

一种面向嵌入式多核系统的任务调度方法

吕鹏伟,刘从新,沈绪榜

(西安微电子技术研究所,陕西 西安 710065)

摘 要: 针对嵌入式多核处理器资源有限的特点,提出了一种基于软件流水的任务调度方法.该调度方法使用整数线性规划方程对软件流水中的工作负载、通信开销和存储空间进行建模,通过实现多核处理器的负载均衡、减少核间通信开销和优化存储空间的使用来提高程序的性能.最终在 Revealer 处理器上进行实验,实验表明该调度方法相对于其他任务调度方法,程序性能获得不同程度的提高,能够更好地适应于嵌入式多核处理器.

关键词: 嵌入式系统;多核处理器;软件流水;任务调度

中图分类号: TP314

文献标识码: A

文章编号: 1000—7180(2017)04—0001—07

A Task Scheduling Method for Embedded Multicore Systems

LV Peng-wei, LIU Cong-xin, SHENG Xu-bang

(Xi'an Institute of Microelectronics Technology, Xi'an 710065, China)

Abstract: For the limited resources of the embedded multi-core processor, a task scheduling method based on software pipelining was proposed. The workload, communication overhead and storage resource of this method are modeled as an integer linear programming formulation, and by implementing load balance, reducing inter-core communication and optimizing the usage of storage on-chip to improve application performance. This method is experimented on the Revealer processor, and results show that this method improves program performance compared with other task scheduling method, and can be better adapted to the embedded multicore processor.

Key words: embedded system; multicore processor; software pipelining; task scheduling

1 引言

目前,多核处理器的应用日益广泛,然而多核处理器性能的挖掘严重依赖于多核编程模型.在众多的编程模型中,数据流编程(Data Flow Programming)模型被广泛应用于基于多核处理器的多媒体、图像处理和信号处理等应用领域,它将媒体应用特性与程序设计语言相结合,在简化编程的同时为程序在多核处理器下的执行提供了大量的并行机会.常见的流编程语言有 StreamIt^[1], Brook^[2], CUDA^[3], SignalPU^[4].为了充分挖掘处理器的性能,基于流编程语言的软件流水是一种常用的开发数据流程序并行性的有效方法, Mahlke 等人基于 StreamIt 语言采用软件流水的方法实现了在多核处理器上的

任务调度^[5]; W. Thies 等人使用静态数据流调度方法对程序进行流水化调度^[6]; Abhishek Udupa 等人基于 StreamIt 语言采用整数规划理论在异构多核系统上进行任务划分与调度^[7],最后通过代码生成器生成 CUDA 代码; Matthew J. 等人采用整数规划理论研究了数据流程序在 GPU 上的调度^[8]; Martin Hashemi 等人在分布式内存多核系统下基于流程序提出了一种新的数据流图 FORMLESS 实现任务的同步和并行^[9].对于软件人员来说挖掘嵌入式多核处理器的性能是一项艰巨的任务,本文面向国产航天专用嵌入式多核处理器(Revealer)提出了一种基于软件流水的任务调度方法对程序进行优化,并开发了适用于嵌入式多核系统的编程模型 MPMD,最后在 Revealer 处理器上测试了上述

方法.

2 Revealer 架构与编程模型

2.1 Revealer 处理器架构

Revealer 处理器是本单位自主研发的一款面向航天应用领域的多核 DSP 处理器, 内含 16 个同构 DSP 核, 核间通过 2D-meshed NoC 网络互联, 采用分布式存储, 各个 DSP 具有各自的私有存储 (Local Storage, LS), 私有存储采取哈佛结构, 数据存储与指令存储区分开, 核间通过 DMA 可以进行通信, 如图 1 所示.

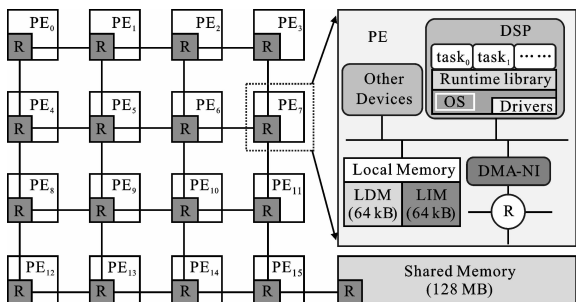


图 1 Revealer 处理器架构

Revealer 具有在线动态可重构能力, 支持 MPP (Massively Parallel Processing)、流计算、多任务等多种运行模式, 集成了可重构算术指令的简单定/浮点计算模式和复杂计算模式. 其中, 简单定/浮点可重构计算模式主要是基本定/浮点计算的可重构扩展, 可实现相同计算类型下的无相关冲突的数据流水计算; 复杂可重构计算模式主要包括 32 位定/浮点类的蝶形运算、累加、乘累加、矩阵矢量乘和 (共轭) 复数乘等, 还包括 16 位定点类的级联乘累加运算, 从而最大化可重构 DSP 处理器核内部资源利用率, 进而提升可重构 DSP 处理器核整体性能. 为方便表达, 本文的后续部分将 Revealer 多核处理器中的 DSP 简称为处理器单元 PE (Processor Element).

2.2 MPMD 编程模型

MPMD 是西安微电子技术研究所开发的一种多核编程模型, 其输入语言是对标准 C 语言的扩展, 该模型允许程序员在编写程序时通过特定的关键字指定可以被并行调度的代码片段, 并通过源到源 (source-to-source) 编译器对源文件进行编译生成 kernel (任务函数), 并构建 kernel 之间的数据依赖图, 当代码运行时运行时库 (runtime library) 会根据数据之间的依赖关系对 kernel 进行调度. MPMD 的

前端编译系统添加了支持数据流编程的关键字 “#pragma stream start” 和 “#pragma stream end”, 表示应用程序中可以进行流处理的代码区域, 如图 2 所示.

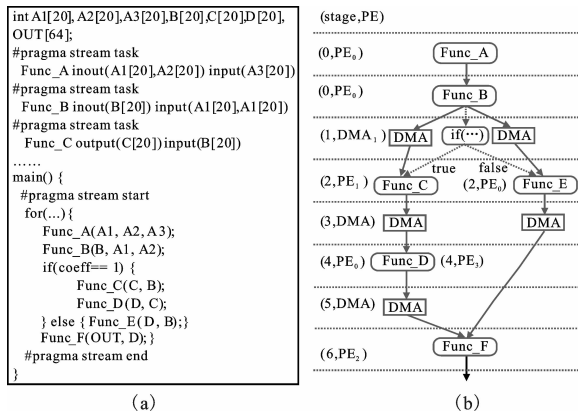


图 2 MPMD 编程实例

MPMD 的代码生成器会为每一个 PE 生成线程化的代码, Revealer 多核处理器运行嵌入式实时多核操作系统, 其中一个运行主线程 (Main thread) 的 PE, 通过 MPMD 运行时库中的软件流水调度器 (Software Pipelining Scheduler), 将数据流图中的 kernel 结点调度到 Revealer 多核处理器所包含的 PE 上, 并为任务分配输入/输出数据的缓冲区以及相应的 DMA 操作来满足给定的存储约束和数据流模型的时序约定.

3 MCPM 软件调度

为了方便表达, 我们假设处理器有两个处理单元 PE0 和 PE1, 程序有两个不同的任务 u 和 v , 且这两个任务被分配到不同的处理单元上, DMA 操作作为 d , 如图 3(b) 所示. 随着数据不断流入到任务中, 多个任务之间可以形成软件流水迭代执行, 启动间隔为 II (Initiation Interval), 如图 3(c) 所示. 从图 3 中可以看到, II 越短流水线的吞吐率越高, 程序的执行速率越快.

本文提出 MCPM (Memory-constrained and Communication-constrained Processor Element Mapping) 算法所要解决的问题的关键是: 在最小化通信开销且满足片上局部内存约束的同时合理地分配处理器和 stage 以提高流水的速度 (即最小化 II). 算法主要可以分为三部分:

(1) 任务负载均衡. 即通过对任务进行合理地划分, 求解任务与处理器间合理的映射方案, 达到核间负载均衡以寻找最小的 II ;

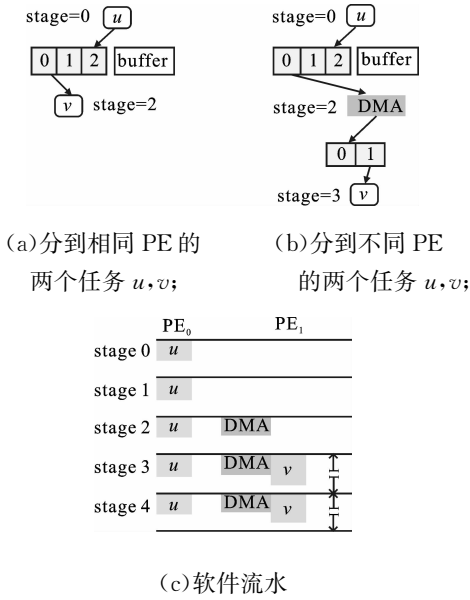


图3 任务级软件流水的例子

(2)通信开销最小化. 即引入通信开销的约束条件,对(1)中的任务映射方案进行调整再次寻找最小的 Π ;

(3)适应片上存储限制. 通过调整的 stage,可以减少核间的通信数据对 buffer 的使用,从而减少片上存储的溢出,最终提高存储访问性能.

3.1 MCPEM 算法的 ILP 方程

MCPEM 算法中的所涉及的关键问题:负载平衡、通信开销和存储限制都被抽象为整数线性规划方程(Integer Linear Programming, ILP).

3.1.1 任务的划分与 PE 的分配

数据流图用 $G = (V, E)$ 表示,结点 $v \in V$, 表示任务或者条件分支语句; $e \in E$, 表示结点之间的数据传输; 令 $M = |V|$ 表示任务的数目, P 表示处理单元 PE 的数目; 使用 0-1 变量 $m_{v,p}$ 表示任务 v 是否被分配到处理单元 p 上, $m_{v,p} = 1$ 表示任务 v 分配到 p 上, 否则 $m_{v,p} = 0$. 每个任务只能映射到唯一的处理器核上:

$$\sum_{p=0}^{P_{\max}-1} m_{v,p} = 1 \quad (1)$$

$\omega(v)$ 表示任务 v 的工作负载, 在处理器 p 上的运行时间不能超出软件流水的启动间隔 Π :

$$\sum_{v=0}^{M-1} \omega(v) \times m_{v,p} \leq \Pi \quad (2)$$

为达到负载平衡:

$$\text{Min } \Pi \quad (3)$$

3.1.2 通信开销限制

假设任务 v 被分配到处理器 p 上, 而 u 不在 p

上, 则存在 $m_{v,p} = 1, m_{u,p} = 0$. 另外假设数据通信操作都是由数据的接受者发起的. 0-1 变量 $d_{u,v,p} = 1$ 时, 说明任务 u 和任务 v 之间需要 DMA 操作, $\langle u, v \rangle$ 表示任务节点之间的边, u 是数据的生产者, v 是数据的消费者, 否则 $d_{u,v} = 0$, 如不等式(4)所示.

$$d_{u,v,p} = \lceil (m_{v,p} - m_{u,p} + 1) / 2 \rceil \quad (4)$$

$c_{u,v}$ 表示两个结点 u, v 之间的通信开销, B 表示通信带宽. 数据通信时间不大于启动间隔 Π :

$$\sum_{\langle u, v \rangle \in E} m_{v,p} \times d_{u,v,p} \times \frac{c_{u,v}}{B} \leq \Pi \quad (5)$$

3.1.3 片上存储限制

任务 v 的前驱结点 $u \in P(v)$ 为数据生产者, 任务 v 的后继结点 $w \in S(v)$ 为数据消费者. st_v 表示任务 v 的阶段值, $sd_{u,v}$ 和 $sd_{v,w}$ 分别表示任务 u 与 v 之间、 v 与 w 之间数据通信操作的阶段值, DMA 操作的 stage 要满足式(6)和式(7).

$$st_v \geq \max_{u \in P(v)} \{sd_{u,v}\} + \sum_{p=0}^{N-1} d_{u,v,p} \quad (6)$$

$$\min_{w \in S(v)} \{sd_{v,w}\} \geq st_v + \sum_{p=0}^{N-1} d_{u,v,p} \quad (7)$$

PE 的局部存储器采取哈佛结构, 指令存储与数据存储分开, tcm_k 表示任务 k 的代码所需要的存储空间大小, tdm_k 表示任务 k 的数据所需要的数据空间大小, lim_p 和 ldm_p 表示 p 处理器指令存储区和数据存储区的大小. 不等式(8)表示指令存储空间约束.

$$\sum_{v=0}^{|T|-1} tcm_v \times m_{v,p} \leq lim_p \quad (8)$$

$dm_{v,p}$ 表示任务 v 在处理器 p 上所占据的数据空间, 等式(9)表示任务 v 在处理器 p 上所需要的数据存储区大小.

$$dm_{v,p} = tdm_v \times m_{v,p} \quad (9)$$

式(10)和式(11)分别表示分配到处理器 p 上的任务 v 所需要的数据输入缓冲区和输出缓冲区的大小, $bi_{v,p}$ 和 $bo_{v,p}$ 分别表示任务 v 在处理器 p 上的输入缓冲区和输出缓冲区的大小.

$$bi_{v,p} = \sum_{u \in P(v)} (st_v - sd_{u,v} + 1) \times m_{v,p} \times c_{u,v} \quad (10)$$

$$bo_{v,p} = \sum_{u \in S(v)} (sd_{v,w} - st_v + 1) \times m_{v,p} \times c_{v,w} \quad (11)$$

不等式(12)表示处理器 p 的数据存储空间约束.

$$\sum_{v=0}^{|T|-1} (dm_{v,p} + bi_{v,p} + bo_{v,p}) \leq ldm_p \quad (12)$$

3.2 MCPEM 算法

式(1)~(12)构成 MCPEM 算法. 为了求解目

标方程,必须找到初始的最小启动间隔 II , $RecMII$ 表示递归约束,即流图中最长环路上的工作负载总和,由于我们所使用的数据流图是无环的,所以 $RecMII=0$; $ResMII$ 表示资源约束,本文中将其设置为图中所有任务结点(不考虑数据依赖)的工作负载的平均值(负载总和/PE 数目)。

在求解 MCPEM 方程之前,要对数据流图中的任务结点和 DMA 结点进行 stage 赋值,如算法 1 所示. 首先对 $G=(V,E)$ 进行拓扑排序,然后使用算法 1 进行 stage 赋值,假设 p 是数据生产者, n 是数据消费者, d 表示 DMA 操作。

算法 1 为流图 G 中每个结点设置 stage

输入: L 是 G 中节点的拓扑队列以及 PE 的调度方案;
输出: 流图 G 中的结点获得 stage;

```

1. BEGIN
2. FOR all node  $n$  in  $L$  DO
3.   FOR all parent  $p$  of node  $n$  DO
4.     IF  $PE\_Assign(p) \neq PE\_Assign(n)$  THEN
5.       Temp_stage = StagePE[ $p$ ]+2;
6.       StageDMA[edge( $p,n$ )] = StagePE[ $p$ ]+1;
7.     ELSE Temp_stage = StagePE[ $p$ ];
8.   END IF
9.   IF Temp_stage > Current_stage THEN
10.    Current_stage = Temp_stage[ $n$ ];
11.   END IF
12. END FOR
13. StagePE[ $n$ ] = Current_stage;
14. END FOR

```

MCPEM 算法将数据流图作为输入,调度方案作为输出,具体分为以下几步:

(1) 计算初始 II ;

(2) 求解 (1) ~ (5) 获得 PE 调度方案 PE_Sched , 然后进入 (III);

(3) 若 $PE_Sched = \Phi$, 则 $II = II * (1 + 1\%)$, 返回 (II) 继续处理; 否则使用算法 1 对处理器调度方案 PE_Sched 中的 kernel 结点设置 stage, 然后进入步骤 (IV);

(4) 将 (III) 中选择的处理器调度方案及其分配的 stage 代入 (6) ~ (12) 方程, 若对于所有的结点都满足约束, 则进入 (VI); 若存在不满足约束的结点, 将其放入 $MemOverflow$ 集合中, 然后进入 (V);

(5) 使用算法 2 对 $MemOverflow$ 中 DMA 的 stage 进行优化, 进入 (VI);

(6) 保存调度方案 PE_Sched 和 stage 编号, 算法结束;

由于 Revealer 处理器的 PE 上的局部存储 LS 有限, 所以必须对 PE 存储区的使用进行限制, 找出不满足约束的 PE 加入到 $MemOverflowPE$ 集合中, 需要对 stage 的调整来减少 PE 上局部存储空间的溢出. 具体调整细节如算法 2 所示, $SrcDMA[d]$ 和 $DstDMA[d]$ 分别表示 DMA 操作的源结点和目的结点, $PE_Assign(SrcDMA[d])$ 表示源结点所在的 PE, $FreeMemForPE[p]$ 表示处理单元 p 可调整的存储空间, $FreeMemOfSrcPE(PE_Assign(SrcDMA[d]))$ 表示源结点所在的 PE 上可用的存储空间。

算法 2 stage 分配方案的优化

输入: 流图 G , 集合 $MemOverflowPE$;
输出: 流图 G 中的结点获得新的 stage;

```

1. FOR all PE  $p$  in  $MemOverflowPE$  DO
2.   FOR all input DMA  $d$  of  $p$  DO
3.     IF ( $PE\_Assign(SrcDMA[d]) \neq p$  & StagePE[ $p$ ] - StageDMA[ $d$ ] > 1) THEN
4.       Addto(DMAStageOpt[ $p$ ],  $d$ );
5.       FreeMemForPE[ $p$ ] += FreeMemOfSrcPE(PE_Assign(SrcDMA[d]));
6.     END IF
7.   END FOR
8.   IF (FreeMemForPE[ $p$ ] > BufferUsed[ $p$ ] - Mem[ $p$ ]) THEN
9.     Addto(PEStageOptQue,  $p$ );
10.    SpilledBuffer[ $p$ ] = BufferUsed[ $p$ ] - Mem[ $p$ ];
11.   END IF
12. END FOR
13. SortByCanAllocMem(PEStageOptQue, SpilledBuffer)
14. FOR PEStageOptQue != NULL DO
15.   PE  $p$  = getNextStageOptQue();
16.   SortByDMANeedsBuffer(DMAStageOpt[ $p$ ]);
17.   FOR all DMA  $d$  in DMAStageOpt[ $p$ ] DO
18.     IF (BufferUsed[ $p$ ] < Mem[ $p$ ]) THEN break;
19.     FOR  $PE\_Assign(SrcDMA[d]) \neq p$  & StagePE[ $p$ ] - StageDMA[ $d$ ] > 1 DO
20.       IF (buffer( $d$ ) + BufferUsed[ $p$ ] < Mem[ $p$ ]) THEN
21.         StageDMA( $d$ ) ++;
22.         BufferUsed[SrcDMA[ $d$ ]] += buffer( $d$ );
23.         BufferUsed[DstDMA[ $d$ ]] -= buffer( $d$ );
24.       ELSE break;
25.     END IF
26.   END FOR
27. END FOR
28. DelPE( $p$ , PEStageOptQue);
29. UpdateStageOptQue(PEStageOptQue);

```

30. END FOR

算法 2 的第 1~7 行找出 p 的所有输入 DMA 操作中 stage 可以调整的 DMA,以及所有对于 p 结点可调整的存储空间;第 8~11 找出可调整的存储空间大于内存溢出空间的处理单元 p ,然后加入到 PESTageOptQueue 队列中等待处理,并计算 p 溢出的存储空间大小;第 13 行 PESTageOptQueue 队列中的 PE 按照溢出的存储空间由大到小进行排序,溢出空间大的 PE 将被优先进行调整;第 14~31 行对 PESTageOptQueue 中的处理单元 p 所对应的输入 DMA 的 stage 进行处理,第 16 行 SortByDMA-NeedsBuffer(DMAStageOpt[p])根据一次 DMA 传输所需要的 buffer 大小,对处理单元 p 所有的输入 DMA 进行排序,然后在第 17~28 行中按照顺序依次对 DMA 的 stage 进行调整,第 28~29 行删除已经处理过的处理单元 p ,然后更新 PESTageOptQueue 队列。

4 实验结果及分析

本实验是在 Revealer 处理器上进行试验的,采用 MPMD 编译系统作为多核编译系统的前端编译环境,后端编译环境使用 revealer-elf-gcc, revealer-elf-as 以及 revealer-elf-gdb 等基于 GNU 开源编译器工具链上改进和优化的交叉编译工具链. 为了方便描述,将(1)~(3)组成的方程称为 PEM(Processor Element Mapping),将(1)~(5)所组成的方程称为 CPEM(Communication-constrained Processor Element Mapping). PEM 表示只有负载平衡限制的算法,CPEM 表示同时考虑负载均衡和通信开销的算法,MCPEM 表示同时考虑负载均衡、通信开销和存储限制的算法。

4.1 MCPEM 的性能分析

为了展示 MCPEM 算法的有效性,分别比较了没有使用任何调度算法的情况(用 None_Sched 表示),与使用 PEM 算法,CPEM 算法和 MCPEM 算法三种情况下测试程序的执行时间,为了直观的表现性能变化,将 None_Sched 的程序执行时间归一化,如图 4 展示的使用不同的算法所取得的性能提升. None_Sched 的执行时间明显慢于后面三种算法,Revealer 处理器中的 16 个 DSP 没有被充分利用,测试程序是按照单线程执行的. PEM 算法将任务分配到 16 个 DSP 核心上,算法主要目标是达到负载均衡,并没有考虑到通信开销和片上存储的约束,然而通信开销会严重制约多核之间的并行执行,

CPEM 在 PEM 基础上又添加了通信开销的约束,可以看到 CPEM 算法的性能比 PEM 有明显的提高,因为通过将通信开销时间限制到 II 范围之内,在多任务形成的软件流水调度处于稳定状态之后,DMA 操作就可以隐藏于流水阶段中,通信就可以与任务重叠执行不会影响任务的执行时间,所以程序的性能得到提升. 考虑到 Revealer 处理器中 DSP 的片上存储有限,当 DMA 操作所需要的缓冲区超过片上可用存储大小的时候,就会导致内存溢出(spilled),溢出的数据就要被放到片外的共享存储中,存储访问时间延迟较长,这显然会严重影响到程序的性能,MCPEM 算法在 CPEM 的基础上对片上局部存储的使用增加了限制条件,并使用算法 2 对 DMA 的 stage 进行优化,减少片上存储的溢出. 所以 MCPEM 算法的性能要比 CPEM 算法的性能高,如图 5 所示。

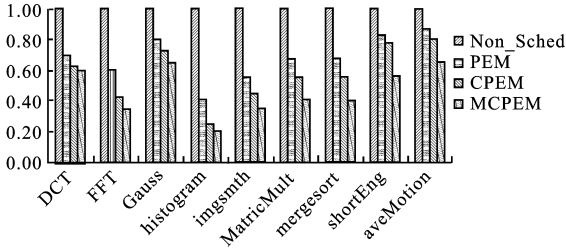


图 4 算法的性能比较

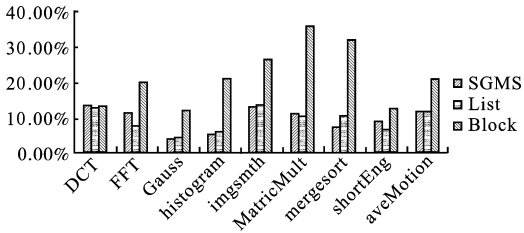


图 5 相对其他算法 MCPEM 在启动时间上的减少

4.2 MCPEM 与其他算法的性能比较及分析

本节将 MCPEM 算法与 SGMS(Stream Graph Modulo Scheduling)^[5] 算法、启发式表调度算法(Heuristic List)^[10] 和基本块的调度算法(Block)^[11] 三种算法进行比较. SGMS 是一种基于整数线性规划理论的算法,主要关注的是计算负载的平衡,并没有考虑到通信开销带来的性能损失;启发式表调度(List)是一种基于优先级的表调度方法;基本块调度在很多流式处理系统中被广泛应用,但是在基本块调度中来自不同迭代的任务不能重叠执行. 针对每一个测试程序给出了 MCPEM 与其他三种调度方法在启动时间、实际执行时间、通信开销方面的性

能分析与比较,如图 5、图 6 和图 7 所示。

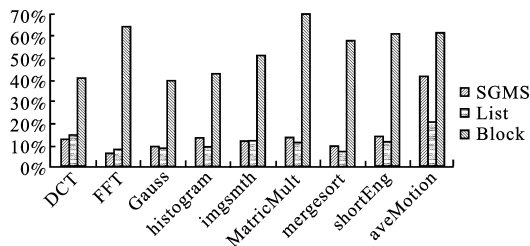


图 6 相对其他算法 MCPEM 在实际执行时间上的减少

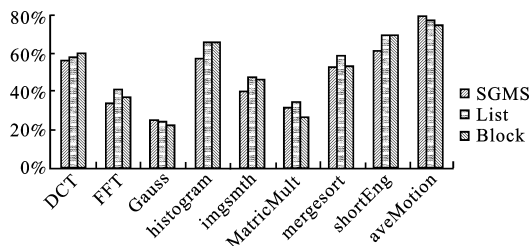


图 7 相对其他算法 MCPEM 在通信开销上的减少

4.2.1 通信开销性能分析

在软件流水算法中,启动时间越短,流水线执行速率越高,图 5 显示的是 MCPEM 调度与 SGMS 调度, List 调度以及 Block 调度相比在启动时间 (Initiation Interval) 上分别缩短 4.0%~13.6%, 4.4%~13.5% 和 12.1%~35.9%。从图中可以看出,对于测试程序例如 Histogram 和 Gauss 来说, MCPEM 调度相对于 SGMS 调度和 List 调度仅取得约 4%~6% 的性能提高,而对于那些计算量和通信开销比较大的测试程序,例如 imgsmth, MatrixMult 以及 aveMotion, MCPEM 在启动时间上会取得更好的表现,这是因为这些程序具有更大的数据流图,为 MCPEM 提供了更多的并行机会来提高性能。

对于程序实际的执行时间,如图 6 所示 MCPEM 算法在实际的执行时间上分别比 SGMS, List 以及 Block 缩短了 6.7%~41.8%, 7.5%~20.8% 和 40.4%~69.4%。这是因为 MCPEM 同时考虑了通信开销与计算负载,并将通信开销与计算负载进行重叠执行,隐藏了数据通信带来的时延,而 SGMS 调度只考虑了计算负载, Block 调度算法没有将不同迭代中的任务进行流水化执行,并且二者都将通信开销暴露出来,这严重影响程序性能,另外 MCPEM 的处理器分配算法优于 List 的启发式算法,所以在程序的实际执行时间上也优于 List 调度。

在通信开销方面的比较如图 7 所示,与 SGMS,

List, Block 等调度算法相比 MCPEM 算法在通信开销方面的减少分别为 24.7%~79.6%, 24.4%~77.5%, 22.3%~75.2%。一些测试程序,例如图 7 中的 FFT, Gauss, 以及 mergesort 来说,在实际执行时间上 MCPEM 只比 SGMS 调度和 List 调度提高了 10% 左右,这是由于这些测试程序所需要的 DMA 操作较少,即通信开销较少,所以在通信性能上的提高并不明显。MCPEM 调度算法不仅通过通信开销约束减少了大量的通信开销,同时通过对 DMA 操作的 stage 进行优化,减少了片上存储于片外存储之间的 DMA 操作,大大减少了存储访问带来的延时,所以能够比 SGMS、Block 和 List 调度能取得更好的性能。

4.2.2 存储性能分析

本节分析片上存储限制对软件流水性能的影响,图 8 显示的是程序执行性能与片上存储尺寸之间的关系。在实验中我们逐步增加局部存储的容量从 2 kB 到 64 kB,将 64 kB 时的性能作为参考标准进行归一化,图 8 的实验结果显示随着片上存储空间增加,所有的测试程序的性能都出现了不同程度的提高。程序性能之所以随着局部片上存储空间的增加而提高,是因为当局部存储不足时,任务产生的数据将被放到片外存储器上,数据的访问需要通过 DMA 操作在片上存储与片外存储之间进行数据的传递,会增加存储访问延迟,影响流水线的性能。从图 8 来看,对于一些测试程序来说,增加局部存储对其性能影响并不明显,例如 DCT, shortEnergy, 流水线性能的变化相对比较平缓,这是由于这些测试用例产生的数据流图经过 stage 优化算法的调整, DMA 操作的 stage 设置更加合理,会减少处理器上 buffer 的使用量,即减少了数据从片上存储溢出到片外存储的机会,从而减少了因数据通信带来的时延。

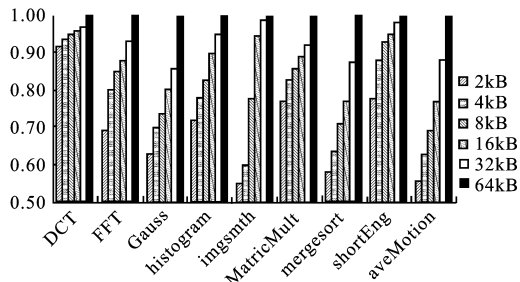


图 8 存储对性能的影响

5 结束语

本文基于嵌入式多核处理器提出了一种任务调度方法 MCPEM,该方法主要非为三个阶段:任务划分、通信开销限制和存储访问性能优化. MCPEM 利用整数线性规划理论对软件流水中的通信开销、任务和局部存储进行统一的形式化建模,求解出最佳调度方案.最后通过实验与其他算法进行对比表明了 MCPEM 能有效的解决嵌入式多核上的任务调度问题.

参考文献:

- [1] Thies W, Karczmarek M, Amarasinghe S P. Stream-It: a language for streaming applications[C]// Proceedings of the 11th International Conference on Compiler Construction. UK, London: Springer, 2002:179-196.
- [2] Buck I. Brook for GPUs: stream computing on graphics hardware[J]. ACM Transactions on Graphics, 2004, 22(3): 777-786.
- [3] David Kirk. NVIDIA CUDA software and GPU parallel computing architecture[C]//Proceedings of the 6th international symposium on Memory management. USA, New York: ACM, 2007:103-104.
- [4] Farouk Mansouri, Sylvain Huet, Dominique Houzet. SignalPU: a programming model for DSP applications on parallel and heterogeneous clusters[C]// Proceedings of the 2014 IEEE Intl Conf on High Performance Computing and Communications. USA, Washington, DC: IEEE Computer Society, 2014: 937-944.
- [5] Mahlke S, Kudlur M. Orchestrating the execution of stream programs on multicore platforms[C]//Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation. USA, New York: ACM, 2008: 114-124.
- [6] Thies W, Karczmarek M, Amarasinghe S. Phased scheduling of stream programs[C]//Proceedings of the Conference on Languages, Compiler, and Tools for

Embedded Systems. USA, New York: ACM, 2003: 103-112.

- [7] Abhishek Uduoa, Govindarajan R, Matthew J, et al. Synergistic execution of stream programs on multicores with accelerators[C]// Proceedings of the 2009 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems. USA, New York: ACM, 2009: 99-108.
- [8] Matthew J, Abhishek Uduoa, Govindarajan R, et al. Software pipelined execution of stream programs on GPUs[C]//Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization. USA, Washington, DC: IEEE Compute Society, 2009: 200-209.
- [9] Matin Hashemi, Mohammad H Foroozannejad, Christoph Etzel. FORMLESS: scalable utilization of embedded manycores in streaming applications[C]//Proceedings of the 13th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, Tools and Theory for Embedded Systems. USA, New York: ACM, 2012: 71-78.
- [10] Lam M. Software pipelining: an effective scheduling technique for VLIW machines[C]// Proceeding PLDI'88 Proceedings of the ACM SIGPLAN 1988 conference on Programming language design and implementation. USA, New York: ACM, 1988: 318-328.
- [11] Labonte F. The stream virtual machine[C]//Proceeding PACT '04 Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques. USA, Washington, DC: IEEE Compute Society, 2004: 267-277.

作者简介:

吕鹏伟 男,(1984-),博士研究生. 研究方向为并行计算、编译优化技术. E-mail: lvpwxidian@126.com.

刘从新 男,(1976-),博士,高级工程师. 研究方向为并行计算、编译优化技术.

沈绪榜 男,(1933-),中科院院士. 研究方向为系统芯片设计.