一种用于异构环境中任务调度的高效算法①

刘侃侃　(杭州电子科技大学 计算机学院 浙江 杭州 310018)

摘　要: 在异构计算环境中，有效的任务调度对于获得高性能是十分重要的。现在虽然已经有许多异构处理器调度算法,但它们或者不具有良好的效果,或者算法代价太高。提出了一种新的基于表的调度算法 APS。APS 利用有向无环图来计算任务优先级，并采用基于调度的策略分配任务到不同处理器，以获得任务最少完工时间。将 APS 和 LMT，HEFT，CPOP 算法做比较之后得出：在大多数情况下 APS 算法都能获得更好性能。

关键词: 任务调度; 并行计算; DAG; 异构系统

High Performance Algorithm for Task Scheduling in Heterogeneous Environment

LIU Kan-Kan

(Hangzhou Dianzi University, Hangzhou 310018, China)

Abstract: Efficient task scheduling is critical for obtaining high performance in a heterogeneous computing environment. Although there are many scheduling algorithms already, they may not have good results, and take high cost. In this paper, a new list scheduling heuristics, named Accurate Priority Scheduling (APS), is presented. The APS selects task with random directed acyclic graph (DAG) and assigns tasks to processors with the insertion based scheduling policy to minimize the makespan. Compared with LMT, HEFT, CPOP based on, show the APS results good performances in most situations.

Keywords: task scheduling; parallel processing; DAG; heterogeneous systems

众所周知，异构环境下的任务调度是一个 NP 问题[1-4]。目前在众多的调度算法中，基于表的静态调度算法，由于其低复杂性与性能优异，具有很强的吸引力。但是在涉及到具体做法时，现有的基于表的算法[1-9]多少都存在着一些不足。一些算法总是把任务分配到最合适该任务的处理器，如 DPS[2]。这种方法对某个任务本身来说确实达到了最优，但对于整个调度来说确不一定能达到最佳。一些算法把所有任务分成不同的层次，同一层的任务不含有依赖关系，如 DLS[5]。这类算法没有考虑所有任务之间的优先级，不一定能获得比较好的结果。还有一些算法在有限处理器条件下能获得比较好的结果，如 HEFT,CPOP[6]，但在一些简单条件下并不能获得很好的结果[7]。

在本文中，我们提出了一种新的以表为基础的算法 APS，用来尽量减少最早结束时间。APS 主要有两个主要阶段：任务选择和处理器的选择。在第一阶段，APS 计算每个任务的优先级，并在下一个阶段把优先级最高的任务分配给最合适的处理器。第二步采用的是基于插入的原则。

1　问题定义

在一个静态的算法中，一个并行应用被分解成若干子任务，子任务之间的依赖关系在编译的时候已经得知。这种应用被称作 DAG(图 1.a)。

为了更好的理解文章的内容，我们把一些文中涉及的概念及它的含义总结如表 1 所示：

DAG 被定义为 G=(V, E)，边 $e_{(i,j)}$代表子任务 v_i 必须在 v_j 之前，其中是 v_i 父亲，v_j 是孩子。孩子不可能在父亲之前被执行。没有父亲的任务被称作入口任务，同样的，没有孩子的任务是出口任务。

① 收稿时间:2010-03-04;收到修改稿时间:2010-03-16

表 1 符号及其表示的含义

符号	含义	符号	含义
V	任务集合	$startup_m$	p_m 的启动代价
E	任务之间关系的集合	$data(i, j)$	v_i 转换到 v_j 的数据量大小
P	核的集合	$rate(i, j)$	v_i 转换到 v_j 的数据传输率
$e = \|E\|$	关系的数目	$w(v_i, p_j)$	v_i 在 p_j 上的执行代价
$p = \|P\|$	核的数目	$pred(v_i)$	v_i 的直接前驱任务
v_i	第 i 个任务	$succ(v_i)$	v_i 的直接后继任务
p_j	第 j 个核	SRank	任务调度优先级
$e(i, j)$	v_i 和 v_j 之间的关系	BRank	自底向上的任务调度优先级
$c(i, j)$	v_i 转换到 v_j 的代价	DiffRank	任务在不同处理器上执行代价的差值

$$c(i, j) = startup_m + data(i, j) * rate(m, n) \quad (1)$$

如果 v_i 和 v_j 执行在同一个核上，即 m 和 n 代表同一个核，则 $c_{(i, j)}$ 等于 0。

在一个 DAG 图中，只能有一个入口任务和一个出口任务。假设现在有不止一个入口任务和出口任务时，可以将这些任务链接到一个虚拟任务上，并将代价设为 0。这不会影响整个 DAG 图(如图 1.b)。

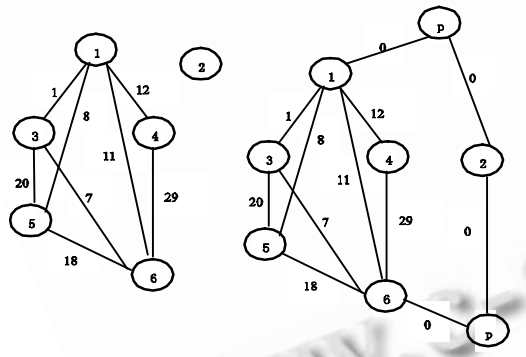

图 1 DAG 图实例

为了计算没有分配的任务的优先级，平均执行代价定义如下：

$$\overline{w_i} = \sum_{j=1}^{p} w(i, j) / p \quad (2)$$

在本文的实验环境中，我们假设如下：(1)所有的核完全拓扑相连，即任何一个核都可以和其他核直接相连；(2)当一个任务被启动的时候，它不能在完成之前被中止；(3)当一个任务结束，它可以立即并行的把数据传输给它的所有孩子；(4)每个核都可以同时进行数据传输和任务执行。

2 APS算法

在前面提出的一些概念的基础上，这一节将介绍 APS 算法。

2.1 APS 算法属性

计算任务优先级前，SRank，BRank 和 DiffRank 将被计算。BRank 定义如下：

$$BRank(v_i) = \max_{v_j \in succ(v_i)} \{BRank(v_j) + \max_{k=1}^{|P|}(w(v_j, p_k)) + c(j, i)\} \quad (3)$$

相似地，假设前提任务 v_j 被分配到核 p_k，则 SRank 定义如下：

$$SRank(v_i) = \max_{A(v_i, p_k)} \{w(v_j, p_k) + c(j, i)\} \quad (4)$$

SRank 将从入口任务开始计算，且入口任务的值等于 0。与 BRank 不同的是，SRank 的计算采用了它的前提任务的真实的任务执行代价值。

DiffRank 代表一个任务在不同核上执行代价最大值与最小值之间的差，定义如下：

$$DiffRank = \max_{k=1}^{|P|}(w(v_i, p_k)) - \min_{k=1}^{|P|}(w(v_i, p_k)) \quad (5)$$

任务优先级的定义则是通过将这 3 个 Rank 相加而得到的。在大多数算法中，任务优先级的计算都是通过估计值来获得，但 APS 算法将优先级分成 3 个部分，其中的 SRank 部分是通过真实值来获得，有更高的准确性。

最后，如果前提任务 pred(vi)被分配到了 pk，我们可以定义最早完成时间 EFT 如下：

$$EFT(v_i) = \min_{1 \le j \le p}\{\max(EFT(pred(v_i)) + c(A(pred(v_i), p_k), A(v_i, p_j)), PA) + \overline{w_i}\} \quad (6)$$

对于所有的 k=j 来说，$c(A(pred(v_i),p_k),A(v_i,p_j))$ 等于 0，否则它等于 $data(pred(v_i),v_i)$。

2.2 APS

APS 算法主要用来处理有界的异构多核处理器的情况。主要过程分为两部：任务选择和处理器选择。

第一个部分中，APS 计算各个任务的优先级，并在下一个部分中把最高优先级的那个任务分配给最合适的处理器获得最小的完成时间。

2.2.1 任务选择

这个部分中，BRank 通过平均的执行代价来计算(step1)，SRank 的值在任务插入到就绪队列之后被计算(step12，13)。就绪队列中存放的是目前所有可以被调度的任务，这些任务所有的父亲，或所有的前提任务都已经被调度了。然后 APS 把 SRank，Brank，DiffRank 相加作为就绪任务的优先级。最后选择优先级最高的那个任务(step5)。如果有两个或两个以上的就绪任务的优先级的值相同，则选择具有较小 SRank 值的任务；如果 SRank 的值依然相同，则随机抽取一个任务。

2.2.2 处理器选择

在这个部分中，被选择的任务将被分配到最合适的处理器，使得最早任务完成时间最小(step6-10)。如果某个任务分配到几个处理器所得到的完成时间相同，则选择代价最小的那个核进行分配。我们的算法采用了基于插入的原则，考虑任务尽早执行[8]。这就有两个约束要遵守：第一个是空闲的时间空间必须可以容纳下该任务的执行，第二是该任务不可以比它的前提任务更早的执行。APS 算法伪代码如表 2：

表 2　APS 算法伪码

1.	计算给出的 DAG 图中所有任务的 Brank 值;
2.	建立一个就绪任务队列，并初始化为入口任务;
3.	while (就绪任务队列不为空) do
4.	选择就绪任务队列中优先级最高的任务 v_i;
5.	for ($p_k \in P$) do
6.	采用基于插入的原则来计算 EFT(v_i, p_k);
7.	end for
8.	分配 v_i 到 p_j，使得 EFT 的值最小;
9.	从就绪任务队列中删除 v_i;
10.	更新就绪任务队列，计算新加入的任务的 Srank 值;
11.	对每一个就绪任务计算它的优先级(SRank+ BRank+ DiffRank);
12.	end while

2.3 APS 实例

这一节中，给出了执行代价矩阵如表 3，并假设有 3 个处理器。采用的 DAG 图如图 1 所示。

表 3　执行代价矩阵

任务	P1	P2	P3
1	1	2	3
2	39	42	39
3	2	4	3
4	26	20	24
5	17	28	14
6	39	38	32

利用 APS 算法，任务执行顺序如下{v_1, v_5, v_3, v_4, v_6, v_2}，SRank，Brank，DiffRank，和优先级如表 4 所示：

表 4　APS 优先级

任务	BRank	SRank	DiffRank	Priority
V_1	123	0	2	125
V_2	42	0	3	45
V_3	113	8	2	123
v_4	94	19	6	119
v_5	85	26	14	125
v_6	39	49	7	95

由 APS 生成的最后结果如图 2 所示，调度长度是 86 个时间单元，相对而言，HEFT 则要 100 个时间单元。就算法复杂性而言，两个算法复杂性都等于 $O(e\times p)$。

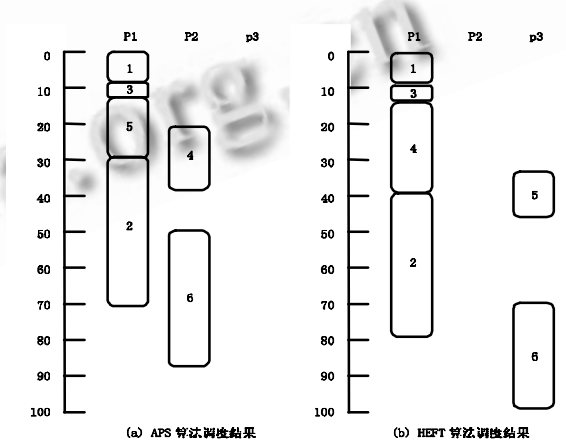

图 2　APS 和 HEFT 算法调度结果比较

3　实验结果和讨论

这一部分中，我们把 APS 和 HEFT，CPOP，LMT 进行比较。首先我们介绍几个比较的准则，然后将介绍随机 DAG 图生成的原则，最后一部分将显示比较结果并做一些讨论。

3.1 比较原则

以下将介绍几个评估算法优劣的原则：

(1) 调度长度比(SLR)。由于大量不同属性的应用图被使用，很有必要使调度长度趋向于一个较小的边界，我们把这种就叫做 SLR[8,9]。SLR 定义如下：

$$SLR = \frac{makespan}{\sum_{v_i \in CP_{\min}} \min_{p_j \in P}\{w(i,j)\}} \quad (7)$$

CPmin 表示 DAG 中为调度部分的重要路径，它等于不同任务执行代价的最小值。

(2) Speedup。一个调度的 Speedup 定义为所有任务的最小调度长度与对同一个处理器来说最小代价值的比值，定义如下：

$$Speedup = \frac{\min_{p_j \in P}\{\sum_{v_i \in V} w(i,j)\}}{makespan} \quad (8)$$

3.2 随机 DAG 生成

在我们的试验中，随机 DAG 生成时通过一些输入参数来决定 DAG 图的属性，这些属性及实验中的约束值如下：

① 任务数量(|V|∈{10,20,30,40,50,60})

② 核数量(|P|∈{3,4,5,6})

③ 各节点的出度(SETout_degree={0,1,2,3,4,5})

④ 联系与代价比率(CCR)[8，9] (SETCCR={0.1, 0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0})

⑤ 不同处理器平均执行的代价范围(β)[8,9](β ∈ {0.1,0.25,0.5,0.75,1.0})

3.3 实验结果

在试验中，对于每组不同的任务数和核数，将会生成 100 个随机的不同 DAG 图。结果如图 3。

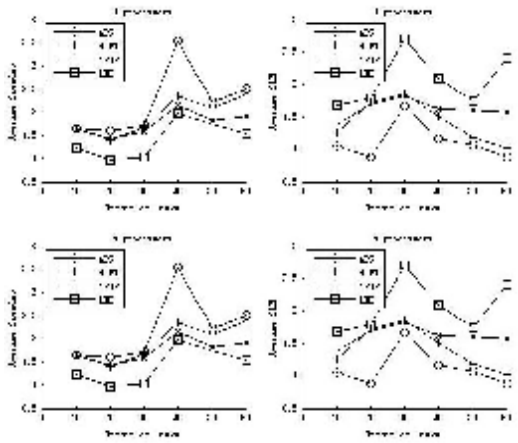

图 3 APS, HEFT, CPOP, LMT 算法比较结果

在这些图中，左边的都是不同数目的核的平均 SLR，右边是平均的 Speedup。在大多数情况下，APS 都要比其他算法表现更好。大概的说，APS 的 SLR 比 HEFT 的好 5%，比 CPOP 的好 11%，比 LMT 的好 36%。而 APS 得 Speedup 超过 HEFT 的 6%，CPOP 的 8%，LMT 的 20%左右。

4 结论

本论文在异构计算环境下提出了一种 APS 算法，它主要分为两个步骤：任务选择和处理器选择。在第一部分中，APS 计算每个任务的优先级，并在第二个步骤中把优先级最高的任务分配给最合适的处理器，来获得最少完成时间。在基于随机生成的图的基础上，我们把 APS 与 HEFT，CPOP，LMT 算法进行了比较，实验结果显示在大多数情况下，APS 都有较好的结果。今后，还可以将 APS 扩展到具有无限个处理器的任务调度上。

参考文献

1 Sanjeev B, Kiran Kumar P. Low Power Scheduling of DAGs to Minimize Finish Times. Bangalore: High Performance Computing, 2006:353—362.

2 Ahmad I, Dhodhi MK, U Mustafa R. DPS: dynamic priority scheduling heuristic for heterogeneous computing systems. Computers and Digital Techniques, 1998,145(6):411—418.

3 Hagras T, Janecek J. A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems. International Parallel and Distributed Processing Symposium, 2005,31(7):653—670.

4 Mohammad ID, Nawwaf K. A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing, 2008,68(4):399—409.

5 SIH GC, LEE EA. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Trans. Parallel and Distributed Systems, 1993,4(2):175—187.

6 Haluk T, Salim H, Minyou W. Performance-Effective

(下转第 97 页)

(上接第 105 页)

and Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE Trans Parallel Distributed Systems, 2002,13(3):260—274.

7 Vincent boudet. Heterogeneous task scheduling: a survey.http://lara.inist.fr/handle/2332/752. 2008.12.25.

8 Lee W, Siegel HJ, Vwani P Roychowdhury, Anthony A Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. Journal of Parallel and Distributed Computing, 1997,47(1):8—22.

9 Deepa R, Srinivasan T, Doreen Hephzibah Miriam D. An Efficient Task Scheduling Technique in Heterogeneous Systems Using Self-Adaptive Selection-Based Genetic Algorithm. Los Alamitos: Parallel Computing in Electrical Engineering, 2006: 343—348.