ISCAS-SKLCS-14-14 **August, 2014**

中国科学院软件研究所
计算机科学国家重点实验室
技术报告

基于异构多核平台的同步数据流图帕累托优化与调度

by

顾玉磊，朱雪阳，晏荣杰，张广泉

State key Laboratory of Computer Science
Institute of Software
Chinese Academy of Sciences
Beijing 100190. China

基于异构多核平台的同步数据流图帕累托优化与调度*

顾玉磊[1,2]，朱雪阳[2+]，晏荣杰[2]，张广泉[1]

[1](苏州大学 计算机科学与技术学院,苏州 215006)

[2](中国科学院 软件研究所 计算机科学国家重点实验室,北京 100190)

Pareto Optimization and Scheduling of Synchronous Dataflow Graphs on a Heterogeneous Multicore Platform

GU Yu-Lei[1,2], ZHU Xue-Yang[2+], YAN Rongjie[2], ZHANG Guangquan[1]

[1](School of Computer Science and Technology, Soochow University, Suzhou 215006, China)

[2](State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

+ Corresponding author: Phn: +86-10-62661655, E-mail: zxy@ios.ac.cn, http://www.ios.ac.cn/~zxy

Abstract: Synchronous Dataflow Graphs (SDFGs) are widely used to model streaming applications such as multimedia and digital signal processing applications. Streaming applications are usually required to reach a high throughput to guarantee a smoothly running. The use of heterogeneous multicore processors to improve the throughput of streaming applications has become a feasible solution. However, a higher throughput is usually achieved with the increase of energy consumption. In this paper, we present a method to explore the Pareto space of energy consumption and throughput and to find the schedule of each Pareto point. A system model we consider includes an SDFG description for the application and a heterogeneous multicore platform. The system model is transformed to a network of timed automata (NTA) and the optimization goals are formalized as temporal logic formulae. The NTA and formulae are then used as the input of the real time model checking tool UPPAAL. We obtain the Pareto points and corresponding schedules by the trace provided by UPPAAL. Our method is exact, which is benefited from the exhaustive search nature of model checking technique. The method can be used to help designers to understand the quantitative relationship between energy consumption and throughput in the design period, and therefore shorten development cycles and reduce costs of system developments.

Key words: synchronous dataflow graphs; multicore; Pareto optimization; scheduling; model checking

摘 要: 同步数据流图被广泛用于为流应用程序建模,如多媒体和数字信号处理程序等.流应用程序需达到一定吞吐量才能流畅运行,利用异构多核处理器来进一步提高流应用程序的吞吐量已经成为当今嵌入式系统的发展趋势,但是高吞吐量往往伴随着能耗的增加.本文针对基于异构多核平台的同步数据流图系统模型,给出求解所有能耗和吞吐量的帕累托优化点及其相应静态调度的方法.首先将系统模型转换为时间自动机网络,并将分析目标转换为时序逻辑公式;再使用实时模型检测工具 UPPAAL 寻找解决方案;最后对 UPPAAL 返回的结果进行分析,找出满足要求的调度.由于模型检测方法可对问题空间进行穷尽搜索,本文方法得到的是精确结果.本文方法可帮助设计者在系统开发早期了解系统能耗和吞吐量的量化关系,有利于缩短系统的开发周期、降低开发成本.

关键词: 同步数据流图;异构多核平台;帕累托优化;调度;模型检测

*

1 简介

当今电子设备中的一类重要程序是流应用程序[19],如音、视频处理程序.这类应用程序一般要达到较高的吞吐量才能流畅运行,越来越多的设备采用异构多核处理器来提高程序性能.使用这类应用的用户不但关心设备的执行性能,而且对续航时间也有较高的要求.续航时间通常与硬件设备执行应用程序产生的能耗相关.电子系统设计者需为设备设计出能耗较低、吞吐量较高的调度,以满足用户需要,但日益复杂的系统使工程师面临着巨大的挑战.

基于模型的性能分析优化方法能够帮助工程师在设计阶段发现并解决问题,从而有效缩短电子系统的开发周期,降低开发风险和成本.同步数据流图(Synchronous Dataflow Graphs, SDFGs)[4]被广泛用于为多媒体和数字信号处理等流应用程序建模,本文将针对基于异构多核平台的同步数据流图进行能耗和吞吐量的帕累托(Pareto)优化与调度.

Karp 利用最大关键环(Max Cycle Mean, MCM)[11]、Groote 等人利用极大加代数[12]方法求解吞吐量,但都需将同步数据流图转化为同构同步数据流图[15](Homogeneous SDFGs),其规模可能是原同步数据流图的指数倍[16].Stuijk 等人计算吞吐量和缓存的帕累托优化点[6],但基于无限同构处理器的假设,当考虑有限处理器时,此问题是 NP 完全问题,直接基于同步数据流图的优化与调度难度较大.

时间自动机[1]是描述并发实时系统的形式化模型,因为其成熟的建模与验证技术,在工业界成果丰硕[7].实时模型检测工具 UPPAAL[2]将时间自动机扩展为时间自动机网络,可解决复杂实时系统的优化与调度问题,并已集成有效的状态空间约减技术,执行效率较高.但对电子系统设计者而言,由于缺乏形式化方法的背景知识,直接利用时间自动机工具进行问题求解仍比较困难.本文通过模型转换,使电子系统设计者只需关心执行平台和同步数据流图的建模和优化结果而不需要关心优化过程.

通过时间自动机和模型检测方法,Fakih 等人计算异构多核系统中总线的最大延迟[10].Madsen 等人分析任务图在异构多核平台执行时的缓存占用和能耗[8].Ahmad 等人求解处理器个数和同步数据流图吞吐量的帕累托优化问题[9].而本文关心的是能耗和吞吐量的帕累托优化与调度问题.

本文采用基于模型的设计分析方法,用基于异构多核执行平台和同步数据流图的系统模型对流应用程序在异构多核处理器上的执行问题建模,然后把系统模型转换为时间自动机网络,将分析目标形式化为时序逻辑公式,利用 UPPAAL 进行模型检测,获得满足性质约束的仿真路径,由路径得到相应的静态调度、能耗和吞吐量,最终迭代求解处理器总能耗和流应用程序吞吐量的帕累托优化与调度问题.异构多核平台和同步数据流图的建模、本文的模型转换以及帕累托优化与调度算法均集成于工具 iDFOS[20]中,用户可直接在此工具中对同步数据流图于多核平台的执行问题进行分析和优化.

本文结构如下:第 2 章介绍基本概念并定义系统模型;第 3 章介绍从系统模型到时间自动机网络的转换方法;第 4 章介绍调度的获得,能耗和吞吐量的计算,以及求解帕累托优化点的算法;第 5 章用实验评估本文方法的可用性;最后一章总结全文.

2 基本概念与系统模型

2.1 执行平台

定义 1. 执行平台由三元组 $Arch(PT,\Pi,\Psi)$ 定义. PT 表示处理核类型集合, $\Pi: PT \to \mathbb{N}^+ \times \mathbb{N}^+$ 为每类处理核 $pt \in PT$ 指定了二元组 $(idlePC, usingPC)$, $\Psi: PT \to \mathbb{N}$ 指定每类处理核 $pt \in PT$ 的个数.其中 $idlePC$ 表示处理核处于空闲(idle)状态的功耗, $usingPC$ 表示处理核处于运行状态的功耗, $\mathbb{N}$ 表示非负整数集合, $\mathbb{N}^+$ 表示正整数集合.

图 2. 1 所示的执行平台 *Arch*1 共有 *ptA* 和 *ptB* 类型的处理核各一个, *ptA* 类型的处理核处于空闲和运行状态的功耗分别为 10 和 90 瓦,而 *ptB* 为 20 和 30 瓦.

处理核类型	功耗(瓦)	
	idlePC	usingPC
ptA	10	90
ptB	20	30

(a)处理核类型及功耗

执行平台名	核个数	
	ptA	ptB
Arch1	1	1

(b)执行平台

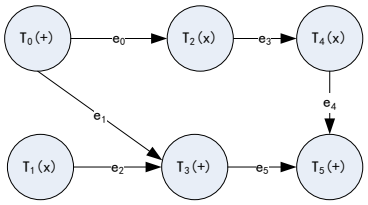
图 2. 1　执行平台 *Arch*1

2.2 同步数据流图

定义 2. 同步数据流图由三元组 $SDFG(A,C,\Theta)$ 定义. A 是节点(actor)集合,每个节点 $a \in A$ 表示一个进程或处理模块,节点执行开始时会消耗一定数量的数据(token),执行结束时会生成一定数量的数据. C 是边(channel)集合,用于存储节点生成的数据,每条边 $c \in C$ 包含 5 个属性: $src(c)$ 和 $snk(c)$ 分别表示其源节点和目标节点; $del(c)$ 表示边 c 的初始数据个数; $prd(c)$ 表示节点 $src(c)$ 一次执行生成的数据个数,称为**输出数率**; $cns(c)$ 表示节点 $snk(c)$ 一次执行消耗的数据个数,称为**输入数率**. $\Theta: A \times PT \to \mathbb{N}^+$ 指定节点 $a \in A$ 在处理核 $pt \in PT$ 上的执行时间.

当节点所有输入边上的数据个数均大于等于输入数率时,节点允许执行.节点执行开始时,需在所有输入边上消耗与输入数率相等个数的数据.节点执行结束时,在所有输出边上生成与输出数率相等个数的数据.当节点没有输入边时,表示节点随时可以执行.

以文献[3]中的加乘运算为例,其同步数据流图 G_{AM} 如图 2. 2 所示. G_{AM} 共有 6 个节点和 6 条边.每条边的输入和输出数率均为 1,在图中省略.所有边的初始数据个数均为 0.节点 T_0 和 T_1 随时可以执行.当边 e_0 至少有 1 个数据时, T_2 才可以执行, T_2 执行开始时会消耗 e_0 上的一个数据,执行结束时在边 e_3 上生成一个数据.各节点在不同类型处理核上的执行时间 Θ 如图 2. 2(b)所示,例如 T_0 在 *ptA* 和 *ptB* 类型处理核上的执行时间分别为 2 和 5(ps).



(a)加乘运算同步数据流图

A(ps) / PT	T_0	T_1	T_2	T_3	T_4	T_5
ptA	2	3	3	2	3	2
ptB	5	7	7	5	7	5

(b)节点在不同类型处理核上的执行时间

图 2. 2　加乘运算同步数据流图 G_{AM}

对同步数据流图 $SDFG(A,C,\Theta)$ 的每条边 c ,存在一个平衡等式 $prd(c)\times q(src(c))=cns(c)\times q(snk(c))$.如果 $SDFG(A,C,\Theta)$ 的平衡等式组存在正整数解,则称 $SDFG(A,C,\Theta)$ 是一致的.不一致的同步数据流图可能导致缓存溢出或死锁[4],因此本文仅考虑一致的同步数据流图. $SDFG(A,C,\Theta)$ 的平衡等式组的最小正整数解称为迭代向量(Repetition Vector),记为 $q(A)$.

同步数据流图的每个节点 $a\in A$ 执行 $q(a)$ 次称为一次迭代.同步数据流图经过一次迭代后,所有边的数据个数与迭代前相同,不会影响下一次迭代的执行,因此同步数据流图经过多次迭代后仍会回到初始状态.

2.3 系统模型与调度

定义 3. 系统模型由二元组 $M(Arch,SDFG)$ 定义,其中 *Arch* 表示执行平台, *SDFG* 表示同步数据流图.

如加乘运算同步数据流图 G_{AM} 在异构双核处理器 *Arch*1 上执行的系统模型可用 $M_{AM}=M(Arch1,G_{AM})$ 表示.

定义 4. 系统模型 $M(Arch,SDFG)$ 的**调度** *Sch* 是指同步数据流图 *SDFG* 在执行平台 *Arch* 上的多次迭代的时间安排和处理核映射,每个节点 $a\in A$ 的每次执行可由**调度记录** $SchR(a,pro,st,et)$ 表示,即将节点 a 分配到处理核 $pro\in Arch$ 上执行,执行开始时间为 st ,执行结束时间为 et .

因此流应用程序在异构多核平台的帕累托优化与调度问题可以视为系统模型 $M(Arch,SDFG)$ 的调度优化.本文的目的是优化系统模型的调度,使得在该调度下的能耗和吞吐量达到帕累托最优.获得所有帕累托优化点及相应调度的算法将在 4.3 节详细介绍.

3 模型转换

3.1 UPPAAL时间自动机网络

定义 5. 时间自动机[1](Timed Automata, TA)由六元组 $\langle L,X,V,E,Inv,l_0\rangle$ 定义. L 为有穷节点集合, X 为时钟集合, V 为整型变量集合, $E\subseteq L\times\Omega(X,V)\times U(X,V)\times L$ 为边集合, $Inv:L\to\Omega(X,V)$ 为每个节点指派约束作为不变式, $l_0\in L$ 表示初始节点.记 $k\in\mathbb{N}$, $x\in X$, $\sim\{<,\le,=,>,\ge\}$, ε 为常数或变量表达式. $\Omega(X,V)$ 表示 X 和 V 上的约束,可由 $g::=true\mid x\sim k\mid\varepsilon_1\sim\varepsilon_2\mid g_1\wedge g_2$ 描述. $U(V,X)$ 用于更新时钟和变量值.

$TA=\langle L,X,V,E,Inv,l_0\rangle$ 对应一个执行系统 $\nabla(S,s_0,\to)$, S 为状态集合, $s_0\in S$ 为初始状态.状态由三元组 (l,x,v) 表示: l 是时间自动机的一个节点; $x:X\to R_0^+$ 记录每个时钟的一个非负实数值; $v:V\to\mathbb{Z}$ 记录每个变量的一个整数值.用 $x+\delta$ 表示将 x 中所有时钟值加上 δ , $\alpha\models\beta$ 表示 α 中的时钟和变量满足约束 β , $g\subseteq\Omega(X,V)$, $u(x,v)\subseteq U(X,V)$,变迁关系 $\to$ 包含的两种变迁[18]如下:

- 延时变迁(delay transition): $(l,x,v)\longrightarrow(l,x+\delta,v)$,当且仅当 $\forall\delta':0\le\delta'\le\delta\Rightarrow(x+\delta',v)\models Inv(l)$,其中 $\delta\in\mathbb{N}^+$
- 离散变迁(discrete transition): $(l,x,v)\longrightarrow(l',x',v')$,当且仅当 $\exists e=(l,g,u,l')\in E$,并且 $(x,v)\models g$, $(x',v')=u(x,v)$, $(x',v')\models Inv(l')$

一条**路径**(trace)是状态和变迁的交替序列 $s_0\to s_1\to\ldots$,路径长度可以有限或者无限, $\to$ 可以是延时或离散变迁.

时间自动机网络[18](Network of Timed Automata, NTA)由多个时间自动机组成,可由 $nta_M(TA_0\|TA_1\|...\|TA_{n-1},\mathrm{K})$ 表示,其中 $\|$ 表示并发, K 为共享时钟和变量.时间自动机网络在 UPPAAL 中由声明(Declarations)、模板(Template)和系统声明(System Declarations)组成:声明部分声明系统中的共享时钟和变量,被所有时间自动机共享;模板部分定义一组时间自动机模板;系统声明部分声明时间自动机网络的具体组成,即由每个模板实例化出的多个时间自动机共同组成描述系统的时间自动机网络.

UPPAAL 时间自动机模板中的条件表达式(guard)用于判断时钟和变量是否满足约束;更新表达式(update)用于更新时钟和变量;同步表达式(synchronization)形如 *sig*! 和 *sig*？,其中 *sig*! 表示发送同步信号, *sig*？表示接收同步信号.下文中用 (g,u,syn) 表示时间自动机边上的表达式标签,其中 g 表示条件表达式, u 表示更新表达式, *syn* 表示同步表达式.

3.2 系统模型转换到时间自动机网络

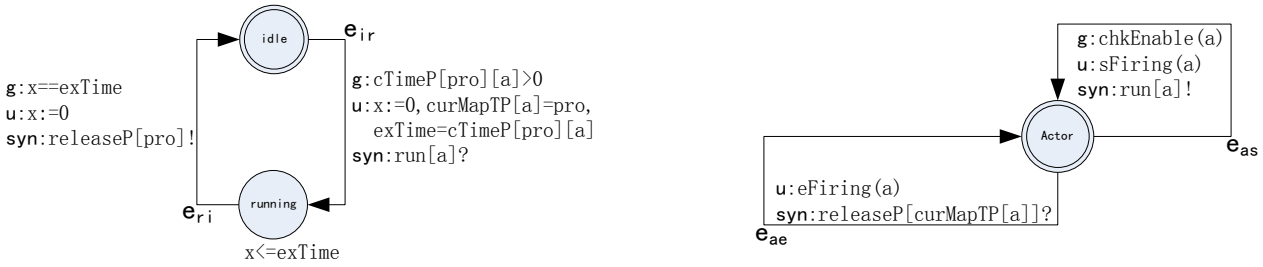
本文将系统模型转换成时间自动机网络来解决系统模型的调度优化问题.在转换过程中,将系统模型转换成若干个处理核时间自动机和若干个节点时间自动机,处理核时间自动机用于描述处理核行为,而节点时间自动机用于描述同步数据流图节点的行为.

3.2.1 执行平台转换

执行平台可转换成若干个处理核时间自动机,其个数与执行平台的处理核个数相等.处理核可处于空闲或运行状态.当同步数据流图节点分配到处理核执行时,处理核由空闲状态转换到运行状态.处理核运行相应时间,即节点在该处理核执行结束后,处理核由运行状态转换到空闲状态.

每个处理核时间自动机可由 $TA_p=(L_p,X_p,V_p,E_p,Inv_p,l_p^0)$ 表示,如图 3. 1(a)所示. $L_p=\{idle,running\}$; $l_p^0=idle$; $X_p=\{x\}$; E_p 含边 e_{ir}, e_{ri}; $Inv_p(running):x\le exTime$, $Inv_p(idle):true$.其中:

$cTimeP[pro][a]$ 表示节点 a 在处理核 pro 上执行所需的时间,由同步数据流图 $SDFG(A,C,\Theta)$ 中的 Θ 得到,当其大于 0 时,表示 a 可以在处理核 pro 上运行; $x:=0$ 表示将时钟变量 x 置为 0; $curMapTP[a]$ 用于存储 a 当前分配执行的处理核; $run[a]?$ 表示接收执行节点 a 的同步信号; $releaseP[pro]!$ 表示发送释放处理核 pro 的同步信号.



(a)处理核时间自动机 TA_p　　(b)节点时间自动机 TA_a

图 3. 1　时间自动机模板

3.2.2 同步数据流图转换

同步数据流图可转换成若干个节点时间自动机,其个数与同步数据流图的节点个数相等.用户可给定调度的迭代次数 $iterN$,使得所有节点 $a\in A$ 的执行次数等于 $q(a)\times iterN$.为了便于描述,将节点 a 的已执行次数记为 $sfTimes(a)$;节点 a 的输入和输出边集合分别记为 $inE(a)$ 和 $outE(a)$;将边 c 现有数据个数记为 $tkn(c)$;用 $run[a]!$ 表示发送执行节点 a 的同步信号;用 $releaseP[curMapTP[a]]?$ 表示接收释放处理核的同步信号,用于释放节点 a 分配到的处理核.

每个节点时间自动机可由 $TA_a=(L_a,X_a,V_a,E_a,Inv_a,l_a^0)$ 表示,如图 3. 1(b)所示. $L_a=l_a^0=\{Actor\}$; X_a 为 $\varnothing$; E_a 含边 e_{as} 和 e_{ae},分别描述节点执行开始和结束时的行为; $Inv_a(Actor):true$.其中:

- $chkEnable(a)$:用于检查节点 a 是否允许执行.形式化表示为: $\forall c\in inE(a),(tkn(c)\ge cns(c))\wedge(sfTimes(a)<q(a)\times iterN)$
- $sFiring(a)$:用于描述节点 a 执行开始时的行为.形式化表示为: $\forall c\in inE(a),tkn(c)=tkn(c)-cns(c)$,并且 $sfTimes(a)++$
- $eFiring(a)$:用于描述节点 a 执行结束时的行为.形式化表示为: $\forall c\in outE(a),tkn(c)=tkn(c)+prd(c)$

3.2.3 系统模型转换

节点 $a\in A$ 允许执行后,还需有空闲的处理核可分配才可以执行.当 TA_a 检查并发现节点允许执行后, TA_a 会通过 $run[a]!$ 发送同步信号,如果有处于空闲($idle$)状态的处理核时间自动机 TA_p 通过 $run[a]?$ 接收同步信号后,触发 TA_p 的边 e_{ir} 和 TA_a 的边 e_{as}, TA_a 消耗输入边数据, TA_p 更新时钟和变量后转换到运行($running$)状态.当 TA_p 处于运行状态的时间等于节点在该处理核上所需的执行时间时,即节点执行结束后, TA_p 通过 $releaseP[pro]!$ 发送同步信号, TA_a 通过 $releaseP[pro]?$ 接收释放处理核的同步信号,触发 TA_p 的边 e_{ri} 和 TA_a 的边 e_{ae}, TA_a 在输出边生成数据,节点执行结束, TA_p 转换到空闲状态.

对一个由含 m 个处理核的执行平台和含 n 个节点的同步数据流图组成的系统模型 $M(Arch,SDFG)$,其等价的 UPPAAL 时间自动机网络可由 $nta_M(TA_p^0\|TA_p^1\|...\|TA_p^{m-1}\|TA_a^0\|TA_a^1\|...\|TA_a^{n-1},K)$ 表示.其中处理核和节点时间自动机模板如图 3. 1 所示; K 表示所有共享时钟和变量,如全局时钟 $glbClk$ 和变量 $curMapTP[a]$ 等,均在 UPPAAL 的声明中定义.由于语义的一致性,系统模型的帕累托优化与调度问题可通过与其等价的时间自动机网络来解决.

以系统模型 $M_{AM}=M(Arch1,G_{AM})$ 为例,其对应的时间自动机网络为 $nta_{AM}=nta_M(TA_p^0\|TA_p^1\|TA_a^0\|TA_a^1\|...\|TA_a^5,K)$.

4 能耗和吞吐量的帕累托优化与调度

工具 UPPAAL[17]可验证系统模型 $M(Arch,SDFG)$ 对应的时间自动机网络 nta_M 是否满足性质约束,当 nta_M 满足性质约束时,UPPAAL 会返回一条满足性质约束的可行路径,即状态和变迁的交替序列,记为 σ.通过路径 σ 可得到系统模型的一个满足约束的调度,由调度可进一步计算该调度的吞吐量及所需能耗.通过在性质中加入能耗约束条件并进行最快路径(Fastest trace)搜索,对系统模型调度进行优化,最终可迭代求解出所有能耗和吞吐量的帕累托优化点及相应的可行静态调度.

4.1 调度获取

工具 UPPAAL 中的性质可由计算树逻辑(Computation Tree Logic, CTL)表达式 $EF\varphi$ 描述,其表示至少存在一条路径,在它的某个状态下条件 φ 为真.将时间自动机网络 nta_M 满足性质 $EF\varphi$ 记为 $nta_M \models EF\varphi$.由于 nta_M 中节点时间自动机受迭代次数的限制,最终所有时间自动机将进入死锁状态,该性质可用 $EF\ \ deadLock$ 描述.因此用 UPPAAL 检测 $nta_M \models EF\ \ deadLock$ 时,会返回一条可行路径,在该路径中每个节点的执行次数等于 $q(a)\times iterN$,即同步数据流图进行了 $iterN$ 次迭代,也就是说,该路径隐含了系统模型的一种调度.

算法 1 通过 UPPAAL 检测性质约束后返回的路径 σ ,可得到其对应的系统模型调度 sch_σ .在 σ 中的离散变迁集合中,将由处理核自动机边 e_{ir} 触发的变迁集合记为 Tir_σ .则对某一变迁 $t \in Tir_\sigma$,由边 e_{ir} 上的同步信号 $run[a]$? 可得到执行节点,记为 $t.a$;由边 e_{ir} 所在的处理核自动机可得到节点执行时分配的处理核,记为 $t.pro$;将变迁 t 的源状态记为 $t.os$,由该状态的全局时钟,即 $(t.os).glbClk$ 可得到节点执行开始时间;由于节点在该处理核的执行时间为 $cTimeP[t.a][t.pro]$,执行结束时间可由节点执行开始时间加上节点所需执行时间得到.

算法 1 getSch(σ)

输入: UPPAAL 返回的路径 σ

输出: σ 对应的系统模型调度

```
1:  Sch sch_σ        //系统模型调度
2:  SchR schR_s      //一条调度记录
3:  for all t ∈ Tir_σ do   //遍历 σ 中由 e_ir 触发的变迁
4:      originalS = t.os   //变迁的源状态
5:      schR_s.a = t.a
6:      schR_s.pro = t.pro
7:      schR_s.st = originalS.glbClk
8:      schR_s.et = schR_s.st + cTimeP[t.a][t.pro]
9:      sch_σ .add( schR_s )
10: end for
11: return sch_σ
```

算法 2.1 getIterT (sch_σ)

输入: 路径 σ 对应的调度 sch_σ

输出: 调度 sch_σ 执行总耗时

```
schR_l = sch_σ .getLastSchR()
iterT = schR_l.et
return iterT
```

算法 2.2 getThr(sch_σ)

输入: 同算法 2.1

输出: 调度 sch_σ 下的吞吐量

```
return iterN /getIterT( sch_σ )
```

算法 2.3 getEC (sch_σ)

输入: 同算法 2.1

输出: 调度 sch_σ 下的总能耗值

```
1:  TEc = 0
2:  int[] occT    //每个处理核处于运行状态的总时间
3:  iterT = getIterT( sch_σ )
4:  for all schR_i ∈ sch_σ do
5:  occT[schR_i.pro] += schR_i.et - schR_i.st
6:  end for
7:  for p ∈ Pro do
8:  TEc += occT[p] * usingPC(p) +( iterT - occT[p] )* idlePC(p)
9:  end for
10: return TEc
```

4.2 能耗和吞吐量计算

算法 2.1 可计算调度 sch_σ 下的执行总耗时, $sch_\sigma.getLastSchR()$ 用于得到调度 sch_σ 的最后一条调度记录, sch_σ 所需的执行总耗时可由最后一条调度记录中节点的执行结束时间得到.

定义 6. 吞吐量指单位时间内同步数据流图迭代的次数,可由 $iterN$ / $iterT$ 计算得到,其中 $iterT$ 表示同步数据流图 $iterN$ 次迭代执行的总耗时.

调度 sch_σ 下的吞吐量可由算法 2.2 可得到,其中迭代执行的总耗时由算法 2.1 得到, $iterN$ 在 UPPAAL 的声明中定义.

通过计算 $\sum_{p\in Pro}[occT(p)\times usingPC(p)+(iterT-occT(p))\times idlePC(p)]$,算法 2.3 可得到调度 sch_σ 下的总能耗值.其中 $occT(p)$ 表示处理核 p 处于运行状态的总时间, $idlePC(p)$ 和 $usingPC(p)$ 分别表示处理核 p 处于空闲和运行状态的功耗,迭代执行总耗时 $iterT$ 由算法 2.1 得到, Pro 表示所有处理核,所有变量均在 UPPAAL 的声明中定义.

对与系统模型 M_{AM} 等价的时间自动机网络 nta_{AM} ,由 UPPAAL 返回的路径和算法 1 可获得如图 4.1(b)所示的调度,其对应的迭代执行总耗时、能耗和吞吐量如图 4.1(a)所示.

4.3 帕累托优化

利用 UPPAAL 的最快路径搜索来检测 $nta_M \models EF\ \ deadLock$,可优化系统模型调度,使得该调度下的同步数据流图的吞吐量最高.为了得到给定能耗约束 ecC 下吞吐量最高的调度,需在性质中加入能耗约束条件,可表示为 $EF\ \ deadLock \wedge con(ecC)$.同步数据流图迭代结束后,执行平台产生的总能耗应小于等于 ecC ,即 $\sum_{p\in Pro}[occT(p)\times usingPC(p)+(iterT-occT(p))\times idlePC(p)]\le ecC$,由于迭代结束后的执行总耗时 $iterT$ 可由全局时钟 $glbClk$ 得到,因此整理后的能耗约束条件 $con(ecC)$ 如式(1)所示.

$$con(ecC) \equiv_{def} glbClk \le \frac{ecC-\sum_{p\in Pro}[occT(p)\times(usingPC(p)-idlePC(p))]}{\sum_{p\in Pro} idlePC(p)} \tag{1}$$

定义 7. **记录点**由 $P(ec,thr,sch)$ 定义,其中 ec 表示能耗, thr 表示吞吐量, sch 表示该能耗和吞吐量下的一种可行静态调度.当同一组能耗和吞吐量值可对应多种调度时,本文只选取一种调度作为同一个记录点.当某能耗和吞吐量约束下无对应调度时,不作为记录点.

定义 8. 记录点是**优化点** $OP(ec,thr,sch)$,当且仅当对任意记录点 (ec',thr',sch') ,如果 $ec'=ec$,则 $thr'\le thr$,即表示所有记录点中同一能耗下吞吐量最大的记录点.

利用 UPPAAL 的最快路径搜索来检测 $nta \models EF\ \ deadLock \wedge con(ecC)$,可得到能耗约束下吞吐量最大的调度.由调度进一步计算该调度下的实际能耗和吞吐量,可得到一个优化点,如算法 3 所示.当不存在满足能耗约束的调度时算法 3 返回空,否则返回优化点.其中,函数 *getUPPAALTrace* 用于得到 UPPAAL 通过最快路径搜索检测 $nta \models EF\ \ deadLock \wedge con(ecC)$ 后返回的路径,当性质不满足时返回空.

算法 3 getMaxThrUnderEC(nta_M , ecC)

输入: 时间自动机网络 nta_M 和能耗约束 ecC

输出: ecC 约束下的优化点 $optP_{ec}$

```
1:  OP  optP_ec
2:  σ = getUPPAALTrace ( nta_M , ecC )
3:  if σ == null  then
4:      return  null
5:  end if
6:  sch_σ = getSch( σ )
7:  ec = getEC( sch_σ )
8:  thr = getThr( sch_σ )
9:  optP_ec.ec = ec ,  optP_ec.thr = thr ,  optP_ec.sch = sch_σ
10: return  optP_ec
```

算法 4 getAllParetoPoints(nta_M)

输入: 时间自动机网络 nta_M

输出: 所有的帕累托优化点

```
1:  ParetoPList  paretoPList = null
2:  ecC = INT_MAX ,  thrC = R_MAX
3:  while true
4:      optP_ec = getMaxThrUnderEC( nta_M , ecC )
5:      if optP_ec == null  then
6:          return  paretoPList
7:      Else
8:          if optP_ec.thr == thrC  then
9:              paretoPList .removeLastP()
10:         end if
11:         paretoPList .add( optP_ec )
12:         ecC = optP_ec.ec -1,  thrC = optP_ec.thr
13:     end if
14: end while
```

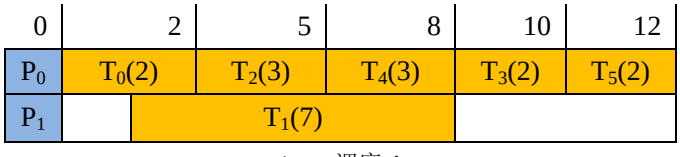
定义 9. 优化点是**帕累托优化点** $ParetoP(ec,thr,sch)$,当且仅当对任意优化点 $OP(ec',thr',sch')$,如果 $thr'=thr$,则 $ec'\ge ec$.

算法 4 可得到所有的帕累托优化点,其中变量 *paretoPList* 用于存储所有帕累托优化点, *INT_MAX* 和 *R_MAX* 分别表示最大的整型数和浮点数,函数 *removeLastP* 用于删除最后一个记录点.其首先调用算法 3 获得一个优化点,将其作为候选帕累托优化点加入 *paretoPList* .通过降低能耗约束,获得下一优化点,称为当前优化点.由于能耗约束值降低,因此当前优化点的吞吐量小于等于候选帕累托优化点的吞吐量.如果两优化点的吞吐量相等,则将候选帕累托优化点删除,将当前优化点作为候选帕累托优化点加入 *paretoPList* ;否则候选帕累托优化点即为找到的帕累托优化点,并将当前优化点作为下一候选帕累托优化点加入 *paretoPList* .降低能耗约束重复上述过程直至无优化点存在.由于能耗约束逐渐降低,因此算法不会遗漏问题空间内的任何帕累托优化点.

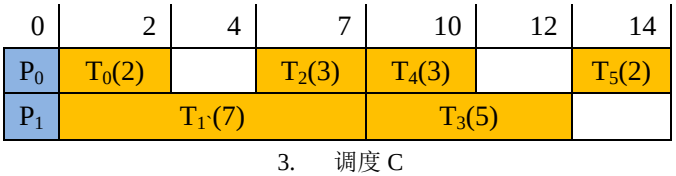
对系统模型 M_{AM} 的一次迭代,由其等价的时间自动机网络 nta_{AM} 和算法4可以得到如图4. 1(a)所示的四个帕累托优化点,每个帕累托优化点相应的静态调度如图 4. 1(b)所示.

调度	迭代执行总耗时(ps)	能耗(纳焦)	吞吐量(次/ps)
A	12	1.39	0.083
B	13	1.37	0.077
C	14	1.34	0.071
D	17	1.32	0.059

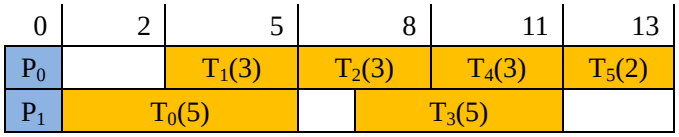
(a)帕累托优化点



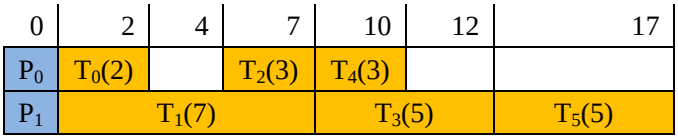
1. 调度 A



3. 调度 C



2. 调度 B



4. 调度 D

(b)调度策略图

图 4. 1 系统模型 M_{AM} 一次迭代执行的帕累托优化与调度

5 实验

5.1 实验设置

iDFOS[20]是同步数据流图的分析与优化工具,它以 XML 描述的执行平台和同步数据流图作为输入,用户可直接获得各种分析与优化结果而不需要关心优化过程.本文算法已经集成在工具 iDFOS 中,它将系统模型转换为 UPPAAL 时间自动机网络,然后利用工具 UPPAAL 提供的 verifyta 脚本[17],通过算法 4 获得所有能耗和吞吐量的帕累托优化点及相应的可行静态调度.如果在工具 iDFOS 中输入能耗和吞吐量约束,iDFOS 可以给出小于等于能耗约束并且大于等于吞吐量约束的所有帕累托优化点.除了能耗和吞吐量约束外,iDFOS 中还集成了迭代次数和边缓存大小等约束.

本文实验在 24M 缓存、284G 内存、2.90GHz 主频的服务器上进行.第一个实验的目的是评估不同同步数据流图对本文算法执行效率的影响.第二个实验的目的是评估不同执行平台对本文算法执行效率的影响.本文实验均在 1 次迭代和无边缓存约束的条件下进行.所有实验的执行平台如表 5. 1 所示,处理核的功耗见图 2. 1(a),例如执行平台 $Ar3m1$ 包含了 1 个 *ptA* 类型的处理核和 3 个 *ptB* 类型的处理核.由于本文实验目的是为了评估本文算法的可用性,因此实验结果中的吞吐量和能耗的单位可视为一个标准单位.

表 5. 1　实验执行平台

执行平台名		Ar1m1	Ar2m0	Ar2m2	Ar1m3	Ar3m1	Ar3m3	Ar4m4
核个数	ptA	1	2	2	1	3	3	4
	ptB	1	0	2	3	1	3	4

5.2 实验结果

5.2.1　综合实验

综合实验采用多个应用领域的同步数据流图,包括数字信号处理领域的调制解调器[13](Modem)应用程序,多媒体领域的 MP3 解码[9](MP3Decoder)、MP4 解码[14](MP4Decoder)和音频回波消除[9](AudioEchoCanceller)应用程序,以及加乘运算[3](AddMultiply)和经常用到的二分同步数据流图[9](Bipartite).

本实验中,所有同步数据流图的执行平台均是 $Ar1m1$,实验结果如表 5. 2 所示.它给出了每个系统模型通过本文算法得到的帕累托优化点数、迭代求解过程中调用 UPPAAL 脚本的次数、执行时间和每个系统模型的最大吞吐量及最小能耗对应的两个帕累托优化点.通过这两个帕累托优化点,可初步了解系统模型的能耗和吞吐量关系.同步数据流图的节点数、边数及迭代向量总和会影响状态空间的大小,进而影响算法的执行效率.通常节点数、边数及迭代向量总和越大,算法执行时间越长,帕累托优化点数越多,执行时间也越长.从表 5. 2 可以看出,本文算法就大多数同步数据流图在执行平台 $Ar1m1$ 上的一次迭代执行而言,在本实验的环境下,有较好的执行效率,最长的执行时间仅约为 18 秒.

表 5. 2　执行平台 Ar1m1 下的综合实验结果

同步数据流图名	Modem	MP3Decoder	MP4Decoder	AudioEchoCanceller	AddMultiply	Bipartite
节点/边数	16/19	14/16	5/10	4/6	6/6	4/8
迭代向量总和	48	27	13	70	6	73
吞吐量	0.031	0.05	0.077	0.013	0.059	0.010
最小能耗	3840	2160	520	5600	1320	5840
最大吞吐量	0.031	0.053	0.125	0.018	0.083	0.016
能耗	3840	2190	720	6350	1390	6800
帕累托优化点数	1	2	6	26	4	33
UPPAAL 执行次数	2	4	10	27	5	34
执行时间/ms	18010	4650	120	16580	40	3360

5.2.2　加乘实验

通过本文算法,加乘运算同步数据流图 G_{AM} 在不同执行平台下的优化结果如表 5. 3 所示.当处理核个数增加时,算法执行时间相应增加.对同步数据流图 G_{AM} 及 8 核处理器平台 $Ar4m4$,在本实验环境中的执行时间仅约为 3.6 秒,因此本文算法在多核执行平台下也有较高的执行效率.

通过对表 5. 3 的进一步观察,还可以发现一些有价值的信息:当 *ptA* 类型的处理核个数增加到 2 时,G_{AM} 达到最大吞吐量 0.1,继续增加 *ptA* 或 *ptB* 类型的处理核个数都不会提升其吞吐量,反而由于处理核空闲时产生的能耗,使得最大吞吐量下的能耗值增加.例如 G_{AM} 在执行平台 $Ar2m0$ 和 $Ar3m1$ 下所能达到的最大吞吐量均为 0.1,执行平台 $Ar2m0$ 产生的能耗为 1400,而 $Ar3m1$ 却达到 1590,因此就 G_{AM} 的一次迭代而言,处理核个数较少的执行平台 $Ar2m0$ 优于 $Ar3m1$.

表 5.3　加乘运算在不同执行平台下的实验结果

执行平台名	Ar1m1	Ar2m0	Ar2m2	Ar1m3	Ar3m1	Ar3m3	Ar4m4
吞吐量	0.059	0.1	0.071	0.071	0.091	0.1	0.1
最小能耗	1320	1400	1590	1730	1580	1990	2290
最大吞吐量	0.083	0.1	0.1	0.083	0.1	0.1	0.1
能耗	1390	1400	1690	1870	1590	1990	2290
帕累托优化点数	4	1	2	2	2	1	1
UPPAAL 执行次数	5	2	4	4	4	3	3
执行时间/ms	40	20	240	230	240	910	3640

6　总结

本文对流应用程序于异构多核处理器平台执行时,能耗和吞吐量的帕累托优化与调度问题进行探讨.将从问题域抽象出的系统模型转换成时间自动机网络,再利用实时模型检测工具 UPPAAL 找到能耗和吞吐量的帕累托优化点及相应的可行静态调度.通过实验评估可以看出,本文算法对大多数同步数据流图在一定核数目平台下的优化分析有较好的可用性.由于问题本身的复杂性和基于状态空间搜索的方法,当流应用程序或异构多核处理器平台规模达到一定程度时,可能会产生状态空间爆炸问题,因此如何继续有效地缩减状态空间有待更深入的研究.

References:

[1] Alur R, Dill DL. A theory of timed automata. Theoretical computer science, 1994,126(2):183–235.

[2] Larsen KG, Pettersson P, Yi W. UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer(STTT), 1997,1(1):134–152.

[3] Bouyer P, Fahrenberg U, Larsen KG, Markey N. Quantitative analysis of real-time systems using priced timed automata. Communications of the ACM, 2011,54(9):78–87.

[4] Lee EA, Messerschmitt DG. Static scheduling of synchronous data flow programs for digital signal processing. IEEE Trans. Comp, 1987,36(1):24–35.

[5] Stuijk S. Predictable Mapping of Streaming Application on Multiprocessors [Ph.D. Thesis]. Netherlands: Eindhoven University, 2007.

[6] Stuijk S, Geilen M, Basten T. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. DAC Proceedings of the 43rd annual Design Automation Conference, 2006. 899-904.

[7] Ravn AP, Srba J, Viqhio S. A formal analysis of the web services atomic transaction protocol with UPPAAL. ISoLA'10 Proceedings of the 4th international conference, 2010. 579-593.

[8] Madsen J, Hansen MR, Knudsen KS, Nielsen JE, Brekling AW. System-level verification of multi-core timed-automata. Proceedings of the 17th World Congress International Federation of Automatic Control Seoul, 2008. 9302-9307.

[9] Ahmad W, Groote R, Holzenspies PKF, Stoelinga M, Pol J. Resource-constrained optimal scheduling of SDF graphs via timed automata(extended version). Technical report accompanying the ACSD, 2014.

[10] Fakih M, Gruttner K, Franzle M, Rettberg A. Towards performance analysis of SDFGs mapped to shared-bus architectures using model-checking. Proceedings of the Conference on Design, Automation and Test, 2013. 1167-1172.

[11] Karp RM. A characterization of the minimum cycle mean in a digraph. Discrete Mathematics, 1978,23(3):309-311.

[12] Groote R, Kuper J, Broersma H, Smit GJM. Max-plus algebraic throughput analysis of synchronous dataflow graphs. Software Engineering and Advanced Applications(SEAA), 2012. 29-38.

[13] Bhattacharyya SS, Murthy PK, Lee EA. Synthesis of embedded software from synchronous dataflow specifications. Journal on VLSI Signal Process, 1999,21(2):151-166.

[14] Theelen B, Katoen JP, Wu H. Model checking of scenario-aware dataflow with CADP. In: Proceedings of the Conference on Design, Automation and Test in Europe, 2012. 653-658.

[15] Sriram S, Bhattacharyya SS. Embedded multiprocessors: scheduling and synchronization. CRC Press, 2009.

[16] Zivojnovic V, Schoenen R. On retiming of multirate DSP algorithms. In: Proceedings of the Acoustics, Speech, and Signal Processing, IEEE Computer Society, 1996. 3310-3313.

[17] The UPPAAL Model Checker. http://www.uppaal.com, 2005.

[18] Behrmann G, David A, Larsen KG. A tutorial on Uppaal. Formal methods for design of real-time systems, 2004. 200-236.

[19] Yang Y, Geilen M, Basten T, Stuijk S, Corporaal H. Iteration-based trade-off analysis of resource-aware SDF. Proceedings of the 14th Euromicro Conference on Digital System Design(DSD), 2011. 567-574.

[20] The iDFOS Tool. http://lcs.ios.ac.cn/~zxy/tools/idfos.html, 2014.