软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(2):343−357 [doi: 10.3724/SP.J.1001.2013.04190]

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

一种面向非对称多核处理器的综合性调度算法*

陈锐忠, 齐德昱, 林伟伟, 李 剑

(华南理工大学 计算机系统研究所,广东 广州 510006)

通讯作者: 陈锐忠, E-mail: chen.rz02@mail.scut.edu.cn, http://www.scut.edu.cn

摘 要: 在非对称多核处理器上进行任务调度时,现有的操作系统调度器没有考虑其非对称性.针对单一指令集非对称多核处理器上的操作系统调度问题,首先建立线性规划模型,分析各种因素,得出行为匹配、减少迁移和负载均衡的调度原则.然后,基于调度原则提出一种综合性调度算法.该算法包括两个部分:1) 集成负载表征,提出集成行为的概念,全面衡量任务的整体性和阶段性行为;2) 基于集成行为的调度算法,有效开发非对称多核处理器的特性,能够保证各核心负载均衡,同时可以避免不必要的任务迁移.另外,该算法通过参数调整机制实现了算法的通用性.该算法是一种综合处理任务的整体性和阶段性行为,并具备通用性的调度算法.实际平台上的实验结果表明,该算法可通用于多种环境,且性能比其他对应算法提高 6%~22%.

关键词: 非对称多核处理器;操作系统调度;负载表征;负载均衡;任务迁移

中图法分类号: TP316 **文献标识码**: A

中文引用格式: 陈锐忠,齐德昱,林伟伟,李剑.一种面向非对称多核处理器的综合性调度算法.软件学报,2013,24(2):343−357. http://www.jos.org.cn/1000-9825/4190.htm

英文引用格式: Chen RZ, Qi DY, Lin WW, Li J. Comprehensive scheduling algorithm for asymmetric multi-core processors. Ruanjian Xuebao/Journal of Software, 2013,24(2):343−357 (in Chinese). http://www.jos.org.cn/1000-9825/4190.htm

Comprehensive Scheduling Algorithm for Asymmetric Multi-Core Processors

CHEN Rui-Zhong, QI De-Yu, LIN Wei-Wei, LI Jian

(Institute of Computer Systems, South China University of Technology, Guangzhou 510006, China)

Corresponding author: CHEN Rui-Zhong, E-mail: chen.rz02@mail.scut.edu.cn, http://www.scut.edu.cn

Abstract: This research focuses on the problem of operating system (OS) scheduling on asymmetric multi-core processors (AMP). A task scheduling model based on linear programming is proposed. Several attributes of AMP factors are taken into account in this model. Scheduling principles of behavior matching, migration avoiding, and load balancing are adhered as well. A comprehensive scheduling algorithm is also proposed based on the model. The algorithm has two parts: an integrated workload characterization, which proposes integrated behavior to measure the global and local behaviors of tasks comprehensively, and an integrated behavior- based scheduling algorithm, which efficiently utilizes the asymmetric multi-core processors without frequent task migration. This guarantees the load balance between cores. In addition, the algorithm achieves universality with a flexible parameter adjustment mechanism. It is an algorithm to achieve universality as well as the first to handle the global and local behaviors of tasks comprehensively. The evaluation on real platform demonstrates that the algorithm is universal for different conditions, and it always outperforms other scheduling algorithms on asymmetric multi-core processors (by 6%~22%).

Key words: asymmetric multi-core processor; operating system scheduling; workload characterization; load balancing; task migration

随着处理器的晶体管数量不断增加,其频率不断提升,已达到现有条件下的极限,计算机体系结构的发展趋势正从单核处理器转向多核处理器[1].相对于对称多核处理器,单一指令集非对称多核处理器(asymmetric

* 基金项目: 国家自然科学基金(61070015); 广东省中国科学院全面战略合作项目(2009B091300069)
收稿时间: 2011-04-21; 修改时间: 2011-08-31; 定稿时间: 2012-01-16

single-ISA multi-core processor,简称 AMP)在能耗、面积等方面有着巨大的优势,将成为未来的主流[2−5].现有操作系统调度器从单核处理器发展而来,并为对称多处理器(symmetric multi-processor,简称 SMP)做了相应扩展,不能利用 AMP 的特性和优势.这为操作系统调度带来了新的机遇和挑战.

由于 AMP 上每个核心支持同一指令集结构,任务可以在不同核心上正确执行,但由于核心间的性能异构性,任务在不同核心上的完成时间却是不同的.如何处理任务的阶段性行为特征[6−8]和 AMP 的非对称性,实现高效的操作系统调度,是该形势下的一个关键问题.近年来,有一些研究关注这一问题,但都只是片面考虑任务的行为特征,也没有提供适应不同硬件环境的机制,不能很好地解决该问题.例如,文献[9]没有考虑任务的行为特征;文献[10,11]忽略了任务的行为变化,导致任务与核心不匹配;文献[11,12]在处理行为变化的同时造成频繁的任务迁移.

因此,本文以高性能、可通用、可实用为目标,为 AMP 上的操作系统调度问题建立了线性规划模型,分析任务阶段性行为、迁移开销、核心计算能力等因素,得出行为匹配、减少迁移和负载均衡的调度原则.在此基础上提出一种综合性调度算法.它由两个部分组成:

1) 集成负载表征.本文提出集成行为的概念,用来综合度量任务整体性[6,7]和阶段性[6−8]的行为.其特点包括:① 更加全面地度量一个任务,提高其与核心匹配的精确性;② 通过可调节的参数,使调度器体现不同程度的稳定性和敏感性,从而通用于多种环境.
2) 基于集成行为的调度算法.该算法包括行为监控机制和重调度算法.其特点如下:① 将任务调度到与其集成行为最匹配的核心上执行,有效地开发了 AMP 的特性;② 筛选出稳定的行为特征,避免了不必要的任务迁移;③ 可根据环境调节的负载均衡.

该算法是一种综合度量任务的整体性和阶段性行为,并对不同环境具备通用性的调度算法.Linux 2.6.27 和多种配置的 AMD Opteron 2384 上的实验结果表明,该算法性能比其他多核处理器上的调度算法提高了 6%~22%.

本文第 1 节对问题进行建模分析,给出调度的目标和原则.第 2 节详细描述综合性调度算法,包括集成负载表征和基于集成行为的调度算法.第 3 节对所提出方法进行实验和比较分析.第 4 节介绍相关研究.第 5 节进行总结,并对未来工作进行展望.

1 任务调度模型

本文研究 AMP 上的操作系统调度问题,其关注的目标包括:

1) 性能:最小化任务的总完成时间.
2) 通用性:适用于不同的硬件环境.
3) 实用性:可应用于实际的操作系统中.

我们可以为 AMP 上的操作系统调度问题建立线性规划模型:

设 $C=\{c_1,c_2,\ldots,c_m\}$表示 m 个核心的集合,第 i 个核心的单位时间计算能力为 c_i;$W=\{w_1,w_2,\ldots,w_n\}$表示 n 个相互独立的任务的集合.文献[6−8]表明,任务行为呈现阶段性变化的特征.因此,我们把 W 的生命周期划分成阶段,同一阶段中各任务行为相对稳定;当任务行为变化巨大时,W 进入新的阶段,系统进行重调度.我们以一个阶段作为建模对象.AMP 上的核心支持单一指令集结构,因此,任务在各个核心上都可以正确执行,但由于核心的性能不同,任务在不同核心上的完成时间却是不同的.Δ是一个 $m\times n$ 的矩阵,表示上一阶段的任务分配情况:$\Delta_{ij}=1$,表示任务 w_j 分配给处理器 c_i;否则,$\Delta_{ij}=0$.当任务进入新阶段时,需求解新的分配矩阵Δ',重调度以使任务总完成时间最小.一个任务同一时间只能在一个核心上运行,由此可以得到条件(1).$A=\{a_1,a_2,\ldots,a_m\}$表示核心上的任务数量,第 i 个核心的任务数为 a_i,每个核心上的任务数量之和应等于任务总数 n,可得条件(2).

假设新阶段的持续时间 T_{total} 大于任务 j 的内存访问时间 Tj_{memory} 和迁移时间 $Tj_{\text{migration}}$ 之和,可得条件(3).新阶段任务 j 有效利用核心 i 的时间如条件(4)所示.当核心负载相差太远时,有可能出现低负载的核心空等该阶段

完成的情况.我们用 $\frac{\varepsilon}{a_i}$ 对这一情况进行近似处理:ε是一常量,当 a_i 很高,即核心负载高时,其有效利用时间较长;当a_i很低,即核心负载低时,核心将出现空转,降低其有效利用时间.当$|\Delta'_{ij}-\Delta_{ij}|=0$时,该任务没有产生迁移;反之,则需减去迁移时间.假设同一核心上的所有任务平均共享其计算能力,则新阶段任务 j 执行的指令数如条件(5)所示.新阶段执行的指令总数如条件(6)所示.

对于一个动态调度算法来说,最大化系统性能,等价于最大化新阶段执行的指令总数.因此,我们可得该问题的线性规划模型如下:

$$\text{Maximize } Inst_{\text{total}}$$

$$\text{s.t.}\begin{cases}\sum_{i=1}^{m}\Delta_{ij}=1,\sum_{i=1}^{m}\Delta'_{ij}=1 \text{ for } 1\leqslant j\leqslant n & (1)\\ \sum_{i=1}^{m}a_i=n & (2)\\ T_{\text{total}}>Tj_{\text{memory}}+Tj_{\text{migration}} \text{ for } 1\leqslant j\leqslant n & (3)\\ Time_{ij}=T_{\text{total}}-Tj_{\text{memory}}-Tj_{\text{migration}}\times|\Delta'_{ij}-\Delta_{ij}|-\frac{\varepsilon}{a_i} & (4)\\ Inst_j=\sum_{i=1}^{m}\frac{c_i}{a_i}\times\Delta'_{ij}\times Time_{ij} & (5)\\ Inst_{\text{total}}=\sum_{j=1}^{n}Inst_j & (6)\end{cases}$$

但是,在实际中无法求得该问题的最优解,原因如下:

1) T_{total},Tj_{memory},$Tj_{\text{migration}}$ 和ε的值与太多因素相关,且因不同硬件环境而异,无法准确获取;

2) 当任务的阶段性变化频繁时,求解该问题将带来巨大的开销.

因此,我们采用启发式算法来求问题的近优解.其中,条件(4)~条件(6)可合成如下等式:

$$Inst_{\text{total}}=\sum_{j=1}^{n}\sum_{i=1}^{m}\frac{c_i}{a_i}\times\Delta'_{ij}\times\left(T_{\text{total}}-Tj_{\text{memory}}-Tj_{\text{migration}}\times|\Delta'_{ij}-\Delta_{ij}|-\frac{\varepsilon}{a_i}\right) \quad (7)$$

可见,当以下条件成立时,可以取得较大的值:

$$\begin{cases}\Delta'_{ij}=1 \text{ for } (i,j)=\arg\min_{(i',j')}\left(\frac{c_{i'}}{a_{i'}}\times Tj'_{\text{memory}}\right) \text{ for } 1\leqslant i'\leqslant m,1\leqslant j'\leqslant n & (8)\\ \left|\Delta'_{ij}-\Delta_{ij}\right|\to 0 \text{ for } 1\leqslant i\leqslant m,1\leqslant j\leqslant n & (9)\\ a_i=\frac{n}{m} \text{ for } 1\leqslant i\leqslant m & (10)\end{cases}$$

基于以上讨论,调度算法应遵循以下原则:

(1) 把任务分配到与其当前阶段行为特征最匹配的核心上执行——条件(8);

(2) 减少任务迁移——条件(9);

(3) 各核心负载均衡——条件(10).

这些原则是互相冲突的,因此需要根据不同环境调整其优先级,以提高算法的通用性.

2 综合性调度算法

2.1 集成负载表征

任务在快、慢核心运行的加速比,与其产生的外部延迟密切相关,这主要由内存访问时间决定[13].因此,本文使用最后一级 cache 缺失率(last level cache miss rate,简称 LLCMR)表达任务的行为特征.访问 cache 可能是缺失或命中,LLCMR 是缺失次数在最后一级 cache 总访问次数中所占的比例,取值范围为[0,1].LLCMR 越低,任务内

存访问时间越短,计算密集程度越高,越受益于复杂的核心;LLCMR 越高,任务访问内存越多,因此可放在简单的核心上运行.

2.1.1 集成行为

任务的行为特征可分为整体行为特征(global behavior 或 average behavior,简称 GB)[6,7]和阶段性行为特征(local behavior 或 average behavior,简称 LB,也有文献称为 phase behavior)[6−8],其中,GB 用任务在整个运行期间的 LLCMR 表示,LB 用任务在当前运行阶段的 LLCMR 表示.因此,两者取值范围都是[0,1].

我们提出一个新的概念——集成行为(integrated behavior,简称 IB),用于综合衡量一个任务的 GB 和 LB.其定义如下:

$$IB=\alpha GB+(1-\alpha)LB,\alpha\in[0,1] \tag{11}$$

参数α用于在 GB 和 LB 之间进行调节,其中,α越大,GB 越重要,调度器也就越稳定;α越小,LB 越重要,调度器也就越敏感.这使得调度器通用于不同的架构.由于 GB,LB,α取值范围均为[0,1],IB 也以[0,1]为取值范围.

任务在整个运行期间的 LLCMR 采用基于复用距离的预测方法[14]获得.该方法是微体系无关的,可一次生成后整合进任务的可执行程序,终生使用.任务在当前运行阶段的 LLCMR 用性能寄存器在线捕捉[15].该方法开销极小,可在任务执行过程中随时调用.具体过程这里不再赘述.

2.2 基于集成行为的调度算法

该算法由行为监控机制和重调度算法组成,其执行模型如图 1 所示.行为监控机制循环运行,当其检测到任务行为进入新的稳定的阶段时,调用重调度算法把任务重新分配给各个核心.

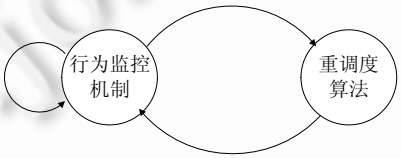


Fig.1 Execution model of algorithm

图 1 算法执行模型

2.2.1 行为监控机制

行为监控机制是控制任务迁移的有效手段.根据第 2 节给出的调度原则,当以下条件同时满足时,任务需要进行重调度:

1) 任务的行为特征变化巨大;

2) 新阶段持续稳定.

任务执行体现阶段性的行为特征[6−8].条件 1)表示任务行为与核心不匹配,需要另外找一个匹配的核心来运行;条件 2)用于过滤短暂的行为变化,以避免不必要的任务迁移.

该机制详述如下:

一个刚创建的任务只有 GB,此时 $IB=\alpha GB$,可调用第 2.2.2 节描述的算法进行调度.

此后,系统开始以 *Interval* ms 为间隔记录每个任务的 LLCMR.为了解决 cache 的冷启动效应,当任务刚启动,或者刚迁移到某核心上时,其第 1 个间隔的 LLCMR 不记录.

只有当一个运行阶段稳定了,才能进行任务迁移.因此,本方法用 LLCMR 的移动平均数表示 LB,如式(12)所示,其中,$LLCMR_i$ 表示第 i 个间隔的最后一级 cache 缺失率.

$$LB=\frac{LLCMR_1+LLCMR_2+...+LLCMR_n}{n} \tag{12}$$

任务在同一运行阶段,其行为特征会有细微的变化[6−8];当该变化超过一个阈值 $Threshhold_1$ 时,表示新的阶段出现了.本方法为每个任务维护两个 LB:当前间隔的行为特征 LB_{cur} 和上一间隔的行为特征 LB_{prev}.若下式成

立,则表示出现了新的阶段:

$$\frac{|LB_{cur}-LB_{prev}|}{LB_{prev}}>Threshhold_1 \tag{13}$$

当该运行阶段比较短暂时,不需要迁移任务.因此,继续记录两个间隔的 LLCMR,若其变化小于阈值 $Threshhold_2$,则表示该阶段已稳定,如公式(14)所示.此时,把 LB_{cur} 代入公式(11)计算 IB 值,并调用第 2.2.2 节描述的算法进行调度.

$$\frac{|LB_{cur}-LB_{prev}|}{LB_{prev}}<Threshhold_2 \tag{14}$$

2.2.2 重调度算法

文献[3,4]表明,由快、慢两种核心组成的 AMP 已可大幅提高性能.因此,本文假设 AMP 由快核心和慢核心两种核心组成.如何推广到 n 种核心,是我们下一步研究的内容.

根据第 1 节给出的调度原则,算法的基本思想是,把 IB 值低的任务调度到快核心上运行,把 IB 值高的任务调度到慢核心上运行,同时兼顾各核心可调整的负载均衡.

该算法保证系统满足以下规则:

1) 运行于快核心上的任务,IB 值小于所有运行于慢核心上的任务;

2) 各核心可调整的负载均衡.

对于规则 2),由于同类核心属于对称多核处理器,其负载均衡可使用操作系统现有的算法.本算法主要关注快、慢核心间的负载均衡.

规则 1)可转化为:快核心上任务的最高 IB 值,低于慢核心上任务的最低 IB 值.为了提高效率,算法维护两个运行队列:QF 包含所有在快核心上运行的任务,按 IB 值降序排列;QS 包含所有在慢核心上运行的任务,按 IB 值升序排列.设 F 和 S 分别表示快、慢核心的数量,TF 和 TS 分别表示在快、慢核心上运行的任务的数量,$IB(T)$表示任务 T 的 IB 值,则两个规则可转化如下:

1) $IB(QF[0])\leqslant IB(QS[0])$;

2) $\frac{TF}{F}\in\left[(1-\beta)\times\frac{TS}{S},\frac{1}{(1-\beta)}\times\frac{TS}{S}\right],\beta\in[0,1)$.

参数β用于调整负载均衡,当β较小时,算法倾向于负载均衡;当β较大时则倾向于容许负载不均衡.

系统其他任务分布如图 2 所示.IB 的取值范围为[0,1],低 IB 值的任务在快核心上与运行,如图 2 中的区域 a:[0,快核心上任务的最大 IB 值];高 IB 的任务在慢核心上运行,如图 2 中的区域 c:(慢核心上任务的最小 IB 值,1);区域 b 为[快核心上任务的最大 IB 值,慢核心上任务的最小 IB 值].该算法接受一个任务作为输入,处理流程如下:

1) 当该任务的 IB 处于区域 a 时,将其放在快核心上执行,并根据负载情况调整两个队列;

2) 当该任务的 IB 处于区域 c 时,将其放在慢核心上运行,并根据负载情况调整两个队列;

3) 当该任务的 IB 处于区域 b 时,将其放在低负载的核心上运行.

算法伪代码描述如下:

```
Procedure reschedule(任务 t) {
  if (IB(t)<IB(QF[0])) {
    t→QF;
    //负载均衡
    while (TF/F>TS/[(1−β)×S]) {
      将 QF 队首任务迁移到 QS 上;
    }
  } else if (IB(t)>IB(QS[0])) {
    t→QS;
```

```
        //负载均衡
        wihle(TF/F<(1−β)×TS/S) {
          将 QS 队首任务迁移到 QF 上;
        }
      } else {
        //放在低负载的核心上
        if (TF/F>TS/S) {
          t→QS;
        } else {
          t→QF;
        }
      }
    }
```

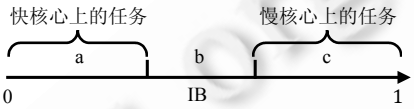


Fig.2 Distribution of other tasks

图 2 其他任务分布图

对于整体特征为存储密集型的任务,其经历的存储密集型阶段较多、较长,计算密集型阶段较少、较短.在大部分时间里,由于该任务的 GB 值很高,中等值的 LB 无法有效降低 IB 值,算法将其放在慢核心上运行;只有当该任务经历计算密集程度很高的阶段时,其 LB 值极低,从而 IB 值变低,算法将其迁移到快核心上.计算密集型程序则相反.这兼顾了原则(1)和原则(2).

对于整体特征介于计算密集型和存储密集型之间的任务,其 GB 值中等,当任务交替经历高 LB 值和低 LB 值的阶段时,算法将其交替放到慢核心和快核心上运行.这体现了原则(1).

设 N 表示任务总数.算法需要维护队列 QS 和 QF,其空间复杂度为 $O(N)$.算法的时间开销来源于在调度决策时将任务放到队列中的恰当位置,以及均衡负载时带来的任务迁移,时间复杂度为 $O(N)$.但集成行为和行为监控机制有效避免了不必要的重调度;并且任务的行为变化后,任务在队列中的位置有可能不变,此时也不会进行任务迁移,因此算法的开销并不大.这将在第 3.2.2 节的实验中得到进一步验证.

2.3 参数调整机制

如第 1 节所述,原则(1)~原则(3)互相冲突,且在不同架构的处理器上优先级不同.为了使算法在各种环境下都获得良好的性能,需要提供优先级调整的机制.

如第 2.1 节和第 2.2 节所述,该算法提供了两个参数:α和β.α用于调整原则(1)和原则(2),β用于调整原则(3).应根据实际情况设置参数,比如对于 NUMA 架构的 AMP,任务迁移开销巨大,此时可取α为接近 1 的数,以减少迁移;对于核心差异不大的 AMP,则可取β为接近 0 的数,以通过负载均衡提高性能.第 3.2.3 节给出了参数取值对算法性能的影响.

3 实验与分析

3.1 实验平台与方法

本节将综合性调度算法(下文简称 Comprehensive)与 HASS-S[10],HASS-D[11],Sample-Run[12],FF[9]和 Linux 自带的调度器进行比较.其中,Linux 调度器不对处理器的非对称性进行处理.

本文采用 Linux 2.6.27 来实现和运行上述各种算法.

本文的实验平台是一台 NUMA 架构的 4 路 AMD Opteron 2384 服务器.AMD Opteron 2384 是对称多核处理器,包含 4 个 2.7GHz 的核心.本文用 DVFS 调整各个核心的频率,以体现非对称性.为了测试算法的通用性,本文使用了 3 种配置,见表 1.本文通过实验确定(α,β)的取值,详见第 3.2.3 节.

Table 1 Test platform configurations

表 1 测试平台配置

平台	(α,β)	描述
UMA	(0.3,0)	使用 1 个 AMD Opteron 2384,其中,2 个核心运行在 1.5GHz 下,2 个核心运行在 2.7GHz 下
NUMA1	(0.8,0.3)	使用 4 个 AMD Opteron 2384,每个 AMD Opteron 2384 只使用 1 个核心,其中,2 个核心运行在 2.0GHz 下,2 个核心运行在 2.7GHz 下
NUMA2	(0.6,0.3)	使用 4 个 AMD Opteron 2384,每个 AMD Opteron 2384 只使用 1 个核心,其中,2 个核心运行在 1.5GHz 下,2 个核心运行在 2.7GHz 下

本文的测试基准程序选自 SPEC CPU2006,每个测试集包含 4 个基准程序.为了测试算法在不同软件环境下的性能,程序按行为差异分为以下 3 组(见表 2):

1) 高度差异(high diversity,简称 HD):前 2 个程序为计算密集型,后 2 个程序为存储密集型.共 3 个测试集.
2) 低度差异(low diversity,简称 LD):构成同 HD 组,但差异程度比较小.共 3 个测试集.
3) 没有差异(no diversity,简称 ND):包含 4 个一样的基准程序.共 1 个测试集.

Table 2 Benchmarks

表 2 测试基准程序

测试集	测试程序
HD1	gamess, namd, mcf, soplex
HD2	gobmk, gamess, mcf, sphinx
HD3	povray, perlbench, soplex, sphinx
LD1	astar, sjeng, gcc, libquantum
LD2	cactusADM, calculix, lbm, gemsFDTD
LD3	gromacs, bzip2, xalancbmk, bwaves
ND	omnetpp, omnetpp, omnetpp, omnetpp

我们把每个测试集复制一份(使任务数大于核心数,以使负载均衡生效),在每种调度算法下分别运行 3 次,取相应指标的平均值作为度量.当其中某个程序提前完成时,我们让其重新运行,以保持测试环境的稳定性.第 2.2.1 节所述各参数用于检测稳定的新阶段.文献[11]认为,当程序的 LLCMR 变化超过 12%时,可认为新阶段出现,并且 200 ms 是一个合理的时间间隔.因此,我们取公式(12)中的 n=3,公式(13)中的 $Threshold_1$=12%,LLCMR 的采集间隔 $Interval$=200.由文献[6−8]的实验数据可知,当处于同一阶段时,程序的 LLCMR 变化比较细微,不超过 5%,因此我们取公式(14)中的 $Threshold_2$=5%.

3.2 实验结果与分析

3.2.1 性能和通用性分析

图 3~图 5 给出了 3 个平台上测试程序的完成时间,包括每个测试程序的相对完成时间以及每个测试集的平均相对完成时间(geo-mean).为了便于比较实验结果,本文以 Linux 调度器的完成时间为基准,对数据进行归一化处理,小于 1 表示完成时间小于 Linux 调度器,否则相反.

下面对结果进行分析:

1) UMA 配置下各个算法的相对完成时间如图 3 所示.在 HD 测试集上,Comprehensive 的完成时间比 Linux 调度器短 19%~20%,也比 HASS-S 和 HASS-D 短 6%~10%;HASS-D 比 HASS-S 快 2%~3%,Sample-Run 比 FF 快 6%~10%.在 LD 测试集上,Comprehensive 的完成时间比 Linux 调度器短 15%~21%,也比 HASS-S 和 HASS-D 短 6%~12%;HASS-D 比 HASS-S 快 3%~6%,Sample-Run 比 FF 快 5%~10%.
2) NUMA1 配置下各种算法的相对完成时间如图 4 所示.在 HD 测试集上,Comprehensive 的完成时间比

Linux 调度器短 16%~17%,也比 HASS-S 和 HASS-D 短 8%~10%;HD1 上 HASS-S 比 HASS-D 好 2%, Sample-Run 和 FF 则互有长短.在 LD 测试集上,Comprehensive 的完成时间比 Linux 调度器短 11%~17%,也比 HASS-S 和 HASS-D 短 7%~13%;HASS-D 和 HASS-S 互有长短,FF 则比 Sample-Run 好 1%~ 4%.

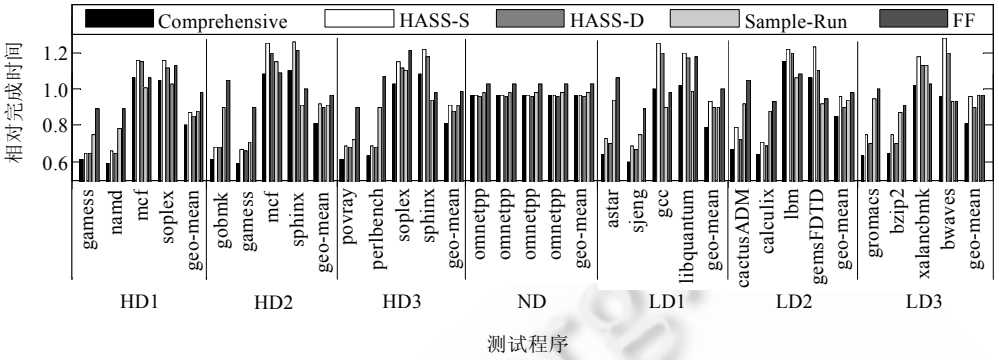


Fig.3 Relative completion time on UMA platform

图 3 UMA 平台上的相对完成时间

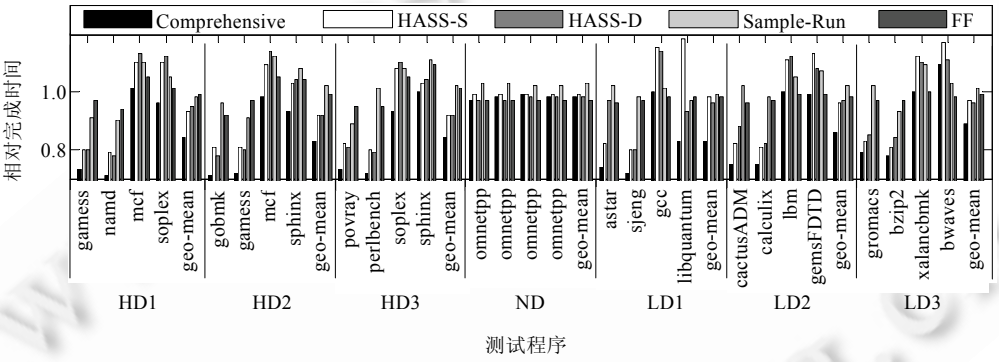


Fig.4 Relative completion time on NUMA1 platform

图 4 NUMA1 平台上的相对完成时间

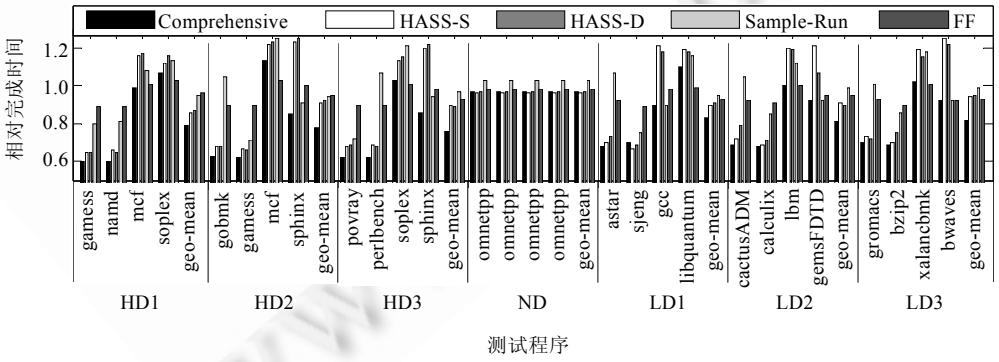


Fig.5 Relative completion time on NUMA2 platform

图 5 NUMA2 平台上的相对完成时间

3) NUMA2 配置下各种算法的相对完成时间如图 5 所示.在 HD 测试集上,Comprehensive 的完成时间比 Linux 调度器短 15%~22%,也比 HASS-S 和 HASS-D 短 8%~14%;HASS-S 比 HASS-D 快 1%~2%,Sample-Run 和 FF 则互有长短.在 LD 测试集上,Comprehensive 的完成时间比 Linux 调度器短 17%~19%,也比 HASS-S 和 HASS-D 短 8%~13%;HASS-S 与 HASS-D 互有长短,FF 比 Sample-Run 快 2%~6%.

4) 对于 ND 数据集,3 种配置下 Comprehensive 表现接近 HASS-S 和 HASS-D,相对于 Linux 调度器只有 3%的提高.这是因为程序的行为没有差异,Linux 调度器已足以解决问题,并不需要其他处理.

由实验结果可以得出以下结论:

1) 在 3 种配置下进行任务调度,Comprehensive 的完成时间小于其他算法.特别是当程序整体行为差异较小时(LD 测试集),HASS-S 和 HASS-D 未能很好地区分;而 Comprehensive 全面度量了任务的行为,且过滤了不稳定的阶段,性能明显优于其他算法.这说明了综合性调度算法的性能优势.

2) HASS-S,HASS-D,Sample-Run 和 FF 在不同硬件环境下性能表现不同,且互有长短;而 Comprehensive 得益于灵活的参数调整机制,可根据不同环境的特性调节各个原则的优先级,兼顾行为匹配,减少迁移和负载均衡,从而在每个环境下都有性能提高.这说明了综合性调度算法的通用性.

3.2.2 开销分析

相对于 Linux 调度器,AMP 上的操作系统调度算法的开销主要来源于两方面:

1) 调度决策开销.本文采用决策运算的 CPU 时间占任务运行总 CPU 时间的百分比来度量.

2) 任务迁移开销.本文采用每分钟 CPU 时间的跨类型(即快、慢核心间)任务迁移次数来度量.

测试集在 3 种配置下的调度决策开销比较如图 6 所示,任务迁移开销比较如图 7 所示.

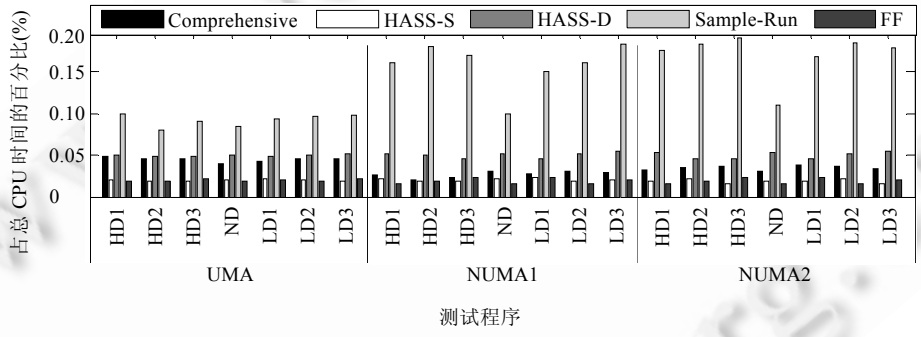


Fig.6 Comparison of schedule decision overhead on three platforms

图 6 3 种平台下的调度决策开销比较

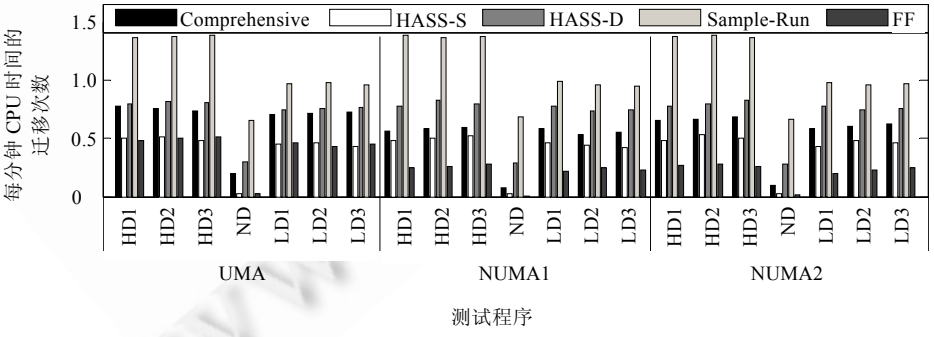


Fig.7 Comparison of task migration overhead on three platforms

图 7 3 种平台下的任务迁移开销比较

下面给出结果分析.

1) Comprehensive 的调度决策开销为 0.02%~0.048%,任务迁移开销为 0.2 次/分钟~0.78 次/分钟.这一开销给系统造成的负担很小,且 Comprehensive 具有明显的性能和通用性优势,是可以接受的.这验证了第 2.2.2 节中对算法复杂度的分析.
2) Sample-Run 的开销最大,HASS-D 次之,HASS-S 和 FF 开销最小,Comprehensive 的开销介于 HASS-S 和 HASS-D 之间.这是因为 HASS-S 和 FF 不对任务的阶段性行为做处理,调度决策和任务迁移次数较少.Sample-Run 和 HASS-D 根据任务的阶段性行为变化做调度决策,因此开销比较大.另外,Sample-Run 在做调度决策时需要进行采样——把任务迁移到不同核心上运行并记录其 IPC 值,这使得调度决策时间和任务迁移次数大量增加.而 Comprehensive 综合考虑了任务的整体和阶段性行为,并过滤掉了不稳定的阶段性行为,因此可用少于 HASS-D 的调度决策和任务迁移次数来处理任务行为变化.这进一步解释了 Comprehensive 在第 3.2.1 节中的性能表现.
3) HASS-S 和 HASS-D 在不同配置下开销变化不大.Sample-Run 的任务迁移次数一致,但 NUMA 架构下的调度决策开销大于 UMA 架构.这是因为 NUMA 架构下任务迁移开销较大,增加了采样的时间.FF 则采用 NUMA-Aware Thread Migration 技术减少了 NUMA 下跨节点的任务迁移.而 Comprehensive 在不同配置下开销不同,它提供灵活的参数调整机制,在 UMA 架构下倾向于负载均衡和核心匹配,在 NMUA 架构下则优先限制任务迁移,这使得 Comprehensive 在各种环境下都表现出优秀的性能.这也体现了算法的通用性.

3.2.3 参数灵敏度分析

该实验分析(α,β)的取值对算法性能的影响.图 8~图 10 分别给出了 UMA,NUMA1,NUMA2 平台上各程序集实际完成时间随(α,β)取值的变化.

由实验结果可知:

1) UMA 平台上,算法在(α,β)=(0.3,0)时取得最佳性能.这是因为 4 个核心处于同一处理器上,共享 6MB 的 L3 Cache,任务迁移开销较小;同时,快、慢核心的频率相差近 1 倍,任务与核心的匹配程度对性能的影响较大.此时,参数设定应优先任务核心匹配和负载均衡,容许任务迁移.

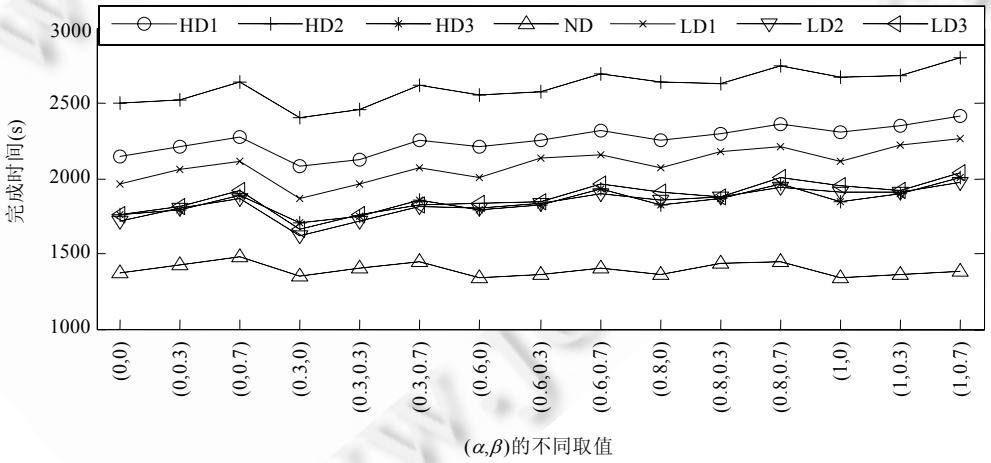


Fig.8 Variation of actual completion time of workloads on UMA platform

图 8 UMA 平台上各程序集的实际完成时间变化

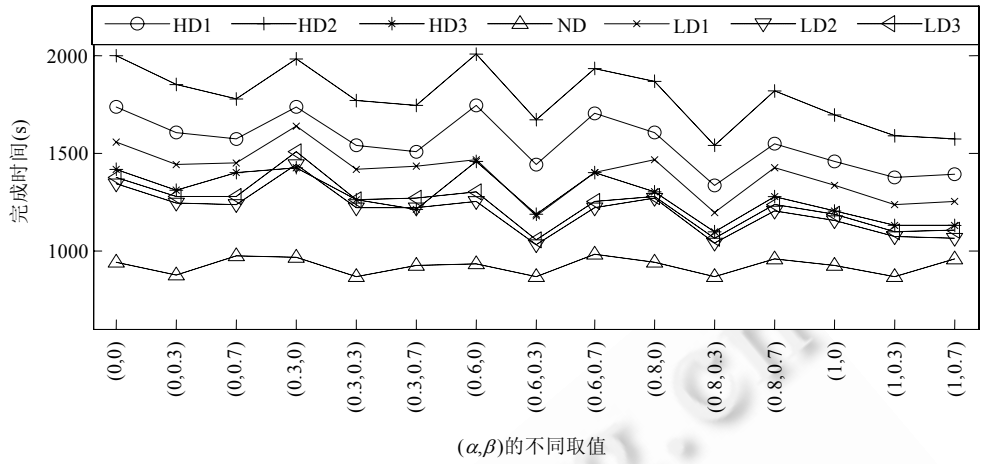


Fig.9 Variation of actual completion time of workloads on NUMA1 platform

图 9 NUMA1 平台上各程序集的实际完成时间变化

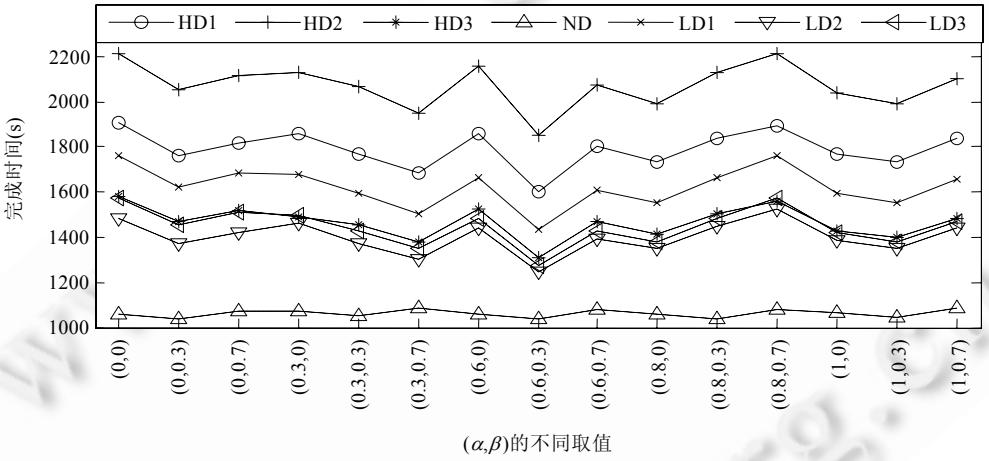


Fig.10 Variation of actual completion time of workloads on NUMA2 platform

图 10 NUMA2 平台上各程序集的实际完成时间变化

2) NUMA1 平台上,算法在(α,β)=(0.8,0.3)时取得最佳性能.这是因为 4 个核心处于不同的处理器上,且测试平台为 NUMA 架构,任务迁移开销较大;同时,快、慢核心性能仅相差 0.7GHz,任务与核心的匹配程度对性能的影响不大.此时,参数设定应优先控制任务迁移,任务核心匹配次之,并放宽对负载均衡的限制.

3) NUMA3 平台上,算法在(α,β)=(0.6,0.3)时取得最佳性能.这是因为 4 个核心处于不同的处理器上,且测试平台为 NUMA 架构,任务迁移开销较大;同时,快、慢核心性能相差近 1 倍,任务与核心的匹配程度对性能的影响较大.此时,参数设定应优先保持任务核心匹配,控制任务迁移.

3.3 实验总结

基于以上实验,我们可以得出以下结论:

1) 性能:相对于其他算法,综合性调度算法具有明显优势,能够把任务总完成时间缩短 6%~22%.

2) 通用性:HASS-S,HASS-D,Sample-Run,FF 等算法在不同环境下完成时间不同,且互有长短;综合性调度

算法则在 3 种环境下都表现出优于其他算法的性能,具有很强的通用性.

3) 开销:综合性调度算法的调度决策开销为 0.02%~0.048%,任务迁移开销为 0.2 次/分钟~0.78 次/分钟,介于 HASS-S 和 HASS-D 之间,这给系统造成的负担很小,是可以接受的.

4 相关研究

AMP 是一个新生的体系结构[2−5].传统的多核调度研究主要关注对称多核处理器和对称多处理器,这类平台上的调度算法主要是处理公平性[16]、负载均衡[17]等,没有针对 AMP 的非对称性做处理,无法开发其优势.

对于 AMP 的调度问题,之前的工作主要是独立任务[18−20]和依赖任务[21]的调度.这些方法都依赖于任务执行时间等参数已知,但在操作系统调度时,这些参数无法获得.而且它们所采用的遗传算法等开销太大,不适合操作系统线程级的调度.因此,这些方法无法应用于实际的操作系统.

本文讨论的 AMP 上的操作系统调度问题,需要可以实际应用的解决方案.操作系统应该在代码容量等因素受到限制的情况下,充分利用 AMP 的特性和任务行为特征进行优化.根据对任务行为特征的处理情况,现有的研究工作可分成以下 3 类:

(1) 不处理任务的行为特征

文献[9]优先将任务调度到快核心上,同时保持核心负载与其频率成正比,并对 NUMA 下跨节点的任务迁移做了限制.该方法考虑了核心的非对称性,开销也比较小;但没有处理任务的行为特征,从而容易出现把快核心分配给存储密集型任务的情况,无法充分开发 AMP 的特性和优势.

(2) 根据任务的整体行为进行调度

文献[10,11]提出的 HASS-S 算法基于任务的复用距离,离线预测了任务在快、慢核心上的平均加速比,在调度时结合核心的负载情况,将平均加速比大的任务分配给快核心.

该方法忽略了任务阶段性的行为变化[8−10],导致其与核心匹配度不高,影响性能.特别是当几个任务的整体行为接近时,它们的阶段性行为可能差异很大,此时,该方法无法将任务调度到与其当前行为最匹配的核心上,降低了性能.这在第 3.2.1 节的实验中得到了验证.

(3) 根据任务的阶段性行为进行调度

文献[12]提出了“采样-运行(sample-run)”的方法——周期性地重复采样和运行阶段:在采样阶段,把任务放到各种核心上试运行,收集任务阶段性行为信息;在运行阶段,根据这些信息做重调度决策.该方法的问题在于采样阶段带来的开销——由于核心的性能异构性,容易出现负载不均衡,以及多个已在快核心上完成采样的任务争用慢核心的情况.本文的算法不需要采样,没有这个问题.

文献[11]提出的 HASS-D 算法用性能寄存器在线检测各任务的阶段性行为,并根据行为的变化进行重调度.这类方法都存在一个问题:当任务的行为特征变化频繁时,将出现频繁的任务迁移,从而造成巨大开销.这在第 3.2.2 节的实验中得到了验证.

同时,这 3 类方法都没有对 AMP 上的操作系统的调度问题进行建模分析,也没有提供适应不同硬件环境的机制.

综上所述,现有方法并不能很好地解决 AMP 上的操作系统调度问题.

本文提出的综合性调度算法以高性能、可通用、可实用为目标,通过集成行为,全面衡量了任务的整体和阶段性行为,把任务调度到与其行为最匹配的核心上运行;结合行为监控机制,避免不必要的任务迁移,重调度算法同时保证各核心负载均衡.此外,该算法提供灵活的参数调整机制,可根据实际环境调节行为匹配、减少迁移和负载均衡这 3 个原则的优先级,从而具有通用性.另外,该算法不需要依赖于任务执行时间等操作系统调度时无法得到的参数,开销也在可接受的范围内,可在实际的操作系统中应用.

另外,有一些研究从不同于本文的角度开发了 AMP 的特性.文献[22]根据任务的线程级并行程度来做调度决策,文献[23]则假设同时创建的线程执行时间相同,把剩余时间长的线程优先调度到快核心上.当系统中单线程任务居多时,这些方法不再适用.但本文的方法与这些研究并不冲突,可以互相结合.文献[24]朝着这个方向做

了尝试,这也是我们下一步研究的内容.

最后,有些研究假设多核处理器的性能非对称性来源于制作工艺偏差,此时,核心性能差异不大,但体现出了动态的非对称性.文献[25]对这类平台上的操作系统调度和电源管理方法进行了比较.在本文讨论的问题中,核心的非对称性源于其架构的不同.也有些多核处理器具有功能非对称性[26],如 IBM CELL BE,这类处理器由一组对称的通用CPU和加速器(如GPU等)组成.由于每个核心支持的指令集不同,任务只能在指定核心上运行.此类平台上的调度问题不在本文的讨论范围内.

5 总结与展望

本文对 AMP 上的操作系统调度问题进行分析讨论,建立了任务调度模型,给出了调度的目标和原则,并基于调度原则提出了综合性调度算法.该算法提出了集成行为的概念,用于综合度量任务的整体行为和阶段性行为.本文基于集成行为构造了有效的行为监控机制和重调度算法,有效地开发了处理器的非对称性,能够保证各核心的负载均衡,同时可以筛选出稳定的运行阶段,避免不必要的任务迁移.同时,该算法提供了参数调整机制,可根据不同环境调节各个调度原则的优先级,从而具有通用性.该算法是一种综合度量任务的整体行为和阶段性行为,并可通用于多种环境的调度算法.

操作系统调度问题应用驱动的特点十分明显,任务调度的研究成果最终要应用到实际平台上.操作系统应该在代码容量等因素受到限制的情况下,充分利用AMP的特性和任务行为特征进行优化.我们在实际平台上实现了综合性调度算法并与同类算法进行比较,实验结果表明:综合性调度算法在性能、通用性上具有明显优势,并且开销也在可接受的范围内.

度量 fork-join,pipeline 等执行模型对程序行为的影响,对 AMP 的不同核心进行协同调度,将是我们下一步的研究重点.

References:

[1] Deng YD, Jing N, Xiong W. Nested loop join optimization based on radix-join in chip multi-processor. Journal of Computer Research and Development, 2010,47(6):1079−1087 (in Chinese with English abstract).

[2] Hill MD, Marty MR. Amdahl's law in the multicore era. IEEE Computer, 2008,41(7):33−38. [doi: 10.1109/MC.2008.209]

[3] Fedorova A, Saez JC, Shelepov D, Prieto M. Maximizing power efficiency with asymmetric multicore systems. Communications of the ACM, 2009,52(12):48−57. [doi: 10.1145/1610252.1610270]

[4] Kumar R, Tullsen DM, Ranganathan P, Jouppi NP, Farkas KI. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In: Dubois M, Bode A, eds. Proc. of the 31st Annual Int'l Symp. on Computer Architecture. Munich: IEEE Computer Society Press, 2004. 64−75. [doi: 10.1109/ISCA.2004.1310764]

[5] Kumar R, Tullsen DM, Jouppi NP, Ranganathan P. Heterogeneous chip multiprocesors. IEEE Computer, 2005,38(11):32−38. [doi: 10.1109/MC.2005.379]

[6] Tam DK, Azimi R, Soares LB, Stumm M. RapidMRC: Approximating L2 miss rate curves on commodity systems for online optimizations. In: Soffa ML, Irwin MJ, eds. Proc. of the 14th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. Washington: ACM Press, 2009. 121−132. [doi: 10.1145%2f1508284.1508259]

[7] Sherwood T, Perelman E, Hamerly G, Sair S, Calder B. Discovering and exploiting program phases. IEEE Micro, 2003,23(6): 84−93. [doi: 10.1109/MM.2003.1261391]

[8] Dhodapkar AS, Smith JE. Comparing program phase detection techniques. In: Smith BM, ed. Proc. of the 36th Annual IEEE/ACM Int'l Symp. on Microarchitecture. San Diego: IEEE Computer Society Press, 2003. 217−227. [doi: 10.1109/MICRO.2003.1253197]

[9] Li T, Baumberger D, Koufaty DA, Hahn S. Efficient operating system scheduling for performance-asymmetric multi-core architectures. In: Verastegui B, ed. Proc. of the 2007 ACM/IEEE Conf. on Supercomputing. Reno: ACM Press, 2007. 1−11. [doi: 10.1145/1362622.1362694]

[10] Shelepov D, Alcaide JCS, Jeffery S, Fedorova A, Perez N, Huang ZF, Blagodurov S, Kumar V. HASS: A scheduler for heterogeneous multicore systems. ACM SIGOPS Operating Systems Review, 2009,43(2):66−75. [doi: 10.1145/1531793.1531804]

[11] Saez JC, Shelepov D, Fedorova A, Fedorova A, Prieto M. Leveraging workload diversity through OS scheduling to maximize performance on single-ISA heterogeneous multicore systems. Journal of Parallel and Distributed Computing, 2011,71(1):114−131. [doi: 10.1016/j.jpdc.2010.08.020]

[12] Becchi M, Crowley P. Dynamic thread assignment on heterogeneous multiprocessor architectures. In: Banerjee U, ed. Proc. of the 3rd Conf. on Computing Frontiers. Ischia: ACM Press, 2006. 29−40. [doi: 10.1145/1128022.1128029]

[13] Koufaty D, Reddy D, Hahn S. Bias scheduling in heterogeneous multi-core architectures. In: Morin C, Muller G, eds. Proc. of the 5th European Conf. on Computer Systems. New York: ACM Press, 2010. 125−138. [doi: 10.1145%2f1755913.1755928]

[14] Zhong Y, Shen X, Ding C. Program locality analysis using reuse distance. ACM Trans. on Programming Languages and Systems, 2009,31(6):21−59. [doi: 10.1145%2f1552309.1552310]

[15] Azimi R, Tam DK, Soares L, Stumm M. Enhancing operating system support for multicore processors by using hardware performance monitoring. ACM SIGOPS Operating Systems Review, 2009,43(2):56−65. [doi: 10.1145/1531793.1531803]

[16] Li T, Baumberger D, Hahn S. Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin. In: Reed DA, Sarkar V, eds. Proc. of the 14th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. Raleigh: ACM Press, 2009. 65−74. [doi: 10.1145/1504176.1504188]

[17] Hofmeyr S, Iancu C, Blagojević F. Load balancing on speed. In: Govindarajan R, Padua DA, Hall MW, eds. Proc. of the 15th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. Bangalore: ACM Press, 2010. 147−158. [doi: 10.1145/1693453.1693475]

[18] Chen J, John LK. Efficient program scheduling for heterogeneous multi-core processors. In: Kahng AB, ed. Proc. of the 46th Annual Design Automation Conf. San Francisco: ACM Press, 2009. 927−930. [doi: 10.1145/1629911.1630149]

[19] Chen J, John LK. Energy-Aware application scheduling on a heterogeneous multi-core system. In: Christie D, Lee A, Mutlu O, Zorn BG, eds. Proc. of the 4th Int'l Symp. on Workload Characterization. Seattle: IEEE Computer Society Press, 2008. 5−13. [doi: 10.1109/IISWC.2008.4636086]

[20] Jiang JC, Wang TQ. Task scheduling algorithm for heterogeneous multi-core processor. Computer Engineering and Applications, 2009,45(33):52−56 (in Chinese with English abstract).

[21] Peng MM, Xu LC, Wang Y. Task allocation and energy on heterogeneous multi-core processors. Application Research of Computers, 2010,47(6):1729−1731 (in Chinese with English abstract).

[22] Saez JC, Fedorova A, Prieto M, Vegas H. Operating system support for mitigating software scalability bottlenecks on asymmetric multicore processors. In: Amato N, ed. Proc. of the 7th ACM Int'l Conf. on Computing Frontiers. Bertinoro: ACM Press, 2010. 31−40. [doi: 10.1145/1787275.1787281]

[23] Lakshminarayana NB, Lee J, Kim H. Age based scheduling for asymmetric multiprocessors. In: Pinfold W, ed. Proc. of the Conf. on High Performance Computing Networking, Storage and Analysis. Portland: ACM Press, 2009. 1−12. [doi: 10.1145/1654059.1654085]

[24] Saez JC, Prieto M, Fedorova A, Blagodurov S. A comprehensive scheduler for asymmetric multicore systems. In: Morin C, Muller G, eds. Proc. of the 5th European Conf. on Computer Systems. New York: ACM, 2010. 139−152. [doi: 10.1145/1755913.1755929]

[25] Winter JA, Albonesi DH, Shoemaker CA. Scalable thread scheduling and global power management for heterogeneous many-core architectures In: Salapura V, Gschwind M, Knoop J, eds. Proc. of the 19th Int'l Conf. on Parallel Architectures and Compilation Techniques. Vienna: ACM Press, 2010. 29−40. [doi: 10.1145/1854273.1854283]

[26] Huang GR, Zhang P, Wei GB. Key techniques of multi-core processor and its development trends. Computer Engineering and Design. 2009,30(10):2414−2418 (in Chinese with English abstract).

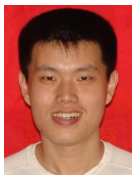
附中文参考文献:

[1] 邓亚丹,景宁,熊伟.多核处理器中基于 Radix-Join 的嵌套循环连接优化.计算机研究与发展,2010,47(6):1079−1087.

[20] 蒋建春,汪同庆.异构多核处理器的任务调度算法.计算机工程与应用,2009,45(33):52−56.

[21] 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.计算机应用研究,2010,57(5):1729−1731.

[26] 黄国睿,张平,魏广博.多核处理器的关键技术及其发展趋势.计算机工程与设计,2009,30(10):2414−2418.



陈锐忠(1985－),男,广东揭阳人,博士生,主要研究领域为计算机体系结构,系统软件.

E-mail: chen.rz02@mail.scut.edu.cn

齐德昱(1959－),男,博士,教授,博士生导师,CCF 会员,主要研究领域为软件开发方法,分布式系统,系统结构.

E-mail: qideyu@gmail.com

林伟伟(1980－),男,博士,副教授,主要研究领域为分布式系统,云计算.

E-mail: linww@scut.edu.cn

李剑(1976－),男,博士,讲师,主要研究领域为数据挖掘,信息集成,社会网络.

E-mail: lijian767@hotmail.com