文章编号: 1001-0920(2011)09-1315-06

求解异构并行系统任务分配的混合离散粒子群算法

蒋建春[1,2a], 汪同庆[1], 曾素华[2b]

(1. 重庆大学 光电技术及系统教育部重点实验室, 重庆 400044;
2. 重庆邮电大学 a. 自动化学院, b. 计算机科学与技术学院, 重庆 400065)

摘 要: 针对异构并行任务分配的最小完成时间和负载均衡组合优化问题, 提出一种混合离散微粒群算法, 将启发式 Sufferage 算法引入离散微粒群算法 (DPSO) 中, 改进 DPSO 算法中的位置速度关系模型, 提高 DPSO 算法的搜索效率和精度. 通过实验验证, 从算法效率和收敛速度上均优于 DPSO 算法和 GA 算法, 且负载均衡度较好.

关键词: 异构多核处理器; 任务分配; 最大完成时间; 负载均衡; 混合离散微粒群算法

中图分类号: TP311 **文献标识码:** A

Solving task assignment problem of heterogeneous parallel systems by hybrid DPSO algorithm

JIANG Jian-chun[1,2a], *WANG Tong-qing*[1], *ZENG Su-hua*[2b]

(1. Key Lab of Optoelectronic Technique and System of Ministry of Education, Chongqing University, Chongqing 400044, China; 2a. College of Automation, 2b. College of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China. Correspondent: JIANG Jian-chun, E-mail: dennis_jjc@yahoo.com.cn)

Abstract: A hybrid discrete particle swarm optimization(DPSO) algorithm is proposed for heterogeneous parallel system to get the minimal completion time and the optimal load balancing, in which the Sufferage algorithm is introduced to optimize the position and velocity model of DPSO and improve the DPSO ability. Compared with the standard generation algorithm(GA) and single DPSO algorithm, simulation results show that, the modified DPSO is better in efficiency and searching ability and can get a good load balance.

Key words: heterogeneous multi-core processor; task assignment; maximum completion time; load balancing; sufferage-discrete particle swarm optimization

1 引 言

异构并行系统的任务分配是控制优化的研究热点之一, 如生产车间的流水线作业调度、多处理器任务分配、物联网的组网等问题. 如何根据系统特性合理地分配任务, 使系统各个部分达到最优组合, 是实现系统性能高效运行的关键. 但在任务的分配过程中因受到多种因素的影响, 异构环境下的任务分配和作业调度问题往往是局部目标和全局目标相互制约, 不能同时满足, 被证明是一个 NP-Hard[1]的组合优化问题. 因此针对这种问题寻找一种高效的算法快速地找到一个较好的最优或次优解是目前异构系统任务分配的研究重点.

本文主要针对异构并行系统的作业调度和任务分配问题, 提出一种基于 Sufferage 启发式算法[2] 和 DPSO 算法[3]的混合离散微粒群算法 (SDPSO), 改进 DPSO 算法效率和搜索精度. 通过在异构多处理器系统的任务分配的组合优化问题中进行实验, 验证了该算法的可行性.

2 相关工作

在异构并行系统中, 如异构多处理器系统, 任务可以调度或分配到不同的处理器上运行, 以满足系统总体性能的需要. 任务静态分配及调度一般采用启发式算法进行任务静态调度分配, 如 Min-min, Max-min,

收稿日期: 2010-06-24; **修回日期:** 2010-08-29.

基金项目: 核高基重大专项基金项目(2009ZX01038-002-002); 重庆市科技攻关计划项目(CSTC, 2009AB2244); 重庆市教委科学技术研究项目(KJ090526).

作者简介: 蒋建春(1975−), 男, 讲师, 博士生, 从事嵌入式系统、智能控制等研究; 汪同庆(1949−), 男, 教授, 博士生导师, 从事模式识别、智能控制等研究.

Sufferage 等算法[2,4]. 这些算法大多以单一的指标进行任务分配, 对于解决组合优化问题能力较弱.

以群体智能优化为目标的进化算法, 如蚁群算法、进化策略、遗传算法等[2,4-8], 能够较好地处理多目标组合优化问题, 获得一个较好的解, 但这些算法较复杂, 效率低. 所以, 寻求新的更加有效的任务调度及分配算法一直是任务调度领域最关注的问题.

粒子群优化算法 (PSO) 是一种新的全局优化算法, 与其他群体智能算法相同, 都属于群智能演化计算技术, 随机初始化一群体, 然后根据适应值对其进行评价, 从而决定是否进一步进行搜索. 但是 PSO 算法没有诸如繁殖、交叉和变异等遗传操作, 只通过简单的算术运算进行演化, 算法简单容易实现. 作为一种重要的优化工具, PSO 算法已经在许多优化问题中得到成功的应用.

如何利用有效的数学工具对 PSO 算法的运行行为、收敛性及计算复杂性进行分析是目前的研究热点之一. 文献 [9-11] 已经阐述了 PSO 算法在任务分配方面的研究和应用, 但主要限于同构和分布式系统中, [12-15] 对异构多处理器和分布式系统中的 PSO 算法进行了分析和研究, [3,16,17] 对 PSO 算法在作业调度方面作了大量阐述和研究. 在这些文献中, 对异构并行系统的任务分配研究较少. PSO 算法在解决离散组合优化问题方面还有待于进一步研究和完善. 本文针对异构并行系统的任务分配组合优化问题特点, 在研究 DPSO 基础上, 通过融入启发式 Sufferage 算法, 改善 DPSO 算法模型, 提高 DPSO 算法在异构环境下的搜索效率和精度.

3 异构并行系统组合优化问题描述

在异构并行系统中, 任务在不同的处理单元的处理时间是不相同的, 为了提高系统的总的执行效率而对任务分配和调度进行优化, 任务可以按照不同的顺序分配到不同的处理单元并行执行. 系统对任务分配的要求, 主要是最小最大完成时间、任务完成的最小消耗、各个处理单元的负载平衡等. 针对具体应用, 可以选择一个或几个指标作为任务分配调度的目标. 本文以异构多核处理器任务分配为例, 针对系统独立任务分配中的最小最大完成时间和负载均衡两个方面来进行算法设计和验证.

假设异构多核处理器中的处理器核个数为 m, 存在一个任务集, 任务的总数为 n, 任务 i 在处理单元 j 上的执行时间为 $e_{i,j}$, 则每个任务的执行时间可表示为

$$E_i = \{e_{i,1}, e_{i,2}, \cdots, e_{i,j}, \ldots, e_{i,m}\}, \tag{1}$$

其中 $1 \leqslant i \leqslant n$, 这样便形成了任务的执行时间矩阵 $E_{n\times m}$. 假设划分的任务具有较小的耦合性, 忽略任务间通信时间, 则处理单元 j 上的所有任务完成时间为

$$T_j = \sum_{i\in P_j} e_{i,j}. \tag{2}$$

其中: $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$, P_j 表示分配到处理单元 j 上的任务集, 则最小最大完成时间任务分配适应度函数

$$f_{\text{completion}} = \min(\max_{1\leqslant j\leqslant m}(T_j)). \tag{3}$$

负载均衡是指各个处理单元上所有任务的完成时间的差值变化范围. 如果变化较大, 则负载不均程度较大; 如果变化较小, 则负载均衡较好. 对于这种情况, 可采用均方差来描述负载均衡程度, 用 $T_j(k)$ 表示第 k 次分配后处理单元 j 上的负载, $\bar{T}(k)$ 表示第 k 次分配后平均负载, 设总的分配次数为 l, 则负载均衡适应度函数可表示为

$$f_{\text{balance}} = \min_{1\leqslant k\leqslant l}\Big(\frac{1}{m}\sqrt{\sum_{1\leqslant j\leqslant m}(T_j(k)-\bar{T}(k))^2}\Big). \tag{4}$$

而平均负载 $\bar{T}$ 在不同分配结果时, 其值也不一样. 在一般的启发式算法中, 对于负载均衡问题, 采用多次参数调整重新分配可以获得一个较好的结果.

为了便于计算, 可以直接采用每次分配后各处理器核的负载和平均负载的差值的绝对值之和来表示, 则负载均衡适应度函数变为

$$f_{\text{balance}} = \min_{1\leqslant k\leqslant l}\Big(\frac{1}{m}\Big|\sum_{1\leqslant j\leqslant m}(T_j(k)-\bar{T}(k))\Big|\Big). \tag{5}$$

式 (3) 和 (5) 作为任务分配的目标函数, 在任务分配过程中不断对目标函数值进行更新和优化, 以满足系统要求.

4 SDPSO 算法设计

4.1 SDPSO 算法设计思想

在 DPSO 算法中, 微粒的初始状态一般是随机的, 采用随机数和位置更新算法来获得微粒下一个位置, 在微粒位置的优化过程中通过适应度函数确定微粒的最佳位置, 这样需要对局部最优和全局最优进行不断更新, 需要较多的迭代次数. 因此怎样确定微粒的初始状态和位置更新算法, 使微粒快速接近全局最优位置是 SDPSO 算法设计的重点.

Sufferage 算法是基于调度损失最小而进行任务调度分配的, 它不仅将任务的最早完成时间作为调度目标, 还考虑每个任务在相应处理单元的最早完成时间和次早完成时间差, 即 $\text{sufferage}(i) = e_{i,\text{next}} - e_{i,\min}$. sufferage 值反映了某个任务如果不分配到完成时间最早的处理单元上将会造成的损失. 因此, 要尽量调度分配 sufferage 值大的任务到使之完成时间最少的处理单元上, 使系统的执行效率更高. Sufferage 算法

的优点是 makespan 较小, 算法实现简单. 但是, 对于异构并行多处理单元系统的作业调度和任务静态分配, 主要考虑任务在执行前的分配过程. 怎样根据任务在各个处理单元的执行时间进行任务分配, 使整个系统根据这种分配结果实现最大完成时间最小、负载均衡.

考虑到异构并行系统任务分配问题的特殊性和 Sufferage 算法与 DPSO 算法各自的特点, 将任务所在的处理单元编号作为微粒群算法中的粒子位置, 将处理单元的所有任务最小最大完成时间作为微粒的位置调整的适应度函数, 将 Sufferage 算法和 DPSO 算法相结合, 通过 DPSO 算法获得任务的分配位置, 结合 Sufferage 算法获得相应微粒运动的速度, 使微粒向系统最大完成时间最小的方向运动.

在任务分配中, 每个任务都有一个局部最优的位置, 任务分配到这个处理单元上可以获得最高效率和最少的执行时间花费. 通过 Sufferage 算法在任务执行时可根据任务的完成时间将任务分配到完成时间最少的处理单元上. 而对于任务在执行前静态分配时, 通过分配可以获得一个局部最优的任务分配结果, 这些局部最优的位置会造成系统的负载不均衡和最大完成时间较大, 不能满足全局最优, 因此需要改变任务的位置来适应系统的要求.

在异构并行系统中, 一个微粒调度与分配需要将 sufferage 值进行两方面的比较: 一是横向比较, 二是纵向比较. 横向比较是指微粒在不同处理单元上的 sufferage 值, 该值可用来判断微粒可能调整的位置. 在不同处理单元中, 微粒的 sufferage 值越小, 微粒朝该处理单元运动的可能性越大. 这种比较是微粒当前位置和局部最优位置的比较. 纵向比较是指微粒和其他在同一处理单元上的微粒的 sufferage 值比较, 此时可用来判断是哪个微粒调整时更符合系统要求, 这种比较是微粒当前位置和全局最优位置的比较. 同时在每一轮迭代中, 在同一处理器核上, 一个微粒与其他微粒相比, sufferage 值越大, 该微粒调整时给系统带来的执行时间增量越大, 系统效率越低, 因此位置调整的概率越小.

为了提高 DPSO 算法的效率, 微粒的初始位置不是随机的, 而是按照微粒最小执行时间对微粒进行初始化, 将其分配到执行时间最少的处理单元中, 便于按照 Sufferage 算法进行微粒位置更新. 该初始位置也是微粒的局部最优位置, 但并不是全局最优位置. 因而微粒在位置调整过程中, 微粒会向执行时间大的处理单元移动, 这样会带来平均负载的部分增加.

4.2 SDPSO 算法设计

在异构多核处理器中, 任务在不同处理单元中的执行时间存在差异, 每改变一个位置会对其他处理器核的负载产生影响, 从而影响任务的完成时间. 与 DPSO 算法类似, SDPSO 算法的关键就是要根据问题领域定义位置和速度的表示, 以及它们的运算规律和粒子的运动方程. 下面对独立任务分配的 SDPSO 算法的基本量及其运算法则进行设计.

4.2.1 微粒的位置

在异构多核处理器的任务分配时, 将每个任务看成一个微粒, 将任务所在的处理器核编号看作微粒的位置. 用一个任务分配表 X 来表示, 它是一个 N 维向量, 用 $X = (x_1, x_2, \cdots, x_i, \cdots, x_n)$ 表示所有微粒的位置, 其中 $1 \leqslant x_i \leqslant m$, 用于表示任务 i 所在的位置, 即所在的处理器核编号. 如 $x_5 = 4$ 表示任务 5 被分配到处理器核 4 上运行.

4.2.2 微粒的速度

在 DPSO 算法中, 微粒的位置和速度都不连续, 速度的作用是改变微粒的位置. 这里的速度是指微粒将要改变的位置, 即任务将要分配的处理器核编号. 与微粒的位置的定义类似, 速度是一个 N 维向量, 用 $V = (v_1, v_2, \cdots, v_i \cdots, v_n)$ 表示, 其中 $0 \leqslant v_i \leqslant m$, 表示微粒 i 将要更新的位置, 即下一个处理器核编号, 0 表示不需要更新位置.

4.2.3 微粒位置更新公式

微粒的位置更新公式表示任务下一次调整的位置, 即

$$x_i(k+1) = x_i(k) + v_i(k+1), \tag{6}$$

如果 $v_i(k+1) > 0$, 表示将要调整的处理器编号, 则 $x_i(k+1) = v_i(k+1)$; 否则微粒位置保持不变, 即 $x_i(k+1) = x_i(k)$.

4.2.4 微粒速度更新公式

微粒的速度是微粒位置搜索的关键, 不仅影响算法的收敛性, 还与算法的全局搜索效率相关.

在算法中, 采用 Sufferage 算法设计微粒速度更新公式, 即 sufferage 值作为微粒位置调整的依据, 使系统满足适应度要求. 在异构多核处理器任务分配过程中, 为了获得更小的最大完成时间, 微粒的速度除了与微粒本身在各个处理单元中的执行时间和 sufferage 值相关外, 还与其他微粒的 sufferage 值和执行时间都相关.

在每一次迭代分配过程中, 定义微粒可以移动的位置有 3 个 (任务以当前位置为中心): 1) 保持位置不变; 2) 向 sufferage 值为正的方向移动; 3) 向 sufferage 值为负的邻近位置移动. 向 sufferage 值正的方向移动时会增加系统的总的完成时间, 相反会减小总的完成时间, 因此可用下式描述任务的速度更新公

式:

$$v_i(k+1) = wv_i(k) + c_1r_1(x_{i,\text{pbest}} - x_i(k)) + c_2r_2(x_{i,g\text{best}} - x_i(k)). \quad (7)$$

$x_i(k)$ 是微粒第 k 次迭代后的位置, $x_{i,\text{pbest}}$ 是微粒 i 的局部最优位置, 在调整过程中, 用微粒的 sufferage 排序值中比当前位置次小的前一个位置表示; $x_{i,g\text{best}}$ 是微粒的全局最优位置, 表示为了获得更好的解, 下一个可能调整的最优位置, 用 sufferage 值比当前位置次大一个位置表示. 由于微粒的位置不只是在最少执行时间和次少执行时间的处理器核中分配, 为了满足系统最小最大完成时间和负载均衡要求, 微粒可以在任意处理单元中调整. 为了结合 DPSO 算法, 需要修改 Sufferage 算法计算公式, 采用 $S = (s_1, s_2, \cdots, s_i, \cdots, s_n)$ 表示微粒的 sufferage 值, s_i 表示任务 i 被分配到下一个次少执行时间的处理单元上与当前处理单元的执行时间之差, 即

$$s_i(x_i(k)) = e_i(x_i'(k)) - e_i(x_i(k)), \quad (8)$$

其中 $x_i'(k)$ 是微粒 i 下一个将要调整的新位置, 即任务的处理器核编号, 可以是 $x_{i,\text{pbest}}$ 或 $x_{i,g\text{best}}$. 式 (8) 反映了微粒在朝系统分配目标前进过程中自身位置与局部最优位置和全局最优位置的关系.

1) 参数 w. w 是当前速度对下一个速度影响的权值, 与 $x_{p\text{best}}$ 和 $x_{g\text{best}}$ 紧密相关. 当前微粒与其他同处相同位置的微粒相比, 如果在这些微粒的前一位置对当前位置的 sufferage 绝对值从小到大的排序中位置越靠前, 向前一个位置移动所减少的时间就越小, 则微粒向前一个位置移动的概率越小. 相反, 在当前位置所有微粒后一位置的 sufferage 值从小到大的排序中的位置越靠后, 在微粒移动时带来的时间增量也越大, 位置调整的概率也越小. 当这两个条件都满足时, 微粒位置保持不变的概率就越大, 即 w 越大. 根据上述分析, w 采用下式计算:

$$w = 1 - \frac{\underset{x_i \in \text{sort}(S_j)}{x_i} \bigvee \underset{y_i \in \downarrow \text{sort}(S_j')}{y_i}}{\text{count}(P_j)}. \quad (9)$$

其中: S_j 表示分配到位置 j 上的所有微粒的次少执行时间与当前位置的差值, x_i 表示微粒 i 在 S_j 从小到大排序中的位置; S_j' 表示位置 j 上的所有微粒的次多执行时间与当前位置的差值, y_i 表示微粒 i 在 S_j' 从大到小排序中的位置; $\text{count}(P_j)$ 表示在位置 j 上的微粒个数. 取 x_i 和 y_i 中较大值来计算 w.

2) 参数 c_1. c_1 反映了微粒局部最优位置对速度的影响. 在当前位置所有微粒上一个位置的 sufferage 值中, sufferage 绝对值越大, 微粒位置调整节省的时间越多. 与其他微粒相比, 该微粒调整到该位置的概率越大, 因此 c_1 就越大, 则 c_1 可用下式表示:

$$c_1 = \frac{s_i(x_{i,\text{pbest}})}{\sum\limits_{t \in P_j}(s_t(x_{t,\text{pbest}}))}. \quad (10)$$

$s_i(x_{i,\text{pbest}})$ 越大, 则 c_1 越大, 调整到位置 $x_{i,\text{pbest}}$ 上的概率就越大.

为了保证算法的收敛性, 必须对算法速度和位置进行限制. 根据最优化原则, 微粒将被分配到执行时间较小的处理单元, 可以根据微粒的执行效率对微粒的速度进行限制. 微粒的执行效率用微粒在不同处理单元的执行时间与其最少执行时间的比值表示, 即任意微粒在处理单元上的执行效率比为

$$r_{i,j} = \frac{e_{i,j}}{\min\limits_{1 \leqslant k \leqslant m}(e_{i,k})}. \quad (11)$$

$r_{i,j}$ 越大, 说明微粒 i 在该处理单元上的执行时间 $e_{i,j}$ 与最少执行时间相差越大, 微粒在该处理单元上执行的效率越低. 在微粒位置调整时, 设定一个阈值 h 来限定微粒的位置移动, 微粒只能在 $r_{i,j}$ 小于 h 的位置移动.

参数 c_1 可更改为

$$c_1 = \begin{cases} \dfrac{s_i(x_{i,\text{pbest}})}{\sum\limits_{t \in P_j}(s_t(x_{t,\text{pbest}}))}, & r_{i,j} \leqslant h; \\ 0, & r_{i,j} > h. \end{cases} \quad (12)$$

3) 参数 c_2. c_2 反映了微粒与全局最优位置间的关系, 它与其他微粒位置相关联, 用式 (11) 计算. c_2 越大说明该微粒调整到下一个位置时比其他微粒增加的时间少, 因此移动的概率较大.

$$c_2 = 1 - \frac{s_i(x_{i,g\text{best}})}{\sum\limits_{t \in P_j}(s_t(x_{t,g\text{best}}))}. \quad (13)$$

在系数 w, c_1 和 c_2 中, 值越大的参数说明该项的影响越大, 微粒向对应的位置移动的概率越大. 为了防止在微粒运动过程中出现早熟和陷于局部最优, 需要在速度更新公式中加入在 $[0, 1]$ 范围的随机数 r_1 和 r_2. 取 $\max(w, c_1r_1, c_2r_2)$ 对应的项作为下一次迭代的速度值.

对于处理器核 $x_i(k)$ 和 $x_i'(k)$, 微粒在 i 的位置调整后的负载分别为

$$T(x_i(k)) = T(x_i(k)) - e_i(x_i(k)), \quad (14)$$

$$T(x_i'(k)) = T(x_i'(k)) + e_i(x_i'(k)). \quad (15)$$

为了满足负载均衡的要求, 处理器核 $x_i(k)$ 中负载大于平均负载时, 才对该处理器核上的部分微粒进行位置更新, 不然会带来新的负载不均. 同时, 因为按照 Sufferage 算法分配原则, 每一次重分配必将带来平均负载的变化, 所以必须要预留一定分配余量, 以保证负载均衡. 为了位置更新更精确, 在完成一轮微粒更新后, 需重新计算平均负载, 计算公式如下:

$$L_{\text{avg}}(k)=\frac{1}{m}\sum_{j=1}^{m}T_j(k), \tag{16}$$

其中 $T_j(k)$ 按式 (2) 计算求得.

4.2.5　位置的减法

在 SDPSO 算法中位置的减法被用于求解微粒的速度, 如式 (7), 实际上是微粒位置的一种置换, 减法的结果与最后的参数相关联, 只有 w, c_1r_1, c_2r_2 中最大的一个参数对应的项有效.

从上面分析可以得出, 该速度更新公式仍旧体现出了微粒群优化的机制, 即根据微粒原始位置、全局最佳位置和局部最佳位置的差距来调整得到微粒新的位置.

4.3　SPSO 算法分析

通过该算法进行微粒调整时, 每次调整的是负载最大的处理器核上的微粒, 先比较各个微粒前一个位置和后一个位置的 sufferage 值, 以及微粒所在的位置情况, 选择式 (7) 中系数最大的一项作为微粒调整的速度值, 使调整的微粒所增加的负载较小, 从而使微粒位置调整后带来的 makespan 也较小. 依次类推, 搜索所有大于平均负载的处理器核的微粒, 并进行位置调整, 直到最小完成时间和负载均衡满足要求. 根据算法设计, 对算法执行步骤如下:

Step 1: 首先以最少执行时间原则初始化微粒的初始位置, 即得到微粒初始局部最佳位置 $x_{i,p\text{best}}$ 和 $x_{i,g\text{best}}$, 并记录最大完成时间, 设置迭代次数.

Step 2: 根据式 (2) 和 (16) 分别计算各个处理器负载与平均负载.

Step 3: 根据式 (5) 计算各处理器核负载均衡程度.

Step 4: 在负载最大的处理单元上, 根据式 (7)~(13) 计算该处理器核上微粒速度.

Step 5: 根据式 (6) 更新微粒位置, 并按式 (14) 和 (15) 计算调整后的负载.

Step 6: 更新 $x_{i,p\text{best}}$ 和 $x_{i,g\text{best}}$.

Step 7: 重复 Step 3~Step 6, 直到该处理器负载小于平均负载要求.

Step 8: 返回 Step 2, 直到所有处理器核负载符合要求.

Step 9: 判断最大负载是否满足条件, 如果不满足, 则放弃分配结果, 返回 Step 2; 否则更新当前最大完成时间, 并按式 (4) 计算判断负载均衡程度.

Step 10: 返回 Step 2, 直到迭代次数完成或最大完成时间不再减小.

5　仿真实验及结果分析

为了验证该算法的有效性, 本文采用文献 [1] 中的 DPSO 算法进行对比分析. 采用 Matlab 作为仿真工具, 分别用 6 组随机产生的数据对 SDPSO, DPSO, GA 三种算法进行对比分析验证.

表 1 所示是 3 种算法在相同迭代次数时最大负载比较. 通过对比分析可以看出, 在较小的相同迭代次数时, SDPSO 算法分配获得的最大负载最小.

表 1　算法性能比较

$m\times n$	SDPSO	DPSO	GA	迭代次数
3×10	0.861 0	0.907 1	0.938 4	5
8×60	0.855 2	0.926 3	0.987 6	20
8×120	1.725 5	1.951 7	1.982 3	40
8×160	2.269 5	2.328 0	2.372 6	50
16×80	0.382 1	0.415 8	0.418 7	60
16×200	0.896 5	0.973 5	0.991 4	100

表 2 是 3 种算法对于获得理想结果时算法的迭代次数数据. 通过表 2 可以看出, 在获得相近或相同理想分配结果时, SDPSO 算法比 DPSO 算法迭代次数少, DPSO 算法比 GA 算法的迭代次数少. 通过表 1 和表 2 数据对比分析可以得出, SDPSO 算法的收敛速度是 3 种算法中最快的.

表 2　最大负载接近于理想情况时的迭代次数

$m\times n$	SDPSO		DPSO		GA	
	最大负载	迭代次数	最大负载	迭代次数	最大负载	迭代次数
3×10	0.861 0	4	0.861 0	16	0.861 0	23
8×60	0.855 2	16	0.855 2	35	0.855 2	43
8×120	1.725 5	37	1.748 9	82	1.756 6	126
8×160	2.269 5	42	2.283 2	87	2.271 6	143
16×80	0.382 1	57	0.405 8	177	0.398 7	286
16×200	0.896 5	98	0.896 5	241	0.914 8	328

图 1 是 SDPSO, DPSO 和 GA 算法在 $m=8$, $n=16$ 时任务分配优化过程的状态图. 从图 1 中可以看出, 在任务分配初始化时, SDPSO 算法得到的初始最大负载是最大的, 因为采用的是任务最少执行时间来进行初始化, 任务被分配到执行时间最小、效率最高的处理器核中, 所以造成个别处理器比其他处理器分配更多的任务. 从图 1 中也可看出, SDPSO 算法的收敛速度比其他两种算法快得多.

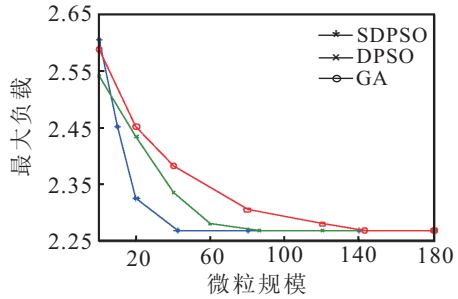

图 1　$m=8, n=160$ 时 3 种算法迭代性能比较

由于在实验数据中针对每一组实验数据是不相同的, 为了分析采用SDPSO算法进行任务分配后的负载均衡性和负载均衡稳定性, 采用下式进行计算:

$$f'_{\text{balance}} = \frac{1}{m\bar{T}}\Big|\sum_{1\leqslant j\leqslant m}(T_j-\bar{T})\Big|. \qquad (17)$$

其中: T_j 表示分配结束后各个处理器核上的负载, $\bar{T}$ 是分配结束后的平均负载. 图2是分配后的负载变化范围, 从图2中可以看出负载变化范围较小, 负载均衡度较好.

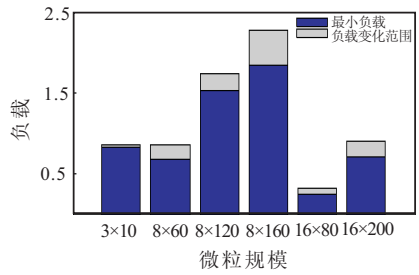

图 2 不同规模的负载变化范围

图3所示为根据式(17)算出的6组分配结果的负载均衡情况. 从图3中可以看出, 采用SDPSO算法进行任务分配, 负载均衡程度较稳定.

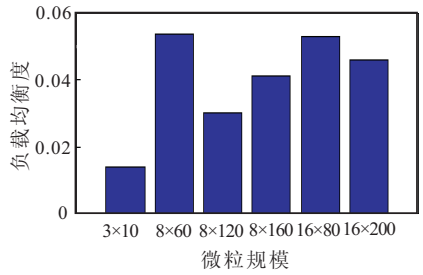

图 3 不同规模下的任务分配后负载均衡度比较

与文献[6,16]中的DPSO算法相比, 该算法中保留了速度更新公式中的第1项, 为微粒位置调整提供了更多的选择, 提高了算法搜索精度; 同时, 参数 w 采用了动态值, 也提高了算法的搜索速度.

6 结 论

随着异构并行系统的广泛应用, 任务分配是影响系统总体性能的一个关键因素, 怎样减少算法的复杂度和提高算法效率来满足系统要求是异构并行系统任务分配的重点. 通过理论分析和实验验证, SDPSO算法提高了DPSO算法的搜索效率. 与GA算法相比, 在快速接近最小最大完成时间的过程中, SDPSO效率更高. 在实际应用中需要选择合适的负载调节参数来提高SDPSO算法的效率.

参考文献(References)

[1] Fernandez-Baca D, Allocating modules to processors in a distributed system[J]. IEEE Trans on Software Engrg, 1989, 15(11): 1427-1436.

[2] Braun T D, Siegel H J, Beck N, et a1. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing Systems[J]. J of Parallel and Distributed Computing, 2001, 6l(6): 810-837.

[3] Rameshkumar K, Suresh R K, Mohanasundaram K M. Discrete particle swarm optimization(DPSO) algorithm for permutation flowshop scheduling to minimize makspan[C]. Proc of ICNC 2005. Changsha, 2005: 572-581.

[4] Izakian H, Abraham A, Snášel V. Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments[C]. Proc of the IEEE Int Workshop on HPC and Grid Applications. Sanya, 2009: 8-12.

[5] 李宗勇, 彭霞, 等.基于蚁群算法的参数相关网格任务调度算法研究[J].系统仿真学报, 2007, 19(l4): 3196-3252.
(Li Z Y, Peng X, et a1. Scheduling interrelated tasks in grid based on ant algorithm[J]. J of System Simulation, 2007, 19(l4): 3196-3252.)

[6] Chen H, Cheng A M K. Applying ant colony optimization to the partitioned scheduling problem for heterogeneous multiprocessors[J]. ACM SIGBED Review, 2005, 2(2): 11-14.

[7] 马学彬, 温涛, 郭权, 等. 一种基于遗传算法的网格任务调度算法[J]. 东北大学学报: 自然科学版, 2007, 28(7): 973-977.
(Ma X B, Wen T, Guo Q, et a1. GA-based algorithm for task scheduling on computational grid[J]. J of Northeastern University: Natural Science, 2007, 28(7): 973-977.)

[8] 钟求喜, 谢涛, 陈火旺. 基于遗传算法的任务分配与调度[J]. 计算机研究与发展, 2000, 37(10): 1197-1203.
(Zhong Q X, Xie T, Chen H W. Task matching and scheduling by using genetic algorithms[J]. J of Computer Research & Development. 2000, 37(10): 1197-1203.)

[9] 钟一文, 杨建刚. 独立任务分配问题的离散粒子群优化算法[J]. 模式识别与人工智能, 2006, 19(3): 399-405.
(Zhong Y W, Yang J G. Discrete particle swarm optimization algorithm for independent task assignment problem[J]. PR&AI, 2006, 19(3): 399-405.)

[10] Salman A, Ahmad I, AI-Madam S. Particle swarm optimization for task assignment problem[J]. Microprocessors and Microsystems, 2002, 26(8): 363-37l.

[11] 陈养平, 王来雄, 黄士坦. 基于粒子群优化的多处理器任务调度算法[J]. 吉林大学学报: 信息科学版, 2007, 25(3): 277-285.
(Chen Y P, Wang L X, Huang S T. Task scheduling algorithm for multiprocessor based on particle swarm optimization[J]. J of Jilin University: Information Science Edition, 2007, 25(3): 277-285.)

(下转第1326页)