIns_1$的参数以代替 GLIKE的方法来消除竞争条件,得到消除竞争条件后的 $StepIns_1$算法如下。

2.2.2 $Parallel StepIns_1$算法

```
INPUT: HMM模型节点 n_t^i;
       时刻的语音向量 O_t;
       代替全局变量 GLIKE的 LIKE;
BEGIN
1)  计算节点 n_t^i声学模型概率:P ← P(O_t | n_t^i);
2)  计算节点 n的似然率:like ← like + P;
3)  IF like > LIKE THEN
      LIKE ← like
    END IF
END
```

重新修改 $Parallel Pass_1$-1算法得到消除竞争条件后的并行化 $Pass_1$算法 2如下。

2.2.3 $Parallel Pass_1$-2算法

```
BEGIN
1)  将 N_t分成 P份:N_t^1, N_t^2, …, N_t^P,每分的大小为 m/P;
2)  创建 P个线程:t_1, t_2, …, t_P;分配并初始化 P个局部变量:LIKE_1, LIKE_2, …, LIKE_P;
3)  PFOR每个线程 t_j DO
      FOR n_t^i ∈ N_t^j DO
        Parallel StepIns_1(n_t^i, O_t, LIKE_j);
      END FOR
    END PFOR
4)  GLIKE ← MAX(LIKE_1, LIKE_2, …, LIKE_P)归约
END
```

算法 $Parallel Pass_1$-2的执行时间为:$T'_P = \frac{m}{P} \cdot \Delta t + \Delta t_1 + \Delta t_2 + \Delta t_3$,其中 Δt_2为第 4)步所消耗的时间,在 CPU核数较少的情况,其值相对于 Δt来说一般也可以忽略不计,即:

$$T'_P = \frac{m}{P} \cdot \Delta t + \Delta t_3 \tag{3}$$

故并行化算法 $Parallel Pass_1$-2的加速比为:

$$S(P) = \frac{T}{T'_P} = \frac{m \cdot \Delta t}{\frac{n}{P} \cdot \Delta t + \Delta t_3} = \frac{P \cdot m \cdot \Delta t}{m \cdot \Delta t + P \cdot \Delta t_3} \tag{4}$$

由于 Δt_3与线程的创建和调度有关,所以该项不能忽略。在理想的情况下,即假设系统创建和调度 $t_1, t_2, \cdots, t_P$的时间开销 $\Delta t_3 = 0$,则并行化算法 $Parallel Pass_1$-2加速比为:

$$S'(P) = P \tag{5}$$

完成算法 $StepIns_1$和 $Pass_1$的并行化后,修改算法 ProcessObservation得到并行化语音向量处理算法如下。

2.2.4 $Parallel$ ProcessObservation算法

```
INPUT: t时刻的语音向量 O_t;
       O_t对应的所有 HMM模型节点集合 N_t;
BEGIN
1)  初始化 GLIKE和 Beam等;
2)  Parallel Pass_1-2();
3)  设置剪枝阈值 Gthresh ← GLIKE - Beam;
4)  初始化 N_{t+1}为空;
5)  Pass_2();
END
```

2.3 完整的语音识别并行化算法

上面详细地介绍了语音向量处理的并行化过程并给出了相应的并行化算法,现在给出完整的语音识别并行化算法如下。

$Parallel$ Recognition算法

```
INPUT: 语音数据 O = O_1, O_2, …, O_T;
BEGIN
1)  初始化 t ← 1;
2)  初始化 N_1;
3)  WHILE t ≤ T DO
      Parallel ProcessObservation(O_t, N_t);
```

```
        t← t+1;
    END WHILE
4)  输出识别结果;
END
```

3 实验及结果分析

在 HTK3.4源码上实现了前面所描述的语音识别并行化算法,涉及到的文件主要有 HVite、HRec等,主要修改了 HRec中的 StepInst()、ProcessObservation等函数从而完成了语音识别的并行化。

本文在一台 1 GB内存、2.8 GHz主频的 Dell320双核计算机上测试了以上实现的并行化算法的性能,使用的模型是在 WSJ0语料库上训练得到的从 8到 128高斯的 HMM模型,使用的测试集是从 WSJ0语料库中选取的 500个语音数据,使用的系统为 Ubuntu8.04。下面从 ParallelPass1-2和总体识别的性能这两方面对实验结果进行比较和分析。

3.1 ParallelPass1-2的性能

表 2给出了 Pass1和 ParallelPass1-2的比较结果,其中 T、T_p分别表示在识别 500个语音数据的过程中 Pass1和 ParallelPass1-2所花费的总时间,S表示 ParallelPass1-2的加速比。从表中可以看出随着高斯数的增加,Pass1的运算量越来越大即并行的粒度越大,所花费的时间也越多,ParallelPass1-2的加速比也越大,在 128高斯的 HMM模型下加速比达到了 1.33。

表 2　ParallelPass1-2算法的加速比

高斯数	T/s	T_p/s	S
8	7 158.319 7	5 982.044 4	1.20
16	7 636.064 2	6 140.446 2	1.24
32	8 276.872 6	6 611.366 4	1.25
64	9 293.882 8	7 007.360 6	1.33
128	9 791.476 1	7 354.375 5	1.33

由式(5)可知,当核数 $p=2$时,在理想情况下 ParallelPass1-2的加速比应为 2.0,但是从表中可以看出目前的结果与理想情况还有比较大的差距。造成这种差距的原因除了是在消除竞争条件时需要一定的时间开销外,更主要的是因为线程创建和调度的时间占了较大的比例,也就是说处理单个语音向量的运算量远远没有达到 2.0的加速比所要求的运算量。

3.2 总体识别的性能

表 3描述了在不同高斯数下总体识别的性能,f表示总体识别时间中 Pass1所占比例,S_0和 S分别表示理想情况和实际情况下的总体识别加速比。根据 Amdah定律,总体识别的加速比可由下式给定:

$$S(p)=\frac{1}{(1-f)+f/p} \tag{6}$$

从表中可以看出,在只并行化 Pass1的情况下,高斯数为 128时总体识别的加速比达到了 1.21。因此我们相信在实现整个识别过程的并行化如同时实现 Pass2的并行化之后,可以更大程度地提高总体识别的性能,从而更有效地减少识别的时间。

4 结语

本文在分析了整个语音识别过程的基础上提出了一种语音识别的并行化算法。由于每个 HMM节点似然率的计算是独立进行的,因此根据这个特点设计的语音识别并行化算法不会降低语音识别的准确率。最后的实验结果表明了该并行算法能够有效地提高语音识别的实时性能。在多核技术不断发展以及多核计算机日益普及的背景下,语音识别的并行化算法研究具有较大的实际应用价值。

表 3　总体识别的加速比

高斯数	f	S_0	S
8	0.39	1.24	1.07
16	0.44	1.28	1.10
32	0.53	1.36	1.13
64	0.66	1.49	1.21
128	0.77	1.63	1.21

参考文献:

[1] GALES M J F, KNILL K M, YOUNG S J. State-based Gaussian selection in large vocabulary continuous speech recognition using HMMs[J]. IEEE Transactions on Speech and Audio Processing, 1999, 7(2): 152-161.

[2] PELLOM B L, SARIKAYA R, HANSEN J H L. Fast likelihood computation techniques in nearest-neighbor based search for continuous speech recognition[J]. IEEE Signal Processing Letters, 2001, 8(8): 221-224.

[3] BOCCHIERI E. Vector quantization for the efficient computation of continuous density likelihoods[C]// Proceedings of the 1993 IEEE International Conference on Acoustics, Speech and Signal Processing. ICASSP'93. Washington, D C: IEEE Computer Society, 1993: 692-695.

[4] LINARES G, NOCERA P, MATROUF D. Partitionnement dynamique des distributions pour le calcul des émissions dans un DAP Markovien[J]. XXIIIèmes Journées d'Etude sur la Parole, 2000: 177-180.

[5] POPESCU V, BURILEANU C. Parallelizing HTK: Message passing version of the training modules[EB/OL]. [2008-08-11]. http://www.clips.imag.fr/geod/User/vladimir.popescu/COMM2006.pdf.

[6] POPESCU V, BURILEANU C. Parallel implementation of acoustic training procedures for continuous speech recognition[EB/OL]. [2008-07-10]. http://www.clips.imag.fr/geod/User/vladimir.popescu/SPED2005.pdf.

[7] POPESCU V, BURILEANU C, RAFAILA M, et al. Parallel training algorithms for continuous speech recognition, implemented in a message passing framework[EB/OL]. [2008-07-10]. http://www.eurasip.org/Proceedings/Eusipco/Eusipco2006/papers/1568981978.pdf.

[8] YOUNG S J, EVENMANN G, GALES M, et al. The HTK Book (for HTK Version 3.4)[EB/OL]. (2007-12-12)[2008-07-08]. http://htk.eng.cam.ac.uk/prot-docs/htk_books.html.

[9] YOUNG S J, RUSSELL N H, THORNTON J H S. Token passing: A simple conceptual model for connected speech recognition systems[R]. Cambridge, UK: Cambridge University Engineering Department, 1989: 1-23.

[10] HUANG XUE-DONG, ACERO A, HON H W. Spoken language processing-A guide to theory, algorithm, and system development[M]. Upper Saddle River, N J: Prentice Hall, 2001.

[11] DREPPER U, MOLNAR I. The native POSIX thread library for Linux[EB/OL]. [2008-07-09]. http://people.redhat.com/drepper/nptl-design.pdf.