doi:10.11887/j.cn.201702019 http://journal.nudt.edu.cn

调度感知同步数据流建模*

唐 麒,吴尚峰,施峻武,魏急波
(国防科技大学 电子科学与工程学院,湖南 长沙 410073)

摘 要:对流应用系统进行吞吐量分析需要将周期静态顺序调度建模到数据流图中,吞吐量分析效率依赖于数据流图的规模及建模时间。为了提高吞吐量分析效率,提出基于同构同步数据流图的调度感知同步数据流模型及相应建模方法。通过利用应用模型结构特征及周期静态顺序调度,可减少模型中的任务、边和初始符号数目;可以使用已有分析方法对模型进行吞吐量分析。实验结果表明,所提建模方法优于已有方法,可有效提高吞吐量分析效率。

关键词:同步数据流图;调度感知;多处理器;状态空间
中图分类号:TP399 **文献标志码**:A **文章编号**:1001-2486(2017)02-128-06

Modeling of schedule-aware synchronous dataflow

TANG Qi, *WU Shangfeng*, *SHI Junwu*, *WEI Jibo*
(College of Electronic Science and Engineering, National University of Defense Technology, Changsha 410073, China)

Abstract: To analyze the throughput of streaming application systems, it is necessary to model the periodic static order schedule into the synchronous dataflow graph. The throughput analysis efficiency depends on the size of the dataflow graph and the modeling time. To improve the throughput analysis efficiency, a schedule-aware dataflow model and its modeling method were proposed on the basis of the homogeneous synchronous dataflow graph. By exploiting the structure of the application model and the periodic static order schedule, the task number, edge number and initial token number were reduced. Besides, the throughput of the model can be analyzed using available analyzing methods. Experimental results show that the proposed modeling method outperforms the available methods, with the throughput analysis being optimized efficiently.

Key words: synchronous dataflow graph; schedule-aware; multiprocessor; state space

同步数据流图(Synchronous DataFlow Graph, SDFG)[1-2]广泛用于建模现代流应用,包括音/视频编解码、通信协议、软件无线电等。这些应用大多是计算密集型嵌入式应用,对平台计算能力及系统功耗有严格要求,因此通常采用多处理器平台来实现。为了满足这些应用的实时性要求,需要在多处理器上对应用进行合理调度。然而,多处理器上的并行调度问题通常是NP难问题,对大规模问题求取最优解存在计算复杂度高的问题,因此,已有研究工作大多设计启发式算法来解决这个问题,包括一次性启发式算法和基于设计空间搜索的算法。在一些软硬件协同综合设计或需要同时优化系统功耗、面积和实时性等方面的系统设计中,需要不断评估解决方案,根据当前或历史解决方案的质量决定是否终止优化或进行下一步迭代优化。在上述这些算法中,分析评估所获得调度方案的性能极为重要,尤其对于基于设计空间搜索的算法,高效的建模与分析技术可以有效减少设计时间。

Ghamarian等提出了一种无资源约束的基于状态空间搜索的SDFG吞吐量分析方法(状态空间法)[3]。这一方法采用SDFG边上的符号分布及任务已执行次数与时间来表征应用执行状态。这一方法要求SDFG具有强连通性,而执行状态中各元素值的范围是有限的,状态空间也是有限的。经过有限次的状态转移后,将重新遇到已经经历过的状态,因此会形成状态环。获得状态空间中的状态环后,即可以计算出吞吐量。Geilen提出了采用max-plus代数建模SDFG的自同步执行并使用max-plus矩阵特征值的倒数来计算应用吞吐量的方法(max-plus法)[4]。然而,上述吞吐量分析方法没有考虑应用的映射和调度,只适

* **收稿日期**:2015-10-21
基金项目:国家自然科学基金资助项目(61471376)
作者简介:唐麒(1986—),男,湖南益阳人,博士研究生,E-mail:q.tang.andy@qq.com;
魏急波(通信作者),男,教授,博士,博士生导师,E-mail:wjbhw@nudt.edu.cn

应于无资源约束的应用场景。Bambha 和 Kianzad 等提出了将周期静态顺序调度(Periodic Static-Order Schedule, PSOS)建模到 SDFG 中的方法[5-6],基于调度对 SDFG 进行扩展从而得到调度感知的 SDFG。由于调度感知的 SDFG 仍然是 SDFG,而且其自同步执行严格遵从调度所施加的任务分配和排序约束,因此可以采用已有吞吐量分析方法[3-4]进行吞吐量分析。

Bambha 等提出了将 PSOS 建模到处理器间通信(InterProcessor Communication, IPC)图中的 eIPC 模型[5]。由于所有一致的 SDFG 均可转换成同构同步数据流图(Homogeneous Synchronous DataFlow Graph, HSDFG)然后转换成 IPC 图[2],且实际应用均具有一致性,因此这种方法可以应用于所有实际的流应用。这种建模方法需要将 SDFG 转换成 IPC 图;然后根据每个处理器上的子调度,为这个处理器上相继执行的任务添加有向边使其执行串行化,确保分配到该处理器上的所有任务的执行顺序满足调度的要求;并在该处理器上的结束任务和开始任务间添加具有一个初始符号的有向边,保证在自同步执行时该处理器上的所有任务按照调度给定的顺序循环执行。这种建模方法没有考虑映射到不同处理器上的任务之间的边及这些边上初始符号的冗余性,增加了所获得调度感知 SDFG 的规模,进而增加了吞吐量分析的复杂度。

Damavandpeyma 提出了将 PSOS 直接建模到 SDFG 中的 eSDFG 模型[6],避免了将 SDFG 转化成 IPC 图的步骤,同时减少了模型中的任务数。对一些 SDFG,转换成 IPC 图时可能导致任务数的指数增长,eSDFG 模型则可以避免这种情况的发生。然而,该模型建模过程中极大地增加了边上初始符号的数目,导致采用 max-plus 法进行吞吐量分析时的效率较低。另外,这种建模方法存在建模时间长的缺点,增加了复杂度。

上述吞吐量分析方法的效率依赖于同步数据流图的复杂度,如任务、边或初始符号的数目;另外,将调度建模到同步数据流图所需要的时间也会对总的分析时间产生影响。本文研究了如何高效地将调度建模到同步数据流图中,减少建模时间及所获得模型的复杂度,从而提高吞吐量分析效率。

本文所提建模方法基于开源工具包 SDF3[7]实现。采用了一组随机生成的 SDFG 对所提建模方法进行性能评估,并对所提模型与 eIPC 模型[5]和 eSDFG 模型[6]进行了深入比较。

1　同步数据流图

有向无环图(Directed Acyclic Graph,DAG)是广泛采用的一种应用模型,近来,数据流图如 SDFG[1-2]和场景感知数据流图[8-9]由于其强大的表达能力和可分析性得到大量研究。本文采用 SDFG 来建模流应用,如软件无线电、通信协议、多媒体应用等。SDFG 可以建模流应用的多速率执行特征,同时提供很多可分析特性,如死锁、重复向量和内存需求分析,故其比 DAG 及其他数据流模型更有吸引力。本文所采用 SDFG 模型的定义如下所示。

定义(SDFG)　同步数据流图是一种有向图,由二元组 $G=(V,E)$ 表示,其中 V 为节点的有限集合,每个节点表示一个应用任务,E 表示有向边的有限集合,每条边表示被连接任务间的依赖关系或通信需求。每个节点 $v\in V$ 分配了一个属性 $c(v)$,$c(v)$ 是节点的代价函数,表示执行一次任务所需要的时钟周期。每条边 $e\in E$ 用一个五元组 $(src,p,dst,q,iniTok)$ 表示,其中 src 表示边的源任务,p 表示数据产出速率,dst 表示边的宿任务,q 表示数据消费速率,$iniTok$ 为边上的初始数据数目(数据单位为符号)。对于边 e,采用记号 $src(e)$、$p(e)$ 等表示边上的相应元素。当源任务 $src(e)$ 结束执行时,向边输出 $p(e)$ 个符号的数据;当宿任务 $dst(e)$ 开始执行时,从边上消费 $q(e)$ 个符号的数据。边 e 称为任务 $src(e)$ 的输出边及任务 $dst(e)$ 的输入边。

应用调度是将应用的任务映射到不同处理器、确定映射到同一处理器上的任务的执行顺序以及任务开始/结束时间的过程。如果考虑通信延时/竞争则需要考虑边调度[8-11]。SDFG 的多速率特征使得其任务以不同频率执行并在调度中出现不同次数,SDFG 迭代及重复向量可描述这一特征。

定义(SDFG 迭代)　SDFG 迭代是执行每个任务最小正整数次数同时使各边符号数目恢复初始值的过程。

定义(重复向量)　假定 SDFG 有 n 个任务,各任务从 0 到 $n-1$ 编号,其重复向量 $\boldsymbol{R}$ 是一个长度为 n 的列向量,其中第 k 个元素表示任务 k 在一次迭代中执行的次数。对任务 v,用 $r(v)$ 表示该任务在重复向量中的相应数值。

SDFG 的重复向量可以通过解平衡方程而得[1-2],当平衡方程存在非零解时,重复向量 $\boldsymbol{R}$ 存在,并称 SDFG 是一致的[1-2]。无死锁性是

SDFG 的另一重要特性,这一特性保证 SDFG 可以正常执行。本文只考虑一致并无死锁的 SDFG。

图 1 所示为一个由 5 个任务组成的 SDFG 实例。边末端所标记数字为边的产出速率和消费速率,为简单起见,只有当边速率大于 1 时才在图中显示。边附近的文字为边的标签,标签后面括号中的数字表示边上的初始符号数,如果初始符号数为 0,则予以忽略。图 1 中的 SDFG 的重复向量为 $[2,2,3,1,1]^{\mathrm{T}}$,表示在一次应用迭代中,任务 a_0, a_1, a_2, a_3 和 a_4 分别需要执行 2,2,3,1 和 1 次。

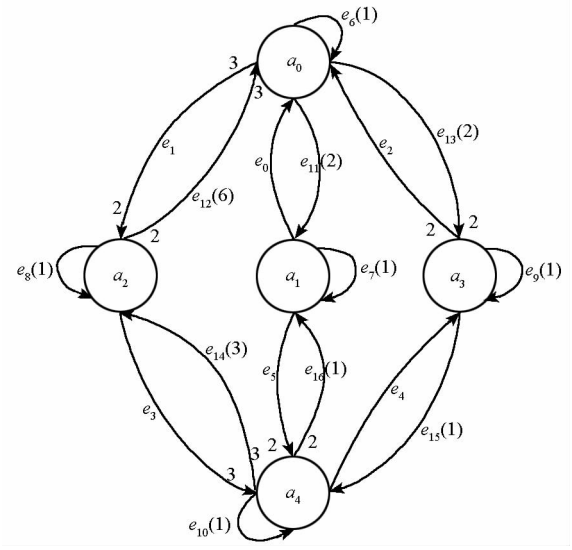

图 1 SDFG 结构示例

Fig. 1 Example of SDFG structure

HSDFG 是 SDFG 的一种特例。与 SDFG 中的多速率不同,HSDFG 中边的速率均为 1,即一个任务开始/结束执行时仅从各输入/输出边消耗/产出一个符号的数据。如果 SDFG 满足一致性条件,则 SDFG 总可以转换成 HSDFG[2]。然而,这一转换可能导致模型规模指数增加[2]。

HSDFG 中的边表明相连任务在执行过程中具有依赖关系。如果边上无初始符号,表明在 HSDFG 的任意一次迭代中,边所连接任务的执行在时间上不能重叠,宿任务只能在源任务执行结束后才能开始执行;如果边上有初始符号,例如对边 $e=(src,1,dst,1,iniTok)$,如果 $iniTok\neq 0$,表明第 k 次迭代中的任务 $dst(e)$ 依赖于第 $k-iniTok$ 次迭代中的任务 $src(e)$,必须在其执行结束后开始执行。

为了描述同一次迭代中任务间的依赖关系,定义 HSDFG 的路径,其定义如下所示。

定义(HSDFG 路径) 对同构同步数据流图 $G=(V,E)$,$L=\{v_0,v_1,\cdots,v_{n-1}\}$,其中 $v_i\in V, i\in\{0,1,\cdots,n-1\}$,如果 $\forall i\in\{0,1,\cdots,n-2\}$,$(v_i,1,v_{i+1},1,0)\in E$,那么 L 是一条从任务 v_0 到 v_{n-1} 的路径。v_0 和 v_{n-1} 分别为路径 L 的源任务和宿任务。

需要注意:HSDFG 路径中相邻任务在 HSDFG 中对应的边没有初始符号,这表明这两个任务在 HSDFG 的执行中需要满足优先约束;路径中的子路径也是路径;对 HSDFG 中的任意两个任务,如果它们之间存在路径,则它们的执行存在依赖关系。

2 周期静态顺序调度

调度包含任务映射、排序与定时,这里只考虑映射与排序,定时可以通过自同步执行来确定。假定任务与处理器之间的映射关系是一个单射,即每个任务只能映射到一个处理器。在系统设计中采用这种映射方式可以避免诸如任务同步的问题[12]。

定义(映射) 给定同步数据流图 $G=(V,E)$ 和多处理器集合 P,V 到 P 的映射可以表示为一个函数 $map:V\rightarrow P$,$map(v)$ 表示任务 $v\in V$ 被分配到这个处理器上。

同一处理器上可能映射有多个任务,且在一次 SDFG 迭代中,任务可能需要执行多次,在调度中也会出现多次,因此,处理器上的任务排序是调度的重要组成部分。

定义(排序) 给定同步数据流图 $G=(V,E)$、多处理器集合 P 和映射函数 $map:V\rightarrow P$,处理器 $p\in P$ 上任务的排序是一个满射 $order_p:V_p\rightarrow\{1,2,\cdots,|V_p|\}$,其中 $V_p=\{v_{k,i}\mid map(v_k)=p, v_k\in V, i\in\{0,1,\cdots,|r(v_k)-1|\}\}$,$v_{k,i}$ 表示任务 v_k 的第 i 个实例。

流应用的一个重要特征是需要对有限或无限长的数据流进行处理,因此应用需要循环执行,循环执行的静态顺序调度称为周期静态顺序调度。

定义(周期静态顺序调度) 给定处理器 p 上的任务排序 $order_p:V_p\rightarrow\{1,2,\cdots,|V_p|\}$,处理器 p 上的周期静态顺序调度是根据排序 $order_p$ 产生的循环调度 $[order_p^{-1}(1), order_p^{-1}(2),\cdots, order_p^{-1}(|V_p|)]$,其中在执行完任务 $order_p^{-1}(|V_p|)$ 后,重新从第一个任务 $order_p^{-1}(1)$ 开始执行。

3 问题阐述

设计嵌入式流应用系统时,吞吐量分析是其中的关键步骤之一,通过分析吞吐量可以确保系统性能满足设计要求。本文研究如何将调度高效

简洁地建模到同步数据流图当中,从而提高吞吐量分析效率,减少系统设计时间。一个有效的调度感知的 SDFG 模型需要满足如下条件:

1)所获得的模型属于同步数据流图,从而可以采用已有分析方法[3-4]对其进行吞吐量分析。

2)分配到不同处理器上的任务在自同步执行时无资源竞争。将调度感知的同步数据流图转换成同构同步数据流图,对其中的任意一对任务 v_a,v_b,如果它们在 SDFG 中无直接或间接依赖关系且被分配到不同处理器,则在转换成的 HSDFG 中,v_a 的任意一个实例与 v_b 的任意一个实例之间没有路径。

3)任务执行顺序满足调度中的排序要求。对任意处理器 p 上的任务排序 $order_p$,对任意一对任务 v_a,v_b,如果在排序中 v_a 的任意一个实例与 v_b 的任意一个实例之间存在依赖关系,则在调度感知的同步数据流图转换成同构同步数据流图后,这两个任务实例间存在路径。

4)对任意处理器 p 上的任务排序 $order_p$,定义源和宿任务分别为第一个开始执行的任务和最后一个执行的任务,在调度感知的同步数据流图的自同步执行中,源任务的执行依赖于上次迭代中的宿任务。

5)保留 SDFG 中的任务依赖关系。SDFG 转换成 HSDFG 后,如果 HSDFG 中的两个任务实例间存在路径,即存在依赖关系,则在调度感知的 SDFG 中这两个任务实例间也存在依赖关系。

4　基于 HSDFG 的调度感知同步数据流模型

本节介绍所提出的调度感知同步数据流模型,该模型满足上节所提出的条件,确保调度所施加的映射和排序约束在自同步执行中得到了满足,因此采用状态空间法[3]和 max-plus 法[4]可以精确分析应用调度到实际平台后的性能。为了避免 eIPC 模型边冗余的问题及 eSDFG 模型初始符号过多的问题,提出了一种基于 HSDFG 的调度感知同步数据流模型。建模的具体步骤如下:

1)构建空的同构同步数据流图 $scheG=(V_s,E_s)$。

2)依次将 SDFG 在一次迭代中出现的所有任务实例加入到 $scheG$ 中:对每个 $v_i \in V$,将 v_i 的 $r(v_i)$ 个实例依次添加到 V_s。为方便起见,将任务 v_i 的第 j 个实例表示为 $v_{i,j}$。

3)根据各处理器上的任务调度,为任务间增加顺序依赖边,确保任务执行按照调度指定顺序进行:对处理器 p 上的调度 $order_p$ 中相继执行的任务实例 $v_{p,i}$,$v_{p,i+1}$,其中 $i \in [0,|V_p|-2]$,添加依赖边$(v_{p,i},1,v_{p,i+1},1,0)$到 E_s。

4)为各处理器 p 上的调度中的首尾任务间增加循环依赖边$(v_{p,|V_p|-1},1,v_{p,0},1,1)$,确保应用执行时分配到同一处理器的任务在相邻迭代中的执行不会产生资源竞争。所添加边的初始符号数为 1,连接调度中最后执行的任务和最开始执行的任务,表示当前迭代必须在前一次迭代完成之后才可以开始。

5)为分配到不同处理器上的任务实例间增加依赖边。对 E 中的任意边 e,如果 e 的源和目的任务不相同且它们被分配到不同处理器,则为这两个任务的实例之间添加依赖边。如果 SDFG 的边上有初始符号,即应用存在迭代间依赖关系,那么在模型中构建新的依赖关系时需要重新计算初始符号数,即 $d=ceil(-i/r(src(e)))$,从而确保迭代间依赖关系在模型中得以保留。具体步骤如图 2 所示。

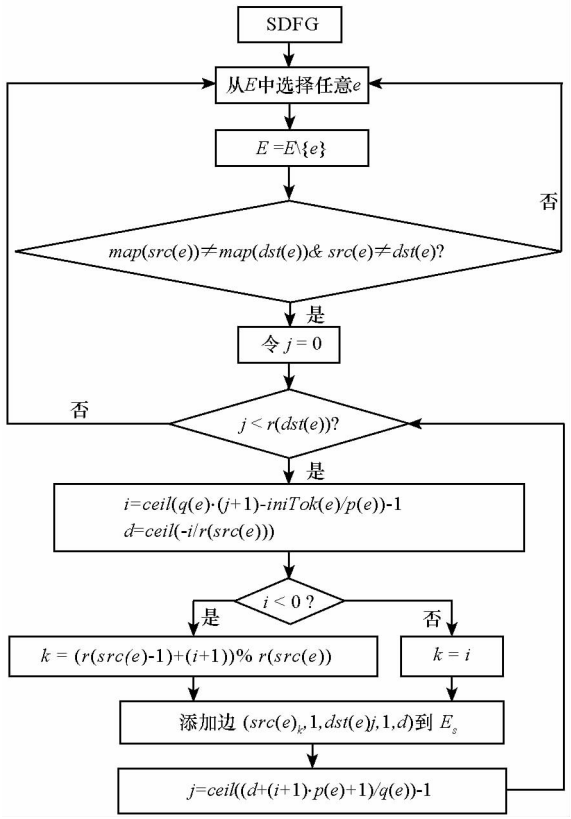

图 2　处理器间依赖边构建流程图

Fig. 2　Workflow for constructing inter-processor dependency edges

5　实验与结果

5.1　测试基准

使用一组随机生成的具有不同大小的 SDFG 进行性能测试。使用开源工具 SDF3[7]来产生随

机 SDFG,其中任务数为 5 到 15。SDFG 重复向量在约束和范数的基础上随机产生。在实验中,重复向量和范数限制为任务数的 5 倍,因此,在一次应用迭代中有 25 到 75 个任务实例。SDFG 属性如入/出度和边产出/消费速率均是在指定平均值、最小/大值和方差的基础上随机产生的。任务入度和出度的平均值和方差均为 2,最小值和最大值分别为 0 和 4。边的产出速率和消费速率的平均值和方差分别为 5 和 7,最小值和最大值分别为 1 和 9。对每个图尺寸,均生成 3 个随机 SDFG。SDFG 参数,包括任务执行时间和边上的符号大小,均随机生成。任务累积执行时间在 400 和 1000 间均匀产生。给定应用图和平台结构,采用文献[13]介绍的映射算法将上述同步数据流图分别映射到有 2,4 和 6 个处理器的多处理器平台并获得有效的任务映射;然后采用最早开始时间算法获得各处理器上的周期静态顺序调度;最后采用不同建模方法将获得的调度建模到同步数据流图当中,以此来分析不同建模方法的性能。所有实验在 Intel i5 处理器(3.0 GHz、4 GB RAM)、64 位 Windows 7 环境下进行。

5.2 度量参数

为了评估所提建模方法的性能,采用如下一些度量参数对其进行评估:

1)模型中初始符号数。该度量参数是影响基于 max-plus 代数的吞吐量分析方法的效率的关键因素,通过比较不同模型的初始符号数,可以评估不同建模方法的优劣。

2)模型规模。模型规模包括模型中的任务数和边数,这个参数是影响基于状态空间搜索的吞吐量分析方法的效率的关键因素,因为该方法的效率依赖于状态大小,而状态大小由任务和边的数目决定。通过比较不同建模方法的这个参数,可以评估不同建模方法的优劣。

3)建模与吞吐量分析时间。建模时间与吞吐量分析时间是比上述参数更为直观的评价方法,可以更全面地表征不同建模方法的性能。

5.3 实验结果与分析

图 3 和图 4 采用随机生成的 SDFG 对三种调度感知同步数据流模型的规模、构建时间和吞量分析时间进行了比较。“eIPC”“eSDFG”和“eHSDFG”分别表示文献[5]、文献[6]和本文所介绍的建模方法。

图 3 对采用不同建模方法所获得的调度感知同步数据流模型的规模(包括任务数、边数)

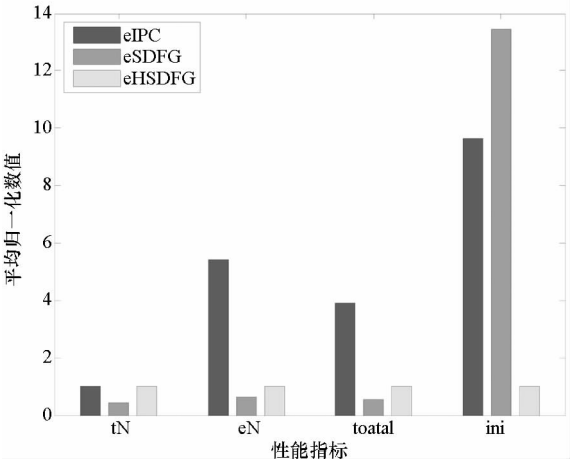

图 3 不同调度感知同步数据流模型归一化大小

Fig. 3 Normalized sizes of different schedule-aware SDFGs

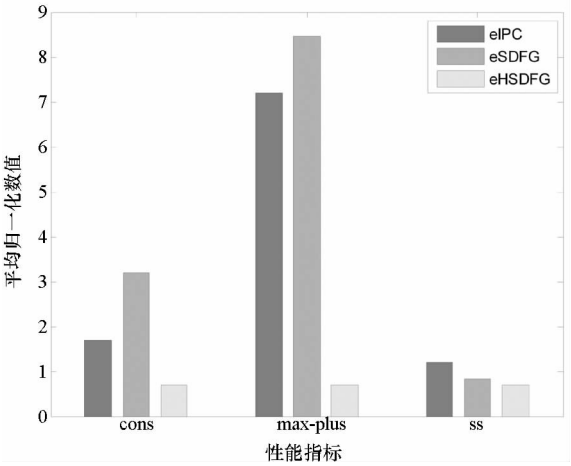

图 4 不同调度感知同步数据流模型归一化构建时间与吞吐量分析时间

Fig. 4 Normalized construction time and throughput analysis time of different schedule-aware SDFGs

和初始符号数进行了比较。图中“tN”“eN”“total”和“ini”分别表示模型中的任务数、边数、任务和边的数目总和、初始符号数。图中所有数据采用 eHSDFG 模型对应数值进行归一化。从图中数据可以看出,eIPC 和 eHSDFG 这两种建模方法所获得的模型有相同的任务数目,因此平均归一化任务数为 1。然而,eIPC 模型有更多的边,平均达到 eHSDFG 模型的 5.434 倍。另外,eIPC 模型中初始符号也更多,平均为 eHSDFG 模型的 9.628 倍。与 eIPC 模型不同,eSDFG 模型中任务与边更少些,平均为 eHSDFG 模型的 0.426 倍和 0.623 倍;然而,eSDFG 模型有更多的初始符号,与 eHSDFG 模型相比,初始符号数目的平均比达到 13.455。

图 4 对采用不同建模方法构建调度感知同步数据流模型及采用 max-plus 法和状态空间法进

行吞吐量分析所需时间进行了比较。“cons”“max-plus”和“ss”分别表示模型构建时间、采用 max-plus 法的吞吐量分析时间与构建时间之和及采用状态空间法的吞吐量分析时间与构建时间之和。图中所有数据采用 eHSDFG 模型对应数值的一半进行归一化后取对数。从图中数据可以看出,eIPC 和 eSDFG 这两种模型的构建时间与分析时间均显著高于 eHSDFG 模型。具体而言,eIPC 模型的平均构建时间是 eHSDFG 模型的 2.736 倍,max-plus 分析时间和状态空间分析时间分别是 675.666 和 1.662 倍,这是由于 eIPC 模型的规模和初始符号数都明显高于 eHSDFG 模型。而 eSDFG 模型的平均构建时间是 eHSDFG 模型的 12.265 倍,max-plus 分析时间是 eHSDFG 模型的 2372 倍;虽然 eSDFG 模型的任务数要少于 eHSDFG 模型,但其状态空间分析时间达到 eHSDFG 模型的 1.154 倍,因为 eSDFG 模型的平均构建时间要远高于 eHSDFG 模型。

综上所述,采用本文所提出的 eHSDFG 模型建模 PSOS 可以有效提高吞吐量分析效率。采用基于状态空间和基于 max-plus 代数的吞吐量分析方法分别可以减少 15.4% 和 66.2% 的吞吐量分析时间。

6　结论

设计嵌入式流应用系统时,满足系统吞吐量指标是系统设计的重要要求,因此系统吞吐量分析是系统设计过程中不可或缺的评估和验证手段。本文对如何将周期静态顺序调度高效简洁地建模到同步数据流图中进行研究,以提高应用调度的吞吐量分析效率。所提出的调度感知同步数据流模型避免了 eIPC 模型所存在的边与初始符号冗余的问题及 eSDFG 模型所存在的初始符号冗余及模型构建时间长的缺点,可以有效提高系统吞吐量分析效率。

参考文献(References)

[1] Lee E A, Messerschmitt D G. Static scheduling of synchronous data flow programs for digital signal processing[J]. IEEE Transactions on Computers, 1987, 36(1): 24 – 35.

[2] Sriram S, Bhattacharyya S S. Embedded multiprocessors: scheduling and synchronization[M]. New York, NY, USA: Marcel Dekker Inc, 2012.

[3] Ghamarian A H, Geilen M, Stuijk S, et al. Throughput analysis of synchronous dataflow graphs[C]//Proceedings of 6th International Conference on Application of Concurrency to System Design, 2006: 25 – 36.

[4] Geilen M. Synchronous dataflow scenarios[J]. ACM Transactions on Embedded Computing Systems, 2010, 10(2): 16.

[5] Bambha N, Kianzad V, Khandelia M, et al. Intermediate representations for design automation of multiprocessor DSP systems[J]. Design Automation for Embedded Systems, 2002, 7(4): 307 – 323.

[6] Damavandpeyma M, Stuijk S, Basten T, et al. Schedule-extended synchronous dataflow graphs[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2013, 32(10): 1495 – 1508.

[7] Stuijk S, Geilen M, Basten T. SDF^3: SDF for free[C]// Proceedings of 6th International Conference on Application of Concurrency to System Design, 2006: 276 – 278.

[8] Damavandpeyma M, Stuijk S, Geilen M, et al. Parametric throughput analysis of scenario-aware dataflow graphs[C]// Proceedings of IEEE 30th International Conference on Computer Design, 2012: 219 – 226.

[9] Damavandpeyma M, Stuijk S, Basten T, et al. Throughput-constrained DVFS for scenario-aware dataflow graphs[C]// Proceedings of IEEE 19th Real-Time and Embedded Technology and Applications Symposium, 2013: 175 – 184.

[10] Sinnen O. Task scheduling for parallel systems[M]. USA: John Wiley & Sons, 2007.

[11] Sinnen O, Sousa L. List scheduling: extension for contention awareness and evaluation of node priorities for heterogeneous cluster architectures[J]. Parallel Computing, 2004, 30(1): 81 – 101.

[12] Bilsen G, Engels M, Lauwereins R, et al. Static scheduling of multi-rate and cyclo-static DSP-applications[C]// Proceedings of VLSI Signal Processing, 1994: 137 – 146.

[13] Tang Q, Basten T, Geilen M, et al. Mapping of synchronous dataflow graphs on MPSoCs based on parallelism enhancement[J]. Submitted to Journal of Parallel and Distributed Computing, 2017, 101: 79 – 91.