异构计算中体系结构感知的并行任务分簇方法

郝水侠[1,2] 曾国荪[2]

(同济大学计算机科学与技术系 上海 201804)[1] (江苏师范大学数学科学学院 徐州 221116)[2]

摘 要 异构计算是高效能计算发展的必然趋势,针对异构计算运行中并行任务和体系结构难匹配的问题,提出了实现并行任务和体系结构匹配的并行任务分簇方法。首先给出效能的概念及异构计算中体系结构感知的分簇问题,然后从理论上分析了异构匹配与效能的关系,提出了实现异构计算匹配和结构匹配的分簇理论,目的是发挥异构计算中机器的潜能,协同处理并行任务,实现高效能。在此基础上,给出相应的算法。最后通过仿真实验说明,该方法可通过簇图与体系结构的匹配缩短通信开销在执行时间上所占的比例,从而缩短并行执行时间,以提高系统利用率,最终实现异构计算的高效能。

关键词 异构计算,并行任务匹配,体系结构感知,分簇

中图法分类号 TP338 **文献标识码** A

Architecture-aware Parallel Task Clustering Policy in Heterogeneous Computing

HAO Shui-xia[1,2] ZENG Guo-sun[2]

(Department of Computer Science and Technology, TongJi University, Shanghai 201804, China)[1]

(School of Mathematical Science, Jiangsu Normal University, Xuzhou 221116, China)[2]

Abstract Heterogeneous computing has been a trend of high-productivity computing. Matching between parallel task and architecture in heterogeneous computing becomes a key idea to realize high productivity. We provided parallel task clustering policy based on matching between parallel task and architecture. Firstly we gave the concept of high-productivity and the problem of clustering on heterogeneous computing. Secondly after theoretically analyzing the relation between heterogeneous matching and productivity, we gave the method of realizing respectively computing and structure matching. Thirdly we gave accordingly the architecture-aware parallel task clustering algorithm. Finally the simulation experimental results show that such algorithms can effectively realize heterogeneous matching and enhance the heterogeneous computing productivity.

Keywords Heterogeneous computing, Parallel task matching, Architecture-aware, Clustering

1 引言

异构计算是传统并行计算的发展和提升,正如李国杰院士指出:高性能计算机遇到挖掘并行性和可扩展的困难是现有技术无法逾越的,但目前的异构计算技术仍可能在现有条件下对高性能的改善取得一定进步。异构计算已成为高性能计算研究的新热点,它是指将性能和功能各异的计算机(如个人机、工作站、向量机、单指令多数据流、多指令多数据流计算机、中央处理器、图形处理器、数字信号处理器、专用机等)通过高速网络并以一定的耦合方式连成并行计算环境,充分利用程序和处理部件的异构性,各尽潜能,合理分治,使得完成时间最小的过程。

如何充分利用异构资源实现并行计算一直是异构计算研究的热点,本文试图利用图来挖掘并行任务和体系结构异构特征,实现两者异构匹配,从而达到设定目标。目前分簇法主要分为两类,一类是并行任务图上的节点和边用时间描述任务量和通信量,利用这种描述实现分簇适用于同构处理器。另一类是并行任务和体系结构分开,并行任务图的节点和边分别表示计算量和通信量,体系结构图的节点和边分别表示处理器的能力和通信带宽,不同的映射执行结果不同,这种方法用于异构处理器。故本文采用后一类描述方法,它能体现出异构处理器及结构特征。第一类描述方法中,典型的分簇方法有关键路径启发法[1]、表调度法[2]、图分解法[3]和近时序分簇算法[4]。关键路径启发式法包括 Kim 和 Browne 提出的线性分簇法(Linear Clustering, LC)[5]、Gerasoulis 和 Yang 提出的主序分簇法(Dominant Sequence Clustering, DSC)[7]、Wu 和 Gajski 的改变的关键路径法(Modified Critical Path, MCP)[6]。这些算法缩小了并行任务 DAG(Direct Acycle

到稿日期:2012-10-19 返修日期:2012-12-30 本文受国家863高技术研究发展计划(61103068),NSFC-微软亚洲研究院联合项目(2009AA012201),国家自然基金项目(60970155),教育部博士点基金项目(20090072110035),上海市优秀学科带头人计划项目(10XD1404400),高效能服务器和存储技术国家重点实验室开放基金项目(2009HSSA06)资助。

郝水侠(1973-),女,博士生,副教授,主要研究领域为并行计算、异构计算,E-mail:sxhaotj@gmail.com.cn;曾国荪(1964-),男,博士,教授,主要研究领域为并行计算、异构计算。

Graph)中关键路径的执行时间，方法是移走关键路径通信需求。Baxter 和 Patel 提出的静态任务局部簇聚(Localized Allocation of Static Tasks，LAST)表调度法是基于当前未处理节点和已分配簇节点的连接强度来进行簇聚。Mccreary 和 Gill 提出 clan 图分解法(clan-graph decomposition method，CLANS)，解析树的根是代表整个 DAG，孩子节点是 clan 的子图，且将簇分为线性簇、独立簇和主簇(primitive)。另外，Sarkar 提出 EZ(Edge-Zeroing)算法[6]，该算法每一步合并相连的、边权最大的两个簇成为一个更大的簇。Yeut Ming 提出将分簇、映射、调度 3 步综合在一起来实现[9]。在第二类描述中，Gilbert 提出 DLS (Dynamic Level Scheduling)[10]算法，与表调度类似，其仍根据任务节点等级进行调度，但是在任务节点等级定义时考虑了机器处理能力，而没有考虑机器间通信开销。Nectarios 提出了以通信开销最小为目标的分簇算法[11]，考虑了簇间通信量和处理器之间的通信处理能力。Neal 提出了满足处理器间 h 跳的柔性分簇的 FFS (Feasible Flexibility Scheduling)[12]算法，这些在不同程度上考虑了异构计算处理器的异构性或者处理器间的通信能力，但是处理器之间的通信能力仍然是相同的，或者体系结构是全连接的，现实中体系结构往往不是全连接的，而且处理器之间通信能力有差异。因此，本文提出体系结构感知的并行任务分簇法，其将体系结构特征(包括处理器能力和拓扑结构特征)作为并行任务分簇参考标准，从而为异构计算中任务和体系结构匹配提供条件，进而实现异构计算的高效能。

2 异构计算的分簇问题

2.1 并行任务及体系结构的形式化定义

定义 1(并行任务 PTG) 一个并行任务可定义为一个有向无环图，用四元组表示，记为 $PTG=(V,E,W,C)$，其中：$V=\{v_1,v_2,\cdots,v_n\}$ 是子任务节点集合，节点 $v_i\in V$ 代表计算任务中的一个子任务。$E=\{e_1,e_2,\cdots,e_m\}$ 是子任务节点间的有向边集合，$E=V\times V$，$e_k=\overrightarrow{v_iv_j}\in E$ 表示子任务 i 和子任务 j 之间存在依赖关系，子任务 v_j 必须等待任务 v_i 完成后才能开始执行。$W=\{w_1,w_2,\cdots,w_n\}$ 是任务的负载向量，w_i 表示子任务 v_i 的计算量。$C=(c_{ij})_{n\times n}$ 是有向边的通信量矩阵，c_{ij} 表示子任务 v_i 和子任务 v_j 的数据链路通信量。

定义 2(体系结构 UAG) 体系结构可定义为简单图，用四元组来表示，记为 $UAG=(P,L,O,Q)$。其中：P 是处理单元集合，节点 $p_i\in P$ 代表体系结构中的一个处理单元。L 是处理单元之间通信的集合，$L=P\times P$，$l_k=\overrightarrow{p_ip_j}\in L$ 表示处理单元 p_i 和 p_j 有直接通信链接。$O=\{o_1,o_2,\cdots,o_m\}$ 是对应处理单元单位时间内处理的计算量，o_i 为处理器 p_i 处理单位时间处理的任务量。$Q=(q_{ij})_{m\times m}$ 是处理单元之间单位时间处理的消息量。$q(p_i,p_j)$ 表示单位时间内处理单元 p_i 和 p_j 之间处理的消息量。

2.2 异构计算效能的概念

高效能是高性能计算发展的进一步目标，异构计算是目前高性能计算的典型模式，因为它能为高性能计算实现高效能提供可能的方法。下面给出效能的定义。

定义 3(并行任务完成时间 T) 在并行任务 PTG 中，出度为 0 的所有子任务完成时间的最大值，即

$$T=\max_{outdegree(v_i)=0\cap v_i\in V}T(v_i)$$

式中，$T(v_i)$ 表示并行子任务 v_i 的完成时间。

定义 4(体系结构利用率 U) 指体系结构中所有处理器利用率的综合指标，用 $U=\sum T(P_k)/\max\{T(P_k)\}*m$ 表示，其中，$T(P_k)$ 指单个处理器执行时间，m 指处理器个数。

定义 5(效能 PR) 用来描述体系结构完成并行任务属性特征，该属性用来衡量单位代价完成计算任务量的比率，是包括在特定机器、特定任务完成的性能、效率、质量和消耗代价等的一个综合指数。本文重点考虑两个指标：运行时间和系统利用率，效能公式定义为：

$$PR=\frac{U}{T}$$

2.3 体系结构感知的分簇问题

给定并行任务 $PTG=(V,E,W,C)$，体系结构 $UAG=(P,L,O,Q)$，其中 $|V|=n$，$|P|=m$，$n\gg m$。如何对 PTG 分簇形成簇图 $CG=\{V',E',W',C'\}$ 使得 $PR(CG,UAG)$ 最大。其中 $V'=\{V_1,V_2,\cdots,V_m\}$，$V_i\cap V_j=\varphi\ (i,j\in(1,2,\cdots,m)$ 且 $i\neq j)$，$\bigcup_{i=1}^{m}V_i=V$；$W'=\{W_1,W_2,\cdots,W_m\}$，$W_i=\sum_{v_x\in V_i}w(v_x)$；$E'=\{e_1',e_2',\cdots,e_r'\}$，$E'=V'\times V'$，$e_k'=\overrightarrow{V_iV_j}\in E'$，$k\in(1,2,\cdots,r)$，$e_k'$ 是簇间关系；$C'=\{c_1',c_2',\cdots,c_r'\}$ 是簇间通信量，$c_k'=\sum_{v_x\in V_p}\sum_{v_y\in V_q}c(v_x,v_y),v_x,v_y\in V,V_p,V_q\in V'$。

3 异构匹配与效能之间的关系

3.1 单纯考虑计算的优化匹配

并行任务 DAG 与多处理器的映射属于 NP 难问题，为了简化，假定任务及处理器之间通信忽略。根据定义，PTG 分簇形成簇图 CG 后总的任务量没有发生变化，上述分簇问题则可转换为数学优化问题：

已知 CG 总的计算量为 $W'=W_1+W_2+\cdots+W_m=W$，体系结构 UAG 总的处理能力为 $O=o_1+o_2+\cdots+o_m$。$W_i(i\in(1,2,\cdots,m))$ 取何值时 $PR(CG,UAG)$ 最大。

定理 1 若不考虑计算任务之间的方向，当每个并行子任务簇计算量为 $W_i=\frac{o_i}{\sum_{j=1}^{m}o_j}W$，即 $\frac{W_1}{o_1}=\frac{W_2}{o_2}=\cdots=\frac{W_m}{o_m}$ 时，体系结构的运行时间 $T=\max_i\frac{W_i}{o_i}$ 达到最小，且值为 $\frac{W}{O}$。

当 $\frac{W_1}{o_1}=\frac{W_2}{o_2}=\cdots=\frac{W_m}{o_m}$ 时，$\min\max_i\frac{W_i}{o_i}$ 取最小值，即并行任务在体系结构上的运行时间最小，同时由于所有机器同时开始工作又同时结束，因此每个机器的利用率为 1，这时达到利用率的最大值。根据效能的定义，PTG 在 UAG 上运行时效能最高。因此可得到如下定理。

定理 2 若不考虑计算任务之间的通信，当 $W_i=\frac{o_i}{\sum_{j=1}^{m}o_j}W$，即 $\frac{W_1}{o_1}=\frac{W_2}{o_2}=\cdots=\frac{W_m}{o_m}$ 时，达到 $PR(CG,UAG)$ 最大，其值为 $\frac{O}{W}$。

证明略。

此定理也说明当不考虑任务之间的先后关系，且并行任务的计算量和处理器之间的处理能力成正比时，系统的效能达到最高。在异构计算中，并行任务簇图的子任务量和对应处理器能力成正比时，并行任务可在对应体系结构运行时实现高效能。

3.2 单纯考虑通信的优化匹配

从并行算法设计的实际例子中，可以看到当算法结构和体系结构相适应时，算法复杂度会大大减小，结构匹配可以转化为优化问题。为了研究方便，只考虑通信开销，PTG 分簇后形成簇图 $CG=\{V',E',W',C'\}$，体系结构 $UAG=(P,L,O,Q)$，寻找一个映射 $M:V'->P$，满足

$$\sum_{(V_i,V_j)\in E'}\ \sum_{(M(V_x),M(V_y))\in l}\frac{C(V_i,V_j)}{q(M(V_x),M(V_y))}$$

最小，其中 l 表示 $M(V_i)$ 与 $M(V_j)$ 间的最短路径。

当处理器通信能力与距离成反比，且处理器相邻时，通信能力最强。相邻簇与相邻处理器对应时，通信开销最小。所以有如下定理。

定理 3 若只考虑通信开销，当 CG 与 UAG 同构时，

$$\sum_{(V_i,V_j)\in E'}\ \sum_{(M(V_x),M(V_y))\in l}\frac{C(V_i,V_j)}{q(M(V_x),M(V_y))}$$

取得最小值。

证明略。

通过理论分析实现了单纯考虑计算、通信下异构计算并行任务簇图与体系结构匹配高效能的方法，另外给出如何避免图分簇成形成环的问题。下面给出任意情况下异构计算中并行任务分簇的实现方法。

4 体系结构感知的并行任务分簇算法

4.1 簇相关概念

下面给出分簇所涉及的相关参数及计算表达式。

定义 6(任务 v_i 的前驱和后继) 对任务 $v_i\in V$，用参数 $pred(v_i)$ 和 $succ(v_i)$ 分别表示其前驱任务集合和后继任务集合。即：

$pred(v_i)=\{v_j\mid e(v_j,v_i)\in E\}$

$succ(v_i)=\{v_j\mid e(v_i,v_j)\in E\}$

定义 7 任务的深度值 $level(v_i)$：

$$level(v_i)=\begin{cases}0, & v_i \text{ 无父节点}\\ rand-select(1+\max(level(parent(v_i))),\max level), & v_i \text{ 无子节点}\\ 1+\max(level(parent(v_i))), & \text{其它}\end{cases}$$

式中，$parent(v_i)$ 返回的是子任务 v_i 的父节点的集合，$\max(parent(v_i))$ 的返回值是 v_i 父节点集合中最大深度的节点的深度值，可通过遍历方法来获取深度信息。

对任务 $v_i\in V$，最早开始时间 $t_s(v_i)$、完成时间 $t_e(v_i)$ 分别如下定义：

$$t_s(v_i)=\begin{cases}0 & pred(v_i)=\phi\\ \max\{t_e(fpred(v_i)),DAT(cpred(v_i),v_i)\}, & \text{其它}\end{cases}$$

$$t_e(v_i)=t_s(v_i)+w(v_i)/o(p_j)$$

式中，$o(p_j)=\max_{k\in \text{可选处理器}} o(p_k)$。

定义 8(数据到达时间、贵宾前驱和非贵宾前驱) 对任务 v_x，以 $DAT(v_i,v_x)$ 表示 v_x 的父任务 v_i 的数据到达时间；以 $fpred(v_x)$ 表示 v_x 的贵宾前驱，即 v_x 的数据到达时间最晚的前驱任务，$cpred(v_x)$ 表示 v_x 的次贵宾前驱，即 v_x 的数据到达时间次晚的前驱任务，即：

$DAT(v_i,v_x)=t_s(v_i)+c(v_i,v_x)/l(p_i,p_y)$

$fpred(v_x)=\{v_i\mid DAT(v_i,v_x)\geqslant DAT(v_j,v_x),\forall v_j\in pred(v_x),j\neq i\}$

$cpred(v_x)=\{v_i\mid DAT(v_i,v_x)\geqslant DAT(v_j,v_x),\forall v_j\in (pred(v_x)-fpred(v_x)),j\neq i\}$

4.2 体系结构感知的并行任务分簇算法

算法 1 体系结构感知的并行任务分簇算法 APC( )

输入：并行任务 PTG=(V,E,W,C)，体系结构 UAG =(P,L,O,Q)，其中|V|=n，|P|=m，n>>m

输出：簇 PT=$\{V_1,V_2,\cdots,V_m\}$，匹配序列 Assign =$\{(v_i,p_j)\}$，执行时间 T 及利用率 U

```
{
L1. Compuate η=W/O;//单位处理计算簇与处理器的比例
L2. PT[]= Clustering(PTG); //存储 PTG 分簇结果
L3. L←{v_i|indegree(v_i)=0,1≤i≤n};
    //PTG 入度为零的子任务进入准备队列 L
L4. ρ←{p_1,p_2,…,p_m};
    //按通信距离最短确定计算资源集合
L5. Assign←Φ; //子任务处理器匹配序列
L6. ε←L; //子任务执行队列初始化
L7. t_s(v_i)=0,t_e(v_i)=0,1≤i≤n;
    //所有子任务的开始和结束执行时刻初始化为 0
L8. V_a←Φ,1≤a≤m; //簇集合初始化为空
L9. T_total=0,U=0;
    //初始化任务执行时间和体系结构利用率为 0
L10. do until ε=Φ
L11. {for each v_i∈ε
     {//基于通信优化匹配思想，每次都选择通信能力最强的邻接资源 p_j
L12.   (v_i,p_j)←select(v_i,ρ);
L13.   If V_a=Φ then V_a←{v_i};w(V_a)=w(v_i);
L14.   Assign←Assign+{(v_i,r_j)};
L15.   ε←ε-{v_i};
L16.   ρ← ρ-{r_j};
       //将处理任务的计算资源从空闲队列中去除
L17.   t_s(v_i)=max(t_s(v_i),τ_idle(p_j));
L18.   t_e(v_i)= t_s(v_i)+w(v_i)/o(p_j);
L19.   τ_idle(p_j)=t_e(v_i);
L20.   for each immediate successor v_x of task v_i
L21.  { v_x=fpred(v_i);//v_x 是 v_i 的贵宾前驱
L22.   if w(V_a)+w(v_x)<=ηo(p_j) V_a←{v_x}∪{v_a};
         //将贵宾前驱加入簇中
L23.     indegree(v_x)=indegree(v_x)-1;
L24.   if indegree(v_x)=0 then ε←ε+{ v_x};
L25.    if w(V_a)+w(v_x)>ηo(p_j) a=a+1;
          //超过给定比例进入下一个分簇
        }
L26.   ρ←ρ+{ p_j};T(p_j)=t_e(v_x)-t_s(v_i);
          //处理器和其对应的簇分配完毕
        } //end for each v_i∈ε
      } //end do
L27. T_total=max{t_e(v_i),1≤i≤n};// 任务完成时间
L28. U=∑T(P_k)/max{ T(P_k)} * m;//体系结构利用率
L29. P=U/T;//体系结构的效能;
}
```

上面 L11－L26 中的 for each 循环是针对并行任务的子任务的，所以要循环 n 次。内层为子任务选择资源节点(L12)基于贪婪算法思想，在空闲资源中选择一个资源 p_j 使得计算时间接近单位 1，平均需要 $|m|/2$ 次。L21－L25 处理任务节点 v_i 的后继的时间复杂性是 O(1)。故分簇算法 APC

( )的执行时间复杂度为 O(|V||P|)。

5 模拟实验及仿真实验

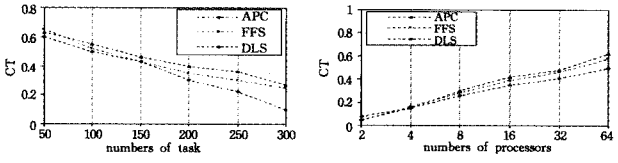
5.1 评价指标及实验环境

这里采用 3 种评价指标进行分析，即通信时间与执行时间之比值 CT（其中通信时间为 $\sum_{(V_i,V_j)\in E'}\sum_{(M(V_x),M(V_y))\in l}\frac{C(V_i,V_j)}{q(M(V_x),M(V_y))}$）任务执行时间 T 和系统利用率 U。CT 是在衡量考虑结构特征后系统通信性能是否可以改善。任务执行时间是并行计算问题的基本点。利用率是系统效能衡量指标之一。

本实验平台是由澳大利亚墨尔本大学开发的网格模拟器 GridSim5.0 进行实验，GridSim 能够模拟生成随机任务关系图及建立异构体系结构。并行任务图的相关参数包括：任务节点个数、计算任务量、任务间依赖关系及通信量；异构体系结构相关参数包括：处理器个数、处理器能力、网络拓扑关系及处理器之间的通信能力。并行任务图的大小控制在最小 50 个节点、300 条边和最大 300 个节点和 3000 条边。体系结构 UAG 也是采用随机产生的，包括处理器个数、每个处理器的能力及处理器间的通信带宽。体系结构图的大小控制在最小 2 个节点、1 条边到最大 64 个节点、500 条边。在实验中，当并行任务中任务个数及任务量发生变化时，选择处理器个数为 8、边数为 16 的体系结构。当体系结构的处理器个数及处理能力发生变化时，选择 160 个节点、300 条边的并行任务。每组实验分别进行 10 次，最终结果采用平均值。时间采用单位处理时间。

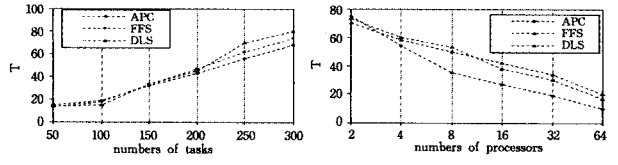
5.2 结果与实验分析

本实验通过改变计算任务个数及任务量、处理器个数及处理能力，来对算法的通信开销与执行时间之比、执行时间和系统利用率进行比较。为评价本文所提算法的性能，本文还将该算法与典型异构计算 DLS 算法[10]和 FFS 算法[12]在不同情况下进行比较。图 4 和图 5 所示为不同任务节点、不同处理器数分簇算法的性能比较。

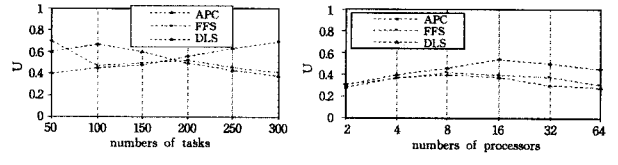


(a) 不同节点数 3 种算法的 CT 比较　(b) 不同处理器数 3 种算法的 CT 比较

图 4　3 种算法在不同情况下 CT 的比较



(a) 不同节点数下 3 种算法的执行时间(b) 不同处理器下 3 种算法的执行时间



(c) 不同节点数下 3 种算法的系统利用率　(d) 不同处理器下 3 种算法的系统利用率

图 5　APC 的执行时间和系统利用率比较

5.2.1 APC 算法结构匹配的有效性

由图 4 可知，当体系结构固定时，随着任务节点个数的增加，3 种算法的通信时间与完成时间之比也逐步减小。在任务节点较多时，APC 比其他两种算法下降得更快。当并行任务固定时，随着处理器节点个数的增加，3 种算法的通信时间与完成时间之比逐步增加。APC 比其他两种算法增加得更慢一些。这说明在考虑了通信结构后的 APC 算法可以大大减少通信开销在整个任务执行时间所占的比例。

5.2.2 APC 算法的执行时间 T 及系统利用率 U

由图 5 可知，当体系结构固定时，随着任务节点个数的增加，3 种算法的执行时间逐步增加，当任务节点个数较小时，FFS、DLS 算法较优于 APC 算法，随着任务个数的增大，APC 算法优于 FFS、DLS 算法。当并行任务固定时，随着处理器节点个数的增加，3 种算法的执行时间逐步减少。APC 比其他两种算法减少得更快。当体系结构固定时，随着任务节点个数的增加，FFS 和 DLS 算法的处理器利用率不断变化，而 APC 随着任务的增多利用率逐步增大。当并行任务固定时，随着处理器节点个数的增加，处理器的利用率都出现了先增后减的局势。分析原因如下。

本文提出的 APC 算法考虑了计算能力、结构特征，因此能够从全局考虑体系结构特征，包括处理器计算能力及通信结构特征。随着计算任务和处理器个数的增加，APC 的算法优越性逐步明显，主要体现在通信开销与执行时间比、算法执行时间及处理器利用率上。

通过上述实验分析表明：(1)本文提出的在异构计算中体系结构感知的分簇算法的通信开销与执行时间比大大减小。当任务和处理器个数较小时，算法的性能改善较小，但随着任务和处理器的增加，体系结构感知算法以小的时间花费为代价，获得了较小的通信代价，并减少了整个任务的运行时间。(2)本文提出的体系结构感知分簇算法的性能依赖于体系结构节点的数目和应用任务的大小。随着网格节点数或应用任务的增加，分簇算法在通信方面增加的性能是时间花费代价的几倍，突出了本文所提出的算法在大规模环境下对于大规模工程科学任务的实用性。(3)任何条件下，任务在时间花费和通信开销方面存在权衡/制衡，不可能同时达到两方面的高质量服务，通常是以一方为代价换取另一方更高的服务需求。

结束语　本文针对异构计算特点及高效能要求，提出了异构计算中效能的概念，给出了计算、通信结构异构匹配实现原理、异构匹配与高效能之间的关系及实现高效能的理论依据及相关证明。在此基础上给出了基于体系结构感知的并行任务分簇方法，其将并行任务与体系结构特征相结合来实现分簇，最大限度地发挥异构系统的异构性能，从而实现高效能。这种方法最大的特点是符合客观现实，并且并行任务特点与体系结构特点相结合实现分簇，可以最大化地利用处理器处理能力和体系结构的结构特征，以便节省硬件资源，提高资源利用率，从而减少了不必要的开销。这种思想为异构计算提供理论依据。但是目前在如何实现并行任务与体系结构异构匹配判定理论方面仍然有待改进，这也是我们今后研究的重点。

参 考 文 献

[1] Gerasoulis A, Yang T. A Comparison of Clustering Heuristics for Scheduling DAGs on Multiprocessors[J]. Journal or Parallel

and Distributed Computing,1992,12(16):276-289
[2] Baxter J, Patel J H. The LAST algorithm: a heuristic-based static task allocation algorithm[C]//Proceedings of the 1989 International Conference on Parallel Processing. 1989:217-222
[3] McCreary C, Gill H. Automatic determination of grain size for efficient parallel processing[J]. Communication of ACM, 1989, 32(9):1073-1078
[4] Yang Y, Chen K. Temporal data clustering via weighted clustering ensemble with different representations[J]. IEEE Transactions on Knowledge and Data Engineering, 2011, 23(2):307-319
[5] Kim S J, Browne J C. A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures[C]//Proceedings of International'l Conference on Parallel Processing. 1988:1-8
[6] Wu M Y, Gajski D D. Hypertool: A programming aid for message-passing systems[J]. IEEE Transations on Parallel and Distributed Systems, 1990, 1(3):330-343
[7] Yang T, Gerasoulis A. DSC: scheduling parallel tasks on an unbounded number of processors[J]. IEEE Transactions on Parallel and Distributed Systems, 1994, 5(9):951-967
[8] Sarkar V. Partitioning and scheduling parallel programs for multiprocessors[M]. MIT Press, Cambridge, MA, 1989
[9] Lam Y M. Integrated task clustering , mapping and scheduling for heterogeneous computing systems[J]. International Journal of Computer Science&Information Technology, 2012, 4(1):127-146
[10] Sih G C, Lee E A. A Compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures [J]. IEEE Transactions On Parallel and Distributed System, 1993, 4(2):175-187
[11] Koziris N, Romesis M, Tsanaksa P, et al. An efficient algorithm for the physical mapping of clustered task graphs onto multiprocessor architectures[C]// Proceeding Eighth Eurmicro Workshop Parallel and Distributed Processing. 2000:406-413
[12] Bambha N K, Bhattacharyya S S. Joint Application Mapping/Interconnect Synthesis Techniques for Embedded Chip-Scale Multiprocessors [J]. IEEE Transactions On Parallel and Distributed System, 2005, 16(2):99-112

(上接第 88 页)

询效率比传统的方法高很多。K-means 算法之所以会比传统算法效率高，是因为其对案例库进行了聚类。DRR 算法采用二级检索的方式，能够快速地锁定与目标案例接近的案例聚类，从而在效率上比传统算法和 K-means 算法都要快得多。传统检索方式每次都将所有数据读取内存进行查询比较，随着案例空间的数据逐渐增长，查询数据的时间也逐步增长，当数据到达一定数量时则会发生内存溢出。

表 2 几种算法检索效率比较表

案例库大小	传统算法(ms)	K-means(ms)	DRR(ms)
50000	29.33	15	11
80000	31.78	16	13
100000	37	16	15
200000	67.67	51.67	36.67
300000	93.33	73.25	53
500000	内存溢出	内存溢出	235

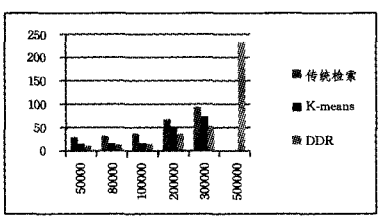


图 5 查询结果柱状图

K-means 和 DRR 算法将案例库中的数据读入，建立索引，以后的每次查询都是基于索引的查询，因此查询会比较快；K-means 索引建立的基础是整个案例库的数据，所以当数据到达一定数量的时候也会发生内存溢出；DRR 算法先对案例库中的记录通过降维的手段进行二维聚类，之后针对二维聚类结果建立 R 树索引，这样在内存中 R 树索引的空间比 K-means 算法的索引要小很多。因此在数据集很大的时候，DRR 算法依然可以实现查询，不仅不会出现内存溢出情况，而且能很好地解决 K-means 算法中由于样本点过大而造成的搜索效率下降问题，提高了大数据量案例库的检索效率。

结束语 基于 DRR 算法的案例检索算法将原案例库中的点降维计算映射成一个二维空间点表示，从而实现原案例库记录的二维空间聚类，再针对二维空间聚类建立 R 树索引；当前故障检索时，先通过 R 树索引找到与之最接近的案例二维聚类，再通过 KNN 算法进行精确的相似度计算，从而返回最接近新任务的案例。该算法通过降维聚类、二级检索的方法，加快了检索的效率，并且案例的二维聚类是基于业务无关的，避免了人工分类造成的分类错误。下一步工作将对算法进一步改进，考虑案例中特征项的权重对检索的影响，以提高案例检索的准确率和效率。

参 考 文 献

[1] Aamodt A, Plaza E. Case—based reasoning: Foundational issues, methodological variations, and system approaches[J]. AI Communications, 1994, 7(1):39-59
[2] Gu Yin-shan, Hua Qiang, Zhan Yan, et al. Case-base maintenancebased on representative selection for I-NN algorithm[C]// 2003 International Conference on Machine Learning and Cybemetics. 2003:2421-2425
[3] Derere L. Case-based reasoning: Diagnosis of faults in complex-systems throughreuse of experience[C]// Proceedings of InternationalTest Conference. 2000:77-105
[4] 耿焕同，肖明军，邹翔，等. 聚类算法在范例库维护中的应用研究明[J]. 计算机工程，2005，31(12)：166-168
[5] 冯征. 一种基于粗糙集的 K-Means 聚类算法[J]. 计算机工程与应用，2008，44(20)：141-142
[6] 刘长征，董冬. 基于特征加权 C 均值聚类算法的案例索引和检索[J]. 计算机应用与软件，2010，27(2)：111-114
[7] 乔丽，姜慧霖. 一种 k-means 聚类的案例检索算法[J]. 计算机工程与应用，2011，47(4)：185-187
[8] Guttman A. R-Trees: A dynamic index structure for spatialsearching[C]// Proc. ACM S IGMOD Conf, Ann. Meeting. 1984:47-57