doi:10.3969/j.issn.1005-3026.2016.08.006

基于SVM的CPU-GPU异构系统任务分配模型

王彦华,乔建忠,林树宽,赵廷磊
(东北大学 计算机科学与工程学院,辽宁 沈阳 110819)

摘 要:为了改善异构系统的性能和效率,提出并实现了一个两阶段的任务分配模型.该模型对预分配给CPU和GPU的任务集进行多轮调整,以此最大程度地缩短程序的执行时间.首先,使用支持向量机进行任务预处理,支持向量机将任务分成CPU型和GPU型;然后,根据预处理结果以及处理器的特征和状态,并在对分配集合进行多轮调整后实施实际的任务分配.本模型在具体的异构系统中实现,使用多种基准程序进行检测.实验结果表明,对比其他任务分配算法,本文算法能够使性能获得平均43.54%的提升.

关 键 词:图形处理单元;支持向量机;异构系统;机器学习;任务预处理;任务分配
中图分类号:TP 391 **文献标志码**:A **文章编号**:1005-3026(2016)08-1089-06

A Task Allocation Model for CPU-GPU Heterogeneous System Based on SVMs

WANG Yan-hua, QIAO Jian-zhong, LIN Shu-kuan, ZHAO Ting-lei
(School of Computer Science & Engineering, Northeastern University, Shenyang 110819, China. Corresponding author: WANG Yan-hua, E-mail: eighthr@163.com)

Abstract: To improve the performance and efficiency of heterogeneous system, a two-stage task allocation model is proposed and implemented, by which the workload allocated to CPU and GPU is adjusted several times to decrease the execution time to the maximum extent. Firstly, the support vector machine (SVM) is used to classify a task into CPU and GPU in pre-treating. Then, after adjusting the allocation sets several times, the model carries out task allocation in the light of the characteristic and status of processors and the result produced by the first stage. Moreover, a real heterogeneous system is evaluated through several benchmarks on the proposed model. Experimental results demonstrate that the proposed model can achieve an average 43.54% of performance improvement, compared with some of the leading-edge allocation techniques.

Key words: GPU (graphics processing unit); SVM (support vector machine); heterogeneous system; machine learning; task pre-treat; task allocation

GPU适合并行计算和密集计算.GPU通用计算能大幅度改善应用程序的性能和能耗.在CPU和GPU异构系统中,处理器协作处理负载,CPU将部分负载分配给GPU并控制其执行[1].如果分配的计算任务以及任务量适当就能减轻CPU的计算负担,并缩短应用程序执行时间,改善系统的吞吐量和性能.不同的任务分配份额和分配集合能够不同程度地改变系统性能.为了最小化执行时间,文献[2]提出了一个适应性方法,将计算任务映射给CPU和GPU.文献[3]设计了一个动态的负载分配算法,根据上一周期的处理时间调整下一周期的任务分配比例,在有限的周期内使CPU和GPU的处理时间达到一致.文献[4]提出了一个多CPU和多GPU中平衡负载的适应性优化架构.文献[5]从静态程序中提取代码,使用机器学习的方法为负载预测其分片,产生了一个预测模型.文献[6]提出了一个优化的整数规划解决方案,为任务在异构系统中的分配问题提供了

收稿日期:2015-05-20
基金项目:国家自然科学基金资助项目(61272177);辽宁省软件系统开发与应用重点实验室项目.
作者简介:王彦华(1982-),女,河北保定人,东北大学博士研究生;乔建忠(1964-),男,辽宁兴城人,东北大学教授,博士生导师.

理论支持. 针对 GPU 的异构集群,文献[7]提出了一个基于 Hadoop 的混合任务分配处理技术. 目前的任务分配方法直接依据处理器的处理能力或利用率进行任务分配,分配前未对任务进行预处理. 有文献在任务分配时仅按照任务量的比例大小进行分配[3-4],没有考虑任务的基本特征,并且分配过程产生了一定的开销. 本文提出了一个智能的两阶段任务分配方案. 首先进行任务预处理,基于任务的基本特征使用 SVM 将其分成适合在 CPU 或 GPU 上执行的任务. 然后,综合考虑处理器的运行状态和处理能力以及预处理的分类结果,并在多轮任务集调整后进行任务的实际分配. 有效地提高了任务分配效率和系统整体性能,且分配开销较小.

1 CPU 和 GPU 异构平台上的任务分配模型

具有并行特征的程序能够分解成多个任务,任务关联不同的数据,可将任务映射到不同的处理器. 合理的映射方法能充分利用处理器的计算能力.

1.1 任务模型

根据输入输出数据量和计算量将任务分为输入型 in,输出型 out,逻辑型 log 和计算型 com. 一个复杂的任务是几种类型的组合. 任务 x 的一般形式为 x(type,size,datasize,concurrency,kind). type x 的类型,type $\subseteq$ {in,out,com,log} 并且 type $\neq \varnothing$. size x 的大小,单位 KB. datasize x 需要输入和输出的数据量,单位 KB. concurrency x 的并行化程度,由 x 的并行执行的线程数量决定. kind x 经过预处理的结果,kind = $\{y \mid y = +1, -1\}$.

1.2 任务预处理模型

本系统包括两类处理器. 现有实例是已知函数库中的有限数量的函数. 使用 SVM[8] 在小样本的基础上根据机器学习的方法训练出分类函数. 图 1 为部分学习样本的分类效果.

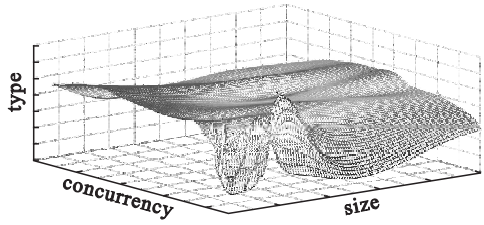

图 1 样本的分类效果

Fig. 1 Result of samples classification

根据图 1,任务的分类属于非线性分类问题,使用核函数将低维特征空间映射到高维特征空间.

1) 定义任务 x 的分类超平面:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \kappa(x_i, x) + b, \tag{1}$$

选取核函数:

$$\kappa(x_1, x_2) = (\langle x_1, x_2 \rangle + 1)^2. \tag{2}$$

参数 α 由式(3)的对偶问题求解得到.

$$\left.\begin{aligned} &\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\alpha_i \alpha_j y_i y_j \kappa(x_i, x_j); \\ &\text{s. t. } \alpha_i \geqslant 0, i = 1, 2, \cdots, n; \sum_{i=1}^{n} \alpha_i y_i = 0. \end{aligned}\right\} \tag{3}$$

2) 根据训练所得分类超平面 $f(x)$,将预分类的任务 x 输入到 $f(x)$,若 $y = +1$,则 x 适合在 CPU 上执行;若 $y = -1$,则 x 适合在 GPU 上执行.

定义 1(任务不可分解) 对于任意任务 x,若 size(x) $\leqslant s$(阈值,$s = 0.5$),则认为该任务是不可分解的.

性质 1 对于任意不可分解的任务 x,将其放置于指定处理器上执行. 本文设定为 CPU,令 $y = +1$.

任务 x 的预处理过程如下:

①调用式(1).

②若 $0 < yf(x) < 1$,任务 x 无法明确分类. 将 x 分解为一系列子任务 $x_j, j \geqslant 2$.

③使用 SVM 对 x_j 分类.

④若 $yf(x_j) \geqslant 1$,则 x_j 分类成功. 否则若 x_j 符合定义 1,将 x_j 放入增量样本集 S 中,同时根据性质 1 处理,否则返回步骤③.

⑤若任务 x_j 分类成功,则预处理结束. 得到两个分别适合在 CPU 和 GPU 处理器上运行的任务集 C_x 和 G_x. 并且任务集满足 $C_x \cap G_x = \varnothing$, $|x| = |C_x| + |G_x|$.

当增量样本集 S 中样本数量 $|S| \geqslant S_0$($S_0 = 50$)时,将 S 中的样本加入到训练集并置 $S = \varnothing$,调整参数,实现任务预处理模型的不断优化,提高其泛化能力. 随着增量样本集的扩大以及不断迭代,有可能会产生过拟合的现象,使用结构风险最小化来防止过拟合.

1.3 基于预处理的任务分配模型

1.3.1 系统模型

本系统由异构的多种处理器组成. 处理器 p(type,util,comp). 系统 system = {bandwidth(PCI-E), $p_i, i = 1, 2, \cdots, q$, $q > 1$}. 定义 $q = 2$,令

type(p_1) = CPU，type(p_2) = GPU. 其中，type：处理器 p_i 的类型，type ∈ {CPU，GPU}；util：处理器 p_i 的利用率；comp：处理器 p_i 的浮点计算能力；bandwidth(PCI-E)：PCI-E 总线带宽.

1.3.2　数据依赖

定义 2(数据依赖 data-depend)　对于任意两个预处理后的任务 x_i，x_j($i \neq j$)，若 x_j 或 x_i 必须等待 x_i 或 x_j 的输出数据才能执行，则 x_i，x_j 存在数据依赖关系. 对于存在数据依赖关系的 x_i，x_j，若满足条件 $x_i \in C_x$ 且 $x_j \in C_x$，或者 $x_i \in G_x$ 且 $x_j \in G_x$，则 x_i，x_j 存在内部数据依赖关系. 若 x_i，x_j 满足条件 $x_i \in C_x$ 且 $x_j \in G_x$，或着 $x_i \in G_x$ 且 $x_j \in C_x$，则 x_i，x_j 存在外部数据依赖关系.

异构系统中应用程序的执行时间是分配在 CPU 上任务的执行时间与分配在 GPU 上任务执行时间的较大值. 若能最大程度地缩小两者的差距，则系统的性能更优. 任务的执行时间包括任务的计算时间和等待数据时间. 任务计算前需要其输入数据已经就绪，而该数据可能是由其他任务计算得到，该段时间为任务的等待数据时间. 等待数据时间因数据依赖关系而不同. 任务的数据依赖关系由图 2 所示的基本数据依赖组成，任意应用程序任务之间的数据依赖关系均可以最终分解为该组基本数据依赖关系，表 1 计算出所有基本数据依赖关系的等待时间.

表 1　等待数据时间

Table 1　Waiting time for data

依赖关系	Wait(x_j)	Wait(x_k)，…，Wait(x_l)	依赖类型
(a)，(b)	$\mathrm{Exec}(x_i)$	—	内部依赖
(c)，(d)	$\mathrm{Exec}(x_i) + \mathrm{Trans}(x_i)$	—	外部依赖
(e)，(l)	$\mathrm{Max}(\mathrm{Exec}(x_i), \mathrm{Exec}(x_k) + \mathrm{Trans}(x_k))$	—	内部、外部依赖
(f)	$\mathrm{Exec}(x_i) + \mathrm{Trans}(x_i) + \mathrm{Exec}(x_k)$	$\mathrm{Exec}(x_i) + \mathrm{Trans}(x_i)$	内部、外部依赖
(g)，(i)	$\mathrm{Exec}(x_i) + \mathrm{Max}(\mathrm{Trans}(x_i), \mathrm{Exec}(x_k)) + \mathrm{Trans}(x_k)$	$\mathrm{Exec}(x_i)$	内部、外部依赖
(h)，(k)	$\mathrm{Exec}(x_i) + \mathrm{Trans}(x_i) + \mathrm{Exec}(x_k) + \mathrm{Trans}(x_k)$	$\mathrm{Exec}(x_i) + \mathrm{Trans}(x_i)$	内部、外部依赖
(j)	$\mathrm{Exec}(x_i) + \mathrm{Trans}(x_i) + \mathrm{Exec}(x_k)$	$\mathrm{Exec}(x_i) + \mathrm{Trans}(x_i)$	外部依赖
(m)，(o)	$\mathrm{Exec}(x_i) + \mathrm{Trans}(x_i) + \frac{1}{n}(\sum_{\rho=n-1}^{\rho=1,k=j-1} \rho \cdot \mathrm{Exec}(x_k))$	Wait(x_j)	外部依赖
(n)，(p)	$\mathrm{Exec}(x_i) + \mathrm{Trans}(x_k) + \sum_{i}^{k-1} \max(\mathrm{Exec}(x_{i+1}), \mathrm{Trans}(x_i))$	$\frac{1}{n}\sum_{\rho=n-1}^{\rho=1,i=k-1} \rho \cdot \mathrm{Exec}(x_i)$	外部依赖
(q)，(r)	$\sum_{i}^{k} \mathrm{Exec}(x_i)$	$\frac{1}{n}\sum_{\rho=n-1}^{\rho=1,i=k-1} \rho \cdot \mathrm{Exec}(x_i)$	内部依赖
(s)，(t)	$\mathrm{Exec}(x_i) + \mathrm{Exec}(x_k)$	$\mathrm{Exec}(x_i)$	内部依赖

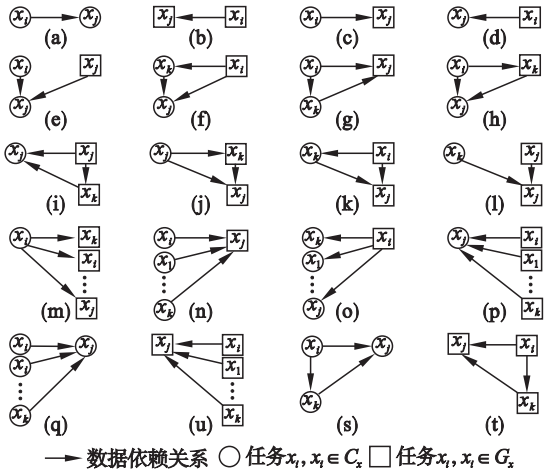

图 2　任务之间的基本数据依赖关系

Fig. 2　Basic data relation among tasks

$\mathrm{Exec}(x_i)$ 为任务 x_i 的计算时间，根据式(5)计算. $\mathrm{Trans}(x_i)$ 为任务 x_i 在 PCI-E 总线上的传输时间，根据式(8)计算. Wait(x_j)，Wait(x_k) 为任务 x_j，x_k 的等待数据时间.

任何应用程序的数据依赖关系图 $D(x)$ 均可拆分成多个基本数据依赖关系，拆分依据如下：

①对于任意任务 x_j ($x_j \in D(x)$)，选取边集 $E(j) = \{\langle x_i, x_j \rangle\}$. 选取边集 $E(i) = \{\langle x_m, x_n \rangle, | x_m \in \{x_i\}, x_n \in \{x_i\}, m \neq n\}$. 选取边集 $E(k) = \{\langle x_i, x_k \rangle\}$. 选取边集 $E(l) = \{\langle x_k, x_j \rangle\}$.

②将边集 $E(j)$，$E(i)$，$E(k)$ 和 $E(l)$ 及其所关联的节点构成子图 subD(x_j).

③如果 subD(x_j) 能与图 2 中的基本数据依赖关系匹配，则 Wait(x_j) 和 Wait(x_k) 使用表 1

计算.

④否则将 subD(x_j)优先按照基本数据依赖关系(e) ~ (t)以节点 x_j 为界进行拆分. 按照表 1 计算Wait(x_j)和 Wait(x_k).

⑤若节点 x_j, x_k 包含在多个子图中,则 Wait(x_j)和 Wait(x_k)分别取最大值.

当任务 x_j 从 C_x 转移到 G_x,或者从 G_x 转移到 C_x,其数据依赖关系发生改变,表 2 列出了数据依赖关系的转换.

表 2 数据依赖关系的转换

Table 2 Transformation of data relation

任务 x_j 从 C_x 转移到 G_x	任务 x_j 从 G_x 转移到 C_x
(a)→(c);(d)→(b);	(b)→(d);(c)→(a);
(e)→(l);(f)→(k);	(g)→(s);(j)→(h);
(h)→(j);(i)→(t);	(k)→(f);(l)→(e);
(o)→(b),(o);(p)→(r);	(m)→(a),(m);(n)→(q);
(q)→(n);(s)→(g).	(r)→(p);(t)→(i).

2 CPU 和 GPU 异构平台上的任务分配算法

1) 计算系统的各处理器当前处理能力 handle_{pi},是处理器固有计算能力和当前利用率的综合指标. 其中,参数 θ_1 和 θ_2($\theta_1+\theta_2=1$, $0<\theta_1<1$, $0<\theta_2<1$)在应用程序中定义,决定了处理器当前利用率对比固有计算能力的重要性. 实验选取 $\theta_1=0.315$, $\theta_2=0.685$. handle_{pi} 计算方法如式(4)所示.

$$\text{handle}_{p_i}=\theta_1\text{util}(p_i)+\theta_2\text{comp}(p_i). \tag{4}$$

对于程序 x 由预处理分类得到的两个任务集合 C_x 和 G_x,根据式(5)和式(6)计算 Exec(C_x), Exec(G_x). 参照表 2 和式(7)计算 Wait(C_x), Wait(G_x). 由式(9)计算 x 的执行时间 Manage(x). 其中,经验参数 λ 和 μ($0<\lambda<0.5$, $0<\mu<0.5$)根据一组微基准程序经多次实验获得, $\lambda=0.3$, $\mu=0.3$.

$$\text{Exec}(x_i)=\begin{cases}\lambda\cdot\dfrac{\text{size}(x_i)}{\text{handle}_{p_1}}, x_i\in C_x;\\ \mu\cdot\dfrac{\text{size}(x_i)}{\text{handle}_{p_2}}, x_i\in G_x;\\ \dfrac{(1-\mu)}{\mu}\cdot\dfrac{\text{size}(x_i)}{\text{handle}_{p_2}}, x_i\to G_x;\\ \dfrac{(1-\lambda)}{\lambda}\cdot\dfrac{\text{size}(x_i)}{\text{handle}_{p_1}}, x_i\to C_x.\end{cases} \tag{5}$$

$$\left.\begin{aligned}\text{Exec}(C_x)&=\sum_{i=1}^{|C_x|}\text{Exec}(x_i),\\ \text{Exec}(G_x)&=\sum_{i=1}^{|G_x|}\text{Exec}(x_i).\end{aligned}\right\} \tag{6}$$

$$\left.\begin{aligned}\text{Wait}(C_x)&=\sum_{i=1}^{|C_x|}\text{Wait}(x_i),\\ \text{Wait}(G_x)&=\sum_{i=1}^{|G_x|}\text{Wait}(x_i).\end{aligned}\right\} \tag{7}$$

$$\text{Trans}(x_i)=\frac{\text{datasize}(x_i)}{\text{bandwidth(PCI-E)}}. \tag{8}$$

$$\left.\begin{aligned}&\text{Manage}(x)=\text{Exec}(C_x)+\text{Wait}(C_x)+\text{Exec}(G_x)+\\ &\qquad\text{Wait}(G_x)-\text{Exec}(K_C)-\text{Exec}(K_G),\\ &K=\{x_i\mid\langle x_i,x_j\rangle, x_i\in D(x), x_j\in D(x)\},\\ &K_C=\{x_i\mid x_i\in C_x\cap x_i\in K\},\\ &K_G=\{x_i\mid x_i\in G_x\cap x_i\in K\}.\end{aligned}\right\} \tag{9}$$

2) 依次将任务(除任务依赖关系图的起点和终点)从其所属集合假定移动到另外一个集合,即 $C_x\to G_x$ 或者 $G_x\to C_x$,重新计算 x 的执行时间 Manage(x). 若小于原时间,则记录该时间和产生该时间的假定移动任务. 当所有任务都假定移动一次后,得到一个能够使 Manage(x)变短的假定移动任务集合.

3) 从假定集合中选取一组没有内部依赖关系的假定移动任务,移动这组任务,更新任务依赖图.

4) 进行下一轮的假定移动,该轮假定移动不再考虑已经移动的任务. 直到假定移动任务集合为空,本程序的任务移动结束.

5) 将任务集合 C_x 和 G_x 分别分配到处理器 p_1 和 p_2 上执行.

异构系统中,存在有依赖关系任务的同步和无依赖关系任务的异步执行方式. 本文综合考虑了任务的内部和外部依赖关系,并根据依赖关系拆分依据、图 2 和表 1,计算任务的等待时间,等待时间包括了同步的影响. 式(9)使用Exec(K_C)、Exec(K_G)消除了任务异步执行方式对时间的影响.

筛选假定移动任务集合所针对的是其中一个被 SVM 分类后的集合,因此首轮筛选的平均任务数量为总任务数量的一半. 若实际移动集合为筛选集合,则以后每轮的筛选和移动任务数量均大幅度降低.

任务的分配算法(最小化程序执行时间)如下.

define CG and GC as set for task may move from C_x to G_x and from G_x to C_x; define M_x as a set for task moved from G_x to C_x or C_x to G_x;

compute Exec(C_x), Exec(G_x), Wait(C_x), Wait(G_x), Exec(K_C), Exec(K_G), Manage(x).

while(true)

{for(int $i=1$; $i \leqslant |x|$; $i++$)

if(x_i is not the original or the destination and x_i is not belong to M_x)

{if($x_i \in C_x$) {$x_i \rightarrow$CG; new$C_x = C_x -$ CG; new$G_x = G_x +$ CG;}

else

{$x_i \rightarrow$GC; new$C_x = C_x +$ GC; new$G_x = G_x -$ GC;}

compute Exec(newC_x), Exec(K_C), Exec(newG_x), Wait(newC_x), Wait(newG_x), Exec(K_G), newManage(x).

if(newManage(x) < Manage(x)) small[$j++$] = newManage(x);

else

delete x_i from CG or GC;}

if(small is not empty)

{find a task set {x_j} dose not have independency with others in newManage(x) in CG and GC;

if({x_j} $\in$ CG)

move {x_j} from C_x to G_x;

else

move {x_j} from G_x to C_x; put {x_j} into M_x; refresh $D(x)$;

compute Exec(C_x), Exec(G_x), Wait(C_x), Wait(G_x), Exec(K_C), Exec(K_G), Manage(x).}

else break;}

3　实验结果与分析

3.1　实验环境

主机处理器 Intel Pentium(R) Dual-Core CPU,3.5 GB **内存**. **设备** NVIDIA GeForce GT 430 **显卡**, GPU **包含** 2 **个流多处理器**(SM)**和** 96 **个处理核心**(SP). **使用** PCI_E2.0×16 **进行数据交换**. CUDA **开发驱动和工具包**: driver version 263.06, CUDA3.2 Toolkits.

3.2　实验结果与分析

选取并开发了 10 **个微基准程序作为预处理模型的测试用例**. **应用程序** ExampApp1 **在预处理过程被** SVM **分成** 24 **个任务**. **图** 3 **描述了分类结果,在预处理过程中除任务** Task17 **分类错误外,** SVM **对其他** 23 **个任务的分类都是正确的,其分类准确率达到** 95.83%. **其他程序的分类准确率均高于** 90%.

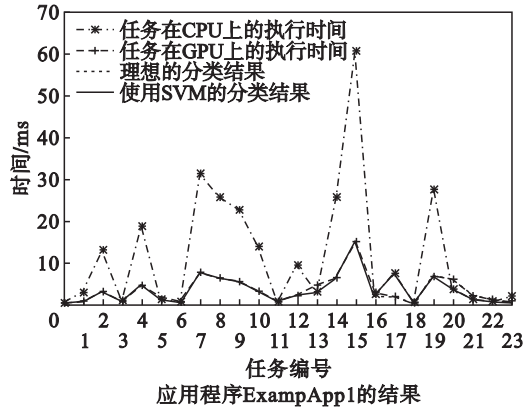

图 3　SVM **对任务的分类结果**

Fig. 3　Classification result of tasks by SVM

实验采用了 10 **个不同类型应用程序作为测试分配模型的用例**. **经本模型处理的不同类型的程序性能均有不同程度的改善**. **图** 4 **所示为** 4 **个测试用例**. **图** 4a **中经过** 6 **轮调整后的** GPU **的执行时间增量比** CPU **缩短的执行时间略长,因为移动的任务本不适合在** GPU **上执行**. **而由于任务的移动将其存在的大量外部依赖转化为内部依赖,因此总的等待数据时间大幅缩减**. **综合两部分时间的改变,程序执行时间下降了** 40.63%. **图** 4b ~ **图** 4d **经调整后均获得了不同程度的性能提升**. **图** 4c **与图** 4a **的计算量相当,但图** 4c **仅经** 2 **轮调整后即获得最短的执行时间,这是因为其任务间存在较少的内部数据依赖关系,因此,即使程序分解为大量任务,本算法在计算移动任务集时会依据任务的数据依赖关系. 每轮转移的是一批任务,从而使下一轮计算量大幅缩减**.

图 5 **中,与按任务量比例和按处理器计算能力的分配算法相比,本模型性能分别提升** 50.46% **和** 38.61%. **两种算法均未考虑任务之间数据依赖关系导致的等待数据时间过长**. **按任务量比例进行分配的算法没有考虑任务特征和处理器的特征,使得** CPU **和** GPU **上分配的任务不均衡,从而影响执行时间**. **本模型使用** SVM **进行任务分类并考虑了数据依赖的影响,使等待数据时间减少,从而使总执行时间大幅缩短**.

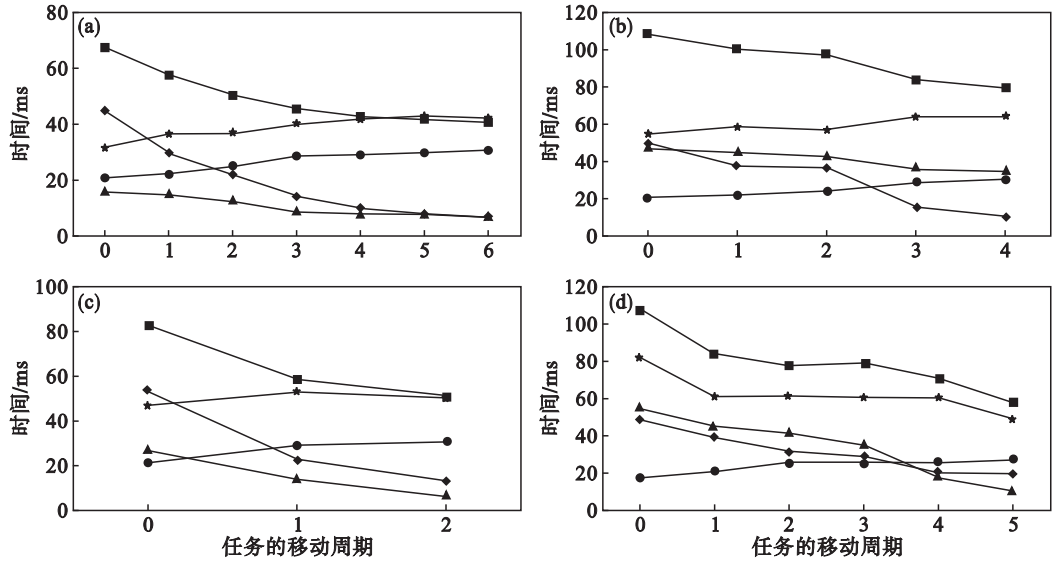

图 4 分配模型的性能

Fig. 4 Performance of the allocation model

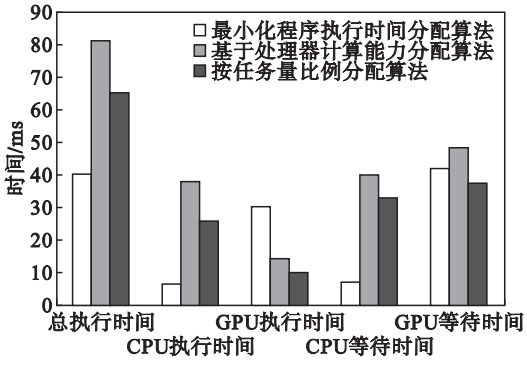

图 5 几种分配算法的性能比较

Fig. 5 Performance comparison of different allocation algorithms

4 结 论

本文提出了一个基于 SVM 的任务分配模型,该模型综合考虑任务特征和处理器特征及当前运行状态,预测了任务的最优分配方案. 在分配前进行多轮分配集合的调整,使系统性能得到显著提升. 对开发的程序和 CUDA SDK 例程的研究分析表明,本文的分配方法实用,有效提高了应用程序的性能. 与其他算法相比,性能平均提升了 43. 54% ,而且模型的分配开销小,对于任何类型的应用程序均能获得最优的分配方案. 下一步,准备将模型改进成适用于多台设备和多台主机的异构系统,使其更具普遍性.

参考文献:

[1] Shen J, Varbanescu A L, Zou P, et al. Improving performance by matching imbalanced workloads with heterogeneous platforms[C]//Proceedings of the 28th ACM International Conference on Supercomputing. Müenchen, 2014: 241 – 250.

[2] Luk C K, Hong S, Kim H. Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping[C]// Proceedings of 42nd Annual IEEE/ACM International Symposium on Microarchitecture. New York: IEEE, 2009: 45 – 55.

[3] Ma K, Li X, Chen W, et al . GreenGPU: a holistic approach to energy efficiency in GPU-CPU heterogeneous architectures [C] // Proceedings of 41st International Conference on Parallel Processing. Pittsburgh: IEEE Computer Society, 2012: 48 – 57.

[4] Yang C, Wang F, Du Y, et al. Adaptive optimization for petascale heterogeneous CPU/GPU computing [C] // Proceedings of 2010 IEEE International Conference on Cluster Computing. Heraklion, 2010: 19 – 28.

[5] Grewe D, Boyle M. A static task partitioning approach for heterogeneous systems using OpenCL[C] // Proceedings of 20th International Conference on Compiler Construction. Saarbrücken, 2011: 286 – 305.

[6] Zhou H, Liu C. Task mapping in heterogeneous embedded systems for fast completion time[C] //Proceedings of 2014 International Conference on Embedded Software (EMSOFT). New Delhi: ACM, 2014: 1 – 10.

[7] Shirahata K, Sato H, Matsuoka S. Hybrid map task scheduling for GPU-based heterogeneous clusters [C] // Cloud Computing Technology and Science. Bloomington, 2010: 733 – 740.

[8] Li C H, Kuo B C, Lin C T, et al. A spatial contextual support vector machine for remotely sensed image classification[J]. *IEEE Transactions on Geoscience and Remote Sensing*, 2012, 50(3): 784 – 799.