·移动互联与通信技术· 文章编号：1000-3428(2018)11-0083-07 文献标志码：A 中图分类号：TN911.7

异构信号处理平台中层次性流水线调度算法

杨平平，岳春生，胡泽明

（信息工程大学 信息系统工程学院，郑州 450001）

摘 要： 现有实时任务调度算法在系统异构性方面没有考虑处理节点计算能力的差异，导致任务划分不均衡。为此，根据异构信号处理平台实时任务的特点，提出一种面向异构信号处理平台的层次性流水线调度算法。基于数据流图的多层次任务划分算法并借鉴多层图划分思想，实现任务调度负载均衡和低通信同步开销。采用同步流水线调度方法达到低延迟实时数据处理的目的。实验结果表明，与事件触发数据流调度算法相比，该算法能有效提高异构信号处理平台的实时处理能力。

关键词： 异构信号处理平台；有向无环图；负载均衡；任务调度；同步流水线

中文引用格式： 杨平平，岳春生，胡泽明. 异构信号处理平台中层次性流水线调度算法[J]. 计算机工程，2018，44(11)：83-89.

英文引用格式： YANG Pingping，YUE Chunsheng，HU Zeming. Multi-level pipeline scheduling algorithm in heterogeneous signal processing platform[J]. Computer Engineering，2018，44(11)：83-89.

Multi-level Pipeline Scheduling Algorithm in Heterogeneous Signal Processing Platform

YANG Pingping，YUE Chunsheng，HU Zeming

（College of Information System Engineering，Information Engineering University，Zhengzhou 450001，China）

【Abstract】 The existing real-time task scheduling algorithm do not consider the difference of node computing power in system heterogeneity，which leads to the imbalance of task partition. Therefore，according to the characteristics of real-time task in the heterogeneous signal processing platform，a multi-level pipeline scheduling algorithm for heterogeneous signal processing platform is proposed. The multi-level task partition algorithm based on data flow graph and the idea of multi-layer graph partition are used to realize the task scheduling load balancing and low communication synchronization overhead. The synchronous pipeline scheduling method is used to achieve the purpose of low delay real-time data processing. Experimental results show that，compared with the event-triggered data flow scheduling algorithm，this algorithm can effectively improve the real-time processing capability of heterogeneous signal processing platform.

【Key words】 heterogeneous signal processing platform；Directed Acyclic Graph(DAG)；load balancing；task scheduling；synchronous pipeline

DOI： 10.19678/j.issn.1000-3428.0048806

0 概述

随着无线通信技术的快速发展，任务计算和数据信号的处理量出现爆炸式增长，对通信装备平台的快速研发、实时处理能力提出更高要求，使得软件无线电[1]平台成为了研究热点。目前，通用软件无线电平台有加泰罗尼亚大学的 ALOE[2]、爱尔兰圣三一学院的 IRIS[3] 和中国微软研究院的 SORA[4] 等，这些平台在信号处理方面应用较少，而且弗吉尼亚理工学院研究的 OSSIE[5] 和被誉为“黑客无线电”的 GNU Radio[6] 只能用来完成通信系统的建模和仿真学习[7]。异构信号处理平台通过软件和硬件的解耦，解决了不同平台间的可移植性和可扩展性等问题，充分利用了异构可重构特性。但是当前大多数实时任务调度算法是针对同构系统提出的，对于异构系统的实时调度算法研究还不成熟，需要进一步深入研究[8]。

异构平台相比于同构平台，由于其处理节点的异构性，提高了平台处理复杂任务的灵活性和效率。异构和同构平台算法的不同之处主要在于任务划分

基金项目： 国家科技支撑计划项目(2013BAH56F00)。

作者简介： 杨平平(1992—)，男，硕士研究生，主研方向为软件无线电、通信信号分析与处理；岳春生，教授、博士；胡泽明，副教授、博士。

收稿日期： 2017-09-26 **修回日期：** 2017-11-22 **E-mail：** yppcom@sina.com

阶段,即建立任务模型有无考虑处理节点的异构性带来的影响。异构平台的实时调度本质上就是资源映射和任务调度,在满足任务约束关系的前提下,对任务进行合理的划分,实现资源的负载均衡,减少任务的调度长度,同时构造低延迟的流水线调度,提高任务调度的实时性。

层次性流水线调度包括多层次任务划分和同步流水线调度。文献[9-10]提出一种应用于 ALOE 框架的资源映射算法,利用框架的同步机制和流水线调度[11],能够保证应用的服务质量,但该算法缺少对任务粒度划分的研究。文献[12-13]研究一种面向多核处理器实时数据流的低通信软件流水调度方法,利用整数线性规划理论对通信和计算资源进行统一建模,但该方法没有考虑分布式环境对调度的影响,缺少对网络拓扑和层次性任务并行优化的研究。文献[14]提出一种面向多核集群的数据流层次流水线并行优化方法,利用任务中的并行性构造低延迟流水线调度,但该方法针对同构集群采用的是均衡划分策略,在任务划分时没有考虑节点计算能力和节点间的通信速率。文献[15]针对异构计算中的非均衡划分问题,提出一种基于图的多级划分算法,利用处理器计算性能的不同将图与节点计算能力成比例划分,然而,该算法没有考虑任务间的依赖关系。

现有流水线调度算法在任务划分时未考虑硬件体系结构,即处理器节点的计算性能和通信带宽资源带来的影响。本文根据异构信号处理平台实时任务的特点,提出层次性流水线调度方法。针对异构信号处理平台的任务划分,提出负载均衡低通信同步开销的多层次划分算法;在平台对已有的事件触发调度实时处理能力不强的情况下,采用低延迟的同步流水线调度。最后在 ATCA 平台下,以雷达等典型应用为测试程序,验证该层次性流水线调度算法的有效性。

1 系统描述

1.1 异构信号处理平台

异构信号处理平台是将模块化、标准化和通用化的硬件单元以总线或交换方式连接起来构成通用平台,通过在平台上加载可复用、可移植、可扩展和易升级的标准化软件模块,实现各种信号处理功能[16]。异构信号处理平台在硬件体系架构上采用通用的高性能硬件平台,如 ATCA、CPCI、VPX 等,平台内存在大量不同类型的处理器,包括 FPGA、DSP、GPU、GPP 等。在软件架构上通过构建标准化、规范化的层次式新型软件体系架构,通过系统抽象屏蔽底层之间的差异,解决组件在异构平台中跨平台操作和可移植性的难点。异构信号处理平台软件体系架构如图 1 所示。

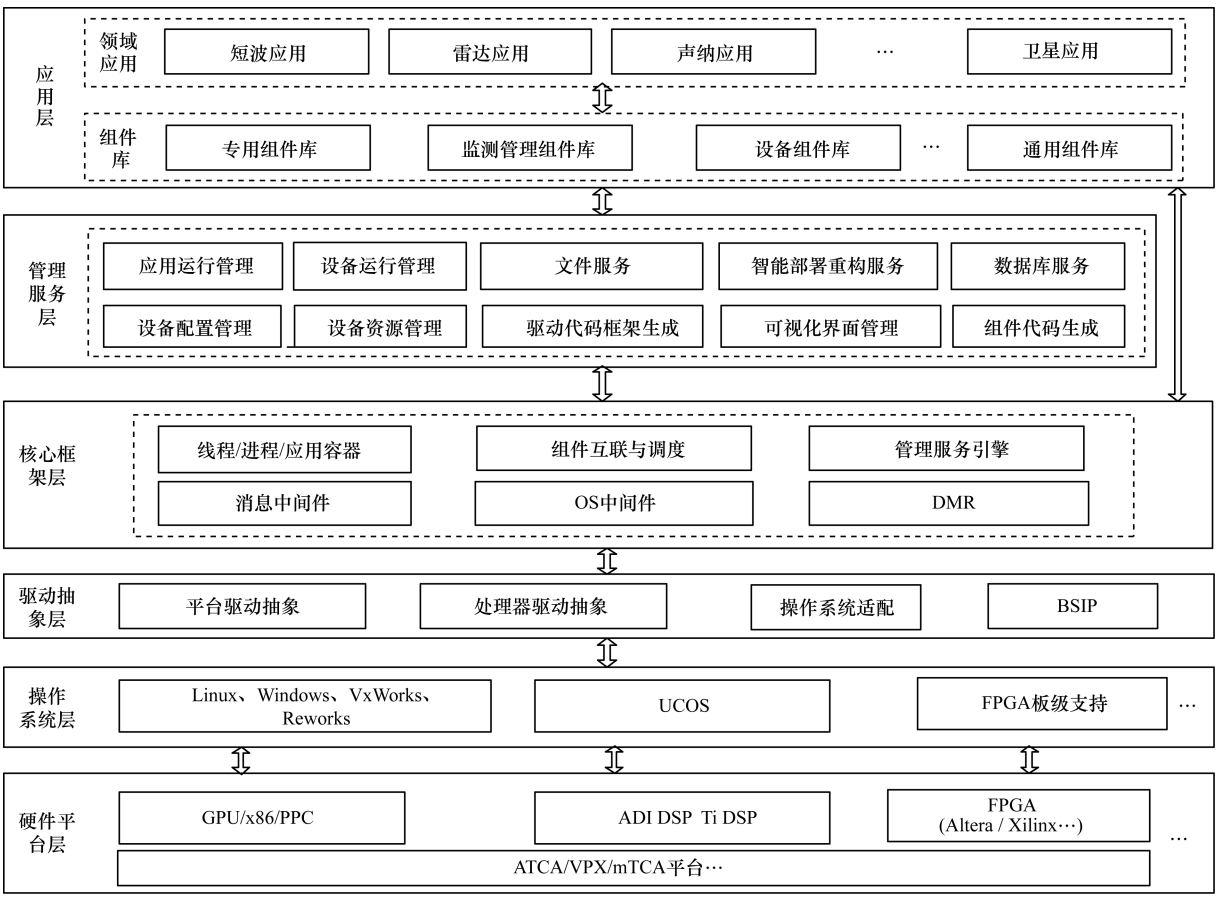


图 1 层次化新型信号处理平台软件体系架构

图 1 所示的异构信号处理平台软件体系架构主要分为硬件平台层、操作系统层、驱动抽象层、核心框架层、管理服务层和应用层。其中,操作系统层与驱动抽象层属于运行支撑服务层,能够屏蔽底层处理单元的操作系统、通信、内存及文件操作等硬件差异性,为上层提供统一规范的接口;核心框架层基于容器技术(包括 FPGA 容器、进程容器、线程容器等)屏蔽处理器任务调度上的差异,实现应用组件化调度服务;管理服务层包括系统运行管理环境和可视开发管理两大部分,提高应用系统开发的可视性以及便捷性;应用层包括组件库和应用控制台,负责完成信号处理功能。目前,异构信号处理平台的开发模式已经成为一种发展趋势。

1.2　异构信号处理平台任务模型

本文利用有向无环图(Directed Acyclic Graph, DAG)描述异构信号处理平台应用的处理流程,数据流程图中节点和有向边分别代表应用的组件和通信链路。软件无线电应用可分解为一系列具有相互依赖关系、周期性执行、可部署的软件模块或组件[17]。各软件模块之间存在数据流水依赖性,其中,某个组件的输出可以是相应下一步组件的输入,因此,系统模型可用一个四元数组表示,记为 $G=\{V,E,C,L\}$。其中,G 代表平台的一个应用,$V=\{v_1,v_2,\cdots,v_n\}$ 是应用所包含组件的集合,节点 $v_i\in V$ 表示应用中的一个组件,$E=\{e_{12},e_{23},\cdots,e_{ij}\}$ 表示依赖关系的有向边的集合,$E\subseteq V\times V$,$e_{ij}=(v_i,v_j)\in E$ 表示组件 v_i 和组件 v_j 有直接通信链接,组件 v_i 是组件 v_j 的前驱节点,$C=\{c_1,c_2,\cdots,c_n\}$ 表示组件的计算量集合,$C(v_i)$ 表示组件 v_i 的计算数据量,$L(v_i,v_j)$ 表示组件间的通信量集合。如图 2 所示,每一个应用都可以用 DAG 来表示。

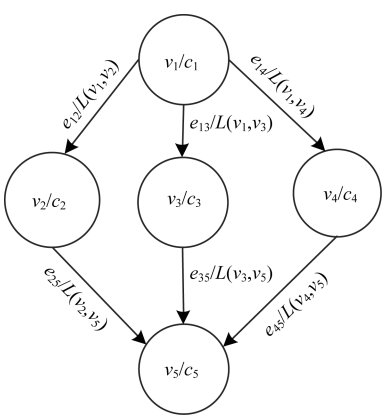


图 2　应用的 DAG 示意图

异构信号处理平台的硬件体系架构可以抽象为具有一定拓扑结构的硬件资源模型。其中,不同类型的处理器处理能力和内存大小都不相同,不同的总线连接传输速率和通信带宽也有很大差异。本文用 $P=\{N,H,W,T\}$ 表示硬件体系结构。其中,$N=\{N_1,N_2,\cdots,N_i\}$ 表示处理器的集合,$H=\{H_1,H_2,\cdots,H_k\}$ 表示处理器特征,取值为{0,1,2,3},0 代表 CPU,1 代表 FPGA,2 代表 DSP,3 代表 GPU。因此,P_N^H 用来表示异构处理器。W 是一个节点数 × 处理器个数($n\times i$)的二维数组,$w_{i,j}$ 表示组件 v_i 在处理器 P_j 上的计算开销,采用任务执行时间作为计算开销,由于节点的异构性,同一组件在不同处理器的执行时间有所不同,如表 1 所示,以组件 v_1 在处理器 P_1^0 执行时间为基准单位。$T=\{t_{1,2},t_{2,3},\cdots,t_{i,j}\}$ 表示组件在处理器间的通信开销,通信开销包括通信延迟和通信速率,如果组件 v_i、v_j 分配在同一节点上,则 $t_{i,j}$ 为 0。

表 1　异构信号处理平台任务执行时间

组件	异构处理器					
	P_1^0	P_2^0	P_3^1	P_4^1	P_5^3	P_6^3
v_1	1	4	5	4	6	4
v_2	5	6	4	5	4	7
v_3	3	4	2	4	5	6
v_4	4	5	3	5	6	5
v_5	4	6	4	3	5	4
v_6	7	5	6	4	7	3

2　异构信号处理平台实时任务调度策略

2.1　基于 DAG 的多层次任务划分方法

任务划分是实时调度的首要步骤,快速有效的划分方法是调度成功的关键。本文提出的基于 DAG 多层次划分算法借鉴了多层图划分的思想,主要由 3 个阶段组成:对 DAG 进行分层拓扑排序,顺序融合分组进行初始划分以及局部非均衡聚簇。任务划分的目的是为了保证节点间的负载均衡性,减少节点间的同步开销,构造低延迟高吞吐率的流水线调度。

2.1.1　DAG 的分层拓扑排序

分层拓扑排序是对 DAG 进行预处理。由图论可知,有向无环图节点存在入度值,入度值指的是邻接到某顶点弧的数目。预处理是根据节点的入度值,将有向无环图分成多层,每层的组件没有数据依赖关系的节点,可以进行并行处理。

DAG 分层算法描述如下:

算法 1　任务分层算法

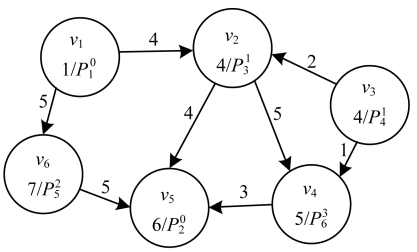
输入　原始 DAG $G(V,E)$

输出　经过分层后的 DAG $G:(V,E)$

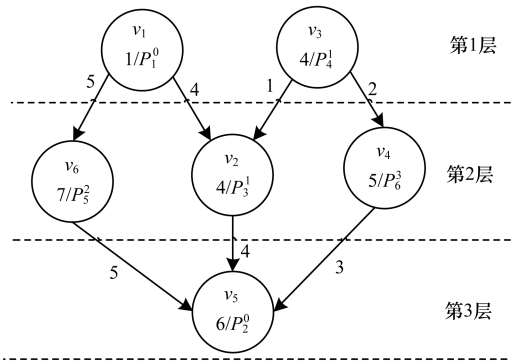
```
Void TopologicalSort (int Num); {//节点数目
Int gInDegree[MAX_NODE];
FindInDegree();//对各顶点求入度值
IniStack(S);//建立入度值顶点栈
for( int i = 0; i < Num; i++ )
if (! gInDegree[i] )
push (S,i);
```

```
//入度为 0 者进栈,k 层
While(! StackEmpty(S)){
pop(S,i);
gVisited[i] = true;//输出 i 节点
for(int e = gHead[i]; e! = -1; e = gEdges[e].next){
int v = gEdges[e].to;
gInDegree[v]--;
//对 i 顶点的每个邻节点的入度值减 1
if( ! gInDegree[v] ){
push (S,v);
//如果入度值减为 0,则入栈,k +1 层
}// END IF
}//END FOR
}//END WHILE
for (int i =0; i < Num; i + + )
if(! gVisited[i]) return ERROR; //存在回路
}//输出分层结果
```

图 3 所示为 DAG 分层拓扑排序处理的示例,其中,图 3(a)为一个异构信号处理平台应用的 DAG 映射到处理器的示例,V_i 表示组件,权值为组件在处理器上的执行时间,图 3(b)描述了经过任务分层后的 DAG。首先计算图 3(a)中各个顶点的入度值,将所有入度值为 0 的节点放在第 k 层,然后将上一层顶点的所有边消除,再计算新图入度值为 0 的顶点,放入第 $k+1$ 层,直到 DAG 中不存在入度值为 0 的节点。分层拓扑排序可以检测 DAG 中是否出现环路,防止执行时出现死锁。



(a)DAG示例



(b)分层排序后的DAG

图 3 DAG 分层拓扑排序示例

2.1.2 顺序融合聚簇算法

顺序融合聚簇是对分层处理后的 DAG 进行初始划分,将相邻的节点进行融合。任务划分粒度越小,子任务数越多,并行程度越高,但通信开销就越大;相反,任务划分粒度越大,通信开销越少,但并行度就越小。顺序融合聚簇方法是消除通信延迟划分算法中的一种,其将相邻节点融合可以降低通信开销的收益,但有可能会丧失后驱节点的并行性。因此,顺序融合聚簇时要尽量使任务之间的通信开销减小,又同时保持任务自身并行性以及预防出现环路情况。本文采用文献[14]的 DAG 粗粒度融合方法,将相邻节点融合产生的收益定义为:

$$Gain = \frac{L(v_i, v_j)}{C(v_i) + C(v_j)} \tag{1}$$

其中,任务模型中的元素 $L(v_i, v_j)$、$C(v_i)$、$C(v_j)$ 分别表示组件间的通信量、组件 v_i 的计算量(即负载)和组件 v_j 的计算量。

顺序融合算法描述如下:

算法 2 顺序融合算法

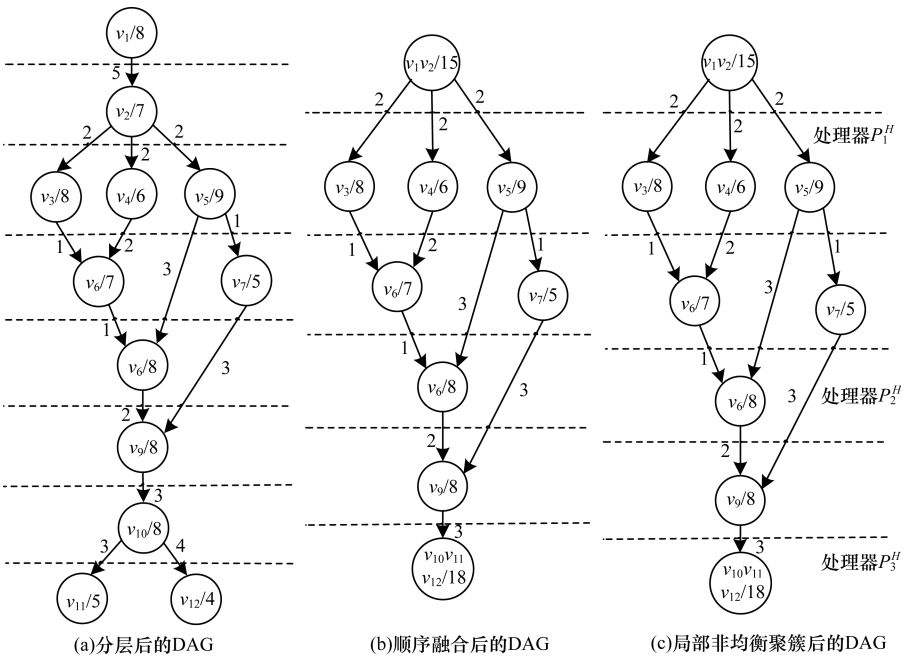
输入 经过分层后的 DAG $G:(V,E)$

输出 经过顺序融合后的 DAG $G:(V,E)$

```
Graph G =ConstructGraph(DAG);
VertexNum = Count(G);//当前节点数
averageWorkload = Weight(DAG)/ClusterNum;
// 理论上平均最佳负载
priorityQueue = ConstructpriorityQueue(G);
//收益为权值的优先队列
While(VertexNum > ClusterNum){
MaxGain = GetMaxGain(priorityQueue);
//优先队列中选择收益最大的融合
If(maxGain < =0)break;
Pair < Vi,Vj > maxGainCluster =
GetMaxGainCluster(priorityQueue,G);
ClusterWeight =Weight(cluster);
//计算当前 cluster 的负载
If (ClusterWeight < averageWoekload && ErrorDAG)
//当前相邻节点融合负载小于划分后理论平均负载值且新
//DAG没有环路
PriorityQueue.delete(maxGainCluster);
//优先队列删除最大收益值
newVertexr = Fused(Vi,Vj);
//融合后的新节点
G.update(Vi,Vj,newVertex);//更新 DAG
PriorityQueue.update(maxGainCluster,
newVertexr);//更新收益优先队列
-- VertexNum;
}//END WHILE
```

ClusterNum 是对节点数量设置的下限阈值,防止过度融合。顺序融合算法首先计算理论上的平均最佳负载,采用贪心算法计算相邻节点融合的收益,并构造以收益为权值的优先队列。然后,从队列中选择收益最大的相邻节点进行融合,计算融合后节点的负载,如果当前融合后的负载小于划分后理论平均负载且新 DAG 没有环路,那么就将参与融合的

相邻节点删除,插入融合后的新节点并更新 DAG,最后更新收益优先队列。算法经过多次迭代,当收益为负或者 DAG 中的节点数小于下限阈值时,DAG 中节点停止融合。图 4 所示为 DAG 多层次划分处理的示例,其中,图 4(a)中的节点 v_i 表示组件,权值为组件的负载,图 4(b)描述了经过融合后的 DAG。



(a)分层后的DAG　(b)顺序融合后的DAG　(c)局部非均衡聚簇后的DAG

图 4　DAG 多层次划分示例

2.1.3　局部非均衡划分算法

局部非均衡划分算法考虑到异构信号处理平台中处理器的计算能力不同,为保证任务实时调度适应更广泛意义上的异构负载均衡,任务粒度划分要随计算能力成正比例改变。局部非均衡算法要尽量保证任务间通信开销减小,注意预防执行死锁。

由表 1 可知,异构处理器的计算能力影响任务的执行时间,本文采用文献[15]的计算方法来表示处理器的计算能力。在异构信号处理平台中,单个处理器的计算能力(Computing Power,CP)是一个综合量,主要包括 I/O、内存读写、处理器、网络通信开销。因此,单个处理器的计算能力可以定义为:

$$CP=\frac{\dfrac{f_{\mathrm{i/o}}}{\bar{t}_{\mathrm{i/o}}}+\dfrac{f_{\mathrm{r/w}}}{\bar{t}_{\mathrm{r/w}}}+\dfrac{f_{\mathrm{comp}}}{\bar{t}_{\mathrm{comp}}}+\dfrac{f_{\mathrm{comm}}}{\bar{t}_{\mathrm{comm}}}}{f_{\mathrm{i/o}}+f_{\mathrm{r/w}}+f_{\mathrm{comp}}+f_{\mathrm{comm}}} \tag{2}$$

其中,$f_{\mathrm{i/o}}$、$f_{\mathrm{r/w}}$、f_{comp}、f_{comm}分别表示 I/O、内存读写、计算、通信操作的频率,$\bar{t}_{\mathrm{i/o}}$、$\bar{t}_{\mathrm{r/w}}$、$\bar{t}_{\mathrm{comp}}$、$\bar{t}_{\mathrm{comm}}$分别为上述各类操作的平均执行时间,可由基准测试程序得到[15]。

局部非均衡聚簇算法描述如下:

算法 3　局部非均衡聚簇算法

输入　经过顺序融合后的 DAG $G:(V,E)$,initPartitionMap(G)

输出　经过非均衡聚簇后的 DAG $G:(V,E)$

```
ProcessNum = Count(P);//实际处理器个数
PriorityQueue_CP = ConstructpriorityQueue(CP);
//构造以处理器计算能力为权值的优先队列
PriorityQueue_Workload = ConstructpriorityQueue(G);
//构造以任务负载为权值的优先队列
Match(PriorityQueue_CP,PriorityQueue_Workload);
//将处理器的计算能力与任务的负载进行匹配
averageRunTime = Weight(Cluster)/CP;
//计算理论上平均最佳执行时间(即最佳同步周期)
TopologicalSort[ ] = ConstructClusterTopoLogicSort(G);
//构造聚簇拓扑排序,进行划分
Int curPartitionNo = 0;//当前的划分编号
While(SearchClusterTopoLogicSort(G)){
//依次遍历 DAG 中分簇节点
CurPartitionWorkload = Weight(Vertex) +
Weight(curPatitionNo);
If(Weight(curPartitionWorkload) < averageRunTime ×
matched(CP)){
//根据任务负载量匹配相应的处理器
initPartitionMap.insert(curPartitionNo,Vertex);
//当前负载不足,添加节点
}//END IF
else
++curPartitionNo;
G.update(newVertex);//更新 DAG
}//END WHLIE
```

局部非均衡划分算法首先计算所有节点的负载以及系统中各个处理器的计算能力,构造相对应的优先队列。然后,将队列中任务的负载与处理器计算能力进行匹配,计算理论上平均最佳执行时间(即最佳同步周期),对 DAG 进行拓扑排序,依次遍历 DAG 中的节点,确定各个簇的划分编号。最后,根

据子任务负载情况匹配相应的处理器,对后驱节点依次进行融合,如果融合后形成新节点的负载小于最佳时间乘以处理器计算能力的值,那么添加节点,反复迭代上述过程直到确定划分的编号。经过非均衡划分后的 DAG 见图 4(c)。

2.2 同步流水线调度

异构信号处理平台采用同步流水线调度作为调度模型。同步流水线调度根据应用组件的依赖关系,将一个应用分割成若干个流水线阶段,同一阶段内的组件是相互独立的,同时采用全局同步时钟保证流水线各个执行阶段具有相同的时间片。在任一时间片内,组件最多只能执行一次处理任务,同时将数据传输到后驱组件,后驱组件要等待下一个时间片,才能处理前驱组件的输出数据。因此,同步流水线调度利用时间片消除了组件间前后依赖关系,实现了组件的并行执行。

同步流水线调度的性能取决于执行时间最长的组件,因此,任务划分的负载均衡至关重要。图 5所示为软件无线电应用分配到 2 个处理器的同步流水线调度,其中,应用包含 4 个组件 $V=\{V_1,V_2,V_3,V_4\}$,V_1和 V_2分配到处理器 P_1,V_3和 V_4分配到处理器 P_2。图 5 的调度流程存在透明的组件,表示组件尚未激活,其中,组件 V_2和 V_3是组件 V_1的后驱节点,是组件 V_4的前驱节点,因此整个应用被分割成 3 个不同的流水阶段。当应用经过 3 个时间片时,流水线进入满状态,此时同步流水线能够产生高吞吐量。

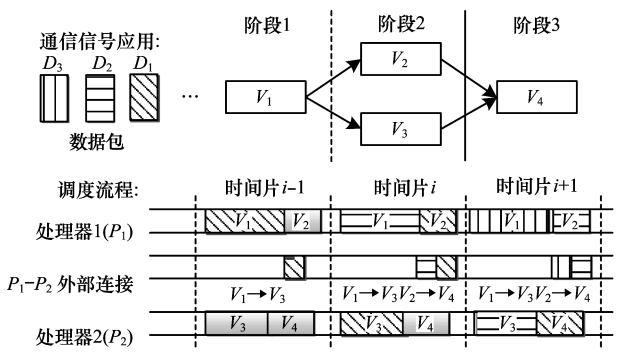


图 5 应用分配到 2 个处理器时的流水线调度

2.3 调度策略分析

异构信号处理平台的实时任务调度策略主要包括多层次任务划分和同步流水线调度。本文提出的基于 DAG 的多层次任务划分算法借鉴多层图划分的思想并对其进行改进,同时考虑任务之间的数据依赖关系、节点的异构性以及处理器间的通信速率。预处理阶段的目的是对 DAG 进行分层排序,使同层上的任务可以进行并行处理。初始划分阶段的顺序融合聚簇算法,采用贪心算法计算融合收益,减小通信开销。非均衡划分阶段考虑到节点的计算能力不同,采用广度优先搜索算法进行任务匹配,保证粒度划分与节点计算能力成正比例改变,确定划分后的簇与处理器之间的映射,实现了任务的负载均衡和低通信同步开销。最后采用同步流水线调度方法,实现节点的并行执行,达到了数据低延迟实时处理的目的。

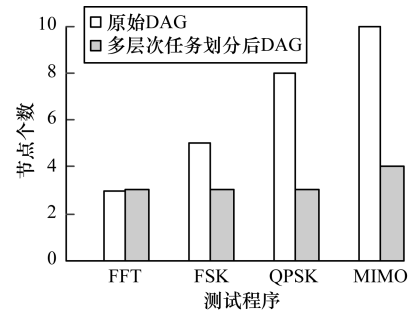
3 实验与性能分析

本文采用异构信号处理平台来测试多层次划分算法以及同步流水线调度的性能。异构信号平台由通用 PC 平台和高性能 ATCA 平台组成,2 个平台通过网线互连形成局域网,其中,PC 是域管理器客户端,用于组件的开发和应用的搭建;ATCA 是运行平台,主要实现应用的软件功能,对应用进行实际处理。在恒为公司 ATCA 平台内,插有计算刀片、承载板以及 AMC-FPGA 3 种板卡,计算刀片内存在 GPP 处理器,运行 Windows Server 2008 操作系统,AMC-FPGA 板卡内存在 DSP 处理器,移植 μC/OS II 操作系统,以及一块 Kintex-7 FPGA 处理器。平台内计算刀片板卡和承载板内通信协议为万兆网,板卡间通信协议也为万兆网,AMC-FPGA 板卡内处理器 FPGA 与 DSP 之间通过 RAPID I/O 总线相连,其性能参数配置如表 2 所示。

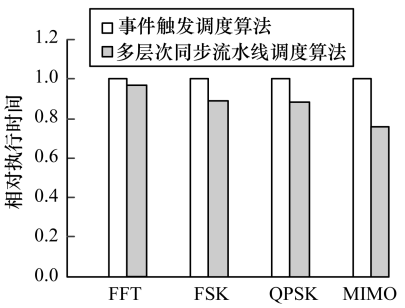
表 2 异构信号平台处理器参数配置

平台	处理器	性能参数	编译语言
PC 客户端	CPU	主频 3.3 GHz,内存 4 GB,带宽 1 Gb/s	C/C++
ATCA 平台	GPP-1	主频 2.8 GHz,内存 4 GB,带宽 1 Gb/s	C++
	GPP-2	主频 2.8 GHz,内存 4 GB,带宽 1 Gb/s	C++
	DSP	主频 1.8 GHz,内存 2 GB,带宽 1 Gb/s	C/C++
	FPGA-1	逻辑单元 325 000 个,内部存储资源 20 MB	VHDL
	FPGA-2	逻辑单元 700 000 个,内部存储资源 18 MB	VHDL

本文选取 FFT、短波 FSK、QPSK 和 MIMO 雷达信号处理 4 个典型应用作为测试程序,4 个测试程序的 DAG 的节点数和拓扑结构复杂度依次增加。其中,FFT 应用的 DAG 包含 3 个节点,FSK 包含5 个节点,QPSK 包含 8 个节点,MIMO 包含 10 个节点。针对平台目前采用的事件触发数据流调度算法与多层同步流水线调度算法进行对比实验,其中,事件触发调度算法按照应用原始 DAG 进行调度,不再进行任务划分。多层次流水线调度与事件触发调度算法的性能对比如图 6 所示。



(a)多层次任务划分结果



(b)不同调度算法的执行性能比较

图 6　多层次流水线调度与事件触发调度算法对比

图 6(a)为测试程序的 DAG 经过多层次划分结果的示意图,可以看出,测试程序 FSK、QPSK 和 MIMO 经过多层次划分 DAG 节点数明显减少,但对于 FFT 应用,划分结果并不理想,原因是 FFT 应用的组件较少,DAG 拓扑结构过于简单。实验测试选择的性能指标为应用完成执行的时间,图 6(b)为测试程序在 2 种调度算法下执行性能的比较,可以看出,测试程序随着 DAG 节点数以及拓扑结构复杂度的增加,多层次流水线调度算法执行时间呈递减趋势,执行时间越小,说明改进效果越好。但 FFT 应用在 2 种算法下执行时间基本相同,多层次同步流水线调度并没有取得很好的性能收益,原因是 FFT 应用的节点数太少且拓扑结构过于简单,但随着任务 DAG 节点数的增多,拓扑结构复杂度增加,多层次流水线调度的优势也越明显。因为事件触发调度算法在依赖节点数过多且约束关系复杂的情况下,其节点在运行时会频繁唤醒,所以带来的开销过大。MIMO 雷达信号处理应用节点数最多、拓扑结构最复杂,相对于事件触发调度算法,多层次同步流水线调度算法的运行时间能够缩短 25% 左右。

4　结束语

本文基于异构信号处理平台提出利用 DAG 的任务模型,设计实现多层次任务划分算法,保证任务节点间的异构负载均衡,减少同步开销。同时,在多层次任务划分的基础上设计同步流水线调度策略,实现节点的并行执行。实验结果表明,该算法适合异构信号处理平台中实时任务的调度。但是本文多层次任务划分算法还存在不足,在保证任务划分均衡的情况下降低节点通信开销,是下一步研究的方向。

参考文献

[1] MITOLA J. The software radio architecture[J]. IEEE Communications Magazine,1995,33(5):26-38.

[2] MAROJEVIC V, BALLESTÉ X R, GELONCH A. A computing resource management framework for software-defined radios [J]. IEEE Transactions on Computers,2008,57(10):1399-1412.

[3] ROY F L,LABABIDI R. A survey of frameworks and open environments applied to cognitive radio design[J]. International Journal of Emerging Technology and Advanced Engineering,2014,4(3):356-364.

[4] TAN K,LIU H,ZHANG J. Sora:high-performance software radio using general-purpose multi-core processors [J]. Communications of the ACM,2011,54(1):99-107.

[5] CHEN D, VANHOY G, BEAUFAIT M, et al. OSSIE/GNU radio generic component [C]//Proceedings of 2011 Wireless Telecommunications Symposium. Washington D. C. ,USA:IEEE Press,2011:1-5.

[6] MARLOW R L. Making radios with GReasy: GNU radio with FPGAs made easy [D]. [S. l.]: Virginia Polytechnic Institute and State University,2014.

[7] 李丹丹,马金全,杨平平. 信号处理平台中可重构组件的设计与实现[J]. 计算机工程,2018,44(5):33-39.

[8] 尹杨美,徐　成,刘　彦. 改进的异构多处理器的实时任务调度算法研究[J]. 计算机应用研究,2010,27(4):1236-1238.

[9] MAROJEVIC V. Computing resource management in software-defined and cognitive radios[D]. Barcelona, Spain:Universitat Politècnica de Catalunya,2009.

[10] GOMEZ I, MAROJEVIC V, GELONCH A. Aloe: an open-source SDR execution environment with cognitive computing resource management capabilities[J]. IEEE Communications Magazine,2011,49(9):76-83.

[11] SRIRAM S, BHATTACHARYYA S S. Embedded multiprocessors: scheduling and synchronization [M]. [S. l.]:CRC Press,2009.

[12] WEI H, YU J, YU H, et al. Software pipelining for stream programs on resource constrained multicore architectures[J]. IEEE Transactions on Parallel and Distributed Systems,2012,23(12):2338-2350.

[13] WEI H, QIN M. StreamTMC: stream compilation for tiled multi-core architectures[J]. Journal of Parallel and Distributed Computing,2013,73(4):484-494.

[14] 于俊清,张维维,陈文斌,等. 面向多核集群的数据流程序层次流水线并行优化方法[J]. 计算机学报,2014,37(10):2071-2083.

[15] 沈轶炜,曾国荪. 异构计算中一种图的非均衡划分算法[J]. 计算机科学,2006,33(6):260-263.

[16] MITOLA J. The software radio architecture[J]. IEEE Communications Magazine,1995,33(5):26-38.

[17] 贺靖卿,岳春生,胡泽明. 考虑综合性约束的软件无线电应用部署算法[J]. 计算机工程与应用,2017,53(15):239-243.

编辑　司森森