一种面向数据流程序的软件流水并行化方法

魏海涛[1)] 于俊清[1),2)] 余华飞[1)] 秦明康[1)]

[1)](华中科技大学计算机科学与技术学院 武汉 430074)

[2)](华中科技大学网络与计算中心 武汉 430074)

摘 要 数据流编程被广泛应用于多媒体、图像处理和信号处理等领域.多核处理器为数据流程序提供了强大并行计算资源,如何利用多核处理器的并行性以提高数据流程序性能,对满足媒体处理等实时性需求具有重要意义.基于多核处理器提出了一种面向数据流程序的软件流水并行化方法,利用整数线性规划理论对软件流水中的计算、通信资源和流水线执行阶段等进行统一的形式化建模,在最大化流水线计算速率的同时最小化通信开销;同时对存储资源进行了形式化建模,提高存储访问的性能.通过设计数据流编程语言 DFBrook,在 Cell 处理器实现了上述方法.实验结果表明,该软件流水并行方法比其它方法在提高数据流程序性能的同时,降低了通信开销.

关键词 数据流程序;多核处理器;软件流水;并行

中图法分类号 TP302 **DOI 号**: 10.3724/SP.J.1016.2011.00889

A Method on Software Pipelined Parallelism for Data Flow Programs

WEI Hai-Tao[1)] YU Jun-Qing[1),2)] YU Hua-Fei[1)] QIN Ming-Kang[1)]

[1)](*School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan* 430074)

[2)](*Center of Network and Computation, Huazhong University of Science and Technology, Wuhan* 430074)

Abstract Data flow programs have been widely used in multimedia, image process and signal processing domains. Multi-core processor provides plenty of computation resources for applications. It is significant to exploit the parallelism of data flow programs for real-time media application on multi-core processor. This paper proposes a method on software pipelined parallelism for data flow programs. The computation resources, communication resources and storage resources of the software pipelining schedule are modeled as an integer linear programming formulation which achieves the maximum throughput while minimizing the communication overhead. The authors implement the method above for DFBrook data flow programs on Cell processor. The experimental results show that the software pipelined method obtains good performance and reduces the communication overhead.

Keywords data flow programs; multi-core processor; software pipelining; parallelism

1 引 言

多核处理器已经成为主流和工业标准,如 Sony、Toshiba 和 IBM 联合开发的 Cell[1] 多核处理器集成 9 个核;Nvidia 公司开发的 GeForce8800 GPU[2] 集成 16 个流处理器,每个流处理器包含 8 个处理单元;Sun 公司开发的 Niagara[3] 集成 8 个核;Intel 公司和 AMD 公司将推出 16 核的 x86 多核处理器.多核处理器为应用提供了强大的并行计算能力,但同

收稿日期:2010-09-26;最终修改稿收到日期:2011-04-11.本课题得到中国博士后科学基金(20100480899)、中国科学院计算技术研究所国家重点实验室开放基金和 IBM X10 Innovation 基金资助.**魏海涛**,男,1982 年生,博士,主要研究方向为并行处理和编译优化.E-mail:whtaohust@163.com.**于俊清**(通信作者),男,1975 年生,博士,教授,主要研究领域为数字媒体处理与检索、多核处理器编程环境研究.E-mail:yjqing@hust.edu.cn.**余华飞**,男,1984 年生,硕士,主要研究方向为并行与运行时系统.**秦明康**,男,1989 年生,硕士研究生,主要研究方向为并行编译.

时将其复杂的数据划分、存储结构和通信机制等底层设计暴露给了编程人员,从而给编程带来了巨大的挑战.传统的编程模型,如 C、C++和 FORTRAN 已经无法很好地适应多核处理器的架构.数据流编程(Data Flow Programming)模型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,为编译器在多核处理器下的优化提供了大量的并行性,目前受到广泛的关注.

数据流编程语言如 StreamIt[4]、CUDA[2]和 SPUR[5]等一般都是基于同步数据流模型(Synchronous Data Flow, SDF)[6].在该模型中,每个 actor是一个运算过程,代表一系列的指令,数据从输入队列进入,经过 actor 的处理后到输出队列.actor的每次运行由输入队列上的数据到达速率来决定,只要输入数据队列中有数据,且输出队列有空闲的存储,actor 就可以运行.一般来说,一个数据流程序由多个 actor 和连接 actor 的数据队列组成.actor一般被表示为一个函数,数据队列采用先进先出方式组织.每个 actor 有相应的触发规则,当规则满足时,该 actor 被触发,读取输入队列上的数据,产生输出数据.同步数据流模型的这种机制为编译器提供了并行优化的机会.

软件流水是一种开发数据流程序并行性的有效方法.通过将数据流程序看作一个循环,不同迭代中的 actor 能够在流水线中实现重叠执行.然而,获得的性能常会被处理器核间的通信和同步开销所抵消.同时,处理器核、通信带宽和片上存储等系统资源限制将会带来流水线的停滞.软件流水的性能由每次启动迭代的时间来衡量,一个具有最小启动时间的调度称为最优计算速率的软件流水调度.因此,如何设计软件流水调度方法,在满足系统资源受限的情况下,获得最优计算速率的同时最小化通信开销是本文研究的主要问题.

本文基于多核处理器提出了一种面向数据流程序的软件流水并行化方法,利用整数线性规划理论对软件流水中的计算、通信资源、存储资源以及流水线执行的阶段等进行统一的形式化建模,在满足最大化流水线计算速率的同时最小化通信开销.通过设计数据流编程语言 DFBrook,在 Cell 处理器上现了上述方法.实验结果表明,本文提出的方法比现有的其他方法在较大地提高性能的同时减小了开销.

本文第 2 节介绍相关性研究工作;第 3 节介绍 DFBrook 数据流编程语言;第 4 节详细讨论数据流程序的软件流水调度方法;第 5 节给出相应的实验结果;第 6 节对论文进行总结.

2 相关工作

软件流水最初在超标量和超长指令字计算机中被提出来,用于开发指令级的并行[7-8].作为一种开发并行性的方法,软件流水开始逐渐被用于开发数据流程序的并行性.Kudlur 等人采用了的软件流水线调度方法实现了 StreamIt 在多核处理器下的并行优化[9].该方法首先对计算节点进行划分,然后采用阶段赋值算法来实现流水线阶段调度,但是该方法只对计算资源的调度进行了研究,没有考虑到通信和存储资源.Choi 等人在 Kudlur 的基础上提出了在嵌入式系统下的存储受限软件流水调度[10],但是该方法对解的集合采用保守估计,即假设了所有相连的计算节点都被调度到不同的处理器,因此过多地计算量存储的开销,同时,该方法也没有考虑通信开销.Govindarajan 等人采用线性规划研究了数据流程序在 GPU 上的调度框架[11],但是该模型只是针对共享存储结构的处理器,针对分布式存储结构无法使用.

针对上述问题,本文基于多核处理器提出了一种面向数据流程序的软件流水并行化方法,利用整数线性规划理论对软件流水中的计算、通信资源、存储资源以及流水线执行的阶段等进行统一的形式化建模,在取得最优化流水线计算速率的同时最小化处理器间的通信开销.同时对存储受限的多核处理器进行了形式化建模,以提高存储访问的性能.

3 DFBrook 数据流语言与编译

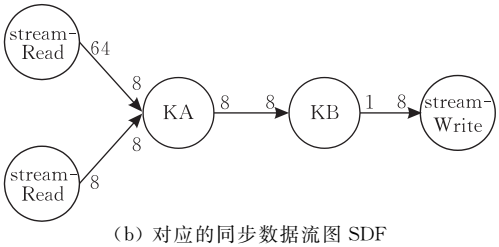
DFBrook 在语义上对 Brook[12]语言进行了数据流扩展.DFBrook 程序由普通的 C 代码和数据流代码组成新的数据类型,表达了数据流模型中的数据队列的语义.数据流类型只能限制在核函数中使用.流操作符则是一组对流类型数据进行特定运算操作的集合,用于与其它数据类型之间的转换.

图 1 给出了一个 DFBrook 的例子程序和相应的 SDF 图.核函数 KA 具有 2 个输入流和 1 个输出流,每次执行消耗从输入流 vec_a 和 vec_b 中各消耗 8 个数据单位,向输出流 vec_result 中生成 8 个数据单位,其中 vec_a 和 vec_b 是只读的,vec_result 是只写的.在 main 函数中,streamfor 语句块对应 DFBrook 程序的数据流代码,流类型数据由引用参

数从一个 kernel 函数传递到另一个 kernel 函数. streamRead 和 streamWrite 操作符是两个特殊的流操作符，streamRead 用于将外部 C 代码的数据“读入”流，指明一个流的起始点，因此 streamRead 只有第 3 个参数用于指定输出码率；streamWrite 将流“写回”外部 C 代码，指明流的终止，因此 streamWrite 只有输入数据流. 在例子中，streamRead 将数据从外部数组 data_matrix 和 data_vector 读入数据流中，该数据流作为 kernel 函数 KA 的输入进行处理后结果输出到 tempmv 流，tempmv 流被 kernel 函数 KB 消耗产生 result 流，最后 result 流被写回到数组 data_result 中. 这些流在 kernel 函数间以队列的方式进行传输，队列的长度由编译器指定.

```
kernel void KA(stream float vec_a⟨8⟩, stream float
vec_b⟨8⟩, out stream float vec_result⟨8⟩){…}
kernel void KB(stream float vec⟨8⟩, out stream float
result⟨1⟩){…}
main(){…
float data_matrix[64];
float data_vector[8];
stream float matrix⟨⟩, vector⟨⟩;
stream float tempmv⟨⟩, result⟨⟩;
streamfor(…){
streamRead(matrix, data_matrix,64);
streamRead(vector,data_vector,8);
KA(matrix,vector,tempmv);
KB(tempmv,result);
streamWrite(result, data_result,8);}
}
```

(a) 数据流编程语言 DFBrook 的一个例子



(b) 对应的同步数据流图 SDF

图 1　数据流编程语言 DFBrook 及其对应的 SDF

编译器首先对 DFBrook 程序进行语法和语义检查，然后采用数据流分析理论，建立各个核函数和流操作符之间的依赖关系，构造中间表示——SDF；同步数据流图的每个节点代表一个核函数或者流操作符，图的每条有向边代表核函数之间的数据流通路，边上的两个权值代表起点核函数的输出流速率和终点的输入流速率，即生产和消耗数据的速率；流水线调度模块对同步数据流图进行分析，将图中的节点调度到目标处理器上的处理核上，同时为调度到不同处理器核上的节点间生产 DMA(Directed Memory Access)，实现软件流水调度；最后生成多线程目标代码. 本文采用的目标结构为基于加速器的多核处理器结构如 Cell，每个处理器核只能访问自己私有的片上局部存储，通过异步 DMA 来实现数据的通信.

4　数据流程序的软件流水并行

4.1　多核处理器下的软件流水调度

在基于多核处理器的软件流水中，一次循环迭代作为一个任务，被分成若干阶段，各个阶段在不同的处理器上执行，当一个处理器完成了它负责的阶段后，结果就作为输入传送到流水线中的下一个处理器. 图 2 给出了一个基于多核处理器的软件流水调度的例子，图左边为一个数据依赖图，K1 执行后利用 DMA，将数据传给 K2 执行，图右边为对应的流水线的调度表. 计算任务被分为 3 个阶段，第 0 位置的处理器核完成处理后，结果用 DMA 传输到第 1 位置的处理器核，同时第 0 位置的处理器核开始处理第 2 个任务，即下一次循环迭代. 按照这种方式，流水线被逐渐填满，处理器逐渐进入忙状态. 在流水线满的状态下，不同的循环迭代在流水线中并行执行.

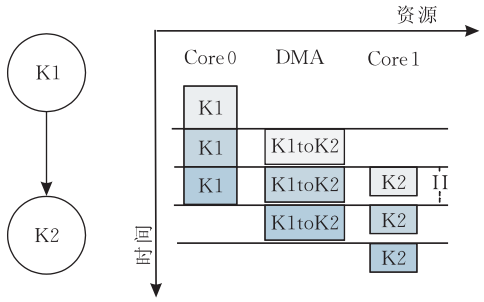


图 2　基于多核处理器的软件流水调度例子

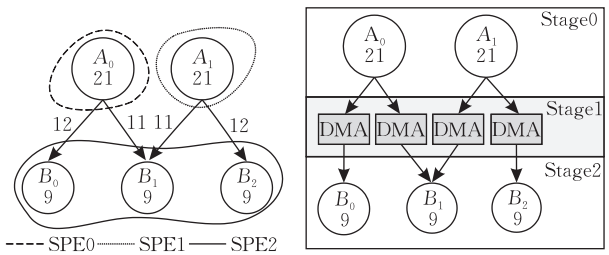
在软件流水线中，相邻两次循环迭代(任务)开始进入流水线的时间间隔称为启动间隔(Initiation Interval，II)，启动间隔的长短代表了流水线的速度，启动间隔越小，流水线的执行速度越快. 流水线进入充满状态后，单位时间内流水线完成循环迭代(任务)的次数称为吞吐率(Throughput，T)，吞吐率代表了流水线的响应速率，吞吐率越大，流水线的响应时间越短，吞吐率 T 和启动间隔 II 满足以下公式：$II=1/T$.

4.2　通信最小化的最优化软件流水调度

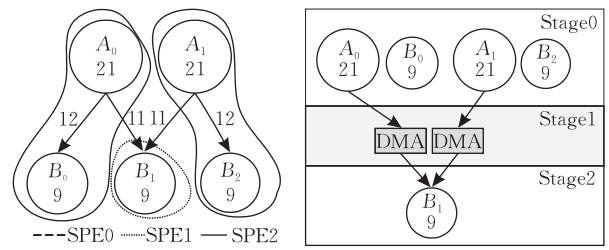
本节设计的软件流水调度的目的是根据任务的依赖关系和当前系统中资源的情况，构造一个软件流水调度，在最小化启动间隔 II 的基础上，最小化通信开销.

图 3 给出了两种调度方法，方法 1 不考虑通信

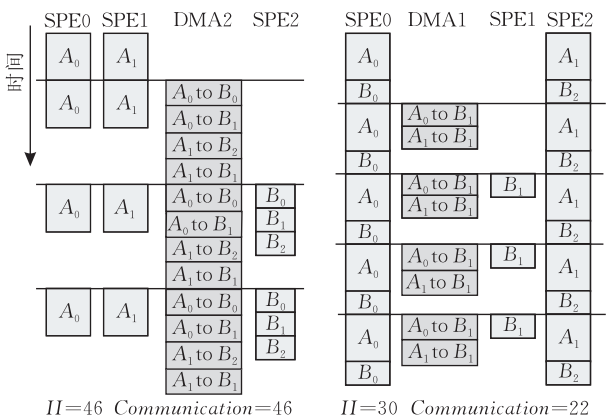
开销，首先将节点 A_0 和 A_1 分别调度到处理器 SPE0 和 SPE1，节点 B_0、B_1 和 B_2 被调度到处理器 SPE2；然后检查每一条边，如果边的两个节点被调度到不同处理器，那么将该边调度到对应的 DMA；最后对每个实例和边赋值阶段(Stage)，节点 A_0 和 A_1 被赋值阶段 Stage0，节点 B_0、B_1 和 B_2 被赋值阶段 Stage2，4 个 DMA 被赋值阶段 Stage1. 低通信开销软件流水调度方法将处理器调度、DMA 和阶段赋值作为一个整体来考虑. 如图 3(b)所示，A_0 和 B_0 被调度到处理器 SPE0，B_1 被调度到处理器 SPE1，A_1 和 B_2 被调度到处理器 SPE2. 由于调度到相同的处理器上的实例共享存储，无需通信开销，因此，A_0 和 B_0 之间以及 A_1 和 B_2 之间的通信开销被消除. 图 3(c)和(d)比较了以上两种软件流水调度方法的执行结果. 方法 1 得到的启动间隔 II 为 46 个单位，总通信开销为 46 个单位；低通信开销软件流水调度方法得到的启动间隔 II 为 30 个单位，总通信开销为 22 个单位，因此，该方法在取得高计算速度的同时降低了通信的开销.



(a) 不考虑通信的软件流水调度



(b) 低通信开销软件流水调度



(c) 图(a)中流水线执行过程　(d) 图(b)中流水线执行过程

图 3　两种软件流水调度方法

以上例子阐述的软件流水调度问题可以被形式化为一个整数线性规划问题. 考虑一个数据流程序的数据流图中间表达 $G=(V,E)$，V 为节点集，E 为边集. 对于每条边，为了满足产生的数据个数与消耗的数据个数相等，起点和终点所需要执行的最小次数，称为该节点的重复运行次数，所有节点的重复执行次数构成了重复执行向量 $\boldsymbol{r}_G$. 通过将 G 中的节点进行 $\boldsymbol{r}_G$ 次展开，可以得到数据流图对应的数据依赖图 $G_d=(V_d,E_d)$，具体方法可参见文献[13]. 以下将对数据依赖图 G_d 进行形式化建模，G_d 中的每个节点由 $v\in V_d$ 表示，每条边由 $(u,v)\in E_d$ 表示，其中 $u\in V_d$. 令 P 为数据处理器的集合，$P=\{0,1,\cdots,P_{\max}-1\}$，$P_{\max}$ 为数据处理器的个数.

为每个节点 v 定义 0-1 变量 $a_{v,p}$，表示节点 v 是否被调度到处理器 p 上. 处理器资源的限制被形式化为等式(1)，该限制条件确保了每个节点只能被调度到一个处理器上.

$$\sum_{p=0}^{P_{\max}-1} a_{v,p}=1,\ \forall v\in V_d \tag{1}$$

令 $work(v)$ 表示函数节点 v 的执行时间，II 表示软件流水的启动间隔，不等式(2)形式化给出了以下限制条件：调度到一个处理器上的总工作负载必须在给定的启动间隔 II 内完成.

$$\sum_{v\in V_d} a_{v,p}\times work(v)\le II,\ 0\le p<P_{\max} \tag{2}$$

以上两个限制条件给出了处理器的调度限制，没有考虑数据的依赖限制，即没有考虑图中边的限制条件. 如果两个相连接的节点被调度到不同的处理器上，将需要 DMA 传输数据的操作. DMA 传输是双向的，既可以由源处理器发起，也可以由目的处理器发起. 考虑大多系统处理器的性能设计，采用由目的处理器发起 DMA. 对于边 $(u,v)\in E_d$，定义 0-1 变量 $d_{u,v,p}$ 来表示边 $(u,v)\in E_d$ 是否被调度到 DMA p 上. 变量 $d_{u,v,p}$ 为 1 当且仅当 v 被调度到处理器 DP p，同时 u 被调度到其他不同的处理器上. 不等式组(3)确保了当两个相互连接的节点被调度到同一个处理器上，将不需要 DMA 进行数据传输.

$$\begin{cases} d_{u,v,p}\ge a_{v,p}-a_{u,p} \\ d_{u,v,p}\le a_{v,p} \\ d_{u,v,p}\le 1-a_{u,p} \end{cases},$$
$$\forall(u,v)\in E_d,\ 0\le p<P_{\max} \tag{3}$$

此处约定一对相连接的节点之间的 DMA 传

输总是由目的节点所调度到的处理器发起. 令 $Commun(u,v)$ 表示两个节点 u 和 v 之间的数据传输负载，流水线的计算速率同样也受所有的 DMA 的数据传输负载的限制. 限制不等式(4)确保了在给定一个 DMA 上的数据传输总负载不大于给定的启动间隔 II.

$$\sum_{(u,v)\in E_d} d_{u,v,p} \times Commun(u,v) \leq II,$$
$$0 \leq p < P_{\max} \tag{4}$$

不等式(1)～(4)只对节点在各个处理器上的调度以及相应的数据传输在 DMA 的调度给出了限制条件，即只在空间维上对软件流水调度进行了形式化限制. 为了在时间维上调度节点和边，引入 Stage 概念来描述节点和边在时间上的调度. 定义整数变量 sv_v 表示节点 v 被赋值的阶段号；定义整数变量 $se_{u,v}$ 表示边 $(u,v)\in E_d$ 被赋值的阶段号. 不等式(5)和(6)给出了软件流水调度在时间上的限制条件.

$$se_{u,v} \geq sv_u + \sum_{p=0}^{P_{\max}-1} d_{u,v,p},$$
$$\forall (u,v)\in E_d,\ se_{u,v} \geq 0,\ sv_u \geq 0 \tag{5}$$

$$sv_v \geq se_{u,v} + \sum_{p=0}^{P_{\max}-1} d_{u,v,p},$$
$$\forall (u,v)\in E_d,\ se_{u,v} \geq 0,\ sv_u \geq 0 \tag{6}$$

对于给定的边 (u,v)，目的节点的阶段号应该在源节点后面，因为在时间上，目的节点必须在源节点运行后运行，即 $sv_v \geq sv_u$ 表示了源节点和目的节点之间的数据依赖关系. 如果 u 和 v 被调度到不同处理器上，两者需要 DMA 操作进行数据传输，那么从 u 到 v 的数据传输必须分配一个单独的阶段号 $se_{u,v}$，节点 u、v 和 DMA 必须满足不等式 $sv_u < se_{u,v} < sv_v$. 由于每个节点只能调度到一个处理器，即 $a_{v,p}$ 只能对于某一个处理器 p 取到 1，从不等式(3)可以得知：对于给定的一条边，要么只被赋值给一个 DMA，要么不被赋值给任何 DMA. 当 u 和 v 被调度到不同处理器上，求和式 $\sum_{p=0}^{P_{\max}-1} d_{u,v,p}$ 的值为 1；当 u 和 v 被调度到相同处理器上，值为 0. 当源节点 u 和目的节点 v 被调度到不同处理器上时，不等式(5)确保了 DMA 传输的阶段号至少在源节点 u 后面一个阶段；类似的，不等式(6)确保了目的节点 v 的阶段号至少在 DMA 后面一个阶段. 当节点 u 和节点 v 被调度到相同处理器上时，不等式 $sv_v \geq sv_u$ 保持，此时 $d_{u,v,p}$ 无意义.

$$\min \sum_{p=0}^{P_{\max}-1} \sum_{(u_i,v_j)\in E_d} d_{u,v,p} \times Commun(u,v) \tag{7}$$

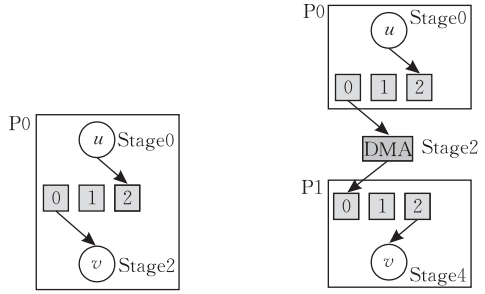
函数(7)描述了所有节点之间的总通信开销，规划的目标函数是最小化通信开销. 式(1)～式(7)为通信最小化的最优化流水调度(Communication Minimized Rate-Optimal scheduling，CMRO)问题提供了精确的整数线性规划形式化描述. 该模型精确描述了软件流水调度中的计算资源和通信资源，在保证高计算速率的同时，最小化通信开销，提高软件流水的性能.

因为最小化通信是在最优化流水的前提下，因此首先必须求得满足限制条件(1)～(6)的最小化 $II_{\min}$，将式(7)用 min II 代替，可以得到最优化流水调度 RO(Rate Optimal). 通过求解 RO 问题，可以得到 $II_{\min}$，以此作为 CMRO 问题的输入，从而求出最小化通信开销.

4.3　内存限制的流水调度

4.3.1　存储分配机制

这里采用文献[10]的方法来阐述节点对 (u,v) 的两种缓存分配机制. 如图 4(a)所示，u，v 两个节点被调度到同一处理器 P0 上，被赋值的阶段分别为 0 和 1，为了保证流水线的重叠执行，此时分配的缓存个数为 3；在图 4(b)中，u，v 两个节调度不同处理器上，被赋值的阶段分别为 0 和 4，此时，u 所在的处理器 P0 和 v 所在的处理器 P1 分别分配 3 个缓存. 这是因为调度到不同处理器上的节点对需要 DMA 进行数据传输，在软件流水的执行中，u，v 的执行以及两者间 DMA 的传输在时间上是重叠的，因此都需要缓存进行保存中间结果.



(a) 源和目的节点调度到同一处理器上　　(b) 源和目的节点都调度到不同处理器上

图 4　软件流水调度中两种缓存分配机制[10]

4.3.2　形式化建模

定义生产者-消费者对为普通的相连的计算节点对或者 DMA-计算节点对；定义缓存组为在软件流水调度中，生产者-消费者对所分配的缓存队列. 针对以上两种缓存分配机制有：(1) 当两个相连计算节点对 (u,v) 被调度到同一处理器上，缓存组被共享，节点 u 的输出数据存入缓存组作为节点 v 的输入数据，缓存组的大小计算如下：$sv_v - sv_u + 1$；

(2)当两个相连计算节点对(u,v)被调度到不同处理器P0和P1上,分配在处理器P0的存储上u的输出数据,需要DMA传输到处理器DP1的存储上,作为v的输入数据.在DP0上用于缓存u输出数据的缓存组大小由$se_{u,v}-sv_u+1$计算;在P1上用于缓存v输入数据的缓存组的大小由$sv_v-se_{u,v}+1$计算.

不等式(8)给出了处理器p的内存限制的形式化描述,不等式左边是处理器p上的存储消耗总和.其中,$Buffer(u,v)$为程序在一次迭代执行中,边(u,v)所需要的缓存大小,可以由边上的数据传输个数和相应的数据类型大小的乘积得到.其中,$M_1=se_{u,v}-sv_u+1$,$M_2=sv_v-se_{u,v}+1$,$M_3=sv_v-sv_u+1$,表示各种缓存组大小所关联的阶段差.

$$\sum_{(u,v)\in E_d}[M_1a_{u,p}+(M_3-M_1)a_{v,p}+(M_1+M_2-M_3)d_{u,v,p}]\times Buffer(u,v)\leq Mem_p,\quad 1\leq p<P_{\max} \tag{8}$$

通过把式(8)中的M_1、M_2和M_3用数值常量替换,限制条件则从非线性转换为线性.从3.2节可知当相连两个节点u和v被调度到不同的处理器上,那么DMA和u的阶段差至少为1,同样,v和DMA的阶段差至少也为1,即$se_{u,v}-sv_u$和$sv_v-se_{u,v}$的最小值都为1;如果节点u和v被调度到相同的处理器上,那么两者至少是在同一个阶段,即sv_v-sv_u的最小值为0.因此,得到估计值:$M_1=2$,$M_2=2$,$M_3=1$,限制条件(8)可以被简化为(9).

$$\sum_{(u,v)\in E_d}[2a_{u,p}-a_{v,p}+3d_{u,v,p}]\times Buffer(u,v)\leq Mem_p,\quad 1\leq p<P_{\max} \tag{9}$$

模型的目标函数为最优化流水线吞吐率,即最小化启动时间,如式子(10)所示.

$$\min\ II \tag{10}$$

限制条件(1)~(7)、(9)和目标函数(10)给出了数据流程序在内存限制多核处理器下的软件流水调度模型的整数规划问题形式化表达.因此可以采用经典的分支界定法或割平面法进行求解.虽然整数线性规划求解是NP的,但是当前已经有许多高效的产品级的求解器,如CPLEX[14]混合整数规划求解器(Mixed Integer Programming Solver,MIP Solver)可以较快地求解整数线性规划问题,该MIP采用分支界定算法来求解,并且可以通过设定最优解的精度和求解的时间限制来加速求解的过程.

由于限制条件(9)是对内存消耗的一个最小估计,因此,真实的内存消耗可能会大于限制条件.我们通过对DMA的阶段号进行动态调整,从而平衡各个存储之间的内存使用量,以消除内存溢出.首先,计算每个处理器的真实存储消耗,针对存储是否溢出将处理器分为溢出和非溢出两类;其次对调度到溢出处理器上的节点的DMA进行查询,如果有DMA可以调整,使得该处理器的存储消耗减小,则调整DMA阶段,直到消除所有存储溢出或者没有可以调整的DMA.

5 实验结果与分析

在第3节中描述的DFBrook数据流编译系统中,实现了上述的软件流水调度方法.本节通过实验对该软件流水调度方法进行性能评价,同时通过与其它方法的比较来验证方法的有效性.

5.1 测试平台和实验方法

编译前端对DFBrook语言进行语法和语义分析后,采用数据流分析,生成数据流图,调度程序以数据流图为输入,按照上述的模型进行建模,求解出调度结果,用于进一步的优化和代码生成.实验的硬件测试平台为PlayStation3,配有一个Cell处理器(6个SPE可用)和256MB的主存.Cell处理器采用主从式的组织结构,集成了一个控制处理器单元(PowerPC Processor Element,PPE)和8个数据处理器单元(Synergistic Processor Element,SPE),每个SPE都配有一个256K的局部存储器LS和DMA数据传输引擎,PPE和SPE通过片上环形网络互连,共享片外存储,PPE和SPE可以通过消息同步.实验采用IBM的Cell SDK3.0作为DFBrook编译器软件支持,采用的本地编译器为ppe-gcc和spe-gcc.整数线性规划采用的是CPLEX的混合整数规划求解器MIP Solver.DFBrook程序中存在3种通信模式:streamRead到kernel函数、kernel函数之间以及kernel函数到streamWrite,为了测试模型中的通信量参数,使用1个PPE和2个SPE测量每个kernel函数、数据流操作符以及每个DMA传输量.

实验采用表1中的测试程序集来评价调度模型的性能.表中对各个测试程序的特征:如kernel节点个数、读写节点个数以及通信的边条数进行了详细描述.大多数的测试程序来自于多媒体处理领域,如:Gausslap测试程序实现了高斯-拉普拉斯算子进

行边缘检测算法；histogram 实现了并行化的直方图算法；shortEnergy 实现了声音短时能量特征算法的 DFBrook 程序；averageMotion 实现了视频处理中的运动向量提取算法.

表 1　实验的测试程序集实验结果与分析

测试程序	kernel 节点数	读写节数	通信边数	程序描述
DCT	35	2	48	8×8 DCT 变换
FFT	12	2	14	8×8 FFT 变换
Gausslap	28	6	54	高斯拉普拉斯算子
histogram	11	2	15	图像直方图
imagesmooth	28	6	54	图像平滑
MatrixMult	24	3	37	分块矩阵乘
mergesort	17	2	24	归并排序
shortEnergy	20	2	26	音频短时能量
averageMotion	28	3	43	视频平均运动向量

5.2　可扩展性

实验首先通过加速比来评价模型的可扩展性，加速比采用以下方法计算：当前程序 $Program_i$ 在 p 个 SPE 下取得的加速比由当前程序在 1 个 SPE 调度获得的启动间隔 II 与当前 p 个 SPE 调度获得的启动间隔 II 相比得到.

图 5 给出了表 1 各个测试程序在不同个数 SPE 下的加速比. 结果显示，调度方法对绝大多数程序取得了近似线性加速比. FFT 和 histogram 在 SPE 个数大于 4 和 5 时，程序的加速比不随处理器增加而提高，这是由于这两个程序只有有限个数的 kernel 函数，即问题的规模较小，无法带来更多的加速比提升. 如表 1 所示，FFT 只有 10 个 kernel 节点（表中 kernel 节点数＋读写节点数＝总节点数），histogram 只有 9 个 kernel 节点. 通过循环展开数据流图来增加并行性（问题的规模）可以进一步地提高加速比. 与上述两个程序相比，DCT 和 MatrixMult 程序分别有 33 个和 21 个 kernel 节点，因此，取得了近似线性的加速比. 同理，Gausslap、imagesmooth 和 averageMotion 也取得近似线性加速比.

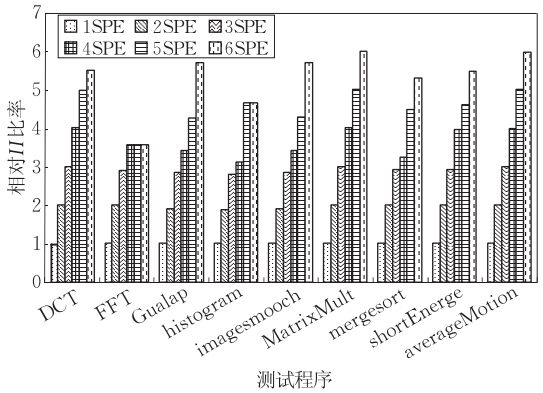


图 5　表 1 中测试程序的加速比（以 II 的比率计算）

5.3　性能比较与分析

本节将本文的通信最小化的最优化流水调度方法 CMRO 与经典的 List 表调度方法[7-8]、周期可行的并行调度算法[6]（Periodic Admissible Parallel Schedule，PAPS）和最优计算速率调度[9] 3 种方法（Rate-Optimal Schedule，RO）进行比较. 表调度算法是广泛应用于传统指令级软件流水调度中的一种方法[7-8]. 实验中，通过修改经典的基于优先级的表调度算法[15]，实现了基于 List 方法的软件流水调度. 具体方法如下：首先采用 List 调度构造节点到处理器上的调度，实现空间上的调度；然后采用 Kudlur 和 Mahlke 提出的阶段赋值算法[9]，对每个节点和数据传输进行阶段赋值. PAPS 算法不考虑实例在不同迭代中的重叠执行，循环的每次迭代为一个基本调度块，一旦调度结果确定以后，后面的迭代采用相同的调度方法. RO 的目标函数是最优化计算速率，即最小化 II. 实验中，通过修改 MCRO 整数线性规划调度模型中的目标函数来实现 RO 调度.

图 6 给出了本文的通信最小化的最优化软件流水调度方法和表调度及 PAPS 调度在启动间隔 II 上的比较结果. 从图中可以看出，低通信开销调度方法比表调度算法有较大的性能提高. 对于具有大计算量的测试程序，如 DCT、shortEnergy 和 averageMotion，MCRO 调度方法比表调度在软件流水线的计算速率上取得了 17.8%～24.7%的提高. MCRO 调度方法比 PAPS 调度算法在软件流水计算速率上，平均有 47%较大的性能提高. 特别，对于 FFT 和 mergesort 两个程序，MCRO 调度比 PAPS 调度在流水线计算速率上，分别提高了 58.9%和 53.9%. 对于计算速率提高最小的 imagesmooth 程序. MCRO 调度的解是通过逐步增加 II，求解规划问题，直到使得模型具有最优解停止，MCRO 方法得到的 II 是最小的 II. 因此，MCRO 调度和 RO 调度具有相同的计算速率.

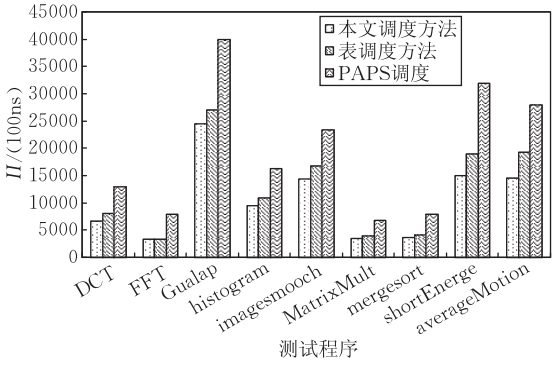


图 6　本文调度方法与其它方法在 II 上的比较

图 7 给出了本文的通信最小化的最优化软件流水调度与表调度、PAPS 调度和 RO 调度在通信开销上的比较. MCRO 调度比 List 调度在通信开销上平均降低了约 23%,而对于具有高通信量的程序如 FFT 和 mergesort,MCRO 调度方法比 List 调度在通信开销上降低了 31.8%～40.7%;MCRO 调度对于大部分程序比 PAPS 调度减小了 10%～20%;MCRO 调度方法比 RO 调度算法在通信开销上,有 7%～40.9%的较大降低,平均减小了 21.7%的通信开销.

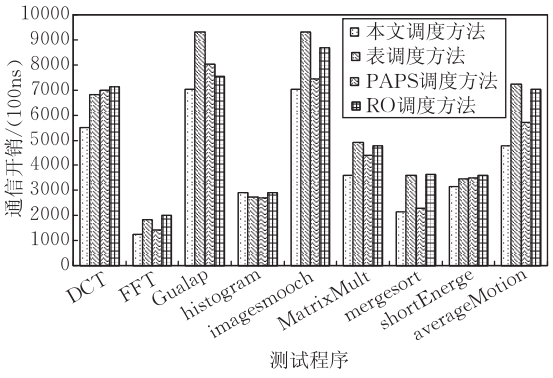


图 7 本文调度与其它方法在通信开销上的比较

5.4 存储性能分析与比较

以下将分析存储限制对流水线性能的影响. 存储限制软件流水调度的目的是最大化地利用片上局部存储来减小程序对主存的访问延迟. 对图 8 给出了本文 4.3 节中存储限制的软件流水调度对程序性能的改进. 这里,处理器的个数为固定值,通过逐渐减小片上存储的容量来记录流水线的启动时间. 片上存储从能够容纳全部程序数据的 *MaxMem* 开始,逐渐减小,程序的性能也随之下降. 这是因为随着存储减小,调度器将尽可能地把节点调度到相同的处理器上来减小存储开销,从而导致处理器上的工作负载加大. 另一方面当片上存储耗尽,将引入新的 DMA 将数据放到主存,从而带来 DMA 的开销.

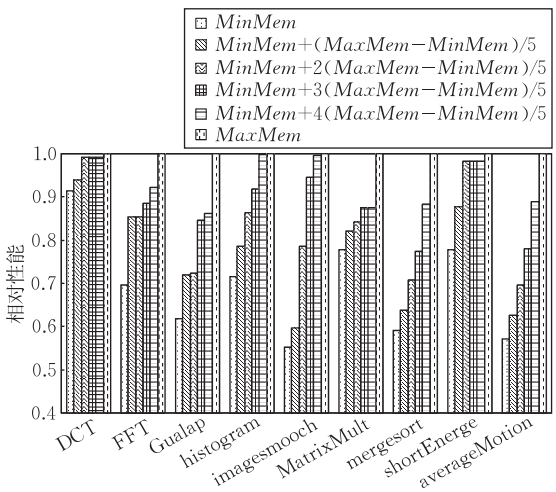


图 8 本文的存储限制调度对程序性能的改进

表 2 本文的存储限制调度与文献[10]在求得解上的比较

测试程序	片上存储容量					
	MinMem	*MinMem*+(*MaxMem*−*MinMem*)/5	*MinMem*+2(*MaxMem*−*MinMem*)/5	*MinMem*+3(*MaxMem*−*MinMem*)/5	*MinMem*+4(*MaxMem*−*MinMem*)/5	*MaxMem*
DCT	√	√	√	√	√	*
FFT	√	√	√	√	√	*
Gausslap	√	√	*	*	*	*
histogram	√	√	√	*	*	*
imagesmooth	√	√	√	√	*	*
MatrixMult	√	√	√	*	*	*
mergesort	√	√	√	√	√	*
shortEnergy	√	√	*	*	*	*
averageMotion	√	√	√	√	*	*

另一方面,将从求得解上来说明本文的存储限制软件流水调度的有效性. 表 2 给出了本文的存储限制调度与文献[10]在每个不同的片上存储大小情况下求得解的情况比较. 在表中,符号"*"表示本文的调度方法与文献[10]的方法都能得到解;符号"√"表示本文的调度方法可以求得解,但文献[10]的方法无法求得解. 从表中可以看出,文本的调度方法可以求得文献[10]无法求得的解. 这是因为文献[10]做了假设:数据流图中相连的节点调度到不同的处理器上,因此过多的计算了存储的开销. 本文的方法避免了这个问题,从而能求得更有效的解.

6 总 结

数据流程序为多核处理器提供了并行优化的机会,但处理器间的通信和同步给程序带来较大的性能开销. 本文基于多核处理器提出了一种面向数据流程序的软件流水并行方法,在实现最大化流水线吞吐率最小化通信开销,同时针对内存受限系统提出了存储限制的流水线调度方法来改进内存的访问

性能.实验结果表明了方法的有效性.本文提出的方法主要针对分布式存储结构的多核处理器架构,如何设计面向共享和层次性存储结构的软件流水并行方法是将来需要进一步研究的工作.

参 考 文 献

[1] Hofstee H P. Power efficient processor design and the Cell processor//Proceedings of the 11th International Symposium on High-Performance Computer Architecture (HPCA05). San Francisco, USA, 2005: 258-262

[2] PDavid Kirk. NVIDIA CUDA software and GPU parallel computing architecture//Proceedings of the 6th International Symposium on Memory Management. Montreal, Canada, 2007: 103-104

[3] Kongetira P, Aingaran K, Olukotun K. Niagara: A 32-way multithreaded SPARC processor. IEEE Micro, 2005, 25(2): 21-29

[4] William Thies, Michal Karczmarek, Saman Amarasinghe. StreamIt: A language for streaming applications//Proceedings of the Compiler Construct. Grenoble, France, 2002: 179-196

[5] Zhang D, Li Z, Song H, Liu L. A programming model for an embedded media processing architecture//Proceedings of the 5th International Symposium on Systems, Architectures, Modeling, and Simulation. Samos, Greece, 2005: 251-261

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

[7] Rau B R. Iterative modulo scheduling: An algorithm for software pipelined loops//Proceedings of the 27th Annual International Symposium on Microarchitecture. San Jose, California, USA, 1994: 63-74

[8] Lam M. Software pipelining: An effective scheduling technique for VLIW machines//Proceedings of the SIGPALN'88 Conference on Programming Language Design and Implementation. Atlanta, Georgia, 1988: 318-328

[9] Kudlur M, Mahlke S. Orchestrating the execution of stream programs on multicore platforms//Proceedings of the 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). Tucson, AZ, USA, 2008: 114-124

[10] Choi Yoonseo, Lin Yuan, Chong Nathan et al. Stream compilation for real-time embedded multicore systems//Proceedings of the 2009 International Symposium on Code Generation and Optimization (CGO). Seattle, Washington, USA, 2009: 210-220

[11] Abhishek Udupa, Govindarajan R, Thazhuthaveetil M J. Software pipelined execution of stream programs on GPUs//Proceedings of the 2009 International Symposium on Code Generation and Optimization. Seattle, Washington, USA, 2009: 200-209

[12] Ian Buck, et al. Brook for GPUs: Stream computing on graphics hardware. ACM Transactions on Graph, 2004, 23(3): 777-786

[13] Govindarajan R, Gao G, Desai P. Minimizing memory requirements in rate-optimal schedules//Proceedings of the International Conference on Application Specific Array Processors (ASAP'94). San Francisco, CA, USA, 1994: 75-86

[14] ILOG. CPLEX 10.1 User's Manual. 2006: 20-61

[15] Oliver Sinnen. Task Scheduling for Parallel Systems. Wiley, 2007: 74-105

WEI Hai-Tao, born in 1982, Ph.D.. His research interests include parallel computing, compiler and multi-core architecture.

YU Jun-Qing, born in 1975, Ph.D., professor. His research interests include digital media processing and retrieval, multi-core programming environment.

YU Hua-Fei, born in 1984, M.S.. His research interests include parallel computing and runtime systems.

QIN Ming-Kang, born in 1989, M.S. candidate. His research interests include parallel computing and compiler.

Background

With multi-core processors have become mainstream and the industry standard, how to simplify the programming in the shield of low-level details of architecture, while making full use of the parallelism between the processor cores to improve application performance, has become a huge programming challenge. Traditional programming model like C, C++ and Fortran are poorly suited to multi-core architectures because of the assumed single instruction stream execution model and centralized memory structure. Domain specific programming like Dataflow Programming Model combines the features of media applications and programming languages to simplify programming and provide the compiler a lot of parallelism optimization for multi-core processor. However, a large number of media processing applications have real-time requirements. And the performance obtained through parallel execution can be overshadowed by the costs of com-

munication and synchronization. To deal with the real-time requirement and the code efficiency, the authors systematically do the research on the key compilation technology for dataflow programs on multi-core processor.

In order to exploit the parallelisms of data flow programs on multi-core processors, a method on software pipelined schedule is proposed for real-time data flow programs. In the software pipelining schedule, the computation resources, communication resources, memory resources and the stage assignment of software pipelining are formulated in a unified model and presented as an integer linear programming problem. The schedule model is implemented to formulate the schedule for DFBrook on Cell architecture. And a comparison with other schedule methods has demonstrated the performance superiority of the proposed method.

The authors appreciate many helpful suggestions and discussions from professor GuangR. Gao at CAPSL in University of Delaware, USA.

This work is financially supported by IBM X10 Innovation Award, China Postdoctoral Science Foundation and Intel grant for a study of multi-core programming environment, In recent years, our group concentrated on the research on parallel computing, compiler and runtime systems for multi-core architecture. Several papers have been published in international conference and journal.