异构计算中的时间和能耗优化执行方法

俞莉花[1] 曾国荪[2]
(同济大学计算机科学及技术系 上海 201804)[1]
(国家高性能计算机工程技术中心同济分中心 上海 201804)[2]

摘 要 计算环境的异构性以及应用任务的复杂多样性导致异构计算的必要性。异构计算的目的是重视并行处理系统和计算任务的差异,寻求系统和任务的有效匹配,从而获得并行任务在系统上执行的最佳效果。当前,异构计算中的时间优化执行方法较成熟,但同时将时间和能耗联合起来作为异构计算优化执行目标方面的研究很少。以高性能计算和绿色计算为总目标,针对异构计算环境中并行任务分配调度执行问题,提出了异构任务模型、异构计算速率矩阵、异构计算功率矩阵,利用能耗时间归一思想,给出并行任务在异构处理机上时间与能耗启发式优化执行算法,并通过实例分析证实算法的可行性和有效性。

关键词 异构计算,任务执行,时间优化,能耗优化
中图法分类号 TP338 **文献标识码** A

Executing Method of Time and Energy Optimization in Heterogeneous Computing

YU Li-hua[1] ZENG Guo-sun[2]
(Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)[1]
(Tongji Branch, National Engineering & Technology Center of High Performance Computer, Shanghai 201804, China)[2]

Abstract Both the heterogeneity of the computing environment and the complexity of various application tasks lead to heterogeneous computing. The purpose of heterogeneous computing is to obtain the best executing effect of the parallel task running in the processing system by putting emphasis on the difference between the parallel system and the task and exploring the optimal match between the system and the task. Currently, in heterogeneous computing, the scheduling method only for time optimization is quite mature, but the research on the executing method both for time and energy optimization is very few. This paper aimed at the high performance computing and green computing, and payed more attention to the scheduling problem of parallel task in heterogeneous computing environment. We proposed the heterogeneous task model, the heterogeneous computing velocity matrix and the heterogeneous computing power matrix. And making use of the idea that energy can be unified time, this paper presented heuristic executing algorithms to achieve both time and energy optimization for parallel task on heterogeneous system. Finally, a case study shows the feasibility and efficiency of proposed algorithms.

Keywords Heterogeneous computing, Task executing, Time optimization, Energy optimization

1 引言

高性能并行计算一直是计算机科学技术领域的研究热点。由于并行计算系统中存在硬件和软件的差异,以及应用任务的多样性,因此,异构计算正在成为高性能计算的重要发展方向。在异构处理器系统中,一方面,可以根据不同任务的不同特征,选择合适的处理器,实现不同处理器的优势互补,从而能够以非常短的时间完成任务的执行。另一方面,在不断追求计算高性能以获得强大计算能力的过程中,大量的能量消耗已成为计算机发展中亟待解决的问题。以超级计算机为例,一台百万亿次级超级计算机的系统功耗在1000kW左右,而一台千万亿次超级计算机系统的功耗已达到数兆瓦,每年的电费开销高达数千万元。除了消耗大量的电力能源外,还带来了严重的环境影响,包括大气质量恶化等。全球权威技术调查机构Gartner指出,全球二氧化碳排放量2%来自于IT业。可见,绿色计算作为一种可持续发展的低成本、低能耗的新型计算系统、模型和应用的研究,已成为未来信息技术领域面临的重大挑战。信息技术领域的研究必须综合考虑性能与能耗问题,以响应全球"节能减排"的伟大倡议。

在异构计算中,异构任务的执行采用不同的任务分配与调度算法,会产生不同效果。寻找最优的调度方案已被证明是NP完全问题[1],国内外学者对此展开了大量的研究,以便

到稿日期:2010-11-16 返修日期:2011-02-25 本文受863项目(2009AA012201),973计划课题(2007CB316502),国家自然基金项目(90718015),NSFC-微软亚洲研究院联合资助项目(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。

俞莉花(1987—),女,硕士生,主要研究领域为异构重构计算;曾国荪(1964—),博士,教授,博士生导师,主要研究领域为异构计算、信息安全。

寻找一种次优的启发式任务执行方法。目前常用的优化技术有遗传算法[2,3]、模拟退火算法[4]、A * 算法[5]等。1994 年，Edwin 提出了一种遗传算法在同构系统中的通用调度算法[2]。2000 年，国防科学技术大学的陈火旺院士提出了遗传算法在异构环境下的任务分配与调度的进化算法[3]。2006 年，湖北大学的周双娥等人结合模拟退火方法提出了一种启发式任务调度算法[4]。但这些方法都只是考虑任务完成时间为优化目标。2010 年，湖南大学的彭曼曼教授[6]进行了异构多核处理器的任务分配及能耗的研究，提出了一种任务分配调度算法来节省能耗，但是这个方法只单独考虑了能耗优化。可见，在目前并行任务分配调度算法中，几乎都是时间或能耗方面的单独优化方案，很少综合考虑时间和能耗的优化执行。最近，武汉理工大学的李春林教授[7]提出了一种平衡时间和能耗的算法，来解决并行任务调度执行问题，但该算法仅考虑了限制执行时间条件下如何获得较少能耗的情况，没有从根本上解决时间和能耗同时优化的问题。

本文以高性能计算和绿色计算为总目标，重视和利用异构计算环境的差异和特性，探索并行任务在异构处理机上的时间与能耗启发式优化执行方法。

2 异构计算与异构任务模型

2.1 异构计算

所谓异构计算(heterogeneous computing)是指将性能各异的计算机(如 PC、工作站群、向量机、SIMD、MIMD 及专用机)通过高速网络连成并行计算环境，充分利用程序和结构的异构性，各尽所能，合理分治，协同计算一个应用任务，使得完成时间最小以及期望目标最优的过程[8]。

上面所说的异构性在计算机系统中表现为两方面：(1)应用程序代码的异构性，也称为计算需求或计算类型，对并行处理来说，一般认为存在着向量、超标量、数据流、systolic、对象等并行计算类型；(2)体系结构的异构性，也称为执行模式，主要体现在现有的计算机型号千差万别，即使在同构机中也有多种系列。既然程序和体系结构都存在异构性、多样性，必然导致应用任务的异构执行会产生不同的完成时间和不同的能耗。因此，我们应该正视异构性的客观存在，正视其合理性，扬长避短，优势互补，主动寻求异构计算提高计算性能及降低能耗的新途径。

异构计算中，通常把由不同处理机组成的计算环境称为异构系统，用集合 $P=\{p_1,p_2\cdots p_n\}$ 来表示，其中 $p_i\neq p_j(i\neq j)$。在本文中，处理器的异构性主要体现在处理器执行任务的速度不同、功率不同。同样，把将在异构系统中执行的并行应用程序叫异构任务。

2.2 异构任务模型

在并行计算中，一般用无环有向图 DAG 来表示并行任务[9]。本文仍用 DAG 图，但对其进行扩展，以此来描述异构任务。定义一个异构任务为一个六元组 HT-DAG=(N,R,W,C,PW,V)，其中各元组的具体含义如下：

(1)N 是顶点集合$\{n_i\}$，表示一个异构任务中包含的所有子任务，n_i 表示第 i 个子任务。

(2)R 是有向边的集合$\{r_{ij}\}$，表示子任务间执行顺序先后关系即子任务间的偏序关系，以及数据依赖关系。

(3)W 是子任务的计算量集合$\{w_i\}$，w_i 表示子任务 n_i 的计算量或工作负载，例如子任务的总指令数或运算量等。

(4)C 是子任务间的通信量集合$\{c_{ij}\}$，c_{ij} 表示子任务 n_i 到子任务 n_j 的通信量，它完成子任务之间的数据传送。假设用 b_{ij} 来表示子任务 n_i 与子任务 n_j 通信时对应的通信带宽，b_{ij} 由通信链路和发送/接收处理器共同决定，那么计算 c_{ij}/b_{ij} 可得出子任务 n_i 与子任务 n_j 的通信开销。

(5)V 是子任务在异构系统中不同处理器上执行的速率矩阵$[v_i]$，$v_i=\{v_{ij},1\leqslant j\leqslant n\}$表示子任务 n_i 在异构系统中所有各个处理器上执行时产生的不同速率所组成的向量，v_{ij} 表示子任务 n_i 在处理器 p_j 上执行时的速率。

(6)PW 是子任务在异构系统中的不同处理器上执行的功率(power)矩阵$[pw_i]$，$pw_i=\{pw_{ij},1\leqslant j\leqslant n\}$表示子任务 n_i 在异构系统中各个处理器上执行时处理器对应的不同功率组成的向量，pw_{ij} 表示任务 n_i 在处理器 p_j 上执行时的功率。

从静态来看，一旦并行任务及异构系统确定，那么异构任务的六元组属性都将固定不变。然而，异构计算中，不同的子任务按不同的方式来分配调度至不同的处理器上执行时，由于不同处理器的执行速率和执行功率不同，因此，整个任务的完成时间不同，产生的能耗也不同，即异构任务在异构处理器上执行的动态效果不一样。本文正是基于该六元组 HT-DAG=(N,R,W,C,PW,V)，探索异构计算的时间与能耗优化分析。

例 1　图 1 是一个 HT-DAG 任务图模型，一共包含 9 个子任务，每个子任务包含上述属性并存在一定的依赖关系。本文第 5 节还将用到此例子。

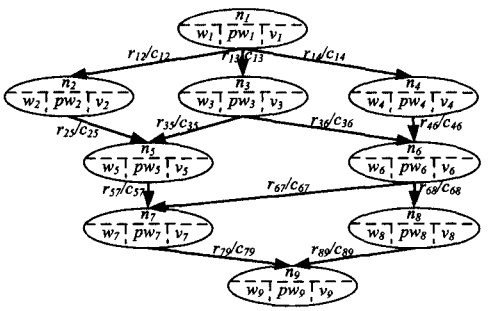

图 1　一个异构任务 HT-DAG 示例

2.3 计算任务与处理系统的关联

在异构计算中，由于异构系统存在性能和功耗迥异的处理器，因此异构任务安排至异构处理器上执行会有多种分配方法，而不同的分配方法会产生不同的计算性能和效果。可见，异构任务能否高效执行，不仅取决于处理部件的执行速度、并行算法的好坏，还取决于异构任务与异构系统的匹配程度。因为一个程序在某种处理器上运行效果好，不一定在另一种处理器上也好，相反，其效果可能很差。因此，研究整个异构计算的优化执行时，必然要考虑任务与处理器间的匹配情况。为了表达和刻画异构系统对异构任务执行的影响与匹配程度，本文引入异构计算执行速率矩阵和异构计算功率矩阵这两个度量矩阵。

2.3.1 异构计算速率矩阵

由于异构系统中处理器存在的差异，不同的子任务在不

同处理器上执行时的速率不同，为了刻画该执行速率差异，我们提出异构计算执行速率矩阵，如图 2 所示。

$$V_{m\times n}=\begin{array}{c} \\ n_1 \\ n_2 \\ \vdots \\ n_m \end{array}\begin{array}{c} \begin{array}{cccc} p_1 & p_2 & \cdots & p_n \end{array} \\ \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{m1} & v_{m2} & \cdots & v_{mn} \end{bmatrix} \end{array}$$

图 2　异构计算执行速率矩阵

其中，$P=\{p_1,p_2,\cdots,p_n\}$ 表示处理器集合；$N=\{n_1,n_2,\cdots,n_m\}$ 表示子任务集合；v_{ij} $(1\leqslant i\leqslant m,1\leqslant j\leqslant n)$ 表示子任务 n_i 在处理器 p_j 上的执行速率，其值可通过子任务在相应处理部件上实际执行，反复实验，测量完成时间，并对其进行统计计算来得到。一般情况下，如果 $i\neq x,j\neq y$，则 $v_{ij}\neq v_{xy}$。根据该矩阵 $V_{m\times n}$，结合子任务 n_i 的计算量 w_i，可计算得出单个子任务 n_i 在某处理器 p_j 上的执行完成时间 span：

$$\text{span}_{ij}=w_i/v_{ij} \tag{1}$$

2.3.2　异构计算功率矩阵

同样，为了刻画不同子任务在不同处理器上执行时处理器对应的不同功率，以便计算处理器执行任务时产生的能耗，我们提出异构计算功率矩阵，如图 3 所示。

$$PW_{m\times n}=\begin{array}{c} \\ n_1 \\ n_2 \\ \vdots \\ n_m \end{array}\begin{array}{c} \begin{array}{cccc} p_1 & p_2 & \cdots & p_n \end{array} \\ \begin{bmatrix} pw_{11} & pw_{12} & \cdots & pw_{1n} \\ pw_{21} & pw_{22} & \cdots & pw_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ pw_{m1} & pw_{m2} & \cdots & pw_{mn} \end{bmatrix} \end{array}$$

图 3　异构计算功率矩阵

其中，$P=\{p_1,p_2,\cdots,p_n\}$ 仍然表示处理器集合，$N=\{n_1,n_2,\cdots,n_m\}$ 仍然表示子任务集合，pw_{ij} $(1\leqslant i\leqslant m,1\leqslant j\leqslant n)$ 表示子任务 n_i 在处理器 p_j 上执行时该处理器对应的功率，其值可根据文献[11]提出的指令级功率模型得到。结合式(1)以及矩阵 $PW_{m\times n}$，子任务 n_i 在某处理器 p_j 上执行时产生的能耗为：

$$e_{ij}=pw_{ij}\times\text{span}_{ij}=pw_{ij}\times(w_i/v_{ij}) \tag{2}$$

根据式(2)可知，对于单个子任务来说，w_i 是固定的，所以它所产生的处理器能耗仅仅取决于子任务在某处理器上执行时该处理器的功率及执行速率。

3　异构计算中时间与能耗优化问题描述

在异构计算中，已知：(1)异构系统 $P=\{p_1,p_2,\cdots,p_n\}$；(2)异构任务 $N=\{n_1,n_2,\cdots,n_m\}$，各个子任务用上节中的 HT-DAG 描述成任务图；(3)异构计算速率矩阵 $V_{m\times n}$；(4)异构计算功率矩阵 $PW_{m\times n}$。在这些前提条件下，异构任务按不同顺序和方式调度至异构处理器上执行，完成的总时间以及处理器所消耗的能量都会不同。根据式(1)和式(2)可得总执行时间为 $T_{\text{total}}=\max\{t_{\text{end}}(i)\}$，$1\leqslant i\leqslant m$，其中 $t_{\text{end}}(i)$ 表示子任务 n_i 的执行结束时间；总能耗为 $E_{\text{total}}=\sum e_{ij}=\sum pw_{ij}\times(w_i/v_{ij})$。

然而在异构计算过程中，整个任务获得最短的执行时间，并不意味着处理器产生最少的能量消耗。同样，处理器产生最少的能量消耗，也不意味任务获得最短的执行时间。所以异构计算最理想的目标是同时满足执行时间最小且能耗最小，即异构计算中时间和能耗调度优化的目标函数为：

(1) $T_{\text{opt}}=\min\{T_{\text{total}}\}$；

(2) $E_{\text{opt}}=\min\{E_{\text{total}}\}$。

一般情况下，解决这类优化问题是困难的。通常是先针对其中一个变量限定一个范围，在该范围内讨论另一目标变量的最优化。因此，实际应用中，时间与能耗优化问题就有如下两种求解方式：

(1)时间一定的情况下，求能耗最小的任务调度执行方式，即 $E_{\text{opt}}=\min(E_{\text{total}})$，满足 $T_{\text{total}}\leqslant T_{\text{expected}}$；

(2)能耗一定的情况下，求时间最短的任务调度执行方式，即 $T_{\text{opt}}=\min\{T_{\text{total}}\}$，满足 $E_{\text{total}}\leqslant E_{\text{expected}}$。

下面针对第一种情况，使用示例 2 做出说明。

例 2　以图 4 所示异构任务图为例，该任务 $N=\{n_1,n_2,n_3,n_4,n_5\}$，假设在异构系统 $P=\{p_1,p_2,p_3\}$ 上完成任务执行。其中，每个子任务对应的计算量假设为 $W=\{24,120,24,30,60\}$，对应的异构计算速率矩阵和异构计算功率矩阵如图 4 所示。子任务间的通信量假设为 $C=\{c_{12},c_{23},c_{24},c_{35},c_{45}\}=\{4,2,2,2,2\}$，为了简化通信，假设不同处理器间的通信带宽 b 都一样，设为 $b=1$。整个任务执行完成时间限制为 22 个时间单位。

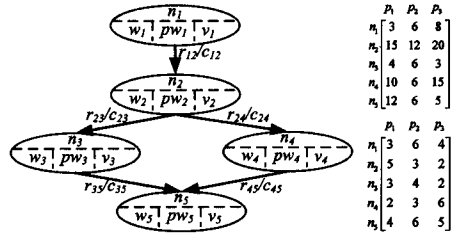

图 4　左边为异构任务图模型，右上为该任务计算速率矩阵，右下为该任务计算能耗率矩阵

基于以上数据，根据直觉经验，一种总执行时间最短的调度执行方案如图 5 所示。

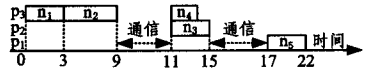

图 5　任务调度执行顺序

此时总的执行完成时间包括计算时间及通信时间，为 22 个时间单位，正好满足时间限制要求。此时总能耗为 $E=3\times4+6\times2+4\times4+2\times6+5\times4=72$ 能量单位。按这种调度方案，子任务 n_4 在处理器 p_3 上执行，用时为 2，能耗为 12。但如果将子任务 n_4 分配至处理器 p_1，用时为 3，但能耗降为 $3\times2=6$，且并没有增加总执行完成时间。这样，可以把子任务 n_4 调度到处理器 p_1 上，以减少执行所需的能耗。

同样，如果给定整个异构任务执行的能耗限制，寻求时间最少的任务分配执行方案也存在类似的情况。

从例 2 可看出，为了高效地完成异构计算，就必须开发利用异构性和多样性，让不同性能的处理器扬长避短、优势互补，从而提高计算性能并降低能耗。所以，异构任务合理分配调度是改善异构计算效能的一个重要方法。

4　时间和能耗优化执行的可行方法

异构计算中，任务执行过程中的时间与能耗优化需求实

质上是一个双目标优化问题。然而，与单目标优化问题相比，多目标问题对最优解的定义通常很复杂，它的解不明确，很难客观地评价解的优劣性。因此，我们考虑将时间与能耗这个双目标优化问题，通过利用归一思想转为单目标问题。

4.1 能耗和时间的归一处理

在异构任务调度决策执行过程中，如果只是希望任务的总执行时间最短，那么采取“运行时间最短的处理器与子任务相匹配”的策略即可。该做法实质上是解决以时间为单目标的优化问题，并且具有成功求解算法[3]。然而在本文提出的异构任务模型中，既涉及完成时间，又涉及能耗问题，必须综合考虑时间与能耗两者间的联系，使得异构任务更有效地执行。我们知道，既考虑时间又考虑能耗的多目标优化问题并不是那么简单，为了简化问题的求解，并注意到时间与能耗又存在着一定的必然联系，不妨假定某一子任务在某一处理器上执行时产生的时间与能耗存在某种函数关系，例如线性关系，那么使用能耗对时间归一因子，将能耗参数统一转化为时间参数。这样，既综合考虑时间与能耗属性，又变成了单目标优化问题，从而方便求解。

基于这一思想，同时考虑异构处理器产生能耗的差异性以及能耗的实际情况，引入能耗时间转化因子矩阵 $\alpha_{m\times n}$，描述同一子任务在不同处理器上执行对应的能耗时间转化因子各不相同，如图 6 所示。

$$\alpha_{m\times n}=\begin{array}{c} \\ n_1 \\ n_2 \\ \vdots \\ n_m \end{array}\begin{array}{c} \begin{array}{cccc} p_1 & p_2 & \cdots & p_n \end{array} \\ \begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m1} & \alpha_{m2} & \cdots & \alpha_{mn} \end{bmatrix} \end{array}$$

图 6 能耗对时间归一因子

其中，α_{ij} 表示子任务 n_i 在某处理器 p_j 上执行时对应的能耗与时间的转化因子。假设产生的能耗为 e_{ij}，执行完成时间为 span_{ij}，则可使用如下式子把能耗属性转为与时间相关属性：

$$\text{span}'_{ij}=\alpha_{ij}\times e_{ij} \tag{3}$$

根据式(3)，将能耗与时间进行归一处理之后，子任务的执行完成时间可以“广义近似”看成为 $\text{span}''_{ij}=\text{span}_{ij}+\text{span}'_{ij}$。例如，假设子任务 n_1 在处理器 p_1 上执行时，时间为 10，能耗为 20，且子任务 1 在处理器 1 上的能耗时间转化因子为 0.5，那么扩展后的执行完成时间 $\text{span}''_{ij}=\text{span}_{ij}+\text{span}'_{ij}=10+0.5\times 20=20$。

很显然，从式子 $\text{span}''_{ij}=\text{span}_{ij}+\text{span}'_{ij}=\text{span}_{ij}+\alpha_{ij}\times e_{ij}$ 可知，由于异构任务 HT-DAG 中不同属性的存在，任一子任务 n_i 选择不同的处理器执行时，对应的 span_{ij}、α_{ij} 和 e_{ij} 都会随之改变。

4.2 时间和能耗优化执行的贪婪算法

基于 4.1 节的能耗时间归一处理思想，重点关注每个子任务扩展处理后的执行完成时间，提出如下异构计算中时间与能耗优化执行的贪婪算法。

算法 1 时间归一贪婪执行算法(Time_Unify_Greed_Sch-eduling())

输入：HT-DAG 任务图 G，子任务间通信链路的带宽集合 $B=\{b_{ij}\}$，速率矩阵 $V_{m\times n}$，功率矩阵 $PW_{m\times n}$，能耗与时间转化因子矩阵 $\alpha_{m\times n}$，其中，m 为子任务个数，n 为处理器个数

输出：整个任务执行时间 T_{total}，能耗 E_{total}，子任务处理器分配序列 Assign=$\{(n_i,p_j)\}$

```
{
s1 L←{ n_i | indegree(n_i)=0,1≤i≤n};// HT-DAG 图中，入度为零的子任务进入子任务准备队列 L
s2 ρ←{ p_1,p_2,…,p_n} //空闲处理器集合
s3 Assign←ϕ; //子任务处理器分配序列
s4 ε←ϕ; //子任务执行队列初始化为空
s5 t_s(n_i)=0,t_e(n_i)=0,1≤i≤m; //所有子任务的开始和结束执行时刻初始化为 0
s6 τ_idle(p_j)=0,1≤j≤n;//处理器空闲等待时刻初始化为 0
s7 T_total=0,E_total=0; //初始化任务执行时间和能耗
s8 for(i=1;i≤m;i++) //将能耗归一转为时间
s9     for(j=1;j≤n;j++)
s10        span'_ij=α_ij×pw_ij×(w_i/v_j);
s11 do until L=ϕ
s12 { for each idle processor p_j∈ρ
    //基于贪婪算法思想，选择“广义近似”完成时间最短的子任务和处理器配对(n_i,p_j)
s13        (n_i,p_j)←select_n_p(min{w_i/v_ij+ span'_ij,…});
s14        Assign←Assign+(n_i,p_j);
s15        ε←ε+{n_i};
s16        ρ←ρ({p_j}; //将正在处理任务的处理器从空闲队列中去除
s17        L←L({n_i};
s18        t_s(n_i)=max(t_s(n_i),τ_idle(p_j) );
s19    for each executable task n_i∈ε
s20        p_j←processor(n_i); //取得子任务 n_i 对应的处理器 p_j
s21        ε←ε({n_i};
s22        ρ←ρ+{p_j}; //将已处理完子任务的处理器加入空闲队列中
s23        e_ij= pw_ij×w_i/v_ij;//计算出子任务 n_i 此时执行时对应能耗
s24        E_total=E_total+e_ij; //更新任务总能耗
s25        for each immediate parent task n_k of task n_i
s26            T_comm(k,i)=c_ki/b_ki; //计算子任务 n_i 与父任务 n_k 间的通信时间
s27        T_comm(n_i)=max{T_comm(k,i),n_k∈{parent tasks of task n_i} );
s28        t_e(n_i)=t_s(n_i)+ w_i/v_ij+T_comm(n_i);
s29        τ_idle(p_j)=t_e(n_i);
s30        for each immediate successor n_x of task n_i
s31            t_s(n_x)=max(t_s(n_x),τ_idle(p_j) );
    //考虑 HT-DAG 中数据依赖关系
s32            indegree(n_x)=indegree(n_x-1;
s33            if indegree(n_x)=0
s34                L←L+{n_x};
    }
s35 T_total=max{t_e(n_i),1≤i≤m};
}
```

从语句 s13 可以看出，本算法使用子任务的“广义近似”执行完成时间选择处理器与之匹配，进而得到所有子任务的调度序列以及整个并行任务的总执行时间和总能耗。

4.3 算法分析

4.3.1 时间优先算法

一个并行应用程序，有时候特别关注执行完成时间，例如实时系统、天气预报服务等。也就是说，完成时间是优先考虑的因素，而不太计较产生的能耗。此时，可对上述 Time_Unify_Greed_Scheduling 算法进行改造，得到如下时间优先的执行完成时间最小的任务分配调度方案。

算法 2 时间优先执行算法(Time_First_Scheduling())

算法描述类似算法 1，在此省略。关键差别在于算法 2 无需算法 1 中的语句 s8 到 s10 的归一处理，以及语句 s13 选择与处理器匹配的子任务的决策标准变成时间优先，将 s13 改为$(n_i,p_j)\leftarrow$select_n_p($\min\{w_i/v_{ij},\cdots\}$)。由于时间优先算法只考虑任务的执行完成时间 w_i/v_{ij}，因此任务执行的完成时间可能最小，但是任务执行的能耗可能会很大。

4.3.2 能耗优先算法

一个并行应用程序，有时候特别关注机器产生的能耗，以便降低计算中心的运行和维护费用。也就是说，能量消耗是优先考虑的因素，而不太计较执行完成时间。此时，可对上述 Time_Unify_Greed_Scheduling 算法进行改造，得到如下能耗优先的能耗最小的任务分配调度方案。

算法 3 能耗优先执行算法(Energy_First_Scheduling())，同样，算法描述类似算法 1，在此省略。关键差别也是在于算法 3 无需算法 1 中的 s8 到 s10 的归一处理，以及 s13 选择的与处理器匹配的子任务决策标准变成能耗优先，即将 s13 改为$(n_i,p_j)\leftarrow$select_n_p($\min\{p_{ij}\times w_i/v_{ij},\cdots\}$)。所以，对于能耗优先的算法，优先考虑任务执行的能耗 $p_i\times w_i/v_{ij}$，那么任务执行的能耗可能最小，但是任务的执行完成时间可能会很大。

4.3.3 综合比较

从算法复杂度方面分析，时间归一贪婪执行算法 Time_Unify_Greed_Scheduling、时间优先算法 Time_First_Scheduling、能耗优先算法 Energy_First_Scheduling 这 3 个算法的关键区别在于子任务与处理器匹配时的优先考虑策略和目标不同，这在 4.3.1 节和 4.3.2 节中已做出详细说明。显然，算法 1(Time_Unify_Greed_Scheduling)并不会因为能耗时间的归一处理而明显提高该算法的复杂度。总的来说，这 3 个算法的算法复杂度相似，都为 $O(m^2)$，其中 m 为子任务个数。

仅从执行时间上考虑，对于本文第 3 节提出的异构计算中时间与能耗优化问题，暂且不关心它是 NP 难题，理论上总存在整个任务执行完成的最短或最长时间，分别记为 T_{shortest}，T_{longest}。另外，如果采用算法 1(Time_Unify_Greed_Scheduling)，记整个任务执行完成时间为 $T_{\text{total}}^{(1)}$。如果采用算法 2(Time_first_Scheduling)，记整个任务执行完成时间为 $T_{\text{total}}^{(2)}$。如果采用算法 3(Energy_First_Scheduling)，记整个任务执行完成时间为 $T_{\text{total}}^{(3)}$。显然，这 3 个时间处于最短时间和最长时间之间，即 $T_{\text{shortest}}\leqslant T_{\text{total}}^{(1)},T_{\text{total}}^{(2)},T_{\text{total}}^{(3)}\leqslant T_{\text{longest}}$，可见这 3 种算法都在一定程度上完成了时间优化。

仅从执行能耗方面考虑，对于本文第 3 节提出的异构计算中时间与能耗优化问题，暂且不关心它是 NP 难题，理论上同样也总存在整个任务执行产生的能耗最小值或最大值，分别记为 $E_{\max}$，$E_{\min}$。另外，如果采用算法 1(Time_Unify_Greed_Scheduling)，则记整个任务执行能耗为 $E_{\text{total}}^{(1)}$。如果采用算法 2(Time_first_Scheduling)，则记整个任务执行能耗为 $E_{\text{total}}^{(2)}$。如果采用算法 3(Energy_First_Scheduling)，则记整个任务执行能耗为 $E_{\text{total}}^{(3)}$。显然，这 3 个能耗值处于最大能耗和最小能耗之间，即 $E_{\min}\leqslant E_{\text{total}}^{(1)},E_{\text{total}}^{(2)},E_{\text{total}}^{(3)}\leqslant E_{\max}$，可见这 3 个算法都在一定程度上都完成了能耗优化。

综上，这 3 个算法都从不同程度解决了异构计算中时间与能耗优化问题，实现了时间与能耗优化。比较这 3 个算法本身，算法 2(Time_first_Scheduling)得出的任务执行完成时间 $T_{\text{total}}^{(2)}$ 在一般情况下被认为是任务执行完成的最短时间，那么可以得到 $T_{\text{total}}^{(2)}\leqslant T_{\text{total}}^{(1)}\leqslant T_{\text{total}}^{(3)}$。同样，算法 3(Energy_First_Scheduling)得到的任务执行能耗在一般情况下被认为是任务最小执行能耗，可得 $E_{\text{total}}^{(3)}\leqslant E_{\text{total}}^{(1)}\leqslant E_{\text{total}}^{(2)}$。由此可得，算法 1 给出的任务执行调度方案，任务的总执行完成时间处于算法 2 和算法 3 两者之间，任务的总执行能耗也处于算法 2 和算法 3 之间，而 3 者的算法复杂度相近，可得算法 1 (Time_Unify_Greed_Scheduling)相比算法 2 和算法 3 而言是解决时间与能耗优化问题的一种可行且有效的方法。

5 实例分析

沿用第 2 节中的例 1，HT-DAG 图如图 1 所示，假设该任务 $N=\{n_1,n_2,n_3,n_4,n_5,n_6,n_7,n_8,n_9\}$ 在异构系统 $P=\{p_1,p_2,p_3\}$ 上完成执行。其中，每个子任务对应的计算量设为 $W=\{24,120,24,30,60,90,36,96,30\}$，子任务间的通信量假设为 $C=\{c_{12},c_{13},c_{14},c_{25},c_{35},c_{36},c_{46},c_{57},c_{67},c_{68},c_{79},c_{89}\}=\{2,2,2,2,1,1,2,3,1,2,4,2\}$。为简化通信，假设异构系统内所有处理器间的通信带宽 b 都一样，设 $b=1$。异构计算速率矩阵、异构计算能耗率矩阵和能耗对时间归一因子矩阵为如图 7 所示。

$$V_{9\times3}=\begin{array}{c}\\ n_1\\ n_2\\ n_3\\ n_4\\ n_5\\ n_6\\ n_7\\ n_8\\ n_9\end{array}\begin{array}{c}\begin{array}{ccc}p_1&p_2&p_3\end{array}\\ \begin{bmatrix}3&6&8\\15&12&20\\4&6&3\\10&6&15\\12&6&5\\9&10&6\\6&3&12\\6&16&8\\5&3&6\end{bmatrix}\end{array}\qquad PW_{9\times3}=\begin{array}{c}\\ n_1\\ n_2\\ n_3\\ n_4\\ n_5\\ n_6\\ n_7\\ n_8\\ n_9\end{array}\begin{array}{c}\begin{array}{ccc}p_1&p_2&p_3\end{array}\\ \begin{bmatrix}3&6&4\\5&3&2\\3&4&2\\2&3&6\\4&6&5\\5&6&15\\3&2&6\\6&16&12\\3&6&5\end{bmatrix}\end{array}$$

$$\alpha_{9\times3}=\begin{array}{c}\\ n_1\\ n_2\\ n_3\\ n_4\\ n_5\\ n_6\\ n_7\\ n_8\\ n_9\end{array}\begin{array}{c}\begin{array}{ccc}p_1&p_2&p_3\end{array}\\ \begin{bmatrix}0.2&0.3&0.1\\0.5&0.3&0.2\\0.3&0.4&0.2\\0.2&0.3&0.6\\0.4&0.6&0.5\\0.1&0.2&0.4\\0.3&0.2&0.1\\0.3&0.4&0.5\\0.3&0.4&0.2\end{bmatrix}\end{array}$$

图 7 上左为异构计算速率矩阵，上右为异构计算能耗率矩阵，下为能耗对时间归一因子矩阵

5.1 任务执行完成时间和能耗值的计算

根据不同的调度执行算法，异构任务执行完成时间和能耗值的计算结果如表1所列。

表1 5种调度执行算法得到的时间、能耗数值

任务执行调度方案	时间	能耗
时间归一贪婪执行算法	35	255
时间优先算法	27	265
能耗优先算法	36	248
最长时间调度方案	79	550
最大能耗调度方案	50	610

从表1可以看出时间归一贪婪执行算法、时间优先算法、能耗优先算法分别得到的时间值和能耗值都小于最长时间调度方案和最大能耗调度方案的时间值和能耗值，可见，这3个算法都以不同程度实现了时间与能耗优化。

分析表中的前3行数据可知，时间归一贪婪执行算法得出的时间值介于时间优先算法与能耗优先算法之间(27<35<36)，能耗值也介于时间优先算法与能耗优先算法之间(248<255<265)，所以时间归一贪婪执行算法是解决时间与能耗优化问题的一种可行且合理的方法。

5.2 能耗时间转化因子影响分析

不同的能耗时间转化因子可能对调度算法的效果影响很大，为了分析能耗时间转化因子在时间归一贪婪执行算法中所起的作用，仍然以图1为例，不妨选取小、中、大、随机4类转化因子，进行深入具体的分析。任意取值情况如图8所示，其他数据仍沿用上文。根据时间归一贪婪执行算法得到的异构任务执行完成时间和能耗值的数据如表2所列。

$$\alpha 1_{9\times3}=\begin{array}{c} \\ n_1\\ n_2\\ n_3\\ n_4\\ n_5\\ n_6\\ n_7\\ n_8\\ n_9 \end{array}\begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_3 \end{array}\\ \begin{bmatrix} 0.2 & 0.3 & 0.1\\ 0.5 & 0.3 & 0.2\\ 0.3 & 0.4 & 0.2\\ 0.2 & 0.3 & 0.6\\ 0.4 & 0.6 & 0.5\\ 0.1 & 0.2 & 0.4\\ 0.3 & 0.2 & 0.1\\ 0.3 & 0.4 & 0.5\\ 0.3 & 0.4 & 0.2 \end{bmatrix} \end{array}$$

$$\alpha 2_{9\times3}=\begin{array}{c} \\ n_1\\ n_2\\ n_3\\ n_4\\ n_5\\ n_6\\ n_7\\ n_8\\ n_9 \end{array}\begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_3 \end{array}\\ \begin{bmatrix} 2 & 3 & 1\\ 5 & 3 & 2\\ 3 & 4 & 2\\ 2 & 3 & 6\\ 4 & 6 & 5\\ 1 & 2 & 4\\ 3 & 2 & 1\\ 3 & 4 & 5\\ 3 & 4 & 2 \end{bmatrix} \end{array}$$

$$\alpha 3_{9\times3}=\begin{array}{c} \\ n_1\\ n_2\\ n_3\\ n_4\\ n_5\\ n_6\\ n_7\\ n_8\\ n_9 \end{array}\begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_3 \end{array}\\ \begin{bmatrix} 20 & 30 & 10\\ 50 & 30 & 20\\ 30 & 40 & 20\\ 20 & 30 & 60\\ 40 & 60 & 50\\ 10 & 20 & 40\\ 30 & 20 & 10\\ 30 & 40 & 50\\ 30 & 40 & 20 \end{bmatrix} \end{array}$$

$$\alpha 4_{9\times3}=\begin{array}{c} \\ n_1\\ n_2\\ n_3\\ n_4\\ n_5\\ n_6\\ n_7\\ n_8\\ n_9 \end{array}\begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_3 \end{array}\\ \begin{bmatrix} 0.2 & 3 & 10\\ 5 & 3 & 0.2\\ 30 & 0.4 & 2\\ 20 & 3 & 6\\ 4 & 0.6 & 5\\ 0.1 & 20 & 4\\ 3 & 0.2 & 10\\ 0.2 & 4 & 0.5\\ 30 & 0.4 & 2 \end{bmatrix} \end{array}$$

图8 4种不同的能耗转化因子

表2 4类转化因子对应能耗、时间对比

能耗转化因子不同数量级取值	时间	能耗
10^{-1}(α1)	35	255
100 (α2)	44	248
101 (α3)	55	295
随机(α4)	59	357

从表2可以看出：4类转化因子得出的总执行时间范围处于[27,79]，其中27为时间优先算法得到的总执行时间，79为执行时间最大值(表1中给出)。能耗范围处于[248,610]，其中248为能耗优先算法得到的执行能耗最小值，610为执行能耗最大值(表1中给出)。这再一次说明，时间归一贪婪执行算法是解决异构计算中时间与能耗优化问题的一种可行且有效的方法。特别地，当能耗对时间转化因子数量级为10^{-1}时，与其它3种数量级的转化因子相比，任务执行时间与执行能耗都是相对较小的。因此，数量级小于10^{-1}的能耗时间转化因子，在使用时间归一贪婪执行算法解决异构计算中时间与能耗优化问题时更加有效。

结束语 本文针对异构计算中的时间与能耗优化问题，提出了一种异构任务在异构处理机上时间与能耗启发式优化执行方法。使用有向无环图DAG对异构任务进行建模，形成HT-DAG。与传统DAG相比，HT-DAG加入时间与能耗两个异构属性：执行速率和执行功率，并给出了异构速率矩阵和异构功率矩阵作为两个度量矩阵。利用能耗时间归一处理思想，提出了一种时间归一贪婪执行算法，算法相比文献[2-5]中的时间或能耗单独优化算法，是一种综合考虑时间优化与能耗优化的可行且有效的解决方案。同时本文指出，时间优先算法和能耗优先算法都是时间归一贪婪执行算法的特殊情况。另外，本文给出了各种算法的时间与能耗数量大小的比较和分析，以及能耗时间转化因子影响分析。在以后的研究中，我们将深入分析能耗与时间的精确转化关系，开展模拟仿真实验，将本文的研究成果应用到云计算有效能耗管理服务中。

参考文献

[1] Kasahara H, Narita S. Practical multiprocessor scheduling algorithms for efficient parallel processing[J]. IEEE Transactions on Computers, 1984, 33(11): 1023-1029

[2] Hou E S H, Ansari N. A genetic algorithm for multiprocessor scheduling[J]. IEEE Transactions on Parallel and Distributed Systems, 1994, 5(2): 113-120

[3] 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

[4] 周双娥，雷辉. 基于改进的遗传-模拟退火的有序任务调度算法[J]. 微电子学与计算机，2006，23(10)：62-64

[5] 温钰洪，王鼎兴，郑纬民. 异构集群系统中的最优处理机分配算法[J]. 计算机学报，1996，19(3)：161-167

[6] 彭蔓蔓，徐立超，王颖. 异构多核处理器的任务分配及能耗的研究[J]. 计算机应用研究，2010，27(5)：1729-1731

[7] 宋曼，李春林. 一种低能耗低反应时间的并行任务调度方法[J]. 计算机应用研究，2009，26(6)：2251-2253

[8] 曾国荪，陆鑫达. 异构计算中的负载共享[J]. 软件学报，2000，11(4)：551-556

[9] Park C I, Choe T Y. An optimal scheduling algorithm based on task duplication[J]. IEEE Transactions on Computers, 2002, 51(4): 444-448

[10] 王樱，李琳，王杰. 基于有向无环图的时间-费用优化调度算法[J]. 衡阳师范学院学报，2010，31(3)：84-87

[11] Tiwari V, Malik S, Wolfe S. Power analysis of embedded software: a first step towards software power minimization[J]. IEEE Transactions on Very Large Scale Integration Systems, 1994, 2(4): 437-445