·体系结构与软件技术· 文章编号：1000-3428(2017)06-0001-10 文献标志码：A 中图分类号：TP393

面向 MPI 集合操作的定制化片上网络

陆思羽，王宏伟，张悠慧，杨广文，郑纬民

（清华大学 计算机科学与技术系，北京 100084）

摘 要： 根据计算趋近数据的原则，提出面向 MPI 集合操作的定制化片上网络设计方法，通过增强现有片上路由器的硬件功能实现 MPI 集合操作在网络层的加速。设计 MPI 归约操作，将其扩展至多种集合操作，并与一种针对确定性路由算法且可动态学习消息传输路径的自适应方法相结合，使集合操作可在扩展后的片上路由器上完成，加速处理过程并减少处理器核负载。此外，提出片上路由器的微体系结构设计方法，比较不同片上网络中扩展后的片上路由器布局并评估相应性能、功耗和片上面积。测试结果表明，与基于软件的最优实现相比，该方法在仅消耗有限功耗与片上面积的基础上，可使 MPI 归约性能提升 6.4～41.7 倍，广播性能提升 15.3～31.2倍，全局归约性能提升 5.4～9.7 倍，收集性能提升 1.3～1.8 倍。

关键词： 片上网络；片上多核处理器；消息传递接口；集合操作；定制化

中文引用格式： 陆思羽，王宏伟，张悠慧，等. 面向 MPI 集合操作的定制化片上网络[J]. 计算机工程，2017，43(6)：1-10，18.

英文引用格式： Lu Siyu，Wang Hongwei，Zhang Youhui，et al. Customized Network-on-Chip Oriented to MPI Collective Operations[J]. Computer Engineering，2017，43(6)：1-10，18.

Customized Network-on-Chip Oriented to MPI Collective Operations

LU Siyu，WANG Hongwei，ZHANG Youhui，YANG Guangwen，ZHENG Weimin

（Department of Computer Science and Technology，Tsinghua University，Beijing 100084，China）

【Abstract】 According to the principle of computations approaching data，this paper proposes a design method of Network-on-Chip（NoC） oriented to MPI collective operations，which focuses on the hardware enhancement of common NoC routers to speed up MPI collective operations on the network layer. It designs MPI reduction，extends it to support more operations and combines it with an adaptive method for the deterministic routing algorithm，which can learn transmission paths of messages dynamically. Thus，enhanced routers can complete message processing in place，which not only speed up the processing procedure but also coalesce messages. The design method for detailed micro-architecture of NoC is presented. Different layout strategies of enhanced routers are compared and the corresponding performance，power consumption and extra chip-area are evaluated. Testing results show that，compared with ideal software-based implementation，the proposed method can improve the reduction performance by 6.4～41.7 times，broadcast by 15.3～31.2，global reduction by 5.4～9.7 times，and gather by 1.3～1.8 times，while the consumption of power and chip-area is limited.

【Key words】 Network-on-Chip（NoC）；Chip Multi-Processor（CMP）；Message Passing Interface（MPI）；collective operation；customization

DOI： 10.3969/j.issn.1000-3428.2017.06.001

0 概述

随着计算机系统结构的发展，越来越多的处理器核被集成到同一个芯片上，并过片上网络（Network-on-Chip，NoC）连接在一起[1]。分布式的片上缓存系统让每个核都有一定容量的局部存储空间，这样的设计让片上多核处理器（Chip Multi-Processor，CMP）与传统的集群计算机在结构上具有一定的相似性，使消息传递机制成为片上编程机制的一个重要选项。

消息传递接口（Message Passing Interface，MPI）是消息传递模型的标准之一，通常用于在分布式内

基金项目： 国家“863”计划项目（2013AA01A215）。

作者简介： 陆思羽（1990—），女，硕士研究生，主研方向为计算机体系结构；王宏伟，硕士；张悠慧、杨广文、郑纬民，教授。

收稿日期： 2016-05-16 **修回日期：** 2016-07-20 **E-mail：** lusiyuhere@163.com

存系统上编写并行程序。MPI 提供了多种集合操作[2],如 MPI_Reduce 和 MPI_Allreduce 等。这些集合操作的使用可简化复杂的并行程序,但也消耗了相当一部分通信时间,文献[3]证实这些集合操作占据 MPI 执行时间的 40% 以上。因此,为实现 MPI 集合操作的加速,本文将计算能力集成至路由硬件上完成 MPI 操作,设计原则为:

1)将计算趋近传输中的数据。由于 MPI 操作产生的消息包可以通过 NoC 直接传输(特别是对小数据集),经过多个片上路由器的转发到达目的节点,因此可以在消息传递路径上做数据处理。

2)取得性能提升和资源消耗的平衡。MPI 操作需要较强的处理能力,如归约计算使用浮点运算单元(Float Point Unit, FPU)实现,但是一个 FPU会占据较大的片上面积,将其集成至每个片上路由器中所产生的功耗是不可接受的。因此,如何分布处理能力、获得性能与功耗的平衡是一个关键问题。

根据以上原则,本文进行以下工作:

1)提出一个软硬件结合的方法,在使用确定性路由算法的前提下,使片上路由器自适应学习 MPI 集合消息的传输路径。

2)设计 2 种类型的片上路由器:一种增加路径学习能力,另一种在前者基础上集成 MPI 消息处理功能。这两种片上路由器在片上网络中的布局对性能功耗比有重要影响,因此,本文提出 3 种不同的布局,并给出各个操作相应的性能模型。

3)实现一个周期精确的模拟器,并针对扩展后的片上路由器进行功能模拟和 RTL 级仿真。

1 相关工作

1.1 定制化硬件辅助的 MPI 加速

在片上网络领域,文献[4]提出了一个通过集成总线来提高点对点及广播方式性能的定制化 NoC,并设计了一个 MPI 引擎,该引擎与每个核相连,实现了基本的 MPI 原语,以此减轻处理器核的负载。但是这个工作并不支持集合通信。

与本文工作相似的一个工作是文献[5],该工作通过添加对归约与广播操作的硬件支持扩展了 FPGA 上的 NoC,在网络上进行这两种 MPI 操作。但是该工作只能在特定的 FPGA 互联结构和拓扑上实现(BEE3);而本文工作使片上路由器具有自适应地学习路径的能力,扩展性更强。另一个相似的工作是 SMART-FanOut/FanIn[6]:在 NoC 上支持消息分散/聚集的文献[7-10]的基础上,完成了 MPI 归约操作的实现。但是该研究没有给出进一步的微体系结构设计或优化的布局。

1.2 片上多核处理器上的集合通信优化

有不少工作提供了 NoC 上针对多播通信(一对多)和归约通信(多对一)的硬件支持,但大多数工作的目的都是提升缓存一致性(Cache-Coherence, CC)通信的效率,并非针对 MPI。例如,文献[11]针对 Cache 一致性协议提出了归约和多播通信在片上网络中的硬件支持方法;文献[12]提出了一个高效的框架支持确认(Acknowledgement Messages, ACK)消息包的归约,并提出了一个创新平衡自适应多播算法(Balanced, Adaptive Multicast, BAM)。其他支持消息多播的片上路由器设计还有 VCTM[7], bLBDR[8], MRR[9], FANOUT[10]等。文献[10]实现了基于树的多播路由算法,文献[12]更高效地利用了网络带宽和缓冲存储。

在上述工作的基础上,本文将 MPI 集合操作移至网络层实现,提出一个软硬件结合的方法来自适应地学习 MPI 集合消息的传输路径。由于 CC 通信通常对软件透明,上述工作难以实现路径的学习,并且 MPI 消息的操作需要更强的处理能力。因此如何在 NoC 上分布处理能力也是本文要解决的问题。

2 路径学习与 MPI 集合操作的网络层处理

在传统 CMP 上,NoC 仅负责内核间的数据传输,计算相关的任务全部由处理器核负责,这种机制导致了计算与传输的分离。为了加速 MPI 集合操作,本文提出了一个软硬件结合的方法,使扩展后的路由器能够智能地学习消息的传输路径,并在传输过程中完成集合操作,不需要处理器核的参与。这个设计的核心思想是使计算趋近数据,进而减轻核的负载并减少中断,同时也节省了核上实现 MPI 集合操作的软件开销。

扩展后的路由器分为 2 种,一种被称为一级路由器,它在传统路由器的基础上集成了集合处理单元(Collective Processing Unit, CoPU),支持路径的学习与集合操作的计算;另一种扩展后的路由器被称为二级路由器,它在传统路由器的基础上集成了一个简化版的 CoPU(simplified CoPU, sCoPU),仅支持集合操作消息的路径学习,不能进行计算。

2.1 系统概览

系统使用经典的 2D-mesh 拓扑结构的片上网络,为描述方便,假设网络的节点组成为 $N \times N$;每个节点由一个带有本地缓存的处理器核及一个扩展后的路由器组成。每个扩展后的路由器都有 5 个端口,其中 4 个端口(东西南北)与其邻居节点的路由器相连接,另一个端口连接本地处理器核。本文设计使用确定性的路由算法及虚拟通道流控方法。

每个处理器核的一级缓存条被设置为 64 Byte。本文设计限制一个集合操作消息包的最大数据负载为 cache-line 的大小,即可直接支持不大于 64-byte 的小数据集的集合操作,其原因为:1)大数据集可被分割为多个小数据集,继而被本文设计支持;2)小数据集(而不是数据所在的地址)可以直接放入消息包中使之在网络中进行传输,既可以实现在传输过程中的计算,又可以减少访存。另外,一个消息分片(Flit)被设置为 128 位,5 个Flit 组成一个最大负载的消息包。

在消息传递时,一个核会发出一个 MPI 数据包(更确切地说,是发送组成一个数据包的多个有序 Flit)到本地的路由器;本地路由器会通过逐跳(hop by hop)的方式传输数据包。在传统的 NoC 中,路由器只负责数据传输,而本文方法可以在网络层完成集合操作。具体的过程将会在后面的部分进行阐述。

2.2　路径学习方法

路径学习方法的目标是使扩展后的路由器能够自适应地学习集合消息包的传输路径。归约操作是一个典型的 MPI 集合操作,所以,本文的学习方法以归约操作为路径学习的基础,通过模拟归约操作实现的伪归约过程实现路径的学习。学习过程完成后,每个路由器上都保存了相应的路径信息,这个路径信息可被所有集合操作使用。

传输路径的具体表示形式是:每个路由节点保存了归约过程时它需要处理的所有消息包的流入方向及每个方向上消息包的数量。以归约过程为例,系统进行实际归约操作时,通过对比当前状态与保存的路径信息,可以判断一个中间的路由节点是否已经接收了所有应该经过这个节点的数据包:若是,则表示这个节点已经完成了这个节点上的子归约操作,可以发出包含结果的数据包;若否,则继续等待所有消息包到达。

本节首先介绍学习算法的详细原理,然后以归约操作为例,详细阐述归约操作实现过程。在下一节中,本文将其拓展至多种集合操作的实现,使得通过学习算法而建立的路径信息可以用来实现其他集合操作。

学习过程描述如下:当一个并行的 MPI 任务开始执行时,运行时软件会在默认的 MPI 通信域上启动一个伪归约过程,所有处理器核发送一个不带数据负载的特殊归约消息包至根节点。在这些特殊归约消息包的传输过程中,一个一级路由器每接收到一个归约消息包时,都会使相应方向上的接收消息包数量增一,并把该值存储到 RPU 内部的一个状态位阵列中。这个状态位阵列被称作学习状态位阵列(Learned Status-Bit-Array,LSBA)。

图 1 展示了一级路由器的 LSBA 示例。它包含了 5 个域,每个域都记录了相应方向上总共需要接收的归约消息包的数量。从图 1 可以看出,这个路由器会从其西边的路由器接收到 3 个归约消息包,从其北边的路由器接收到 5 个归约消息包,以及从本地处理器核接收一个归约消息包。对于一个有 $N \times N$个节点、mesh 结构的片上网络而言,LSBA 每个域的长度不大于 lb($N \times N$)位即可实现消息包的计数。唯一的例外是记录本地处理器核方向的域:对于一个特定的通信域,由于本文设定一个处理器核最多只能发出一个归约消息包,因此它只有 1 位的长度。如果 1 个核运行 2 个或以上的 MPI 任务并发出多个归约消息包,本文设计假设软件会在本地完成归约操作,合并这些消息包。

东	南	西	北	本地核
0000	0000	0011	0101	1

图 1　一级路由器的 LSBA 示例

记录收到消息包的信息后,另一个重要的问题是对伪归约消息包的处理。在实际的归约操作中,每个一级路由器会对所有接收到的归约消息包进行归约操作,然后仅发出一个包含归约操作结果的包。为了模拟这个过程,路由节点只转发到达的第一个空的归约消息包,而剩下的包在记录了相关信息后被直接丢弃。

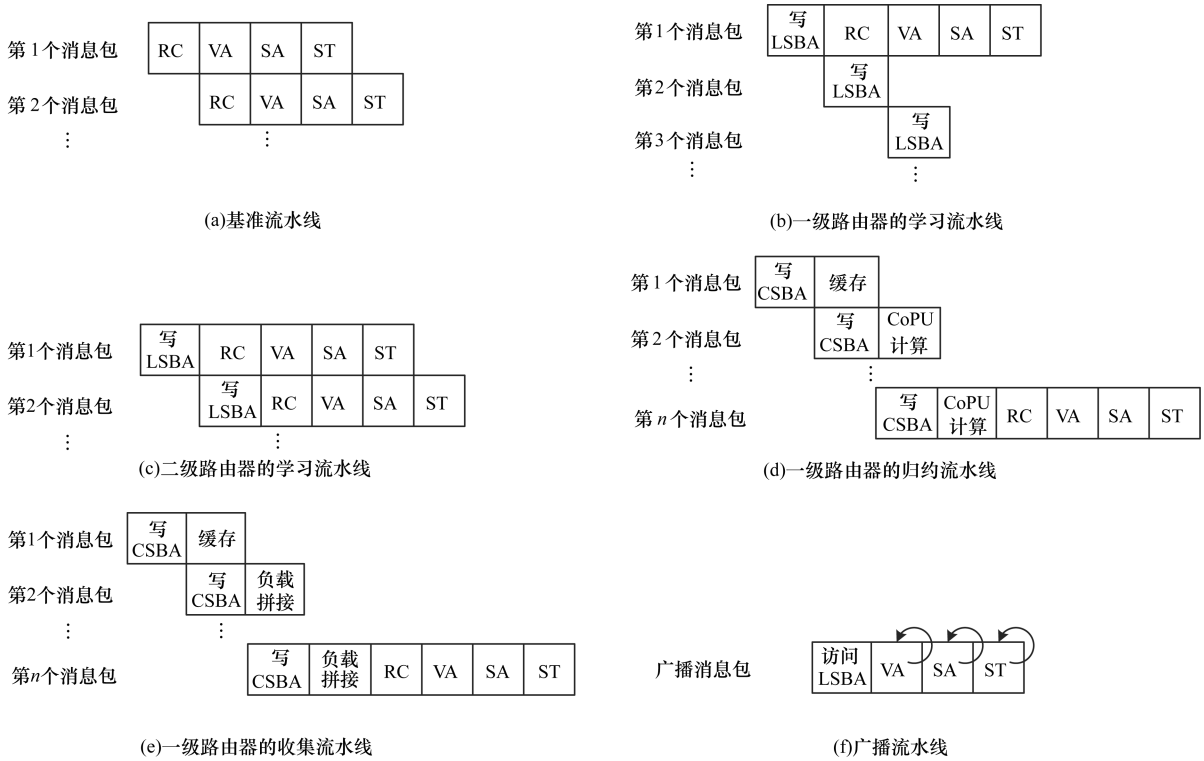
在一个实际的归约操作过程中,一级路由器会实时记录接收到的归约消息包的信息至另一个状态位阵列——当前状态位阵列(Current Status-Bit-Array,CSBA)。CSBA 的结构和 LSBA 大致相同。当一个一级路由器中的 CSBA 与 LSBA 值完全相同时,代表这个路由器已经接收到了所有会经过它的归约消息包,继而发出一个包含归约结果的消息包,并完成归约操作。

每当一个 MPI 任务创建或修改通信域时,会再次启动伪归约过程来获取这个通信域下的消息包传输路径信息。因此,一个拥有 m 组 LSBA 的路由器可以同时支持 m 个不同的通信域。每一个 LSBA 都通过通信域来进行索引。

相较于 LSBA,CSBA 要复杂一些,因为在一个相同的通信域下可能会同时进行多个集合操作,应予以区分,所以软件会给每个集合操作分配一个 ID,这个 ID 包含相应的通信域及一个特定长度的唯一序号。路由器可以使用这个 ID 来找到相应的 CSBA。

2.3　MPI 集合操作实现

MPI 集合操作过程如图 2 所示。



(a)基准流水线 (b)一级路由器的学习流水线 (c)二级路由器的学习流水线 (d)一级路由器的归约流水线 (e)一级路由器的收集流水线 (f)广播流水线

图 2 MPI 集合操作过程示意图

2.3.1 基本过程

本文使用一个经典的虚拟通道路由器作为基础。典型的路由转发过程如图 2(a)所示。传统路由结点的工作流程主要分为路由计算(Routing Computation,RC)、虚拟通道分配(Virtual Channel Allocation,VA)、交叉开关分配(Switch Allocation,SA)、交叉开关传输(Switch Traversal,ST)。

2.3.2 学习过程

一级路由器的学习过程如图 2(b)所示。如上所述,这个过程在一个任务运行在一个新的通信域时发生。因此,在学习过程中,在某种通信域下,扩展后的路由器收到了第一个伪归约过程的消息包后,它会首先分配一个新的 LSBA 表项,并通过写 LSBA(LSBA Write)过程更新相应 LSBA 的信息,即在某个方向上计数增一。后续到来的消息包通过 LSBA 写的步骤记录路径信息后,被该路由器丢弃。

在学习过程中,二级路由器仅记录消息包来源的方向,而不记录数量,且不会丢弃任何包。因此,在二级路由器中,每个 LSBA 表项只有 5 位的宽度。二级路由的学习流水线如图 2(c)所示。

2.3.3 归约操作

一级路由器的归约操作流水线如图 2(d)所示。每一个消息包首先经过 CSBA 写(CSBA Write)阶段,更新相应的 CSBA 表项,然后通过 CoPU 计算(CoPU Com)阶段完成归约。需要注意的是,第1 个消息包到达时,由于没有与其进行计算的操作数,因此第 1 个包仅会被 CoPU 缓存,等待第 2 个包的到达。同时,CoPU 计算阶段的计算时间与消息包切片(Flits)的数量成正比。

假设第 n 个信息包的到达使得 CSBA 与对应的 LSBA 值完全相同,控制逻辑会发出信号通知 CoPU 将这个通信域下的归约结果打包并转发。因此,当消息包全部到达后,CoPU 会生成一个带有计算结果的消息包,并接着通过 RC,VA,SA,ST 等阶段,被注入网络中进行后续传输。

对于二级路由器,进行归约操作时消息包仅会进行普通的传递,没有 sCoPU 的参与(如图 2(a)所示))。

2.3.4 收集操作

归约操作的实现机制可以有效地被拓展到收集操作的实现。收集操作的实现形式与归约操作类似,两者的区别在于负载处理:归约是对负载进行计算,而收集是对负载进行拼接。如图 2(e)所示,对于一级路由器,收集操作过程除了 CoPU 计算阶段被信息负载拼接操作取代外,其他部分与归约操作基本相同。对二级路由器,消息包依然沿着基础流程进行常规转发,不涉及 sCoPU 的参与。

根据收集操作的特点,消息包的负载会随着处理过程的推进而增大。根据本文的设定,可以在定制的 NoC 上直接对小数据集完成操作,并限制最大的消息包为 64 Byte(一级缓存条的大小),因此,当信息包超过 64 Byte 时,路由器就不能直接处理这个操作了。当网络规模扩大时,随着参与操作节点的

增多,每个节点的消息负载大小会受到更多限制。根据这个特点,本文对 Gather 做了进一步的优化,以增加 Gather 操作中消息负载大小的上界:每一个收集消息包只在其经过的第 1 个一级路由器时进行收集操作,完成消息后,路由器会在发出的结果消息包中的某个特定位做标记(消息包中的 Op type 域),使这个消息包在第 2 次经过一级路由器时,只被正常转发,而不进入 CoPU 参与操作;最终的 Gather 操作由根节点处的处理器核完成。

2.3.5　广播操作

广播操作可被看作归约通信过程的反向过程:如果一个路由器在归约操作过程中时接收到某个方向的消息包,那么在进行广播任务的时候,这个路由器就应该向这个方向发送一个广播消息包。当根节点需要广播一个消息包时,会在到达每个节点时都使用 LSBA 的信息获得接下来要转发的方向。

广播操作的一级,二级路由器的流程如图 2(f)所示。LSBA 访问(LSBA access)阶段取代了传统路由器的路由计算阶段。同时,在 CoPU 和 sCoPU 的控制下,有多个域非零时,广播消息包会在路由器内部进行自我复制,发送至多个输出 端口。所有端口发送完成前,原始的广播消息包会一直保存在 CoPU/sCoPU 的 FIFO 中。在 VA,SA 和 ST 阶段,广播消息包会被当做多个单播包处理。

2.3.6　分散操作讨论

在分散操作中,根节点会切分一段数据,将不同数据片段发至不同的接收者。Scatter 在软件上的实现方式是:将网络划分为 2 个子网,同时将数据分为两部分分别发向 2 个子网;由于不同接收者需要接收的消息是特定的,因此数据不是做简单的二分操作,而是要将数据片段与接收节点相对应。按照这种方式,每个子网会继续二分,递归地将所有片段发至特定的接收者[13]。由于每个消息片段都精确地对应了一个接收者,因此本文所提出的机制很难有效地扩展至 Scatter 操作的实现,因为在 Scatter 操作的硬件实现过程中,需要记录更细粒度的路径信息,仅仅记录消息发送的方向和每个方向的消息数是远远不够的;多余的信息记录会导致较高的存储开销,降低性能功耗比。

2.3.7　全局归约操作

全局归约通信可通过执行一个普通的归约操作,然后把结果从根节点广播至这个通信域下的所有节点来实现。

3　扩展后的路由体系结构与节点布局

3.1　扩展后的路由器微体系结构

扩展后的路由器结构如图 3 所示。如上文所述,本文设计将具有 MPI 集合通信处理能力的 CoPU 和 sCoPU 分别集成至经典片上路由器内部的数据通路中,2 种片上路由器分别称为一级路由器和二级路由器。

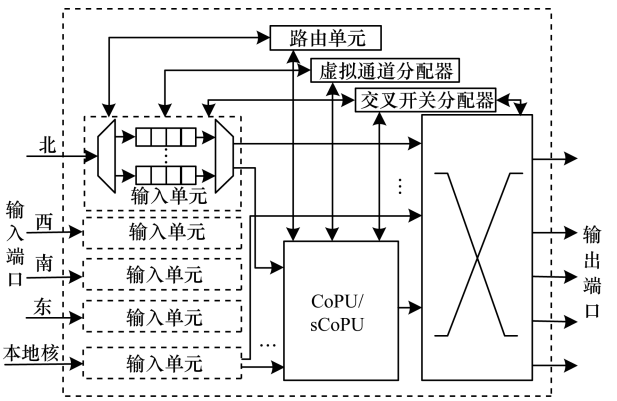


图 3　扩展后的路由器微体系结构

本文设计对输入单元也做了相应的优化:输入单元可以通过消息包的头分片(Head Flit)中的一位信息来判断该消息包是否属于 MPI 集合操作。如果是集合操作类型的消息包,当 CoPU/sCoPU 空闲时,这个消息包会被送到 CoPU/sCoPU,如果 CoPU/sCoPU 忙碌,则在输入端口进行等待。如果不是集合操作消息包,那么这个包会通过另一条数据通路作为普通消息包被直接转发。

头分片的格式如图 4 所示,其中,集合操作类型(Op type)、数据类型(Data type)、通信域(Com)等是针对集合操作消息所特殊设置的域;Src 表示在网络层的源 ID,Dst 表示目的节点 ID。其中,集合操作类型(Op type)表明了 MPI 集合操作的类型:一个特定位为 1 代表这是集合操作所产生的包;其他位不同的值代表了不同的集合操作类型;如果这个域中所有操作位都是 0,表示这个包是学习过程产生的伪归约消息包。

图 4　集合消息包头分片格式

3.1.1　CoPU 微体系结构

如图 5 所示,CoPU 包含如下主要模块:

1)控制单元(Control Unit)。控制单元负责控制集合操作的进行,包括解析消息包、控制计算操作、结果数据打包等。同时包含了一些寄存器来保存消息包的参数,如操作类型、通信域、数据类型等基本信息。

2)简化的浮点运算单元(FPU)。该模块可以完成 2 个双精度浮点数或者整型数据的计算。由于归约计算不需要除法等功能,因此采用精简的 FPU 以节约功耗和片上面积。

3)3 个缓存队列(FIFO),其中 2 个 FIFO 用来保存接收到的消息,另一个 FIFO 用来缓存计算结果;3 个FIFO 都可以向 FPU 输送数据。

4)2 种状态位阵列(LSBA 和 CSBA)。LSBA 和 CSBA 使用内容寻址存储器(Content Addressing Memory,CAM)实现。LSBA-CAM 的结构如图 6(a)所示,集合操作消息包通过通信域索引所需信息,其宽度是 LSBA 和 Com 域宽度的和,深度不大于(lb($n\times n$) +10)位,深度决定了系统支持的通信域数量。CSBA 使用通信域和任务 ID(如图 6(b)所示)索引。CSBA-CAM 的深度表示了 MPI 集合操作能同时支持的最大集合操作任务数量。

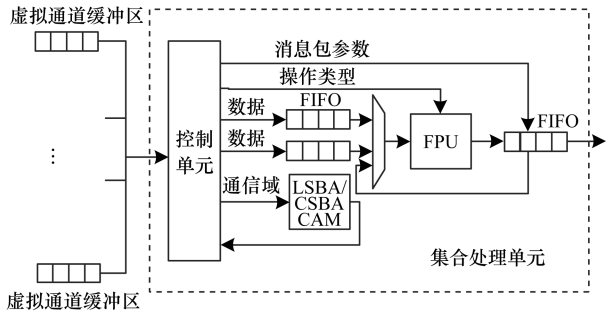


图 5 CoPU 结构

	通信域	LSBA				
		东	南	西	北	本地核
同时支持的最大通信域数量						
	←10位→	←宽度不多于lb($n\times n$)位→				

(a)LSBA-CAM

	通信域+任务ID	CSBA				
		东	南	西	北	本地核
同时支持的最大通信域数量						
	←12位→	←宽度不多于lb($n\times n$)位→				

(b)CSBA-CAM

图 6 一级路由器的 LSBA-CAM 和 CSBA-CAM

3.1.2 sCoPU 微体系结构

本文设计的轻量级路由器内部嵌入了 sCoPU 结构。sCoPU 仅含有一组较小的状态位阵列、一个 FIFO 缓存及控制逻辑。较小的状态位数组仅包含 LSBA,不包含 CSBA,且每个 LSBA 都是 5 位的固定长度,用来表示 5 个方向。sCoPU 结构如图 7 所示。

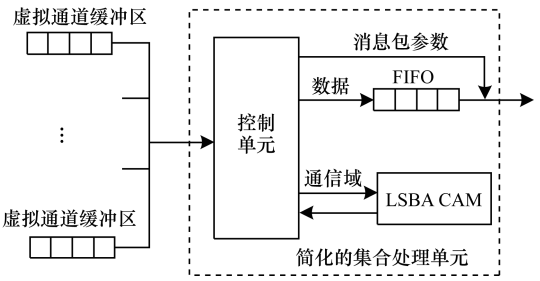


图 7 sCoPU 结构

3.2 2 种路由器在 NoC 上的布局

虽然更多一级路由器可以在数据传输过程中完成尽可能多的数据计算,但一级路由器中的 FPU 能耗高、占据片上面积大,因此,一级路由器与二级路由器的良好布局是提高性能功耗比的关键因素,使静态功耗增加较少的情况下,尽可能地提高性能。

图 8 展示了一个能平衡性能与功耗的示例,这种布局仅有 2/N 个一级路由器。系统将离中心最近的 27 号节点设为根节点。

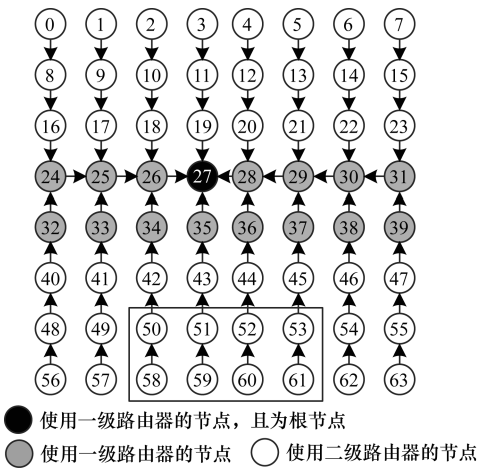


图 8 平衡性能与功耗的布局示例

由于采用 Y-X 路由算法,消息的传输在每列并行;每列至少有一个一级路由器,因此系统在 Y 轴方向获得传输与计算的并行;每列两个一级路由器可以保证每列中从南和从北发出的消息包的并行处理。这个布局策略实现了多个维度的并行。

实际的 MPI 集合操作可能无法指定根节点。在这种情况下,系统依然会把 27 号节点设为根节点,并增加额外的一步:在任务完成后,27 号节点把结果消息包发送到真正的根节点(多对一操作的情况下),或在任务开始时,27 号节点从真正的根节点处接收广播包(一对多的情况下)。这种操作仅增加了传递一个消息包的极少跳数。

此外,实际的应用存在某个通信域不包含任何一级路由器的情况,如图 8 所示(节点 50 ~ 节点 53 和节点 58 ~ 节点 61)。在这种情况下,系统仍按照既定的处理方法发送消息包至根节点(或者从根节点接收包)。尽管这样增加了跳数,但增加的跳数有限,且由此带来的性能提升可以弥补额外开销,这点已经在实验中得以验证。

本文提出了 3 种布局策略,如图 9 所示。其中,策略 1 是一个未经优化的朴素实现。在这种情况下,只有一个路由是一级路由器,这个路由器被放置在中间的位置,并被设置为根节点。策略 2 将根节点所在的一行设为一级路由器。策略 3 已在上文描述。

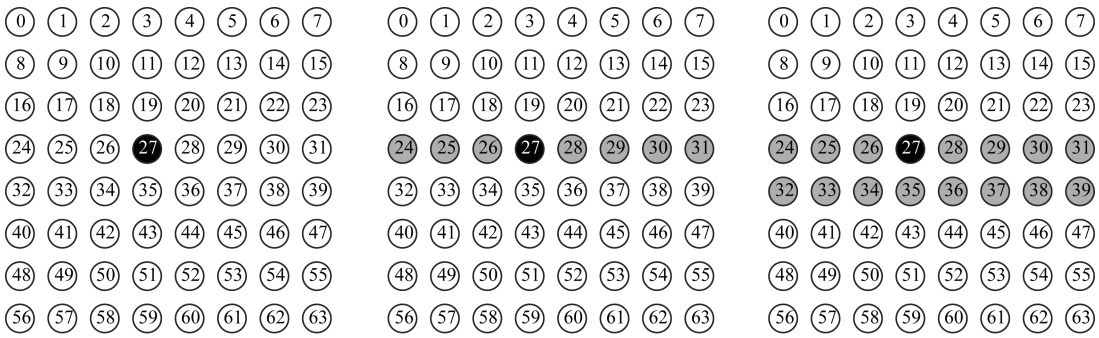


图 9　3 种布局策略示意图

4　实验与结果分析

4.1　实验方式

为了评估本设计的效果,使用 Xtensa Xplore 工具包[14]实现了一个周期精确的 CMP 模拟器。这个模拟器包含多个 Xtensa LX4 核;Xtensa LX4 核是低功耗的简单核,45 nm 工艺下的主频为 1.4 GHz。核间由本设计定制的 NoC 相连。

表 1 列举了本文所有实验使用的 NoC 配置。为了衡量扩展性,网络规模可从 4 × 4 规模的 mesh 网络扩展至 16 × 16 规模。

表 1　NoC 配置

项目		配置
网络配置	拓扑	2D mesh
	路由算法	Y-X 路由
	信道带宽/分片尺寸	128 位(16 Byte)
	消息包的最大尺寸	5 Flits
扩展后的路由器	端口数	5
	每个端口的虚通道数	2
	每个虚通道的 buffer 数	5 Flits
	FIFO 尺寸	5 Flits
	LSBA/CSBA 长	lb($N \times N + 10$)
	LSBA 数	5
	CSBA 数	10
	FPU 延迟	6 cycles

为方便对结果的描述,本文定义完成一次完整的集合操作所花费的时间为硬件延迟。硬件延迟取 5 次连续模拟的时间的平均值。

相应地,软件的实现采用最优的基于 Recursive Halving 和 Recursive Doubling[15]的算法。本文将这个最优算法下完成一次完整的集合操作所花费的时间称为软件延迟。软件延迟由 lb$N$ 性能模型评估获得,所有相关的软件开销,如消息包预处理时间、数据传输时间、计算时间等,都由 CMP 模拟获得。此外,软件实现没有考虑网络冲突。详细的性能模型会在下一节给出。

由于本文使用小数据集,因此采用一次通过(One-pass)的通信协议:参与集合操作的核不需要握手,可直接发出消息包。为公平起见,软件实现也采用了相似的策略。

本文在集合操作消息包的多种负载以及多种网络规模下进行了多次测试。在所有测试中,本文将消息包的大小限制在 64 Byte 以内(即一个缓存行的大小)。

4.2　性能模型

为了分析性能,本文定义了以下参数来描述详细的性能模型:

1)节点编号:假设 mesh 网络的规模为 $n \times n$,因此,网络中共有 n^2 个节点。节点由 0 ~ (n^2 – 1)顺序标号,每个节点都由一个处理器核和一个片上路由器组成。

2)通信开销:由于不考虑网络冲突,定义2 个相邻节点间发送一个消息包所花费的时间为 kT_t。T_t

代表了每字节数据的传输时间,k 代表了消息包的字节数。对于软件实现,这个开销被表示为 $a+kT_t$,a 代表了消息包的软件预处理时间。

3)计算开销:执行一次算术操作的开销被表示为 T_c(每字节数据)。

4)生成 Gather 数据的开销:生成每字节数据负载所花费的时间被表示为 T_g,表示 Gather 操作在 CoPU 中拼接消息的开销。

5)传输 Gather 数据的开销:表示在根节点处从片上路由器传输数据至处理器核所花费的时间。这个开销被表示为 T_f。

4.2.1 跳数

本文通过消息包在网络中的跳数衡量动态功耗。根据理论计算,表 2 展示了不同布局策略下每种操作的跳数。从表 2 可以看出,对于归约操作,采用朴素实现的策略 1 的跳数没有减少,而策略 2 和策略 3 的跳数减少了一半。收集和归约操作具有相同的性能。

表 2 跳数统计

操作	无优化	策略 1	策略 2	策略 3
归约	$n^3/2$	$n^3/2$	$n^3/4+n-1$	$n^3/4-n^2/2+2n-1$
广播	$n^3/2$	n^2-1	n^2-1	n^2-1
全局归约	n^3	$n^3/2+n^2-1$	$n^3/4+n^2+n-2$	$n^3/4+n^2/2+2n-2$
收集	$n^3/2$	$n^3/2$	$n^3/4+n^2/4$	$n^3/4+n$

广播的跳数由 $n^3/2$ 减至(n^2-1)。由于其实现主要依赖于二级路由器,因此对于硬件实现,无论采取哪个策略,跳数都是一样的。

全局归约的策略 2 与策略 3 获得了比 Reduction 更好的跳数降幅:相比无优化的情况减少了 3/4 的跳数,甚至比策略 1 的跳数还少一半。

本文工作所设计的机制通过优化片上路由器,保证了消息包数的减少;传输过程中消息跳数的大幅减少有效地降低了动态功耗,减少传输上的开销。

4.2.2 传输延迟

根据各个操作的实现,传输延迟的性能模型见表 3。软件延迟的模型取自文献[13]。对于硬件实现,模型的延迟取一个消息包从最远的节点传输到根节点所花费的时间(例如,图 8 中的 63 号节点到根节点)。

表 3 延迟的性能模型

操作	软件实现	策略 1	策略 2	策略 3
归约	$\mathrm{lb}(n^2)(a+kT_t+kT_c)$	$nkT_t+n^2kT_c$	$nkT_t+(3n/2)kT_c$	$nk(T_t+T_c)$
广播	$\mathrm{lb}(n^2)(a+kT_t)$	nkT_t	nkT_t	nkT_t
全局归约	$\mathrm{lb}(n^2)(2a+2kT_t+kT_c)$	$a+2nkT_t+n^2kT_c$	$a+2nkT_t+(3n/2)kT_c$	$a+2nkT_t+nkT_c$
收集	$\mathrm{lb}(n^2)a+((n^2-1)/n)\times k(T_t+T_g)$	–	$a+nT_f+(k/2)(n+n^2)T_t+k(n+n^2)T_g$	$a+2nT_f+k(n^2/4+n-1)T_t+k(n/2+n^2)T_g$

由于软件预处理时间 a 是 T_t,T_c,T_g 的上千倍,因此硬件实现比软件实现延迟要低得多。

对于归约的实现,策略 1 的硬件延迟远大于策略 2,而策略 2 的延迟大于策略 3。由于使用了 Y-X 路由,策略 1 没有实现任何并行;而策略 2 实现了每一列的并行;策略 3 在 X 方向和 Y 方向上都获得了并行。由于策略 3 在多个维度上实现了计算和传输的并行,因此策略 3 获得了最好的性能。

对于收集的实现,对于策略 2 来说,最大的消息包负载 k_{max} 为 $64/n$ Byte,因为每个一级路由器都要合并一整列消息包的负载。对于策略 3,k_{max} 是 $64/(n/2)$ Byte。由于从核获取数据的操作所产生的时间 T_f 包含了软件操作,因此 T_f 比 T_t 和 T_g 要大一些,但仍远小于 a。本文没有实现策略 1 的收集优化。

4.3 实验结果

对于归约、广播、全局归约和收集操作,不同策略、不同负载、不同 NoC 规模下的延迟比较如图 10 所示。

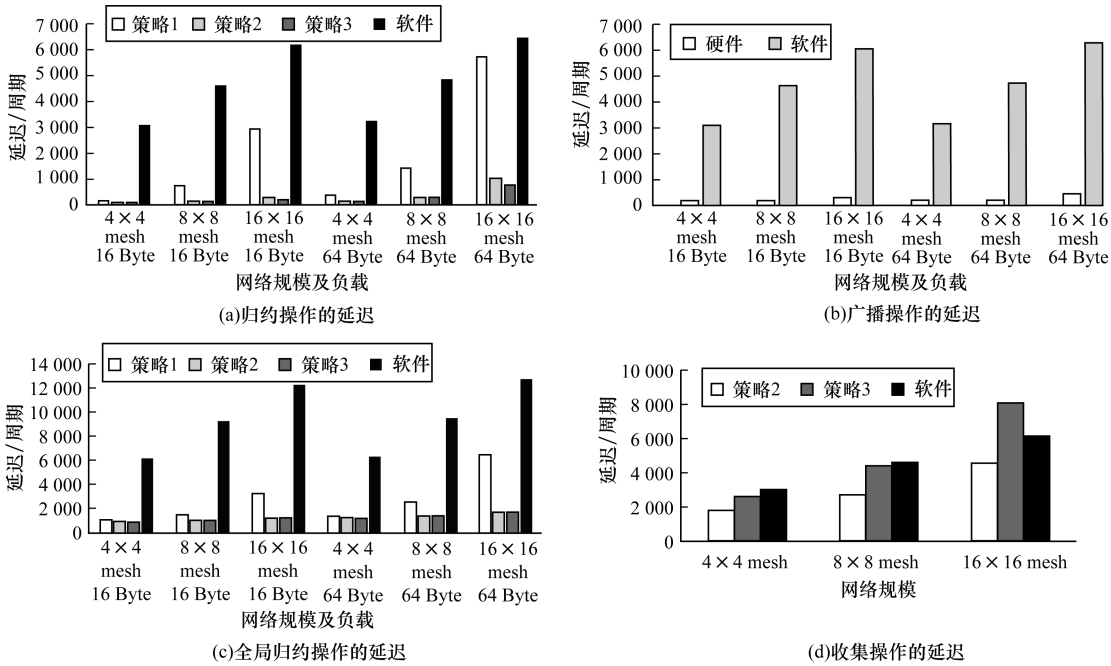


图 10　各种 MPI 集合操作的软硬件延迟比较

4.3.1　归约性能

图 10(a)展示了不同策略、不同负载、不同 NoC 规模下的归约操作的延迟。从结果可知,3 种策略与软件实现相比,均获得了较好的性能加速(最少的加速为 1.13 倍)。在使用策略 3 的 4×4 的 mesh 网络中,基于硬件的实现的速度是软件实现的 42.3 倍,是所有情况下最好的加速。

在布局策略方面,策略 2 和策略 3 都显著好于策略 1,且策略 3 略好于策略 2。相比策略 1,策略 2 和策略 3 最差的性能提升发生在 4×4 规模网络的情况下,分别是策略 1 的延迟的 45.67% 和 45.05%(16 Byte 负载)。这种情况下的网络规模小,单个一级路由器对小规模网络比大规模网络有更大的影响。随着网络规模的增加,策略 2 和策略 3 的优势越来越明显:对于 16×16 规模的网络来说,策略 2 和策略 3 的延迟分别是策略 1 的 9.28% 和 8.23% 。

在硬件消耗方面,策略 3 中一级路由器的数量是策略 2 中的两倍。考虑到策略 2 与策略 3 的延迟相差不大,因此策略 2 获得了最佳性能功耗比,兼顾了性能的提升与功耗的控制。

在上述实验中,所有核都参与了计算。当一个 Reduction 任务仅包含几个节点,而这几个节点都不包含一级路由器时,实验结果表示这种情况比包含一级路由器的情况(如 18 号 ~21 号节点和 27 号 ~29 号节点)仅多消耗了极少的延迟(大约 1% ~2%)。

4.3.2　广播和全局归约性能

广播的延迟如图 10(b)所示。广播的硬件速度是软件的 15.3 ~31.2 倍。由于广播的实现不依赖于一级路由器,因此不同的硬件策略的延迟是相同的。

全局归约相比软件有 1.97 ~9.91 的性能提升,如图 10(c)所示。对于硬件实现,全局归约和归约的优化效果相似,也是策略 3 略好于策略 2,策略 2 和策略 3 都显著好于策略 1。相比归约,全局归约的硬件加速相对较弱,因为全局归约在模拟的实现上并不全依赖于硬件:在全局归约的实现中,在归约步骤结束后有一个软件操作,这个操作用来启动广播操作的执行,因此导致了延迟的增加。

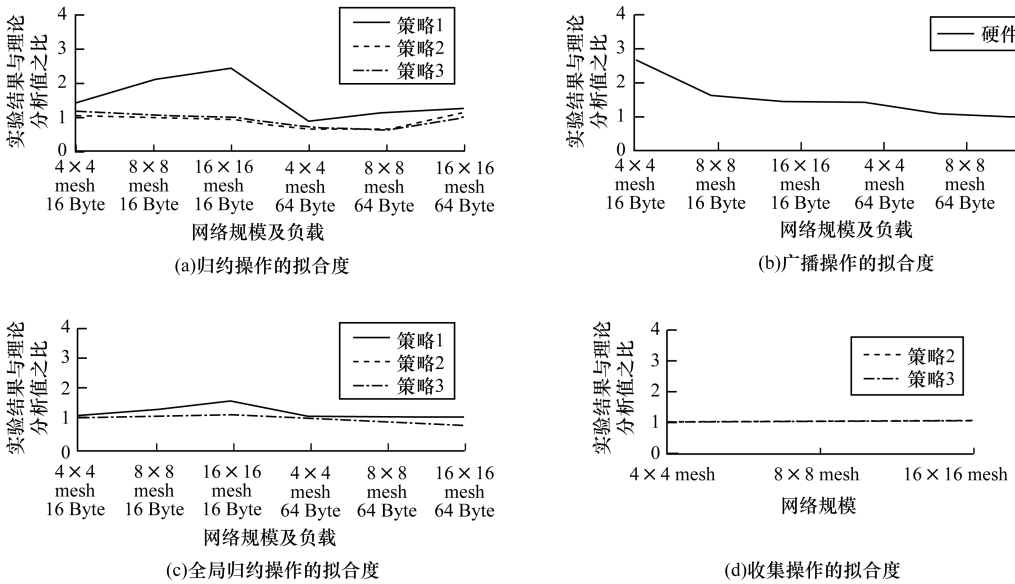
4.3.3　收集性能

图 10(d)展示了收集的延迟。如前文所述,策略 2 下最大的消息包大小分别为 16 Byte(4×4 网络)、8 Byte(8×8 网络)、4 Byte(16×16 网络);对于策略 3,最大消息包大小分别为 32 Byte、16 Byte 和 8 Byte。为了便于比较,所有实现统一使用 4 Byte 的消息负载。

策略 2 比软件实现快 1.3 ~1.8 倍。和归约不同,随着网络规模的增加,策略 3 和软件实现性能相当,甚至要差一点。这是由于策略 3 要使用根节点的处理器核完成最终的 Gather 操作,导致了更多的软件操作。

4.3.4　实验结果与性能模型的拟合度

延迟的拟合度定义为在相同条件下,实验结果与理论分析值的比。图 11 展示了每个集合操作的延迟拟合度。拟合度为 1 代表实验结果完全贴合理论值。根据这个结果,尽管有一些波动,但是拟合度基本都在 1 附近,最高不超过 2.68。由于运行过程中存在的网络冲突和不可预测的情况难以在性能模型中体现,因此较小的波动是十分合理的。



(a)归约操作的拟合度 (b)广播操作的拟合度

(c)全局归约操作的拟合度 (d)收集操作的拟合度

图 11 实验结果与性能模型的拟合度

4.3.5 实现开销

本文通过对扩展后的片上路由器进行硬件语言描述和硬件综合(90 纳米工艺)来评估扩展路由器的实现开销。需要指出的是,芯片面积和功耗都是在芯片所支持的最大频率下获得。

测试结果显示,一个一级路由器比普通片上路由器大 90.8% ,其中,简化的 FPU 占据一级路由器面积的 38% 。在完成一个归约操作时,一级路由器的静态功耗是传统片上路由器功耗的 1.84 倍。对于二级路由器,额外功耗非常有限:仅 1.4% 的额外面积。

在最好的布局策略,即策略 2 中,仅有 $1/n$ 个路由器是一级路由器,其他 $(n-1)/n$ 个片上路由器都是二级路由器,这种布局极大地减少了多余的硬件开销。更具体地说,从片上路由器资源消耗的角度考虑(不包括链路和核),本文的设计分别增加了 24% (4 × 4 mesh)、12.6% (8 × 8 mesh)和 7% (16 × 16 mesh)的片上面积。

5 结束语

本文主要设计了能够在消息传输过程中完成 MPI 集合操作的定制化片上网络,以及一个可以自适应学习消息传输路径的学习算法,展示了片上路由器的微体系结构和各个 MPI 集合操作的流水线。此外,为获取性能提升与额外开销的平衡,还分析了不同布局策略下的性能模型。

测试结果显示,将根结点所在一行设为一级路由器的策略是最优的布局策略,在有限功耗和片上面积增加的情况下,与最优的软件实现相比,该策略布局的系统可以将 MPI 归约操作的性能提升6.4 ~ 41.7 倍,广播提升 15.3 ~ 31.2 倍,全局归约提升 5.4 ~ 9.7 倍,收集提升 1.3 ~ 1.8 倍。

在功耗方面,本文仅使用硬件描述语言综合评估了扩展后的路由器的静态功耗,并通过跳数衡量动态功耗的降低程度,并未给出一个综合的功耗值。下一步将实现 RTL 级别的 Verilog 或 VHDL 代码,并在 FPGA 上进行仿真。

参考文献

[1] Rakesh K, Timothy G M, Gilles P, et al. The Case for Message Passing on Many-core Chips [M]//Hübner M, Becker J. Multiprocessor System-on-Chip. Berlin, Germany: Springer, 2011: 115-123.

[2] Dong Yong, Chen Juan, Yang Xuejun, et al. Low Power Optimization for MPI Collective Operations [C]// Proceedings of the 9th International Conference for Young Computer Scientists. Washington D. C., USA: IEEE Press, 2008: 1047-1052.

[3] Rabenseifner R. Optimization of Collective Reduction Operations[M]//Bubak M, van Albada G D, Sloot P M A, et al. Computational Science-ICCS 2004. Berlin, Germany: Springer, 2004: 1-9.

[4] Huang Libo, Wang Zhiying, Xiao Nong. Accelerating NoC-based MPI Primitives via Communication Architecture Customization [C]//Proceedings of the 23rd IEEE International Conference on Application-specific Systems, Architectures and Processors. Washington D. C., USA: IEEE Press, 2012: 141-148.

[5] Peng Yuanxi, Saldaa M, Chow P. Hardware Support for Broadcast and Reduce in MPSoC [C]//Proceedings of 2011 International Conference on Field Programmable Logic and Applications. Washington D. C., USA: IEEE Press, 2011: 144-150.

(下转第 18 页)

参考文献

[1] 陈 伟,魏 峻,黄 涛. W~4H:一个面向软件部署的技术分析框架[J]. 软件学报,2012,23(7):1669-1687.

[2] 林 闯,陈 莹,黄霁崴,等. 服务计算中服务质量的多目标优化模型与求解研究[J]. 计算机学报,2015,38(10):1907-1923.

[3] Wada H, Suzuki J, Yamano Y, et al. Evolutionary Deployment Optimization for Service-oriented Clouds[J]. Software Practice and Experience,2011,41(5):469-493.

[4] 曹祖凤,孟凡超,周学权,等. 一种多租户 SaaS 应用部署优化算法[J]. 计算机工程,2013,39(10):14-18.

[5] 张晓薇,曹东刚,陈向群,等. 一种网络化移动应用部署方案优化方法[J]. 软件学报,2011,22(12):2866-2878.

[6] Talwar V, Wu Q, Pu C, et al. Comparison of Approaches to Service Deployment[C]//Proceedings of the 25th IEEE International Conference on Distributed Computing Systems. Washington D. C., USA: IEEE Press, 2005: 543-552.

[7] Goldsack P, Guijarro J, Lain A, et al. SmartFrog: Configuration and Automatic Ignition of Distributed Applications[C]//Proceedings of HP Openview University Association Conference. Berlin, Germany: Springer, 2003:1-9.

[8] Li Ying, Qiu Jie, Sun Kewei, et al. Modeling and Verifying Configuration in Service Deployment[C]//Proceedings of IEEE International Conference on Services Computing. Washington D. C., USA: IEEE Computer Society, 2006: 238-248.

[9] Ruiz J L, Duenas J C, Cuadrado F. Model-based Context-aware Deployment of Distributed Systems[J]. IEEE Communications Magazine, 2009, 47(6): 164-171.

[10] Chazalet A, Lalanda P. A Meta-model Approach for the Deployment of Services-oriented Applications[C]//Proceedings of IEEE International Conference on Services Computing. Washington D. C., USA: IEEE Computer Society, 2007:348-355.

[11] Potena P. Optimization of Adaptation Plans for a Service-oriented Architecture with Cost, Reliability, Availability and Performance Tradeoff[J]. Journal of Systems and Software, 2013, 86(3):624-648.

[12] Frey S, Fittkau F, Hasselbring W. Search-based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud[C]//Proceedings of 2013 International Conference on Software Engineering. Washington D. C., USA: IEEE Press, 2013:512-521.

[13] 陈 彬,肖 侬,蔡志平,等. 虚拟机环境下软件按需部署中的预取机制[J]. 软件学报,2010,21(12): 3186-3198.

[14] 林海略,韩燕波. 多租户应用的性能管理关键问题研究[J]. 计算机学报,2010,33(10):1881-1895.

[15] Kecskemeti G, Terstyanszky G, Kacsuk P, et al. An Approach for Virtual Appliance Distribution for Service Deployment[J]. Future Generation Computer Systems, 2011,27(3):280-289.

编辑 陆燕菲

(上接第 10 页)

[6] Krishna T, Peh L S. Single-cycle Collective Communication over a Shared Network Fabric[C]//Proceedings of the 8th IEEE/ACM International Symposium on Networks-on-Chip. Washington D. C., USA: IEEE Press, 2014:1-8.

[7] Jerger N E. Peh L, Lipasti M. Virtual Circuit Tree Multicasting: A Case for On-chip Hardware Multicast Support[C]//Proceedings of 2008 International Symposium on Computer Architecture. Washington D. C., USA: IEEE Press, 2008:229-240.

[8] Rodrigo S, Flich J, Duato J. Efficient Unicast and Multicast Support for CMPs[C]//Proceedings of the 41st IEEE/ACM International Symposium on Microarchitecture. Washington D. C., USA: IEEE Press, 2008:364-375.

[9] Abad P, Puente V, Gregorio J. MRR: Enabling Fully Adaptive Multicast Routing for CMP Interconnection Networks[C]//Proceedings of the 15th IEEE International Symposium on High Performance Computer Architecture. Washington D. C., USA: IEEE Press, 2009:355-366.

[10] Krishna T, Peh L, Beckmann B M, et al. Towards the Ideal On-chip Fabric for 1-to-Many and Many-to-1 Communication[C]//Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture. Washington D. C., USA: IEEE Press, 2011:71-82.

[11] 马 胜. Cache 一致性片上网络路由算法和流控机制优化关键技术研究[D]. 长沙:国防科学技术大学,2012.

[12] Ma Sheng, Jerger N E, Wang Zhiying. Supporting Efficient Collective Communication in NoCs[C]//Proceedings of IEEE International Symposium on High-performance Computer Architecture. Washington D. C., USA: IEEE Press, 2012:1-12.

[13] Chan E W, Heimlich M F, Purkayastha A, et al. On Optimizing Collective Communication[C]//Proceedings of 2004 IEEE International Conference on Cluster Computing. Washington D. C., USA: IEEE Press, 2004: 145-155.

[14] Gonzalez R E. Xtensa: A Configurable and Extensible Processor[J]. IEEE Micro, 2000, 20(2):60-70.

[15] Thakur R, Rabenseifner R, Gropp W. Optimization of Collective Communication Operations in MPICH[J]. Journal of High Performance Computing Applications, 2005, 19(1):49-66.

编辑 金胡考