文章编号: 1001- 2486(2009) 05- 0001- 05

流体系结构技术发展探讨*

张春元, 文 梅, 伍 楠, 任 巨, 管茂林, 何 义
(国防科技大学 计算机学院, 湖南 长沙 410073)

摘 要: 以流计算模型为基础的流体系结构, 是面向未来的单片上集成超 10 亿只晶体管和上千 ALU 时代的新型体系结构, 正成为微处理器体系结构研究关注的前沿焦点之一。首先分析流计算的背景; 总结现有的具有代表性的流体系结构, 并对它们的结构、执行模式、并行性、片上存储使用方式和应用目标等方面进行了比较; 然后归纳流程序设计及其环境, 讨论当前流编译研究的热点方向; 最后探讨流处理器设计的发展趋势。

关键词: 流处理器; 体系结构; 流编程; 流编译
中图分类号: TP303 文献标识码: A

Discussion on Evolution of Stream Architecture

ZHANG Chun-yuan, WEN Mei, WU Nan, REN Ju, GUAN Mao-lin, HE Yi
(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: For epoch of more than one billion transistors and 1000 ALUs in one chip, stream architecture based on kernel-stream model has turned out to be a hotspot. In light of this, a discussion on evolution of stream architechture is made. Firstly, the background of stream computation is given and features of the representative stream architechitures are summaries. Then, the architecture, processing model, parallelism, on-chip memory, application of some stream architectures are presented. The current dramatic changes and ongoing development trends of hardware and software design in stream processor are discussed in this paper.

Key words: stream processor; architecture; stream programming; stream compiling

1 背 景

VLSI 制造技术和工艺的不断发展, 使集成电路芯片上的计算成本大幅下降, 如何实现有效的密集计算成为计算机应用研究的新热点, 并行体系结构研究获得了巨大的基础技术支撑和应用驱动。近十年来, 以高效密集计算为目标的新型并行计算机体系结构模型呈现出涌现状态, 流(Stream) 计算模型[1]就是其中之一。流计算最初来源于以图像和视频为代表的媒体应用, 目前正在逐步渗透到信号处理、图形图像和一些科学计算等密集计算领域。十多年来, 流计算体系结构模型呈现出百花齐放的状态, 学术界和工业界设计了一系列经典的流体系结构和流处理器(原型), 典型的结构有 Imagine、Merrimac、CELL、Trips、RAW、Clearspeed、STORM、Tile64、FT64、MASA 和几乎所有的 GPU[2]。在这些处理器和原型系统上, 流体系结构不但展现出了巨大的计算潜力, 而且在面积利用率、时钟系统设计、功耗效率和程序设计等方面也展现出优势。我们认为, 流体系结构极有可能成为未来高性能处理器的有机组成部分, 流处理的思想会对未来的计算器件产生重要的影响。

流计算模型是对流元素进行处理的两级计算模型。它将应用分解成一连串的生产—消费过程, 在这个过程中被加工的对象称之为流元素, 对流元素进行加工的每个过程就是一个 Kernel, 称为 Kernel 级计算, 流元素在 Kernel 之间受控传递, 称为 Stream 级计算, 所以, 流计算模型也称为 Kernel-Stream 模型。

* 收稿日期: 2009- 07- 03
基金项目: 国家自然科学基金资助项目(60673148, 60703073); 博士点基金资助项目(20069998025); 国家 863 计划资助项目(2007AA01Z286); 教育部"高性能微处理器技术"创新团队资助项目(IRT0614)
作者简介: 张春元(1964—), 男, 教授, 博士生导师。

适合于流计算模型的应用称为流应用,流应用具有丰富的并行性、多层次的局域性和数据访问的可预知性等特点。流体系结构基于这些特点,在程序设计、编译和硬件结构上直接支持流计算模型,从而有效加速流应用。

VLSI技术发展和应用需求是流体系结构产生的根源。一方面,VLSI技术的发展使大量功能单元可以廉价地做在一个芯片内,但集成(连接)这些功能单元所需的全局时钟、通讯线路和功耗却需要更高的开销,即功能廉价,通讯昂贵。体系结构研究人员必须解决如何有效利用芯片面积来计算的问题。另一方面,以密集计算为典型特征的流应用,特别是媒体、图形图像和数字信号处理类应用,成为微处理器面对的一类主要负载。

流体系结构技术特征主要包括:(1)访存和计算解耦合。在组成方面看,流处理器硬件分为两大部分:流调度和流计算,流调度和流计算的解耦和在操作层对应于访存和计算解耦合;(2)包含多种并行执行模型,有SIMD、MIMD和SKMD(Single Kernel Multi-Data),可开发多重级别的并行,包括ILP、DLP和TLP;(3)Stream-Kernel两级程序控制逻辑;(4)流计算部件拥有大规模的采用简单指令流出逻辑控制的ALU阵列;(5)流调度部件负责向流计算部件提供数据流和指令流;(6)拥有大量的局部寄存器文件和高速互联保证流计算所需的数据带宽;(7)提供片内多级存储层次;(8)具有巨大的后备存储和多路访存通道;(9)强调面积、功耗和成本有效的可获得实际性能。

2 流处理器

流体系结构一直受到工业界和学术界广泛关注。第一片流处理器Imagine,2003年在斯坦福大学诞生,早期的流处理器还包括VIRAM,RAW。本节简要介绍近期几款有代表性的流处理器。

STORM[3]:SPI(Stream Processor Inc.)公司2007年推出,原型为Imagine,是商业化的高性能系列流DSP处理器,目前主要应用于视频监控领域。以SP16HP为例,支持SIMD+VLIW执行,集成两个64位MIPS标量核、16个计算簇,片上存储器支持有限索引。

Tile64[4]:Tilera公司于2007年发布的Tile结构的高性能微处理器,原型为RAW,瞄准计算密集型的嵌入式应用,例如网络交换和通信的领域。单片64个Tile,采用VLIW指令执行方式,每个Tile都是一个全功能通用处理核,多个Tile可以组成Lane,为应用提供足够性能。

CELL[5]:IBM、索尼、东芝三家公司在2005年和2008年联合推出了第一代和第二代Cell处理器。Cell处理器结构包含一个64位的Power处理核(PPE)和8个协处理核(SPE)。SPE采用SIMD,具有本地数据和指令存储。第二代Cell处理器提升了双精度浮点处理能力,因此更适合通用计算。

FT64[6]:国防科技大学2007年研制成功的专门面向科学计算的64位流处理器,其体系结构与Imagine流处理器类似。

TRIPS[7]:比较典型的Tile体系结构的处理器,包含两个处理核,每个处理核具有16个同构的执行Tile。对流应用,TRIPS通过采用线程级并行,静态分配任务给片内Tile来获得高的运算性能。

GPU:包括几百个流处理单元的GPU类处理器,例如ATI Radeon 4870和Geforce 8800 GTS 512等。

这些流处理器代表了现今面向密集计算体系结构的研究前沿,它们都特别强调对流应用的支持(这是我们在研究流体系结构时关注它们的原因),但是它们之间仍然存在显著的差别。第一是系统结构不同,处理核主要可分为异构多核、单片单核或对称多核,拓扑结构包括二维Mash(如Tile64)、总线(如Imagine)、环形总线(如CELL)等。各流处理器系统结构如图1所示,图中空心圆表示ALU,正方形表示可独立运行一个线程的处理结点。第二是执行模式不同,例如GPU采用SIMT执行模式,VIRAM和Imagine都是以SIMD方式工作,而RAW和TRIPS都强调MIMD执行模式。但RAW的结点粒度较小,对所有的并行采用统一最小的硬件集合来实现,而TRIPS结点粒度有多种层次,对不同的并行可采用不同的硬件集合来完成,支持几乎各种执行模式。第三是并行性开发的重点不同,它们在指令级并行(ILP)、数据级并行(DLP)和线程级并行(TLP)上各有侧重,DLP和ILP的开发与现有的并行程序设计模式基本兼容,程序设计和编译问题都比较容易解决。图2是各流处理器并行性开发的维度示意图,例如Imagine主要开发DLP和ILP,RAW则着重开发TLP。第四是片上存储器的使用方式,主要有软件(静态)管理、

软硬联合管理和动态管理, 流计算模型下访存具有高可预知性, 所以软件管理和软硬联合管理成为研究热点。第五是应用目标不同, STORM、FT64 和 GPU 等是“专用”流体系结构, 对流的支持单一, TRIPS 和 Tile64 是面向“通用”体系结构, 可很好地支持非流应用, 而 CELL 则采用了一种变化的技术路线, 采用面向单一应用的设计, 目前向通用应用领域发展。

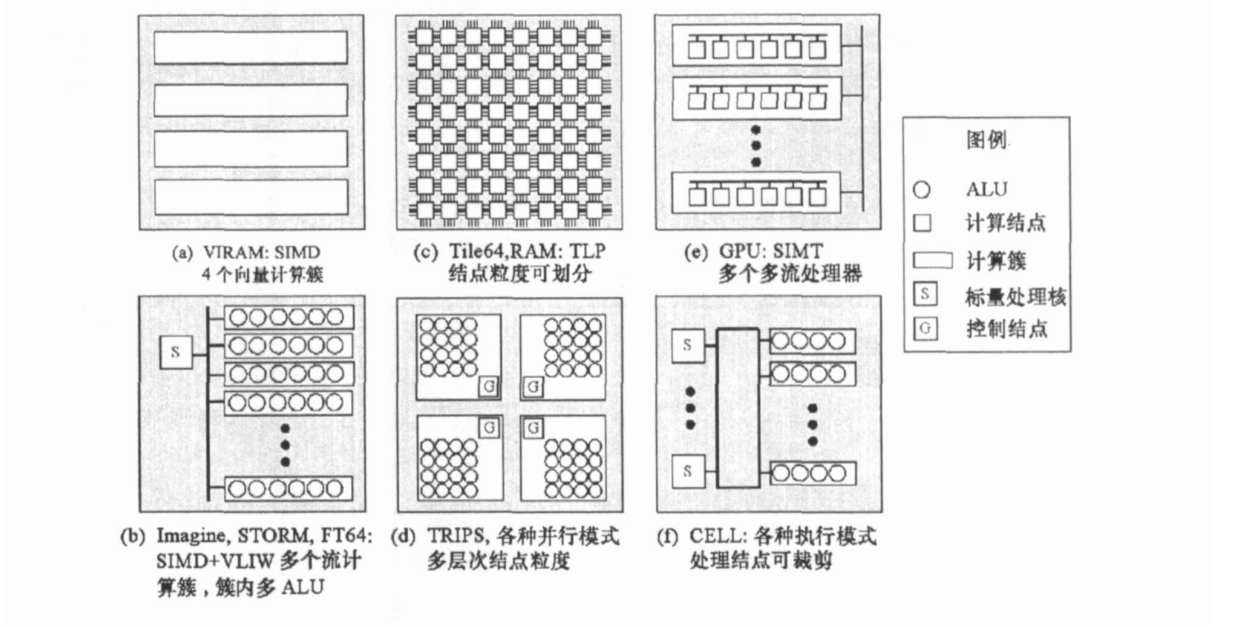

图 1 流处理器体系结构图

Fig. 1 The architecture of stream processor

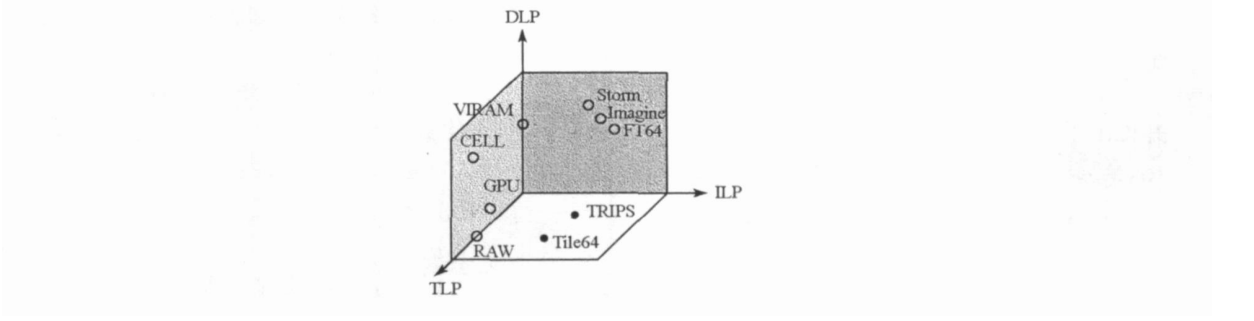

图 2 流处理器并行性开发维度图

Fig. 2 The dimensionality of stream processor parallelism exploitation

3 流编程模型和语言

流体系结构更注重硬件开销、资源使用效率和灵活性, 强调高效能、低复杂度和可编程性, 而不仅仅单一地追求绝对性能。由于流计算模型展现出高可预知性, 流处理器可更多地采用软件管理技术, 将更多的底层硬件资源暴露给程序员。因此, 各个流体系结构往往采用专用编程语言。目前应用较广泛、影响较大的有 StreamIt[8], StreamC/KernelC[9], Brook[10], Sequoia[11] 等, 下面对它们作简单介绍。

StreamIt: MIT 针对 RAW 流处理器开发的一种编程语言, 是 Java 语法的一个子集扩充。它将流处理器的处理过程看做一个一个独立的计算模块 filter, 每个 filter 都有自己的存储器, 能够通过所提供的高带宽的数据通路和自己紧挨着的邻居通信。

StreamC/KernelC: 斯坦福大学为 Imagine 流处理器配套设计实现的流程序设计语言。它是在 C 语言的基础上进行扩展的, 应用被划分为一个或多个 Kernel 和 Stream, 程序分两级编写: 流级(StreamC) 和核

心级(KernelC)。StreamC/KernelC 语言及其扩展版本目前已经在 Imagine、FT64、STORM 等流处理器上应用,我们的 MASA 流处理器使用的语言也是以它为基础进行扩展的。

Brook:最初是为 GPU 图形处理器设计,类似于 StreamC/KernelC,它的一个重要亮点是对多维数组的存储的优化。在流的定义上,Brook 允许用户定义两种不同作用的同构数据集:输入流(input stream)和聚集流(gather stream),前者允许 kernel 以规则顺序读入但不可重用,而后者可以随机读写并能够重用。

Sequoia:为解决多级存储层次中数据的软件管理而设计。它将程序中的计算和访存划分为多个 Task 结构,对同一 Task 可以提供多种实现版本,系统能够根据程序的环境来选择调用特定的 Task 版本。同时,通过配置文件描述数据在多级存储层次中的位置,将所有存储层次完全暴露给程序员,以便于提高程序的局部性和并行性。

流程序设计语言与传统的程序设计语言有较大差异,表 1 为它们的比较对照。

表 1 流程序设计语言与传统程序设计语言比较

Tab. 1 Comparison of stream programming language and traditional programming language

传统的程序设计语言	流程序设计语言
描述能力强,覆涵所有应用	通用语言的扩展,对流应用有极高的描述能力
以数组或指针描述类似于流的数据	可以直接描述各种流(非/变长,不/带索引,一维/多维)
单一层次	一般为分层的程序设计模式
隐藏数据的访问和通信	数据流加载和通信对程序员可见
可交叉调用、嵌套的函数和过程	简单的 Kernel 或 Filter
数据流以图表示	数据流结构化

流应用的多样性以及流计算模型与通用计算模型的显著差别,使得将应用程序映射或移植到流处理器上并高效运行成为一项非常艰难的工作。如何将流编程模型与通用的并行编程模型融合起来,是目前研究的热点,例如 CUDA[12] 和 OpenCL[13]。目前还有人在考虑将流编程融入传统的面向对象的语言中[14],这对于研究多核并行编程也是一条可借鉴的路线。

最近,我们针对通用程序中深层次的函数调用、细粒度的数据处理、大量的分支和跳转等特征,归纳并提出了一种通用的应用流化方法,包括关键函数扁平化、去除数据相关、计算核心封装、计算核心参数化、全局变量本地化等关键技术,为程序员迅速有效地进行程序移植提供了一种有效的方法[15]。

4 流编译

流编译是与流体系结构和流编程模型相关的。Labbonte 等提出了面向流虚拟机(SVM)的两级编译的方法[16],在一定程度上降低了流编译研究的复杂性。由于大多数的流编程模型采用的是分级的程序设计模式,因此流的编译也是分级的。常见的流编译分为两级:Kernel 级编译和 Stream 级编译。Kernel 级编译负责 Kernel 的指令调度和寄存器分配,Stream 级编译则负责 Stream 的编译,对流处理器的各种资源进行分配。

针对处理器内部各种资源限制,例如寄存器文件数目、存储单元大小、通讯信道等,如何根据程序的特征最大限度地利用好这些资源、获得最高的效率,一直是该领域的研究热点。我们针对流体系结构下重负载应用导致寄存器过载的情况,提出了溢出调度技术[2],通过负载平移、指令槽插入和基本块重划分三个互补策略,解决了分布式寄存器文件结构下计算密集型核心程序的寄存器分配和调度问题。文献[17]针对嵌入式多核处理器上运行的流应用具有严格的实时性和资源约束,开发了 Stream 编译器 SPIR,它通过开发流水线并行来同时满足性能需求和存储器资源约束,使用任务调度器提供多核任务调度以满足实时性约束。另外,基于 SDF、RSFG 等类似流计算模型或者流计算模型的流应用调度一直以来是研究的热点。文献[18]提出了针对多核的框架,由程序员在软件辅助下显式地将控制数据和处理过程分配到多个处理阵列和存储阵列上。文献[19]提出了一个介于流应用程序和目标体系结构的一般性中间界面,在 Cell 处理器和 StreamIt 语言构成的研究平台上,提出了动态的和静态的策略,对计算核心

函数和数据通信在多核流处理器上的映射进行了调度处理, 使得流程序能够高效执行, 该机制可以应用到其它的平台上来实现流应用程序的可移植性。文献[20]提出将 StreamIt 程序在 GPU 上启用软流水执行的框架。

5 流处理器的发展趋势探讨

5.1 众核化

研究表明, 具有 64 个 ALU 是单核流处理器的最佳规模, 再扩展将降低流处理器的效率[21], 因此多核/众核流处理器成为规模扩展的必经之路, 它们通常会集成 64 个以上的流处理核。斯坦福大学的 ELM 项目宣称其流式处理核心的数目在未来将达到 3200 个。

5.2 低功耗高性能的嵌入式流处理器

研究证明, 目前的嵌入式应用对性能和功耗的需求同样迫切: 通用体系结构的 DSP, 其指令和数据的供给传送分别占处理器总能耗的 42% 和 28%, 而计算仅消耗 6% 的能量[22]。斯坦福大学的 ELM 项目面向嵌入式领域, 基于流处理思想, 开发低功耗、高性能、可编程的嵌入式微处理器。ELM 通过下列方法来降低处理器的功耗: (1) 使用与计算单元紧密耦合的指令寄存器来减少指令传送的开销; (2) 通过分布式且结构化的数据寄存器开发计算的重用性和局域性; (3) 编译器可管理的指令和数据存储资源都分散在功能单元周围; (4) 通信资源通过分布式开关将结果从功能单元传递到分布式寄存器。这种对于编译器和程序员充分暴露的体系结构允许编译器显示的管理指令和数据在存储层次间的传送, 从而节省了处理器的功耗。

5.3 与通用处理器的融合

通用处理器与流处理器在不同的领域各有优势, 而当前微处理器需要同时面对桌面应用和计算密集应用, 因此通用处理器和流处理器融合的建议被提出, 并迅速得到业界响应, 包括 Intel 公司的 Larrabee 和 AMD 公司的 Fusion。此类处理器通常将多核 CPU 和 GPU 集成在同一芯片内: 其 CPU 核心具有 x86 结构和 Cache 存储层次, 而 GPU 核心采用大量的流处理器单元。其中, 多核 CPU 用于通用任务, 而 GPU 用于对流处理任务进行计算加速。

5.4 可重构和定制

流应用虽然大多具有相似的特征, 但是每个应用仍然具有各自的特点, 结构固定的流处理器并不能发挥最高效率来获得每个应用的最佳性能。针对这一问题, 我们提出可重构流处理器, 根据每一个应用的特征, 通过重构, 使用较小的代价迅速获得适应于该应用的流处理器, 为每个应用量身打造最高效的处理器, 以获得最佳性能。通常可重构流处理器采用异构多核的结构, 每种结构适合应用中不同的任务。

6 结束语

流处理器的研究已经开展了 10 多年, 其中出现了一批非常重要的概念、结构、原型、产品以及配套的软件工具。实践证明, 流处理器在媒体处理、信号处理、科学计算等领域取得了显著的成绩。对于流处理器体系结构, 我们认为众核化、高效能、可重构以及与通用处理器的融合是未来流处理器的重要发展趋势。

(下转第 11 页)

[C]// Proceedings of the 13th International Conference on Field-programmable Logic and Applications, 2003.
[9] Rauwerda G K, Heysters P M, Smit G J M. Towards Software Defined Radios Using Coarse-grained Reconfigurable Hardware[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2008, 16(1): 3-13.
[10] 高德远. 软件无线电的可重构流处理器体系结构[J]. 航空学报, 2008, 29(6).
[11] Ramacher U. Software-defined Radio Prospects for Multistandard Mobile Phones[J]. Computer, 2007, 40(10): 62-69.
[12] Schulte M, Glossner J, Jinturkar S, et al. A Low-power Multithreaded Processor for Software Defined Radio[J]. Journal of VLSI Signal Processing Systems for Signal Image and Video Technology, 2006, 43(2-3): 143-159.
[13] Wu K H, Kanstein A, Madsen J, et al. MT-ADRES: Multi-threading on Coarse-grained Reconfigurable Architecture[J]. International Journal of Electronics, 2008, 95(7): 761-776.
[14] Lin Y. Realizing Software Defined Radio — A Study in Designing Mobile Supercomputers[D]. United States, Michigan: University of Michigan, 2008.
[15] Woh M, Lin Y, Seo S W, et al. From SODA to Scotch: The Evolution of a Wireless Baseband Processor[C]// Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture, 2008.
[16] Svan Berkel K, Heinle F, Meuwissen P P E, et al. Vector Processing as an Enabler for Software-defined Radio in Handheld Devices[J]. Eurasip Journal on Applied Signal Processing, 2005(16): 2613-2625.
[17] Fang X, Wang D, Chen S M. SPVA: A Novel Digital Signal Processor Architecture for Software Defined Radio[C]// Proceedings of the 2008 IEEE/ACS International Conference on Computer Systems and Applications, 2008.
[18] Mamidi S, Blem E, Schulte M J, et al. Instruction Set Extensions for Software Defined Radio[J]. Microprocessors and Microsystems, 2009, 33(4): 260-272.
[19] Mamidi S, Schulte M J, Xie Z P, et al. Arithmetic Units for Software Defined Radio[C]// 2006 Fortieth Asilomar Conference on Signals, Systems and Computers. 2006: 341-346.
[20] Woh M, Lin Y, Seo S, et al. Analyzing the Scalability of SIMD for the Next Generation Software Defined Radio[C]// Proceedings of the 2008 IEEE International Conference on Acoustics, Speech and Signal, 2008.
[21] 车德亮. LS-DSP 数字信号处理器总线的低功耗设计[J]. 国防科技大学学报, 2005, 27(2).

(上接第 5 页)

参 考 文 献:

[1] Rixner S, Dally W J, et al. A Bandwidth-efficient Architecture for Media Processing[C]//The 31st Annual ACM/IEEE International Symposium on Microarchitecture, USA, 1998: 3-13.
[2] 伍楠. 高效能流体系结构关键技术研究[D]. 长沙: 国防科技大学, 2008.
[3] Stream Processors Inc. Storm-1 SP16-G160 Stream Processor Data Sheet[R]. http://www.streamprocessors.com, 2007.
[4] http://www.Tilera.com/products/processors.php, 2007.
[5] Hofstee H P. Power Efficient Processor Architecture and the Cell Processor[C]// Proceedings of the 11th International Symposium on High Performance Computer Architecture, 2005: 258-262.
[6] Yang X, Yan X, et al. A 64-bit Stream Processor Architecture for Scientific Applications[C]// Proceedings of the 34th Annual International Symposium on Computer Architecture, 2007: 210-219.
[7] Karthikeyan S, et al. Distributed Microarchitectural Protocols in the TRIPS Prototype Processor[C]// Proceedings of the 39th Annual International Symposium on Microarchitecture, 2006: 480-491.
[8] StreamIt Group. StreamIt Webpage[R]. http://catfish.csail.mit.edu/streamit/.
[9] Mattson P. A Programming System for the Imagine Media Processor[D]. Dept. of Electrical Engineering, Ph.D. Thesis, Stanford University, 2001.
[10] Foley B T, Horn D, Sugerman J, et al. Brook for GPUs: Stream Computing on Graphics Hardware[C]// SIGGRAPH, 2004: 777-786.
[11] Fatahalian K, et al. Sequoia: Programming the Memory Hierarchy[C]// Proceedings of SC'06, 2006: 207-216.
[12] NVIDIA Inc. http://www.nvidia.com.
[13] Munshi A. OpenCL: Parallel Computing on the GPU and CPU[R]. Technique Report, SIGGRAPH2008, 2008.
[14] Otto F, et al. Tichy: Streaming Extensions for Object-oriented Languages[C]// Workshop on Streaming Systems: From Web and Enterprise to Multicore 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2008.
[15] Wu N, Wen W, et al. Streaming HD H.264 Encoder on Programmable Processors[R]. Technique Report, 2009.
[16] Francois Labonte, et al. The Stream Virtual Machine[C]// PACT 2004: 267-277.
[17] Choi Y, et al. Stream Compilation for Real-time Embedded Multicore Systems[C]// Proceedings of 2009 International Symposium on Code Generation and Optimization (CGO), 2009: 210-220.
[18] Bouzas B, et al. MultiCore Framework: An API for Programming Heterogeneous Multicore Processors[C]// Workshop on Streaming Systems: From Web and Enterprise to Multicore 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2008.
[19] Zhang D, et al. A Lightweight Streaming Layer for Multicore Execution[J]. SIGARCH Computer Architecture News, 2008, 36(2): 18-27.
[20] Udupa A, et al. Software Pipelined Execution of Stream Programs on GPUs[C]// Proceedings of 2009 International Symposium on Code Generation and Optimization (CGO), 2009: 200-209.
[21] Khailany B, Dally W J, et al. Exploring the VLSI Scalability of Stream Processors[C]// Proceedings of the 9th Symposium on High Performance Computer Architecture, Anaheim, California, USA, 2003: 153-164.
[22] Dally W J, et al. Efficient Embedded Computing[J]. IEEE Micro, 2008: 27-32.