EAST PCS 降低延迟方法①

刘 佳, 肖炳甲, 袁旗平

(中国科学院合肥物质科学研究院, 合肥 230031)

摘 要: 现期等离子体控制系统的控制周期是 100 微秒, 由于数据传输、多 CPU 间算法匹配失当等原因造成的延迟可以达到 80 微秒, 但理论上的延迟应当不超过 15 微秒, 因此需要平衡多 CPU 间的负载来提高 CPU 间的配合度, 从而在降低延迟的同时也可以提高控制系统的性能. 同时, 通过多线程并行技术及共享内存技术将系统中较耗时的读取反射内存卡(Reflective Memory, RFM)过程与进程并行进行, 可以将系统的控制周期缩短到 50 微妙, 而延迟也可以降低到 20 微秒以下, 从而极大的提高 PCS 的工作效率, 降低不必要的延迟, 避免成为多系统集群中的时间瓶颈.

关键词: 控制周期; 延迟; 负载均衡; 并行; 线程; 共享内存

Latency Reduction of EAST PCS

LIU Jia, XIAO Bing-Jia, YUAN Qi-Ping

(Hefei Institutes of Physical Science, Chinese Academy of Sciences, Hefei 230031, China)

Abstract: The control cycle of the Plasma Control System(PCS)currently is 100 μs and the delay could reach to 80 μs due to the data transfer and the inadequate cooperation among different CPUs. However, the delay in theory is within 15μs, so the loads of all the CPUs need to be balanced in order to reduce the delay and improve the performance of this control system. In addition, with the multi-thread parallel technique and shared memory, the time-consuming process of reading data from RFM can run in parallel with the main process, which could shorten the control cycle to 50 μs and the delay to 30 μs or less. Therefore, the efficiency of PCS could be highly promoted and the reduction of unnecessary delay could prevent PCS from becoming the bottleneck of the whole TOKAMAK control system.

Key words: control cycle; delay; load balancing; parallel; thread; shared memory

EAST(Experimental Advanced Superconductive Tokamak)是国家大科学工程项目全超导托卡马克实验装置[1], 其等离子控制系统 PCS(Plasma Control System)是一个小型的集群, 由四个节点构成. PCS 作为整个 EAST 控制系统中的一环, 实际延迟远大于理论延迟, 使得 PCS 成为整个控制系统的性能瓶颈之一.

本文通过多线程并行来提高程序性能, 降低延迟. 然而简单地在计算机 CPU 上增加多个核并不能增加传统应用程序代码的运行速度[2], 因为传统应用代码没有充分考虑到双核乃至多核的运行情况, 导致线程的平均分配时间以及线程之间的沟通时间大大增加[3], 从而造成程序在不同 CPU 间不停切换而产生的“乒乓效应”, 这是在做程序并行时需要消除的现象.

1 研究背景及目的

1.1 研究背景

1.1.1 EAST PCS 集群

EAST PCS 集群由一个主节点(Host)和三个实时节点(PCSRT)构成, 各节点间使用 Myrinet 网络进行通讯[4], 集群连接图 1 所示.

① 基金项目:国家磁约束核聚变能发展专项(2012GB105000;2011GB101000);国家 973 重点基础研究发展计划基金(国内配套研究)(2014GB103000);国家自然科学基金青年科学基金(11205200)

收稿时间:2015-06-08;收到修改稿时间:2015-08-10

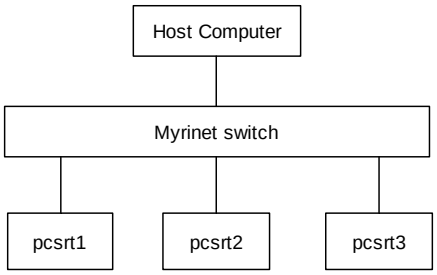

图 1　EAST PCS 集群示意图

如图 1 所示, PCS 集群共有三个实时计算机, 4 个 CPU 分布在这三台机器上, 其中 CPU1 在 rt1 上, CPU2 和 CPU3 均分布在 rt2 上, CPU4 在 rt3 上. rt1 与采集卡相连, 并装有一块反射内存卡(RFM). rt1 从采集卡读取的采集数据以及从 RFM 卡上读取的一些算法控制数据构成了 PCS 系统的输入流. 与此同时, rt3 上也装有一块 RFM 卡, 用于将 PCS 处理后的控制命令传输出去, 供 EAST 中其他系统使用.

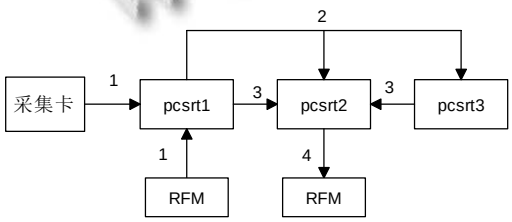

图 2　PCS 数据流示意图

PCS 的数据流如图 2 所示, 分为以下四步:

1) pcsrt1 从采集卡及 RFM 卡读取数据;

2) pcsrt1 将读取到的数据分发到各机器上;

3) 每台机器上各算法对应的 CPU 对数据进行处理, 最后再统一发送给 rt3 上;

4) pcsrt3 上的 CPU4 将收到的数据一次性写入 RFM 卡中.

数据流从 rt1 进入, 再从 rt3 输出到 RFM 的过程所花费的时间即 PCS 在整个系统中的数据流的占用时间叫做 PCS 延迟, 即本文的研究重点.

1.1.2 国外研究现状

韩国 KSTAR 装置也是一个托卡马克装置, 其 PCS 延迟非常低, 原因在于其将所有的 CPU 全部放在一台实时机器上, 采用共享内存的方式让多 CPU 间进行数据交换. 这样实现的好处有以下几点:

1) 避免了输入流数据分发的工作;

2) 避免了各算法处理数据后再统一传输到某一台机器上的工作;

3) CPU 间使用共享内存信息交换更快;

4) 更容易协调统一多 CPU 间的时间, 防止 CPU 配合失当造成的空转或等待.

同时, KSTAR PCS 还采用了多线程的方式将所有对 RFM 读写的工作都移到了线程当中, 线程的不断执行可以保证对 RFM 相关位置数据的不断刷新, 从而使得无论是 PCS 自身还是其它系统总是有数据可用, 从而提高了整个系统的效率和可靠性.

1.2 系统问题

PCS 系统的数据流为从反射内存卡中读取采集数据及 PEFIT(Parallelized Equilibrium FITing code)数据, 在各相关算法处理后控制命令输出, 这一时间长度是 PCS 系统为整个托卡马克控制系统带来的时间延迟. 根据 PCS 内部数据流的流向, 可以判断这一延迟的来源主要如下公式 1 所示.

PCS 延迟=传输时长(*CPU*1 发送给 *CPU*4)

+ 传输时长(*CPU*4 发送到 *RFM*)+ 物理延迟　(1)

基于此公式, 进行 PCS 延迟的理论及实际延迟测试. 理论及实际的延迟测试区别在于理论延迟仅计算数据流的传入传出时间, 将所有数据传输的时间加起来就是理论延迟; 实际延迟为直接计算数据的输入端和数据的输出端二者的时间差, 这其中可能包含了由于程序不合理造成的等待及摇摆行为所形成的延迟.

1.1.1 PCS 理论延迟测试

理论延迟测试的测试方法为在代码中将 pcsrt1 上的 CPU1 发送到 pcsrt3 上的 CPU4 代码及 CPU4 发送命令代码前后添加 readticks64 函数, 在系统运行过程中计算数据传输前后的时间差, 由此根据公式(1)计算出 PCS 的理论延迟. 测试结果如下表(表 1)所示, 表格中的数字单位是 tick, 经过七圈测试取平均值可以计算出 PCS 延迟.

表 1　数据发送时间测试

数据发送方	发送时间/tick						
CPU1	2664	2880	3048	2556	2760	2916	2544
CPU4	37927	37912	37830	37965	37845	37732	38032

本系统中的时钟频率为 3201 ticks/μs, 将上表中的测试数据取平均值, 可以得出 PCS 理论延迟:

(2797.2 + 37873.4)/ 3201 = 12.70 μs;

经测试, PCS 的理论延迟应为 12.70 μs.

1.2.1 PCS 实际延迟测试

1) 硬件连接

如图 2 所示进行, PCS 从采集卡读取数据, 从 rt3 的 RFM 将数据写出, 通过 RFM 交换机实现与其他系统的数据交换. 测试中增加了一台安装了 RFM 卡的机器作为接收端, 将 PCS 与接收端通过两台交换机及一台 RFM 交换机相连, 再将示波器的 1 通道和 2 通道分别连接 PCS 采集卡和接收端 DA 输出. 发送端与接收端的 DA 输出时间相抵, 两通道同一波形之间的时间差就是 PCS 的实际延迟.

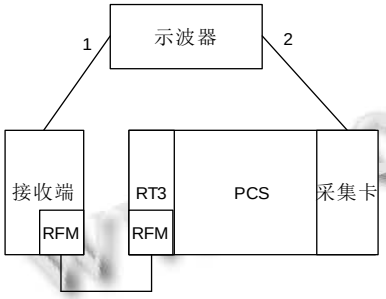

图 3 实际延迟测试连接方法(直连)图示

2) 测试结果

按照上文设置的方式进行测试, 测试五圈后的结果如表 2 所示.

表 2 延迟测试

延迟/μs	80.37	80.32	80.35	80.32	80.32

上图测试结果取平均值后, 可知延迟时间为 80.32μs, 这一测试结果远大于理论测试值. 经计算, 偏差为

(80.32 – 12.70) / 12.70 * 100% = 532.44%

即 PCS 的延迟时间超出理论延迟 532.44%.

1.3 问题原因分析及改进目的

1.3.1 原因分析

根据 PCS 内部运行原理分析, 判断 PCS 的多余延迟来自于 CPU1 和 CPU4 的任务序列不协调, 导致 CPU4 总是读上一个时钟周期的数据, 从而导致同一组数据的发送与接收行为之间多出了不必要的半个多周期.

然而, 两个 CPU 的任务序列不协调主要是因为二者的任务量差别太大, CPU1 任务繁重, 当 CPU1 发送数据时, CPU4 早已接收数据失败返回了, 从而错过了这一轮的数据接收任务, 而等到下一轮 CPU4 读到的实际是上一轮的数据.

1.3.2 改进方法及目标

1) RFM 读取耗时

CPU1 需要从 RFM 上读取 dtacq 及 pefit 的数据, 前者 32 字节, 后者 176 字节. RFM 在用 DMA 方式传输数据时的传输速率如表 3 所示.

表 3 RFM 的 DMA 传输速率

Bytes	Read MBps
32	3.5
128	3.8
256	3.9

根据表 3 中 RFM 的速率来计算, 二者读取数据所耗时间为

32 / 3.5 + 176 / 3.8 = 53μs

对于 100μs 的时钟周期而言, 读取 RFM 这一项工作就占用了超过一半的时间, 因此考虑对这一工作进行优化.

同时可以注意到的是, 当 CPU1 不再需要这 53μs 后, 根据其他 CPU 的运行负荷来观察, PCS 的控制周期可以减少到 50μs, 本文将之确定为目标控制周期.

2) 改进思路

改进思路为减少 CPU1 的任务量及 CPU1 在读取数据时的等待时间, 从而降低 CPU1 的负荷, 协调 CPU1 和 CPU4 的数据收发时间, 降低由于不必要的等待带来的时间延迟. 从 CPU1 中移出的工作由线程来承担, 另选空闲 CPU 来执行之. 线程将 CPU1 上负荷较大的读取 RFM 的工作封装起来, 线程不停地读数据, 保证 CPU4 在需要数据时, 一定有数据已经写好了供其使用.

3) 改进目标

根据以上分析, 可以得出以下三点目标:

a) 控制周期降低为 50 微秒;

b) 利用多线程实现 RFM 读取工作;

c) 保证 CPU1 可以访问到线程读到的数据.

2 方法设计及实现

2.1 数据流设计

数据流设计如下图(图 4)所示. 从图中可以看出, 数组 rfm_dtacq 是线程 thread_dtacq 和采集进程的共享变量, rfm_cperrors 是线程 thread_pefit 与 pefit 进程的共享变量, 线程不断地从 RFM 当中相应的偏移位置读取这两个进程所需要的数据, 并将得到的数据写进对应

的共享变量当中, 相应的需要数据的进程可以直接使用共享变量中的数据.

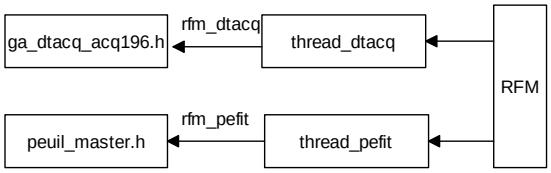

图 4　数据流设计

采用两个线程并行读取 RFM 数据的原因在于, 如前文计算所得二者读取数据用时为 53 微秒, 而目标控制周期为 50 微秒, 因此若二者串行读取则必然超时, 因此这里选择用两个线程来分别进行 RFM 的读取. 根据表 3 的计算, pefit 数据读取用时为 44 微秒, 因此方法可行.

这一改进可以保证采集进程和 pefit 进程在需要用到 RFM 当中的数据时相应的数据就已经读到并存在对应的数组中了, 而不用像之前的方案中那样等待 RFM 将数据读取完成才能使用. 同时, 由于线程在程序一开始就不断地对这两个数组进行刷新, 因此相应的进程获得的都是最新的数据, 不会出现读到脏数据的情况.

2.2 程序流程设计

程序流程设计分三部分进行, 分别为线程的创建和销毁, 线程刷新共享变量, 算法使用共享变量, 具体设计细节如图 5(以 dtacq 数据为例)所示.

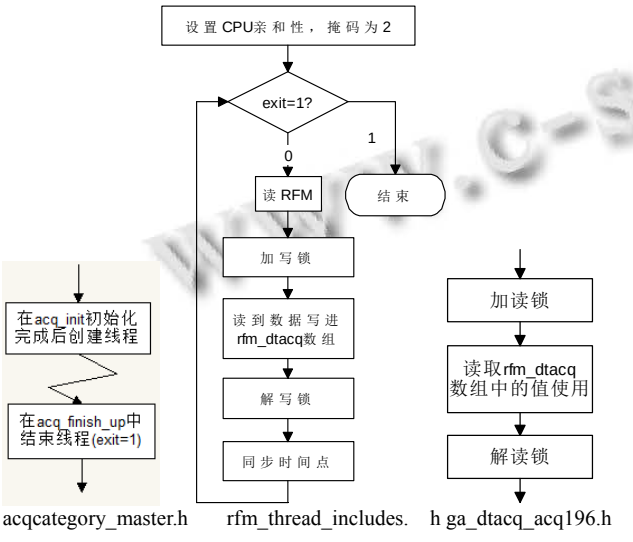

图 5　程序流程设计

如图 5 所示, 线程在 acq 采集进程初始化时创建, 在其结束时销毁. 这样设计的原因在于: 其一, pefit 数据大概在 0.3 秒后才会有, pefit 进程需要用到从 RFM 当中读出的数据的时间点较采集数据靠后, 因此线程的创建以采集进程为准; 其二, acq 采集进程会在整个放炮结束后才会结束, 因此在此处对线程进行销毁, 可以保证有效数据已经被读取且不阻塞主进程. 在 acq 进程结束时, 通过将共享变量 exit 设为 1, 可以使得线程停止循环.

线程的工作主要分为两部分, 一是读取 RFM 当中的数据, 二是同步时间. 在读取 RFM 数据后, 需要将读到的数据存储到相对应的共享数组(即图中的 rfm_dtacq 数组)当中, 而由于共享数组采集进程也会访问, 为防止读写冲突, 需要在这一过程前后加锁形成临界区. 临界区是用于防止多个线程同时执行一段特定代码的机制, 常用读写锁对临界区进行保护, 保证每次只能有一个线程执行临界区内的代码[5]. 读写锁又称为共享锁(shared-exclusive lock) , 将线程操作数据的目的分为读写两类[6]. 因此, 可以在线程中对共享变量进行填充前加写锁, 之后解锁; 而在进程需要读取共享变量前加读锁, 读完后解锁, 通过这种方式就可以防止共享变量的访问冲突.

线程工作的第二部分内容是同步时间, 其意义在于, 经过测试, 采集数据的获取是固定的时间点的, 如果线程只是无休止地读取数据而不与 CPU1 的时间相配合, CPU1 会由于采集数据获取超时而终止进程. 因此, 在线程中, 需要在读取 RFM 数据后, 计算下一个有效的时间点, 再进行数据的读取.

而作为使用共享数组中数据的进程, 数据的获取过程对其是透明的, 只需要在需要使用数据时直接使用就可以了, 好像这些数据本就存在与本地一样.

2.3 CPU 亲和性设计

一般认为, 若不规定线程的运行位置, 则线程是与创建其的进程运行在一个 CPU 上的, 因此二者的并行也只是理论上的并行, 即宏观并行而微观串行, 二者通过切换时间片的方式共同运行. 然而, 此法使用线程的目的是需要减少主进程所使用的时间, 因此, 必须将线程迁移到其他的 CPU 上, 才能够实现这一点. 由于需要两个线程并行, 因此需要在 rt1(即 CPU1 所在的实时计算机)上多申请两个 CPU, 分别运行两个线程, 每个线程负责读一个数据.

CPU 亲和性指的是程序开发者可以将一个或者多

个线程绑定到一个特定的处理器上执行的技术[7], 在多核单处理器环境下, 就是将某个线程绑定到某个 CPU 核心上执行. Linux 内核进程调度器天生就具有被称为软 CPU 亲和性(affinity)的特性, 这意味着进程通常不会在处理器之间频繁迁移[8]. Linux 从 kernel 2.5 开始提供了具有 CPU 硬亲和力功能的系统调用[9], 因此我们可以通过设置 CPU 亲和性的方式来两个线程分别绑定到两个 CPU 上执行.

设置 CPU 亲和性可以通过调用 linux 提供的 set_cpu_affinity 函数来实现, 确定线程运行 CPU 的方法为设置该线程的 CPU 掩码. 对于一台机器, 第 1 个 CPU 的掩码为 1, 其后第 i 个 CPU 的掩码为 1 左移(i-1)位, 读取采集数据的线程 thread_dtacq 固定在 pcsrt1 机器的第 2 个 CPU 上, 因此设置掩码为 2(二进制 10), 如图 5 所示; 而读取 pefit 数据的线程 thread_pefit 固定在第 3 个 CPU 上, 因此掩码设置为 4(二进制 100).

3 改进结果

将控制周期改为 50μs, 对改进后的 PCS 版本进行测试, 测试方法同上文, 测试结果如表 4 所示.

表 4 数据发送时间测试

延迟/μs	13.13	13.15	12.98	13.12	13.12

上图测试结果取平均值后, 可知此时的延迟时间仅为 13.12μs, 相对理论值仅偏差:

(13.12– 12.70) / 12.70 * 100%= 3.30%

即改进后 PCS 到快控的延迟仅比理论值超出 3.30%, 比原有延迟降低了:

(80.32 – 13.12) / 80.32 *100%= 83.67%

从这一测试结果可以看出, 改进后的 PCS 的延迟降低了 83.67%, 这大大缩短了 PCS 数据流的占用时间, 极大地降低了 PCS 延迟.

4 结语

本文介绍了 PCS 系统所具有的高延迟问题, 对这一问题进行了理论及实际情况的测试, 并提出了产生这一问题的可能原因为 CPU 配合失当造成的数据读取延迟. 与此同时, 提出了使用多线程并行的方式将较为耗时的 RFM 读数据操作与主进程并行执行的方法.

在实现过程中, 线程与进程的通信通过共享变量来实现, 利用读写锁保证共享变量的读写互斥性, 通过设置 CPU 亲和性保证线程运行的稳定性. 最终, PCS 的执行周期从 100μs 降低到 50μs, CPU1 也及时地获得了所需数据, 而延迟时间从 80μs 降低到 15μs 以下, 降低了 83.67%, 实现了所有的改进目标.

参考文献

1 夏金瑶,肖炳甲,杨飞.实时交互式虚拟 EAST 系统.测控技术, 2014,33(2):116–119.

2 Reinders J. Programming For Parallelism. 2007. http://www.cajcd.edu.cn/pub/wml.txt/980810-2.html.

3 袁旗平.基于 Linux 集群架构的等离子体控制系统[学位论文].合肥:中国科学院合肥物质科学研究院,2009.

4 Love R. CPUaffinity. 2003. http://www.linuxjournal.com/article/6799.

5 王蕾,崔慧敏,陈莉,冯晓兵.任务并行编程模型研究与进展.软件学报,2013,24(1):77–90.

6 孙建杰,陈佳品.临界区读写锁的实现.计算机与现代化,2011,(9): 193–197.

7 施惠丰,袁道华.基于多核的多线程程序优化研究.计算机技术与发展,2010,20(6):70–73.

8 杨伟.基于多核的入侵防御系统的研究[硕士学位论文].成都:电子科技大学,2009.

9 Bovet DP, Cesati M. Understanding the Linux Kernel.北京:中国电力出版社,2007.