一种基于拓扑特征的数据流调度策略

李嘉欣

中航空天发动机研究院有限公司,北京 100028

摘要　流式处理器针对数据并行的应用,提供多个简单的处理单元及它们之间的高带宽通道,通过高计算密度掩盖传输延迟,支持高速的数据传输和处理,与通用处理器相比,性能达到了数倍乃至数百倍的提升。在流式处理器的研究中,数据流的调度是个比较重要的问题。本文提出了一种基于拓扑特征的数据流调度策略,该策略包含平分策略和选择策略两种,其中平分策略用于体系结构中固有负载较少的情况,选择策略在固有负载较多时利用拓扑权重来辅助平分策略完成数据流任务的调度。建立了该策略的性能模型,在考虑节点间传输时间的情况下进行讨论,对数据流调度策略的选择提出建议。

关键词　数据流调度;拓扑;负载

中图分类号　TP301.6　　**文献标志码**　A　　**doi**　10.3981/j.issn.1000-7857.2013.17.007

A Data Stream Scheduling Strategy Based on Topology Features

LI Jiaxin

China Aviation Engine Research Institute Company Limited, Beijing 100028, China

Abstract　Stream processors are suitable for data parallel applications; the processor is supported by many simple processing units and high bandwidth routing among the units. By means of high density computing, the transfer latency is concealed, therefore the high speed data transfer and processing are able to be supported, making the performance of stream processors raise several or even hundreds of times comparing with general processors. Data stream scheduling is a relatively important problem in the research of stream processor. Methods that utilize the features of architecture topology to compute the weights on every edge are proposed. These methods are called as Stream Scheduling based on Topology Features (SSTF). SSTF mainly includes divide strategy and select strategy. Divide strategy is suitable for the architecture with light load. However, if there is heavy load, select strategy is used to assist divide strategy with the completion of data stream scheduling. A performance model of SSTF is built, which gives a consideration to the transfer time between nodes in one topology; and the way for selecting scheduling methods is suggested.

Keywords　data stream scheduling; topology; load

0　引言

多核处理器种类和特征各异,所支持的应用类型也各不相同。在一些通用多核处理器的设计中,由于核间互连带宽不够大,通常只负责传输比较小的核间消息,而数据主要是通过存储体系结构进行传输,这样的结构适用于核间交互数据量较小的应用,比如任务并行的应用。流式处理模型在图像处理和科学计算领域被广泛使用。流是不间断、连续、移动的记录队列。流式处理就是针对流应用的上述特征进行的特殊处理[1]。流式处理器针对的是数据并行的应用,提供多个简单的处理单元及它们之间的高带宽通道,通过很高的计算密度来掩盖传输延迟,支持高速的数据传输和处理,与通用处理器相比,性能达到了数倍乃至数百倍的提升。现在很多多核处理器的设计都希望借鉴流式处理器的模式,从而提高整体的性能。

类似于通用多核处理器中的负载平衡问题,在流式处理的过程中,数据流的调度是个比较重要的问题。一般的流式处理程序设计模型将应用组织成流(stream)和核心程序(kernel),从而把数据处理与数据调度分离开来。相应地,流体系结构通常包含异构的两种处理器核——流处理核和标量处理器核。在流式处理中,这两种异构核之间协同工作的方式主要为计算加速模式(Computational Acceleration Model),也就是以流处理核为中心,处理计算密集的应用,而标量处

收稿日期:2012-05-17;修回日期:2013-03-29

作者简介:李嘉欣,高级工程师,研究方向为嵌入式与高性能计算,电子信箱:carefulbaby@126.com

理器核主要作为流处理核上运行系统的控制与服务的工具,对并行工作进行切分。在切分的过程中,就需要对数据流在流处理核之间的移动及动态存储器访问操作进行调度[1]。

由于在流式处理中,各处理核上运行的核心程序功能都比较单一,这些单一的计算组合形成了复杂的计算。所以有时数据流需要在几个核之间进行流动才能够进行一次完整的计算。而在核间流动的过程中,如果运算的密度足够,并且步调比较合适,通信开销往往是可以被掩盖的。对于一个具有相同功能的处理核阵列来说,流的调度需要能够达到局部性和并发性有机结合。涉及到核间的数据流调度,则需要考虑多核处理器的片上网络拓扑特征。

本文提出了一种基于拓扑特征的流调度策略(Stream Scheduling based on Topology Features,SSTF)。SSTF 策略包含平分策略和选择策略两种,其中平分策略用于体系结构中固有负载较少的情况,选择策略在固有负载较多时利用拓扑权重来辅助平分策略完成数据流任务的调度。

1 拓扑特征描述模型

多核体系结构片上网络的拓扑结构有很多种,如 Tree、2D-Mesh、2D-Torus、THIN[2]等。不同的拓扑结构对于多核体系结构的调度模式会带来不同的影响。为了描述方便,本文用图来定义各种片上网络的拓扑特征。

定义 使用图 $G_T=(V_T,E_T)$ 来表示片上网络 T 的拓扑结构。其中,V_T 是节点集合,每个节点 v 代表一个处理核,并且具有一个标记位 w_v;E_T 是边集合,当节点 v_1 和 v_2 之间存在直接的连接通道时,就存在一条边 $(v_1,v_2)\in V_T$,并且具有一个标记位 w_e。w_v 和 w_e 分别称作节点 v 和边 (v_1,v_2) 的权重。

在 SSTF 中,权重是个重要的概念,它衡量了在这个拓扑中每个节点和每条通路可能的数据负载状况。根据权重被确定的时机,可以将其分几种类型:拓扑权重、映射权重、动态权重。

1.1 拓扑权重

对于一个给定的拓扑结构,每条边都具有相对固定的拓扑权重,它是由拓扑的静态特征确定的。假设已选定集合 $S_T=\Phi$,一个临时存储集合 $G_T'=(V_T',E_T')$,并将图 $G_T=(V_T,E_T)$ 中节点和边的标记位均设置为 0。计算拓扑权重的方法可以有多种,并且分别对应着相应的数据流调度策略。算法 1 是其中一种方法,称做复制平分非重新分配方法(Copy-Divide-Unreassign,CDU)。下面以算法 1 为例描述拓扑权重的计算。

在算法 1 中,a 为每个节点的额外负载系数,可以根据需要进行指定。在 a 均相同的情况下,就可以算出该拓扑各边的拓扑权重。在节点数为 n 的规模下,该算法的时间复杂度为 $o(n\log_2 n)$。作为一个对称的拓扑,互相对称的边上权重也相同。这样就可以利用拓扑的对称性,将算法进行简化,只需根据 V_T 中非对称节点最大集合中各节点进行计算,再乘以相应的系数。

算法 1 计算拓扑权重的 CDU 方法

名称:CDU

输入:体系结构拓扑 $G_T=(V_T,E_T)$

输出:标记后的体系结构拓扑 G_T

Step 1 将图 G_T' 中的每个节点和每条边都标记为 0,选定图中未包含于 S_T 的一个节点 v_i,将该节点标记为 a,并将该节点放入 S_T。

Step 2 若存在 m 条边与 v_i 相连,当 $m>0$ 时,将这 m 条边都标记为 am^{-1},并将这 m 条边另一端的每个节点标记为 am^{-1};若 $m=0$,那么就将 a 加到 G_T 中相应节点的标记上,回到步骤 Step 1。

Step 3 同时查看 G_T' 中所有标记不为 0 的节点,以及所有与这些节点相连的标记为 0 的边,若其中有 $n(n>0)$ 条边与其中一个节点 v_j(它的标记为 m_j)相连,那么将这 n 条边的标记都加上 m_jn^{-1},并将这 n 条边另一端的节点的标记都加上 m_jn^{-1}。

Step 4 循环进行步骤 Step 3,直至 G_T' 中各边和各节点的标记不再变化。

Step 5 将 G_T' 中每个结点以及每条边的标记值都加到 G_T 中相应节点或相应边的标记值上。若 $S_T\neq V_T'$,跳转到步骤 Step 1;否则,算法结束。

利用拓扑的特征,更多的是体现在利用拓扑的局部性优势上,也就是说,一个节点 v_i 如果要将自己暂时无法处理的数据(称为 v_i 的过载数据)向其他节点分配,那么最好是选择和自己临近的节点,这样可以减少因为数据传输而造成的延迟。算法 1 也是根据这个思想设计的。在算法 1 中,v_i 上的过载数据平分给和其直接相邻的节点,假设 v_j 从 v_i 接收到了数据,但是 v_j 也暂时无法处理,那么 v_j 就继续向其邻近的未标记的节点平分这些数据,发送过后对 v_j 进行标记,以此类推直至所有节点均被标记。标记表明了该节点无能力进行处理的状态,由于标记的时间具有同步性,所以会存在某些节点在将过载数据发送出去之后又会接收到其他节点发来的过载数据的情况,此时它不会再将新的过载数据继续发送,而是会将数据保持在缓冲区中以待空闲时处理。数据每经过一次传输就被分割成更小块的数据,从而保证了越往远(两节点间路径长度越大)的节点得到的过载数据量越小,从而一定程度上降低了远程数据传输的时间。算法 1 描述的是一种最坏情况,也就是拓扑中的节点均是过载的,所计算的拓扑权重就是在最坏情况下,如果各节点在每个时间步有 a 大小数据量的任务,那么拓扑中各条边所需要承担的传输压力。如果模型中各节点的设置十分平均,那么算法 1 算出的权重同时也能够代表该拓扑固有的特征。

1.2 映射权重

根据应用在多核体系结构中的映射关系,可以确定每条边的映射权重,从而估算该映射策略下网络的平均负载情况。映射关系在应用编译时就可以确定,映射权重就可以在

此时进行计算。

在计算映射权重时，可以在算法 1 的基础上进行改进。在 Step 1 中进行一些修改：每选定一个节点 v_i，不再将该节点标记为与其他节点相同的 a，而是标记为实际映射的负载的相对值 a_i。假设节点 v_i 上映射的负载为 L_i，则

$$a_i=\frac{aL_i}{\sum_{j=1}^{n}L_j}$$

1.3　动态权重

拓扑权重和映射权重表示了静态情况下的权重，而在程序运行时，可以根据体系结构的当前状态来计算其动态权重。当前状态包括各节点上的负载情况、各通道上的数据传输情况，所以当前状态的动态权重是确定的。如果需要，就可以根据各节点和各通道上的负载情况及数据流分配的不同策略来预测以后系统中的负载情况。数据流分配的策略通常要考虑数据局部性原理、各节点的负载平衡、节点间任务同步及通道的有效带宽等。动态权重虽然能够较准确地衡量拓扑中的负载状况，但是确定动态权重也需要各节点之间互相通信来实现，会给系统造成额外的负担，并且会提高分配计算的复杂性。拓扑权重显示了需要映射的体系结构的拓扑特征，从而不需要复杂的实时动态权重维护，利用静态值就可以指导数据流的动态传输，从而提高了系统的效率。所以在 SSTF 选择策略实现时，通常只对关键部分计算动态权重，非关键部分仍用拓扑权重或映射权重来衡量。

2　SSTF 的特征分类

分散-收集(scatter-gather)的思想是数据流分配中经常使用的。它将大量的数据分解成小块，分散给多个具有相同功能的处理单元并行处理，然后将处理后的结果集中起来成为最后的结果，以达到提高处理性能的目的。SSTF 策略实际上也是一种分散-收集策略。因以下几个因素的不同，SSTF 策略可以分为不同策略。

(1) 使用复制还是转移的方法。复制本节点当前的数据流到其他相邻节点，共同处理，简称复制(Copy,C)；或者将当前数据流分配到其他节点，本节点继续接收新的数据流，简称转移(Transfer,T)。

(2) 使用何种分配比例。根据邻节点的数量直接平均分配，简称平分(Divide,D)；或者是依据到各邻节点路径中权重的比例等因素进行分配，简称选择(Select,S)。

(3) 对已有负载节点的处理方法。相邻节点已有负载，则不向该节点传输，而由本节点负责该节点所分配的任务，简称非重新分配(Unreassigned,U)；或者不管相邻节点是否有负载，将分配给该节点的任务传输过去，简称正常(Normal,N)；或者在分配时就不把有负载节点计算在内，空闲节点进行分配，简称重新分配(Reassigned,R)。

SSTF 的主体是平分策略，也就是在各节点间均匀分配数据流。但是，如果在平分策略应用时，系统已存在一定的负载，那么就需要使用选择策略来辅助平分策略进行数据流调度。平分策略的执行主要依据各节点的权重及相邻节点的个数，选择策略的执行则依据平分策略的类型及相应的节点权重和边权重。

2.1　平分策略

在图 1 中，以 9 节点 THIN 为应用对象，给出了几种平分策略的图例。深色节点表示当前接收到数据流，以待分配的节点，D 用于标识当前不参与邻节点传递的边，也就是某条边进行过一次数据传输后就要标记成 D。假设起始节点的总数据量为 1。

下面以 9 节点 THIN 使用从端点出发的 TDN 为例，对转移平分策略进行解释：

(1) v_1 得到大小为 1 的数据流。

(2) v_1 将数据流转移平分给 v_2 和 v_3。

(3) v_2 和 v_3 同时将数据转移平分给相邻未访问节点，由于它们是相邻的，所以在向对方送出数据流时也会得到对方传来的数据流，这些数据流不能再进行划分。

(4) v_4 和 v_7 将数据转移平分给相邻未访问节点。

(5) v_5、v_6、v_8、v_9 同时对相邻未访问节点进行转移平分，又会出现(3)中的互传情况。比如 v_5 只有一个未访问邻接节点 v_6，那么其 1/8 数据流就全部转移给 v_6；而 v_6 有 2 个未访问相邻节点 v_5 和 v_8，它将自己的 1/8 数据流转移平分给这两个节点，同时也会收到 v_5 的 1/8 数据流和 v_8 的 1/16 数据流，最终总和为 3/16 数据流。

复制平分策略还有两种可能，一种是平分任务时算入本节点，一种是不算入本节点。如果本节点要向 n 个相邻未访问节点复制平分时，算入本节点的意思是，需要将任务分成 n+1 份，其中顺序处理时需要处理的第一份留在本节点，其余 n 份发送给各相邻节点；不算入本节点的意思是，将本节点的任务分成 n 份并发送给各相邻未访问节点。

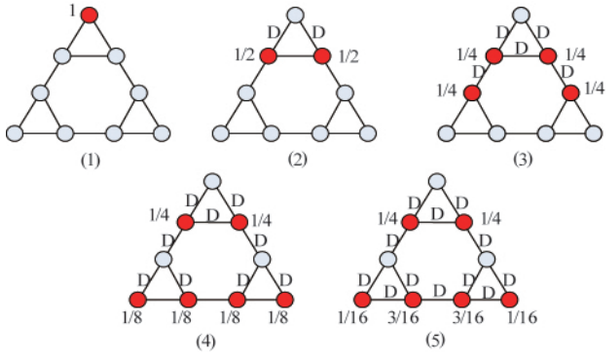

图 1　9 节点 THIN 使用从端点出发的 TDN

Fig. 1　Nine-node THIN by using TDN starting from culminations

转移平分策略大多数用于只有一个入口的大规模数据流调度，也就是说会有源源不断的数据流进入起始节点，该

节点就需要分步骤地将任务分配给其他节点,并且将本节点缓存清空以接收下一段数据流。平分任务时算入本节点的复制平分策略可以用于需要所有节点并行处理的情况;而不算入本节点的复制平分策略存在着对相同数据的冗余处理,可以用于对数据的正确性要求很高的情况,对数据的计算进行验证。

2.2 选择策略

平分策略针对运行前无负载的体系结构,从而在数据流调度的初始阶段就可以决定各处理核的调度策略,这种方法比较简单。如果节点已经有负载在运行,那么新负载就要依据节点的能力进行分配。在这时,就要根据拓扑的权重信息以及整个传输路径的情况, 对数据流的传输方向进行选择。根据这些不同情况,可以将选择策略分成以下几种类型:

(1) 选择权重最低的路径或路径群。权重最低说明该路径上负载最小,那么就越容易达到最大的传输效率。

(2) 避免选择权重最高的路径。权重最高说明该路径上负载最大,如果找到这样的路径,并在传输的时候避开它,那么就可以有效缓解重载节点和路径的压力。

(3) 选择经过节点最少的路径。在一些情况下,影响传输性能主要因素并不是权重,而是每次经过节点的数量,因为需要考虑的是节点中转的额外开销。

(4) 选择周边权重总和最低的节点。在局部性原则的支配下,一些情况需要将数据流调度到固定范围的局部节点群中,选择节点群时就可以用某节点周边权重和为依据,权重和越小,说明该节点周围的传输压力越小,以该节点为圆心进行流式调度则会具有更高的效率。

选择策略可以辅助平分策略进行动态的数据流负载调整,从而逐渐达到负载平衡。但是实现选择策略的过程中需要不断收集各节点的状态信息,这样本身会带来一定的消息传递开销。所以使用选择策略时要更加慎重。

3 SSTF 的理论性能分析

3.1 性能模型

为了适当简化问题,假设一个比较均衡的模型,其中:各数据流中的基本元素类型一致,数据流大小足以维持系统持续的处理过程;各节点的缓冲区大小一致,均为 M;每条边上传送单位数据的延迟一致。这样,利用这个模型,可以更加关注由于拓扑结构的影响造成的性能差异。

如图 2 所示,令初始数据从存储器传输到各个运算节点的时间为 T_0,得到初始数据的源节点标记为 $v^{(1)}$,从 $v^{(1)}$直接传递的第 2 级节点集合中的节点标记为 $v^{(2)}$,传输时间为 T_1,以此类推,在传递 m-1 次后满足条件或者无法再次传输而被截止,最后的传输延迟为 T_{m-1},最后到达的节点集标记为 $v^{(m)}$,其中 m 的最大值为从源节点(v_{source})出发到其他节点的最短路径的最大值,也即

$$m_{max}=\max_{i=1}^{n}\text{distance}(v_{source},v_i)$$

每级传输中,标记为 $v^{(i)}$的第 j 个节点(表示为 $v_j^{(i)}$)向外传输的下一级节点个数为 $N_j^{(i)}$,这个参数决定了该节点中过载数据需要被划分成几个部分并传递到下一级节点中去,其中每个部分需要的运算时间为 $C_j^{(i)}$。算法的优劣是根据整个数据集合被全部处理完毕所用的时间 S 所衡量的。

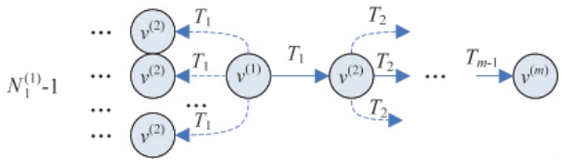

图 2 算法参数示意图

Fig. 2 Scheme of algorithm parameters

3.2 对节点间传输时间的讨论

传输时间决定于数据传输量及固有延迟。如果运算时间不够长,这种不一致就会对程序的同步有一定影响。下面将根据传输时间的不同取值对整体运行时间的计算进行分类。令 $v^{(1)}$到 $v^{(2)}$的的传输时间为 T(T 的大小可能是固定的,也可能是不固定的),为了简化模型,假设每次传输都将数据流分成 n 份,共传输 m 次,对 $v^{(1)}$上数据的顺序运算时间为 C,总时延为 S。S 和传输的次数有关,因为在传输时间达到一个阈值时,不进行分布式处理,整体的计算时间还要小一些。下面根据传输时间的不同类型,对这个阈值进行计算。

(1) 每次传输时间相同,总时延为

$$S=T(m-1)+\lceil Cn^{-m}\rfloor$$

若使传输次数增大 1 次时总时延减小,需要满足

$$T(m-1)+\lceil Cn^{-m}\rfloor\geqslant Tm+\lceil Cn^{-(m+1)}\rfloor$$

即

$$T\leqslant\lceil Cn^{-m}\rfloor-\lceil Cn^{-(m+1)}\rfloor<Cn^{-m}+1-Cn^{-(m+1)}=\frac{n-1}{n^{m+1}}C+1$$

令 $\alpha=\frac{n-1}{n^{m+1}}$,则 $T<\alpha C+1$。

(2) 每次传输时间正比于数据大小,总时延为

$$S=\sum_{i=0}^{m-2}\lceil Tn^{-i}\rfloor+\lceil Cn^{-m}\rfloor$$

若使传输次数增大 1 次时总时延减小,需要满足

$$\sum_{i=0}^{m-2}\lceil Tn^{-i}\rfloor+\lceil Cn^{-m}\rfloor\geqslant\sum_{i=0}^{m-1}\lceil Tn^{-i}\rfloor+\lceil Cn^{-(m+1)}\rfloor$$

即

$$\lceil Tn^{1-m}\rfloor\leqslant\lceil Cn^{-m}\rfloor-\lceil Cn^{-(m+1)}\rfloor<Cn^{-m}+1-Cn^{-(m+1)}=\frac{n-1}{n^{m+1}}C+1=\alpha C+1$$

又因为$\lceil Tn^{1-m}\rfloor\geqslant Tn^{1-m}$,所以 $Tn^{1-m}<\alpha C+1$。令 $\beta=n^{m-1}$,也即 $\alpha=\beta^{\frac{m}{1-m}}(1-\beta^{\frac{1}{1-m}})$,则 $T<\beta(\alpha C+1)$。

(3) 每次传输时间是在一段固定时间加上和数据大小成正比的时间,若固定的传输时间为 T_f,总时延为

$$S=(m-2)T_f+\sum_{i=0}^{m-2}\lceil(T-T_f)n^{-i}\rfloor+\lceil Cn^{-m}\rfloor$$

若使传输次数增大一次时总时延减小,需要满足

$$(m-2)T_f+\sum_{i=0}^{m-2}\lceil(T-T_f)n^{-i}\rfloor+\lceil Cn^{-m}\rfloor\geqslant(m-1)T_f+\sum_{i=0}^{m-1}\lceil(T-T_f)n^{-i}\rfloor+\lceil Cn^{-(m+1)}\rfloor$$

即

$$T_f+\lceil(T-T_f)n^{1-m}\rfloor\leqslant\lceil Cn^{-m}\rfloor-\lceil Cn^{-(m+1)}\rfloor<Cn^{-m}+1-Cn^{-(m+1)}=\frac{n-1}{n^{m+1}}C+1=\alpha C+1$$

又因为

$$\lceil(T-T_f)n^{1-m}\rfloor\geqslant(T-T_f)n^{1-m}$$

所以

$$T<\beta(\alpha C-T_f+1)+T_f$$

对一个并行算法的评价,要结合运算时间和节点间传输时间才能够更加准确得出。以上几种情况计算出了对首次传输时间 T 的约束,如果 T 小于一定值,就可以保证多一次扩展传递可以达到更少的总时延。表 1 给出了运算量 C=1000,T_f=2 时,不同 m、n 下对 T 的约束,分别用 T_1、T_2、T_3 表示第(1)~(3)种情况下 T 的最大取值。约束值越小,对传输时间的要求越严格,而当约束值为负值时,说明将运算扩展到更多计算节点反而会增大总时延。

从表 1 可以看出,T_1 和 T_3 在 m、n 增大时取值会相应减小,说明传输能力有限的时候需要将数据流的传输限制于一定规模内。而 T_2 的取值在 m、n 增大时不会有明显减小,甚至有所增大,说明在传输能力比较强的情况下,使用的运算节点范围越广,就越能够达到较好的性能。T_2 对应的传输时间在支持数据并行的流式体系结构中比较常见,在流式体系结构中有大量的处理节点,各处理节点之间使用了非常高的带宽以尽量提高节点间的传输速度,从而支持高密集度的数据流处理。

表 1　几种情况下 T 的最大取值

Table 1　Maximal values of T on several conditions

m	n	T_1	T_2	T_3
2	2	126	252	250
	3	75	225	221
	4	48	192	186
	5	33	165	157
3	2	64	254	248
	3	26	231	215
	4	13	204	174
	5	7	185	137
4	2	32	258	244
	3	9	249	197
	4	4	252	126
	5	2	285	37
5	2	17	266	236
	3	4	303	143
	4	2	444	−67
	5	1	785	−463

4　结论

本文提出了一种基于拓扑特征的流调度策略,它以多核体系结构片上网络拓扑的基本特征作为依据,对数据流进行调度和分配。侧重考虑不同的因素,SSTF 可以细分为一系列分配策略,其中最基本的策略是平分策略,它本身体现了均衡性和局部性的思想,但是当应用该策略的体系结构已有大量负载时,就需要使用选择策略来对平分策略进行修正。

数据传输和核间消息通信的延迟往往是影响多核处理器性能的主要瓶颈。在 SSTF 配套的性能评估方法中也考虑了核间的传输延迟对整体运行时间的影响。一系列评估表明,当传输时间大于一定阈值时,数据流的处理就需要局限于一定的范围内,否则就会造成使用更多的计算节点后性能反而下降的状况。

参考文献(References)

[1] 张春元, 文梅, 伍楠, 等. 流处理器研究与设计 [M]. 北京: 电子工业出版社, 2009.
Zhang Chunyuan, Wen Mei, Wu Nan, et al. Stream processor research and design[M]. Beijing: Publishing House of Electronics Industry, 2009.

[2] 乔保军. 基三多内核体系结构中互连关键技术的研究[D]. 北京: 北京理工大学, 2007.
Qiao Baojun. Research of key technology in inter-connection of triplet-based multi-processor architecture[D]. Beijing: Beijing Institute of Technology, 2009.

[3] Jayanth G, Joel C, Yoshio T, et al. Streamware: Programming general-purpose multicore processors using streams[C]// Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2008.

[4] Zilles C. Master/slave speculative parallelization and approximate code [D]. Madison, WI: University of Wisconsin-Madison, 2002.

[5] Bedford T M, Walter L, Jason M, et al. Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ILP and streams [C]// Proceedings of the 31st Annual International Symposium on Computer Architecture. Washington DC IEEE Computer Society, 2004.

[6] Karthikeyan S, Ramadass N, Liu H, et al. Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture [C]// Proceedings of the 30th Annual International Symposium on Computer Architecture. New York, USA: ACM, 2003.

(责任编辑　刘志远)