软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(2):240−252 [doi: 10.3724/SP.J.1001.2012.04141]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

众核处理器系统核资源动态分组的自适应调度算法*

曹仰杰[1+], 钱德沛[1,2], 伍卫国[1], 董小社[1]

[1](西安交通大学 电子与信息工程学院,陕西 西安 710049)

[2](北京航空航天大学 计算机学院,北京 100191)

Adaptive Scheduling Algorithm Based on Dynamic Core-Resource Partitions for Many-Core Processor Systems

CAO Yang-Jie[1+], QIAN De-Pei[1,2], WU Wei-Guo[1], DONG Xiao-She[1]

[1](School of Electronics and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China)

[2](School of Computer Science and Engineering, BeiHang University, Beijing 100191, China)

+ Corresponding author: E-mail: caoyj@stu.xjtu.edu.cn

Cao YJ, Qian DP, Wu WG, Dong XS. Adaptive scheduling algorithm based on dynamic core-resource partitions for many-core processor systems. *Journal of Software*, 2012,23(2):240−252. http://www.jos.org.cn/1000-9825/4141.htm

Abstract: With the aim to address the increasing difficulty of efficiently using large number of cores in many-core processors, a core-partitioned adaptive scheduling algorithm, named CASM (core-partitioned adaptive scheduling for many-core systems), is proposed. CASM dynamically aggregates cores into different partitions by splitting or merging task-clusters, which ensures the efficiency of isolated accessing in these core partitions. To improve the scheduling efficiency of CASM, equi-partitioning scheduling algorithm is adopted to reallocate the cores among task-clusters, and the feedback-driven adaptive scheduling algorithm is implemented within the task-clusters. Online competitive analysis shows that CASM achieves 2-competitiveness ratio with respect to the execution time of parallel jobs, which indicates that CASM has better performance and scalability. The experimental results demonstrate that compared with WS (work-stealing), AGDEQ (adaptive greedy dynamic equi-partitioning) and EQUI◦EQUI, CASM reduces the execution time of the same workload by nearly 46%, 32% and 15% respectively. Under the same power consumption, CASM greatly enhances the system throughput.

Key words: many-core processor; cluster-based scheduling; adaptive scheduling; competitive analysis; power-efficient computing

摘 要: 针对众核处理器系统的核资源优化使用问题,提出了一种支持核资源动态分组的自适应调度算法 CASM (core-partitioned adaptive scheduling for many-core systems).该算法通过对任务簇的拆分与合并,动态构建可弹性分区的核逻辑组,实现核资源的隔离优化访问.为了平衡核资源利用率及任务调度效率,CASM 算法针对任务簇间和簇内的不同特点,分别采用公平性较好的均衡调度算法和资源利用率较高的自适应调度算法.在线竞争理论分析表明,CASM 算法的任务执行时间在线竞争比为常数 2,其性能可扩展性较好.实验结果表明,与 WS(work-stealing),

* 基金项目: 国家自然科学基金(61073011, 61133004, 61173039); 国家高技术研究发展计划(863)(2008AA01A202, 2009AA01A131); 中意国际合作项目(2009DFA12110)

收稿时间: 2011-07-12; 修改时间: 2011-09-06; 定稿时间: 2011-11-14

AGDEQ(adaptive greedy dynamic equi-partitioning)和EQUI◦EQUI算法相比,CASM算法使任务集运行时间分别减少了近46%,32%和15%.在相同能耗情况下,CASM算法大幅度地提升了系统吞吐量.

关键词: 众核处理器;分组调度;自适应调度;竞争分析;高效能计算

中图法分类号: TP316 **文献标识码**: A

近年来,随着单核处理器芯片集成度和主频的提高,处理器技术遇到了制造成本、功耗、散热等问题,促使多核、多线程技术成为处理器系统发展的新方向[1].目前集成数十个核的处理器产品已经面世,随着技术发展和需求推动,未来处理器将集成几百甚至上千个核,这类处理器系统一般称为众核处理器系统[2].众核处理器核数的增加保证了计算和数据处理能力持续提高,然而如何使这种硬件能力转变成应用性能的提升,是众核时代面临的严峻挑战之一.

为了充分发挥众核处理器系统的并行处理能力,学术界和工业界都进行了较多的研究,提出了一系列技术和方法,如基于编译的自动并行化技术[3]、改善传统锁机制可编程性的事务存储技术[4]、基于分割全局地址空间模型的编程语言 X10,Fortress 和 Chapel 等[5]以及扩展现有编程语言的共享存储编程模型 OpenMP,Cilk 和 TBB(Intel's Threading Building Blocks)等[6].以上技术和方法从不同层面挖掘和释放应用软件的并行执行能力,以达到提升众核处理器系统利用效率的目的.在众核处理器体系结构下,为了实现资源优化调度和负载均衡,以上技术和方法通常都需要运行时系统的支撑.

运行时系统是上层应用程序到底层硬件资源映射执行的中间桥梁.其主要功能是调度和管理应用程序内部并行任务,并将其正确、高效地映射到底层硬件资源上执行.众核运行时系统通常都包含针对应用程序内部并行任务的负载均衡策略,如 OpenMP 分别支持静态、动态和有指导的调度策略[7],Cilk,TBB 支持深度优先任务窃取调度策略[8,9],X10支持广度优先任务窃取和混合调度策略等[5].然而,研究表明,随着众核处理器核数的增加,当前众核运行时系统的核资源利用效率较低,导致系统的可扩展性较差,应用程序的性能不能与核数成正比增长[6].此外,目前众核运行时系统种类繁多且相互独立,彼此之间不能交换信息,当多种应用程序同时运行时,易造成对核资源的恶性竞争,导致系统吞吐率降低.

针对上述问题,资源分区隔离是提升众核处理器系统使用效率的有效方法.通过为应用划分不同的资源集合,不仅可以降低应用间的资源竞争,而且能够提升资源的调度和使用效率.文献[10,11]中分别对如何管理众核处理器的内存资源和片上网络资源进行了研究,通过控制每个应用访问资源的次数比例,降低应用程序对资源的恶性竞争,提升了系统资源的利用率.基于均衡分配算法 EQUI[12],文献[13]提出了一种支持处理器分组和任务分簇的调度算法 EQUI◦EQUI.该算法将处理器资源的管理分为两层,并在每一层分别采用 EQUI 算法实现资源的动态分组和任务调度,在一定程度上优化了处理器资源的使用效率.此外,虚拟机技术也是目前提升众核处理器系统使用效率的一种有效方法[14,15].虚拟机技术通过将处理器核划分成为不同资源集合,并使应用程序分别运行在不同的虚拟机中,达到对处理器核资源分而治之、提升处理器整体利用率的目的.虚拟机技术通常需要非常复杂的设计才能完成对硬件资源的划分和管理,本身运行时开销较大,并且虚拟机间的核资源不能根据应用负载进行动态调整,对应用程序的实际性能影响较大.因而,虚拟机技术也无法从根本上解决众核处理器的性能优化问题.

基于资源分区自治的思想,本文提出一种支持核资源动态分组的自适应调度算法 CASM(core-partitioned adaptive scheduling for many-core systems).该算法通过任务簇的拆分与合并,动态构建可弹性伸缩的核逻辑分组,实现核资源的隔离优化访问.与以往的研究相比,CASM 的优势在于:(1) 在运行时实现对处理器核资源的动态划分,有利于系统负载的动态均衡;(2) CASM 针对任务簇间和簇内的不同特点,分别采用公平性较好的均衡调度算法和利用率较高的基于反馈机制的自适应调度算法,增强了核资源分配的公平性和资源利用效率;(3) 通过在线竞争理论分析表明,CASM任务执行时间的在线竞争比为常数2,与处理器核数无关,其性能可扩展性较好.实验结果表明,与 WS(work-stealing),AGDEQ(adaptive greedy dynamic equi-partitioning)和 EQUI◦EQUI 算法相比,CASM 算法的任务运行时间分别减少了近46%,32%和15%.

1 支持核资源动态分组的自适应调度算法 CASM

1.1 问题形式化描述及定义

采用三参数法$\alpha|\beta|\gamma$表示调度问题,其中,α,β和γ分别表示处理器核数、任务特征和优化目标函数.因此,面向众核处理器系统的在线调度问题表示为$P|$online,pmtn,$r_j|\Sigma F_\Gamma$,即系统有P个处理核;任务r_j时刻到达,执行过程可中断(pmtn),核资源以在线(online)方式分配;算法优化目标是最小化任务完成时间ΣF_Γ.

任务表示为一个三元组的有向无环图(DAG),即$J=(V,E,W)$,其中,$V=\{v_i|v_i$为任务J的有序子任务,$1\leqslant i\leqslant k,k$是子任务数目$\}$,$E=\{e_{i,j}|e_{i,j}$表示$v_i$到$v_j$存在依赖关系,$v_i$是$v_j$的先驱,$1\leqslant i,j\leqslant k,i\neq j\}$,$W=\{w_i|w_i$为$v_i$的计算工作量,$1\leqslant i\leqslant k\}$.任务$J$关键路径的运行时间为任务最小完成时间,用$L(J)$来表示.与传统基于静态 DAG 的任务调度不同,本文研究在线调度算法,任务在运行时系统中表示为动态展开的 DAG[8].为了便于形式化描述所研究问题,引入以下符号和定义:

定义 1. $\Gamma=\{J_1,J_2,\ldots,J_n\}$为任务集合,$G=\{G_1,G_2,\ldots,G_m\}$称为任务集$\Gamma$的一个任务划分,当且仅当$G_1\cup G_2\cup\ldots\cup G_m=\Gamma$,且$G_i\cap G_j=\varnothing(1\leqslant i,j\leqslant m,i\neq j)$,$G_j$称为任务簇.其中,$\cup$和$\cap$分别表示集合的并和交运算.$|G_j|$表示任务簇$G_j$内的任务个数,$m$为任务簇数量.

定义 2. 如果任务J_i在时刻t称为存活的,当且仅当时刻t任务J_i已被调度且未执行完成,即$r_i<t<c_i$,其中,r_i,c_i分别表示任务J_i的到达和完成时间.如果任务簇G_j中至少存在 1 个存活的任务,则称该任务簇是存活的.t时刻系统中存活的任务簇数量用M_t表示.

定义 3. 任务簇G_j的最小完成时间(makespan)$\pi(G_j)=\max_{J_i\in G_j}\{c_i\}$,系统的任务完成时间$F(\Gamma)$为全部任务簇最小完成时间之和,即$F(\Gamma)=\sum_{i=1}^{m}\pi(G_i)=\int_0^{\infty}M_t\mathrm{d}t$.本文以任务完成时间$F(\Gamma)$为代价函数来分析算法性能.特殊情况下,当系统内只有 1 个任务簇时,系统任务完成时间退化为任务集Γ的最小完成时间;当系统内每个任务簇中只包含 1 个任务时,任务完成时间则为任务集Γ的总响应时间.

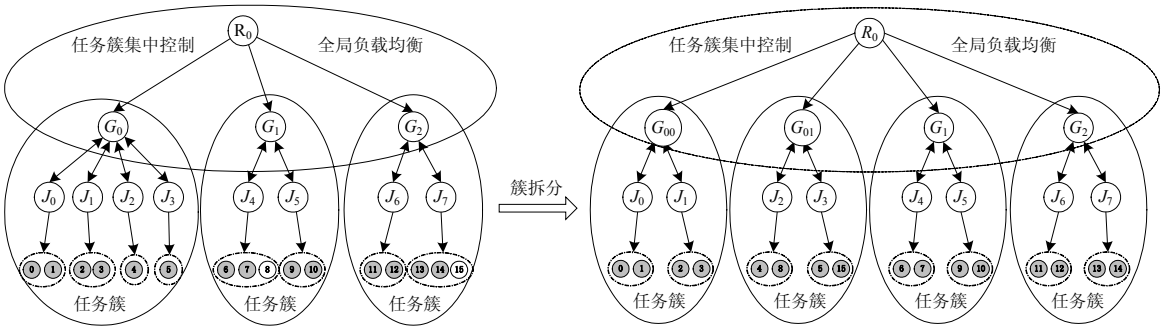
定义 4(工作量函数). $W_+[J_i(t)]$表示存活任务J_i在t时刻完成的工作量,$W_-[J_i(t)]$表示任务J_i在t时刻仍剩余的工作量.任务簇G_j在t时刻完成工作量及剩余工作量分别为$W_+[G_j(t)]=\sum_{J_i\in G_j}W_+[J_i(t)]$,$W_-[G_j(t)]=\sum_{J_i\in G_j}W_-[J_i(t)]$.

定义 5. 对于在线算法 ALG 及任意一个输入实例I,如果有$\max_I\frac{C_{ALG}(I)}{C_{OPT}(I)}\leqslant c$成立,则称$c$为在线算法 ALG 的竞争比,其中$C_{ALG}(I)$,$C_{OPT}(I)$分别表示算法 ALG 和相应最优离线算法 OPT 相对于实例 I 的代价函数.竞争比分析方法常称为对手法(adversary method),是一种用于分析算法在最坏情况下性能的方法.

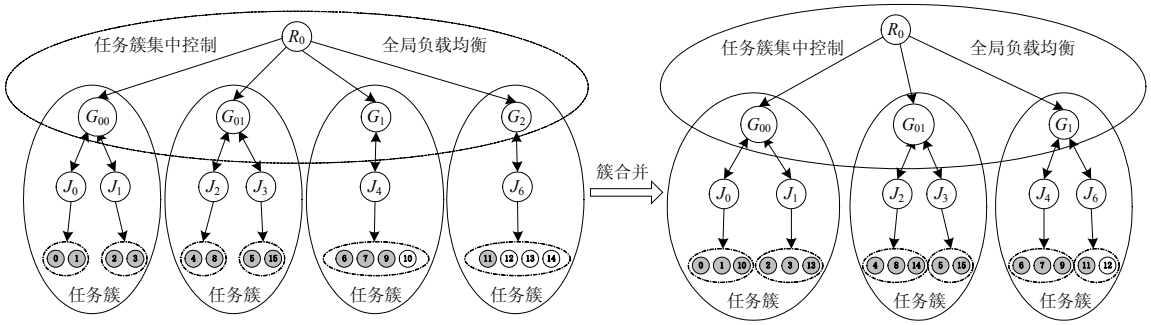
1.2 CASM算法描述

CASM 算法主要包括 3 个部分:核资源的动态逻辑分组、任务簇间的负载均衡以及任务簇内的自适应调度.CASM 通过任务簇的拆分与合并,实现对核资源逻辑分组的动态调整及负载均衡,采用均衡调度算法 EQUI (equi-partition)[12]实现任务簇间核资源的高效、公平分配,任务簇内采用基于反馈机制的自适应调度算法.在具体实现上,CASM 采用多层调度策略,逐级实现对核资源的分配及任务调度,其逻辑结构如图 1 所示(图中R节点代表核逻辑分组及任务簇的控制器;G节点代表每个任务簇相应的控制器;J节点代表每个应用程序的线程控制器;带标号的小圆圈代表处理器核,其中灰色填充表示其为忙状态,白色为空闲状态;箭头表示信息流向,双向箭头表示信息反馈).图 1 中每个结节代表 CASM 不同层次中负责执行资源分配和任务调度的逻辑控制器.最顶层为 CASM 的中央控制器(R节点),负责任务簇的集中控制,并采用 EQUI 算法实现核资源在任务簇间的均衡分配;第 2 层为任务簇层,由逻辑上相互独立的任务簇组成,每个任务簇控制器(G节点)负责其簇内任务运行时反馈信息的收集与统计以及簇内资源分配与调度;第 3 层为任务层,由不同类型的应用程序组成,其线程控制器(J节点)

采用广度优先的贪婪调度算法,将应用分解后的并行任务映射到核资源上执行.



(a) 拆分负载较重的任务簇 G_0 并重新调整核逻辑分组布局



(b) 合并负载较轻的任务簇 G_1 和 G_2 并重新调整核逻辑分组布局

Fig.1 Diagram of how CASM achieves load balancing by splitting or merging task-clusters

图 1 CASM 通过动态拆分或合并任务簇实现负载均衡的示意图

1. 核资源逻辑分组及任务簇间负载均衡

为了较好地实现任务簇间负载均衡,当新任务到达时,CASM 算法采用随机方式对任务进行分簇.其主要原因如下:

(1) 随机分簇策略实现简单,在线决策速度快,引入额外开销小;

(2) 任务并行度的变化规律,即任务对处理器核资源的需求特征,具有较强的随机性,采用随机分配方式有利于系统负载均衡,能够有效地避免某些处理器核由于分配不合理而出现饥饿现象.

以 Cilk 应用负载为例,本文实验对比了不同应用负载的并行度变化特征,实验平台及测试负载详情见第 3.2 节.图 2 显示了不同应用负载在某一时间段内的并行度变化情况.从图 2 可以看出,不同任务的并行度变化特征差异较大,并且与任务自身的计算模型相关,如 FIB 的并行度较高并且变化较为平稳,而 CK 的并行度变化起伏较大.实验结果表明,任务内部并行度具有较强的不确定性和随机性.因此,CASM 算法采用随机方式对任务进行分簇是较为合理的,并有利于整个系统的负载均衡.

CASM 实现核资源逻辑分组及任务簇间负载均衡的具体流程如下:

(1) 初始化.CASM 依据均衡分配算法 EQUI,将系统核资源划分为 $\theta(2\leqslant\theta\leqslant P)$个逻辑组,其中,$\theta$是可调节参数,$P$ 为处理器核的总数量.如图 1(a)所示,系统包含 16 个处理器核,且被划分成 G_0,G_1 和 G_2 这 3 个逻辑组,每组分别占有 6,5,5 个核.

(2) 任务簇聚合.任务到达时,CASM 随机选择某一个核逻辑组,并将该任务与该组已有任务聚合成任务簇,依据簇内调度算法共享该组中的核资源.如图 1(a)所示,系统中共有 8 个任务,聚合成 G_0,G_1 和 G_2 这 3 个任务簇(核逻辑组与任务簇采用相同编号),每个簇内分别包含 4,2,2 个任务及 6,5,5 个核资源.

(3) 任务簇拆分.新任务不断到达及任务并行度的不规则性将引发系统负载不均衡.CASM 基于以下规则进行调整:当某任务簇的核平均利用率大于阈值δ_{max}(通常δ_{max}>85%)时,如图 1(a)所示的 G_0,该任务簇被视为过载;如果任务簇总数 m 满足 $m<P$,则 CASM 将该任务簇(G_0)依据任务数量进行平均拆分,形成两个新的任务簇(G_{00} 和 G_{01}),新任务簇保持每个任务的原有核资源,分簇过程中任务的执行并未中断;依据 EQUI 分配策略,CASM 重新调整核资源在各分组的分布,即将其他分组多余的核划分到新分组中.如图 1(a)所示,G_0 拆分后将编号为 8 和 15 的核划分到 G_{01} 中,即可实现负载均衡.任务簇拆分仅改变核分组的逻辑结构,任务并不需要进行迁移,系统引入的额外开销小.任务簇拆分的优点在于:① 任务簇拆分使核分组逻辑布局进行动态调整,有利于实现负载均衡;② 核逻辑分组数量的增加使每个任务簇管理的核资源和任务减少,有利于提高核资源利用率和任务调度效率.

(4) 任务簇合并.任务运行完成以及任务并行度的动态不规则性也将引发系统负载不均衡.CASM 基于以下规则进行调整:当某任务簇的核平均利用率低于阈值δ_{min}(通常δ_{min}<20%)时,该任务簇被视为轻载;如果任务簇总数 m 满足 $m>\theta$,则 CASM 将轻载任务簇与系统中次轻载的任务簇进行合并,并依据 EQUI 策略重新调整核逻辑分组布局.如图 1(b)所示,当 G_2 出现轻载时,CASM 将其与 G_1 进行合并,并将空闲核均分到其他逻辑分组中,即将编号 10 和 13 核划分到 G_{00},编号 14 核划分到 G_{01},实现负载均衡.

(5) 簇内任务调度.CASM 在任务簇内采用基于反馈机制的自适应调度算法.相对于整个系统,每个任务簇内的处理器核与任务相对较少,自适应调度策略更有利于提升核资源利用率和任务执行效率.如图 1 所示,CASM 为每个任务簇中设置独立的调度控制器,如 G_0,G_1 和 G_2 等,负责簇内核资源与任务的自适应调度,图 1 中双向箭头代表任务运行时信息的反馈.为了适用于众核运行时系统,CASM 对传统自适应调度算法 AGDEQ[16] 进行改进,使其支持基于有向无环图(DAG)依赖关系的任务调度.

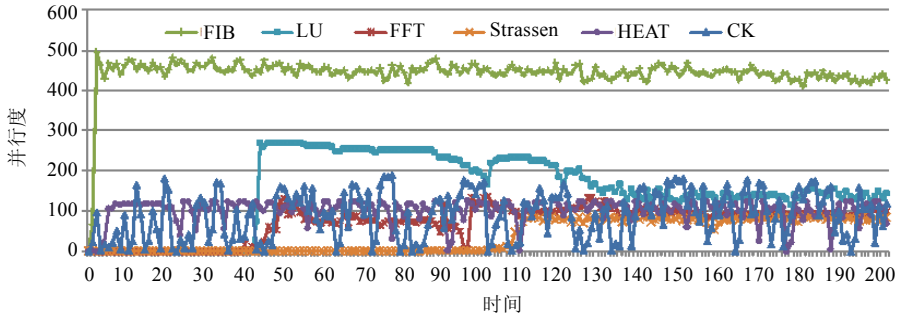


Fig.2 Parallelism variation of different Cilk applications

图 2 不同 Cilk 应用负载的并行度变化特征

2. CASM 任务簇内的自适应调度算法

CASM 簇内调度算法主要包括两部分:一是任务运行时信息统计及其并行度预测;二是核资源的动态分配与任务调度执行.

基于对核资源利用率的周期性统计,CASM 采用启发式推测方法,对任务并行度进行预测.任务并行度的预测值即为任务下一周期的核资源期望值.预测方法见表 1,其中 q 和 q+1 分别代表当前和下一个调度周期,$d(J_i,q)$,$w(J_i,q)$和 $a(J_i,q)$分别代表第 q 调度周期任务 J_i 的核资源期望、完成工作量及其处理器核的实际分配量,Q 为调度周期长度,s 为处理器的运行速度,ρ和δ分别表示资源调节响应系数(通常为 2)和资源利用率阈值(通常为 85%),任务 J_i 的核资源期望初始值设置为 1,即 $d(J_i,0)$=1.任务 J_i 下一周期核资源期望值的计算过程如下:假如第 q 调度周期任务 J_i 满足 $a(J_i,q)\geq d(J_i,q)$且 $w(J_i,q)\geq a(J_i,q)sQ\delta$,则表明 q 调度周期 J_i 的核资源期望得到满足并且核资源被充分利用,因此,J_i 此时的并行度较高,设定下一周期的核资源期望值 $d(J_i,q+1)=d(J_i,q)\cdot\rho$;反之,如果 $w(J_i,q)<a(J_i,q)sQ\delta$,则表明 q 调度周期中 J_i 核资源利用率低,任务 J_i 此时并行度较低,因此设定 $d(J_i,q+1)= d(J_i,q)/\rho$;当 $a(J_i,q)<d(J_i,q)$且 $w(J_i,q)\geq a(J_i,q)sQ\delta$时,任务 J_i 的并行度变化情况不确定,核资源期望值保持不变,见表 1.

Table 1 Parallelism desire of task J_i in the next quantum

表 1 任务 J_i 下一调度周期的核资源期望值

	期望满足,$a(J_i,q)\geqslant d(J_i,q)$	期望未满足,$a(J_i,q)<d(J_i,q)$
有效利用,$w(J_i,q)\geqslant a(J_i,$ $sQ\delta$	$d(J_i,q+1)=d($ $_i,q)\cdot\rho$	$d(J_i,q+1)=d(J_i,q)$
未有效利用,$w(J_i,q)<a(J_i,q)sQ\delta$	$d(J_i,q+1)=d(J_i,q)/\rho$	

由于基于任务 DAG 的子任务在执行过程中存在偏序依赖关系,CASM 对簇内任务调度采用与 AGDEQ 不同的调度策略,即广度优先的贪婪调度策略.如图 3 所示(其中,任务 J_i 的处理器核当前分配量为 3,黑色节点代表上一周期已执行完成的子任务;灰色代表当前周期内就绪子任务;白色代表下一周期将要执行的子任务;虚线之间对应任务 DAG 不同层次的子任务),CASM 在保持任务 DAG 偏序依赖关系的前提下,优先调度任务 DAG 中处于同一层的子任务,调度顺序如图 3 中节点标号所示,最大限度地减少了各子任务间的等待时间.在实际执行过程中,由于各子任务的任务量并不一致,子任务执行顺序会有所改变,但不同层次子任务间仍保持偏序关系.

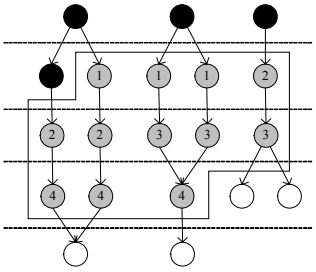


Fig.3 An example of how CASM schedules the tasks J_i at certain quantum

图 3 某一周期 CASM 对任务 J_i 的调度示例

依据任务的反馈信息,CASM 簇内核资源分配采用支持反馈机制的动态均衡分配算法 DEQ(dynamic equi-partition)[16],DEQ 实现如图 4 所示(其中,|·|表示集合大小).

输入:任务簇 G_j,G_j 的核资源分配量 P_j 以及任务下一周期核资源期望值 $d(J_i,q+1)$, $J_i\in G_j$.
输出:第 $q+1$ 周期任务簇 G_j 中任务的核资源分配量 $a(J_i,q+1)$,$J_i\in G_j$.

```
1: if G_j=∅ then
2:     return
3: H={J_i|J_i∈G_j & d(J_i,q+1)≤P_j/|G_j|}
4: if H=∅ then
5:     for each J_i∈G_j do
6:         a(J_i,q+1)=P_j/|G_j|
7: else
8:     for each J_i∈H do
9:         a(J_i,q+1)=d(J_i,q+1)
10:    DEQ(G_j−H,P_j−Σ_{J_i∈H} a(J_i,q+1))
```

Fig.4 Algorithm DEQ of task cluster G_j

图 4 任务簇 G_j 的 DEQ 算法描述

与传统资源分组算法,如 EQUI◦EQUI 相比,CASM 在处理器核分配过程中充分利用了并行任务自身的资源需求特征.通过引入自适应反馈机制,CASM 动态调节核资源在任务间的分配,使任务的资源需求量与核资源实际分配量相匹配,有利于核资源的高效使用,能够提升整个系统的吞吐率.此外,CASM 基于众核运行时系统进行构建,不需要对操作系统做任何修改,因此算法具有较好的通用性和可移植性.

2 CASM 算法的在线竞争分析

从调度理论角度来讲,对在线调度算法性能进行定量分析,可以直观对比不同算法的优劣.本文采用扩展的在线竞争分析方法对 CASM 性能进行分析.为了得出更一般的结论,在算法分析过程中,处理器核在执行过程中可中断挂起且中断开销忽略不计.

引理 1(分摊局部竞争分析(amortized local competitiveness)). 对于并行任务集$\Gamma=\{J_1,J_2,\ldots,J_n\}$,设在线算法 ALG 是$\Gamma$的一个合法调度.假定算法 ALG 调度完成任务集$\Gamma$的时间代价函数为 $F(t)$,相应离线最优算法 OPT 的时间代价函数为 $O(t)$,如果存在一个势能函数$\Phi(t)$并且满足以下 4 个条件:

(1) 有界条件:对于势能函数$\Phi(t)$,除了其初始时刻的值为 0 以外,其他时刻为非负数值,即$\Phi(0)=0$,且存在时刻 t',当 $t\geqslant t'$,则有$\Phi(t)\geqslant 0$.

(2) 到达条件:新的任务时刻 t 到达时,势能函数$\Phi(t)$的值不增加.

(3) 完成条件:任务在时刻 t 完成时,势能函数$\Phi(t)$的值不增加.

(4) 运行条件:对于任意时刻 t,如果系统内没有新任务到达或任务已完成,则

$$\frac{\mathrm{d}F(t)}{\mathrm{d}t}-\gamma\frac{\mathrm{d}O(t)}{\mathrm{d}t}+\frac{\mathrm{d}\Phi(t)}{\mathrm{d}t}\leqslant 0 \tag{1}$$

因此,在线算法 ALG 竞争比为γ[17].

定理 1. 设任意有限任务集$\Gamma=\{J_1,J_2,\ldots,J_n\}$,$G=\{G_1,G_2,\ldots,G_m\}$是任务集$\Gamma$的一个划分,对于 $P|\text{online,pmtn},r_j|\Sigma F_\Gamma$调度问题,如果 CASM 使用运行速度为 $s=\varepsilon+2(1+\rho)/\delta$的处理器核,$\varepsilon$为任意小正常数,$\rho$和$\delta$分别为 CASM 的资源调节响应系数和资源利用率阈值,则在任意时刻 t,相对于定义 3 代价函数 $F(\Gamma)$存在势能函数$\Phi(t)$满足引理 1,且$\Phi(t)$满足以下公式(1)运行条件:

$$M_t+\frac{\mathrm{d}\Phi(t)}{\mathrm{d}t}\leqslant\frac{2s}{\omega}(M_t^o+M_t^B) \tag{2}$$

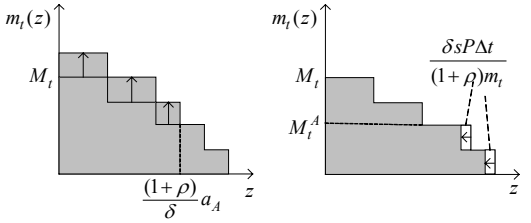
其中,如定义 2 所述,M_t表示在 t 时刻系统内存活的任务簇数量,M_t^B表示 t 时刻系统内满足属性 B 的任务簇数量,M_t^o表示最优离线算法 OPT 在 t 时刻系统内任务簇数量.

证明:依据表 1 将任务簇分成两种不同属性的集合 G_A 和 G_B.G_A 称为具有 A 属性,代表任务的核资源期望被满足,即 G_A 中任务的核资源期望值小于等于其处理器核的实际分配量,而其余任务簇则属于 G_B,称为 B 属性.令 $a(G_i,t)$为 t 时刻 G_i 已分配处理器核的数量,则 G_A 获得处理器核的总数量表示为

$$a_A=\frac{\mathrm{d}\Phi(t)}{\mathrm{d}t}\leqslant\frac{2s}{\varepsilon}(M_t^o+M_t^B) \tag{3}$$

其中,如果 G_i 属于 G_A,则$\{G_i\in G_A\}$为 1,否则为 0.由 CASM 的实现方式及定义 1 可知,$G=G_A\cup G_B$ 是所有任务集合的一个划分.

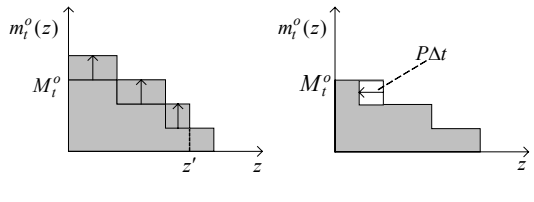
构造势能函数:由定义 4,在任意时刻 t,令 $m_t(z)=\sum_{i=1}^{M_t}\left\{W\_[G_A]\geqslant\frac{1+\rho}{\delta}\cdot z\right\}$, 即 $m_t(z)$表示相对于 CASM,时刻 t 系统中具有 A 属性并且其所有任务剩余工作量之和大于或等于$(\rho+1)\ z/\delta$的任务簇数量,其中 z 是正实数变量.相应地, $m_t^o(z)=\sum_{i=1}^{M_t^o}\{W\_[G]\geqslant z\}$ 表示 t 时刻相对于最优离线算法 OPT,系统中所有任务簇剩余工作量大于或等于 z 的任务簇数量.从图 5 和图 6 易知,$m_t(z)$和 $m_t^o(z)$是单调递减梯形函数.由于 OPT 算法优先完成剩余工作量最小的任务[17],因此在较小Δt 时间内只有最短任务的工作量发生变化(如图 6(b)所示),且减少量为 $P\Delta t$.而由于 CASM 对任务簇使用均衡分配核资源的 EQUI 算法,在任务簇内使用 DEQ 算法,因此,在Δt 时间内,CASM 必将执行具有 A 属性任务簇中的存活任务,其工作量减少量如图 5(b)所示.



(a) 新任务到达时刻 (b) 正常运行Δt时间

Fig.5 Changes of potential function by CASM

图 5 CASM 引发势能函数变化



(a) 新任务到达时刻 (b) 正常运行Δt时间

Fig.6 Changes of potential function by OPT

图 6 OPT 引发势能函数变化

由以上分析可知,$m_t(z)$和 $m_t^o(z)$分别代表了 CASM 和最优离线算法 OPT 使任务计算工作量减少的特征函数,由此定义势能函数如公式(4):

$$\Phi(t)=\eta\int_0^{\infty}\left[\left(\sum_{i=1}^{m_t(z)}i\right)-m_t(z)m_t^o(z)\right]\mathrm{d}z\text{,其中}\eta\text{为常数} \tag{4}$$

由引理 1 可知,势能函数在算法竞争比分析中起着关键作用.公式(4)本质上刻画了在线调度算法 CASM 与最优离线算法 OPT 的代价函数随时间变化的动态关系.

(1) 有界性分析:由于 0 时刻系统不存在存活的任务,即也不存在任何存活的任务簇,因此$\Phi(0)=0$.由$\Phi(t)$的定义可知,对于任意时刻 t,$\Phi(t)$是非负的,即$\Phi(t)\geqslant 0$.有界性成立.

(2) 到达条件:假定 t 时刻一个计算工作量为 z'的新任务到达,由势能函数$\Phi(t)$的定义可知,当 $z\geqslant z'$时,$m_t(z)$和 $m_t^o(z)$都不会发生变化,因此$\Phi(t)$也不会发生变化.当 $z<z'$时,令 $\phi_t(z)=\left(\sum_{i=1}^{m_t(z)}i\right)-m_t(z)m_t^o(z)$,t_-和 t_+分别为任务到达前和任务到达后时刻.分两种情况证明:

① 当 $z\leqslant(\rho+1)a_A/\delta$,$a_A$ 为满足属性 A 任务簇的处理器核分配量,则有:

$$\phi_{t_+}(z)-\phi_{t_-}(z)=\left(\sum_{i=1}^{m_{t_-}(z)+1}i\right)-\left(m_{t_-}(z)+1\right)\left(m_{t_-}^o(z)+1\right)-\left[\left(\sum_{i=1}^{m_{t_-}(z)}i\right)-m_{t-}(z)m_{t_-}^o(z)\right]=-m_{t_-}^o(z)\leqslant 0;$$

② 当$(\rho+1)a_A/\delta\leqslant z\leqslant z'$时,

$$\phi_{t_+}(z)-\phi_{t_-}(z)=\left(\sum_{i=1}^{m_{t_-}(z)}i\right)-m_{t_-}(z)(m_{t_-}^o(z)+1)-\left[\left(\sum_{i=1}^{m_{t_-}(z)}i\right)-m_{t_-}(z)m_{t_-}^o(z)\right]=-m_{t_-}(z)\leqslant 0.$$

因此,$\Phi(t_+)=\eta\int_0^{\infty}\phi_{t_+}(z)\mathrm{d}z\leqslant\eta\int_0^{\infty}\phi_{t_-}(z)\mathrm{d}z=\Phi(t_-)$.综上,新任务到达时势能函数$\Phi(t)$不会增加.

(3) 完成条件:由图 5 和图 6 可知,图中阴影块向左方向的减少代表任务的完成情况,当有任务完成时,图中最上方阴影块将消失,此时 $z=0$.即 $m_t(z)$和 $m_t^o(z)$减 1 时刻 $z=0$,因此,势能函数在此时刻不变.

(4) 运行条件:假设在很短的Δt 时间内,系统没有新任务到达或任务已完成,则

$$\left.\begin{aligned}\frac{\Phi(t)}{\mathrm{d}t}&=\frac{\eta}{\Delta t}\int_0^{\infty}\left[\left(\sum_{i=1}^{m_{t+\Delta t}(z)}i\right)-m_{t+\Delta t}(z)m_{t+\Delta t}^o(z)-\left(\sum_{i=1}^{m_t(z)}i\right)+m_t(z)m_t^o(x)\right]\mathrm{d}z\\&\leqslant\frac{\eta}{\Delta t}\int_0^{\infty}\left[\left(\sum_{i=1}^{m_{t+\Delta t}(z)}i\right)-\left(\sum_{i=1}^{m_t(z)}i\right)\right]\mathrm{d}z+\frac{\eta}{\Delta t}\int_0^{\infty}\left[-m_{t+\Delta t}(z)m_{t+\Delta t}^o(z)+m_t(z)m_{t+\Delta t}^o(z)-m_t(z)n_{t+\Delta t}^o(z)+m_t(z)m_t^o(z)\right]\mathrm{d}z\\&\leqslant\frac{2(1+\rho)\eta}{\delta\varepsilon P\Delta t}\left(-\frac{M_t^A(M_t^A+1)\delta sP}{2(1+\rho)M_t}\Delta t+M_tP\Delta t+M_t^o\frac{\delta sPM_t^A}{(1+\rho)M_t}\Delta t\right)\\&\leqslant\frac{2(1+\rho)}{\delta\varepsilon}\left(1-\frac{\delta sx_t^2}{2(1+\rho)}\right)M_t+\frac{2sx_t}{\varepsilon}M_t^o\end{aligned}\right\} \tag{5}$$

其中,$x_t=M_t^A/M_t$ 并且 $0\leqslant x_t\leqslant 1$.由于任一任务集在某一时刻只能属于 $G_A(\Gamma)$或者属于 $G_B(\Gamma)$,因此 $M_t^B=(1-x_t)M_t$.将公式(5)代入公式(2)即可得出所证结论. □

定理 2. 设任意有限任务集$\Gamma=\{J_1,J_2,\ldots,J_n\}$,$G=\{G_1,G_2,\ldots,G_m\}$是任务集$\Gamma$的一个划分,对于 $P|\text{online,pmtn},r_j|\sum F_\Gamma$调度问题,如果 CASM 使用运行速度为 $s=\varepsilon+2(1+\rho)/\delta$的处理器核,$\varepsilon$为任意小正常数,$\rho$和$\delta$分别为 CASM 的资源调节响应系数和资源利用率阈值,则在线调度算法 CASM 的任务执行时间代价函数 $F(\Gamma)$满足公式(6):

$$F(\Gamma)\leqslant\left(2+\frac{4(1+\rho-\rho\delta)}{\delta(1-\delta)\varepsilon}\right)O(J)+\frac{2msQ}{\varepsilon}\left(\log_\rho P+2\right) \tag{6}$$

即,调度算法 CASM 相对最优离线算法的在线竞争比为 $O\left(2+\dfrac{4(1+\rho-\rho\delta)}{\delta(1-\delta)\varepsilon}\right)$.

证明:由定义 3 可知,CASM 的执行时间代价函数 $F(\Gamma)=\int_0^{\infty}M_t\mathrm{d}t$, 而最优离线算法 OPT 相应代价函数为 $O(\Gamma)=\int_0^{\infty}M_t^o\mathrm{d}t$.因此,由定理 1 公式(2)两边积分可得:

$$F(\Gamma)+\Phi(\infty)-\Phi(0)+\sum_{t\in T}(\Phi(t-)-\Phi(t+))\leqslant\frac{2s}{\varepsilon}O(\Gamma)+\frac{2}{\varepsilon}t_B(\Gamma),$$

其中,T 表示新任务到达时间集,$t_B(\Gamma)=\int s\cdot m_t^B\mathrm{d}t$ 表示 B 类任务簇的所有处理时间之和.由 B 类任务簇的特性,即任务对处理器核资源的需求未得到满足,易证得 $t_B(\Gamma)\leqslant 2L(\Gamma)/(1+\delta)+mQs(\log_\rho P+2)$,因此由定理 1 可知,

$$F(\Gamma)\leqslant\frac{2s}{\varepsilon}\cdot O(\Gamma)+\frac{4}{(1+\delta)\varepsilon}\cdot L(\Gamma)+\frac{2mQs}{\varepsilon}(\log_\rho P+2) \tag{7}$$

由于任务的运行时间大于等于任务的最小完成时间,即 $O(\Gamma)\geqslant L(\Gamma)$,代入公式(7)可得定理结论. □

定理 1 和定理 2 表明,以任务的执行时间为评价指标,CASM 与最优离线算法的在线竞争比为常数,当ε较大时在线竞争比为 2,说明其性能可扩展性较好,本文将进一步通过实验验证 CASM 的实际性能.

3 算法实现与性能分析

基于 Cilk[8]运行时系统,对比测试两类共计 4 种算法:WS(Work-Stealing)[8],AGDEQ[16],EQUI◦EQUI[13]和本文算法 CASM.其中,WS 和 AGDEQ 是不支持任务分簇的调度算法,EQUI◦EQUI 和 CASM 是支持任务分簇的调度算法.采用 Cilk 标准测试集[8],并以 Cilk 运行时系统默认支持的调度算法 WS 为基准,在 64 核服务器上对比测试了不同调度算法的性能.

3.1 Cilk运行时系统改进及算法实现

Cilk 是 MIT 提出的一种基于 C 语言的并行程编程语言,其简单易用且运行效率高.Cilk 运行时系统实现了基于 Work-Stealing 的负载均衡算法,但其并不支持对多个 Cilk 任务的协同调度.基于 Cilk 运行时系统,本文设计了一种自适应调度框架 A-Cilk(adaptive Cilk),其实现结构如图 7 所示.A-Cilk 主要从以下两个方面对 Cilk 进行了改进:

(1) 引入全局核资源分配控制器.核资源分配控制器采用后台守护进程方式构建在操作系统之上,并以共享存储、信号量等方式作为与运行时系统的通信机制.核资源分配控制器通过系统配置模块获取如调度周期、调度算法及低层处理器核配置等系统信息.其基本功能:去除 Cilk 运行时系统需要通过命令行参数 nproc 手动指定处理器核数的限制,由调度器依据系统负载状况自动进行分配;实现核逻辑分组的动态管理;任务的注册登记、任务簇的创建、拆分与合并等.

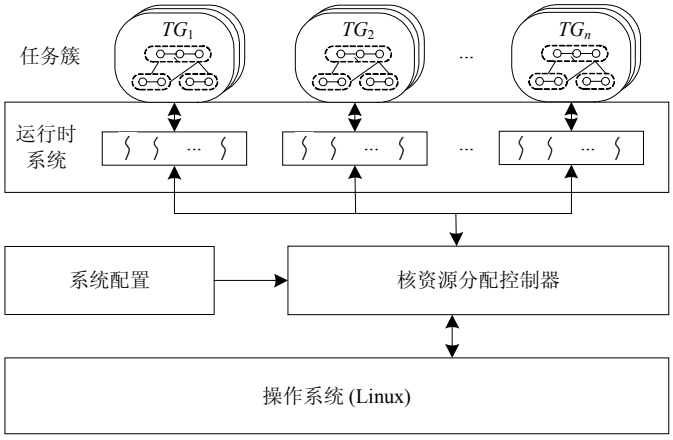


Fig.7 Framework of ACilk

图 7 ACilk 的系统结构图

(2) 核资源的自适应调节.基于 POSIX Threads 重新改写了 Cilk 运行时系统,通过动态控制每个工作线程(Worker)的工作状态实现处理器核的动态分配与调节.工作线程是 Cilk 运行时系统实际执行应用程序的系统线

程,通常与处理器核是一一对应关系.A-Cilk 为每个 Worker 引入了 3 种不同的工作状态:Working,Mugging 和 Sleeping.Working 表示工作线程处于工作状态,即本地工作队列非空,实际占用处理器核进行数据处理;Mugging 表示负载迁移状态,即当 Worker 空闲时,随机选择另一个工作队列非空且状态为 Sleeping 的 Worker,并迁移其所有工作负载.Mugging 不仅保证程序的正确运行而且通过负载整体迁移减少了系统开销;Sleeping 表示工作线程处于休眠状态.工作线程的状态转换如图 8 所示.A-Cilk 的具体控制过程:系统初始时为每个应用程序创建与实际处理器核数量一致的工作线程,并使其处于 Sleeping 状态;当应用程序实际分配的处理器核增加时,A-Cilk 则唤醒相应的工作线程;反之,当实际分配的处理器核减少时,A-Cilk 则休眠相应的工作线程,使其与实际分配保持一致.

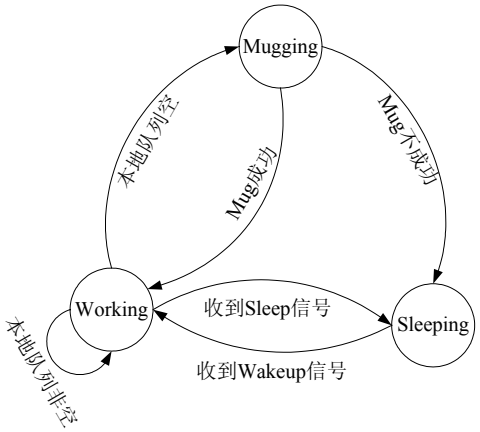


Fig.8 State transition diagram of a worker in running time

图 8 工作线程(Worker)运行时的状态转换图

3.2 测试环境配置

实验环境的基本配置为:8 个 Intel Xeon X7550 2.0GHz 18MB 缓存的处理器,系统共有 64(8×8)个核,内存为 256GB;Linux 内核版本为 2.6.18;采用 GCC 编译器,其版本为 4.1.2,程序编译时的优化级别选项为-O2.采用 Cilk 标准测试集[8],其详细说明见表 2.

Table 2 Description of benchmarks

表 2 测试任务集说明

任务名称	问题规模	程序说明
CK	−b 10 −w 13	西洋跳棋
FIB	50	Fibonacci 数列
FFT	$-n\ 2^{28}$	快速傅里叶变换
HEAT	−g 10 −nx 8192 −ny 8192 −nt 1000	基于有限差分法的热扩散问题
LU	−n 4096	矩阵分解
STRASSEN	−n 4096	矩阵相乘

3.3 实验及结果分析

CASM 核逻辑分组的初始值θ=4,簇外与簇内调度周期分别为 100ms 和 10ms.CASM 的资源利用率阈值δ,δ_{max} 和δ_{min} 和资源调节响应系数ρ分别设置为 85%,85%,20%和 2.AGDEQ 的资源利用率阈值δ和资源调节响应系数ρ分别设为 85%和 2,以上参数值取自于实验过程中的经验值.每组实验分别运行 10 次,实验结果取其均值.由于 Cilk 运行时系统支持的 WS 算法需要手动设定处理器核数,为了体现算法对比的公平性,将 64 个处理器核静态划分成 4 部分,每部分为 16 核,并通过 Cilk 运行时系统提供的命令行参数 nproc 进行指定.

(1) 算法可扩展性

利用表 2 中的应用负载生成 6×2 共 12 个测试任务实例,并采用批量方式将其提交到系统中.以任务运行时

间及处理器核的平均利用率为评价指标,验证不同算法随处理器核数增加时的性能可扩展性.实验结果如图 9 和图 10 所示.

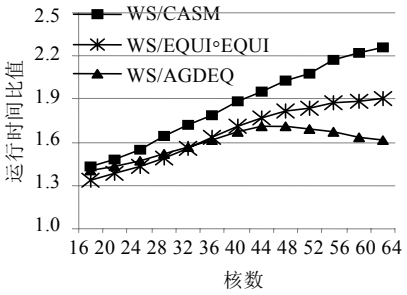


Fig.9 Speedup of different algorithms with increasing number of cores

图 9 不同算法随着核数增加的性能加速比

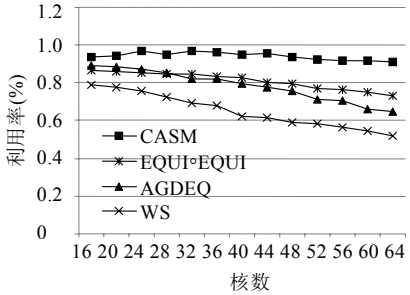


Fig.10 Utilization of different algorithms with increasing number of cores

图 10 不同算法随着核数增加的资源利用率

从图 9 可以看出,与 WS 算法相比,CASM,EQUI◦EQUI 和 AGDEQ 算法的性能都有不同程度的提升,表明自适应调度算法适用于众核处理器系统.随着处理器核数的增加,与 WS 算法相比,CASM 算法使任务的执行效率呈近线性增长,表明其性能可扩展性较好.主要原因是,CASM 对核资源的动态逻辑分组减少了任务对资源的恶性竞争,并增强了核资源分配的灵活性和调度效率.支持核分组的 EQUI◦EQUI 算法也呈现类似的变化规律,但由于其簇内采用基于 EQUI 的调度策略,其性能逊色于 CASM,并且核数较少时其优势并不明显,略低于 AGDEQ.以上实验结果表明,支持核资源动态逻辑分组的调度算法其扩展性较好,并且随着核的增加,其性能提升更加明显.

由于 WS 算法的窃取特性,处理器核总处于“忙”状态,即空闲时立即进行窃取.因此,实验中依据工作线程的工作状态进行利用率统计,可以较为准确地反映处理器核的实际利用情况,统计结果如图 10 所示.由实验结果可知,与 CASM,EQUI◦EQUI 和 AGDEQ 相比,WS 的核资源实际利用率偏低,资源浪费较大,当核数较多时情况较为严重时,其实际资源利用率不及 50%.如图 10 所示的实验结果表明,CASM 的核资源利用率较高,通常都保持在 90%以上.

(2) 算法性能比较

为了较好地比较算法的实际性能,实验采用随机生成的混合负载.任务请求,即任务的到达时间间隔服从指数分布.具体过程为,利用表 2 中 6 个应用负载生成 6×3 共 18 个测试任务实例,并设定任务到达的时间间隔是服从指数分布的泊松过程,其参数λ的取值范围为[1/16,1],代表系统负载由轻到重,处理器核数设定为 64,实验结果如图 11 所示.由实验数据可知,CASM 的性能优于其他算法,且随着系统负载的增加变化较为平稳.与 EQUI◦EQUI,AGDEQ 和 WS 调度算法相比,CASM 使任务平均运行时间减少近 15%,32%和 46%.EQUI◦EQUI 受系统负载的影响较大,随着负载的增加,其性能下降较为明显.相比于 CASM 和 EQUI◦EQUI,不支持核逻辑分组算法 AGDEQ 的性能相对较差.实验结果表明,支持核资源动态分组及任务分簇的调度算法其性能较好.其主要原因在于,核逻辑分组及任务簇的自主管理不仅优化了任务的调度效率,降低了任务间的资源竞争,而且核资源在不同逻辑组的动态调整有利于系统负载均衡.

(3) 运行时开销

资源调度过程中不可避免地引发系统额外开销,采用上述实验相同的配置环境,测试了不同任务负载的运行时开销,即由于调度算法的动态调节,导致工作线程在不同处理器核之间切换引发的时间开销.实验结果如图 12 所示,表明 WS 引发的额外开销较大,与其创建较多的工作线程有关;CASM 与 AGDEQ 也引发了一定量的额外开销且表现较为相似,主要是由于两者都采用了类似的自适应调度策略;与上述算法相比,EQUI◦EQUI 引发的额外开销相对较小.

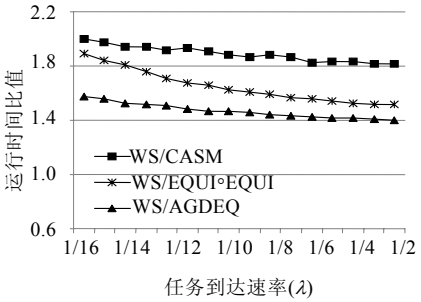


Fig.11 Performance comparison of different algorithms under different loads

图 11 不同负载情况下算法性能比较

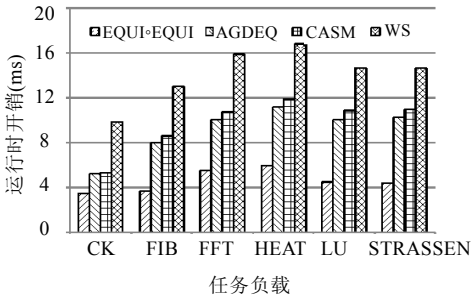


Fig.12 Runtime overhead comparison of different algorithms

图 12 不同算法运行时额外开销比较

综上所述,CASM 在任务执行时间和资源利用率方面不仅优于传统自适应调度算法 AGDEQ 及 Cilk 运行时系统支持的 Work-Stealing 算法,而且比同类分簇调度算法 EQUI◦EQUI 表现出更好的性能.由于处理器核资源的动态调节,CASM 引发了一定数量的额外开销,但并未对系统整体性能的提升产生较大影响.在相同负载情况下,与 EQUI◦EQUI,AGDEQ 及 Work-Stealing 算法相比,CASM 使任务平均运行时间分别减少了近 15%,32%和 46%.随着处理器片内核规模的不断增加,CASM 良好的可扩展性能够更好地发挥众核处理器的性能优势.

4 结束语

绿色计算是当前高性能计算领域最为关注的问题之一,如何充分发挥大规模并行处理部件效能,是实现未来高效能计算亟待解决的关键问题.传统基于单一策略的调度算法,由于缺乏灵活性,不适宜于核规模日趋庞大的众核体系结构.基于众核运行时系统,本文提出一种面向大规模并行处理系统的动态分组调度算法 CASM,增强了核资源分配的灵活性和协同性,提升了系统的整体效能和可扩展能力.理论分析及实验结果表明,CASM 在任务执行时间、资源利用率和可扩展性方面优于其他算法,适宜于众核处理器系统.进一步的研究内容包括:扩展 CASM 算法,使其较好地支持异构众核处理器系统;研究应用负载的资源需求特征,探寻更为高效的自适应调度机制.

致谢 国家超级计算天津中心为本文提供了高性能众核服务器实验环境,尤其是天津超算中心技术人员为实验环境的搭建及调试给予大力协助,在此表示感谢.

References:

[1] Geer D. Chip makers turn to multicore processors. Computer, 2005,38(5):11−13. [doi: 10.1109/MC.2005.160]

[2] Borkar S, Chien AA. The future of microprocessors. Communications of the ACM, 2011,54(5):67−77. [doi: 10.1145/1941487.1941507]

[3] Bridges MJ, Vachharajani N, Zhang Y, Jablin T, August DI. Revisiting the sequential programming model for the multicore Era. IEEE Micro, 2008,28(1):12−20. [doi: 10.1109/MM.2008.13]

[4] Abadi M, Birrell A, Harris T, Isard M. Semantics of transactional memory and automatic mutual exclusion. ACM Trans. on Programming Languages and Systems, 2011,33(1):1−50. [doi: 10.1145/1889997.1889999]

[5] Loh E. The ideal HPC programming language. Communications of the ACM, 2010,53(7):42−47. [doi: 10.1145/1785414.1785433]

[6] Bhattacharjee A, Contreras G, Martonosi M. Parallelization libraries: Characterizing and reducing overheads. ACM Trans. on Architecture and Code Optimization, 2011,8(1):5−29. [doi: 10.1145/1952998.1953003]

[7] Broquedis F, Diakhaté F, Thibault S, Aumage O, Namyst R, Wacremier PA. Scheduling dynamic OpenMP applications over multicore architectures. In: Eigenmann R, Supinski BR, eds. OpenMP in a New Era of Parallelism. Berlin: Springer-Verlag, 2008. 170−180. [doi: 10.1007/978-3-540-79561-2_15]

[8] Frigo M, Leiserson CE, Randall KH. The implementation of the Cilk-5 multithreaded language. ACM SIGPLAN Notices, 1998, 33(5):212−223. [doi: 10.1145/277652.277725]

[9] Long GP, Zhang JC, Fan DR. Architectural support and evaluation of Cilk language on many-core architectures. Chinese Journal of Computers, 2008,31(11):1975−1985 (in Chinese with English abstract).

[10] Ebrahimi E, Lee CJ, Mutlu O, Patt YN. Fairness via source throttling: A configurable and high-performance fairness substrate for multi-core memory systems. ACM SIGPLAN Notices, 2010,45(3):335−346. [doi: 10.1145/1735971.1736058]

[11] Das R, Mutlu O, Moscibroda T, Das C. Aérgia: A network-on-chip exploiting packet latency slack. IEEE Micro, 2011,31(1): 29−41. [doi :10.1109/MM.2010.98]

[12] Edmonds J. Scheduling in the dark (improved result). Theoretical Computer Science, 2007,235(1):109−141. [doi: 10.1145/301250.301299]

[13] Robert J, Schabanel N. Non-Clairvoyant batch sets scheduling: Fairness is fair enough. In: Arge L, Hoffmann M, Welzl E, eds. Lecture Notes in Computer Science, Berlin: Springer-Verlag, 2007. 741−753. [doi: 10.1007/978-3-540-75520-3_65]

[14] Kumar V, Fedorova A. Towards better performance per watt in virtual environments on asymmetric single-isa multi-core systems. ACM SIGOPS Operating Systems Review, 2009,43(3):105−109. [doi: 10.1145/1618525.1618538]

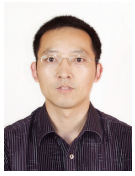
[15] Marr, S. Haupt, M, Timbermont, S, Adams B, D'Hondt T, Costanza P, de Meuter W. Virtual machine support for many-core architectures: Decoupling abstract from concrete concurrency models. In: Beresford RA, Gay S, eds. Proc. of the Programming Language Approaches to Concurrency and Communication-cEntric Software, EPTCS. 2010. 63−77. [doi: 10.4204/EPTCS.17.6]

[16] He YX, Hsu WJ, Leiserson CE. Provably efficient online non-clairvoyant adaptive scheduling. IEEE Trans. on Parallel and Distributed Systems, 2008,19(9):1263−1279. [doi: 10.1109/IPDPS.2007.370303]

[17] Pruhs K. Competitive online scheduling for server systems. ACM SIGMETRICS Performance Evaluation Review, 2007,34(4): 52−58. [doi: 10.1145/1243401.1243411]

附中文参考文献:

[9] 龙国平,张军超,范东睿.众核体系结构对 Cilk 语言的硬件支持及评测研究.计算机学报,2008,31(11):1975−1985.



曹仰杰(1976－),男,河南濮阳人,博士生,CCF 学生会员,主要研究领域为高性能计算.

伍卫国(1963－),男,教授,博士生导师,CCF 高级会员,主要研究领域为嵌入式系统,高性能计算,计算机网络,计算机应用技术.

钱德沛(1952－),男,教授,博士生导师,CCF 高级会员,主要研究领域为体系结构,网格计算,计算机网络.

董小社(1963－),男,教授,博士生导师,CCF 高级会员,主要研究领域为高性能计算机体系结构,网格计算,高性能容错计算机系统,并行计算理论.