ISSN1000-9825 Journal of Software 软件学报 2000,11(4):551~556

异构计算中的负载共享*

曾国荪[1,2] 陆鑫达[1]

[1](上海交通大学计算机科学与工程系 上海 200030)

[2](江西师范大学计算机科学系 南昌 330027)

E-mail: gszeng@263.net

摘要 在基于消息传递的异构并行计算系统中,各处理器或计算机具有自制和独立地调度、执行作业的能力.当一个可划分的作业初始位于一个处理器上时,为了提高计算性能,该处理器可以请求其他异构处理器负载共享,参与协同计算,减少作业的完成时间.该文提出了异构计算负载共享的一种方案.首先,调用负载共享协议,收集当前各处理器参与负载共享的许可数据,包括共享时间段、计算能力等.然后,构造一个作业量与作业完成时间之间的关系函数.该函数是选择一组合适的处理器群、优化作业划分、作业完成时间最小的理论基础.最后,给出了优化负载划分和调度的有效算法以及应用实验实例.该算法在启动作业执行时刻方面具有一定的实时性,并且可扩展到多个初始作业位于多个处理器上的情况.

关键词 负载共享,异构计算,作业划分,调度,加速比.

中图法分类号 TP302

异构计算是近年来并行处理的研究热点.性能各异的计算机通过高速网络连成一体,开辟了一个超级计算的新领域.异构计算的目的就是匹配一组计算类型不同的子任务到一组合适的计算机体系结构上来执行,减少任务的完成时间[1].关于异构计算环境中的负载共享问题,最近也发现了一些研究文献[2~5],它们多半是同构计算基础上的扩展,属于静态负载划分和调度.这方面的理论和方法尚不完善,还处于探索阶段.本文给出了异构计算中负载共享的一种方案,用计算速度刻画系统的异构性,并且假设负载是可任意划分的,同时考虑了系统静态因素和实时因素[6],是一种有效的方案.

1 假设条件

条件 1. 我们假设作业负载模型是可任意划分的,并且将调度到异构多处理器上执行.在早期的并行分布计算中,作业负载模型一般假设为正在处理的作业,它由一组不可划分的任务组成,因此,任务在处理机上的调度只能以原子模块为单位.这意味着只有有限种方式分配给定任务到给定处理机上执行.但是,原子任务模型并不能真正反映现实世界任务的不可分解性.事实上,任何可计算的任务在指令级上都是可任意划分的.

现设有可划分的作业负载 J,它的总计算量记为 λ,令 $\lambda=\lambda_1+\lambda_2+\ldots+\lambda_n$ 表示它的一个划分,λ_i 的取值依赖于划分机制.如果任务代码具有异构性,一般取为各任务代码类型对应的计算量.通常的做法是,根据各处理器的计算能力、通信带宽、当前状态等因素来决定.本文后面将介绍一种优化选择 λ_i 的方法.

条件 2. 多处理器的异构性用执行速度 s_i 来抽象,它是处理器当前各种计算因素的综合,也就是 s_i 为 $\alpha,\beta,\gamma,\varphi,\delta$ 的函数,即 $s_i=\pi(\alpha,\beta,\gamma,\varphi,\delta)$,其中 $\alpha,\beta,\gamma,\varphi,\delta$ 分别表示处理器的时钟、单位时间执行指令的条数、任务代码类型、通信量和当前负载状态.定量推导该函数关系表达式超出了本文的讨论范围,参见文献[7].任务代码类型 γ 对 s_i 影响很大,$\forall\ \gamma_1,\gamma_2$,且 $\gamma_1\neq\gamma_2$,则 $s(\gamma_1)\neq s(\gamma_2)$.例如,向量计算语句在向量机上执行可得到很好的加速

* 本文研究得到国家自然科学基金(No.69773014)资助.作者曾国荪,1964年生,博士生,副教授,主要研究领域为并行处理,异构计算.陆鑫达,1938年生,教授,博士生导师,主要研究领域为并行处理,系统结构,指令级优化编译.

本文通讯联系人:曾国荪,上海 200030,上海交通大学计算机科学与工程系

本文 1998-10-26 收到原稿,1999-04-14 收到修改稿

比,反之,则效率很低[5].有时,如果希望简化 s_i 的表示,可取 $s_i \approx \beta_i/\alpha_i$.

条件 3. 作业负载初始位于一个处理器上,由它请求其他处理器参加计算,以便减少作业的完成时间,提高计算性能.

2 问题的公式描述

令 $P=\{p_1,p_2,\ldots,p_n\}$ 表示异构系统中 n 台处理器或计算机,其中 p_j 是作业初始位于的处理器,s_j 是其计算速度,作业总计算量为 λ,作业假设在 $t=t_0$ 时刻启动执行.如果整个作业由 p_j 单独执行,则完成时间为 $\tau=t_0+\lambda/s_j$.如果 p_j 用负载共享协议 LSP,请求其他处理器,在 $t>t_0$ 的某段时间,参与计算,分治作业,那么,完成时间肯定要短.对于每个处理器 p_i,$p_i\neq p_j$,应答请求有两种情况:当应答为负响应时,表明该处理器不能参与负载共享;当应答为正响应时,同时指出能够和 p_j 共享负载、分治作业的时间段,计算能力 s_i 等许可数据.我们称第 2 种情况为"许可处理器".

令 $A=\{i\in\{1,2,\ldots,n\}\mid p_i$ 为许可处理器$\}$,令 T_i 为各许可处理器 p_i 指出的和 p_j 共享负载的时间段集,则

$$T_i=\bigcup_{k=1}^{Ni}[a_{ik},b_{ik}]=\{[a_{i1},b_{i1}],\ldots,[a_{iNi},b_{iNi}]\},\quad a_{ik}\geqslant t_0,$$

其中 Ni 是许可处理器的总数.

定义 $S=\{i\in A\mid p_i$ 为被选中参与负载共享$\}$,显然 $S\subset A$.这样,向不被选择的处理器 $p_i(i\in(A-S))$ 发送一个负保留时间消息,拒绝它们参与负载共享.对于选中的处理器 p_i,$i\in S$,应该通知它们参与负载共享的时间段,记为 R_i,$R_i\subseteq T_i$,对应的计算量为 $\lambda_i=s_i|R_i|$.

假设每个处理器 $p_i(i\in S)$ 的完成时间为 t_i,那么整个作业的完成时间 $\tau=\max_{i\in S}\{t_i\}$.我们的目标是:初始处理器 p_j,根据许可数据,如何优化选择处理器集 S,如何优化划分负载,即 $\lambda=\sum_{i\in S}\lambda_i$,记 $\Lambda=\{\lambda_i,i\in S\}$,使得 τ 最小.表示成

$$t^*=\min_{S,\Lambda}\tau(S,\Lambda)=\min_{S,\Lambda}(\max_{i\in S}\{t_i\})-\tau(S^*,\Lambda^*).$$

我们构造一个函数 $x(t)$,用它表示给定的总负载量 λ 和最小作业完成时间 t^* 的关系,即 $\lambda=x(t^*)$,通过许可数据,函数 $x(t)$,当 $t\geqslant t_0$,完整定义如下:

设 $c_m=\{a_{ik}\}\cup\{b_{ik}\}=\{c_1,c_2,\ldots\}$,不妨设 $c_{m+1}>c_m$,则

$$x(t)=r_m(t-c_m)+x_m,\quad t\in[c_m,c_{m+1}]. \tag{1}$$

其中 r_m,x_m 是常数,其值依赖 m,可由下面递归式求得.

$$r_m=\begin{cases}r_m+s_i & \text{当 } c_m=a_{ik}\\ r_m-s_i & \text{当 } c_m=b_{ik}\end{cases},\quad \text{特别地},r_1=s_j,$$

$$x_{m+1}=x_m+r_m(c_{m+1}-c_m),\quad \text{特别地},x_1=0.$$

显然,$x(t)$ 是连续的,并且分段线性,单调递增,拐点为 $\{(c_m,x_m)\}$,$m=1,2,3,\ldots$,将 $x(t)$ 绘成图,则以上性质不难验证.

现在,当给定总负载 λ,要求最小完成时间 t^*.由于 $\lambda=x(t)$ 单调连续,处处有定义,所以 t^* 必然存在.通过搜索拐点,将找到 $m=m^*$,使得 $x_{m^*}\leqslant\lambda\leqslant x_{m^*+1}$,那么 $t^*\in[c_{m^*},c_{m^*+1}]$,所以

$$\begin{aligned}\lambda&=r_{m^*}t(^*-c_{m^*})+x_{m^*},\\ t^*&=c_{m^*}+(\lambda-x_{m^*})/r_{m^*}.\end{aligned} \tag{2}$$

3 负载优化划分和调度算法

3.1 具体算法

沿用第 1 节和第 2 节中的假设和定义,下面的算法可优化作业的完成时间.

算法 1. OPT_LOAD_SHARE

(1) 轮询系统中的各处理器,得到许可处理器集 A,并获得作业启动执行时刻 t_0 之后负载共享时间段集 $T_i=\bigcup_{k=1}^{Ni}[a_{ik},b_{ik}]$.

(2) 用 a_{ik},b_{ik},s_i 等数据构造函数 $x(t)$.

(3) 由给定的总负载幅值 λ,按公式(2),算出 $t^*=x^{-1}(\lambda)$.

(4) 确定处理器选中集 S^*,它要求满足 $S^*=\{i\in A\mid a_{ik}<t^*\}$.

(5) 确定每个选中的处理器 $p_i,i\in S^*$,共享负载保留时间段 R_i,它要求满足 $R_i=T_i\cap(t_0,t^*),i\in S^*$.

(6) 确定每个选中的处理器 $p_i,i\in S^*$,从总负载中划分、分派计算量 λ_i^*,它要求满足 $\lambda_i^*=s_i|R_i|,i\in S^*$.

(7) 通知每个选中的处理器,保留执行时间段 R_i,通知非选择的处理器不参加负载共享.

(8) 初始处理器 p_j 分派共享计算量 λ_i^* 给每个选择的处理器 $p_i,i\in S^*$.

3.2 算法性质

定理 1. 算法 OPT_LOAD_SHARE 产生的负载优化划分之和等于负载总幅值,即 $\lambda=\sum_{i\in S^*}\lambda_i^*$.

证明:代入求和可得到验证. □

定理 2. 算法 OPT_LOAD_SHARE 产生最小作业完成时间.

证明:证明过程主要是验证算法中第 3 步的结论,即验证公式 $\lambda=x(t^*)$. 因为初始处理器 p_j,从 $t=t_0$ 启动执行以后,当 $t\in T_i$,以速度 s_i 连续分派负载到处理器 p_i;当 $t\notin T_i$,则不分派负载. 此过程一直进行到整个负载 λ 被执行完成. 接收处理器 p_i,在 $t\in T_i$ 时,以速度 s_i,精确执行被分派的负载量,但不在其他时间执行其他负载. 显然,负载分派是贪婪调度的,没有其他调度能产生更早的完成时间. 总负载幅值 λ 和最小完成时间 t^*,是通过一组被选中的处理器在时间段 $[t_0,t^*]$ 共同作用的结果. 所以,被选中的处理器集应为 $S^*=\{i\in A\mid a_{ik}<t^*\}$,按定理 1 可知,$\lambda=\sum_{i\in S^*}|T_i\cap(t_0,t^*)|s_i$,令 $x_i(t^*)=|T_i\cap(t_0,t^*)|s_i$,那么

$$\lambda=\sum_{i\in S^*}x(t^*). \tag{3}$$

下面用给定的参数 a_{ik},b_{ik} 来表达 $x_i(t^*)$,且记 $r_i(t^*)$ 为满足下面关系的最大下标 $r_i(t^*)=\max\{k\mid a_{ik}<t^*\}$,且 $t^*>a_{i1}$,因此,

$$\begin{aligned}x_i(t^*)&=\sum_{k=1}^{r_i(t^*)-1}s_i(b_{ik}-a_{ik})-s_i(t^*-a_{ir_i(t^*)}), && \text{当 } t^*\in T_i,\\&=\sum_{k=1}^{r_i(t^*)}s_i(b_{ik}-a_{ik}), && \text{当 } t^*\notin T_i,t^*>a_{i1},\\&=0, && \text{当 } t_0<t^*<a_{i1},\end{aligned}$$

参照公式(1)和(2)中关于函数 $x(t)$ 的定义,展开递归关系式,可知公式(3)定义的 λ 和 $x(t^*)$ 是同一的,因此,定理得证. □

3.3 性能分析

由定理 2 可知,在给定共享时间段和计算速度的许可数据条件下,算法 OPT_LOAD_SHARE 产生作业最小完成时间 $t^*=c_{m^*}+(\lambda-x_{m^*})/r_{m^*}$. 对 t^* 的进一步讨论,可以了解该算法对计算性能的改善程度. 假设定义加速比 S_p 为初始处理器单独执行作业 J 的完成时间和作业 J 共享执行完成时间之比,即

$$S_p=\frac{\lambda/s_1}{t^*}=\frac{\lambda/s_1}{c_{m^*}+(\lambda-x_{m^*})/r_{m^*}},$$

其中 c_{m^*} 是某一共享时间段的左边界,r_{m^*} 是当 $t\in[c_{m^*},c_{m^*+1}]$ 时,所有共享处理器计算速度之和. 在下面几种特殊情况下,可直接算出 S_p 的值.

(1) $c_{m^*}=0$,即系统中所有处理器和初始处理器 p_1 一起,同时参与负载共享,则 $x_{m^*}=0,r_{m^*}=s_1+s_2+\cdots+s_n=\sum_{i=1}^{n}s_i$,$S_p=\dfrac{\lambda/s_1}{\lambda/\sum_{i=1}^{n}s_i}=\left(\sum_{i=1}^{n}s_i\right)/s_1\gg 1$. 这是实际应用时最希望达到的情况.

(2) $c_{m^*} \geqslant \lambda/s_1$,即作业在初始处理器 p_1 上完成之后,其他处理器才能参与负载共享,于是 $x_{m^*}=\lambda, S_p=1$. 负载共享算法没有实际意义,只能作为避免盲目、随意负载共享的一种指导.

(3) 一般情况下,$1 \leqslant S_p \leqslant \left(\sum_{i=1}^{n} s_i\right)/s_1$. 计算性能有一定的改善.

4　应用和实验实例

4.1　应用举例

我们用 5 个处理器的系统 $\{p_1,p_2,p_3,p_4,p_5\}$ 来演示算法的应用. 设 p_1 为初始处理器,作业幅值 $\lambda=25$,初始启动执行时间 $t_0=0$.

第 1 步. 轮询系统各处理器参与负载共享,之后,p_1 得到其他处理器提供的共享时间段、计算速度等许可数据. 假设为 $T_1=[0,\infty]$, $T_2=[2,6]$, $T_3=[9,11]$, $T_4=[4,8]$, $T_5=\{[12,16],[18,\infty]\}$, $\{s_1,s_2,s_3,s_4,s_5\}=\{1,2,3,1,1\}$.

第 2,3 步. 因为 $\lambda=25$,应用第 1 步的数据,可得到最小完成时间 $t^* \in [c_{m^*},c_{m^*+1}]=(9,11)$,所以 $t^*=10$.

第 4 步. 选择参与负载共享的处理器集 $S^*=\{i \in A \mid a_{ik}<t^*\}=\{i \in A \mid a_{i1}<11\}=\{1,2,3,4\}$, p_5 不参与负载共享.

第 5 步. 各处理器保留的执行时间段为 $R_i=T_i \cap (t_0,t^*)=T_i \cap (0,10)$.

$$R_1=[0,\infty] \cap (0,10)=[0,10), \qquad R_2=[2,6] \cap (0,10)=[2,6],$$
$$R_3=[9,11] \cap (0,10)=[9,10), \qquad R_4=[4,8] \cap (0,10)=[4,8].$$

第 6 步. 各处理器分派的负载量为 $\lambda_i^*=s_i|R_i|, i \in S^*$,

$$\lambda_1^*=1\times 10=10, \qquad \lambda_2^*=2\times 4=8, \qquad \lambda_3^*=3\times 1=3, \qquad \lambda_4^*=1\times 4=4.$$

显然 $10+8+3-4=25$,并且参照初始处理器的加速比 $S_p=(\lambda/s_1)/t^*=(25/1)/10=2.5$.

4.2　实验实例

我们的实验室配有多台 Sun 和 SGI 工作站,用 100MB 以太网连成工作站群,运行 PVM3.3 异构计算环境. PVM 提供了动态添加处理器和启动子进程进行并行计算的原语,各处理器以 SPMD 方式工作,不存在进程迁移问题. 我们用 4 台 Ultra1 Sun+1 台 Ultra E450 Sun+1 台 SGI O2 工作站(各种机型相对计算速度为 1,2,1.8),做了矩阵相乘计算的负载共享实验. 下面是几种情况下的实验数据和性能比较.

实验情况 1. 分别用本文负载共享算法和投标算法进行调度. 负载共享算法假设各处理器以机会均等的概率参与共享;投标算法是在启动作业执行时,选用计算速度最大的处理器去执行. 矩阵维数为 600×600,实验概率统计结果如表 1 和图 1 所示.

Table 1　The executing time of 600×600 matrix by bidding and load sharing scheme

表 1　600×600 矩阵计算投标、共享算法执行时间数据

Number of processors used (s)①	1	2	3	4	5	6
Time on uni-processor②	134.8					
Time by bidding scheme③	116.3	112.6	107.1	97.81	79.31	72.01
Time by loading sharing scheme④	134.8	67.45	46.07	35.52	29.43	23.47

①使用处理器数(秒),②单机执行时间,③投标执行时间,④共享执行时间.

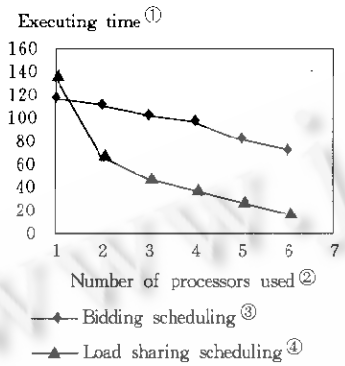
实验情况 2. 分别用本文提出的负载共享算法和负载平衡算法进行调度. 两种调度都假设在启动作业执行时,所有处理器都能参与计算. 实验结果如表 2 和图 2 所示.

Table 2 The executing time and speedup for various size of matrix by balance scheme and load sharing scheme

表 2 各种矩阵维数下平衡法、共享法执行时间、加速比

Size of matrix①	100	200	300	400	500	600	700	800	900	1000
Time on uni-processor②	0.682	4.082	13.41	37.84	75.09	134.8	220.5	355.8	509.5	741.1
Time by balance scheme③	0.481	1.148	3.193	7.762	14.93	25.74	41.05	64.30	89.14	125.4
The by load sharing scheme④	0.302	1.577	3.327	7.207	13.61	23.48	35.91	55.13	75.35	102.1
Speedup of balance scheme⑤	1.415	3.555	4.199	4.875	5.020	5.236	5.371	5.533	5.715	5.909
Speedup of load sharing scheme⑥	1.358	2.588	4.030	5.250	5.517	5.741	6.140	6.453	6.761	7.258

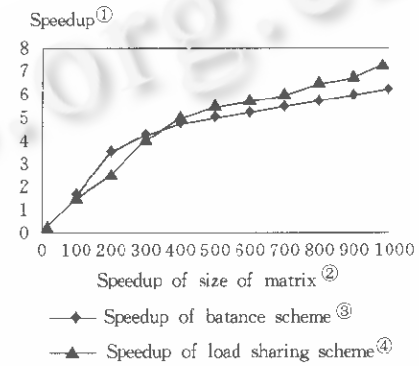
①矩阵维数,②单机执行时间,③平衡法时间,④共享法时间,⑤平衡法加速比,⑥共享法加速比.



①执行时间,②处理器数,③投标调度,④共享调度.

Fig. 1 The curse of executing time versus the number of processors used

图1 执行时间和处理器数曲线



①加速比,②矩阵维数,③平衡法S_p,④共享法S_p

Fig. 2 The curse of speedup versus the size of matrix

图2 加速比和矩阵维数曲线

5 结束语

本文给出了一种可划分负载在异构环境下优化划分和调度的算法.如果求解问题包括大量的重复计算要求,例如,矩阵计算、图像处理、卡尔曼滤波、多媒体查询等,应用该算法比较有效.除此之外,该算法也取得了负载平衡的效果,并且考虑了异构系统在启动作业执行时刻的实时状态,能解决实时系统随机产生的求解问题.尽管我们为了讨论方便,假设初始作业负载位于单个处理器上.实际上,如果多个作业初始位于多个处理器上,算法同样有效.

从实验结果来看,随着处理器数的增多,本文提出的共享算法对性能有较大的改善.实验 2 在计算量较小时,通信比例大,对计算性能有一定的影响.

如果负载不是任意可划分的,而是模块可划分的,情况会怎样,我们将在今后的工作中进行研究.

参考文献

1 Ekmecic I, Tartalja I. A survey of heterogeneous computing: concepts and systems. Proceedings of the IEEE, 1996, 84(8): 1127~1144

2 Wang M, Kim S, Nichols M *et al*. Augmenting the optimal selection theory of super-concurrency. In: Freund R F, Eshaghian E eds. Proceedings of the Workshop on Heterogeneous Processing, Los Alamitos, CA: IEEE CS Press, 1992. 13~21

3 Haddad E. Load distribution optimization in heterogeneous multiple processor systems. In: Freund R F ed. Proceedings of the Workshop on Heterogeneous Processing. Los Alamitos, CA: IEEE CS Press, 1993. 42~47

4 Khokhar A, Prasanna K, Shaaban E. Heterogeneous computing: challenges and opportunities. IEEE Computer, 1993, 26(6): 18~27

5 Marco A, Luigi R, Lorenzo V. A hybrid approach to adaptive load sharing and its performance. Journal of System

Architecture, 1997,42(10):679~696
6 Shin G, Hou C J. Design and evaluation of effective load sharing in distributed real-time systems. IEEE Transactions on Parallel and Distributed Systems, 1994,15(7):704~719
7 Cierniak M, Zaki M J, Li W. Compile-Time scheduling algorithms for a heterogeneous network workstation. The Computer Journal, 1997,40(6):356~372

Load Sharing in Heterogeneous Computing

ZENG Guo-sun[1,2] LU Xin-da[1]

[1](*Department of Computer Science and Engineering Shanghai Jiaotong University Shanghai* 200030)
[2](*Department of Computer Science Jiangxi Normal University Nanchang* 330027)

Abstract In heterogeneous parallel computing systems based on message-passing, every processor or computer has capability of scheduling and executing jobs autonomously and independently. For high-performance computing, a processor, which originally has a divisible job, can call other heterogeneous processors for co-computing with load sharing so that job completing time can be decreased. In this paper, a scheme for load sharing in heterogeneous computing is presented. Firstly, invoking load-sharing protocol, the originating processor collects available data such as load sharing interval time and computing power from its partners. Secondly, a function is constructed which represents the relationship between the total job magnitude and the job completion time. The function is essential for choosing a suite of appropriate processors, optimally partitioning and scheduling jobs, as well as minimizing the job completing time. Finally, an efficient algorithm is proposed through the examples in application and experiments. The algorithm is real-time at the moment when the job is just started, and can be extended to the situation of multiple jobs originated at multiple processors.

Key words Load sharing, heterogeneous computing, job partitioning, scheduling, speedup.