基于三级存储阵列缓存高速数据包及性能分析*

王 鹏[1,2+], 伊 鹏[2], 金德鹏[1], 曾烈光[1]

[1](清华大学 电子工程系,北京 100084)

[2](解放军信息工程大学 电子工程系,河南 郑州 450002)

Buffering High-Speed Packets with Tri-Stage Memory Array and Its Performance Analysis

WANG Peng[1,2+], YI Peng[2], JIN De-Peng[1], ZENG Lie-Guang[1]

[1](Department of Electronic and Engineering, Tsinghua University, Beijing 100084, China)

[2](Department of Electronic and Engineering, Information Engineering University, Zhengzhou 450002, China)

+ Corresponding author: Phn: +86-10-62781409, E-mail: wang-p01@mails.tsinghua.edu.cn, http://www.tsinghua.edu.cn

Received 2003-12-26; Accepted 2005-01-04

Wang P, Yi P, Jin DP, Zeng LG. Buffering high-speed packets with tri-stage memory array and its performance analysis. *Journal of Software*, 2005,16(12):2181–2189. DOI: 10.1360/jos162181

Abstract: High-Performance routers and switches need large throughput packet buffers to hold packets. However, the technique of commercially available memories is limited and can hardly fulfill this high throughput packet buffers. As a result, the development of networks is restricted severely. This paper presents a tri-stage memory array architecture to solve the problem, which can accomplish the arbitrary high-speed packet buffer theoretically. It is proved that the critical queue first algorithm can be applied as the memory management algorithm to get zero delay scheduling as well as minimum scale system. Furthermore, the design of hardware implementation architecture of the tri-stage memory array system is provided finally.

Key words: tri-stage memory array; packet buffer; delay

摘 要: 高速网络设备一般需要大容量高速数据包存储器来缓存收到的数据包.但以目前的存储器工艺水平很难实现这样的存储器,从而限制了整个网络的发展.提出一种新型的三级存储阵列结构可以成功解决数据包存储器的容量和带宽问题,理论上可以实现任意高速数据包的缓存.使用"最关键队列优先"算法完成对三级存储阵列的管理,证明了使用该算法能够保证数据包的无时延调度输出,并且其所需的系统规模最小,同时推导出系统规模的上、下限.最后给出三级存储阵列的一种可实现方案,从而使该结构易于硬件实现.

关键词: 三级存储阵列;数据包存储;时延

中图法分类号: TP302 文献标识码: A

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2001AA121071 (国家高技术研究发展计划(863))

作者简介: 王鹏(1976－),男,天津人,博士生,主要研究领域为高速路由交换,片上系统;伊鹏(1977－),男,硕士生,主要研究领域为高速路由交换;金德鹏(1972－),男,博士,副教授,主要研究领域为数据传输;曾烈光(1947－),男,教授,博士生导师,主要研究领域为专用芯片设计.

在网络高速发展的今天,如何实现高速、大容量数据包缓存已经成为高性能网络设备设计的瓶颈[1,2].比如速率为 160Gbit/s 的路由器线卡,如果使用 64bit 位宽的存储器实现接收缓存,要求随机访问时间必须小于 0.4ns.而对于一个具有 N 个这样接口的共享存储结构路由器,则需要存储器的随机访问时间降到 0.4/N ns 以下,这样高的访问速度是目前商用存储器短期内无法实现的.另外从长远来看,目前存储器的带宽依照摩尔定律*每 18 个月增长 1 倍,而骨干网带宽每 6 个月翻一番,因此存储带宽的瓶颈效应将会越来越严重[2,3].

提高存储器访问速度的方法很多,比如在大型计算机中普遍采用预取[4]、交织[5]以及哈希表[6]等技术完成对存储器的高速访问.但是,由于网络设备的特殊性以及网络数据流的突发性,这些方法很难直接用于数据包缓存.一般的并行方法[7−9]也很难解决多队列调度和在给定时延范围内的存取.Stanford 大学的 Sundar Iyer 等人利用 DRAM 的大容量和 SRAM 高速的特点,提出混合结构(SRAM 与 DRAM)联合工作模式[2,9,10],在一定程度上缓解了目前数据包存储器的容量和访问速度问题,但是其致命缺陷在于该结构受限于 SRAM 的访问速度.即如果线路速率超过 SRAM 所能提供的带宽,这种 SRAM 与 DRAM 混合结构就会因 SRAM 的速度不够而彻底失去作用.

本文提出一种新型三级存储阵列结构,可应用于目前已知的各种存储形式的网络设备中.该结构基于目前常用的DRAM存储器,从理论上解决了任意高速数据包的缓存问题,克服了存储器带宽的瓶颈.为了配合这种存储结构,使用最关键队列优先算法[2,9,10]完成对三级存储阵列的管理更新.该算法可以保证对数据包无延时调度,并在本文中被证明能够使所需的三级存储阵列规模达到最小.另外,为使该结构易于硬件实现,文中给出了相应的可实现方案,简化了三级存储阵列的实现复杂度,使其具有较高的工程应用价值.

1 数据包存储器特点

骨干网络设备中使用的数据包存储器一般具有如下特点:

(1) 高带宽.当线路速率提高时,用于缓存这些高速数据包存储器的速率也必须相应提高,也就是说,其速率位宽积(存储器带宽)必须高于线路带宽.目前的 DRAM 的随机访问周期**最低为 30ns,而 SRAM 约为 3ns,很难满足 160Gbit/s 或者更高线路速率的要求.

(2) 大容量.有资料[11]显示,如果要保证路由器的性能(低时延和丢包率),其内部存储容量应不小于 RTT×Rbit(其中 RTT(round trip time)是环回时间,R 为线路速率).目前的 Internet 中 RTT 的典型值约为 200ms~300ms,对于 160Gbit/s 的线路卡,则需要大约 40Gbit 的缓存空间.目前商用 DRAM 最大可以做到 1Gbit/片,SRAM 可达到 16Mbit/片.由此可见,对于实现大容量缓存,DRAM 是首选的存储器类型.

(3) 使用支持多个 FIFO(先入先出)队列结构.目前的高速路由器和交换机中普遍采用虚拟输出排队(VOQ)技术[12],即要求在每个输入线路卡中为每路输出独立保持数据包 FIFO 队列.而如果采用基于业务流、优先级等排队规则[13],则要求同时维持的队列数会更多.有资料显示,在 OC192 链路速率下,存储器中队列数可以达到 256K 或更多.

(4) 访问的随机性.存储器中数据包的输出是由外部调度器执行调度算法来控制的[12],这个过程对于存储器本身无法预知.同时,网络设备通常要求从调度器发出调度命令到被调度的数据包输出必须在给定的时延范围内完成.

由以上特点可以看出,数据包存储与一般数据存储有一定的区别(尤其是(3)和(4)),所以一般的提高存储器带宽的方法不再适用于网络数据包存储.

---

* 摩尔定律由 Gordon Moore 于 1995 年提出,它原本用来描述集成电路规模(晶体管数)的增长速度为每两年翻一番,后来 Gordon Moore 本人发现(统计了 1975 年~1996 年的数据),这个倍增周期为每 18 个月更为准确.其后,人们同时发现在工业工程、经济等诸多领域,摩尔定律也广泛有效.本文引用摩尔定律来表示目前存储器速度容量的发展情况,类似的引用可在文献[2,9,10]中查到.

** 随机访问周期又称为随机访问时间,是指从一次读/写操作到下一次读/写操作所必须等待的时间.读/写操作本身所占用的时间极少,与随机访问周期相比可近似为 0.

2 三级存储阵列

三级存储阵列的逻辑结构如图 1 所示,包括输入、中间和输出 3 级.为了匹配端口速率,在该结构的输入级和输出级使用特殊的存储阵列结构来完成高速数据的串行写入和读出,对中间级则采用若干 DRAM 单元块并行来完成大容量存储和速率适配.输入级和输出级的阵列结构与中间级的并行数据交换由输入调度器和输出调度器执行存储管理更新算法分别控制.图 1 中的中间级同时保持着 Q 个逻辑数据包 FIFO 队列.

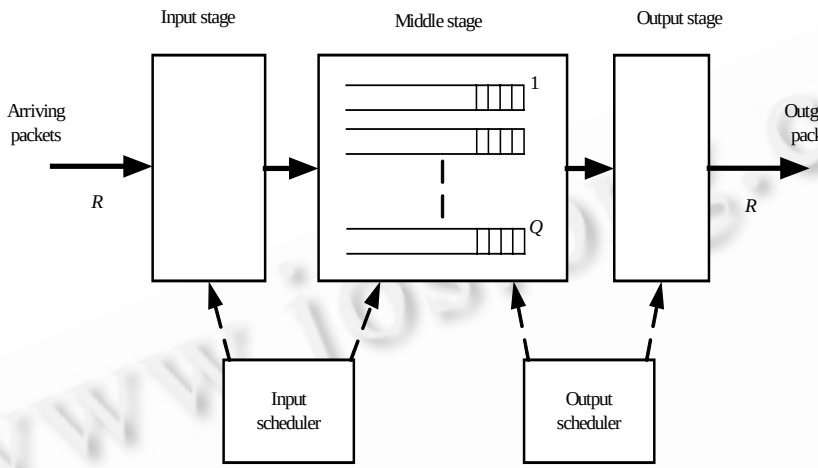

Fig.1 Tri-Stage memory array

图 1 三级存储阵列

下面具体介绍输入、中间和输出级的物理结构.值得注意的是,对三级存储阵列而言,输入级和输出级从某种意义上讲是对称一致的.比如对于输出级各队列调度的随机性表现在输入级则为各个队列的数据包到达的随机性.因此,仅需要分析某一级的访问特性和更新规则,另一级的分析可类似完成.本文后面的讨论主要围绕输出级展开.

为叙述方便,设定如下参数:

考虑线路速率为 R(比特/秒),对位宽为 B 比特、随机访问周期为 T 秒的 DRAM 存储器,需要至少 $b=\left\lceil \frac{R\times T}{B} \right\rceil$ 块分离的并行 DRAM 来完成对数据的缓存.

时隙(单位时间):定义时间长度 T/b 秒为一个时隙,也就是在 T 秒内对 b 个并行 DRAM 块中任一块所分配的访问时间.

信元(cell): $R\times T/b$ 比特,即当线路速率为 R(比特/秒)时,每个时隙能够写入或读出的数据量(为叙述方便,本文考虑每个调度命令针对一个信元做出,并且每次读写都以信元为单位.这样不影响对于一般变长数据包的调度分析,对于大的数据包可以考虑成多个信元的组合,其调度命令也可分解为多个).

存储单元:称容量和位宽都为一个信元的存储模块为一个存储单元.各个存储单元是物理上分离的存储器.

冲突:DRAM 的随机访问时间为 T 秒,则对一个 DRAM 块进行读写操作后必须经过 b 个时隙才能再次进行操作.如果对于同一块 DRAM 相邻的两次操作时间间隔小于 b 个时隙,则称这两个操作为冲突的.

使用 b 块 DRAM 存储单元阵列完成线路速率适配是这样完成的:由于单块 DRAM 存储单元以 T 为访问周期,如果使用 b 个这样的 DRAM 存储单元循环操作,并且每个存储单元的读或写操作仅仅占用一个访问周期中的某个固定时隙,每个时隙仅能对一块 DRAM 进行操作(这里假定两个相邻的 DRAM 块访问之间的切换时间非常小),则可实现无冲突访问,缓存线路速率为 R(比特/秒)的数据流.这一点不同于 Stanford 大学的 SRAM 与 DRAM 混合结构,本结构正是使用了并行存储阵列而解决了存储器带宽的瓶颈问题.

时延:从外部调度命令到其对应的数据信元得到输出的时间间隔.定义,如果被调度的信元已经存在于输出

级并可被输出(对其读取操作不会造成冲突),则该调度命令的信元时延为 0.

如图 2 所示为中间级和输出级的物理结构:中间级是由 b 个并行大容量 DRAM 完成带宽扩展

$$\left(B/T \to R\right),$$

即每隔 b 个时隙完成对 b 个 DRAM 的一次读或写操作.注意,被访问的 b 个存储单元各源于一块 DRAM,并且这 b 个存储单元在这 b 个 DRAM 中具有相同的地址,同属于某一个队列.如图 2 所示,当前时刻写入中间级的为第 k 个队列的 b 个信元:处于每块 DRAM 的第 1 个地址单元的信元同属于第 i 个队列.这样做有利于读写访问控制,否则容易出现在一个访问周期内对某个 DRAM 的连续写入或者读出,造成操作冲突.输出级使用一组分离的容量为一个存储单元的 DRAM 阵列来完成.每个时隙的读操作仅仅针对输出级的某个 DRAM 存储单元块,b 个时隙可以有 b 个信元的数据被读出,并且各个操作不会冲突,这样确保了对外的速率为 R(比特/秒).输出调度器保存各队列在中间级的地址信息,每 b 个时隙,它执行存储器管理更新算法选择某个队列的 b 个顺序信元,写入输出级中 b 个空闲的 DRAM 单元中.以上的操作同样可应用于输入级的设计中.这样从整体来看,三级存储阵列结构是速率 R(比特/秒)接收并发送数据包.

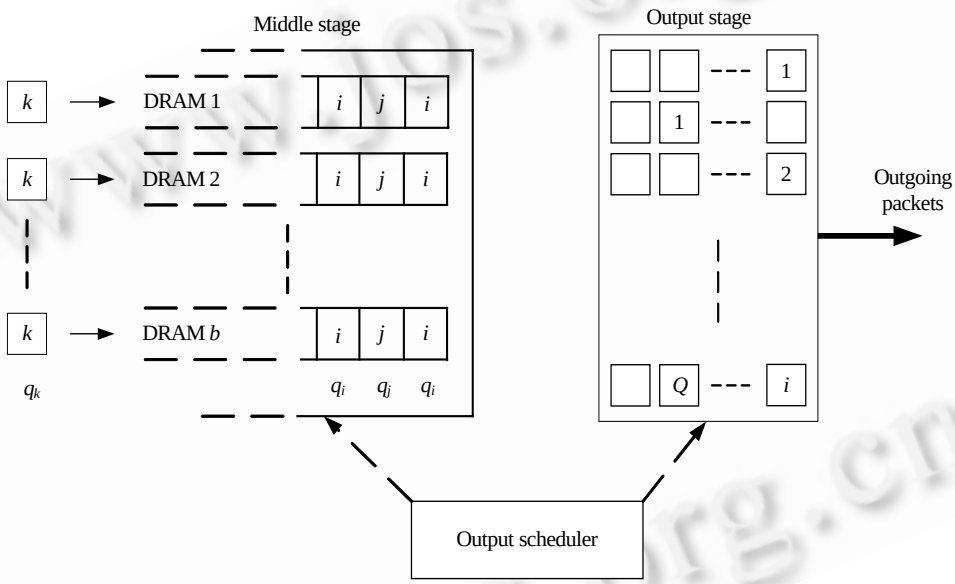

Fig.2 Middle stage and output stage

图 2 中间级和输出级

由以上介绍可以看出,三级存储阵列需要高效存储器管理更新算法来控制.尤其是对于多个队列的管理更新,很容易出现中间级来不及把相应队列的信元写入输出级,从而出现输出级被读空的情况(underrun)[2,9,10].因此需要一种对三级存储阵列的管理更新策略来完成各个队列的高效管理.

3 最关键队列优先算法

我们在介绍算法之前,先进行定义:

$O(i,t)$:表示在 t 时刻,系统输出级中队列 i 的信元个数;

$U(i,t)$:表示在 t 时刻,队列 i 未能被满足的调度命令数(即外部调度算法发出调度命令,但是在输出级没有相应的信元能够被输出);

在本文中假定三级存储阵列中包含 Q 个队列.

定理 **1**(必要性). 任意一种存储器管理算法,如果要保证在三级存储阵列中的信元能够被无延时调度(被调度的信元总是能够在输出级并且被输出),其输出级至少要包含 $Q(b-1)(2+\ln Q)+2b$ 个 DRAM 存储单元.

证明:考虑一个调度命令序列:系统在 1~Q 时隙分别对 1~Q 个队列发出一个调度命令,则在第 Q 个时隙结束时,输出级至多有 Q/b 个队列得到更新,而必然有至少 $Q(1-1/b)$个队列的 $U(i,Q)=1$,这样完成了第 1 轮调度.

从第 $Q+1$ 个时隙到 $Q(2-1/b)$时隙,系统对 $U(i,Q)=1$ 的各个队列发送第 2 轮调度命令,则至多会有 $Q(1-1/b)b$ 个队列得到更新,而至少有 $Q(1-1/b)^2$ 个队列的 $U(i,Q(2-1/b))=2$.

如此下去,则在第 x 轮调度时必然有 $Q(1-1/b)^x$ 个队列的 $U(i,Qb(1-(1-(1/b)^x)))=x$,求解 $Q(1-1/b)^x=1$,可得 $x=(\ln Q)/(\ln(b/b-1))$.由于 $\ln(1+x)<x$,则 $\ln(b/(b-1))<1/(b-1)$,从而 $x>(b-1)\ln Q$.这样得出,如果每个队列在输出级具有$(b-1)\ln Q$ 个存储单元,那么在系统完成$(b-1)\ln Q$ 轮调度命令后,必有一个队列的 $U(i,t)$达到$(b-1)\ln Q$.由此可见,为了避免该队列输出级被读空的情况,需要每个队列在输出级保持$(b-1)(1+\ln Q)$个存储单元.

考虑如果在前 Q 个时隙,系统依次对 Q 个队列发出调度命令然后停止,那么此时的输出级的每个队列都包含$(b-1)(1+\ln Q)-1$ 个信元,但是由于每次更新是从中间级的某个队列一次读取 b 个信元,然后写入输出级中,所以需要在输出级的每个队列能够保存$(b-1)(2+\ln Q)$个信元.另外,考虑 DRAM 的操作周期为 $2b$,所以需要对输出级额外增加 $2b$ 个 DRAM 存储单元.所以整个输出级至少需要包括 $Q(b-1)(2+\ln Q)+2b$ 个 DRAM 存储单元. □

最关键队列优先算法:

当满足如下两个条件时,每 b 个时隙最关键队列优先算法总是选择 $U(i,t)$最大的队列进行更新(当有若干个队列的 $U(i,t)$相同时,从中随机选择一个进行更新).

条件 1:在输入级或者中间级存在该队列的 b 个信元;

条件 2:在输出级有 b 个空闲的 DRAM 存储单元能够在该时刻写入.

下面推导使用该算法的三级存储阵列输出级所需要的 DRAM 存储单元数,假定 $v(t)=(v_1,v_2,\ldots,v_Q)$代表从 1~$Q$ 个队列按照 $U(i,t)$在 t 时刻的值从大到小顺序排列.取 $f(i,t)=\sum_{k=1}^{i}v(i,t)$.定义 $F(i)$为对所有的调度命令形式,任意更新时刻,信元在无延时调度条件下系统的 $f(i,t)$所能达到的最大值.即 $F(i)=\max\{\forall t\in\tau,f(i,t)\}$($\tau$为更新时刻的集合).

引理 **1**. 三级存储阵列使用最关键队列优先算法,则 $F(Q)\le Q(b-1)+b$.

证明:为了证明该引理,考虑如下事实:当算法选择 $U(i,t)$最大的队列进行更新时,若 $U(i,t)<b$ 时,系统中 $\sum_{i=1}^{Q}U(i,t)$ 可以保持增大,如果系统中的某些队列的 $U(i,t)=b$,则系统在最关键队列优先算法条件下 $\sum_{i=1}^{Q}U(i,t)$ 不可能再增大,否则更新的不是 $U(i,t)$最大的队列.考虑最初系统中各个队列的 $U(i,t)=0$,随着时间和外部请求的增加,$\sum_{i=1}^{Q}U(i,t)$ 逐渐增大(并不一定每次更新都增大).设 t 时刻系统中首次有队列 $U(i,t)=b$,并且同时达到 $U(i,t)=b$ 的队列至多有 b 个,其他队列的 $U(i,t)$至多达到 $b-1$,则此时系统的 $\sum_{i=1}^{Q}U(i,t)$ 应该达到 $F(Q)$.所以有 $F(Q)\le Q(b-1)+b$ 成立. □

引理 **2**. 三级存储阵列使用最关键队列优先算法,则 $F(1)<\left(\frac{2Q+1}{Q}+\ln(Q-1)\right)-1$.

证明:系统使用关键队列优先算法,假设在时刻 t 有某个队列 i 第 1 次达到 $F(1)$,容易知道,$U(i,t-b)\ge F(1)-b$.(如果上述条件不成立,则必然出现 $U(i,t)<F(1)$或此时不是系统第 1 次达到 $F(1)$).因此,必然有某个队列 j 在 $t-b$ 时刻得到服务,即

$$U(j,t-b)\ge U(i,t-b)\ge F(1)-b \tag{1}$$

所以有

$$F(2)\ge F(2,t-b)=U(i,t-b)+U(j,t-b) \tag{2}$$

从而推出

$$F(2)\ge F(1)-b+F(1)-b \tag{3}$$

同理可以推出

$$F(3)\geq F(2)-b+\frac{F(2)-b}{2} \tag{4}$$

$$F(i+1)\geq F(i)-b+\frac{F(i)-b}{i} \tag{5}$$

对式(5)两端从 i=0 到 Q–1 求和,并利用引理 1 有

$$F(1)\leq (b-1)+b/Q+b\sum_{i=1}^{Q-1}\frac{1}{i} \tag{6}$$

由于

$$\forall N>1,\sum_{i=1}^{N-1}\frac{1}{i}<1+\ln(N-1) \tag{7}$$

代入式(6)可得

$$F(1)<b\left(\frac{2Q+1}{Q}+\ln(Q-1)\right)-1. \quad \square$$

引理 **3**. 三级存储阵列使用最关键队列优先算法,则

$$F(i)\leq i\left(2b-1+b/Q+b\ln\frac{Q-1}{i-1}\right),2\leq i\leq Q-1.$$

证明:通过对式(5)两边从 i 到 Q–1 求和,可以得出

$$F(i)\leq i\left(b-1+b/Q+b\sum_{k=i}^{Q-1}\frac{1}{k}\right) \tag{8}$$

由

$$\sum_{j=i}^{Q-1}\frac{1}{j}=\sum_{j=i}^{Q-1}\frac{1}{j}-\sum_{j=i}^{i-1}\frac{1}{j},\forall i\in\{2,...,Q-1\} \tag{9}$$

同样,由式(7)和 $\sum_{j=1}^{i-1}\frac{1}{j}>\ln(i-1)$ 可得

$$F(i)\leq i\left(2b-1+b/Q+b\ln\frac{Q-1}{i-1}\right),\forall i\in\{2,3,...,Q-1\}. \quad \square$$

定理 **2**(充分性). 对于使用最关键队列优先算法的三级存储阵列,如果要在任意时刻,输出级都保持有外部调度命令所对应的可输出的数据信元,则在整个输出级需要保证 $b(3Q+1+Q\ln(Q-1))-2Q$ 块 DRAM 存储单元,而在整个输入级需要 $b(2Q+1+Q\ln(Q-1))-Q$ 块 DRAM 存储单元.

证明:由引理 2 得出,如果要保证系统输出级始终保持有被调度的数据包可以输出,则要求系统输出级至少保持 $b(2Q+1+Q\ln(Q-1))-Q$ 个 DRAM 存储单元,即为每个队列保持 $b\left(\frac{2Q+1}{Q}+\ln(Q-1)\right)-1$ 个存储单元.但是,如果某个队列 i 的 $U(i,t)$在 t 时刻达到 $F(1)$,那么到下一次更新时刻 $t+b$ 之前至多会有 b–1 个信元被输出.因此,对于三级存储阵列输出级的每个队列至少具有 $b\left(\frac{3Q+1}{Q}+\ln(Q-1)\right)-2$ 个存储单元,所以整个输入级需要保持 $b(3Q+1+Q\ln(Q-1))-2Q$ 块存储单元.而对于输入级 $F(1)$则代表在输入级一个队列所能达到的最大 $U(i,t)$,所以输入级需要 $b(2Q+1+Q\ln(Q-1))-Q$ 就可以完成无延时调度. $\square$

定理 **3**(最优性). 三级存储阵列使用最关键队列优先算法能够使系统 $F(Q)$最小.

证明(反证):设某算法α能够保证系统 $\sum_{i=1}^{Q}U_\alpha(i,t)$ 在任意时刻均小于 $F(Q)$.设 $F_\alpha(Q)=\max\sum_{i=1}^{Q}U_\alpha(i,t)$,而在时刻 t 是系统使用α算法首次达到 $F_\alpha(Q)$.假定系统从 0 时刻~t 时刻顺序对$(a_1,a_2,\ldots,a_k)$这 k 个队列进行更新操作.由于α算法不是最关键队列优先算法,因此必然会在某个更新时刻或者某些更新时刻对更新队列的选取与最关键队列优先算法所选取的队列不同,即不选择 Q 个队列中 $U_\alpha(i,t)$最大的进行更新操作.假定开始两种算法的更新选择相同,设定α算法与最关键队列优先算法选择被更新队列第 1 次出现不同的时刻为τ,则由最关键队列更

新算法的定义可知,此时必然有 $\sum_{i=1}^{Q}U(i,\tau)\le\sum_{i=1}^{Q}U_\alpha(i,\tau)$.由于每次更新最关键队列优先算法所使用的原则不变,所以不会在接下来的某个更新时刻有 $\sum_{i=1}^{Q}U(i,\tau)>\sum_{i=1}^{Q}U_\alpha(i,\tau)$ 的情况出现.因此,必然有 $F_\alpha\ge F(Q)$,得出矛盾.因此有最关键队列优先算法能够使系统 $F(Q)$达到最小.

由于系统 $F(Q)$在最关键队列优先算法下是最小的,所以对三级存储器的输出级所需要的 DRAM 存储单元数也最小. □

注:本文中的部分分析证明思路借用了文献[10]中的思路,但是结论有所不同.本文所提出的三级存储阵列结构与文献[10]有本质区别,从根本上克服了文献[10]中 SRAM 访问速度的瓶颈问题.另外,在文献[10]的讨论中没有考虑 SRAM 的访问周期问题(该文献中,作者认为 SRAM 的访问周期为 0),这样其分析欠妥,本文提出的结构和分析弥补了这一漏洞.

4 硬件实现考虑

上述设计虽然可以理论上完成任意高速数据包缓存,但考虑到硬件实现的复杂度,在实现三级存储阵列的输入和输出级时需要对其硬件可实现性作出估计.考虑如果线路速率为 160Gbit/s,使用位宽为 256bit 随机访问时间为 30ns 的 DRAM 来完成以上三级存储器阵列结构,系统输出级需要 $b(3Q+1+Q\ln(Q-1))-2Q$ 个相互分离的 DRAM 存储单元.这里取 Q=256k(该值为 OC192 链路的典型值,实际中可能会更大),则实现该系统输出级大约需要 7.5×107 块 DRAM 存储单元.这给硬件实现(布局、布线、功耗、成本等)带来很大问题,依据目前的硬件水平很难实现.因此有必要简化该结构,使其更具有工程应用价值.本节讨论三级存储阵列的简化硬件实现方案.

4.1 使用DRAM结构

这里的改进设计主要是考虑能否有一种实现形式,使系统输出级所使用的分离的 DRAM 块数与 Q 无关.如果系统输出级的规模(主要是使用的分离存储器块数)不随三级存储器阵列中保持的队列数增大而增大,则系统实现方案将会大大简化.考虑如图 3 所示形式的系统输出级.

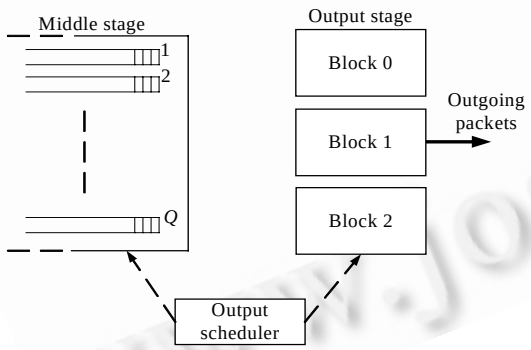

Fig.3 Revised output stage architecture

图 3 系统输出级改进设计结构

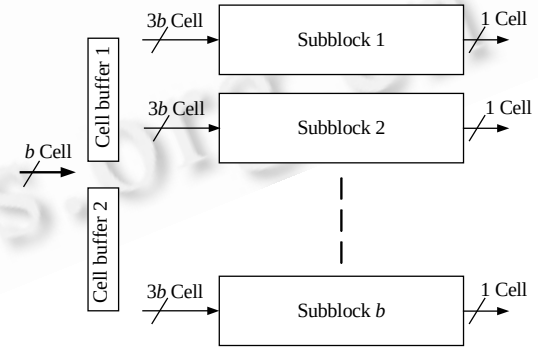

Fig.4 Block architecture in revised output stage

图 4 输出级中单块设计结构

输出级在逻辑上使用 3 个块,如图 4 所示,每块由 2 个信元暂存器和 b 个子块构成.在每块的输入端,每 b 个时隙到达 b 个信元的数据.两个信元暂存器由 $2b$ 个分离的 DRAM 存储单元块组成,可用来缓存两次更新写入的 $2b$ 个数据信元.每个子块则使用 $3b$ 个容量为 $\left\lceil b(3Q+1+Q\ln(Q-1))-2Q\big/3b\right\rceil$ 个存储单元的 DRAM 构成.这样对每一块来说,可以每 $3b$ 个时隙完成一次更新,每次更新将前 $2b$ 个时隙保存在两个信元暂存器的 $2b$ 个数据信元与

本时隙到达的 b 个数据信元同时写入该块内的 b 个子块中,并且这个操作在一个时隙完成***.在对该块进行读操作时,由输出调度器控制依次从该块的第 1 子块开始,每个时隙顺序从一个子块中按照请求读出一个信元数据.对每一块的读写操作时序如下:从时刻 0~时刻 b–1,读第 1 块,同时对第 2 块在时刻 0 进行更新:从时刻 b~时刻 2b–1,读第 2 块,同时对第 3 块在时刻 b 进行更新:从时刻 2b~时刻 3b–1,读第 3 块,同时对第 1 块在时刻 2b 进行更新,以此类推,即从时隙 ib 到时隙(i+1)b–1,系统读取第 i mod 3 块,在 ib 时刻更新第(i+1) mod 3 块.由此可知每一块物理上分离的 DRAM 两次读或写操作之间都至少间隔 b 个时隙,不会造成冲突.

这里仍然使用最关键队列优先算法,只是每次更新计算时考虑 $Q(i,t)$为各组中未被读取队列 i 的信元总数(重复信元只算一次).由于每 3b 个时隙写 3b 个新的数据信元到每一块中,所以可以保证每一块 DRAM 的内容得到及时更新,因而对于读写操作和更新算法没有影响.使用本节开始的例子,需要(2b+3b^2)×3=3363 块 DRAM 来完成发送缓存.由此看出,在这样的改进设计中,独立 DRAM 的块数与 Q 无关,从而可以显著减少 DRAM 的块数降低硬件实现复杂度.

4.2 使用混合结构

一般来讲,DRAM 适于大容量低速存储,而 SRAM 适于较小容量高速存储.在输出级,如果使用比 DRAM 快一些的 SRAM 实现,则可进一步减少输出级中存储器的块数.

考虑在输出级使用的线宽为 B bit 的 SRAM,其随机访问时间为 Ts 秒(Ts<T),那么对于线路速率为 Rbit/s,则至少需要 $a=\left\lceil \frac{R\times Ts}{B} \right\rceil$ 块这样的 SRAM 并行工作就可以完成速率适配.即对于这样的 SRAM,每 a(a<b)个时隙可以进行一次读或写操作,两次相邻操作间隔不能少于 a 个时隙,否则会出现操作冲突.

与上一节同理,使用图 3,图 4 中结构,只是在系统输出级的每个块此时由 a 个子块构成.每个子块由 3b 个容量为 $\left\lceil b(3Q+1+Q\ln(Q-1))-2Q \middle/ 3b \right\rceil$ 个存储单元的单块 SRAM 组成,即共需(2b+3ab)×3 块 SRAM.使用本节开始时的例子,取 SRAM 的随机访问时间为 3ns,则 a=2,输出级只需使用 456 块分离的 SRAM 块.注意,这里的更新时刻与使用 DRAM 时不完全一致,但是由于 a<b,所以更新时刻可以选择无操作冲突的时隙进行,这里不再赘述.

本文提出的结构同样适用于有界时延情况,具体分析参见文献[14].表 1 中显示了目前已知存储器结构的性能对比情况.可以看出,三级存储阵列虽然实现复杂度略高,但它具有高速数据包存储器所需要的所有特性,并且从理论上可以实现任意高速的数据包存取.在存储器带宽已经成为网络发展瓶颈的今天,这样适当增加复杂度来换取包存储带宽是有效的、值得的.

Table 1 Performance comparisons between currently available memory architectures

表 1 各种已知存储结构性能比较

Name	Zero-Delay packet access	Bounded-Delay packet access	Buffering speed surpasses the bandwidth of memory	Implementation scale and complexity (The number of memory blocks in need)	Support large volume buffering
Commonly used architectures in computers	Not support	Not support	Partly support	Medium	Partly support
PPB[3,8]	Support	Support	Support	$Q(Q)$	Support(but difficult)
Hybrid SRAM–DRAM architecture [9,10]	Support	Support ($Q(b-1)$+1 time slots)	Not support (limited by the throughput of SRAM)	$6(2b+3ab)+(R\times RTT)/V_{DRAM}$	Support
Tri-Stage Memory Array	Support	Support ($Q(b-1)+b$+1 time slots)	Support (can achieve arbitrary high-speed theoretically)	$(R\times RTT)/V_{DRAM}$ +1	Support

注:这里的比较是基于参考文献[3,8,9,10]作出的,对于一般计算机系统所使用的方法,由于种类比较多,而

*** 这里,每块 DRAM 的容量为信元×$\left\lceil b(3Q+1+Q\ln(Q-1))-2Q \middle/ 3b \right\rceil$ bit,这样每个子块的容量可以达到$\lceil b(3Q+1+Q\ln(Q-1))-2Q \rceil$个信元,由于其带宽扩展为 3$b$ 个信元,所以可以一次同时接受 3b 个信元.

且不是本文对比的重点,故没有给出解析参数.PPB 结构的实现复杂度除了与系统中保持的队列数 Q 成正比以外(这样使系统复杂度无法被有效控制),它要求各个存储信元尾部保持一个地址指针,用来指向该队列下一个信元的首地址,这样在系统调度命令发出后,可能需要在各个DRAM块中往返切换,给系统的地址管理带来更大的复杂度,并且这样的结构在高速硬件设计中很难实现.VDRAM 表示实现系统中间级时单块 DRAM 的容量, $R \times RTT / V_{DRAM}$ 表示系统中间级所需的 DRAM 块数.

对相关的算法和结构部分进行了硬件验证,使用 Xilinx FPGA xc2v3000(300 万)实现该算法,其占用率为56%.

5 结 论

为了解决高速数据包缓存问题,我们提出了新型的三级存储阵列结构,并且配合该结构使用最关键队列优先算法来完成对存储器的管理和定期更新.我们证明了该设计满足网络数据包存储器的要求,在理论上能够完成任意高速数据包缓冲和无延时调度,从而解决了存储器带宽瓶颈问题.并同时给出了三级存储阵列结构的可实现设计方案.此方案简化了硬件实现的复杂度,提高了该设计的可实现性,为其在高速网络设备中的应用打下了基础.

References:

[1] Shah D, Iyer S, Prabhakar B, McKeown N. Maintaining statistics counters in router line cards. IEEE Micro, 2002,22(1):76−81.

[2] Iyer S, Kompella RR, McKeown N. Techniques for fast packet buffers. http://www.comsoc.org/tcgn/conference/gbn2001/iyer-presentation.pdf

[3] Sailesh K, Venkatesh R, Philip J, Shukla S. Implementing parallel packet buffering: Part 1. 2002. http://www.commsdesign.com/story/OEG20020422S0006

[4] Alexander T, Kedem G. Distributed prefetch-buffer/cache design for high performance memory systems. In. Proc. of the 2nd Int'l Symp. on High-Performance Computer Architecture. Piscataway: IEEE Press, 1996. 254−263.

[5] Rixner S, Dally WJ, Kapasi UJ, Mattson P, Owens JD. Memory access scheduling. In: Proc. of the 27th Annual Int'l Symp. on Computer Architecture. IEEE/ACM Press, 2000. 128−138.

[6] Broder A, Mitzenmacher M. Using multiple hash functions to improve IP lookups. In: Proc. of the IEEE INFOCOM 2001. IEEE Press, 2001. 1454−1463.

[7] Joo Y, McKeown N. Doubling memory bandwidth for network buffers. In: Proc. of the IEEE Infocom. Vol.2, IEEE Press, 1998. 808−815.

[8] Sailesh K, Venkatesh R, Philip J, Shukla S. Implementing parallel packet buffering: Part 2. 2002. http://www.commsdesign.com/story/OEG20020429S00068

[9] Iyer S, Kompella RR, McKeown N. Analysis of a memory architecture for fast packet buffers. In: IEEE High Performance Switching and Routing. 2001. 368−373.

[10] Iyer S, Kompella RR, McKeown N. Designing buffers for router line cards. http://yuba.stanford.edu/~sundaes/publications.html

[11] Shor M, Li K, Walpole J. Application of control theory to modeling and analysis of computer systems. In: Proc. of the Japan-USA-Vietnam Work shop on Research and Education in Systems. Hochiminh City: RESCCE Press, 2000. 502−507.

[12] McKeown N. The *i*SLIP scheduling algorithm for input-queued switches. IEEE Trans. on Networking, 1999,7(4):188−201.

[13] Bhagwan R, Lin B. Fast and scalable priority queue architecture for high-speed network switches. In: Proc. of IEEE INFOCOM 2000. IEEE Press, 2000. 538−547.

[14] Wang P, DP Jin, Zeng LG. Analysis of a tri-stage memory array for high-speed packet buffers. In: Proc. of the SPIE, Vol.5625. 873−883.