一种基于关键路径分析的CPU-GPU异构系统综合能耗优化方法

林一松 杨学军 唐 滔 王桂彬 徐新海

(国防科学技术大学并行与分布处理国家重点实验室 长沙 410073)

摘 要 GPU强大的计算性能使得CPU-GPU异构体系结构成为高性能计算领域热点研究方向.虽然GPU的性能/功耗比较高,但在构建大规模计算系统时,功耗问题仍然是限制系统运行的关键因素之一.现在已有的针对GPU的功耗优化研究主要关注如何降低GPU本身的功耗,而没有将CPU和GPU作为一个整体进行综合考虑.文中深入分析了CUDA程序在CPU-GPU异构系统上的运行特点,归纳其中的任务依赖关系,给出了使用AOV网表示程序执行过程的方法,并在此基础上分析程序运行的关键路径,找出程序中可以进行能耗优化的部分,并求解相应的频率调节幅度,在保持程序性能不变的前提下最小化程序的整体能量消耗.

关键词 异构系统;GPU;AOV网络;动态电压/频率调节;低功耗优化

中图法分类号 TP302 **DOI号**: 10.3724/SP.J.1016.2012.00123

An Integrated Energy Optimization Approach for CPU-GPU Heterogeneous Systems Based on Critical Path Analysis

LIN Yi-Song YANG Xue-Jun TANG Tao WANG Gui-Bin XU Xin-Hai

(*National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha* 410073)

Abstract GPU's powerful computing ability has attracted much attention in the high performance computing field currently, and the CPU-GPU heterogeneous architecture has become a hot research direction. Although the performance/watt ratio of the GPU is higher than general purpose CPU, the power consumption problem is still one of the critical problems in constructing high performance computing system with CPU and GPU. Most existed power optimization researches oriented to the GPU focus on decreasing the power consumption of the GPU solely. Few works have considered the energy optimization for the whole program involved the CPU and the GPU. In this paper, we analyze the execution characteristics of the CUDA program on the CPU-GPU heterogeneous system deeply, and induce the dependency relationships of tasks in the program in detail. Then, we describe how to express the execution of the program with AOV network, based on which we propose the energy optimization method. Our method analyzes the critical path of the program from the AOV network, detects the tasks that can be optimized for energy and calculates the frequency scaling factors to minimize the whole program's energy consumption while maintaining the performance.

Keywords heterogeneous system; GPU; AOV network; DVFS; low-power optimization

---

收稿日期:2011-01-13;最终修改稿收到日期:2011-05-18.本课题得到国家自然科学基金(60921062,60873016)资助.**林一松**,男,1983年生,硕士,主要研究方向为高性能计算和低功耗优化.E-mail: linyisong@live.cn.**杨学军**,男,1963年生,博士,教授,中国科学院院士,主要研究领域为高性能计算、并行计算机体系结构、高性能编译器和操作系统.**唐 滔**,男,1984年生,硕士,主要研究方向为高性能计算和编译优化.**王桂彬**,男,1981年生,硕士,主要研究方向为高性能计算和低功耗优化.**徐新海**,男,1984年生,硕士,主要研究方向为计算机体系结构和编译优化.

1 引 言

半导体工艺的发展使得芯片上集成的晶体管数目越来越多,目前已达到10亿的量级,处理器的计算能力也越来越强.图形处理器(Graphics Processing Units,GPU)由于最初仅用于加速图形处理,结构比较简单,芯片资源可以被更为有效地用于提升计算能力,因此目前GPU普遍比通用处理器的性能高出一个量级.此外,随着GPU的编程环境不断完善,使用GPU作为CPU的加速器构造异构并行计算系统成为高性能计算领域的热点方向[1].例如最新发布的天河-1A系统就采用CPU-GPU异构系统结构,峰值计算性能超过4PFlops,位列2010年12月发布的Top500超级计算机排行榜之首①.

由于采用了专用的加速部件,异构系统的性能/功耗比一般高于传统的同构系统[2].但不可否认的是,其功耗问题依然很严峻.GPU芯片的计算单元密度高,发热量较大,其绝对功耗超出通用处理器.在构建大规模计算系统时,GPU的功耗将提高计算成本,同时也会降低系统的可靠性,因此研究异构系统的功耗优化问题具有重大的意义.

现有的面向CPU-GPU异构系统的低功耗优化研究多关注针对单个任务的划分问题:对于给定的任务,如何在CPU和GPU之间进行任务划分,使得异构系统在满足性能(能量)约束的条件下,取得能量(性能)最优的问题.这里的任务一般是指没有依赖关系的循环迭代,可以任意地在CPU和GPU之间进行划分.然而,很少有工作针对CPU-GPU异构系统的特点研究整个程序的功耗优化问题.

CPU-GPU异构系统上执行的程序具有其自身的特点.对于给定的一个CPU-GPU程序来说,其中的任务大致可以分为3个部分:CPU计算部分、GPU计算部分和CPU-GPU通信部分.在某些条件下,这3种任务可以并行执行.此外,数据依赖、资源冲突等原因还使得各任务间存在一定的先后依赖关系.因此对异构系统进行能耗优化,建立能耗优化模型的关键在于建立CPU-GPU程序运行过程的抽象表示,该抽象表示需要描述CPU计算、GPU计算和CPU-GPU通信这3个部分以及它们之间的依赖和并行关系.基于该抽象表示才能进一步建立能耗优化的方法.

顶点活动网AOV(Activity On Vertex network)是描述一项工程或系统进行过程的有效工具.它们描述的系统运行过程中通常包含若干活动,每个活动都持续一定的时间,某些活动之间可以并行执行,某些活动之间存在依赖关系.这一点和本文面向的CPU-GPU异构系统上程序的执行过程有相似之处,但也不完全相同.后者所描述的任务依赖情况更加复杂.此外,AOV网是一种求解系统关键路径的有效方法,而程序的关键路径正是进行功耗优化的重要依据.值得说明的是,AOV网和传统的程序优化经常使用的任务图(Task Graph)是一种类似的数据结构.从本质上说,使用结点表示任务的任务图也是一种AOV网,在面向具体的应用领域时可以有针对性地对任务图进行附加的定义和操作.而本文只是抽取这种数据结构最基本的特征,即通过任务的时间和依赖关系描述系统运行过程来描述GPU程序的执行过程,并利用该数据结构最基本的分析方法来分析程序的关键路径,从而为功耗优化建模提供依据.具体地说,本文针对CPU-GPU异构系统程序运行的特点,研究如何使用AOV网描述GPU程序的运行过程,并在此基础上对程序的能耗优化问题进行建模和求解.

本文的创新点主要包括以下几点:(1)面向典型的CPU-GPU异构系统编程模型CUDA[3],分析并归纳了其中包含的4种任务依赖关系.(2)给出了将CUDA程序描述为AOV网的两个算法;(3)给出了基于AOV网的CUDA程序能耗优化方法.

本文第2节简要介绍CUDA程序运行模型并对提出能耗优化的问题;第3节给出基于AOV网络的能耗优化求解方法;第4节给出案例分析;第5节介绍相关工作;最后总结全文.

2 问题提出

本节首先简单介绍CPU-GPU异构体系结构和CUDA编程模型,然后面向能量消耗分析CUDA程序在CPU-GPU异构系统上执行时存在的负载不平衡的问题,进而给出功耗优化的问题描述.

2.1 程序模型

由CPU和GPU构成的异构并行系统结构如图1所示.CPU和GPU一般通过外部总线互连,它们各自也拥有独立的片外DRAM存储器,相互之间可以以DMA的方式进行数据通信.

① http://www.top500.org/lists/2010/11

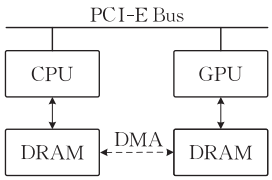


图 1　CPU-GPU 异构并行系统结构

CPU 和 GPU 以 Master-Slaver 的方式协同工作，其中 CPU 是主控端，负责组织数据和调度 GPU 的执行. GPU 上执行的代码被组织成一系列 Kernel 函数，CPU 以函数调用的方式调度 GPU 执行. 图 2 给出了一个典型的 CUDA 程序结构及执行示意图. 左边的框图为 CUDA 程序的伪码，其中 block 1～4 为 GPU 无关的计算代码，我们称为“CPU 任务”. 涉及 GPU 操作的部分包括在 CPU 和 GPU 存储空间之间传输数据以及启动 Kernel 函数的执行，我们将 Kernel 函数的执行称为“GPU 任务”，而数据在 CPU 和 GPU 之间的传输称为“通信任务”.

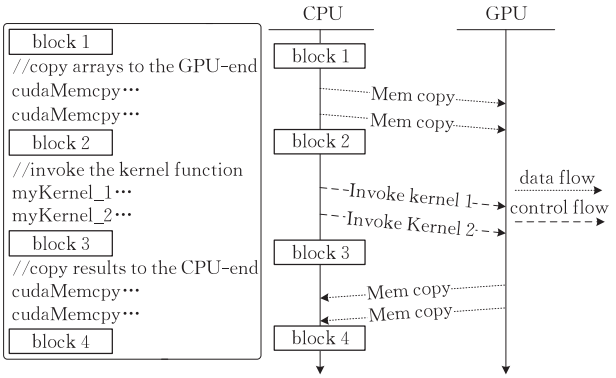


图 2　一个典型的 CUDA 程序及执行示意图

由于 CPU 和 GPU 的执行部件相互独立，并且数据通信可以通过单独的 DMA 硬件完成，因此只要不存在数据依赖，CPU 任务、GPU 任务和通信任务是可以同时执行的. 为此 CUDA 提供了异步的函数调用接口，在调用 Kernel 函数完成 GPU 计算和 CudaMemcpy 完成数据通信时，可以将其指定为异步模式（以“Async”为后缀），使得它们可以和其后的 CPU 代码并行执行. 此外，异步的函数调用之间如果没有资源冲突，即不占用同一种硬件资源，也可以同时执行. 为了保证程序正确执行，CUDA 使用如下同步机制描述异步执行的函数和其后的代码之间的依赖关系：

(1) 为每一个异步的通信任务和 GPU 任务指定一个任务流（task stream），同一个任务流中的任务必须按照到达的顺序依次执行，而不同的任务流之间则没有限制.

(2) 在 CPU 端的程序中调用同步函数为每个任务流指定一个同步点，在同步点处 CPU 必须等待指定的任务流中所有的任务执行完毕才能继续执行.

我们将同步函数称为“同步任务”. 同步任务又可以分为局部同步任务和全局同步任务，前者是指针对某个指定的任务流的同步，而后者是指针对所有任务流的同步. 此外，CUDA 还规定，除了显式地调用全局同步函数外，没有指定为异步模式的通信和 GPU 任务也将阻塞所有的任务流直至它们全部完成，即在这些任务之前存在一个隐含的全局同步点.

2.2　能耗优化问题

处理器的功耗包括静态功耗和动态功耗，其中静态功耗取决于芯片的电气特性，而动态功耗则取决于芯片工作的电压和频率，一般来说后者是低功耗优化的重点. 动态电压/频率调节 DVFS（Dynamic Voltage/Frequency Scaling）[4] 是一种广泛使用的动态功耗优化技术，它通过在一定范围内降低处理器的电压/频率以减少其能量的消耗. 通常，CMOS 电路的核心电压和其工作频率一般要同时调节才能保证电路正常工作，它们之间满足 $f \propto (V-V_t)^{\gamma}/V$，其中 V_t 是阈值电压，γ 为工艺相关的参数. 通常情况下 V_t 远小于 V 且 $\gamma \in [1,2]$. 本文假定 $\gamma=2$，此时频率 f 和电压 V 近似为线性关系. 此时根据 CMOS 电路的功耗公式 $P=\alpha CV^2 f$，功耗 P 可以看成和频率 f 的立方成正比，即 $P \propto f^3$. 此外，我们还近似地认为程序在处理器上的执行时间和处理器的工作频率成反比，即 $T \propto f^{-1}$，因此程序在处理器上运行所消耗的能量满足 $E=PT \propto f^2$. 综上，本文在进行能耗优化时，只考虑对处理器频率的调节. 注意到，这里对 CMOS 电路以及处理器的功耗计算模型进行的假设只是为了简化后续的描述，并不影响本文所提出的使用 AOV 网络对程序进行分析并构建功耗优化模型的方法.

根据 2.1 节的描述，CPU 任务、GPU 任务和通信任务三者间的异步执行意味着 CPU 和 GPU 间可能存在负载不平衡的情况，即在某个同步点处 CPU(GPU) 由于提前完成任务而等待 GPU(CPU) 任务或通信任务的完成. 从能量消耗的角度看，这种情况不是能量最优的：完成较早的任务可以以较低的频率运行，使得它和完成较晚的任务同时到达同步点，此时在不延长整体执行时间的情况下，可以降低程序整体的能量消耗.

一般来说,CUDA 程序在 CPU-GPU 异构系统上运行带来的总能量消耗为

$$E=E_{\text{active}}+E_{\text{static}}=\sum E_{\text{C}}+\sum E_{\text{G}}+\sum E_{\text{T}}+P_{\text{static}}T,$$

其中 E_{C},E_{G}和 E_{T}分别代表 CPU 任务,GPU 任务和通信任务的能量消耗.在本文的模型中,可以通过 DVFS 技术调节的是 CPU 和 GPU 的工作频率,因此通信部分的动态能量消耗不变;此外,能耗优化的前提是保持程序的总运行时间不变,因此由静态功耗带来的能量消耗也可以视为常数.此时,程序的能量最优问题等价于在程序执行总时间不变的前提下最小化 CPU 任务和 GPU 任务的总能量消耗.需要注意的是,这里所指的程序执行总时间是指程序在 CPU 和 GPU 分别处于最高工作频率下的执行时间.

使用 DVFS 技术调节 CPU 或 GPU 处理器的频率时需要考虑两方面的问题:寻找调节时机和确定调节幅度.程序运行过程中,某些任务的执行时间直接影响整个程序的执行时间,我们称其为"关键任务".为保持程序运行时间不变,我们不能对关键任务进行频率调节.因此寻找调节时机是指在程序中寻找不影响程序总体执行时间的前提下可以进行频率调节的 CPU 任务和 GPU 任务,我们统称为"非关键任务";而确定调节幅度则根据非关键任务的执行时间以及不影响程序总时间时该任务最大可能的执行时间确定处理器频率调节的幅度.因此,对于给定的 CUDA 程序,其面向 CPU-GPU 异构系统的能耗优化问题可以描述为:在程序中寻找一组非关键任务并确定相应的 CPU 或 GPU 频率调节系数,使得程序在 CPU-GPU 异构系统上运行的总时间不变而能量最优.

本文求解这一问题的思路为

(1)首先从程序中分离出不同类型的任务,然后将运行的过程描述为任务之间的依赖关系,构造表示程序运行过程的 AOV 网络.

(2)其次对 AOV 网络进行分析,确定关键路径,则非关键路径上的 CPU 和 GPU 任务为可以进行频率调节以节省能耗的非关键任务.

(3)最后根据关键路径上任务执行的时间确定非关键任务可以放松的执行时间,从而求解每个任务的处理器频率调节幅度以最小化能量的消耗.

3 问题求解

3.1 CUDA 程序的 AOV 网表示

由于 CUDA 程序运行过程中存在复杂的并行和同步机制,因此寻找程序中所有的非关键任务的难点在于通过某种抽象表示将程序中任务间的并行和依赖关系描述出来.本文采用 AOV 网描述 CUDA 程序中任务之间的并行和依赖关系.CUDA 程序中任务之间的并行与依赖关系比较复杂,其中并行关系主要包括 CPU 任务、GPU 任务和通信任务之间的并行,而依赖关系则可以归纳为以下 4 种:(1)程序依赖(program dependency),是指由程序序决定的依赖;(2)流依赖(stream dependency),是指同一个任务流内部的任务之间的依赖;(3)同步依赖(synch dependency),是指在同步点处,除 0 号任务流外其它任务流中的任务与同步任务之间的依赖①;(4)资源依赖(resource dependency),是指使用同一种资源,如通信部件或 GPU 的多个任务之间存在的依赖关系.

注意到,程序依赖、流依赖和同步依赖都描述了一种先后依赖关系,即任务间的先后关系,而资源依赖则描述了任务间的互斥关系.传统的 AOV 网可以表示先后的依赖关系,而无法表示互斥关系,因此我们需要将互斥关系转化为先后依赖关系才能完整地描述 CUDA 程序的行为.下面我们首先根据 CUDA 程序构造初步的 AOV 网,我们称为任务依赖图,其中包含前 3 种依赖关系,然后再对任务依赖图进行处理,加入资源依赖关系的描述.

定义 1. 任务依赖图(Task Dependency Graph).程序 P 的任务依赖图 $G=(V,E)$是一个有向无圈图,其中 V 为结点集合,表示程序中的任务,E 为连接结点的有向边,表示任务之间的先后依赖关系.

每个结点可以用一个二元组表示 v:$\langle type, time\rangle$,其中 $type$ 表示结点的类型,包括 CPU 任务、GPU 任务、通信任务和同步任务,分别记为 C、G、T 和 S;而 $time$ 表示该任务需要执行的时间.e:$\langle v_1, v_2\rangle \in E$ 表示 v_2必须在 v_1完成之后才能开始.由于同步任务只为描述异步的 GPU、通信任务和 CPU 任务之间的同步关系,它本身的执行时间为 0;通信任务的时间可以根据带宽和通信量计算得出;而 CPU 任务和 GPU 任务的时间可以参照已有的性能模型[5-6]进行估算.关于任务执行时间的静态分析不是本文研究的重点,因此这里不展开讨论.

对于图 2 中所示的程序段,假定所有的通信任

① 同步任务由 CPU 执行,默认在 0 号任务流中,因此 0 号任务流中的任务与同步任务间的依赖属于程序依赖.

务和 GPU 任务都指定为异步的，并且每组和 GPU 相关的两个任务分别放入两条任务流中. 此外，还假定程序段结束处调用了全局的同步函数. 其任务依赖图如图 3 所示.

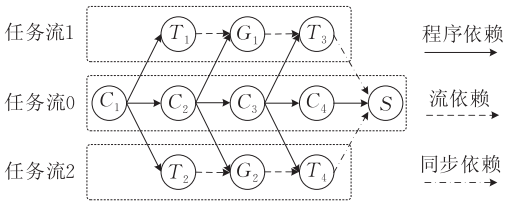


图 3　图 2 中示例程序段的任务依赖图

如图 3 所示，程序中包含 3 条任务流，根据 CUDA 程序的定义，所有的 CPU 任务、同步任务以及同步模式的通信、GPU 任务都被划入 0 号任务流.

下面讨论任务依赖图的构造方法. 假定给定 CUDA 程序的指令序列中和 GPU 相关(通信函数、Kernel 函数、同步函数)的指令有 P 个，并且这些指令将 CPU 指令序列划分为 Q 段连续的子序列，我们将每个连续的 CPU 指令子序列和每个 GPU 相关的指令都对应为一个任务，因此整个程序总共包含 $P+Q$ 个任务. 记 $\mathcal{M}=(M_1,M_2,\cdots,M_{P+Q})$ 为按照程序序排列的任务序列，对于 $1\leq i\leq P+Q$，$Type(M_i)$ 为任务的类型，取值范围为 $\{C,G,T,S\}$，分别表示 CPU 任务、GPU 任务、通信任务和同步任务，$Time(M_i)$ 为任务所需的时间. 此外，记 $Q(M_i)$ 为任务 M_i 所属的任务队列的编号，对于所有的非异步任务，包括 CPU 任务、同步的 GPU、通信任务以及同步任务，都有 $Q(M_i)=0$. 除了默认的 0 号任务流外，假定程序中一共包含 q 条任务流. 特别地，对于同步任务，记 $TS(M_i)$ 为该同步函数所同步的任务流的编号，若同步函数是全局同步函数，则记 $TS(M_i)=0$. 根据 CUDA 的定义，同步的 GPU 和通信任务也具有全局同步的作用，因此对于它们也有 $TS(M_i)=0$. 算法 1 给出构造任务依赖图的具体过程.

算法 1. 构造任务依赖图.

1. Input: $\mathcal{M}=(M_1,M_2,\cdots,M_{P+Q})$
2. Output: $G=(V,E)$
3. let $T_0,T_1,\cdots,T_q$ be the latest node in each task stream
4. $T_0,T_1,\cdots,T_q\leftarrow\varnothing$ //$\varnothing$ means no task in the stream
5. $V\leftarrow\bigcup_{1\leq i\leq P+Q}\{v_i:\langle Type(M_i),Time(M_i)\rangle\}$; $E\leftarrow\varnothing$
6. for $i=1$ to $P+Q$ do
7. if $T_0\neq\varnothing$ then
8. $E\leftarrow E\cup\{e:\langle T_0,v_i\rangle\}$ //a program dependency
9. end if
10. if $Q(M_i)\neq 0$ then //for asynchronous task
11. if $T_{Q(M_i)}\neq\varnothing$ then
12. $E\leftarrow E\cup\{e:\langle T_{Q(M_i)},v_i\rangle\}$ //an stream dependency
13. end if
14. else if $TS(M_i)=0$ then //for global synchronous
15. for $j=1$ to q do
16. if $T_j\neq\varnothing$ then
17. $E\leftarrow E\cup\{e:\langle T_j,v_i\rangle\}$ //an synch dependency
18. $T_j\leftarrow\varnothing$
19. end if
20. end for
21. else //for local synchronous
22. $E\leftarrow E\cup\{e:\langle T_{TS(M_i)},v_i\rangle\}$ //an synch dependency
23. $T_{TS(M_i)}\leftarrow\varnothing$
24. end if
25. $T_{Q(M_i)}\leftarrow v_i$
26. end for

从算法 1 可以看出，在所有的任务按照程序序排列构成的序列中，所有的有向边都是从一个编号较小的结点指向一个编号较大的结点，这就保证了依赖图中必然不包含回路，是有向无圈图.

生成任务依赖图后，我们再考虑可以并发执行的任务之间的资源冲突，即在任务依赖图中加入因资源冲突导致的任务间的依赖关系. 考虑资源冲突的情况下，程序的实际执行过程依赖于一定的冲突仲裁机制. 本文的模型中我们假定 GPU 运行时采用如下冲突仲裁机制：若某一时刻任务流中同时存在多个满足先后依赖关系但存在资源冲突的任务，则优先选择任务流编号较小的任务执行. 例如，当 CPU 任务 C_1 执行完成后，满足依赖关系的任务包括 T_1、T_2 和 C_2，其中 T_1 和 T_2 同为通信任务，存在资源冲突不能同时执行，此时优先执行 T_1.

当两个任务间存在潜在的资源冲突时，必须为其定义一个先后的依赖关系，因此在任务依赖图中加入资源依赖的目标是：对于任意两个任务 M_i 和 M_j，如果它们使用了相同的资源(GPU 或通信部件)，则它们之间必须存在通路. 为了缩小考察的任务范围，我们首先分析任务依赖图中一定存在通路的情况. 根据算法 1 中添加依赖边的方法可以得知 AOV 网满足以下 3 个性质.

性质 1.　若 M_i 和 M_j 中至少有一个是同步模式的任务，则根据 CUDA 的规定，同步模式的 GPU 和通信任务本身隐含着一次全局同步，换言之，它们是任务依赖图中从程序开始到程序结束的必经结

点,它们和任意结点间都存在通路;

性质 2. 若 M_i 和 M_j 都是异步模式的任务,此时如果 $Q(M_i)=Q(M_j)$,即两个任务处于同一条任务流中,则它们之间必然存在通路;

性质 3. 否则,M_i 和 M_j 都是异步模式的任务,且 $Q(M_i)\neq Q(M_j)$,此时不妨假定在程序序中 M_i 早于 M_j(记为 $M_i<M_j$).根据算法 1 的第 8 行可知,对于异步模式的任务 M_j,必然存在一条有向边从某个位于 0 号任务流中任务指向 M_j,记这个 0 号任务流中的任务为 $Pre(M_j)$;此外,记 $Suc(M_i)$ 为由 M_i 可达的程序序最小的 0 号任务流中的任务.此时,从 M_i 到 M_j 存在通路的等价条件为 $Suc(M_i)\leqslant Pre(M_j)$.

只要符合上述 3 个条件之一,任务 M_i 和 M_j 之间一定存在通路,因此分析资源冲突时可以忽略这些情况.根据本文模型中对冲突仲裁机制的假设,加入资源依赖的过程可以简单描述为:对于给定的任务依赖图 G,如果两个异步模式的 GPU 或通信任务 M_i 和 M_j 使用的资源相同,即 $Type(M_i)=Type(M_j)$,且 M_i 和 M_j 之间不存在通路,则计算 M_i 和 M_j 最早可能开始执行的时间(Earliest Start Time,EST),分别记为 $EST(M_i)$ 和 $EST(M_j)$,若 $EST(M_i)=EST(M_j)$,则添加一条有向边,从任务流编号较小的任务指向任务流较大的任务,表明优先执行任务流编号较小的任务;若 $EST(M_i)\neq EST(M_j)$,不妨假设 $EST(M_i)<EST(M_j)$,则添加一条有向边,从任务 M_i 指向任务 M_j,表示 M_i 抢占了资源,M_j 必须等待 M_i 完成后才能开始.

对于给定的任务依赖图 $G=(V,E)$,其中任务 M_i 的最早可能开始执行时间 $EST(M_i)$ 可以由如下递推公式求出:

$$\begin{cases} EST(M_1)=0 \\ EST(M_i)=\max\limits_{\langle v_j,v_i\rangle\in E}\{EST(M_j)+Time(M_j)\} \end{cases} \tag{1}$$

式(1)中,要计算 $EST(M_i)$ 必须先计算所有的满足 $\langle v_j,v_i\rangle\in E$ 的那些任务的最早可能开始时间,换言之,递推的过程必须按照图中所有结点的一个拓扑序列(topological sequence)的顺序进行.注意到算法 1 构造任务依赖图时,所有的有向边都满足目标结点的编号大于源结点的编号,因此这里对结点编号的顺序(程序序)本身就是任务依赖图的一个拓扑序列.

对于给定的任务依赖图 $G=(V,E)$,记 M_1,M_2,…,M_{P+Q} 为按程序序排列的任务序列,$v(M_i)$ 为任务 M_i 在 G 中相应的结点,算法 2 给出了在任务依赖图加入资源依赖的具体过程.

算法 2. 加入资源依赖到任务依赖图.

```
1. Input: G=(V,E)
2. Output: G'=(V,E')
3. let T_T, T_G be the latest tasks on the communication
   and GPU resource
4. T_T, T_G ← ∅ //∅ means no task on the resource
5. E' ← E
6. for i=1 to P+Q do
7.   Calculate EST(M_i) by Equation (1)
8.   if Q(M_i)≠0 then //according to property 1
9.     if T_Type(M_i) = ∅ then
10.      T_Type(M_i) ← M_i
11.    else if Q(T_Type(M_i)) ≠ Q(M_i) then
                              //according to property 2
12.      if Suc(T_Type(M_i)) > Pre(M_i) then
                              //according to property 3
13.        if EST(M_i) = EST(T_Type(M_i)) then
14.          let M_s(M_l) be the task with smaller(larger)
             task stream number among M_i and T_Type(M_i)
15.          E' ← E' ∪ {⟨v(M_s), v(M_l)⟩}
16.          EST(M_l) ← EST(M_s) + Time(M_s)
17.          T_Type(M_i) ← M_l
18.        else
19.          let M_s(M_l) be the task with smaller(larger)
             EST among M_i and T_Type(M_i)
20.          E' ← E' ∪ {⟨v(M_s), v(M_l)⟩}
21.          EST(M_l) ←
                 max{EST(M_s)+Time(M_s), EST(M_l)}
22.          T_Type(M_i) ← M_l
23.        end if
24.      else //there exists a path
25.        T_Type(M_i) ← M_i
26.      end if
27.    else //there exists a path
28.      T_Type(M_i) ← M_i
29.    end if
30.  end if
31. end for
```

由于算法 2 中为两个任务结点添加资源依赖相关的有向边的前提是这两个结点间不存在通路,因此添加有向边之后也不会带来回路,即添加资源依赖关系后得到的 G' 依然是有向无圈图.至此,由算法 1 和算法 2,我们可以得到给定 CUDA 程序的 AOV 网表示.

以图 3 中所示的程序依赖图为例,各任务的一

个拓扑排序(程序序)为 $C_1,T_1,T_2,C_2,G_1,G_2,C_3,T_3,T_4,C_4,S$,假定各任务执行时间为 1,3,4,2,2,3,1,2,2,4,0,经过算法 2 处理得到的 AOV 网如图 4 所示.

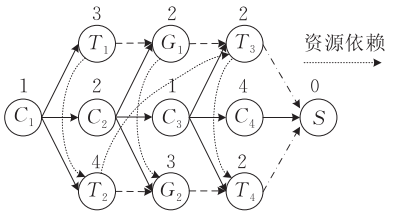


图 4　图 2 中示例程序段的 AOV 网表示

3.2　非关键任务的确定

建立 CUDA 程序的 AOV 网表示后,就可以确定降频后不影响程序执行总时间的非关键任务.

算法 2 在得到 CUDA 程序的 AOV 网表示的同时,还为每个任务 M_i 计算出最早可能开始时间 $EST(M_i)$. 为了确定非关键任务,我们还需要计算每个任务的最晚允许开始时间 LST(Latest Start Time),即在程序的总执行时间不变的前提下,每个任务最晚允许开始的时间. 为了计算 LST,需要对 CUDA 程序 AOV 网重新进行拓扑排序,这是由于算法 2 在分析资源依赖时,根据最早可能开始时间的关系可能加入从程序序编号较大指向程序序编号较小的结点的有向边,因此初始的程序序不一定仍是 AOV 网的拓扑排序. 但需要注意的是,重新构建拓扑排序并不改变算法 2 中计算的各任务的最早可能开始时间,这是由于算法 2 在添加表示资源依赖的有向边时,都对目标结点的最早可能开始时间进行了更新(第 16,21 行).

不失一般性,记 $G=(V,E)$ 为由算法 2 构造的 AOV 网络,其拓扑序列为 $M_1,M_2,\cdots,M_{P+Q}$,此时我们可以按照拓扑序列的逆序列进行如下递推求解总时间不变的前提下每个任务的最晚允许开始时间

$$\begin{cases} LST(M_{P+Q})=EST(M_{P+Q}) \\ LST(M_i)=\min\limits_{\langle v_i,v_j\rangle\in E}\{LST(M_j)-Time(M_i)\} \end{cases} \tag{2}$$

如果任务的最早可能开始时间等于其最晚允许开始时间,则该任务位于 AOV 网的关键路径上,其运行时间直接影响整个程序的运行时间,不能进行放松. 因此可以进行频率调节的任务是那些最早可能开始时间小于最晚允许开始时间的 CPU 任务和 GPU 任务.

3.3　频率调节幅度的求解

为了求解能量最优的目标下,每个非关键的 CPU 任务和 GPU 任务的频率调节幅度,我们还需要对 AOV 网进行进一步的划分,以确定每个非关键 CPU、GPU 任务需要满足的时间约束.

我们考察 AOV 网中的必经结点. 所谓必经结点是指从 AOV 网中第一个任务到最后一个任务的任一路径必然经过的结点. 根据 CUDA 程序的特征容易知道,在 AOV 网中,具有全局同步属性的结点都是 AOV 网的必经结点,包括全局同步任务以及非异步模式的 GPU 和通信任务. 此外,如果一个必经结点的出度为 1,则其唯一的后继结点也是必经结点. 由于必经结点上的任务不和任何其它任务重叠执行,因此必经结点一定在 AOV 网的关键路径上.

假设 $M_1,M_2,\cdots,M_{P+Q}$ 为 G 的一个拓扑序列,G 中包含 k 个必经结点,分别为 $M_1^c,M_2^c,\cdots,M_k^c$,其中 $M_1^c=M_1,M_k^c=M_{P+Q}$. 因此根据必经结点和拓扑序列,我们可以将 G 中的非必经结点划分为 $k-1$ 个集合 $S_i(1\le i\le k-1)$,每个集合包含拓扑序列中连续两个必经结点之间的所有结点,即 $S_i=\{M_x\mid M_i^c<M_x<M_{i+1}^c\}$,其中 $<$ 表示拓扑序列中的序关系. 注意到,若两个必经结点在拓扑序列中相邻,则相应的非必经结点集合为空集. 前面提到,由于必经结点上的任务一定是关键任务,因此进行频率调节的目标一定位于这里构造的非必经结点集合中.

由于我们的能耗优化问题中要求程序的总执行时间不变,显然,程序的总执行时间等于关键路径上的所有任务的执行时间之和,因此每个关键任务的最早可能开始时间和最迟允许开始时间在优化过程中都是常量,不能被改变. 换言之,对非必经结点集合 S_i 中的 CPU 和 GPU 任务进行降频操作只要不增大 M_{i+1}^c 任务的最早可能开始时间即可. 另一方面,只要满足上述条件,由于边界结点时间是常量,因此对某一个非必经结点集合中的任务进行优化不会影响其它集合. 因此整个程序的能耗优化问题可以归结为对每个非必经结点集合的能耗优化问题.

针对非必经结点集合 S_i,我们构造一个 AOV 子网 $AOV_{S_i}=\{M_i^c,S_i,M_{i+1}^c\}$. 考虑到 S_i 中可能包含关键任务结点,并且通信任务和同步任务结点时间不可变,因此我们只考虑对 S_i 中的非关键的 CPU 和 GPU 任务结点进行频率调节. 假定 AOV 子网中有 N 个非关键的 CPU 和 GPU 任务结点,记为 $M_1^i,\cdots,M_N^i$,其初始运行频率记为 $f(M_j^i),1\le j\le N$,进行频率调节后它们的频率分别变为 $f'(M_j^i)$,此时各任务的执行时间变为 $Time'(M_j^i)=$

$\frac{Time(M_j^i)\times f(M_j^i)}{f'(M_j^i)}$，其它任务的时间不变. 在这个AOV子网中，根据式(1)我们可以计算出调节频率后 M_{i+1}^c 的最早可能开始时间，记为 $EST'(M_{i+1}^c)$. 至此，S_i 的能量最优化问题可以归结为以下 N 元极值问题

$$\begin{cases}\min \sum_{j=1}^{N} k_j f'(M_j^i)^2 \\ \text{s.t. } EST'(M_{i+1}^c)=EST(M_{i+1}^c)\end{cases} \tag{3}$$

其中，k_j 表示任务 M_j^i 对应的处理器相关的功耗系数，即 $E=kf^2$.

为了简化式(3)的求解，我们还可以根据AOV子网中非关键任务结点的连接情况缩减极值问题的空间. 由式(1)可知，对某任务进行频率调节，改变其执行时间后，它只能影响从其可达的结点的最早可能开始时间. 因此如果集合 S_i 可以划分为若干个子集合，它们之间两两互相不可达，则对某一个子集合进行频率调节不会影响到其它子集合，它们都可以按照上述方法单独进行求解，因此降低了极值问题的求解空间. 算法3给出了对集合 S_i 进行划分的方法.

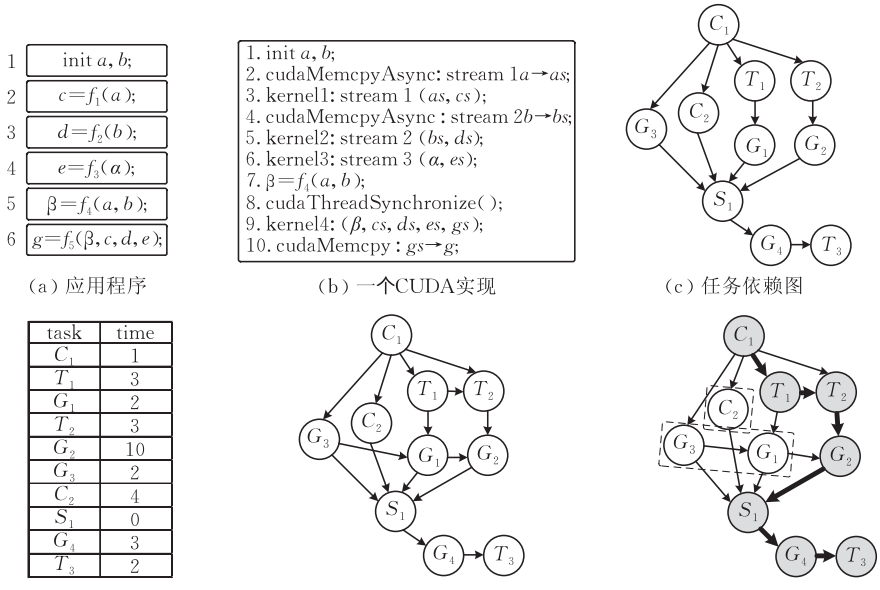
算法3. 划分 S_i 为多个相互不可达集合.

1. Input: S_i
2. Output: $S_i^1, S_i^2, \cdots$
3. $j \leftarrow 1$
4. while $S_i \neq \varnothing$ do
5. $S_i^j \leftarrow \varnothing$
6. let m be an arbitrary node in S_i
7. $S_i^j \leftarrow \{m\}$
8. repeat
9. $S_i^j \leftarrow S_i^j \cup \{(S_i^j)^*\}$//$(S_i^j)^*$ means all reachable nodes from the nodes in S_i^j
10. rotate all edges in AOV_{S_i}
11. $S_i^j \leftarrow S_i^j \cup \{(S_i^j)^*\}$
12. until S_i^j remains unchanged
13. return S_i^j
14. $S_i \leftarrow S_i - S_i^j$
15. $j \leftarrow j+1$
16. end while

4 案例分析

由于我们的能耗优化方法的重点在于归纳并分析CUDA程序中的依赖关系，将其表示为AOV网络，而将最终AOV网中频率的求解归结为一个规划问题——N 元极值问题. 因此从理论上说，我们的方法可以给出问题的最优解. 极值问题的数学求解过程不是本文关注的重点，因此本节通过案例分析给出我们的程序分析方法的执行过程，并给出模拟的能耗优化结果，以验证我们优化方法的有效性.

图5给出了一个案例程序的分析过程. 其中图(a)为原始算法流程，共包括6个步骤：首先是对 a，b 两个数组的初始化，然后调用过程 f_1 和 f_2 分别对 a 和 b 进行处理，得到数组 c 和 d. 第4行表示函数 f_3 由一个标量 α 计算出一个数组 e；第5行则表示函数 f_4 由 a，b 两个数组计算出标量结果 β；最后一行



(a) 应用程序 (b) 一个CUDA实现 (c) 任务依赖图

(d) 任务执行时间 (e) AOV网络 (f) 关键路径

图5 案例分析

表示函数 f_5 由 β,c,d 和 e 计算出数组 g. 图(b)给出了上述算法的一个 CUDA 实现. 假设 f_1,f_2,f_3 和 f_5 函数可以被并行化，因此使用 Kernel 函数实现，分别对应于 Kernel1～Kernel4；f_4 函数不能被并行化，因此仍由 CPU 完成. Kernel1 和 Kernel2 执行之前需要调用 cudaMemcpy 将输入数组 a 和 b 载入 GPU 存储器，Kernel4 执行结束后需将数组 g 回存至 CPU 存储器. 为了开发 Kernel 计算和通信的并行性，隐藏通信开销，CUDA 实现中将 Kernel1、Kernel2、Kernel3 及其对应的通信操作设定为异步模式，而 Kernel4 使用到了前 3 个 Kernel 函数的输出，因此调用 Kernel4 之前先进行全局的任务流同步操作.

图(c)给出了经过算法 1 处理后得到的任务依赖图，假定各个任务的执行时间如图(d)中所列，则根据算法 2 处理后，加入了 T_1 到 T_2，G_3 到 G_1 和 G_1 到 G_2 的资源依赖边，生成的 AOV 网络如图(e)所示. 此时利用式(1)和(2)可以推算出各个结点的最早可能开始和最迟允许开始时间，如表 1 所示.

表 1　各任务的 *EST* 和 *LST*

任务	*EST*	*LST*
C_1	0	0
T_1	1	1
G_1	4	5
T_2	4	4
G_2	7	7
G_3	1	3
C_2	1	13
S_1	17	17
G_4	17	17
T_3	20	20

可以看出，任务 $C_1,T_1,T_2,G_2,S_1,G_4,T_3$ 的最早可能开始时间和最晚允许开始时间相等，它们构成 AOV 网的关键路径，如图(f)中的阴影结点所示. 因此系统中可以进行频率调节的非关键任务包括 G_1、G_3 和 C_2. 根据 3.3 节的分析，这 3 个任务可以通过算法 3 划分为互不可达的两组，如图(f)中的虚线框所示，可以独立进行功耗调节. C_2 的情况比较简单，它的最早可能和最晚允许开始时间分别为 1 和 13，因此根据式(3)给出的限定条件，为 C_2 调节 CPU 的频率只要不影响 S_1 的最早可能开始时间即可. 根据式(1)可知 C_2 的执行时间可以延长为 16，因此执行任务 C_2 时，CPU 的频率最低可降至原来的 1/4，C_2 的能量消耗降为原来的 1/16. 对于 G_1 和 G_3，同理可知对 GPU 的频率调节不能影响 S_1 的最早可能开始时间. 由于 G_1 和 G_3 的初始运行时间都是 2，不妨假设它们在初始频率下消耗的能量都为 E，调节后运行时间变为 t'_{G_1} 和 t'_{G_3}. 根据式(1)，G_1 和 G_3 任务能量最优的运行时间可表示为

$$\begin{cases}\min\left\{\left(\dfrac{2}{t'_{G_1}}\right)^2 E+\left(\dfrac{2}{t'_{G_3}}\right)^2 E\right\}\\ \text{s.t.}\ \max\begin{cases}1+t'_{G_3}\\ EST'(G_1)+t'_{G_1}=17\\ EST'(G_2)+10\end{cases}\end{cases},$$

其中，

$$EST'(G_1)=\max\begin{cases}1+t'_{G_3}\\ 4\end{cases},$$

$$EST'(G_2)=\max\begin{cases}EST'(G_1)+t'_{G_1}\\ 7\end{cases}.$$

求解上式可得，$t'_{G_1}=t'_{G_3}=3$ 时，两者总能量消耗最小，为 $8/9E$.

通过以上的例子说明，我们的方法可以直观地将 CUDA 程序的能耗优化问题转变为基于 AOV 网的数学规划问题. 需要说明的是，实际中很多数学规划问题是 NP 问题，因此本文的方法可以获得的能耗优化效果主要取决于规划问题求解的精度，而这不是本文所关注的主要问题，因此这里不做详细讨论.

5　相关工作

本节分两个部分介绍和本文相关的工作.

首先是异构系统的任务调度和功耗优化研究. Mudge 等人基于 Amdahl 定律建立了传统多核、同构众核和异构众核体系结构的性能模型，通过分析认为由一个高性能计算核心和大量结构简单的高能效计算核心组成的异构体系结构可以有效提高并行处理能力，并能保证串行执行效率[7]. 文献[2]通过建立这 3 种体系结构的功耗模型，指出相对于同构体系结构，异构体系结构具有更高的能效优势. 文献[8]中，作者面向同构多核体系结构建立了性能与能耗的关系，分析了在给定加速比约束的前提下调节处理器的频率以达到能耗最优. 随着 GPU 越来越多地被应用到通用计算领域，面向 CPU-GPU 异构系统的任务调度与功耗优化问题逐渐成为该领域的研究热点. Yang 等人[9]提出了一种动态的异构处理器选择方法，通过动态采样程序在不同计算单元上的执行时间，选择性能较高的处理器作为后续执行单元从而优化程序的性能. 文献[10]提出了一种

多 GPU 自适应负载平衡手段,通过在 CPU 和 GPU 间建立任务队列使得 GPU 可以根据本地忙闲状态自适应地选择任务执行. Hermann 等人[11]提出了一种异构系统的负载平衡策略,在综合考虑任务亲和性和处理器差异性的基础上,结合任务划分和任务窃取指导 CPU-GPU 间任务调度. 文献[12]通过分析程序行为,借用模糊逻辑计算程序与处理器核的适应度,用以指导能量感知的异构多核系统的程序调度. 此外,关于异构系统的任务调度的研究还包括文献[13-15]. 上述研究大部分关于单个任务,通常是循环如何在异构系统上进行调度和功耗优化,而本文则关注整个程序的功耗优化,这里任务在处理器上的分配由程序决定,而优化整个程序的功耗则需要分析各任务在异构的处理器上的依赖关系,这也是本文重点研究的内容.

其次是关于 GPU 性能和功耗分析方面的研究. 精确的性能分析模型是根据程序特征进行处理器低功耗优化的重要基础. 文献[6]从程序并行度的角度出发,分析 GPU 程序中计算并行度和访存并行度的关系,从而确定出性能瓶颈,进而给出 GPU 程序的性能分析模型. 文献[16]则引入了工作流图,作为 GPU Kernel 的一种抽象表示,并基于此评估 GPU Kernel 的执行时间. 文献[17]则提出了一个整个的 GPU 性能分析和功耗分析模型,用于预测给定程序所需要的最优的 GPU 核数. 这些模型都可以用于分析给定程序在 GPU 上的执行时间,因此本文在进行异构系统能耗优化时,假定 GPU 任务的执行时间已知,没有讨论性能分析问题.

6 结 论

异构系统的功耗优化问题成为异构体系结构的研究热点. 本文面向 CPU-GPU 体系结构归纳总结了 CUDA 程序中包含的任务类型以及任务之间的依赖关系,研究了如何将 CUDA 程序在异构系统上的执行过程描述为一种抽象的数学表示 AOV 网络,并基于 AOV 网络求解程序的关键路径,找到在不影响程序总执行时间的前提下可以进行 DVFS 降频优化的非关键任务,进而求解能量最优目标下每个非关键任务的频率调节幅度. 通过案例分析可以说明,本文方法可以有效地将 CUDA 程序的能耗优化问题转变为基于 AOV 网的数学规划问题,从而给出最优或近优的优化策略.

参 考 文 献

[1] Luebke D, Harris M, Krüger J, Purcell T, Govindaraju N, Buck I, Woolley C, Lefohn A. GPGPU: General purpose computation on graphics hardware//Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC'06). New York, 2006: 33

[2] Hill Mark D, Marty Michael R. Amdahl's law in the multicore era. Computer, 2008, 41(7): 33-38

[3] Kirk D. NVIDIA cuda software and GPU parallel computing architecture//Proceedings of the 6th International Symposium on Memory Management (ISMM'07). New York, 2007: 103-104

[4] Brock B, Rajamani K. Dynamic power management for embedded systems//Proceedings of the IEEE SOC Conference, Portland, Oregon, USA, 2003: 416-419

[5] Choi Kihwan, Soma Ramakrishna, Pedram Massoud. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2005, 24(1): 18-28

[6] Hong Sunpyo, Kim Hyesoon. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness//Proceedings of the 36th Annual International Symposium on Computer Architecture. Austin, TX, USA, 2009: 152-163

[7] Mudge T. Power: A first class design constraint for future architectures. IEEE Computer, 2001, 34(4): 52-58

[8] Woo Dong Hyuk, Lee Hsien-Hsin S. Extending Amdahl's law for energy-efficient computing in the many-core era. Computer, 2008, 41(12): 24-31

[9] Yang Canqun, Wang Feng, Du Yunfei, Chen Juan, Liu Jie, Yi Huizhan, Lu Kai. Adaptive optimization for petascale heterogeneous CPU/GPU computing//Proceedings of the IEEE International Conference on Cluster Computing. Heraklion, Greece, 2010: 19-28

[10] Chen Long, Villa O, Krishnamoorthy S, Gao G R. Dynamic load balancing on single- and multi-GPU systems//Proceedings of the IEEE International Symposium on Parallel and Distributed Processing. Atlanta, GA, 2010: 1-12

[11] Hermann Everton et al. Multi-GPU and multi-CPU parallelization for interactive physics simulations//Proceedings of the 16th International Euro-Par Conference on Parallel Processing: Part II (Euro-Par'10). Ischia, Italy, 2010: 235-246

[12] Chen Jian, John Lizy K. Energy-aware application scheduling on a heterogeneous multi-core system//Proceedings of the IEEE International Symposium on Workload Characterization. Seattle, WA, 2008: 5-13

[13] Kumar R, Farkas K I, Jouppi N P, Ranganathan P, Tullsen D M. Single-ISA heterogeneous multi-core architectures:

The potential for processor power reduction//Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture. San Diego, California, 2003: 81

[14] Kumar R, Tullsen Dean M, Norman P. Core architecture optimization for heterogeneous chip multiprocessors//Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques (PACT'06). Seattle, Washington, USA, 2006: 23-32

[15] Maheswaran M, Siegel H J. A dynamic matching and scheduling algorithm for heterogeneous computing systems//Proceedings of the Heterogeneous Computing Workshop. Orlando, FL, USA, 1998: 57-69

[16] Baghsorkhi Sara S, Delahaye Matthieu, Patel Sanjay J, Gropp William D, Hwu Wen-mei W. An adaptive performance modeling tool for GPU architectures//Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. Bangalore, India, 2010: 105-114

[17] Hong Sunpyo, Kim Hyesoon. An integrated GPU power and performance model//Proceedings of the 37th Annual International Symposium on Computer Architecture. Saint-Malo, France, 2010: 280-289

LIN Yi-Song, born in 1983, M. S.. His research interests include high performance computing and low power optimization.

YANG Xue-Jun, born in 1963, Ph. D., professor, member of Chinese Academy of Sciences. His research interests include high performance computing, parallel computer architecture, high performance compiler and operating system.

TANG Tao, born in 1984, M. S.. His research interests include high performance computing and compiler optimization.

WANG Gui-Bin, born in 1981, M. S.. His research interests include high performance computing and low power optimization.

XU Xin-Hai, born in 1984, M. S.. His research interests include high performance computing and compiler optimization.

Background

Heterogeneous architecture has become an important trend of constructing high performance computing system recently. Hybrid systems integrating CPUs and GPUs are typical representations of this architecture. Consequently the power consumption optimization of these heterogeneous systems becomes an urgent research problem in high performance computing area. Most current researches about power optimization of CPU-GPU heterogeneous systems focus on the partition of single kernel task, by balancing the performance constraint and the power constraint via partitioning the task between the CPU and the GPU. However, few works consider the energy optimization of the whole program for such a system.

In this paper, we first analyze the execution characteristics of the CUDA program on the CPU-GPU heterogeneous system deeply, and induce the dependency relationships of tasks in the program in detail. Then, we describe how to express the execution of the program with AOV network, based on which we propose the energy optimization method. Our method analyzes the critical path of the program from the AOV network, detects the tasks that can be optimized for energy and calculates the frequency scaling factors to minimize the whole program's energy consumption while maintaining the performance.

This work is supported by the National Natural Science Foundation of China under grant Nos. 60921062 and 60873016. These projects mainly refer to the high performance computing and the stream architectures. Our research group has worked on several problems in low power optimization area for GPU-related heterogeneous systems, including memory-aware energy consumption optimization of single kernel on GPU, energy consumption optimization for single kernel on the processor of GPU based on software prefetching, and energy consumption optimization for single kernel on the multi-GPU system based on task partition, etc. The work presented in this paper is to address the energy consumption optimization for a whole program on the CPU-GPU system.