高技术通讯 2013 年 第23 卷 第1 期:42 ~48

doi:10.3772/j.issn.1002-0470.2013.01.007

实时信息处理系统多线程框架的高效设计方法①

蔡雨辰②* 赵保军* 邓宸伟**

(*北京理工大学信息与电子学院 北京 100081)

(**卡内基梅隆大学计算机科学学院 宾夕法尼亚州 15213)

摘 要 针对目前的多线程编程模型不能快速有效地解决实时信息处理系统(RIPS)设计的问题,通过对以流水线方式工作的多线程系统进行建模,分析了两种适用于RIPS的多线程模型的时间开销与线程数量、缓冲区数量之间的关系,并基于这两种多线程模型,提出了一种高效的RIPS多线程设计方法。该方法无需过多考虑各线程的设计细节,从而能有效提高多线程RIPS的开发效率,同时保证系统的高效与稳定。在x86通用PC机平台上对该方法进行的仿真验证的结果表明,该方法能够保证RIPS的可靠性和稳定性。与基于MPI搭建的系统的对比测试结果表明,基于该方法搭建的系统具有更高的效率和鲁棒性。

关键词 实时信息处理系统(RIPS),多线程,线程同步,流水线

0 引 言

实时信息处理系统(real-time information processing system,RIPS)是一种对实时输入的外部信号进行接收及处理的系统。随着计算机技术的迅猛发展,RIPS已应用到人脸识别[1]、GPS导航[2]等各个领域。RIPS的设计广泛采用多线程技术[3-5],因为多线程能够充分利用处理器资源,有效地提高系统的吞吐率。目前广泛使用的多线程编程框架包括消息传递接口(MPI)[6]、OpenMP[7]、并行虚拟机(PVM)[8]等。MPI能提供一种易移植、高效、灵活的多线程编程接口,OpenMP能提供一种多平台共享存储器的并行程序开发接口,PVM则能提供开发跨平台的分布式计算系统的软件包。然而,这些编程接口本身没有提供足够的负载均衡能力,它们只是为编程者提供了多线程设计的底层支持,对于每个任务分配多少线程、多少缓冲区等问题都留给编程者自己解决,因而它们不适合对实时性要求十分严格的RIPS。RIPS不允许出现不合理的资源分配,否则不仅不能有效地提供系统吞吐率,甚至可能由于出现处理瓶颈或资源竞争等问题拖慢系统整体的效率,最终导致实时性的破坏。因此,这就要求我们必须研究出针对RIPS的多线程设计方法。

目前关于多线程设计的研究主要集中于线程间调度算法和硬件优化两个方向,例如伺服线程的设计[9]、多核处理器多线程开发[10]、内存或cache优化[11,12]、GPU\CPU协处理并行计算[13]等,尚没有针对RIPS的线程、缓冲区数量的具体设置的方法。鉴于此,本文提出了一种面向RIPS的多线程设计方法,它包含两种线程模型,分别适用于时间开销适中或时间开销过大的系统功能模块。我们通过对RIPS功能模块的执行方式进行建模分析,从理论上推导出两种线程模型各自的适用条件和线程数量、缓冲区数量设置的方法。利用此方法,只需对RIPS的任务进行划分,然后统计出在给定平台上各模块时间开销的均值和峰值,进而选择相应的线程模型和线程、缓冲区数量对模块进行封装,便可以完成RIPS开发。

1 流水线式的多线程系统

为了分析多线程在RIPS中的作用,我们考虑如下的系统:设外部数据按周期 P_i 输入,时间间隔为

① 863计划(2009AA8012320B)资助项目。

② 男,1984年生,博士;研究方向:并行计算,图像处理;联系人,E-mail:rickencai@gmail.com

(收稿日期:2012-03-05)

t_P;系统由两步操作(功能模块)组成,分别用 A_i,B_i 表示(其中下标 i 为周期序号)。A_i 对外部输入进行处理,结果存入一个中间缓冲区;B_i 处理中间缓冲区中的数据并输出最终结果。

如果 A_i,B_i 操作由同一个线程实现,当两步操作的时间开销总和小于数据输入时间间隔时,系统能够正确运行(如图 1(a)所示)。相反,当两步操作的时间开销总和超过数据输入时间间隔时,将导致数据竞争,造成数据丢失,如图 1(b)所示。

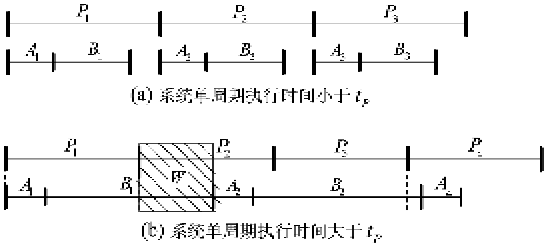


图 1　单线程系统执行时间示意图

根据系统的运作形式,B_i 依赖于 A_i 的输出,但 A_{i+1} 却不需要等待 B_i 的完成。观察图 1(b)可知,由于 A_i、B_i 处于相同的线程之内,A_2 只有等待上一周期的 B_1 结束后才能开始执行,而并未在 P_2 周期开始时刻执行。图 1(b)中阴影框 W 所示的区域为系统在 P_2 周期内浪费的时间。由于延迟时间的不断积累最终导致输入数据在其周期内得不到及时处理而引发数据丢失。因此,当系统各周期的总时间开销恒定时,系统能够稳定的工作,反之如果总时间开销不断增长,就会发生数据丢失。在图 2 中,我们将 A_i、B_i 放入不同的线程中,以流水的方式进行处理,从而获得了稳定的系统时间开销,避免了数据丢失。

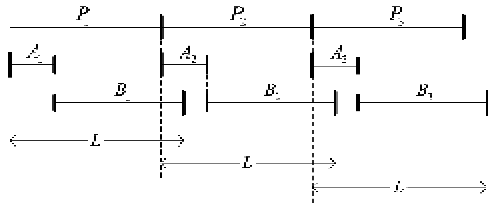


图 2　流水式的多线程系统执行时间示意图

2　One-Task-One-Thread 线程模型

2.1　可靠性分析

设一个系统被划分为 N 步连续的操作,若为每一步操作设置一个单独的执行线程,则称这种线程模型为 One-Task-One-Thread (OTOT)。在 OTOT 模型中,每个功能模块仅由一个线程实现,因此操作在某周期的起始时间受上一周期结束时间的制约。OTOT 模型的每个线程可以包含一个或多个缓冲区,下面首先讨论包含单一缓冲区的 OTOT 线程。

考虑包含两步操作 A_i,B_i 的系统。设 A_i 结束的时刻为 $t_{A_i}^{\text{end}}$,B_i 结束的时刻为 $t_{B_i}^{\text{end}}$,P_i 结束的时刻为 $t_{P_i}^{\text{end}}$。根据 OTOT 模型的特点,A_{i+1} 的开始时间同时受到 A_i 和 P_i 的制约,即 A_{i+1} 的开始时刻为 $\max(t_{A_i}^{\text{end}}, t_{P_i}^{\text{end}})$,同理 B_{i+1} 的开始时刻为 $\max(t_{B_i}^{\text{end}}, t_{A_{i+1}}^{\text{end}})$。若 B_i 的时间开销 t_B 大于数据输入的时间间隔 t_P,那么 B_i 便无法在各个周期内都紧随 A_i 之后执行(见图 3)。定义 W_B 为单周期内被延后的时间,显然有 $W_B = (t_A + t_B) - (t_P + t_A) = t_B - t_P$,容易得到首次发生数据丢失的周期 n 为

$$n = \min(n \mid (n-1) \cdot W_B > t_P) \quad (n \geqslant 2) \tag{1}$$

如图 3 所示,系统在 P_3 周期的超出时间积累量为 $3 \times W_B$,超过了数据输入时间间隔 t_P(图中阴影框 E 所示为超出的时间),导致 P_4 周期 A_4 的中间结果丢失。

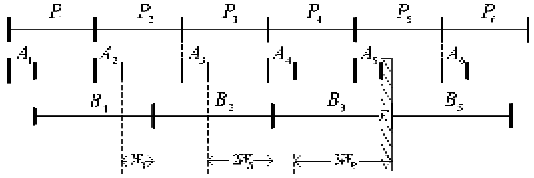


图 3　OTOT 模型搭建的系统执行时间示意图
(B_i 的时间开销大于 t_P)

综上所述,造成数据丢失的原因是由于系统的周期总时间开销不断增大,造成可用的处理时间不断减小,数据最终无法被及时处理。结合以上分析,我们给出由 OTOT 模型搭建的系统,不出现数据丢失的充要条件。

命题 1:设外部数据的输入周期为 t_P,系统被划分为 N 个连续的操作,后一项操作依赖于前一项操作的输出,每个操作由 OTOT 线程模型实现,对应的时间开销为 $t_n(n = 1,2,\cdots,N)$(假设线程在各周期的时间开销相同),则当且仅当 $\forall n$ 有

$$t_n \leqslant t_P(n = 1,2,\cdots,N) \tag{2}$$

时,系统各周期的总时间开销为常数 $L_N = \sum_{n=1}^{N} t_n$。

上述结论以线程各个周期时间开销恒定为前提,然而,实际中线程的时间开销因外部输入的不同

或其他外界因素的影响存在一定程度的波动。对于上述单缓冲区的 OTOT 线程,时间开销波动可能会引发数据丢失。我们通过为 OTOT 线程配置多个输入缓冲区来解决这个问题。当线程包含多个缓冲区时,前一个线程的输出结果便可保存在不同的缓冲区中,随着时间的推移所有缓冲区中的数据都将得到处理,从而避免数据丢失。下面给出多缓冲区 OTOT 模型的适用条件。

命题 2:设操作 K 由 OTOT 线程实现,各周期的时间开销在 t_P 左右波动,则当操作 K 各周期的时间开销均值满足

$$\overline{t_K} \leqslant t_P \tag{3}$$

时,设置多个缓冲区可以保证系统无数据丢失。

2.2 OTOT 模型缓冲区数量设置原则

本节讨论存在时间开销波动时,OTOT 缓冲区数量的选取原则。为便于分析,我们只考虑操作 K 的时间开销波动,并认为其他操作都满足命题 1 中能够保证 OTOT 正确运行的条件,即其他操作各周期的时间开销均小于 t_P。

对于操作 K,需要占用一个缓冲区是在操作 $K-1$ 的完成时刻;释放一个被占用的缓冲区是在操作 K 的完成时刻,故一段时间内操作 K 需要的缓冲区数量最大值等于这段时间内所包含的 $K-1$ 操作与 K 操作的完成次数之差。

设操作 K 在 U 个周期内所需缓冲区的总量为 M,缓冲区占用量达到最大值的周期为 I_K,此时操作 $K-1$ 所在周期为 I_{K-1},其中 $I_K \leqslant I_{K-1} \leqslant U$。设从 I_K 时刻往前第一次出现操作 K 缓冲区占用量为 1 的周期是 I,则操作 K 在周期 I 的起始时刻满足 $t^{\text{start}}_{K_{(I)}} = t^{\text{end}}_{K-1_{(I)}}$。设时间域 $[t^{\text{end}}_{K-1_{(I)}}, t^{\text{end}}_{K-1_{(I_{K-1})}}]$ 内包含操作 $K-1$ 的完成次数为 N_{K-1},包含操作 K 的完成次数为 N_K,则有

$$M = N_{K-1} - N_K \tag{4}$$

根据题设,对于任意的周期 $i \in N^+$,操作 $K-1$ 的时间开销满足 $t_{K-1_{(i)}} < t_P$,故操作 $K-1$ 在时域 $[t^{\text{end}}_{K-1_{(I)}}, t^{\text{end}}_{K-1_{(I_{K-1})}}]$ 的完成次数为

$$N_{K-1} = \lfloor \frac{\sum_{i=I}^{I_K} t_{K_{(i)}}}{t_P} \rfloor + 1 \tag{5}$$

而操作 K 在时域 $[t^{\text{end}}_{K-1_{(I)}}, t^{\text{end}}_{K-1_{(I_{K-1})}}]$ 的完成次数为

$$N_K = I_K - I \tag{6}$$

将式(5),(6)代入式(4)得

$$M = \lfloor \frac{\sum_{i=I}^{I_K} t_{K_{(i)}}}{t_P} \rfloor + 1 - (I_K - I) \tag{7}$$

取 δ_{t_K} 为操作 K 时间开销的峰值,δ_{N_K} 为 $[t^{\text{end}}_{K-1_{(I)}}, t^{\text{end}}_{K-1_{(I_{K-1})}}]$ 内操作 K 的完成次数,则由式(7)可得到

$$\begin{aligned} M &\leqslant \lfloor \frac{\delta_{N_K} \cdot \delta_{t_K}}{t_P} \rfloor + 1 - (I_K - I) \\ &\leqslant \lfloor \frac{\delta_{N_K} \cdot \delta_{t_K}}{t_P} \rfloor + 1 \end{aligned} \tag{8}$$

其中,δ_{N_K} 为一个经验值,本文取 $\delta_{N_K} = 5$。于是,我们得到 OTOT 需要的缓冲区数量的上限由式(8)给出。

3 One-Task-Multi-Thread 线程模型

3.1 可靠性分析

OTOT 模型的弊端是线程对当前周期的操作需要等待之前的操作完成之后才能进行,如果等待时间不断积累则会导致数据丢失。若为操作设置多个并行执行的线程,则可以摆脱线程对自身的时序依赖,具体的实现方法如下:

为操作 K 设置多个并行执行的线程,每个周期使用其中的一个线程来完成。线程初始时处于挂起状态,操作 $K-1$ 完成时,其输出结果被存入某个挂起线程的缓冲区内,同时唤醒该线程处理输入数据,待操作完成后线程重新回到挂起状态。我们称这样的线程模型为 One-Task-Multi-Thread (OTMT)。

下面分析 OTMT 模型的可靠性。根据第 1 节的结论,若系统各周期的总时间开销恒定,则能够保证无数据丢失。由 OTMT 模型搭建的系统,对于任意的操作 $n(n = 1,2,\cdots,M)$ 和任意周期 i,显然有 $t^{\text{start}}_{n_{(i)}} = t^{\text{end}}_{n-1_{(i)}}$ 成立,即 $t^{\text{end}}_{n_{(i)}} - t^{\text{end}}_{n-1_{(i)}} = t^{\text{end}}_{n_{(i)}} - t^{\text{start}}_{n_{(i)}} = t_n$,故系统在周期 i 的时间开销总和满足

$$T_i = \sum_{n=1}^{M} (t^{\text{end}}_{n_{(i)}} - t^{\text{end}}_{n-1_{(i)}}) = \sum_{n=1}^{M} t_n \tag{9}$$

当不考虑系统的时间开销波动时,式(9)的值是一个常数;若考虑时间开销波动,我们不妨令 t_n 取时间开销的峰值,这样式(9)就是各周期时间开销总和的上限。只要拥有足够的线程保证 $t^{\text{start}}_{n_{(i)}} = t^{\text{end}}_{n-1_{(i)}}$ 成立,就可以保证 OTMT 稳定运行。

3.2 OTMT 模型线程数量设置原则

OTMT 模型的每个线程包含一个缓冲区,因此所需的缓冲区数量等于所需设置的线程数。若假设操作在各个周期的时间开销恒定,则 OTMT 模型缓冲区数量设置原则如下:

命题 3:设系统包含 N 个连续的操作,各操作的时间开销 $t_n(n=1,2,\cdots,N)$ 满足

$$\begin{cases} t_n > t_P & (n=K) \\ t_n \leqslant t_P & (n \neq K) \end{cases} \tag{10}$$

若操作 K 使用 OTMT 模型实现,则至少需要的缓冲区数量(线程数量)为

$$n_b = \lfloor \frac{t_K}{t_P} \rfloor + 1 \tag{11}$$

当考虑时间开销波动时,OTMT 模型的缓冲区数量选取类似以下形式:

命题 4:操作 K 由 OTMT 模型实现,其时间开销波动的峰值为 δ_{t_K},则所需的缓冲区数量(线程数量) n_b' 满足

$$n_b' = \lfloor \frac{\delta_{t_K}}{t_P} \rfloor + 1 \tag{12}$$

4 RIPS 多线程设计方法

前文给出了 OTOT 和 OTMT 两种线程模型,OTOT 模型的优点是资源占用少,缺点是无法适用于时间开销较大的任务;OTMT 模型的优点是能够有效地处理时间开销大的任务,缺点是资源占用较大。理想的 RIPS 设计是在保证系统正确、稳定的前提下,尽量降低资源开销,因此在 RIPS 设计中我们优先选择 OTOT 模型。根据前文的讨论,我们给出 RIPS 的具体设计方法:首先对系统的整体操作进行分割,尽量使划分后单步操作的时间开销控制在外部数据输入周期时间的 70% ~80% 之内。设系统被划分为 N 个连续的操作,则操作 $K(K=1,2,\cdots,N)$ 的实现方式如表 1所示。

表 1 RIPS 多线程设计方法

时间开销均值	时间开销峰值	线程类型	线程数量	缓冲区数量
$\leqslant 80\% \cdot t_P$	$\leqslant 50\% \cdot \bar{t}_K$	OTOT	1	1
	$(50\% \cdot \bar{t}_K, 70\% \cdot \bar{t}_K$	OTOT	1	$5 \cdot \lfloor \delta_{t_K}/t_P \rfloor + 1$
	$> 70\% \cdot \bar{t}_K$	OTMT	$\lfloor \delta_{t_K}/t_P \rfloor + 1$	$\lfloor \delta_{t_K}/t_P \rfloor + 1$
$\leqslant 90\% \cdot t_P$	$\leqslant 30\% \cdot \bar{t}_K$	OTOT	1	$5 \cdot \lfloor \delta_{t_K}/t_P \rfloor + 1$
	$> 30\% \cdot \bar{t}_K$	OTMT	$\lfloor \delta_{t_K}/t_P \rfloor + 1$	$\lfloor \delta_{t_K}/t_P \rfloor + 1$
$> 90\% \cdot t_P$	$(0, \infty)$	OTMT	$\lfloor \delta_{t_K}/t_P \rfloor + 1$	$\lfloor \delta_{t_K}/t_P \rfloor + 1$

5 仿真结果

5.1 仿真系统设计

我们搭建了一个 RIPS 模型用于验证前文所述的方法。测试系统通过一个循环发送线程模拟外部输入,以固定时间间隔 t_P 发送测试数据,操作通过 OTOT 或 OTMT 模型实现。称不考虑时间开销波动时为理想情况,引入时间开销波动后为实际情况。操作 K 的时间开销通过延时等待来模拟,验证理想情况时,t_K 取固定值;由于实际情况中线程时间开销波动受外界因素影响,不便于模拟,本文使用高斯分布对时间开销进行建模,即 t_K 设为服从高斯分布的随机变量,其数学期望等于操作 K 时间开销理论值,方差等于时间开销的波动幅度。为了尽可能减少随机误差,我们采取多次测量求均值的方法统计仿真结果。

5.2 仿真结果及分析

测试环境:通用 x86 结构 PC 机,Inter® Pentium® D 3.20GHz 处理器,1GB 内存。固定 t_P 为 100ms,数据包大小为 1MB,每项测试均重复 100 次。以下测试 1、2 验证理想情况,以下测试 3、4、5 验证实际情况,测试内容及结果分析如下:

测试 1. OTOT 模型性能测试。操作 K 由 OTOT 模型实现,令 t_K 从 80ms 逐渐增加,记录丢包情况(见图 4)。由图可知,随着操作时间开销的不断增加,系统丢包率呈对数增长。

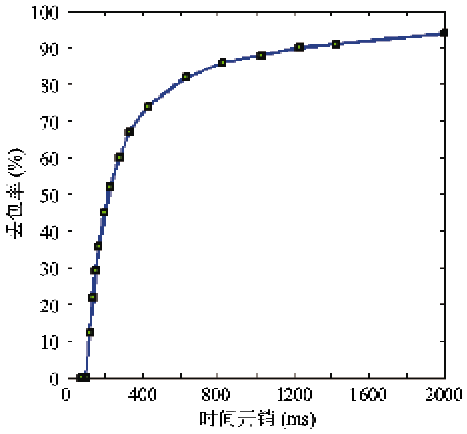


图 4 理想情况下单缓冲 OTOT 模型不同时间开销时的系统丢包率情况

测试 2. OTMT 模型线程数量需求测试。操作 K 由 *OTMT* 模型实现，令 t_K 从 0ms 逐渐增加，记录下满足丢包率为 0 时所需的线程数量最小值（见图 5）。可见，理想情况下所需线程数与时间开销同输入周期的比值成正比。

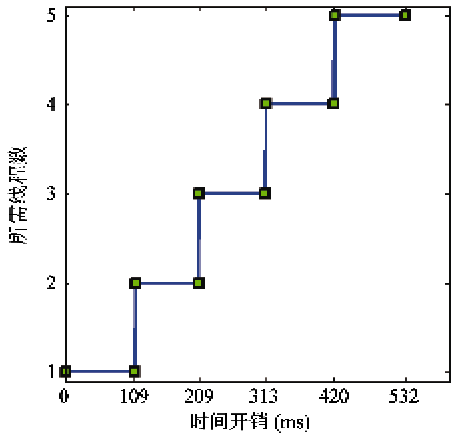


图 5 理想情况下 OTMT 模型保证系统 0 丢包率下的最少线程数

测试 3. 单缓冲区 OTOT 模型可靠性测试。操作 K 由 OTOT 模型实现，令 $\bar{t}_K$ 从 70ms 递增到 120ms，令时间开销的波动范围为 10% ~70%，记录各种组合下的系统丢包率（见图 6）。随着 $\bar{t}_K$ 越来越接近于 t_P，系统对于时间开销波动的容忍能力逐渐降低，当 $\bar{t}_K \geqslant t_P$ 时，OTOT 模型已经不再适用。

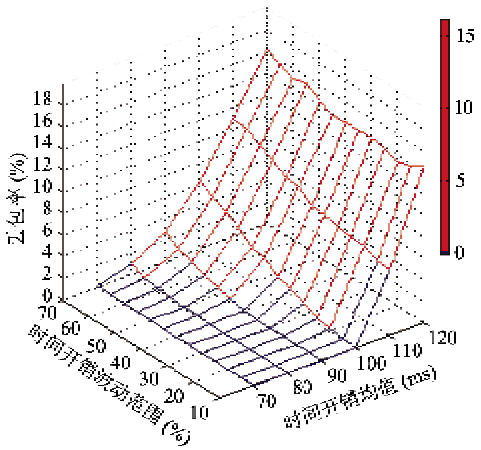
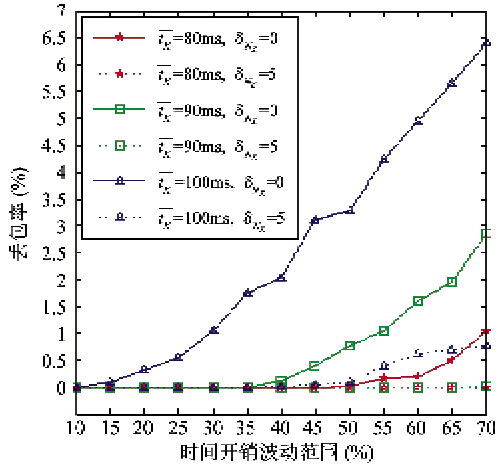


图 6 单缓冲 OTOT 模型时间开销均值、时间开销波动范围与系统丢包率之间的关系

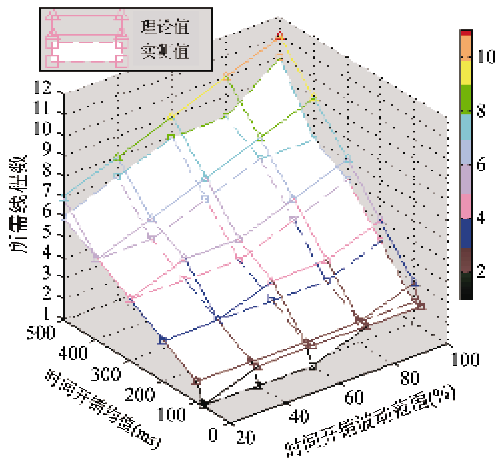
测试 4. 单缓冲区与多缓冲区 OTOT 模型性能对比测试。令 $\bar{t}_K$ 为 80ms，90ms 或 100ms，时间开销的波动范围从 10% 到 70% 逐渐变化，比较 $\delta_{N_K} = 0$（即单缓冲区）与 $\delta_{N_K} = 5$（缓冲区数由式(8)确定）两种参数设置下的系统丢包率（见图 7）。可见，设置多个缓冲区可以提高 OTOT 的鲁棒性，但随着 $\bar{t}_K$ 的增加，多缓冲 OTOT 的容忍能力亦在下降，当 $\bar{t}_K$ 过大时采用多缓冲 OTOT 并不能解决丢包问题。



实线：单缓冲；虚线：多缓冲；星号：$\bar{t}_K$ = 80ms，方框：$\bar{t}_K$ = 90ms，三角形：$\bar{t}_K$ = 100ms

图 7 实际情况下单缓冲区与多缓冲区 OTOT 模型对比测试

测试 5. OTMT 模型线程数量需求测试。令 $\bar{t}_K$ 从 80ms 逐渐增加，时间开销的波动范围从 20% 到 100%，记录下满足丢包率为 0 时所需的线程数量最小值（见图 8）。从图中可以看出，当波动较小时，理论值与实测值基本吻合，当波动较大时，理论值略高于实测值。可见，采用理论值设置线程数量能够满足系统需求，且不会造成过多的资源浪费。



三角实线：由式(12)确定的理论值；方框虚线：实测值

图 8 在无数据丢失前提下，OTMT 系统时间开销均值、时间开销波动范围与所需线程数量之间的关系

测试 1 和测试 2 分别显示了理想情况下 OTOT 和 OTMT 模型的性能；测试 3 与测试 4 分别显示了 OTOT 模型在使用单缓冲区与多缓冲区时对于时间

开销波动的承受能力,其中测试 4 证明了多缓冲 OTOT 模型能够在一定程度上降低因时间开销波动引发的丢包率,并给出了取 $\delta_{N_K}=5$ 时系统的性能指标;测试 5 反映了实际情况下 OTMT 模型在保证正确性的前提下线程数量理论需求与实际需求的对比情况。以上实验数据也表明了表 1 所提出的设计方案能够保证系统正确运行,能够正确指导 RIPS 的多线程开发。

进一步,我们将所提理论应用于一卫星遥感图像的地面接收系统(x86 通用 PC 机平台)。该系统的数据传输速率是 7.5Mbps,传输 4 倍压缩的分辨率为 496×658 的 8 位灰度图像,数据包大小为 482 字节,发送周期为 494μs。系统划分为数据接收、码流组帧、图像解压缩、图像显示及存储 4 步操作,统计各步操作所需的时间开销均值及峰值如表 2 所示。根据各步的时间开销以及表 1 给出的设计方法

表 2　各步操作时间开销统计

操作步骤	时间开销均值	时间开销峰值
数据接收	91μs	150μs
码流组帧	235μs	297μs
图像解压缩	94ms	107ms
图像显示及存储	70ms	87ms

对该 RIPS 进行实现,资源配置如表 3 所示(其中图像解压缩、图像显示及存储两步操作由于要等待一帧图像接收完成后才能进行,所以这两步对应的表 1中的 t_{fr} 不再是外部输入周期,而是一帧图像的传输周期,为83.3ms)。同时,为了评价本文所提设计方法的资源消耗情况,另基于 MPI 模型对该 RIPS 进行实现,在保证系统丢包率为零的前提下通过反复实验获得各操作所需的最少线程数量如表 3 右侧所示(MPI模型各线程均配备1个缓冲区)。由于

表 3　本文方法与基于 MPI 方法的资源消耗对比

资源类型	操作步骤	本文方法	MPI 方法
线程数量	数据接收	1	1
	码流组帧	1	1
	图像解压缩	2	2
	图像显示及存储	1	2
缓冲区数量	数据接收	1	1
	码流组帧	1	1
	图像解压缩	2	1
	图像显示及存储	6	1

对线程的维护等操作会消耗宝贵的 CPU 资源,影响系统处理速度,因此本文方法选择用资源换效率的策略。从结果可以看出,在优先保证最少的线程资源开销前提下,本文方法所使用的线程资源各步操作均少于或等于基于 MPI 的方法,同时也导致了某些操作的缓冲区数量开销高于后者。为了进一步验证本文方法在系统性能上的表现,我们对两种设计方法下系统各步操作的累积时间开销进行统计。系统前两步操作的累积时间开销是从某个数据包到达时刻起直到该步操作完成时刻止所经过的时间;系统后两步操作的累计时间开销是从某帧图像第一个数据包到达时刻起直到该步操作完成时刻止所经过的总时间。操作的累积时间开销相比于操作各自的时间开销,更能够反应操作对系统整体吞吐率和稳定性的影响。对两种系统实现分别进行 1000 帧图像的测试,统计各操作的累积时间开销均值如表 4 所示。从实验结果可以看出,基于本文方法的系统各操作的累积时间开销均值普遍低于基于 MPI 的系统,说明使用本文的设计方法能够获得更高的系统吞吐率,同时由于留给各操作之间的间歇时间更长,使得系统承受突发时间开销波动的能力更强,系统具有更高的鲁棒性。

表 4　本文方法与基于 MPI 方法的累积时间开销对比

累积时间开销	本文方法	MPI 方法
数据接收	100μs	102μs
码流组帧	335μs	370μs
图像解压缩	183ms	191ms
图像显示及存储	257ms	278ms

6　结　论

本文通过对理想情况下 RIPS 任务的执行时间进行建模,得出了 OTOT、OTMT 两种线程模型保证系统稳定运行的必要条件。随后在已有结论的基础上增加对时间开销波动的考虑,对两种线程模型加以改进,给出了实际情况下 OTOT、OTMT 的设计方法。并根据所得结论,给出了 RIPS 系统的多线程设计方法。最后,通过仿真实验和实际测试验证了所提方法的正确性。

根据本文所提方法,开发者只需要将系统划分为若干连续操作,统计出在给定平台下各操作所需的平均时间开销和峰值,即能确定每个操作所使用的线程模型以及相应的线程数量、缓冲区数量。本文所提的多线程设计方法使开发者不需要过多地考

虑各线程的设计细节,从而能够有效提高多线程 RIPS 的开发效率,同时保证系统的高效与稳定。

参考文献

[1] An K H, Chung M. Cognitive face analysis system for future interactive TV. *Consumer Electronics, IEEE Transactions on*, 2009, 55(4): 2271-2279

[2] Garcia-Quinchia A, Guo Y, Martin E, et al. A system-on-chip (SOC) platform to integrated inertial navigation systems&GPS. In: Proceedings of the IEEE International Symposium on Industrial Electronics, Seoul, Korea, 2009. 603-608

[3] Yu Y P. Object oriented teleconsultations in global PACS using multi-thread Java. In: Proceedings of the International Conference on System Sciences, Wailea, USA, 1997. 166-175

[4] Ma J, Ji Z, Cui M, et al. Design and implementation of the web-based real-time remote expert identification system: used for biological quarantine. In: Proceedings of the International Conference on Information Engineering and Computer Science, Wuhan, China, 2010. 1-4

[5] Takeda R, Nakadai K, Takahashi T, et al. Speedup and performance improvement of ica-based robot audition by parallel and resampling based block-wise processing. In: Proceedings of the International Conference on Intelligent Robots and Systems, Taipei, China, 2010. 1949-1956

[6] Traff J L, Gropp W D, Thakur R. Self-consistent MPI performance guidelines. *IEEE Trans Parallel Distrib Syst*, 2010, 21(5): 698-709

[7] Ayguade E, Copty N, Duran A, et al. The design of OpenMP tasks. *IEEE Trans Parallel Distrib Syst*, 2009, 20(3): 404-418

[8] Gropp W, Lusk E. Goals guiding design: PVM and MPI. In: Proceedings of the International Conference on Cluster Computing, Chicago, USA, 2002. 257-265

[9] Stankovic N, Zhang K. A distributed parallel programming framework. *Software Engineering, IEEE Transactions on*, 2002, 28(5): 478-493

[10] Choi G S, Das C R. A superscalar software architecture model for multi-core processors (MCPS). *System Software*, 2010, 83: 1823-1837

[11] 王东滨, 胡铭曾, 智慧等. 面向网络数据实时检测的多线程内存管理技术. 高技术通讯, 2008, 18(12): 1231-1235

[12] Owens J D, Houston M, Luebke D, et al. GPU computing. *Proceedings of the IEEE*, 2008, 96(5): 879-899

[13] Liang S, Wang C, Liu Y, et al. CUKNN: A parallel implementation of *k*-nearest neighbor on CUDA-enabled GPU. In: Proceedings of the IEEE Youth Conference on Information, Computing and Telecommunication, Beijing, China, 2009. 415-418

An efficient multi-thread framework designing method for real-time information processing systems

Cai Yuchen*, Zhao Baojun*, Deng Chenwei**

(*Institute of Information and Electronics, Beijing Institute of Technology, Beijing 100081)

(**Institute of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA)

Abstract

The study was conducted as follows to solve the problem that existing multi-threading programming models cannot be used to effectively design a real-time information processing system (RIPS): A pipeline-wise multi-thread architecture was modeled, and the relationship between the overhead time and the number of threads or buffers in two multi-thread models suitable for RIPS was analyzed, and then, an efficient multi-thread programming method for design of RIPS was proposed based on these two models. The results of the experiment performed on the x86 based PC platform show that the proposed method can guarantee RIPS' efficient and smooth running. And the comparative test shows that the RIPS designed based on the proposed method has the higher efficiency and robustness than that designed based on the programming framework of message passing interface (MPI). The method need not to consider the design details of each thread, so it can improve the efficiency of RIPS design and bring the system a good stability.

Key words: real-time information processing system (RIPS), multi-thread, thread synchronization, pipeline