基于2D-Mesh的容错路由算法

胥大成 樊建席 张书奎
(苏州大学计算机科学与技术学院 苏州 215006)

摘 要 提出一种基于2D-Mesh只使用2条虚通道的容错路由算法,少于需要4条虚通道的Boppana算法,以及需要3条虚通道的Duan算法。算法基于块故障模型,故障块可以是f-ring,也可以是f-chain。无故障时算法用最短路径路由消息,当消息被故障块阻塞时使用绕道策略进行路由。在不重叠和重叠故障区情况下分别给出算法无死锁性的证明过程。

关键词 Mesh,容错,路由,虚通道,片上网络
中图法分类号 TP393 **文献标识码** A

Fault-tolerant Routing Algorithm in 2D-Mesh

XU Da-cheng FAN Jian-xi ZHANG Shu-kui
(Department of Computer Science, Soochow University, Suzhou 215006, China)

Abstract A fault-tolerant routing algorithm for 2D-Mesh that uses only two virtual channels was presented. Previously, Boppana needs 4 virtual channels, and Duan needs 3 virtual channels. The algorithm is based on the block fault model. The fault region can be f-ring and f-chain at the same time. Shortest paths are used for routing if there are no faults, while detour paths are used for blocked messages. We have proved that our algorithm is deadlock-free under the non-overlapping and overlapping situation.

Keywords Mesh, Fault-tolerant, Routing, Virtual channel, NoC

1 引言

VLSI的发展使得工业上可以在一个芯片上集成大量的晶体管,然而传统的基于总线型的通信机制极大地影响了SoC(System-on-Chip)的性能。为了解决SoC中复杂的通信问题,Ahmed Hemani等人提出了NoC的概念[1]。NoC(Network-on-chip)是连接计算单元、存储资源、I/O资源等IP核的交叉开关网络,数据通过交叉开关网络进行传输。目前应用于NoC的拓扑结构有2D-Mesh,Torus,k-ary n-cube,RDT等[2-6]。其中,Mesh是NoC研究中最常见的拓扑结构,它简单且易于实现。

在一给定拓扑结构条件下,NoC的性能决定于NoC采用的交换技术、路由算法等。路由算法决定了每个消息或报文的路由路径。文献中提出了多种基于Mesh的路由算法。维序路由[7]是指消息仅在当前所在维偏移量减少为0时才会进入下一维。XY路由以及eCube路由是典型的维序路由。Glass和Ni在1992年提出转弯模型[8],它提供了一种在给定网络条件下开发最小或非最小路由算法的方法。文献[8]中提出了3种适应性路由算法,即West-First,North-Last,Negative-First。在文献[8]的基础上,Chiu在2000年提出奇偶转弯模型[9],使网络性能更加平均稳定。

在NoC通信网络中,出现故障结点或链路的情况是不可避免的。因此,学者尝试研究容错的路由算法[10-13],即在通信网络出现故障结点或链路的情况下仍能保证其它无故障结点之间正常通信。Wu在2003年提出基于奇偶转弯模型的EX-XY[10]路由。在2D-Mesh中应用Wu提出的构造故障块的方法EX-XY可以很好地容错,但这种容错绕道路由导致网络性能下降。为了解决这一问题,Lin和Huang等在文献[11]提出OAPR路由算法,使得网络性能在容错过程中保持稳定。在文献[12]中,Zou提出了另外一种基于转弯模型的容错路由算法NARCO,它成功地结合了奇偶维序路由及反转奇偶维序路由机制。在文献[13]中,Li和Gu提出了基于人工势场(APF)的容错路由算法,创新地把物理学电场的基本原理应用到NoC路由中。

Linder和Harden首先将虚通道的概念引进容错算法[16],但是他们的算法需要大量的虚通道才能提供容错能力。Boppana和Suresh在文献[14]中引入故障环的概念,为了实现无死锁容错,需要4条虚通道。虚通道的引入虽然简化了容错路由算法的设计,但是带来了新的问题,即需要增加额外的缓冲空间以及复杂的控制逻辑,这极大地提高了制造成本。因此,研究者们尝试在减少虚通道的情况下设计容错路由。在文献[15]中,Duan应用虚拟网络的概念提出了使用

到稿日期:2011-04-16 返修日期:2011-07-15 本文受国家自然科学基金项目(60873047,61070169),江苏省自然科学基金项目(BK2008154)资助。

胥大成(1988—),男,硕士生,主要研究方向为并行计算与分布式系统,E-mail:xudacheng06@163.com;樊建席(1965—),男,博士,教授,博士生导师,主要研究方向为网络与分布式计算、图论、算法设计与分析等。

3 条虚通道的容错路由算法。在文献[6]中，Shih 提出了一种适用于 Torus 的容错路由算法，该算法只需要 3 条虚通道实现矩形故障块的容错。本文把 Shih 的算法成功地移植到 Mesh 网络上，除了能够实现 f-ring 容错外，还能够实现 f-chain[11] 容错。

在虫孔交换中，一个消息报文被分割成多个微片。路由结点输入端和输出端的缓冲器只能存储几个微片。在没有阻塞的情况下，消息报文流水地通过网络；当报文请求的通道忙，则"就地"阻塞消息，致使一条消息占据多个路由器的缓冲器同时阻塞其他消息。因此，路由算法的无死锁性对于网络性能至关重要。

本文第 2 节提供预备知识；第 3 节给出在不重叠和重叠故障区情况下的容错路由算法，并且给出算法无死锁性的证明；最后是结论。

2 准备知识

2D-Mesh 是一个规则的矩形网格，由 $k_1 \times k_0$ 个结点组成，且第 i 维有 k_i 个结点，$i \in \{0,1\}$。每一个结点 X 表示一个路由器，坐标为 (x_1, x_0)，其中 $0 \leqslant x_i \leqslant k_i - 1, i \in \{0,1\}$。若结点 X 与 Y 相邻，那么 $y_1 \in \{x_1+1, x_1-1\}$，$y_0 \in \{x_0+1, x_0-1\}$。连接 X 和 Y 的物理链路表示为 $X \leftrightarrow Y$。图 1 是一个 11×10 的 2D-Mesh。

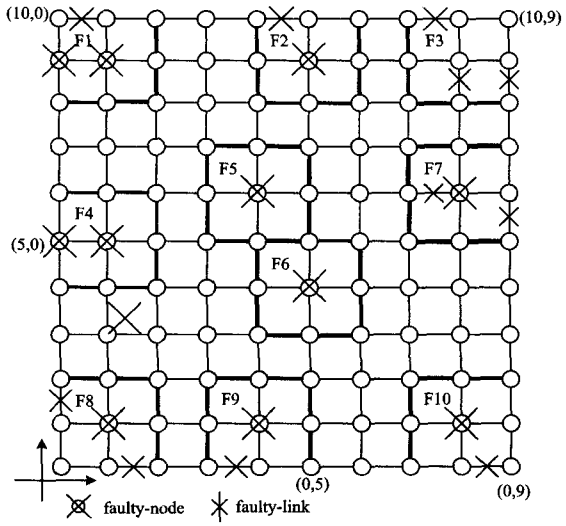


图 1 一个 11×10Mesh

在本文中使用如下约定：

1)故障包含链路故障和结点故障。

2)每一个非故障结点只知道其相邻结点的状态(故障或非故障)。

3)采用块故障模型，每一个块故障区域都是规则的矩形。在矩形边界上没有故障结点或链路；所有的故障结点和链路都被包含在矩形故障区内部。

4)源结点和目的结点不在故障区内，但可以在故障区边界上。

5)本文采用虫孔交换机制，使用两条虚通道。路由过程中禁止 180°转弯。

图 1 中用粗实线相连而成的区域都是故障区 f-region ($F1$-$F10$)。f-region 有两种，即 f-ring 和 f-chain。f-ring 由包围故障链路或故障结点的无故障结点连接而成($F5$-$F6$)。f-chain是处于 Mesh 边界处的故障区($F1$-$F4$，$F7$-$F10$)。根据f-chain位置的不同，称 $F1$ 为 NW-Chain，$F2$ 为 N-Chain，$F3$ 为 NE-Chain，$F4$ 为 W-Chain，$F7$ 为 E-Chain，$F8$ 为 SW-Chain，$F9$ 为 S-Chain，$F10$ 为 SE-Chain。如果两个故障区重叠，那么它们至少有一条公共边。在图 1 中，$F5$ 与 $F6$ 的重叠边为{(5,4)↔(5,5)}。

定义处于矩形 4 个直角处的结点为 Corner node，根据位置不同分为 CN_{nw}，CN_{ne}，CN_{sw}，CN_{se}。对于故障区 $F5$ 而言，$CN_{nw}=(7,3)$，$CN_{ne}=(7,5)$，$CN_{sw}=(5,3)$，$CN_{se}=(5,5)$。f-chain是特殊的 f-ring，因此 f-chain 只包含部分 Corner node。例如，$F1$ 中只含有 CN_{se}，$F4$ 中只含有 CN_{ne}，CN_{se}；$F8$ 中只含有 CN_{ne}。其它 f-chain 与 $F1$，$F4$，$F8$ 类似。若某个结点 X 在 f-region 内部，则记为 $in(X, f\text{-}region)$；若 X 在 f-region 边界上记为 $on(X, f\text{-}region)$；若 X 在 f-region 外部记为 $out(X, f\text{-}region)$。例如，$in((5,0), F4)$，$on((5,3), F5)$，$out((4,8), F7)$。

根据转弯模型[8]，在一个 2D-Mesh 中总共有 8 种直角转变，分别是 ES，EN，WS，WN，SE，SW，NE，NW。Mesh 中从结点 u 到结点 v 的物理通道表示为 $pch(u,v)$。因此，对于 Mesh 中的链路 $u \leftrightarrow v$ 存在两条方向相反的物理通道 $pch(u,v)$和 $pch(v,u)$。令 $c_i = pch(u,v)$，则 $head(c_i)=u$，$tail(c_i)=v$。在虫孔交换中，当一条消息占用一条通道 c_i 时请求使用另一条通道 c_{i+1}，则这两条通道之间存在相关性[7]，即 c_i 依赖于 c_{i+1}，记为 $c_i \sim c_{i+1}$。当网络中出现死锁时，必然存在循环通道依赖[7]。

3 容错路由算法

当 Mesh 中出现故障结点和链路时，为了实现容错，通常需要采用绕道路由机制，即绕过故障区。在这里，每条物理通道 $pch(u,v)$被划分成两条虚通道 $pch^0(u,v)$和 $pch^1(u,v)$，u 和 v 是 Mesh 中任意的相邻结点。虚拟网络 VN_0 由 $pch^0(u,v)$和 $pch^0(v,u)$组成。虚拟网络 VN_1 由 $pch^1(u,v)$和 $pch^1(v,u)$组成。

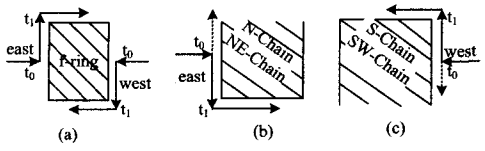
消息 m 的目的结点为 $D(d_1,d_0)$，m 的头微片在结点 $C(c_1,c_0)$处，则定义 m 的路由标记 $R(r_1,r_0)=(d_1-c_1, d_0-c_0)$。$m$ 的类型 $Type(m)$有 4 种：$east$，$west$，$south$，$north$。当 m 产生时，若 $r_0 \leqslant 0$，则 $Type(m)=west$；若 $r_0>0$，则 $Type(m)=east$。若 $r_0=0$ 且 $r_1 \leqslant 0$，则 $Type(m)=south$；若 $r_0=0$ 且 $r_1>0$，$Type(m)=north$。$east$ 和 $west$ 消息统称为 row 消息，$south$ 和 $north$ 消息统称为 $column$ 消息。m 的状态 $Status(m)$有两种：$normal$，$misrouted$。当 m 正常路由时，$Status(m)=normal$，m 采用经典的维序路由(XY)；当 m 阻塞在故障区域时，m 进行绕道路由，此时 $Status(m)=misrouted$。

3.1 无重叠故障区

当消息 m 从源结点产生时为 row 消息，m 首先在 VN_0 中路由，直到 $r_0=0$。当 $r_0=0$ 时，m 转变成 $column$ 消息在 VN_1 中路由。

如图 2 所示，在 VN_0 中绕道的 $east(west)$消息到达 f-ring 上的结点 t_0 时被阻塞，消息状态由 $normal$ 变成 $misrouted$，向结点 t_1 即 $CN_{nw}(CN_{se})$路由。在 t_1 处，消息状态由 $misrouted$ 重新改成 $normal$ 继续路由。绕道的 $east(west)$消息到达 N-Chain 或 NE-Chain(S-Chain 或 SW-Chain)上的结点 t_0 时被阻塞，消息状态由 $normal$ 改成 $misrouted$，向结点 t_1 即 CN_{sw}(S-

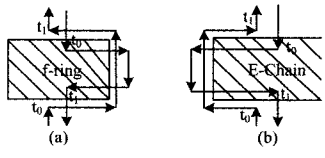
Chain 或 SW-Chain 的 CN_w)路由。在 t_1 处,消息状态由 *misrouted* 重新改成 *normal* 继续路由。



故障区不重叠

图 2　在 VN_0 中绕道的 *east* 和 *west* 消息

如图 3 所示,在 VN_1 中绕道的 *south*(*north*)消息到达 f-ring 上的结点 t_0 时被阻塞,消息状态由 *normal* 改成 *misrouted* 顺时针(逆时针)方向绕道直到结点 t_1 处,在 t_1 处把状态重新改成 *normal* 继续路由。绕道的 *south*(*north*)消息到达 E-Chain 上的结点 t_0 时被阻塞,消息状态由 *normal* 改成 *misrouted* 逆时针(顺时针)方向绕道直到结点 t_1 处,在 t_1 处把状态重新改成 *normal* 继续路由。绕道路由的具体表述见表 1。



故障区不重叠

图 3　在 VN_1 中绕道的 *south* 和 *north* 消息

表 1　绕过不重叠故障区的策略

#No.	Message type	Condition	Message direction
#1	Misrouted east	f-ring, S-Chain, SE-Chain, E-Chain	North
#2	Misrouted west	f-ring, N-Chain, NW-Chain, W-Chain	South
#3	Misrouted south	f-ring, W-Chain	Clockwise
#4	Misrouted north	f-ring, W-Chain	Counter-clockwise
#5	Misrouted east	N-Chain, NE-Chain	South
#6	Misrouted west	S-Chain, SW-Chain	North
#7	Misrouted south	E-Chain	Counter-clockwise
#8	Misrouted north	E-Chain	Clockwise

Algorithm1　无重叠故障区下的容错路由

/ * 消息 M 的头微片处于 $C(c_1,c_0)$,目的结点为 $D(d_1,d_0)$ * /

Step1　更新路由标记 $R(r_1,r_0)=(d_1-c_1,d_0-c_0)$。

Step2　如果 $(r_1,r_0)=(0,0)$,则到达目的结点,返回。

Step3　如果 M 是 *row* 消息并且 $r_0=0$,则(1)如果 $r_1>0$,则把消息类型改为 *north*;(2)如果 $r_1<0$,则把消息类型改为 *south*。

Step4　如果 M 是 *row* 消息,则设置 $vn=0$,否则 $vn=1$。

Step5　如果 M 没有被阻塞,设置 $Status(M)=$ *normal*,否则设置 $Status(M)=$*misrouted* 并根据表 1 决定在 f-ring 或 f-chain 处的绕道路由方向。

Step6　如果 $Status(M)=$*normal*,则按照确定性路由方式在 VN_{vn} 中路由;如果 $Status(M)=$*misrouted*,则根据表 1 决定的方向在虚拟网络 VN_{vn} 中 f-ring 或 f-chain 处绕道路由消息。

下面,我们证明 Algorithm1 是无死锁的。

channels(m)指消息 m 当前占用的通道集合。*message*(#*no*)表示表 1 中编号为 #*no* 的一条绕道消息。若 $m_i,\cdots,m_k$ 消息类型相同,则 $m_i,\cdots,m_k$ 占用的通道 $\bigcup_{j=i}^{k} channels(m_j)$ 可以等价地被一条相同类型的长消息 m 代替,其中 $channels(m)=\bigcup_{j=i}^{k} channels(m_j)$。在以下过程中,*message*(#*no*)都表示长消息。

Lemma1:在虚拟网络 VN_0 中,Algorithm1 在无重叠故障区的情况下是无死锁的。

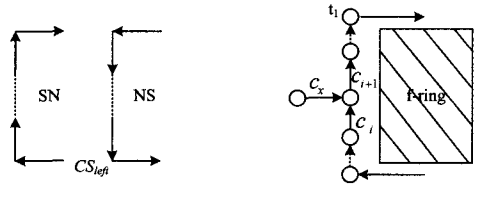
proof:通过反证法来证明。假设存在消息集合 $m_1,m_2,\cdots,m_k$ 产生死锁,那么消息 $m_1,m_2,\cdots,m_k$ 占有的通道集合 $channels(m_i)$ 存在循环依赖。因为 Algorithm1 禁止 180°转弯,那么循环路径中必然存在行通道和列通道。考虑循环路径中最左边的列通道集 CS_{left},也就是说在循环路径中不存在比 CS_{left} 更靠左的通道。很显然,CS_{left} 由一组处于相同列的具有相同方向的列通道组成,如图 4 所示。在 VN_0 中,只有当 $Status(m)=$*misrouted* 时,m 才可能在第 1 维路由。

CASE1:CS_{left} 通道类型为 SN(图 4(a)),则绕道时消息方向为 *north*,且同时存在一 WN 和 NE 方向的直角转弯。从表 1 可知,共有两种可能情形对应于编号 #1 和 #6。反证,若"$\forall c\in CS_{left}\Rightarrow c\in channels(message(\#1))$",很显然 CS_{left} 缺少 WN 方向的转弯,不能构成循环路径。同理,若"$\forall c\in CS_{left}\Rightarrow c\in channels(message(\#6))$",则 CS_{left} 缺少 NE 方向的转弯,也不能构成循环路径。因此,占有 CS_{left} 的消息有两种。

反证,若 Algorithm1 存在死锁,则 $\exists c_i,c_{i+1}$,其中 $c_i\sim c_{i+1}$,并且 $c_i\in channels(message(\#6))$,$c_{i+1}\in channels(message(\#1))$,如图 4(b)所示。因为故障区不重叠,所以 $\exists c_x$ 且 $tail(c_x)=tail(c_i)$。在 $tail(c_x)$ 处 $message(\#1)$ 发生 EN 转弯,则 $out(head(c_x),f-region)$。所以,在 $tail(c_x)$ 处 $message(\#6)$ 必发生 NW 转弯,则 $c_i\not\sim c_{i+1}$,与假设矛盾。因此,Algorithm1 是无死锁的。

CASE2:CS_{left} 通道类型为 NS(图 4(a)),则绕道时消息方向为 *south*。从表 1 可知,共有两种可能情况,对应的编号为 #2 和 #5。证明过程与 CASE1 类似,这里不再重复。因此,Algorithm1 是无死锁的。

综上,Algorithm1 是无死锁的。



(a)等待路径中最左边的列通道集　(b)SN 方向可能的死锁配置

图 4

Lemma2:在虚拟网络 VN_1 中,Algorithm1 在无重叠故障区的情况下是无死锁的。

proof:通过反证法来证明。假设存在消息集合 $m_1,m_2,\cdots,m_k$ 产生死锁,那么消息 $m_1,m_2,\cdots,m_k$ 占有的通道集合 $channels(m_i)$ 存在循环依赖。因为路由算法禁止 180°转弯,那么循环路径中必然存在行通道和列通道。考虑循环路径中处于最上方的行通道 RS_{upper},也就是说在循环路径中不存在比 RS_{upper} 更上方的通道。很显然,RS_{upper} 由一组处于相同行具有相同方向的行通道组成,如图 5(a)所示。在 VN_1 中,只有当 $Status(M)=$*misrouted* 时,M 才可能在第 0 维路由。

CASE1:RS_{upper} 通道类型为 EW(图 5),消息方向为 *west*,且同时存在 NW 和 WS 方向的转弯。

(a)故障区为 f-ring:从表 1 知可能的情形有两种(见图 6(a)中的①②),对应消息的编号为 #4 和 #3。反证,若"$\forall c\in$

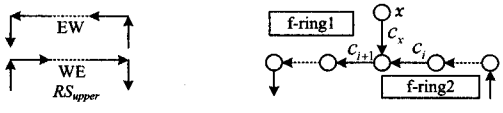
$RS_{upper} \Rightarrow c \in channels(message(\#4))$”，则很显然缺少 WS 方向的转弯，不能构成循环路径。同理，若“$\forall c \in RS_{upper} \Rightarrow c \in channels(message(\#3))$”，则缺少 NW 方向的转弯，不能构成循环路径。因此，占有 RS_{upper} 的消息有两种。

反证，若 Algorithm1 存在死锁，$\exists c_i, c_{i+1}$，其中 $c_i \sim c_{i+1}$，并且 $c_i \in channels(message(\#4))$，$c_{i+1} \in channels(message(\#3))$，见图 5(b)。存在消息 m 经过结点 x 在 $head(c_{i+1})$ 处发生 SW 转弯，则结点 $on(x, f-ring1) \wedge on(head(c_{i+1}), f-ring1) \wedge on(tail(c_{i+1}), f-ring1)$。因为 f-region 不重叠，所以 $on(head(c_{i+1}), f-ring2) \wedge out(tail(c_{i+1}), f-ring2)$。因此，在 $head(c_i)$ 处必发生 WN 转弯，则 $c_i \nsim c_{i+1}$，与假设矛盾。因此，Algorithm2 是无死锁的。

(b)故障区为 f-ring 或 E-Chain(见图 6)，绕道消息方向为 *west*。从表 1 知可能的情形有 4 种。除了图 6(a)中的情形，还有图 6(b)中的③④对应编号为 #7 和 #8 的消息。若 *message*(#7)占有 RS_{upper} 中的通道，则不可能出现 NW 方向的转弯形成循环路径。很显然，*message*(#8)与 *message*(#4)等价，故(b)的证明可以转化为(a)的证明。因此，Algorithm2 是无死锁的。

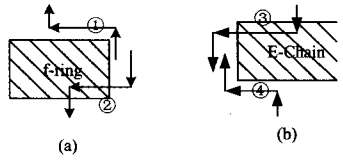
CASE2：RS_{upper} 通道类型为 WE(图 5)，消息方向为 *east*。证明过程与 CASE1 类似，这里不再重复。因此，Algorithm2 是无死锁的。

综上，Algorithm2 是无死锁的。



(a)等待路径中最上边行通道集　(b)WE 方向可能的死锁配置

图 5



(a)　(b)

图 6　可能的从 *east* 向 *west* 路由的消息

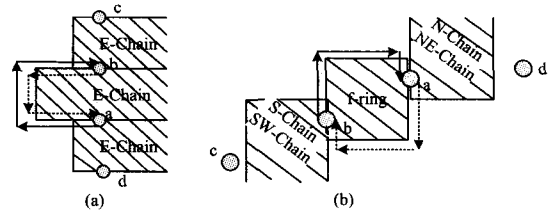
Theorem 1　Algorithm1 在无重叠故障区的情况下是无死锁的。

proof：由 Algorithm1 知，*east* 或 *west* 消息 m 首先在 VN_0 中路由，然后转变成 *south* 或 *north* 消息在 VN_1 中路由。因此，在 VN_1 中路由的消息不会重新进入 VN_0 路由。又由 Lemma1 和 Lemma2 知，在 VN_0 和 VN_1 中 Algortihm1 是无死锁的。因此，Algorithm1 在无重叠故障区的情况下是无死锁的。

3.2　含有重叠故障区

3.1 节已经讨论了无重叠故障区的情况，但如果两个故障区重叠，那么可能出现死锁。在文献[6]中，Shih 讨论了重叠 f-ring 的死锁问题及其应对措施，这里不再重复。本文重点讨论 f-chain 的重叠及 f-chain 与 f-ring 重叠可能引起的死锁问题。

如图 7 的死锁配置，结点 a 向结点 c 发送消息 m_1，结点 b 向结点 d 发送消息 m_2。这里 m_1 和 m_2 都是长消息，根据 Algorithm1 的绕道路由策略，m_1 会被 m_2 阻塞，m_2 同时阻塞 m_1 最终形成死锁。



(a)　(b)

图 7　重叠故障区引入的死锁

在图 8 中，给出如何在重叠的 E-Chain 中路由 *south* 和 *north* 消息避免死锁的策略。在表 2 中给出更加详细的描述。U 和 L 表示上方和下方的 E-Chain。结点 u 和 d 分别表示 U 的 CN_{sw} 和 L 的 CN_{nw}。x_U^{sw} 和 x_L^{nw} 分别表示 U 的 CN_{sw} 的第 0 维坐标和 L 的 CN_{nw} 的第 0 维坐标。如果 u 在 d 的右边，则 $East(u,d)=u$；否则 $East(u,d)=d$。目的结点坐标为 $D(d_1, d_0)$。这里需要特别说明的是，在图 8 中，如果消息路由到 d 点时 $r_1=0$，那么消息将直接向东路由交付目的结点。

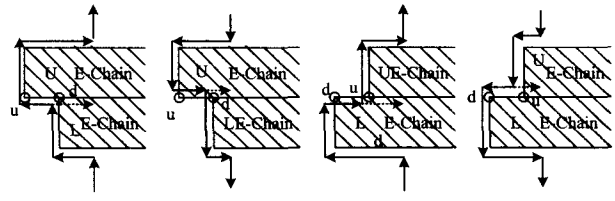


图 8　在重叠的 E-Chain 上绕道的 *south* 和 *north* 消息

表 2　绕过重叠 E-Chain 的策略

#No.	Message type	Condition	Message direction
1	Misrouted south	East(u,d)=d and $d_0 > x_L^{nw}$	Counter-Clockwise along L
2	Misrouted south	East(u,d)=u and $d_0 > x_U^{sw}$	Counter-Clockwise along L
3	Misrouted north	East(u,d)=d and $d_0 > x_L^{nw}$	Clockwise along U
4	Misrouted north	East(u,d)=u and $d_0 > x_U^{sw}$	Clockwise along U
5	Misrouted south or north	$r_1=0$	East

在图 7(b)中，f-ring 与 N-Chain(NE-Chain)以及 S-Chain(SW-Chain)重叠产生死锁。m_1 和 m_2 消息产生时就处于重叠故障区内。避免死锁的方案是 m_1 在 f-ring 与 N-Chain(NE-Chain)垂直重叠区域处在 VN_1 中路由。同理，m_2 在 f-ring 与 S-Chain(SW-Chain)垂直重叠区域处也在 VN_1 中路由。m_1, m_2 一旦通过重叠区域，将再次在 VN_0 中路由。

图 9 中给出了 f-ring，W-Chain，NW-Chain，SW-Chain 重叠时的路由策略，对应于表 3[6]。U 和 L 表示上方和下方的 f-region。结点 u 和 d 分别表示 U 的 CN_{se} 和 L 的 CN_{ne}。x_U^{sw} 和 x_L^{nw} 分别表示 U 的 CN_{sw} 的第 0 维坐标和 L 的 CN_{nw} 的第 0 维坐标。如果 u 在 d 的左边，则 $West(u,d)=u$；否则 $West(u,d)=d$。目的结点坐标为 (d_1, d_0)。

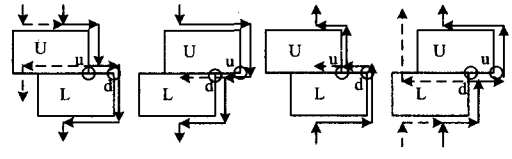


图 9　在重叠的 f-ring 上绕道的 *south* 和 *north* 消息

表 3　绕过重叠 f-ring 的策略

# No.	Message type	Condition	Message direction
1	Misrouted south	West(u,d) and $d_0 > x_L^{nw}$	Clockwise along L
2	Misrouted south	West(u,d) and $d_0 \leqslant x_L^{nw}$	West
3	Misrouted north	West(u,d) and $d_0 > x_U^{sw}$	Counter-Clockwise along U
4	Misrouted north	West(u,d) and $d_0 \leqslant x_U^{sw}$	West

Algorithm2　重叠故障区情况下的容错路由算法

/ * 消息 M 的头微片处于 $C(c_1, c_0)$，目的结点为 $D(d_1, d_0)$，t 为全局变量 * /

Step1　更新路由标记 $R(r_1, r_0) = (d_1 - c_1, d_0 - c_0)$。如果当前结点是一个 corner node，则设置 $t=0$。

Step2　如果 $(r_1, r_0) = (0,0)$，则到达目的结点，返回。

Step3　如果 M 是 *row* 消息，则(1)如果 $r_0 = 0$ 且当前结点在第 1 维方向邻接于两个故障区域，则设置 $t=1$；否则(2)如果当前结点是源结点且在第 0 维方向邻接于两个故障区域，则设置 $t=2$。

Step4　如果 M 是 *row* 消息且 $r_0 = 0$，则(1)如果 $r_1 > 0$，则把消息类型改为 *north*；(2)如果 $r_1 < 0$，则改为 *south*。

Step5　如果 M 是 *row* 消息，那么设置 $vn = 0$。否则，如果 M 是 *column* 消息，设置 $vn = 1$。如果 $t=1$，那么设置 $vn = 0$；否则，如果 $t=2$，设置 $vn = 1$。

Step6　如果 M 没有被阻塞，设置 $Status(M) = normal$，否则设置 $Status(M) = misrouted$ 并根据表 1、表 2 或表 3 决定在 f-ring 或 f-chain 处的绕道路由方向。

Step7　如果 $Status(M) = normal$，按照确定性路由方式在 VN_{vn} 中路由；如果 $Status(M) = misrouted$，根据表 1、表 2 或 3 决定的方向在虚拟网络 VN_{vn} 中 f-ring 或 f-chain 处绕道路由消息。

Lemma3：在虚拟网络 VN_0 中，Algorithm2 是无死锁的。

proof：通过反证法来证明。考虑循环等待路径中处于最左边的列通道 CS_{left}。

CASE1：CS_{left} 通道类型为 SN。尽管 Algorithm2 允许消息 m 由 VN_1 重新进入 VN_0，但 m 必只使用行通道。因此，根据 Lemma1 知，占有 CS_{left} 的消息仍然只有两种，对应于编号 # 1 和 # 6。若故障区不重叠，则由 Lemma1 知 Algorithm2 无死锁。反证，假设 Algorithm2 存在死锁，则故障区必重叠。如图 10(a)所示，考虑 f-ring 与 S-Chain 重叠处最上方的结点 x。x 下方的第一条通道为 c_i，上方的第一条通道为 c_{i+1}，很显然 $c_i, c_{i+1} \in VN_0$，且 $c_i \sim c_{i+1}$。要使 $c_i \sim c_{i+1}$，则必存在一条 *message*(# 1)消息其源结点在重叠处(否则，*message*(# 6)在 x 处发生 NW 转弯，即 $c_i \not\sim c_{i+1}$)。根据 Algorithm2，产生于重叠处的 *east* 消息开始时在 VN_1 中路由，那么 $c_i \notin VN_0$，与假设矛盾。因此，Algorithm2 是无死锁的。

CASE2：CS_{left} 通道类型为 NS(图 4)，证明过程与 CASE1 类似，这里不再重复。因此，Algorithm2 是无死锁的。

综上，Algorithm2 是无死锁的。

Lemma4：在虚拟网络 VN_1 中，Algorithm2 是无死锁的。

proof：f-ring 重叠不会造成死锁，Shih 已经证明。W-Chain，NW-Chain，SW-Chain 路由策略与 f-ring 一致，因此不可能产生死锁。N-Chain，NE-Chain 和 S-Chain，SE-Chain 位置特殊，也不可能发生死锁。因此，本文主要讨论 E-Chain 与 E-Chain，E-Chain 与 f-ring 的重叠。

对于 E-Chain 与 E-Chain 的重叠，根据 Algorithm2 的绕道路由策略，当 $east(u,d) = d$ 且 $d_0 > x_L^{nw}$ 时绕道的 *south* (*north*)在 d 处路由方向为 *south* (*west*)；当 $east(u,d) = u$ 且 $d_0 > x_U^{sw}$ 时绕道的 *south* (*north*)在 u 处路由方向为 *west* (*north*)。当消息路由到 d 点时，若 $r_1 = 0$，消息直接向东路由交付目的结点。因此，E-Chain 与 E-Chain 的重叠可以转化为 Lemma2 的证明。E-Chain 与 E-Chain 的重叠不会出现死锁。

对于 E-Chain 与 f-ring 的重叠，根据 Algorithm2 的路由策略对应的绕道路由如图 10 的(b)、(c)所示，对应的证明过程同样与 Lemma2 类似。因此，E-Chain 与 f-ring 重叠也不可能引起死锁。

综上，Algorithm2 是无死锁的。

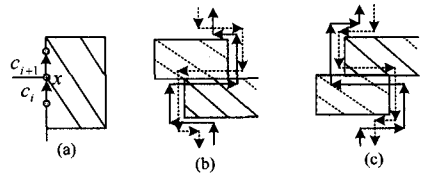


图 10　(a)可能死锁配置；(b)，(c)绕过重叠 f-ring 和 E-chain 的消息

Theorem 2　Algorithm2 是无死锁的。

proof：由 Algorithm2 知，*row* 消息 m 首先在 VN_0 中路由，然后转变成 *column* 消息在 VN_1 中路由。又由 Lemma3 和 Lemma4 知，在各自的虚拟网络中，Algortihm2 是无死锁的。

尽管在 Algorithm2 中 VN_0 中的消息能够进入 VN_1，VN_1 中的消息也能进入 VN_0，但对于 2D-Mesh 而言，在第 1 维，VN_1 中通道可能依赖于 VN_0 中的通道(即产生于重叠处的 *row* 消息在 VN_1 中绕道通过竖直重叠区域的最后一交点时)，但没有 VN_0 中的通道依赖于 VN_1 中的通道；同理，在第 0 维，VN_0 中的通道可能依赖于 VN_1 中的通道(即 *row* 消息在 $r_0 = 0$ 时刚好处于两重叠水平区域处在 VN_0 中绕道通过重叠区域的最后一交点时)，但没有 VN_1 中的通道依赖于 VN_0 中的通道。所以，在 VN_0 和 VN_1 之间，不可能形成死锁。

因此，Algorithm2 是无死锁的。

结束语　本文提出了应用于 2D-Mesh 的容错路由算法，它只使用 2 条虚通道。故障块不仅可以是 f-ring，也可以是 f-chain。当 Mesh 中无故障时，算法可以用最短路径路由消息；当消息被故障块阻塞时，可以使用绕道策略进行路由。本文中给出了路由算法无死锁性的详细证明过程，并且分别对不含重叠故障块和重叠故障块情形进行了讨论。Boppana 和 Duan 提出的容错路由分别需要 3 条和 4 条虚通道，都高于本文中的 2 条虚通道。增加虚通道意味着增加更多的缓冲区空间以及复杂的逻辑控制。根据已有研究，拥有虚通道的路由节点通常需要 2 到 3 倍于一般路由结点的门电路，并且附加的逻辑电路和缓冲空间使得路由结点更加容易故障。因此，更少的虚通道意味着更少的成本及更高的可靠性。

参 考 文 献

[1] Hemani A, Jantsch A, Kumar S, et al. Network on a chip: an architecture for billion transistor era[C]// NORCHIP 2000. Nov. 2000

[2] Dally W J, Seitz C L. The Torus Routing Chip [J]. Journal of Distributed Computing, 1986, 1(3): 1-19

(下转第 134 页)

题，如含有异常处理结构的程序切片问题。但他们的方法都基于传统的数据流迭代分析法，使得最终的切片结果会因变量作用域的问题变得不够准确。

结束语 目前主要的编程语言，如C、C++、Java、VB等都存在变量因作用域问题而被注销、隐藏和覆盖的问题，在进行数据流分析时若采用编译领域中的传统迭代分析法，会造成部分数据流信息的丢失。如果将这样的数据流分析结果应用于各种软件工程任务中，如程序切片、程序测试等，则最终获得的结果可能也是不准确的。本文在传统迭代分析方法基础上提出基于变量作用域的数据流分析方法，在数据流方程中引入两个新的集合分别用于记录变量因作用域的改变而被隐藏和重新可见的信息，解决了数据流信息因作用域变化而丢失的问题。最后利用我们开发的程序分析工具OOAnalyser，分别将传统的和本文提出的数据流分析方法应用于程序切片中。实验结果表明，我们的方法得到的切片结果更加准确。今后，将进一步研究将所得的程序切片结果应用于各种软件工程任务中以便解决实际的问题，如程序切片在测试数据生成中的应用、在错误定位中的应用等。

参 考 文 献

[1] Fosdick L D, Osterweil L J. Data Flow Analysis in Software Reliability[J]. ACM Computing Surveys, 1976, 8(3): 305-330

[2] Alfred V A, Monica S L, Ravi S, et al. Compilers Principles, Techniques and Tools(第2版)[M]. 赵建华，郑滔，戴新宇，译. 北京：机械工业出版社，2009：382-402

[3] 李必信. 程序切片技术及其应用[M]. 北京：科学出版社，2006：17-32

[4] Allen F E. Control flow analysis[J]. Proceedings of a symposium on Compiler optimization, 1970, 5(7): 1-19

[5] Weiser M. Program slicing[C]// Proceedings of the 5th international conference on Software engineering, 1981. New York: IEEE Press, 1981: 439-449

[6] Zhang Xiang-yu, Gupta R, Zhang You-tao. Efficient forward computation of dynamic slices using reduced ordered binary decision diagrams[C]// Proceedings of the 26th International Conference on Software Engineering, 2004. Washington: IEEE Computer Society, 2004: 502-511

[7] Dhamdhere D M, Gururaja K, Ganu P G. A compact execution history for dynamic slicing[J]. Information Processing Letters, 2003, 85(3): 145-152

[8] Weihl W E. Interprocedural data flow analysis in the presence of pointers, procedure variables, and label variables[C]// Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 1980. New York: ACM, 1980: 83-94

[9] Naumovich G. Using the Observer Design Pattern for Implementation of Data Flow Analyses[C]// Proceedings of the 2002 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, 2003. New York: ACM, 2003: 61-68

[10] Larsen L, Harrold M J. Slicing Object-Oriented Software[C]// Proceedings of the 18th international conference on Software Engineering, 1996. Washington: IEEE Press, 1996: 95-505

[11] Sinha S, Harrold M J, Rothermel G. System Dependence Graph Based Slicing of Programs with Arbitrary Interprocedural Control Flow[C]// Proceedings of the 21st International Conference on Software Engineering, 1999. New York: ACM Press, 1999: 432- 441

(上接第117页)

[3] Sarbazi-Azad H, Khonsari A, Ould-Khaoua M. Analysis of k-ary n-cubes with dimension-ordered routing[J]. Future Generation Computer Systems, 2003, 19: 493-502

[4] Yang Y L, Funahashi A, Jouraku A, et al. Recursive Diagonal Torus: an interconnection network for massively parallel computers[J]. IEEE Transactions on Parallel and Distributed Systems, 2001, 12(7): 701-714

[5] Karim F, Nguyen A, Dey S. An Interconnect Architecture for Networking Systems on Chips[J]. IEEE Micro, 2002, 22(5): 36-45

[6] Shih J-D. Fault-tolerant wormhole routing in torus networks with overlapped block faults[J]. IEE Proc. Comput. Digit. Tech., 2003, 150(1)

[7] Dally W J, Seitz C. Deadlock-free message routing in multiprocessor interconnection networks [J]. IEEE Transactions on Computers, 1987, C-36: 547-553

[8] Glass C J, Ni L M. The turn model for adaptive routing[C]// Proceedings of the 19th International Symposium on Computer Architecture. May 1992: 278-287

[9] Chiu G-M. The odd-even turn model for adaptive routing[J]. IEEE Trans. on Parallel and Distributed Systems, 2000, 11: 729-738

[10] Wu J. A fault-tolerant and deadlock-free routing protocol in 2D meshes based on odd-even turn model[J]. IEEE Transactions on Computers, 2003, 52: 1154-1169

[11] Lin S-Y, Huang C-H, Chao C-H, et al. Traffic-balanced Routing Algorithm for Irregular Mesh-based on-Chip Networks [J]. IEEE Transactions on Computers, 2008, 57(9)

[12] Zou Y, Pascricha S. NARCO: Neighbor Aware Turn Model-based Fault Tolerant Routing for NoCs[J]. IEEE Embedded Systems Letters, 2010, 2: 85-89

[13] Li Y H, Gu H G. Fault tolerant routing algorithm based on the artificial potential field model in Network-on-Chip[J]. Applied Mathmatics and Computation, 2010, 217: 3226-3235

[14] Boppana R V, Chalasani S. Fault-tolerant wormhole routing algorithms for mesh networks[J]. IEEE Transactions on Computers, 1995, 44(7): 848-864

[15] Duan X M, Zhang D K, Sun X M. Routing Schemes of An Irregular Mesh-based NoC [C]// 2009 International Conference on NSWCTC. vol. 2, Apr, 2009: 572-575

[16] Linder D H, Harden J C. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes[J]. IEEE Trans. Comput, 1991, 40: 2-12