基于MPSoC并行调度的矩阵乘法加速算法研究

杨 飞[1,2] 马昱春[2] 侯 金[1] 徐 宁[3]
(中南民族大学智能无线通信湖北省重点实验室 武汉 430074)[1]
(清华大学计算机科学与技术系 北京 100084)[2]
(武汉理工大学交通物联网技术湖北省重点实验室 武汉 430074)[3]

摘 要 矩阵乘法是数值分析以及图形图像处理算法的基础,通用的矩阵乘法加速器设计一直是嵌入式系统设计的研究热点。但矩阵乘法由于计算复杂度高,处理效率低,常常成为嵌入式系统运算速度的瓶颈。为了在嵌入式领域更好地使用矩阵乘法,提出了基于MPSoC(MultiProcessor System-on-Chip)的软硬件协同加速的架构。在MPSoC的架构下,一方面,设计了面向硬件约束的矩阵分块方法,从而实现了通用的矩阵乘法加速器系统;另一方面,通过利用MPSoC下的多核架构,提出了相应的任务划分和负载平衡调度算法,提高了并行效率和整体系统加速比。实验结果表明,所提架构及算法实现了通用的矩阵乘法计算,并且通过软硬件协同设计实现的多核并行调度算法与传统单核设计相比在计算效率方面得到了显著的提高。

关键词 矩阵乘法,MPSoC,并行计算,负载平衡
中图法分类号 TP302 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.08.007

Research on Acceleration of Matrix Multiplication Based on Parallel Scheduling on MPSoC

YANG Fei[1,2] MA Yu-chun[2] HOU Jin[1] XU Ning[3]
(Hubei Key Laboratory of Intelligent Wireless Communications, South-central University for Nationalities, Wuhan 430074, China)[1]
(Department of Computer Science & Technology, Tsinghua University, Beijing 100084, China)[2]
(Hubei Key Laboratory of Transportation Internet of Things, Wuhan University of Technology, Wuhan 430074, China)[3]

Abstract Matrix multiplication is the basic algorithm of the numerical analysis, graphics and image processing. General matrix multiplication accelerator has always been a research focus in the embedded system design. However, due to the high complexity and the low processing efficiency, matrix multiplication becomes the bottleneck of computation speed of embedded systems. In order to use matrix multiplication in the embedded field, a synergy acceleration architecture of software and hardware based on MPSoC was proposed in this paper. With MPSoC architecture, the partitioning of the matrix considering hardware constraints is implemented in our HW/SW system to enable the computation of general matrix multiplications. The parallel computation with multiple cores and hardware function unit is realized with the load balance algorithms. Parallel efficiency and speed-up ratio are improved. The experimental results show that the proposed general matrix multiplication approach can achieve significant speed-up over the traditional approaches with single core.

Keywords Matrix multiplication, MPSoC, Parallel computation, Load balance

随着计算机理论技术和体系结构的不断发展,软硬件结合的应用平台逐渐成为了嵌入式领域的一种主流趋势。软和硬的结合可实现实时控制以及加速处理图像/视频信息。

矩阵算法为图像处理、视频分析的基础算法,随着矩阵阶数的增加,软件实现的时间复杂度和空间复杂度都越来越大。由于软件的实现方案已经不能满足一些高速应用的需求,往往需要通过设计实现硬件加速器以提高运算效率。

已有很多相关工作[1-3]将矩阵乘法实现于嵌入式应用平台中,大大加快了矩阵乘法的运算速度,针对特殊矩阵乘法也提供了不错的解决方案[4-7]。但是专用的硬件设计往往只能针对特殊矩阵运算方法,缺少必要的灵活性,导致其应用领域受到限制。软硬件结合的架构[8-10]提高了矩阵乘法运算的通用性。通过软硬件任务的划分,使得软硬件优势互补,提高了矩阵乘法的整体运行效率,但传统设计在内核设计方面只承担了控制逻辑,矩阵乘法几乎完全由硬件实现,一旦用于MPSoC架构,将会造成多核资源的严重浪费。目前,MPSoC[11-15]系统的多核设计为嵌入式系统的软硬件协同设计提供了更多运算单元,同时也对软硬件的协同设计提出了更大

到稿日期:2016-07-30 返修日期:2016-11-10 本文受European Union Seventh Framework Programme(318521),国家自然科学基金面上项目(61076035)资助。

杨 飞(1991—),男,硕士,主要研究方向为基于MPSoC的软硬件协同加速,E-mail:15527403693@163.com;马昱春 女,博士,副教授,硕士生导师,主要研究方向为集成电路设计自动化、算法设计与分析;侯 金 男,博士,副教授,硕士生导师,主要研究方向为光电检测与信息系统等;徐 宁(1968—),男,博士,教授,博士生导师,主要研究方向为集成电路设计自动化、数据挖掘。

的挑战，面向通用型的矩阵乘法运算需要相应的协同设计方法实现软件/硬件设计，充分利用多 CPU 的优势来提升系统的整体性能。

本文在基于 MPSoC 的软硬件协同加速的架构下，通过利用双核并行，调度硬件加速器，达到了软硬件并行加速的效果。双核并行是矩阵乘法并行的前提及关键，在 MPSoC 架构下只能在系统内核实现双核并行，继而实现软硬件并行，而内核程序往往又会受到系统及其他因素的限制，这也是本文的一个难点。通过合理地划分软硬件任务，达到软硬件负载平衡，同时高效地完成矩阵乘法。本文的创新点包括：

1）基于 MPSoC 进行了通用矩阵乘法加速器的设计。针对 MPSoC 架构，对矩阵乘法器进行合理的加速优化和软硬件接口的设计，从而实现了通用的矩阵乘法加速器。

2）面向 MPSoC 多核调度实现了软硬件协同加速的架构。在该架构下实现了双核并行，提出了相应的任务划分和负载平衡调度算法，达到了软硬件并行的协同加速效果。

1　MPSoC 平台架构介绍

在 MPSoC 架构下，利用多 CPU 并行的优势，不仅能够明显提高算法的并行度，而且能够降低整体的系统功耗。

图 1 是 MPSoC 架构的简图。目前，在 MPSoC 架构下支持的实时操作系统体系结构有 SMP（对称多处理器）和 AMP（非对称多处理器）两种。在 SMP 架构的系统中，所有 CPU 共享系统内存和外设资源[16-19]，由操作系统负责处理器间的协作，并保持数据结构的一致性。在 AMP 构架的系统中，需要对不同 CPU 上的操作系统使用的硬件资源进行划分，CPU 间的协作仅限于使用共享存储器的情况。

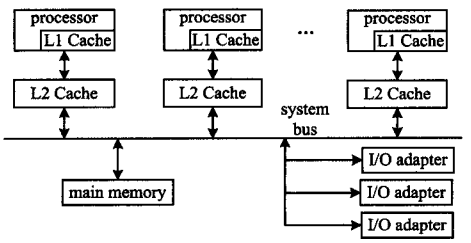

图 1　MPSoC 架构简图

在 SMP 构架下，IPI（内部处理器中断）是实现双核并行的关键，IPI 可以被任意一个处理器用来对另一个处理器产生中断，在此基础上达到双核乃至多核并行。为了更好地发挥资源共享以及多核并行带来的加速效果，本文选择了 SMP Linux。

在 Linux 下挂载硬件外设，完成硬件驱动，实现软硬件协同的架构。硬件乘法器在 MPSoC 架构中有着非常重要的加速作用；但只有有了硬件加速器，MPSoC 才是一个完整的软硬件协同加速架构。

2　硬件加速器的设计

由于是面向基于 MPSoC 的软硬件协同加速的架构，为了实现高效的矩阵乘法加速器，需要软硬件接口的支持和算法的优化设计。而且对矩阵乘法进行软硬件任务划分时，还需要对矩阵分块。

2.1　软硬件接口的实现

硬件 IP 的接口选择使用 AXI4-Lite，这是因为 AXI4-Lite 是一个轻量级的地址映射单次传输接口，占用很少的逻辑单元。在 AXI4-Lite 接口的基础上，控制握手协议使用 ap_ctrl_hs，这个协议共有 4 个单独的寄存器，不占用硬件 IP 的其他通信端口。寄存器的功能如表 1 所列。

表 1　ap_ctrl_hs 协议寄存器

寄存器	功能
ap_start	置 1，控制硬件 IP 开始执行
ap_ready	输出值表明硬件 IP 是否准备好接收新数据
ap_done	输出值表明硬件 IP 是否执行完
ap_idle	输出值表明硬件 IP 是正在执行还是空闲

在 Linux 驱动中，将硬件 IP 注册为 Dev 设备，并将 IP 的物理地址映射为 Linux 内核空间的虚拟地址，通过对虚拟地址的读写，完成对硬件 IP 的访问。并且注册 Dev 设备中断和中断服务函数，当硬件执行完，中断服务程序会立即响应。图 2 所示为 Linux 系统下的软硬件接口。

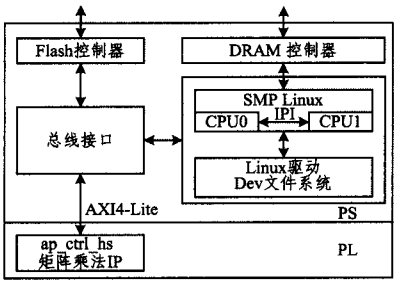

图 2　Linux 下的软硬件接口

2.2　矩阵乘法加速器优化设计

完成软硬件接口设计后，还需要对算法进行优化。用 C 语言实现矩阵乘法的最直接方式是采用 3 个 for 循环，从里层循环到外层循环分别为：Product，Clo，Row。对每层循环都进行 PIPELINE 优化，优化后硬件的并行性更高，执行效率更高。

C 语言实现硬件矩阵乘法的伪代码如下：

```
Row：for 遍历输入矩阵 A 的行 do
Clo：    for 遍历输入矩阵 B 的列 do
Product：        for 遍历输入矩阵 B 的列 do
                        求取输出矩阵 C 的元素
                    end for
                end for
        end for
```

因为是 3 层嵌套的循环，而且是在最里层循环 Product 中进行矩阵元素的乘法和求和，外层循环的 PIPELINE 优化实际上已经包含了里层循环的优化，所以在面向 MPSoC 架构进行优化时，选择从 Product 循环逐步优化到 Row 循环。从里层循环到外层循环的优化的资源利用率会越来越高，优化的过程要避免资源的利用率超过 100%。另外，因为不同的硬件平台的资源是不一样的，而且在 MPSoC 架构下的硬件矩阵乘法 IP 需要以较低的资源利用率达到符合条件的加速效果，所以不能进行全局的 PIPELINE 优化。因为这样会优化硬件 IP 与软件的接口控制协议，大大提高资源利用率。

2.3 矩阵分块

对于大型或超大型矩阵乘法，因为硬件资源有限，不可能用硬件来完成整个矩阵乘法。所以需要对矩阵进行分块，而且针对不同的应用场景，需要选择适合的矩阵分块方式。当硬件 IP 实现的矩阵乘法大小为 $n*n$，要完成 $m*m$（m 是 n 的整数倍）的矩阵乘法时，需要对输入矩阵进行 p 等分，这样会形成 q 个矩阵乘法任务，其中：

$$p=(m/n)^2 \tag{1}$$

$$q=(m/n)^3 \tag{2}$$

如果 m 不是 n 的整数倍，那么就不能对 $m*m$ 进行均匀分块，而硬件是大小完全固定的矩阵乘法，因此也就不能循环调用硬件完成矩阵乘法。

当 $n=32$，$m=128$ 时，矩阵划分的最小粒度为 $32*32$。图 3 所示为 $128*128$ 矩阵 A 的划分示意图。矩阵分块后形成 16 个子矩阵：$32*32$ 的矩阵 A_0-A_{15}。同理，矩阵 B 分块后会形成 l6 个子矩阵：$32*32$ 的矩阵 B_0-B_{15}。矩阵 A 与矩阵 B 的乘法即被分割成各子矩阵之间的乘法与求和。

由于硬件只完成 $32*32$ 的矩阵乘法，矩阵分块后，以 $32*32$ 矩阵乘法为结果的求和就全部由软件来完成。表 2 列出了在 $n=32$ 的情况下不同分块矩阵乘法的任务。其中 q 为完成一次完整的矩阵乘法需要进行的 $32*32$ 矩阵乘法次数。求和则为求结果矩阵中每一个元素的过程。

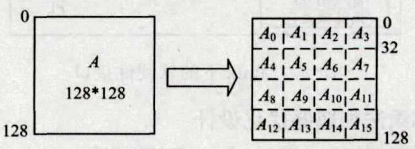

图 3 矩阵分块示意图

表 2 分块矩阵乘法任务

矩阵	分块数(p)	乘法数(q)	求和数(r)
$32*32$	1	1	0
$64*64$	4	$2^3=8$	$2^2=4$
$128*128$	16	$4^3=64$	$4^2=16$
$256*256$	64	$8^3=512$	$8^2=64$

3 软硬件协同加速架构设计

在 MPSoC 架构下，软硬件的并行是在系统双核并行的基础上实现的。本文使用的是 SMP Linux，在系统启动的过程中，它会先启动 CPU0，然后 CPU0 会启动 CPU1。对于应用层的程序，没有办法主动调度 CPU，只能由系统内核完成调度。在该 Linux 系统内核有一套完整的多核调度管理方案，基于该方案，便能完成双核的并行。

3.1 软硬件并行

在 SMP Linux 内核中，文件 smp. h 中的函数 smp_call_function_single 能在系统内核中通过一个 CPU 调度另一个 CPU，实现双核并行；随后，这个 CPU 再调度硬件，实现软硬件并行。在四核或者更多核系统的情况下，同理可以利用 smp. h 中的函数 smp_call_function_many 来实现多核并行。图 4 是软硬件并行的流程图。

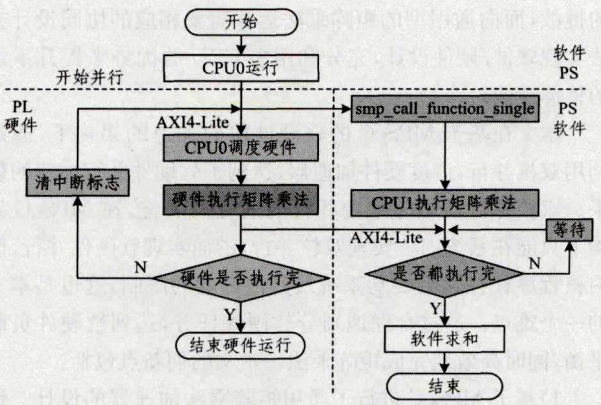

图 4 软硬件并行流程

因为双核调度是在 Linux 系统内核完成的，所以图 4 中的流程是在 Linux 驱动里面实现的。在 Linux 驱动中，CPU0 调度 CPU1 执行矩阵乘法后（无等待），再调度硬件 IP，将矩阵元素传递给硬件，让硬件开始执行矩阵乘法，以此达到软硬件并行。在软件和硬件都完成矩阵乘法后，软件开始对矩阵乘法的结果求和，求取矩阵 C 的各个元素。

在 SMP Linux 中，CPU 之间共享系统资源。共享资源的优势是每个 CPU 对资源数据的访问、处理都很及时高效，不足是多个 CPU 对资源的并发存取会出现竞争，严重的竞争会导致系统崩溃和文件损坏[16-19]。

3.2 软硬件任务划分及负载平衡

对于矩阵乘法 $C=A*B$，其中矩阵 A，B，C 均为 $m*m$ 的矩阵，矩阵分块后，乘法任务分配的最小粒度是 $n*n$ 的矩阵乘法。当 m 较大时，在系统硬件资源允许的前提下可以适当增大 n，以达到对大型矩阵的理想优化效果。对于最小粒度矩阵乘法，软硬件执行的时间分别为 T_k（内核）和 T_h。软硬件的通信时间为 T_c，则矩阵乘法的总时间为：

$$T_a=T_c+T_h \tag{3}$$

当 $m=64$，$n=32$ 时，矩阵分块数 $p=4$，即矩阵 C 分块后会形成 4 个元素 C_0，C_1，C_2，C_3，其中：

$$C_0=A_0*B_0+A_1*B_2 \tag{4}$$

$$C_1=A_0*B_1+A_1*B_3 \tag{5}$$

$$C_2=A_2*B_0+A_3*B_2 \tag{6}$$

$$C_3=A_2*B_1+A_3*B_3 \tag{7}$$

上述 4 个公式中，每个公式中有 2 个矩阵乘法和 1 个矩阵加法。整个矩阵乘法的任务数 $q=8$，求和数 $r=4$。接下来将矩阵乘法任务合理地分配给软硬件，并找到软硬件任务的分割点 d。软硬件任务分配的方式为：软件从矩阵 C_0 开始往后计算矩阵乘法，硬件则从矩阵 C_3 开始往前计算，则有：

$$d=\lfloor q*(T_a/(T_a+T_k))\rfloor \tag{8}$$

由此可知，d 为整数且向下取整。利用软硬件执行最小粒度矩阵乘法的具体时间 T_k 和 T_h 能求出软硬件任务分割点 d。d 的大小确定后，还需要确定 d 具体属于哪个分块矩阵元素，并找到与之对应的具体矩阵乘法。由实验结果可知，$m=64$，$n=32$ 时，$T_a=1000$us，$T_k=513$us，$d=5$。图 5 所示为软硬件任务划分的具体情况。

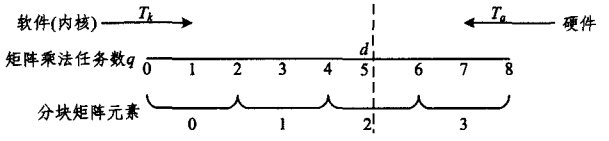

图 5　软硬件任务划分

进行软硬件任务划分后，还需要对软硬件进行负载平衡。由式(8)可知，本文是从软硬件并行时间的角度来求取软硬件任务的划分点 d。而在 MPSoC 软硬件协同加速架构下，软硬件的负载平衡正好需要软硬件的并行时间能最大。所以在该架构下，软硬件任务的分割点 d 也即软硬件负载平衡点。在双核并行调度硬件的情况下，当分割点为 d 时，软硬件并行的时间能够达到最大，剩下的就是软件顺序求和的过程，此时完成整个矩阵乘法的时间最短。依据 d 进行软硬件任务划分，不仅在 MPSoC 架构下实现了软硬件协同加速，而且达到了软硬件负载平衡的效果。

4　实验结果分析

本文的实验平台为 Xilinx 公司的 Zynq-7000 All Programmable SoC，具体开发板为 ZedBoard，Linux 版本为 3.6.0-digilent-13.01，交叉编译器的版本为 gcc version 4.6.1 (Sourcery CodeBench Lite 2011.09-50)。上位机 Linux 系统为 ubuntu 14.04 LTS。Linux 驱动程序和应用程序都采用 C 语言编写。Linux 内核一般不支持浮点数处理，本文实现的矩阵乘法为实点算法。

4.1　硬件矩阵乘法大小的选择

针对不同大小的矩阵乘法，可以选择用硬件实现不同的矩阵乘法。同时，矩阵分块的大小也需要根据硬件 IP 矩阵的大小来确定。对于硬件 IP 的选择，首先需要考虑硬件 IP 的优化情况。表 3 所列为 2.1 节的硬件 IP 在不同优化条件下的执行时间。

表 3　硬件矩阵乘法优化

矩阵	PIPELINE 优化选项									
	无优化		Row		Row+Col		Col		Product	
	时间/us	DSP 利用率/%	时间/us	DSP 利用率/%	时间/us	DSP 利用率/%	时间/us	DSP 利用率/%	时间/us	DSP 利用率/%
4 * 4	4.9	1	0.7	7	0.7	7	0.7	1	2.5	1
8 * 8	37.3	1	5.2	14	5.2	14	5.2	1	15.4	1
16 * 16	292.1	1	41.0	29	41.0	29	41.0	1	102.4	1
32 * 32	2314.9	1	327.8	174	327.8	174	327.8	1	737.3	1
64 * 64	18433	1	不能综合		不能综合		2621.5	1	5570.6	1
128 * 128	127131	1	不能综合		不能综合		20971	1	43254	1

在表 3 中，Row 循环优化的 DSP 利用率最高，执行时间最短，当矩阵大小达到 32 * 32 时，DSP 使用率超过了 100%，虽然执行时间相对 Col 和 Product 来说短很多，但是这种以空间换时间的优化并不可取。而且当矩阵大小达到 64 * 64 时，进行 Row 优化的情况已经不可综合，资源利用率已严重超标。相比 Row 和 Product 的 PIPELINE 优化，Col 循环优化的 DSP 利用率比 Row 低，执行时间处于 Row 和 Product 之间，是最合适的优化方式。

表 4 列出了表 3 中硬件 IP 进行 Col 优化的前提下不同矩阵乘法软硬件实现的时间。结合表 4 中的数据，在 Col 优化不变的情况下，将矩阵乘法的执行时间用图 6 直观地表示出来。

表 4　矩阵乘法执行时间

矩阵	执行时间/us			软硬件通信时间 T_c /us	硬件执行总时间 T_a /us
	内核(T_k)	软件(T_s)	硬件(T_h)		
16 * 16	73	314	41	174	215
32 * 32	513	1970	327	673	1000
64 * 64	4074	15225	2621	2607	5228
128 * 128	85193	142683	20971	10650	31621
256 * 256	849583	1172783	167772	42598	210370

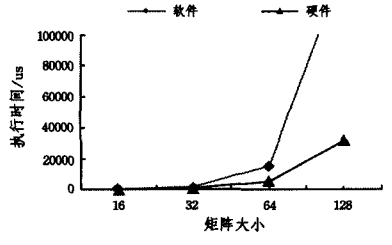

图 6　运行时间随矩阵大小的变化曲线

如图 6 所示，当矩阵达到 16 * 16 时，硬件运行时间(包括通信时间)T_a 小于应用程序 T_s。当矩阵达到 128 * 128 时，硬件运行时间 T_a 小于驱动程序 T_k。由此可见，随着矩阵乘法任务量的加大，使用硬件的收益相对于软件而言越来越大，甚至增大到一定程度时软硬件任务的划分可以由硬件完成所有矩阵乘法，由软件完成所有求和[8]。同时，也需要结合硬件 IP 的资源利用率来合理选择硬件 IP。

由于使用 Linux 系统还会占用部分硬件资源，Linux 系统在 PL 部分会生成一些常用外设的接口，从而在一定程度上限制了硬件实现矩阵乘法的大小。综合考虑使用的硬件平台等多种原因，本文最终选择用硬件完成 32 * 32 的矩阵乘法。

4.2　MPSoC 架构下实现矩阵乘法的加速比

在基于 MPSoC 软硬件协同的架构下，依据 3.2 节中软硬件任务划分的方案，对不同大小的矩阵进行软硬件任务划分，并在软硬件并行的基础上分别实现矩阵乘法。同时，在单核运行的情况下完成矩阵乘法，其中由硬件完成矩阵乘法，由软件完成求和。具体的执行时间和加速比如表 5 所列。

基于表 5 中的加速比数据绘制图 7。如图 7 所示，使用基于 MPSoC 软硬件协同加速的构架实现不同大小的矩阵乘法，获得了不错的加速比，而且矩阵乘法越大，获得的加速比也相对越大。尤其在双核并行的情况下，加速效果更明显。因为相比单核，双核并行的情况下在同一时间会多一个 CPU 执行矩阵乘法。对于 32 * 32 的矩阵乘法，完全由硬件完成，不存在软件的求和过程，因此传统单核调度硬件系统的方法与 MPSoC 下的软硬件协同设计方法的执行方式一样。但是随着乘法规模的增加，多核调度提供了更多的运算资源，通过

合理的应用大大提高了运算效率，对于 256 * 256 的矩阵乘法运算，传统单核调度硬件系统的方法与软件实现相比仅有 2.24的加速比，而本文提出的软硬件协同设计方法可以达到 6.42的加速比。

表 5 矩阵乘法实现的加速比

矩阵	软件/us	单核+硬件/us				软硬件协同(双核并行)/us			
		乘法	求和	总时间	加速比	乘法	求和	总时间	加速比
32 * 32	1970	1000	0	1000	1.97	1000	0	1000	1.97
64 * 64	15225	8035	70	8105	1.88	3031	68	3099	4.91
128 * 128	142683	64033	875	64908	2.20	22025	880	22905	6.23
256 * 256	1172783	512098	11780	523878	2.24	171054	11776	182830	6.42

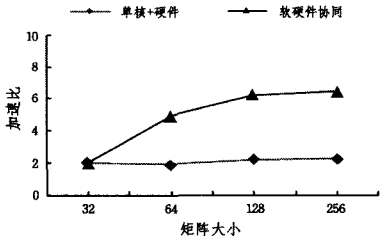

图 7 矩阵乘法运算加速比

在基于 MPSoC 软硬件协同加速的架构下，综合考虑不同大小的矩阵乘法的执行时间以及硬件实现不同矩阵乘法 IP 的收益，对于大型或超大型矩阵乘法，硬件的执行效率明显高于软件，除了要使用硬件加速之外，更应该使用软硬件协同加速的架构，以充分发挥软硬件协同的优势，最大化加速效果。尤其是在 MPSoC 架构下，其拥有多 CPU 资源，CPU 的并行能提高系统软件的整体并行效率，大大提升软件加速比。并且，基于操作系统的软件的控制管理能力更强，对后续整个软硬件协同架构功能的强化与拓展都会起到有效的支撑作用，非常适合对软硬件协同加速进行控制与管理。

结束语 本文提出了一种基于 MPSoC 的软硬件协同加速架构。在该架构下，实现了一系列矩阵乘法的加速算法。可以发现，使用该架构完成矩阵乘法的效率相比一般软件实现矩阵乘法的效率高很多；并可根据实际的矩阵乘法选择不同的硬件 IP，用最少的资源达到最大的加速比。今后将尝试使用 DMA 进行软硬件间的数据传输，减少软硬件通信代价，进一步利用 CPU 资源；探索特殊矩阵乘法在该架构上的加速算法，如稀疏矩阵的乘法，使该架构的应用面更广，实用性更强。

参 考 文 献

[1] LEGALL F. Faster algorithms for rectangular matrix multiplication[C]// Proceedings of Annual Symposium on Foundations of Computer Science (FOCS). Los Alamitos: IEEE Computer Society Press, 2012: 514-523.

[2] BENNER P, REMON A, DUFRECHOU E, et al. Accelerating the general band matrix multiplication using graphics processors [C]// Proceedings of 2014 XL Latin American Computing Conference (CLEI). Los Alamitos: IEEE Computer Society Press, 2014: 1-7.

[3] WU Z C, MAO C, HAN L, et al. Highly Scalable Sparse Matrix Multiplication[J]. Journal of Frontiers of Computer Science and Technology, 2013, 7(11): 973-982. (in Chinese)
吴志川，毛琛，韩蕾，等. 高度可伸缩的稀疏矩阵乘法[J]. 计算机科学与探索，2013，7(11)：973-982.

[4] LIU S P, JIANG X Y, XIAO P, et al. An Efficient Sparse Matrix Multiplier Based on FPGA[J]. Microelectronies, 2013, 43(2): 153-157. (in Chinese)
刘世培，江先阳，肖鹏，等. 一种基于 FPGA 的稀疏矩阵高效乘法器[J]. 微电子学，2013，43(2)：153-157.

[5] MATAM K K, PRASANNA V K. Energy-efficient large-scale matrix multiplication on FPGAs[C]// Proceedings of 2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig). Los Alamitos: IEEE Computer Society Press, 2013: 1-8.

[6] BEAUMONT O, EYRAULD-DUBOIS L, GUERMOUCHE A, et al. Comparison of Static and Runtime Resource Allocation Strategies for Matrix Multiplication[C]// Proceedings of 2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). Los Alamitos: IEEE Computer Society Press, 2015: 170-177.

[7] SALIM M, AKKIRMAN A O, HIDAYETOGLU M, et al. Comparative benchmarking: matrix multiplication on a multicore coprocessor and a GPU[C]// Computational Electromagnetics International Workshop (CEM). 2015. IEEE, 2015: 1-2.

[8] KUMAR V, KUMAR V B Y, SACHIN B P. FPGA based Implementation of M4RM for Matrix Multiplication over GF(2) [C]// Proceedings of 18th International Symposium on VLSI Design and Test. Los Alamitos: IEEE Computer Society Press, 2014: 1-2.

[9] DAMMAK B, BENMANSOUR R, NIAR S, et al. A mixed integer linear programming approach for design space exploration in FPGA-based MPSoC[C]// Proceedings of 2014 24th International Conference on Field Programmable Logic and Applications (FPL). Los Alamitos: IEEE Computer Society Press, 2014: 1-4.

[10] BENNOUR I, SEBAI D, JEMAI A. Modeling sw to hw task migration for MPSOC performance analysis[C]// Proceedings of 2010 5th International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS). Los Alamitos: IEEE Computer Society Press, 2010: 1-6.

[11] ZHAO P, SHEN B L, WANG D W, et al. Research on Task Allocation of Multimedia Application onto Heterogeneous Multiprocessor System-on-Chip[J]. Journal of Computer-Aided Design & Computer Graphics, 2010, 22(10): 1671-1678. (in Chinese)

赵鹏,沈弼龙,王大伟,等. 面向异构 MPSoC 的多媒体应用程序任务分配方法[J]. 计算机辅助设计与图形学学报,2010,22(10):1671-1678.

[12] LI D,HOU Y B,HUANG Z Q,et al. A Fuzzy Dynamic Scheduling Algorithm for Multiple Characteristics of MPSoC System [J]. Journal of Computer-Aided Design & Computer Graphics,2011,23(8):1447-1454. (in Chinese)
李达,侯义斌,黄樟钦,等. 面向 MPSoC 系统多特征的模糊动态调度算法[J]. 计算机辅助设计与图形学学报,2011,23(8):1447-1454.

[13] KARIM M,AMAROUCH M Y. An FPGA-based MPSoC for real-time ECG analysis[C]// Proceedings of 2015 Third World Conference on Complex Systems (WCCS). Los Alamitos: IEEE Computer Society Press,2015:1-4.

[14] ZHANG C,MA Y,LUK W. HW/SW Partitioning Algorithm Targeting MPSOC with Dynamic Partial Reconfigurable Fabric [C]// Proceedings of 2015 14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics). Los Alamitos: IEEE Computer Society Press,2015:240-241.

[15] WICAKSANA A,TANG C M,NG M S. A scalable and configurable Multiprocessor System-on-Chip (MPSoC) virtual platform for hardware and software co-design and co-verification[C]// Proceeding of 2015 3rd International Conference on New Media (CONMEDIA). Los Alamitos: IEEE Computer Society Press,2015:1-7.

[16] Love R. Linux 内核设计与实现(第 3 版)[M]. 陈莉君,康华,张波,译. 北京:机械工业出版社,2011.

[17] 博韦,切萨蒂. 深入理解 LINUX 内核[M]. 陈莉君,张琼声,张宏伟,译. 北京:中国电力出版社,2007.

[18] AAS J. Understanding the Linux 2. 6. 8. 1 CPU scheduler[OL]. http://core. ac. uk/display/24603273.

[19] SAKAMURA K. μ ITRON4. 0 Specification: Ver. 4. 00. 00[R]. TRON Association,2002.

(上接第 8 页)

[47] WANG Y Y,LEE J,MAHAJAN M,et al. Combining statistical and knowledge-based spoken language understanding in conditional models[C]// Proceedings of the Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics,2006:882-889.

[48] KLASINAS I,POTAMIANOS A,IOSIF E,et al. Web data harvesting for speech understanding grammar induction[C]// INTERSPEECH. 2013:2733-2737.

[49] GEMMEKA J F,VAN DE LOO J,DE PAUW G,et al. A Self-Learning Assistive Vocal Interface Based on Vocabulary Learning and Grammar Induction[C]// Interspeech. 2012:831-834.

[50] VAN DE LOO J,DE PAUW G,GEMMEKE J F,et al. Towards shallow grammar induction for an adaptive assistive vocal interface: a concept tagging approach[C]// Natural Language Processing for Improving Textual Accessibility (NLP4ITA) Workshop Programme. 2012:27.

[51] PALOGIANNIDI E,KLASINAS I,POTAMIANOS A,et al. Spoken dialogue grammar induction from crowdsourced data[C]// 2014 IEEE International Conference on Acoustics,Speech and Signal Processing (ICASSP). IEEE,2014:3211-3215.

[52] GEORGILADAKIS S,UNGER C,IOSIF E,et al. Fusion of knowledge-based and data-driven approaches to grammar induction[C]// Interspeech. 2014:288-292.

[53] CHORIANOPOULOU A,ATHANASOPOULOU G,IOSIF E,et al. tucSage: Grammar Rule Induction for Spoken Dialogue Systems via Probabilistic Candidate Selection [C] // SemEval 2014. 2014:668.

[54] GASPERS J,CIMIANO P,WREDE B. Semantic parsing of speech using grammars learned with weak supervision[C]// Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT). 2015.

[55] PARGELLIS A,FOSLER-LUSSIER E,LRR C H,et al. Auto-induced semantic classes [J]. Speech Communication, 2004,43(3):183-203.

[56] PANGOS A,IOSIF E,POTAMIANOS A,et al. Combining statistical similarity measures for automatic induction of semantic classes[C]// IEEE Workshop on Auto-matic Speech Recognition and Understanding (2005). IEEE,2005:278-283.

[57] PALOGIANNIDI E. Using crowdsourcing for grammar induction with application to spoken dialogue systems[D]. Technical University of Crete,2013.

[58] KAISER E C,JOHNSTON M,HEEMAN P A. Profer: Predictive,robust finite-state parsing for spoken language[C]// IEEE International Conference on Acoustics,Speech,and Signal Processing,1999. IEEE,1999:629-632.

[59] DIEKERMA A R,YILMAZEL O,LIDDY E D. Evaluation of restricted domain question-answering systems[C]// Proceedings of the ACL Workshop on Question Aaswering in Restricted Domains. 2004.

[60] FERRUCCI D,LEVAS A,BAGCHI S,et al. Watson: beyond jeopardy![J]. Artificial Intelligence,2013,199:93-105.

[61] LOPEZ V,FERNÁNDEZ M,MOTTA E,et al. Poweraqua: Supporting users in querying and exploring the semantic Web [J]. Semantic Web,2012,3(3):249-265.

[62] LOPEZ V,UREN V,SABOU M,et al. Is question answering fit for the semantic web?: a survey[J]. Semantic Web,2011,2(2):125-155.

[63] BENAMARA F. Cooperative question answering in restricted domains: the WEBCOOP experiment [C] // Proceedings of the Workshop Question Answering in Restricted Domains. ACL,2004:21-26.