A Effective Algorithm of Hardware Synchronization on Network Processor

LI Kang[1], MA Pei-Jun[2], SHI Jiang-Yi[3]

(The Defence Key Lab. Of Wide Band-Gap Semiconductor Materials and Devices,Xidian University, Xi'an 710071, China)

1.lik@mail.xidian.edu.cn，2.pjma@xidian.edu.cn，3.jyshi@mail.xidian.edu.cn

Abstract: The performance of multithreaded network processor can be lowered when its overhead of synchronization increase. So conventional "busy-wait" synchronization algorithm restricted severely the parallel processing capability. A blocking synchronization algorithm of multithreading was proposed for the Multi-Processor System on Chip(MPSoC) , which was used to actively schedule exclusive access by the synchronous arbiter on order of first in first out (FIFO). The algorithm was implemened with hardware structure which consisted of the content addressing based lock unit and synchronous request buffer. The implementation of the algorithm is able to eliminate the overhead of memory access in "busy-wait" synchronization and reduce occupancy of on-chip bus bandwidth and processores resource. Comparing with the other algorithm used in the application of packet processing, the algorithm proposed improved about 54% of the performance of the target system with reducing 29% of system power dissipation, which proved effective improvement for performance of packet processing and power dissipation.

Keywords: Computer architecture, Network processor, Hardware Synchronization,Multithreading

网络处理器的高效硬件同步算法

李康[1]，马佩军[2]，史江义[3]

（西安电子科技大学 宽禁带半导体材料与器件国防重点实验室，陕西 西安 710071）

1.lik@mail.xidian.edu.cn，2.pjma@xidian.edu.cn，3.jyshi@mail.xidian.edu.cn

【摘要】多线程网络处理器性能会随着同步开销的增加而降低，因此传统的“忙-等待”方式会对其并行处理能力造成严重制约。本文提出一个多核片上系统的阻塞式多线程同步算法，由同步仲裁器按照先入先出（FIFO）规则对互斥访问进行主动调度。算法实现基于内容寻址的锁单元和同步请求缓冲硬件结构，能避免传统同步方式的内存访问开销，且降低对片上总线带宽和处理器的占用。在包转发应用中的对比结果表明，该结构使系统吞吐率提高 54%，功耗降低 29%，有效改善了系统包处理性能和功耗。

【关键词】计算机体系结构, 网络处理器, 硬件同步, 多线程

1 引言

随着网络处理需求按照超摩尔定律进行增长，网络设备的处理能力始终面临着巨大的挑战，同时对网络设备低功耗需求也呈现增长趋势。异构多处理器片上系统（Heterogeneous Multi-Processors System on Chip）能够同时提供高性能并行处理能力和更低的系统功耗，因此成为高性能网络处理器结构的主要研究方向[1-5] 。多线程技术能够有效隐藏存储器访问延时，提高处理器利用率，但是多线程处理器性能仍然受到共享数据访问效率的影响[6]。传统的“忙-等待”同步适合于单进程处理器应用，自旋锁是最普遍使用的实现方式之一。有许多基于自旋同步机制优化算法被提出[7] ，如在 test-and-set 原语实现的简单自旋锁基础上添加锁探测延迟，采用类似于网络应用中的载波侦听多路访问（CSMA）协议进行同步性能优化。另一种方式是 Ticket 锁，采用 fetch-and-increment 原语和共享计数器实现。但在处理器规模增大时，总的同步竞争次数会随着处理器数目平方线性增加[8] 。因此提出排队锁（Queuing Lock）方案来解决这一问题，即对每一个锁保存一个锁等待队列，处理器可在本地轮询锁状态，不必频繁的访问存储器，有效降低同步访问竞争开销。排队锁可以采用软件实现，也可以基于硬件原语实现。

HMPSoC 的结构设计要综合考虑高性能、低功耗

基金项目：国家自然科学基金资助项目(60506020)；陕西省科技厅自然科学基础研究计划资助项目（SJ08-ZT13）

Foundation Item：The National Natural Science Foundation of China（60506020）; Natural Science Foundation of Shanxi Province of China (SJ08-ZT13)

和低成本三方面的要求。Monchiero 等提出一个同步缓冲结构来支持自旋锁，通过片上存储控制器来调度完成锁的释放和获取[9] 。该方案能完成常数次的总线交易和获取锁时的内存访问次数，但是采用自旋锁方式限制了多线程处理器利用率的进一步提高。

为了避免“忙-等待”同步在多线程处理器结构中产生的较大开销，本文提出一种基于查找表结构的阻塞式硬件同步算法，该算法不仅能通过片上锁查找结构来消除同步过程中的内存访问请求，而且通过一个同步请求队列来保存未处理的请求，避免了不同线程之间的同步竞争。并且该算法采用一个同步仲裁器进行主动式同步过程调度，提高了多线程处理引擎利用率，并同时降低同步操作对片上总线带宽占用。

2 基于查找表与主动仲裁的同步算法

2.1 同步算法描述

本文提出阻塞式同步算法重点实现两个同步操作原语，即 Search_and_Set 和 Search_and_Clear 原语。这两个原语提供的 API 接口可直接被软件程序调用。如图 1 中所示。

Search_and_Set/Clear 原语主要使用基于片上内容可寻址的查找表(Lookup-Table)结构来加速队锁变量的操作。锁变量 L 用内容寻址单元进行保存，可进行单周期的查找与修改操作。在锁获取过程中，首先根据互斥访问请求在内容寻址单元内查找指定内存地址是否已被保存，如未保存，则将该内存地址存放到内容寻址单元中，然后访问指定的内存位置，即锁获取成功；如地址已存在，则当前锁获取操作失败，请求被缓存至同步请求队列 RQ 中，等待下次锁获取操作。当内容寻址单元地址已满时，应将当前请求按照未决请求处理，保存到同步请求队列。该队列深度 D 由处理器个数 P 与每个处理器包含线程数 C 来决定。队列缓冲的最大深度应为 D = P×C。未决请求队列应该包含(L, Cmd, T)三个字段，即指定的加锁地址 L、内存访问命令 Cmd 和等待线程号 T。

在锁释放过程中，一旦有解锁请求到达，则首先从内容寻址单元中释放指定地址，再从缓冲队列取出等待的同步请求进行连续的锁获取操作，直到锁获取操作失败，再次被保存到队列中。缓冲队列能按照 FIFO 规则对未成功同步请求进行缓存和处理。锁释放过程可根据锁获取操作结果来主动决定如何从同步请求缓冲中取出请求进行处理。同步完成后，由事件进行等待线程唤醒，不需要线程对锁变量进行轮询检查，因此避免同步对计算资源的占用。

```
Type Struct ReqQueue{
      /*  同步请求缓冲结构  */
       Reference ref[D];    //  未成功的同步请求
       Int head, tail;
       Int Queue_length;}

Intial(ReqQueue RQ){
       /*  初始化同步请求缓冲结构  */
       RQ.ref[0…D-1] = Empty;
       RQ.Queue_length = 0;
       RQ.tail = RQ.head = D-1;
       }
```

```
Acquire_Lock(lock * L, int Thread){
   if(Search_and_Set(L)){
        transmit_event_back( “lock acquired” , T);
   }else{/*  将未处理的同步请求入队  */
      RQ.rear= (RQ.rear +1)mod D;
       RQ.ref[(RQ.rear] = (L, Cmd, T );
   }}
 Release_Lock(lock * L, int Thread){
        Search_and_Clear(L);
        Lock  *  L next=  Del_Pending_queue(RQ);
       While( Search_and_Set(L nex) ){
            Lock* L next = Del_Pending_queue(RQ);   }
        Insert_Pending_queue(RQ);
}
```

图 1　基于查找表的阻塞式同步算法协议

2.2　同步算法硬件实现结构

基于阻塞方式的 LT 锁结构中，内容寻址单元用一个 N 入口内容寻址存储器(CAM)实现。CAM 单元进行片上锁变量的操作。同步请求缓冲采用硬件循环队列结构，对未决同步请求进行按序处理。由于线程发出一个同步请求后会被切换，当 HMPSoC 有 P 个处理器，每个处理器有 C 个线程时，缓冲队列深度设计为 P×C-1 即可。同步仲裁器主动调度锁获取与锁释放操作过程，在新请求与缓冲队列中未决请求之间进行仲裁。仲裁逻辑根据解锁命令的出现确定何时处理未决请求。线程发出的失败读锁请求按照 FIFO 顺序保存在一个循环队列中，该循环队列被所有线程共享。

当一个线程发出内存请求，经过命令解码与地址产生后形成存储器访问地址与访问命令。如为互斥操作请求，同步仲裁器首先在 CAM 单元中检查指定地址

978-1-935068-01-3 © 2009 SciRes.

是否已加锁。如果未锁，则继续访问内存地址，并将该地址保存在 CAM 中，并发送锁获取成功的信号；否则，就会拒绝内存访问，并将未完成的请求保存到同步请求缓冲中进行等待。当得到锁释放请求后，同步仲裁器会调度同步请求缓冲内的请求进行处理。

3 验证与分析

3.1 基于周期的仿真环境介绍

算法验证中使用自行研制的 XDNP 仿真框架对 HMPSoC 结构进行基于周期的性能和功耗模拟与评估。它由多个异构处理器和片上并行总线构成。在数据平面采用多个 XDPE 进行并行处理，充分利用网络应用中的包并行特性。XDPE 是 C++描述的多线程 RISC 处理器，具有线程可扩展、可配置能力，专门用于网络包流处理；在控制平面集成了一个 SimIt ARM 仿真器负责对异常与网络管理方面的处理[10] 。为了提供一个高吞吐率并且性能精确的片上总线结构，用 SystemC 设计开发了一个基于周期的并行总线功能模型，用于片上处理器模块和共享存储控制器及其它功能模块的互连。XDNP 中也开发了基于周期的 SRAM 仿真模型，可以提供周期精确的内存访问[11] 。XDNP 采用同步时钟设计，处理器、存储控制器和片上并行总线模块采用 200MHz 时钟工作，片外存储器工作为 100MHz 工作是时钟。XDNP 框架基于 NepSim 仿真器的功耗评价方案提供了一个系统功耗模型[12] ，能够进行各个单元的功耗评估。该功耗模型使用的是 0.25μm 的 CMOS 工艺参数。

在 XDNP 中实现了 4 种同步结构以便于性能对比：(1)基本 Spin 锁算法：将锁变量存放在片外存储器，通过 Test-and-Set 原语来加速同步操作。该原语实现在共享存储控制器中[6]，采用软件方式来实现自旋；(2)硬件排队锁 HQL（Hardware Queuing Lock）算法：由硬件排队机制来管理全部锁变量，用软件轮询的方式来实现锁获取。本实验中的硬件排队锁采用 Fetch_and_store 原语来实现，队列硬件及其调度在共享存储控制器中进行实现。(3) 同步缓冲 SB（Synchronous Buffer）锁算法：采用同步缓冲机制，硬件电路在共享存储控制器中实现。(4)采用基于阻塞方式的查表 LT（Lookup Table）锁结构的硬件实现。

仿真验证测试负载使用 NetBench 工作集的 route 程序[13] 。Route 是一个基于 RFC1812 协议实现的 IP 包转发应用。在多个 XDPE 并行工作时，包处理过程需要并行化处理。实现中需为每一个物理网络接口设计实现一个发送队列并通过队列描述字在 SRAM 中维护。

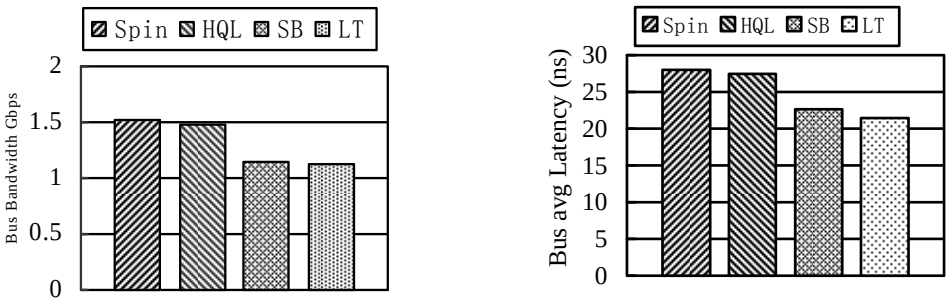

图 2 片上总线带宽占用于延迟情况 (a) 总线带宽占用(左图)； (b) 平均总线延迟（右图）

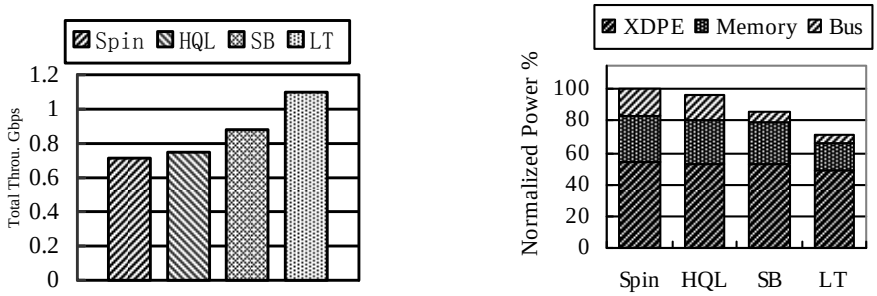

图 3 目标系统总体性能功耗对比 (a) 系统吞吐率对比（左图）； (b) 系统功耗对比（右图）

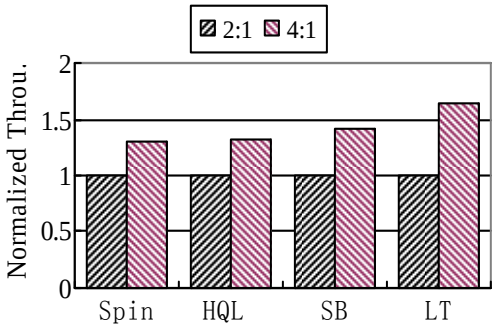

图 4 不同速度比系统吞吐率对比

3.2 总线带宽占用对比与分析

对 XDNP 运行时片上 SRAM 总线的带宽占用情况进行了统计，并在图 2 中对结果进行描述。图 2(a)将 4 种同步方式下 SRAM 总线带宽占用进行了对比。可以发现，采用 LT 锁时系统总线带宽为 1127.7Mbps，相对于 Spin 锁的 1511.1Mbps 下降大约 25%。这是由于 LT 锁中主动调度的同步处理方式避免了多个线程对锁变量的轮询操作，因此同步操作所占用的总线交易显著减少。由于多线程处理器不使用数据 Cache，使得 HQL 锁不能在本地对锁变量进行轮询，因此总线性能改善不明显；而 SB 锁采用了类似的同步缓冲队列来缓冲未处理的同步请求，总线占用为 1150.3Mbps，接近于 LT 锁的性能。图 2(b)中对比了总线平均延迟的变化情况。总线延迟定义为总线交易从源端发出到目的端接收到的平均延迟时间。可以看出，系统总线平均延迟也从 Spin 锁的平均 28ns 下降到 LT 锁的 21.5ns，下降了大约 23%，有较显著的改善。

3.3 对系统总体性能和功耗结果的评估分析

在图 3 中对比了路由转发基准程序在 4 种同步结构应用下 XDNP 的吞吐率与功耗情况。图 3(a)为 4 组吞吐率对比数据。可以发现，LT 锁结构的系统总吞吐率为 1099Mbps，与 Spin 锁结构的 714.4Mbps 相比提高约 54%；HQL 结构为 747.3Mbps，提高了 25%；而 SB 结构为 879.2Mbps，提高了 47%。LT 锁呈现出的高吞吐率性能是由于它不仅能够避免同步过程中的片外 SRAM 访问开销，而且能够通过主动同步策略将同步过程的总线交易数目最小化。另一方面则在于阻塞同步方式避免了每个线程进行互斥访问时的轮询操作，提高了处理器的利用率。图 3(b)为不同结构对系统功耗的影响。针对每一种同步结构，分别对多线程处理单元、存储器访问以及片上总线功耗变化进行了对比分析。从图中其它三种结构的功耗相对于 Spin 锁结构进行的归一化处理结果可看出，LT 锁结构降低总的系统功耗约 29%。功耗的节省主要体现在总线与存储器访问上，对应到 Spin 锁结构的相应模块分别下降了 36%和 57%。XDPE 处理单元的功耗下降约 10%，主要体现在轮询代码的执行开销上。

3.4 不同速度比下的系统性能变化

通过对 XDNP 平台的实验验证，也初步分析了四种同步结构可扩展性。实验中通过将片上时钟增加一倍而 SRAM 的频率不变，对不同同步结构下 XDNP 系统的吞吐率性能改善情况进行对比，来观察不同结构的可扩展情况。图 4 中对 4 种同步结构的吞吐率结果相对于各自的 200MHz 工作情况进行归一化对比。可以看到，当处理器与存储器速度比为 4：1 时， LT 锁结构的吞吐率比 2：1 速度比时增加了 65%，而其它的同步结构只增加了 30%~41%，更加远离多核处理器的线性加速趋势。这是因为随着处理器速度的提高，同步操作中的存储器访问对高速包发送线程的影响就变得越来越大，已成为限制系统吞吐率提高的重要因素。改善系统性能不仅要进一步提高存储器的访问速度，也要考虑到系统同步操作性能的提高。结果表明 LT 锁结构具有较好的可扩展性，能够满足进一步性能扩展的需要。

4 结论

本文提出一个适用于多线程HMPSoC的阻塞式同步算法，算法实现基于片上 CAM 和同步未决请求缓冲硬件结构。从硬件仿真器性能与功耗分析可看出，该算法相对于传统的“忙-等待”同步方式在性能上有50%以上的改善，并且降低大约 30%的系统功耗。结果表明，将该结构应用在多线程网络处理器中能够提高网络数据包线速处理能力，同时也进一步降低了系统功耗水平。

致谢

感谢在本论文成果的研究过程进行各方面配合的

978-1-935068-01-3 © 2009 SciRes.

所有老师与同学。

References (参考文献)

[1] CHEN Guo-Liang, SUN Guang-Zhong, XU Yun, LU Min.Methodology of Research on Parallel Algorithms[J]. CHINESE JOURNAL OF COMPUTERS. 2008, 31(09): 1493-1502.
陈国良，孙广中，徐云. 并行算法研究方法学[J]. 计算机学报. 2008, 31(09): 1493-1502.

[2] LIU Liang, WANG Xue-jing, YE Fan, REN Jun-yan. Design of low power and high speed FFT/IFFT processor for UWB system[J]. Journal on Communications. 2008, 29(09): 40-45.
刘亮，王雪静，叶凡. 应用于超宽带系统中的低功耗、高速FFT/IFFT处理器设计[J]. 通信学报. 2008, 29(09): 40-45.

[3] ZHU Chao, XIE Ying-ke, WANG Jian-dong, ZHAO Zi-li, HAN Cheng-de. Hardware-accelerated real-time IP flow measurement method for multi-core architecture [J]. Journal on Communications. 2008, 29(12): 1-9.
祝超，谢应科，王建东. 多核架构下实时IP流测量的硬件加速方法[J]. 通信学报. 2008, 29(12): 1-9.

[4] Wolf W. The future of multiprocessor systems-on-chips[C]. San Diego, CA, United states: 2004.

[5] Yi K, Gaudiot J. Features of future network processor architectures[C]. Sofia, Bulgaria: IEEE Computer Society, 2006.

[6] Mudigonda J, Vin H M, Yavatkar R. Overcoming the memory wall in packet processing: hammers or ladders? [C]. Princeton, NJ, USA : ACM, 2005.

[7] Mellor-Crummey J M, Scott M L. Algorithms for scalable synchronization on shared-memory multiprocessors [J]. ACM Trans. Comput. Syst. 1991, 9 (1): 21-65.

[8] Patterson D A, Hennessy J L. Computer Architecture:a Quantitative Approach 3rd Edition[M]. CA:Morgan Kaufmann, 2003, 584-604.

[9] Monchiero M, Palermo G, Silvano C, et al. Efficient synchronization for embedded on-chip multiprocessors[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2006, 14(10): 1049-1062.

[10] Qin W, Malik S. Flexible and Formal Modeling of Microprocessors with Application to Retargetable Simulation [C]. IEEE Computer Society, 2003.

[11] Villa O, Schaumont P, Verbauwhede I, et al. Fast dynamic memory integration in co-simulation frameworks for multiprocessor system on-chip[C]. Munich, Germany: IEEE, 2005.

[12] Luo Y, Yang J, Bhuyan L N, et al. NePSim: A network processor simulator with a power evaluation framework[J]. IEEE Micro. 2004, 24(5): 34-44.

[13] Memik G, Mangione-Smith W H, Hu W. NetBench: A benchmarking suite for network processors[C]. San Jose, CA, United states: IEEE Computer Society, 2001.