一种基于数据分块的快速原地归并算法

范时平　汪林林

（重庆邮电学院软件学院　重庆400065）

摘　要　与其它排序算法相比，二路归并最适合于对两个有序子表进行排序。归并长度分别为 m 和 n 的两个有序子表，经典算法有两种。第一种算法完成归并需要◯(m+n)的附加空间，◯(m+n)次比较和移动。第二种算法是原地的，但完成归并需要◯(m+n)次比较和◯(m×n)次移动。经过长期研究，提出了一种基于数据分块的快速原地归并算法。新算法通过将数据分块、对数据块排序等方法最多用◯((m+n)log₂√(m+n))次比较和◯((m+n)^{3/2})次移动完成两个有序子表的原地归并。实验证明，该算法与经典的原地算法相比，极大地降低了元素的移动次数和算法的运行时间。

关键词　原地算法，分治法，二路归并，分块，块交换，块排序

A Fast In-place Merging Algorithm on the Basis of Dividing Data into Blocks

FAN Shi-Ping　WANG Lin-Lin

(College of Software, Chongqing University of Posts and Telecommunication, Chongqing 400065)

Abstract　Comparing with other sorting algorithms, the 2-merge algorithm is the best one to sort two sorted sublists. We have two classic algorithms to merge two sorted sublists A and B of lengths m and n. The first algorithm needs ◯(m+n) additional space, ◯(m+n) comparisons and ◯(m+n) assignments. The second algorithm is in-place and needs ◯(m+n) comparisons and ◯(m×n) assignments. After long study, a fast in-place merging algorithm on the basis of dividing data into blocks is introduced. The new algorithm at most needs ◯((m+n)log₂√(m+n)) comparisons and ◯((m+n)^{3/2}) assignments to merge two sorted sublists in-place by dividing the data into blocks and sorting the data blocks, etc. Comparing with the classic in-place algorithm, the experiments certify that the new algorithm reduces assignments and run time enormously.

Keywords　In-place algorithm, Dividing and conquering method, 2-merge, Dividing data into blocks, Block swapping, Block sorting

1　引言

给定一个长度为 n 的顺序表，每个元素由一个关键字和其它的相关信息构成，排序算法的任务就是根据关键字以非降序（或非升序）重新安排数组中的元素（不失一般性，以下叙述中我们以非降序为准）。排序算法仅允许对元素进行移动和关键字比较两个操作，并用关键字比较次数和元素移动次数来衡量排序算法的时间复杂度。

任何一个可以用计算机求解的问题所需的计算时间都与其规模有关。问题规模越小，解题所需的计算时间往往也越少，从而也越容易计算。想直接解决一个较大的问题，有时是相当困难的。在算法设计中常采用分治法来解决复杂的问题。分治法的思想就是，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。归并排序是基于分治策略的一种算法，其基本思想是把已经排好序的若干子表合并为一个表。二路归并(2-merge)则是对两个有序子表进行重排，使之合并成一个有序表。

将长度分别为 m 和 n 的两个有序子表归并为一个表，经典的算法有两种。第一种算法不是原地的，需要附加◯(m+n)的空间；其核心操作是从左向右逐一比较两个待排子表的元素，将较小元素直接移到附加空间适当位置；该算法实现归并需要◯(m+n)次元素比较和◯(m+n)次元素移动。第二种算法是原地的，思想和实现都非常简单，其核心操作是从左向右逐一比较两个待排子表的元素，将较小元素移到最终位置：当第一个子表中的当前元素不小于第二子表中的当前元素时（第一个子表中的当前元素已经处于最终位置）不产生元素移动；否则需要将第一个子表中当前元素及其后面所有未归并的元素后移一个位置才能把第二子表中的当前元素移到最终位置。第二种算法实现归并需要◯(m+n)次元素比较和◯(m×n)次元素移动。

第一种算法的元素比较和移动次数都是线性阶的，但需要附加线性的空间，在内存紧张的环境下是不可行的。第二种算法是原地的，其元素比较和移动次数远远大于第一种算法，但只需要附加常数的空间用于存储数组下标和辅助移动元素，因而适用于存储空间紧张但实时性要求不高的情况。

根据文[1,2]，归并算法是对两个有序子表进行排序的最好算法。但经典归并算法一个不是原地的，原地的归并算法其时间性能又不好，因此在内存紧张、实时性要求又高的情况下归并两个有序子表，需要我们自己设计快速原地归并算法。

经过长期研究，我们提出了一种基于数据分块的快速原

范时平　讲师，主要研究方向为算法设计与分析、面向对象技术、计算机网络等。**汪林林**　教授，硕士研究生导师，主要研究方向为数据库系统、GIS 系统、数据仓库、算法设计与分析、计算机网络等。

地归并算法。新算法通过使用数据分块和数据块排序等技术，能够极大地减少原地归并的移动次数和运行时间。以下我们分4个部分介绍该算法。第2部分简单介绍快速原地归并算法所使用的基本技术，第3部分介绍快速原地归并算法，第4部分验证快速原地归并算法的正确性并分析其性能，最后为总结。

2 基本技术简介

减少经典原地归并算法中元素的比较和移动次数，要使用一些技术，如数据块交换技术、数据分块和块排序技术、类二分搜索技术等。为使快速原地归并算法的描述简洁易懂，我们先介绍算法中使用的这些技术。

2.1 数据块交换技术

2.1.1 相邻数据块交换 在原地二路归并的过程中，我们有时会遇到第二个子表中未归并的连续 $k(1\leqslant k\leqslant n)$ 个元素比第1个子表中所有未归并的 $j(1\leqslant j\leqslant m)$ 个元素都小的情况，如果采用经典的原地二路归并算法，将这 k 个元素前移需要 k 次循环。每次循环将第1个子表中的 j 个元素后移一个位置后才能把第二个子表中的最小元素移到最终位置，k 次循环共需要 $k\times(j+1)$ 次元素移动。k 次循环后(不失一般性，我们假设 $j\geqslant k$)，第一子表的后 j 个元素(下标从 $i+1$ 到 $i+j$)右移了 k 个位置，占据下标从 $i+k+1$ 到 $i+k+j$ 的空间，第二个子表中的前 k 个元素(下标从 $i+j+1$ 到 $i+j+k$)会前移依次占据下标从 $i+1$ 到 $i+k$ 的空间。如果把两个子表中未归并的连续 $j+k$ 个元素的位置看成一个环形队列，那么这两个子表中的 $j+k$ 个元素都在环形队列中循环右移了 k 个位置。因此我们可以将这 $j+k$ 个元素中的每个元素在环形队列直接右移 k 个位置(而不是将每个元素都右移 k 次)来减少元素的移动次数。

在环形队列中右移上述 $j+k$ 个元素时，每个元素都在某个环中移动，环的个数正好是 j 和 k 的最大公约数。在每个环中完成元素的移动需要增加1次移动，将环中最左(或最右)的元素移到辅助变量中，而其它每次移动都将一个元素移动最终位置。设 $GCD(j,k)$ 表示 j 和 k 的最大公约数，由于环形队列中存在 $GCD(j,k)$ 个移动环，则上述移动所需的元素移动次数为 $j+k+GCD(j,k)$。显然当 j 或 k 较大时，$j+k+GCD(j,k)$ 远小于 $k\times(j+1)$，因此用相邻数据块交换技术能够很好地减少经典原地二路归并算法的元素移动次数。

2.1.2 等长数据块交换 交换两个长度为 L 的等长数据块，每移动3次元素完成两个元素的交换，循环 L 次需要 $3\times L$ 次元素移动。该数据块交换技术的优点是被交换的两数据块不必邻接。

2.2 数据分块与块排序技术

归并的实质就是把一个有序子表的元素分插到另一个有序子表的内部以形成非递减序列，一般是把较短子表的元素分插到较长的子表中。不失一般性，我们假设子表1的长度 m 不小于子表2的长度 n(对于 $m<n$ 的情况可对称处理)。经典的原地归并算法在分插过程中，子表1中的任何1个元素 x 都可能后移多次，其后移的次数等于 x 后面小于 x 的元素的个数 k_x，即子表2中小于 x 的元素的个数。设子表2中小于子表1中的第 $i(1\leqslant i\leqslant m)$ 个元素的元素个数为 $k_i(0\leqslant k_i\leqslant m)$，则经典原地归并算法的移动次数表示为 $\sum_{i=1}^{m}(2+k_i)$。如果我们能够设法减小 k_i，那么算法的移动次数将减少。考虑到两个子表都是有序的，可以动态计算子表2每次前移元素的个数和位置，采用相邻数据块交换技术来改进原地归并算法。但这种方法子表1中仍然有大量元素会多次后移，因此其改进程度是非常有限的。分析表明大约能减少 $n^2/2$ 次移动，移动次数仍然是 $O(m\times n)$ 的。

实际上，1个子表的元素只需要与另一个子表中相近的元素比较即可，而不必考虑与它相差很远的元素，这样可以缩小元素比较和移动的范围，从而减少元素比较和移动的次数。在这里采用算法设计中常用的分治法是合适的：把两个有序子表各分成若干块，将关键字相近的数据块放到一起，例如把每个块的最大关键字作为该块的关键字按非降序对所有数据块排序；先归并关键字最小的块，再归并关键字较大的块，直到所有元素都归并完为止。为解决相邻数据块交换时子表1中大值元素反复后移的问题，我们将两个子表分成若干相等长度的块，利用等长块交换技术将子表2的块前移到适当位置。

2.3 类二分搜索技术

由于每个数据块都是有序的，为减少元素比较次数，在归并排好序的相邻数据块时，我们用类似于二分搜索的方法确定靠后部分中前移元素的范围和它们插入到靠前部分中的位置。为便于叙述，以下我们将当前待归并数据块中位置靠前的部分称为子序列1(the first subsequence)，位置靠后的部分称为子序列2(the second subsequence)。先用类二分搜索法在子序列2中找到比子序列1的最大元素小的最大元素 x 的位置，然后用相邻数据块交换技术将子序列2中 x 及其以前的元素分插到子序列1的最大元素前。重复上述分插过程直到子序列1处理完为止，即完成子序列1和子序列2的归并(子序列2后部的若干大值元素还要参与后续归并)。

3 基于数据分块的快速原地归并算法

利用前面介绍的技术，我们设计了1个快速原地归并算法。

初始化和特殊情况的处理

(1)计算子表1的长度 m 和子表2的长度 n。

(2)确定数据块的长度 $blen$。$Blen=\lceil(m+n)^{1/2}\rceil$。

(3)如果 $n\leqslant Blen$，则用类二分搜索技术和相邻块交换技术将子表2元素分插到子表1中，归并结束。

(4)如果 m 和 n 恰好是 $Blen$ 的整数倍，则两个子表的首块长为 $Blen$；否则首块为不规则块，长度分别为 $m\%Blen$ 和 $n\%Blen$(%为模运算符)。我们还要记住两个首块的位置，以便对不规则首块作特别处理。

数据分块和数据块排序

根据前面确定的块长度 $Blen$ 从前向后扫描并划分两个子表。划分的同时以数据块元素的最大关键字作为数据块的关键字按非递减序排序。步骤如下：

(1)由于两个子表的首块可能是不规则块，用相邻数据块交换技术先将两个子表的首块排好，同时修改被移动首块的起始位置。

(2)找出子表1未排序的最小块，将其与子表2的当前块比较。

①如果子表2的当前块小，则用等长块交换技术将其交换到前面，这会破坏子表1中未排序块的有序性。否则转②。

②如果子表1的当前块是子表1未排序的最小块，不用交换，否则用等长块交换技术将子表1的当前块与子表1未排序的最小块交换。

(3)重复(2)直到子表1或子表2的块排完。

(4)如果子表2排完且子表1的块未排完,则找出子表1未排序的最小块。如果子表1的当前块是子表1未排序的最小块,不用交换,否则用等长块交换技术将子表1的当前块与子表1未排序的最小块交换。

(5)重复(4)到子表1的块排完为止。

数据排序

(1)确定待排序的当前子序列1和当前子序列2。

当前子序列1开始于最左边的未归并数据块,终止于数据块 $i(1\leqslant i)$,数据块 i 是满足其块尾元素关键字大于数据块 $i+1$ 的块首元素关键字的第1个块。当前子序列2只有数据块 $i+1$。

(2)用类二分搜索法和相邻数据块交换技术将子序列2的元素分插到子序列1中。

(3)将(2)处理后子序列2的剩余部分作为1个数据块。重复(1)、(2)直到不能找到当前子序列1和当前子序列2,归并结束。

4 算法的正确性验证与性能分析

编写一个简单的函数,通过比较相邻元素的关键字就可验证算法程序是否满足非递减要求。实验证明我们的快速原地归并算法是正确的。

算法中的每一步都能够用循环实现而不涉及递归;用指针向函数传递待归并表首地址只需要几个字节,其它参数就是几个下标值,而且每个函数内部都只需要有限的空间用于存储下标和辅助实现数据块交换,因此算法的空间复杂度是原地的。由于在算法中没有专门考虑关键字相等的元素来自不同子表的问题,因此算法是不稳定的。

算法的时间性能最好的情况是子表1的所有元素都不大于子表2的最小元素,此时比较 m 次,移动0次。接下来我们分析算法最坏情况下的比较次数和移动次数。根据子表2的长度是否超过 $Blen$ 分两种情况来考虑。

当子表2的长度 n 不超过 $Blen$ 时,如果每次只归并子表2中的一个元素并且将子表1的所有元素都后移时,移动次数最多。由于子表1中的每个元素都只后移1次,共后移 m 次。将子表2的倒数第 i 个元素移到最终位置要将子表2中的前 $(n-i+1)$ 个元素和子表1未归并的后 $k(1\leqslant k\leqslant m-i+1)$ 个元素交换。用相邻数据块交换技术完成本次交换除把 $(n-i+1)$ 个元素前移外,辅助移动次数为 $GCD(k,n-i+1)\leqslant(n-i+1)$。所以当子表2的长度不超过 $Blen$ 时,新归并算法总的元素移动次数不超过 $m+2\times\sum_{i=1}^{n}(n-i+1)=m+n\times(n+1)<2m+n+3\sqrt{m+n}+2$ 次(因为 $n\leqslant Blen<\sqrt{m+n}+1$)。由于每次比较后要么将子表2的一个元素归并(不移动元素),要么确定将子表1的最右元素后移,因此最多比较 $m+n-1$ 次。

当子表2的长度 n 大于 $Blen(\sqrt{m+n}\leqslant Blen<\sqrt{m+n}+1)$ 时,子表1有 $B1=\left\lceil\frac{m}{Blen}\right\rceil$ 块,子表2有 $B2=\left\lceil\frac{n}{Blen}\right\rceil$ 块,显然 $B1<\frac{m}{\sqrt{m+n}}+1$,$B2<\frac{n}{\sqrt{m+n}}+1$,$B1+B2<\sqrt{m+n}+1$。在块排序阶段,当子表1还有 i 个块未归并时,块排序阶段在子表1中查找最小块的比较次数最多为 $2\times(i-1)$,所以为在子表1中查找最小块而所需的比较次数最多为 $2\times\sum_{i=1}^{B1}(i-1)=\frac{B1(B1-1)}{2}<\frac{m}{2}$ 次。在块排序阶段为在两个子表的最小块中确定当前最小块的比较次数最多为 $2\times(B1+B2-1)<2\times\sqrt{m+n}$ 次(块首和块尾元素都比较时)。所以块排序阶段最多比较次数为 $2\times(B1+B2-1)+\frac{B1(B1-1)}{2}<\frac{m}{2}+2\times\sqrt{m+n}$。数据排序阶段,为确定当前子序列1和子序列2,除表的首块和尾块只比较1次外,其余块都比较2次时,所需的比较次数最多,为 $2\times(B1+B2-1)<2\times\sqrt{m+n}$。数据排序分插阶段,当两子表的块交替排列,并且除首块和尾块的其余块都先作子序列2,再作子序列1时,数据排序所需比较次数最多。设每次将长度为 L_i 子序列2的 $K_i(k_1=0)$ 个元素分插到子序列1中,子序列2余下的 L_i-K_i 元素作为下一次的子序列1,并且每次数据块交换只归并子序列2中的1个元素时,本次分插所需比较次数最多,不超过 $\sum_{j=1}^{l_{i-1}-k_{i-1}}\log_2 j+\sum_{j=1}^{k_i}\log_2 j$。所以分插的总比较次数最多为 $\sum_{i=2}^{B1+B2}\left(\sum_{j=1}^{l_{i-1}-k_{i-1}}\log_2 j+\sum_{j=1}^{k_i}\log_2 j\right)\leqslant(\sqrt{m+n}+1)\log_2(\sqrt{m+n})!\leqslant(m+n+\sqrt{m+n})\log_2\sqrt{m+n}$。综上,当子表2的长度大于 $Blen$ 时,新算法最多比较 $\frac{m}{2}+4\sqrt{m+n}+(m+n+\sqrt{m+n})\log_2\sqrt{m+n}$ 次。

下面我们分析当子表2的长度 n 大于 $Blen$ 时元素的最多移动次数。在块排序阶段,子表2的首块前移到子表1的前面时移动次数最多,最多移动 $m+2\times(1+\sqrt{m+n})$ 次。数据排序阶段,当两子表的块交替排列,并且除首块和尾块的其余块都先作子序列2,再作子序列1时,数据排序所需移动次数最多。设每次将长度为 L_i 子序列2的 $K_i(k_1=0)$ 个元素分插到子序列1中,子序列2余下的 L_i-K_i 元素作为下一次的子序列1,则本次分插所需移动次数最多,为 $2\times(L_{i-1}-K_{i-1})+\frac{k_i(k_i+1)}{2}$ 次 $(2\leqslant i\leqslant B1+B2)$。所以分插所需总的移动次数最多为 $\sum_{i=2}^{B1+B2}\left[2\times(L_{i-1}-K_{i-1})+\frac{k_i(k_i+1)}{2}\right]<3\times(m+n)+\frac{(m+n)\sqrt{m+n}}{2}-\sqrt{m+n}-2$。综上,当子表2的长度大于 $Blen$ 时,新算法最多移动 $4m+3n+\sqrt{m+n}+\frac{(m+n\sqrt{m+n})}{2}$ 次。

根据以上分析,新算法元素比较次数最多为 $\frac{m}{2}+4\sqrt{m+n}+(m+n)+\sqrt{m+n})\log_2\sqrt{m+n}$,移动次数最多为 $4m+3n+\sqrt{m+n}+\frac{(m+n)\sqrt{m+n}}{2}$ 次。

下面我们通过实验方法分析算法的平均时间性能。为使两个有序子表中元素的值能随机排列并节省输入时间,我们用 *rand* 函数动态产生运行时确定长度的两个子表,并先对两个子表排序。对相同的两个有序子表,用经典本地归并算法(以下简称经典算法)和快速原地归并算法(以下简称新算法)分别处理。两个算法的元素比较次数、移动次数分别如表1、表2所示。

表1 经典原地归并算法与快速原地归并算法的元素比较次数

子表1长 m	子表2长 n	经典原地归并算法比较次数 CO	快速原地归并算法比较次数 CN	CO:CN	$\frac{CO/CN}{\frac{m+n}{(m+n)\log_2\sqrt{m+n}}}$
10	10	19	45	1:2.37	1.82
50	50	97	237	1:2.44	2.72
500	500	999	2720	1:2.72	3.66
5000	5000	9998	33357	1:3.34	3.97
50000	50000	99995	353514	1:3.54	4.69
100000	100000	199995	547573	1:2.74	6.43
150000	150000	299991	632437	1:2.11	8.62
200000	200000	399990	681426	1:1.70	10.70
300000	300000	599990	756827	1:1.26	15.23
500000	500000	999985	882612	1.13:1	22.52
1000000	1000000	1.99997e+006	1.1684e+006	1.71:1	35.79

表2 经典原地归并算法与快速原地归并算法的元素移动次数

子表1长 m	子表2长 n	经典原地归并算法比较次数 MO	快速原地归并算法比较次数 MN	MO:MN	$\frac{MO/MN}{\frac{m+n}{(m+n)\log_2\sqrt{m+n}}}$
10	10	91	63	1.44:1	1.29
50	50	1241	433	2.87:1	1.15
500	500	124481	8119	15.3:1	1.94
5000	5000	2.5308e+007	138780	182:1	7.28
50000	50000	1.25416e+009	2.6205e+006	479:1	6.05
100000	100000	5.02197e+009	5.64877e+006	889:1	7.95
150000	150000	1.12429e+010	7.18465e+006	1565:1	11.43
200000	200000	1.99975e+010	9.945e+006	2011:1	12.72
300000	300000	4.51232e+010	1.08786e+007	4148:1	21.42
500000	500000	1.25067e+011	1.57319e+007	7950:1	31.80
1000000	1000000	4.99551e+011	2.33087e+007	21432:1	60.62

根据表1，一定范围内，新算法的元素比较次数可能会比经典算法增加2倍多，但随着待排序表的增长，新算法的元素比较次数会逐渐减少而小于经典算法的比较次数。在最坏情况下，$\frac{CO/CN}{\frac{m+n}{(m+n)\log_2\sqrt{m+n}}}=\frac{CO\times\log_2\sqrt{m+n}}{CN}$应趋于某个常数，表1中该项的值反映了一般情况相对于最坏情况的改进程度。从表1中该项的值随着表的增长而增加，这表明待归并表越长新算法的改进效果越明显。表2说明新算法能够大大减少元素的移动次数，待排序序列越长，减少的比列越大。在最坏情况下，$\frac{MO/MN}{\frac{m+n}{(m+n)\sqrt{m+n}}}$应趋于某个常数，表2中该项的值反映了一般情况相对于最坏情况的改进程度。从表2中该项的值随着表的增长而增加，这表明待归并表越长新算法的改进效果越明显。

为进一步考察新算法增加比较次数和减少移动次数的综合效果以及辅助操作对算法的影响程度，我们在赛扬1.7GCPU、256M 内存，Win98操作系统、VC++6.0环境下运行两个算法，运行时间如表3所示。

表3 经典原地归并算法与快速原地归并算法的运行时间

子表1长 m	子表2长 n	经典原地归并算法比较次数 MO	快速原地归并算法比较次数 MN	TO:TN	$\frac{TO/TN}{\frac{m+n}{(m+n)\log_2\sqrt{m+n}}}$
50000	50000	11秒	110毫秒	100	1.26
100000	100000	44秒	170毫秒	259	2.31
150000	150000	98秒	220毫秒	445	3.25
200000	200000	176秒	270毫秒	652	4.12
250000	250000	278秒	280毫秒	993	5.62
300000	300000	395秒	310毫秒	1274	6.58
350000	350000	542秒	330毫秒	1642	7.85
400000	400000	706秒	330毫秒	2139	9.57
450000	450000	902秒	380毫秒	2373	10.01
500000	500000	1070秒	440毫秒	2815	11.26
1000000	1000000	8222秒	660毫秒	12458	35.24

根据表3，新算法能够大大减少运行时间，待排序表越长，减少的比列越大。在最坏情况下，$\frac{TO/TN}{\frac{m+n}{(m+n)\sqrt{m+n}}}$应趋于某个常数，表3中该项的值反映了一般情况相对于最坏情况而言算法运行时间的减少程度。表3中该项的值随着表的增长而增加，这表明待归并表越长新算法的运行时间减少越显著。

表3中新算法运行时间的减少比例小于表2中新算法元素移动次数的减少比列，这是由于新算法所增加元素的比较次数和辅助操作，部分抵消了元素移动次数的改进效果。

总结 通过采用数据分块、对数据块进行排序等方法归并长度分别为 m 和 n 的两个有序子表($m\geq n$)，新算法最多用 $\frac{m}{2}+4\sqrt{m+n}+(m+n+\sqrt{m+n})\log_2\sqrt{m+n}$ 次比较和 $4m+3n+\sqrt{m+n}+\frac{(m+n)\sqrt{m+n}}{2}$ 次移动。新算法在最坏情况下将经典原地归并算法的比较次数从$\bigcirc(m+n)$增加到$\bigcirc((m+n)\log_2\sqrt{m+n})$，但实验表明在一般情况下比较次数增加并不明显，而且随着待排序表的增长还会比经典算法的比较次数少。新算法将经典原地归并算法的移动次数从$\bigcirc(m\times n)$降低到$\bigcirc((m+n)^{3/2})$，从而大大缩短了算法的运行时间，待排序序列越长，算法改进效果越显著。因此该快速原地归并算法具有较高的理论和实用价值。

参 考 文 献

1 严蔚敏，吴伟民. 数据结构(C 语言版). 北京：清华大学出版社，2002

2 Kruse R L，Ryba A J. Data structure and program design in C++ (Copyright 1999). 北京：高等教育出版社，2001

3 龙昭华. 面向对象程序设计教程. 西安：西安电子科技大学出版社，2003

(上接第203页)

7)、*Simulation Output*(图8)、*Message Sequence Chart*(图9)。Data Values 和 Finite States Machine(FSM)二图从略。

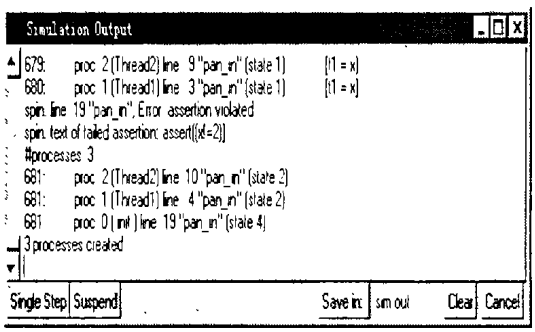

图8 模拟输出图

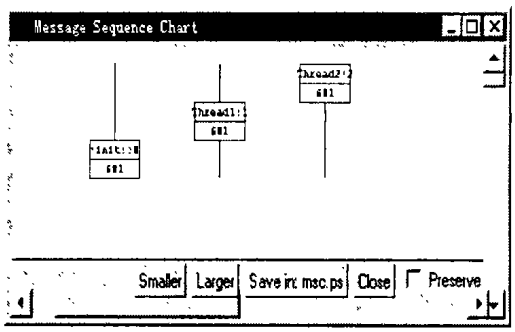

图9 MSC 图

结论 随着对模型检测技术的深入研究，出现了许多与所验证系统以及系统性质表示有密切相关的验证工具，比如：SPIN、SMV、CWB、DESIGN/CPN，UPPAAL，KRONOS，HYTECH，JPF 等。相比而言，SPIN 以其简洁明了和自动化程度高而备受注目，2001、2002年著名的 ACM"Software System Awards"分别授予 SPIN、JAVA 就是一个明证。SPIN 使用 Promela 语言对所需验证系统建模，系统所要验证的性质用 LTL 公式表示，它不仅具有很好的 GUI 以及"press on the button"验证，而且内嵌了用于状态空间压缩的优化算法(partial order redution、bitstate hashing、minimised automaton、dataflow analysis、slicing algorithm 等)，SPIN 已成功应用在安全协议验证、控制系统验证、软件验证及最优化规划等领域。

参 考 文 献

1 Clarke E M，Grumberg O，Peled D A. Model checking，Cambridge，MA：MIT Press，1999

2 林惠民，张文辉. 模型检测：理论、方法与应用. 电子学报，2002，12(A)

3 Holzmann G J. The SPIN Model Checker，Primer and Reference Manual. Addison-Wesley，2003

4 Bérard B，Bidoit M，Finkel A. System and Software Verification：Model Checking Techniques and Tools. Springer-Verlag，2001

5 http://netlib.bell-labs.com/netlib/spin

6 SPIN Online Documentation，Concise Promela，Reference，Rob Gerth，Eindhoven University，Accessible from[5]

7 Ruys T C. SPIN Tutorial：How to Become a SPIN Doctor，LNCS2318，2002

8 http://www.acm.org/awards/ssaward.html