第44卷第4期 中国科学技术大学学报 Vol. 44, No. 4
2014年4月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Apr. 2014

文章编号:0253-2778(2014)04-0310-07

支持动态部分重构特性的异构多核体系结构

冯晓静[1,2],李 曦[1,2],王 超[1],陈 鹏[1],周学海[1,2]

(1.中国科学技术大学计算机科学与技术学院,安徽合肥 230027;
2.中国科学技术大学苏州研究院嵌入式系统实验室,江苏苏州 215123)

摘要:嵌入式应用固有的多样性、多变性等特点,及其对系统计算性能的苛刻要求,使嵌入式系统结构面临着严峻的挑战.通过在软硬件协同流程、底层通信接口、并行编程模型及运行环境等方面提供对动态部分重构特性的支持,将动态部分重构技术引入面向服务的异构多核(SOMP)系统,从而在不损失系统计算性能的前提下,有效地提高系统的灵活性,使系统满足更多嵌入式应用的要求.SOMP原型系统已在基于Xilinx Virtex-5 FPGA芯片的开发板上实现.此外,支持动态部分重构特性的SOMP体系结构的正确性及其带来的灵活性优势通过实验得以验证.

关键词:可重构计算;动态部分重构;面向服务体系结构;异构多核系统

中图分类号:TP331.2 **文献标识码**:A doi:10.3969/j.issn.0253-2778.2014.04.008

引用格式:Feng Xiaojing, Li Xi, Wang Chao, et al. A multiprocessor architecture supporting dynamic partial reconfiguration[J]. Journal of University of Science and Technology of China, 2014,44(4):310-316.
冯晓静,李曦,王超,等. 支持动态部分重构特性的异构多核体系结构[J]. 中国科学技术大学学报, 2014,44(4):310-316.

A multiprocessor architecture supporting dynamic partial reconfiguration

FENG Xiaojing[1,2], LI Xi[1,2], WANG Chao[1], CHEN Peng[1], ZHOU Xuehai[1,2]

(1. *School of Computer Science and Technology, USTC, Hefei* 230027, *China*;
2. *Embedded System Lab, Suzhou Institution for Advanced Study, USTC, Suzhou* 215123, *China*)

Abstract: The intrinsic characteristics of embedded applications such as diversity and variability, together with their stringent requirements for computing performance, impose significant challenges on embedded system design. By providing a hardware/software co-design flow, an underlying communication interface, a parallel programming model and the relevant runtime environment, dynamic partial reconfigurable computing (DPR) technology was presented for service-oriented multiprocessor (SOMP) system. The DPR technology can effectively improve the system flexibility without performance loss, enabling the system to satisfy the requirements of more diverse embedded applications. An SOMP prototyping system has been implemented on the development board for the Xilinx Virtex-5 FPGA. A series of experiments were conducted and the results demonstrate the correctness and the resulting flexibility of the proposed architecture.

收稿日期:2013-03-18;**修回日期**:2013-07-16
基金项目:国家自然科学基金(61202053),江苏省自然科学基金(BK2012194)资助.
作者简介:冯晓静,男,1984年生,博士生. 研究方向:可重构计算技术,多核系统设计,形式化验证技术.
E-mail:bangyan@mail.ustc.edu.cn
通讯作者:周学海,博士/教授. E-mail:xhzhou@ustc.edu.cn

Key words: reconfigurable computing; dynamic partial reconfiguration (DPR); service-oriented architecture (SOA); heterogeneous multicore system

0 引言

多核处理器已经成为微处理器体系结构的发展方向.特别是异构多核系统,它集成了多种异构计算单元,可以充分发挥异构计算单元在不同应用领域的优势,为多种类型的应用提供高性能的计算能力,因而成为嵌入式系统领域的研究热点[1].随着嵌入式系统广泛应用于各领域,嵌入式应用对计算系统的要求越来越苛刻.一方面,嵌入式应用对系统计算性能的要求不断提高;另一方面,嵌入式应用固有的多样性和多变性等特点要求嵌入式系统具备一定的灵活性,以适应不断变化的应用需求.在传统的计算系统中,高性能和高灵活性是一对相互矛盾的性能指标,如何同时满足嵌入式应用对系统性能和灵活性的要求是嵌入式系统的设计面临的严峻挑战之一.

可重构计算是近年出现的一种崭新的计算模式[2],它利用可重构逻辑器件的硬件可编程特性,允许针对特定的应用重新配置逻辑器件,改变其功能以满足变化的应用需求.动态部分重构是一种特殊的可重构计算技术,它允许用户只对逻辑器件的部分区域进行重新配置,而其余部分在重构过程中仍然保持正常运行.与普通的全局重构方式相比,动态部分重构具有更高的灵活度和较小的重构开销.

本文拟将动态部分重构技术引入面向服务的异构多核(SOMP)体系结构[3],允许用户针对特定的应用对计算资源(服务体)进行在线定制,在不损失计算性能的前提下提高系统的灵活性,使系统兼有与专用集成电路相当的高计算性能和类似通用处理器计算方式的灵活性.为此,本文就系统设计流程、通信接口与编程模型等相关问题展开研究,构建了支持动态部分重构特性的 SOMP 系统.具体地,本文的贡献体现在以下几个方面:

(Ⅰ)支持动态部分重构特性的系统设计流程.由于缺乏高效、规范的设计方法指导,部分重构技术的普及应用受到了制约.目前还没有任何以书面形式提出的动态部分重构系统软硬件协同设计流程.本文提出一个支持部分重构特性的软硬件协同设计流程,允许软硬件设计人员并行开发,可以有效地降低部分重构系统的设计难度、加快系统设计过程.

(Ⅱ)模块间通信接口.在动态部分重构系统中,重构模块与其他模块之间需要有不受部分重构过程影响的通信接口,以保证重构操作前后模块间通信的正确性.本文针对 SOMP 系统设计了一种支持动态部分重构特性的服务体通信接口.

(Ⅲ)面向服务的并行编程模型.本文提出了面向服务的并行编程模型,并实现了一种支持该编程模型的运行环境,可以隐藏任务调度、计算资源分配和动态部分重构操作等实现细节,因而有助于减轻系统的编程难度、提高程序员编程效率.

我们在基于 Xilinx Virtex-5 FPGA 芯片的开发板上实现了支持动态部分重构特性的 SOMP 原型系统,证明了本文提出的设计流程和服务体通信接口的正确性.此外,我们还通过实验对原型系统的资源使用量和重构性能进行了评估.实验结果证明了动态部分重构特性仅引入较少的计算资源开销,就可以有效地降低系统重构时间、提高系统的性能.

1 相关工作

异构多核可重构计算平台结合了异构多核系统的可重构计算技术的特点,兼有高性能和高灵活性等优点,因而受到了广泛的关注.代表性的异构多核可重构系统包括:Xilinx 公司的 Zynq[4],Cornell 大学的 ReMAP[5]及 IBM 公司的 Cell[6]等.由于针对异构多核可重构计算平台设计方法的发展远赶不上计算平台的性能提升速度,从而限制了异构多核可重构计算系统的应用范围.

一些早期的研究工作,如文献[7-8]等,对可重构系统的编译工具进行了功能性扩展,使得可重构器件可以有效地对特定应用进行性能加速.这些设计方法中,可重构逻辑器件是主控处理器的加速部件,可重构逻辑器件上运行的硬件任务和普通的软件任务在运行机制上存在巨大的差异,这种差异性给计算资源管理、任务间通信以及用户编程等带来不便.

致力于研究如何使软硬件任务实现机制透明化的工作有很多.例如,文献[9]提出多线程编程模型 Hthread,将运行在可重构计算资源上运行的任务抽象为硬件线程,由操作系统对硬件线程和运行在通用处理器上的软件线程进行统一的管理.类似地,

ReconOS[10]对已有的实时操作系统进行扩展，添加对硬件线程概念的支持.这些工作通过为硬件线程和软件线程统一进行抽象，隐藏了硬件任务的底层实现细节，从而有效降低了软件设计的难度，但是，它们不支持系统的动态部分重构特性.

目前，对于部分重构异构多核系统的研究也取得了一些进展[11-14].其中，文献[11]对操作系统的虚拟内存管理功能进行了修改，让可重构协处理器和主处理器共享用户虚拟存储空间以隐藏软硬件任务通信等细节.BORPH[12]通过将可重构计算资源抽象成硬件进程，BORPH 为系统中的软硬件任务提供一致的运行环境.由于这些工作都忽略了对可重构计算资源的运行管理，因此很难实现计算任务的动态分配和自动并行化.文献[13]和[14]都将硬件计算资源抽象成操作系统中的线程，分别提出了基于硬件线程的动态部分重构系统设计框架.它们的运行环境支持硬件线程的动态创建和动态资源分配，有利于实现应用的自动并行化.由于硬件线程通信接口结构上的限制，文献[13]提出的体系结构仅适用于对流式多线程应用进行性能加速；文献[14]使用速度较慢的 PLB 总线进行通信，限制了系统的整体性能.

我们在先前的研究中将异构多核系统中的软硬件任务抽象为服务，并为软硬件服务提供统一的编程接口，提出了具有易编程、高性能等优点的 SOMP 体系结构[3].本文基于 SOMP 体系结构，在软硬件协同设计流程、硬件通信接口、并行编程模型及其运行时环境等方面实现了系统对动态部分重构特性的支持，提高了系统的灵活性，使 SOMP 系统突破了嵌入式应用多样性、多变性等需求的限制，有广阔的应用前景.

2 面向服务的异构多核体系结构设计方法

2.1 软硬件协同设计流程

支持动态部分重构的可编程逻辑器件发展迅速，然而，可重构计算系统的设计方法却一直比较落后，这导致可重构器件无法充分发挥其性能优势，阻碍了可重构计算技术的普及.为此，本文提出了一种适用于动态部分重构系统的软硬件协同设计流程.在该设计流程的指导下，软硬件设计人员可以并行地进行系统开发，这样有利于降低系统设计难度、缩短系统设计时间.设计流程可分为四个阶段，如图 1 所示.

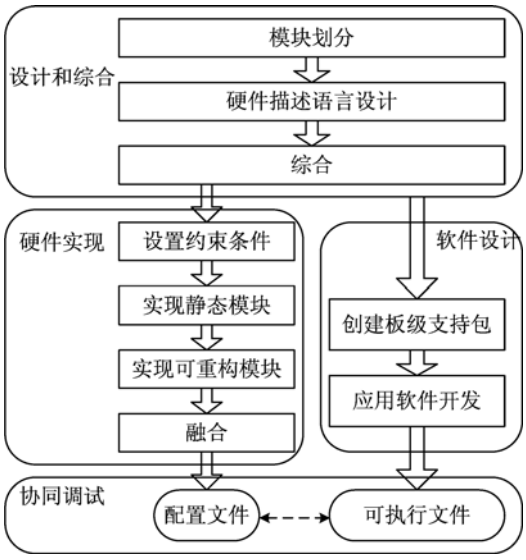

图 1 软硬件协同设计流程

Fig. 1 Software/Hardware co-design flow

（Ⅰ）设计和综合阶段.首先，设计人员需要将应用划分为若干个功能模块.这些模块又分为两类：系统在运行过程不需要重构的模块叫做静态模块，其功能是固定的；另一种叫做可重构模块，每个可重构模块有多个版本的配置文件，每种配置文件实现不同的功能，在系统运行过程中通过加载不同的配置文件实现多种功能切换，加载配置文件的过程就是动态部分重构操作.一个可重构模块可以有多个版本的设计，在不同版本的设计中，该可重构模块与其他模块的接口必需保持一致，这样才能保证重构前后模块间通信的正确性；随后，对各模块进行综合，其中可重构模块的每个设计版本都要分别进行综合，得到各模块的网表文件，网表文件确定了系统的硬件规格，因此，软/硬件设计人员可以并行工作，分别进行硬件和软件设计.

（Ⅱ）硬件实现阶段.设计人员需首先设置设计约束条件，这些约束条件包括可重构模块约束、面积约束、引脚位置约束及时序约束等.其中，可重构模块约束用于指定哪些模块是可重构的；面积约束用于限定静态区域和可重构区域的范围，即静态模块与可重构模块在逻辑器件上的位置；引脚位置约束用于指定静态模块与可重构模块接口的位置，以保证接口在重构操作前后的位置保持一致.然后，需要依次实现各静态模块和可重构模块，即为各模块生成满足所有约束条件的配置文件.可重构模块可能有多个版本的配置文件，但每个静态模块有且仅有

一种配置文件. 最后，要对独立实现的模块进行融合，融合的结果是生成若干个完整配置文件和部分配置文件.

（Ⅲ）软件设计阶段. 软件设计人员需要首先根据系统的硬件规格设计板级支持包，设计人员可以基于板级支持包提供的编程接口进行应用程序的开发. 开发过程中，需要添加一些特殊代码来控制动态部分重构操作，程序最终被编译成可执行文件.

（Ⅳ）联合调试阶段. 前两个阶段生成的配置文件和可执行文件要被加载到逻辑器件上进行协同调试，以验证其正确性. 如果发现错误，则需要修改设计并从引发错误的位置重新执行设计流程，直至可执行文件在可重构硬件平台上正确运行为止.

在 SOMP 系统的设计过程中，核心服务体和控制重构操作的相关功能部件必须位于静态模块中，而计算服务体都应设计成可重构模块. 每个计算服务体有多个版本的设计，每种设计对应一种系统可提供的计算服务，且设计人员需要为计算服务体的每种实现方式设计一个配置文件. 在系统运行前，所有配置文件存放在系统的离线存储器中；系统运行时，通过加载配置文件动态地改变计算服务体的逻辑功能，从而实现对计算服务体的动态重构操作. 重构操作的发生时间和重构过程受应用程序中的相关代码控制. 需要强调的是，该协同设计流程不但适用于 SOMP 系统，而且适用于所有的动态重构系统.

2.2　服务体通信接口

动态部分重构系统通常包含一个静态模块和多个可重构模块，可重构模块与静态模块的接口设计必须遵守一致的规范，这样才能保证重构操作前后模块间通信的正确性. 本文提出了一种针对 SOMP 系统的服务体通信接口，用以连接系统中的核心服务体和计算服务体.

服务体通信接口的内部结构如图 2 所示. 该接口主要由输入/输出缓冲、读写控制逻辑和状态控制逻辑等四个功能部件组成. 其中，输入/输出缓冲的作用是缓存核心服务体发送的输入数据和计算服务体的输出结果，而读写控制逻辑的作用是控制与核心服务体的通信过程. 这三个功能部件必须位于逻辑器件的静态区域.

状态控制逻辑位于可重构区域，其主要作用是控制计算服务体完成相应的计算功能. 它是一个有限状态机，控制计算服务体在读数据、计算、写结果及空闲四个状态间转换，其状态转换如图 3 所示. 图

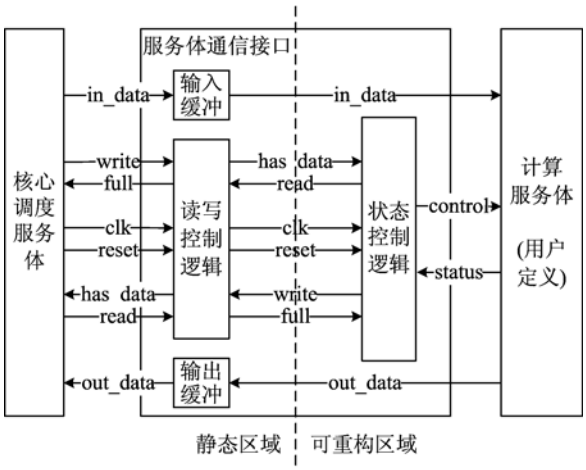

图 2　服务体通信接口

Fig. 2　Servant communication interface

2 中的计算服务体就是用户定义的计算逻辑，用户可以为计算服务体实现不同版本的设计，而计算服务体可以通过动态部分重构操作在各设计版本间进行切换以提供不同的计算服务.

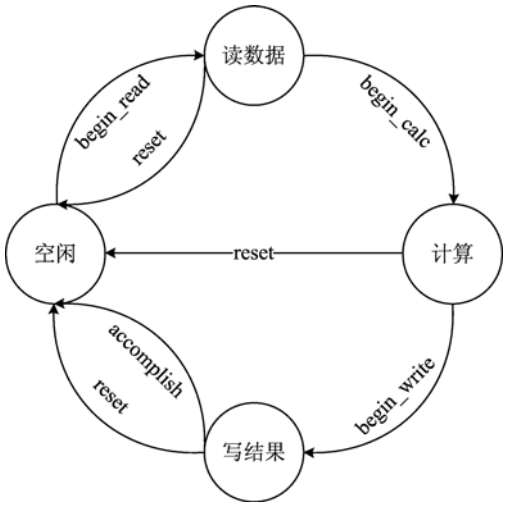

图 3　状态控制逻辑的状态转换图

Fig. 3　State transition graph of state control logics

读写控制逻辑位于静态区域，不受动态部分重构操作的影响，因而能够保证重构操作前后可重构模块与静态模块之间通信过程的一致性. 输入/输出缓冲也位于静态区域，它们在重构操作过程中对数据进行暂存，这是保证重构操作过程中输入/输出数据有效性的关键. 状态控制逻辑位于可重构区域，由于本文已规范了其状态转换模式，因此无需针对不同的可重构模块对状态控制逻辑进行重复设计. 得益于该特性，硬件设计人员只需专注于计算逻辑和相应算法的具体实现，从而有效地降低了系统设计

的复杂度.

2.3 面向服务的并行编程模型

为了降低应用程序的编程难度,本文提出了一种简单易用的面向服务的并行编程模型,并设计了一个支持 SOMP 系统的运行环境.本文提出的编程模型是一种基于标注的并行编程模型,它通过在传统串行程序中加入编译制导语句来指导系统的重构操作和应用的自动化并行.下面给出了该编程模型的一段示例代码.

```
/* Declaration Region */
#pragma svc do_aes_enc(a:int[N] out, b:int[N] in, c:int
[N] in)
#pragma svc do_aes_dec(a:int[N] out, b:int[N] in, c:int
[N] in)
#pragma svc do_idct(a:int[N] out, b:int[N] in)
/* End of Declaration Region */

int A[M][N], B[M][N], C[M][N], KEY[M][N];

/* Parallel Code Block */
#pragma pblock {
  for(i=0; i<M; i++){
    do_aes_enc(A[i], KEY[i], B[i]);
    do_idct(C[i], A[i]);
  }
  reconfig(PM1, DES_ENC); //reconfigure operation
  for(i=0; i<M; i++)
    do_aes_dec(B[i], KEY[i], C[i]);
}
```

示例代码的前半部分是一些编译制导语句,每条语句都有编译制导前缀 #pragma svc.这些语句的作用是声明系统提供的计算服务,声明的内容包括计算服务的名称、每种服务的参数数目和参数类型.用户只需根据声明的格式调用系统提供的计算服务,而不用考虑计算服务的实现细节,面向服务并行编程模型的运行环境会自动解决这些问题.

用户使用编译制导语句 #pragma pblock 定义并行代码块(如示例代码的后半段所示),并通过 do_aes_enc(A[i], KEY[i], B[i])等语句就可以调用系统的计算服务.运行环境动态地检测并行块内计算服务间的依赖关系,对计算服务进行调度,并自动地根据系统负载情况将计算服务分配给相应的服务体,使任务尽可能并行执行.

本文的编程模型还提供重构操作相关的应用编程接口,用于控制系统的动态部分重构过程.例如:示例代码中 reconfig(PM1, DES_ENC)语句的作用就是对可重构模块 PM1 进行重构操作.在系统运行时,一旦遇到重构命令,运行时环境会自动进行栅同步,阻塞相关的计算服务直至重构操作完成,以保证计算服务的正确性.

面向服务并行编程模型实现了异构多核可重构系统中计算任务的自动化并行,这是该并行编程模型最显著的特点.目前广泛使用的并行编程模型(MPI[15]、CUDA[16]等)都将任务并行化的相关问题强加给编程人员.例如,在 CUDA 中,编程人员必须显式地指明如何将完整的应用划分成较小的块;在 MPI 中,程序员必须以手动方式将处理器指派给每个任务并对任务间通信进行管理.本文提出的并行编程模型及相应的运行环境能够自动解决这些问题,隐藏计算资源分配、任务并行化和数据交换等底层实现细节,因此有利于降低编程人员的负担.

3 实验

3.1 原型系统

为了验证 SOMP 体系结构及其部分重构特性,我们在基于 Xilinx XC5VLX110T 芯片的开发板上搭建了原型系统,该原型系统的结构框图如图 4 所示.

系统中有若干个嵌入式通用微处理器 MicroBlaze.其中的一个处理器 MicroBlaze_0 为原型系统的主控处理器,它为面向服务并行编程模型提供运行时环境.其他的 MicroBlaze 处理器都是从处理器.我们对 EEMBC 测试集[17]中的数据加密标准(DES)、高级加密标准(AES)等测试用例进行移植,使之运行在从处理器上以提供软件计算服务.理论上,除主控处理器以外的 MicroBlaze 也具有动态部分重构的特性,但是由于 MicroBlaze 处理器核的重构开销太大,在实际应用中几乎不会对它们进行重构,因此原型系统中的 MicroBlaze 都被设计成静态模块.此外,每个 MicroBlaze 都可以被替换成其他种类的嵌入式微处理器.原型系统中的可重构模块($\text{RM}_1 \sim \text{RM}_n$)是针对特定应用进行加速的 IP 核,分别对应 SOMP 体系结构中的硬件计算服务体.每个可重构模块有多个设计版本,分别实现 AES 编解码或 DES 编解码等功能.每个设计版本都有相应的部分配置文件,所有的配置文件都离线存储在 CF 卡中.

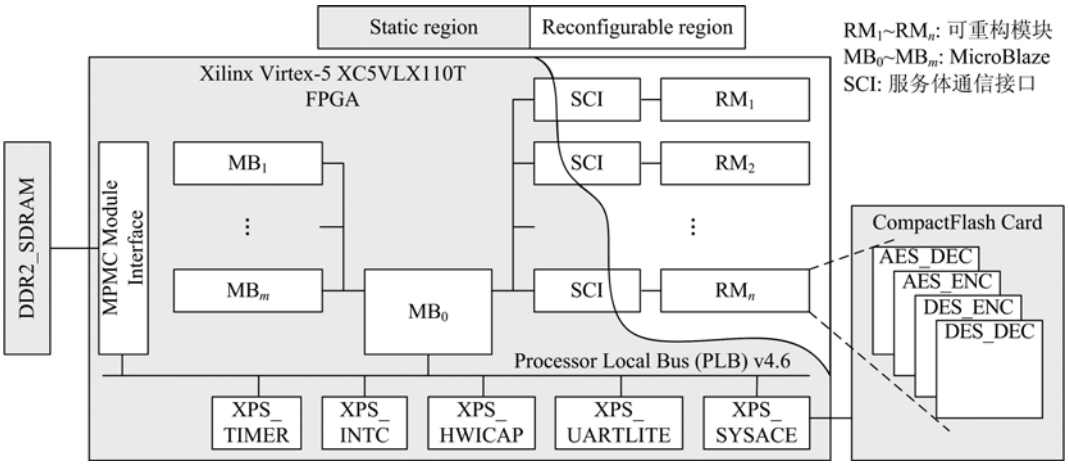

图 4　面向服务异构多核原型系统的结构框图

Fig. 4　Block diagram of SOMP prototyping system

主控处理器通过服务体通信接口与可重构模块连接.此外,它还通过处理器局部总线与定时器、中断控制器、HWICAP 等功能模块连接.其中,HWICAP 的作用是在系统运行时协助主控处理器读写配置文件,以实现动态部分重构操作.

3.2　资源利用率

SOMP 系统需要集成一些额外的功能部件才能支持动态部分重构特性,这些功能部件包括服务体通信接口和 HWICAP 模块.表 1 和表 2 分别列出了上述功能部件的资源消耗情况.从实验数据可以看出,服务体通信接口和 HWICAP 模块对各类片上资源的消耗都处于较低的水平.其中,服务体通信接口对各类资源的消耗不超过片上资源总量的 2.03%,HWICAP 模块对各类资源的消耗不超过总量 1.36%.

表 1　服务体通信接口的资源利用率

Tab. 1　Resource utilization of servant communication interface

计算资源种类	服务体通信接口	计算资源总量	使用率
LUT	1 278	69 120	1.85%
FF	307	69 120	0.44%
SLICE	321	17 280	2.03%
BRAM	3	148	1.86%

表 2　HWICAP 模块的资源利用率

Tab. 2　Resource utilization of HWICAP module

计算资源种类	HWICAP 模块	计算资源总量	使用率
LUT	806	69 120	1.17%
FF	768	69 120	1.11%
SLICE	235	17 280	1.36%
BRAM	2	148	1.35%

3.3　重构开销

对于不支持动态部分重构的计算系统来说,一旦系统设计发生改变,就必须对逻辑器件进行全局重构,即重新配置整个逻辑器件.本文工作的主要贡献是使 SOMP 系统具备动态部分重构的能力.这样,SOMP 系统只需重新配置逻辑器件的部分区域即可适应系统设计的变化.为了证明动态部分重构技术的优势,我们对两种重构方式下的重构数据量和重构时间开销进行了比较,结果如表 3 所示.

表 3　不同重构方式的重构开销

Tab. 3　Reconfiguration overhead of different reconfiguration modes

重构方式	重构数据量/kB	重构时间
全局重构	3 799 (100%)	≈10 s
部分重构 AES 模块	177 (4.66%)	409 ms
部分重构 DES 模块	119 (3.13%)	344 ms

表 3 中,重构数据量就是重构操作使用的配置文件的大小.从表 3 中可以看出,部分重构方式的重构数据量约为全局重构方式的 4.66%.借助原型系统中的硬件计时器和 Xilinx iMPACT 工具,我们分别测量出部分重构和全局重构操作的时间开销.实验结果显示,部分重构的时间开销远小于全局重构的时间开销,表明引入动态部分重构技术可以有效地降低系统的重构开销.

在系统运行的过程中,需要采用一定的重构策略对系统进行动态部分重构操作,以达到加快应用执行速度的目的.用户在设计重构策略时,需要考虑重构操作引入的时间开销等因素.对于单个任务,如

果当前的系统中不存在能够执行该任务的加速功能部件,那么只有当

$$T_{DPR}+\frac{T_{EXEC}}{S}<T_{EXEC} \tag{1}$$

条件成立时,动态部分重构操作才能加快任务执行过程;否则,重构操作反而会降低系统的性能.其中,T_{DPR}表示动态部分重构操作的时间开销,T_{EXEC}表示该任务在通用处理器上的执行时间,S 表示 IP 核等加速功能部件对通用处理器的加速比.根据式(1)可以推出:

$$T_{DPR}<T_{EXEC}\cdot\left(1-\frac{1}{S}\right) \tag{2}$$

当重构时间开销满足式(2)时,才应该对系统进行动态部分重构操作.该结论对重构策略的选择具有重要的指导意义.

4 结论

本文实现了支持动态部分重构特性的 SOMP 系统,该系统不仅具有高扩展性、易编程性等特点,还可以针对特定的应用动态地改变系统,从而更加灵活地使用多样且多变的应用需求.此外,我们还在 FPGA 开发板上实现了 SOMP 原型系统,对系统设计方法的正确性和有效性进行了验证.

参考文献(References)

[1] Wolf W, Jerraya A A, Martin G. Multiprocessor system-on-chip technology[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2008, 27(10):1 701-1 713.

[2] Compton K, Hauck S. Reconfigurable computing: A survey of systems and software[J]. ACM Computing Surveys, 2002, 34(2): 171-210.

[3] Wang C, Zhang J N, Zhou X H, et al. SOMP: Service-oriented multi processors [C]// IEEE International Conference on Services Computing. Washington, USA: IEEE Computer Society, 2011: 709-716.

[4] Zynq-7000 All Programmable SoC[EB/OL]. http://www.xilinx.com/.

[5] Watkins M A, Albonesi D H. ReMAP: A reconfigurable heterogeneous multicore architecture [C]// 43rd Annual IEEE/ACM International Symposium on Microarchitecture. Atlanta, USA: IEEE Press, 2010: 497-508.

[6] Kahle J A, Day M N, Hofstee H P, et al. Introduction to the Cell multiprocessor[J]. IBM Journal of Research and Development, 2005, 49(4/5): 589-604.

[7] DeHon H, Markovsky Y, Caspi E, et al. Stream computations organized for reconfigurable execution [J]. Microprocessors and Microsystems, 2006, 30(6): 334-354.

[8] Chamberlain R D, Franklin M A, Tyson E J, et al. Auto-pipe: Streaming applications on architecturally diverse systems[J]. Computer, 2010, 43(3): 42-49.

[9] Peck W, Anderson E, Agron J, et al. Hthreads: A computational model for reconfigurable devices[C]// Proceedings of International Conference on Field Programmable Logic and Applications. Madrid, Spain: IEEE Press, 2006: 1-4.

[10] Lubbers E, Platzner M. ReconOS: An RTOS supporting hard-and software threads [C]// International Conference on Field Programmable Logic and Applications. Amsterdam, Netherlands: IEEE Press, 2007: 441-446.

[11] Vuletic M, Pozzi L, Ienne P. Virtual memory window for application-specific reconfigurable coprocessors [C]// Proceedings of the 41st annual Design Automation Conference. San Diego, USA: ACM Press, 2004: 948-953.

[12] So H K H, Brodersen R. A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH[J]. ACM Transactions on Embedded Computing System, 2008, 7(2): 1-28.

[13] Wang Y, Yan J, Zhou X G, et al. A partially reconfigurable architecture supporting hardware threads [C]// International Conference on Field-Programmable Technology. Seoul, Korea: IEEE Press, 2012: 269-276.

[14] Ismail A, Shannon L. FUSE: Front-end user framework for O/S abstraction of hardware accelerators [C]// IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines. Salt Lake City, USA: IEEE Press, 2011: 170-177.

[15] Gropp W, Lusk E, Skjellum A. Using MPI: Portable Parallel Programming with the Message Passing Interface[M]. Cambridge, USA: MIT press, 1999.

[16] Stratton J A, Stone S S, Hwu W M. MCUDA: An efficient implementation of CUDA kernels for multi-core CPUs[C]// Languages and Compilers for Parallel Computing, Lecture Notes in Computer Science. Berlin, Germany: Springer, 2008, 5335: 16-30.

[17] EEMBC benchmark suite [EB/OL]. http://www.eembc.org/, 2009.