软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(8):2408−2424 [doi: 10.13328/j.cnki.jos.006190]

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

# 基于强化学习的温度感知多核任务调度*

杨世贵[1], 王媛媛[1,2,5], 刘韦辰[3], 姜 徐[4], 赵明雄[1], 方 卉[1], 杨 宇[1], 刘 迪[1,3]

[1](云南大学 软件学院,云南 昆明 650504)

[2](中国科学院 信息工程研究所,北京 100093)

[3](School of Computer Science and Engineering, Nanyang Technological University, Singapore)

[4](东北大学 计算机科学与工程学院,辽宁 沈阳 110169)

[5](中国科学院大学 网络空间安全学院,北京 100049)

通讯作者: 刘迪, E-mail: dliu@ynu.edu.cn

**摘 要**: 随着计算机中内核数量的增多,温度感知的多核任务调度算法成为计算机系统中的一个研究热点.近年来,机器学习在各个领域展现出巨大的潜力,很多基于机器学习的系统温度管理研究工作应运而生.其中,强化学习因其较强的自适应性,被广泛地运用于温度感知的任务调度算法中.然而,目前基于强化学习的温度感知任务调度算法系统建模不够准确,很难做到温度、性能和复杂度的较好权衡.因此,提出一种基于强化学习的多核温度感知调度算法——ReLeTA.在该算法中提出了更全面的状态建模方式和更加有效的奖励函数,从而帮助系统进一步降低温度.实验部分通过 3 个不同的真实计算机平台验证该方法,实验结果表明了该方法的有效性以及可扩展性,与现有方法相比,ReLeTA 可以更好地控制系统温度.

**关键词**: 温度感知;多核系统;强化学习;Q-Learning

**中图法分类号**: TP316

中文引用格式: 杨世贵,王媛媛,刘韦辰,姜徐,赵明雄,方卉,杨宇,刘迪.基于强化学习的温度感知多核任务调度.软件学报,2021,32(8):2408−2424. http://www.jos.org.cn/1000-9825/6190.htm

英文引用格式: Yang SG, Wang YY, Liu WC, Jiang X, Zhao MX, Fang H, Yang Y, Liu D. Temperature-aware task scheduling on multicores based on reinforcement learning. Ruan Jian Xue Bao/Journal of Software, 2021,32(8):2408−2424 (in Chinese). http://www.jos.org.cn/1000-9825/6190.htm

## Temperature-aware Task Scheduling on Multicores Based on Reinforcement Learning

YANG Shi-Gui[1], WANG Yuan-Yuan[1,2,5], LIU Wei-Chen[3], JIANG Xu[4], ZHAO Ming-Xiong[1], FANG Hui[1], YANG Yu[1], LIU Di[1,3]

[1](School of Software, Yunnan University, Kunming 650504, China)

[2](Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China)

[3](School of Computer Science and Engineering, Nanyang Technological University, Singapore)

[4](School of Computer Science and Engineering, Northeastern University, Shenyang 110169, China)

[5](School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China)

**Abstract**: With the increase of the number of cores in computers, temperature-aware multi-core task scheduling algorithms have become a research hotspot in computer systems. In recent years, machine learning has shown great potential in various fields, and thus many work using machine learning techniques to manage system temperature have emerged. Among them, reinforcement learning is widely used for temperature-aware task scheduling algorithms due to its strong adaptability. However, the state-of-the-art temperature-aware task

* 基金项目: 国家自然科学基金(61902341)
Foundation item: National Natural Science Foundation of China (61902341)
本文由“泛在嵌入式智能系统”专题特约编辑郭兵教授、王泉教授、邓庆绪教授、陈铭松教授、张凯龙副教授推荐.
收稿时间: 2020-07-24; 修改时间: 2020-09-07; 采用时间: 2020-11-02; jos 在线出版时间: 2021-02-07

scheduling algorithms based on reinforcement learning do not effectively model the system, and it is difficult to achieve a better trade-off among temperature, performance, and complexity. Therefore, this study proposes a new multi-core temperature-aware scheduling algorithm based on reinforcement learning—ReLeTA. In the new algorithm, a more comprehensive state modeling method and a more effective reward function are proposed to help the system further reduce the temperature. Experiments are conducted on three different real computer platforms. The experimental results show the effectiveness and scalability of the proposed method. Compared with existing methods, ReLeTA can control the system temperature better.

**Key words**: temperature-aware; multicore system; reinforcement learning; Q-Learning

随着计算机在日常生活中的普及,人们对其高性能和便携性的需求日渐增强,由此推动着现代计算机不断向多内核、高集成度的方向发展.芯片中晶体管尺寸缩小(最新晶体管达到 nm 级别),因此在同一尺寸内可以放置更多的处理器.高集成度让多核系统的处理器比以往具有更高的功率密度,可以带来较高的性能,但也使系统的运行温度大幅度提高且难以耗散,从而导致了芯片的老化不均匀,加速芯片磨损和故障,降低系统的可靠性以及性能.通常采用两种方法来帮助多处理器系统有效的管理系统温度.

1) 物理方式:通过加散热效果更好的散热设备来降低系统温度,然而散热效果更好的散热设备不可避免地提高系统的整体制造成本以及系统的能耗,从而使得芯片成本效益降低;
2) 软件方式:通过设计一个温度感知的程序调度算法来优化系统温度,即:任务进入系统时,调度算法为程序选择合适的处理器来运行程序,从而达到控制系统温度的目的.

因灵活性高且不需要任何外部设备的辅助,基于软件方式的温度控制方法一直是多处理器研究领域的一个热门研究课题.

早期的研究多采用基于线性规划[1,2]、动态规划[3]或者启发式算法[4]设计温度感知的调度算法,然而随着计算机内核数量的快速增长,以及各种复杂计算任务的出现,传统的温度感知调度算法不能够有效地适应不同的复杂计算环境[5].与此同时,随着机器学习[6]的兴起,机器学习技术在不同的领域展现出了很强的自我学习能力,因此有很多工作开始研究基于机器学习的温度感知调度算法[7].使得温度感知调度算法研究工作朝着智能化的方向发展.首先是基于监督学习的方法:线性回归[8,9]和分类算法[10],虽然这些方法在特定的情况下取得了很好的效果,但是模型的准确率依赖高质量及多样化的标签数据,且训练模型泛化性能不佳,即,针对一个硬件训练的方法很难直接使用到新的硬件平台.另一方面,强化学习的出现,在一定程度上克服了监督学习存在的数据依赖问题以及模型泛化性弱的问题.强化学习不需要大量的训练数据,而通过和运行环境的动态交互来学习一个策略,在面对完全陌生的情况时,强化学习可以根据自己学习的策略来进行最优的决策,比监督学习的方法灵活性更强.出于以上原因,强化学习被广泛应用于各种复杂的动态决策的场景,并且在各个领域取得很好的效果,例如,DeepMind 在 2015 年提出的强化学习玩游戏完胜人类[11],DeepMind 提出的基于强化学习的 AlphaZero 击败了顶级的职业围棋选手[12],围棋在之前一直被认为是一个复杂度很高、机器很难击败人类的项目.计算机系统状态有较强的动态性以及程序调度本身是一个复杂的决策问题,适合采用强化学习来解决.一些工作已经研究了基于强化学习的温度感知任务调度算法,但是目前存在的基于强化学习的系统温度优化方法中均存在一些问题,如环境建模不合理、奖励设置不合理,因此未能够使基于强化学习的温度感知调度方法达到最好性能,此部分将在后文中详细讨论.

针对现有方法中存在的问题,针对多处理器系统,我们提出了一种全新的基于强化学习的温度感知任务调度算法来进行系统温度的管理和控制,根据其英文名(reinforcement learning temperature-aware task allocation)将其命名为 ReLeTA.本文具体的研究贡献如下:

1) 针对现有方法存在的缺点,本文提出了一种新的系统状态建模方法,新状态可以更准确地反映系统运行状况与温度之间的关系;同时,提出了一种新的奖励函数用于协助优化强化学习算法;对比了现有的两种基于强化学习的研究方法.实验证明,我们的建模方式更加有效.基于新的状态模型和奖励函数,本文提出了一种新的基于强化学习的温度感知多核任务分配算法——ReLeTA.
2) 我们在不同的真实系统上对所设计的算法进行了评估,对比了现有的两个基于强化学习的温度感知

任务调度算法和 Linux 默认的调度算法,本文所提新方法能够在基本不损失性能的基础上降低系统温度:最好情况下降低平均峰值温度 6°C,降低平均温度 3°C.

本文第 1 节讨论温度感知调度方面的相关工作.第 2 节介绍了强化学习相关的理论知识方便理解本文的贡献.第 3 节详细讨论现有方法的缺陷,以此说明新方法的必要性.第 4 节详细介绍本文算法的模型和参数设置.第 5 节详细展示了在真实平台的实验结果.第 6 节对本文进行总结并讨论了未来工作的方向.

## 1 相关工作

目前,温度管理技术在学术界和工业界都受到了极大的关注,已有很多研究致力于控制芯片的温度[7].早期的研究多采用基于线性规划[1,2]、动态规划[3]或者启发式算法[4]设计温度感知的任务调度算法,然而随着计算机内核数量的快速增长,以及各种复杂计算任务的出现,传统的温度感知调度算法不能够有效地适应不同的复杂计算环境.

随着机器学习的发展,各类机器学习算法被广泛地应用在各个领域内,解决着大量过去难以处理的问题,展现着巨大的应用潜力.因此,更多的研究将 ML(machine learning[6])利用到多核系统温度管理领域,提出了多种系统温度优化算法.本文总结了目前主流的温度优化方法,分别从静态和动态两方面介绍.

- 静态方法通常利用监督学习,通过训练大量的数据得到温度、性能预测模型,在设计系统的过程中训练好一个固定的算法模型,并将其应用于任务调度中[13].文献[14]基于自适应学习树算法来预测空闲期开始时间并选择性地将其调至低功耗睡眠状态,从而达到降低系统功耗、降低系统温度的目的.另外,还有一种使用贝叶斯分类技术的电源管理框架[15],利用监督学习来预测处理器的性能状态,查找并执行最优的电源管理操作,从而达到降低系统温度的目的.文献[9]提出一种基于线性回归模型的方法,通过采集任务运行时性能计数器的值和传感器的值来训练预测模型进行任务映射.以上使用监督学习的方法需要大量的训练数据,然而高质量且多样化的训练数据集往往难以获取,导致了这些方法对于数据集外其他任务的分配效果未必能够达到预期目标;同时,基于一个硬件训练的模型很难移植到其他不同的硬件上.
- 动态方法区别于静态方法,其能够在运行过程中持续地学习改进自身算法.在温度感知调度中,动态方法能够在运行过程中优化温度,不会受到训练数据和系统硬件的局限.目前已有的动态算法,如借用松弛技术动态管理峰值温度[16]、基于神经网络的自适应技术来降低温度[17]、基于强化学习的自适应技术通过迭代温度变化来控制任务映射以优化核心温度等.其中,强化学习能够更灵活地处理动态决策问题,在性能和复杂性方面达到一个均衡.因此,利用强化学习进行芯片温度管理的研究工作逐渐增多.

一种启发式的基于强化学习的温度管理方法在文献[18]中被提出,该方法通过设置温度阈值和功耗阈值来限制状态空间和动作空间.该方法每个内核有一个属于自己的 Q 表,能有效提高算法的收敛速度,但是随着内核的增多,存储 Q 表会带来巨大的空间开销.此外,如何有效保证每个内核的 Q 表都能收敛是最大的问题.文献[19]提出一种根据当前芯片温度状态来预测并执行能最大限度降低未来最高温度的任务分配策略算法,但是该方法算法模型的设置不合理,对于奖励和状态模型的设置过于简单,没有全面考虑到影响芯片温度变化的多个因素,造成了算法的效果和性能不能达到最佳,且该方法只是在仿真平台运行和测试.文献[20]设计了一个基于强化学习的温度感知调度算法,同时加入了系统动态调频方法,该方法基于温度循环,并将系统的延迟作为奖励的一个部分.虽然该方法同时考虑了温度和性能,但因其奖励函数设置不合理导致其很难实现温度和性能的最佳权衡.后面我们在真实的系统上对比了这两种方法.

温度感知的调度方法有些时候也可以和系统能耗控制方法结合在一起,首先是动态电源管理(dynamic power management,简称 DPM)[21]技术根据工作设备的负载情况动态的选择各个内核的电源状态,切换各个处理器的工作模式(活动模式或低功耗模式等).另一种是动态电压频率调整(DVFS)[22]技术,利用了芯片的特性,可以根据内核的实际功耗来调节其工作电压或频率.这两项技术的本质都是降低系统功耗,达到有效控制芯片整体温度的目的.在后面的工作中,我们会考虑将 DPM 和 DVFS 技术与 ReLeTA 相结合,探索更优的任务分配算法.

## 2 基础知识

### 2.1 强化学习概述

强化学习属于机器学习范畴内的一个重要分支,其本质是通过与环境连续的交互来寻找一个最优的决策.强化学习中包含有智能体(agent)、环境(environment)、状态(state)、动作(action)和奖励(reward).强化学习模仿了生物通过与外界交互来在环境中进行决策的行为(如图1所示):智能体通过执行动作来影响环境,环境接收到新动作之后会产生新的状态,并对刚才的动作进行一个奖励反馈,根据所返回的新的状态来决定其需要进行的动作.强化学习的训练过程就是智能体不断地根据当前环境状态进行动作决策,同时根据环境反馈的奖励值及时改进其决策的过程,最终的目标是找寻一个最优策略,即智能体能够从环境状态改变的过程中累积尽可能多的奖励值.

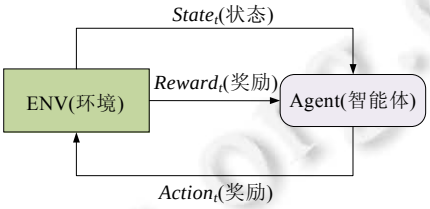



Fig.1 Reinforcement learning

图1 强化学习

强化学习与机器学习中其他需要大量训练数据的学习算法不同,它不需要使用大量的数据样本,而是通过不断的试错进行自我学习,这种自动进行学习和决策的方法能够解决监督学习方法所面临的由于系统动态性所导致的训练数据不全面的问题,更适用于多核温度感知调度算法.

### 2.2 Q-Learning算法

随着强化学习的发展,在Q-Learning的基础上出现了很多效果更稳定的强化学习算法,如A3C[23]、DDPG[24]等.但是这些算法相比于 Q-Learning 会带来更大的开销:更新策略过程中的算力开销和推理最佳动作的时间开销,这对于系统级的算法来说是不可接受的.为了权衡调度算法的开销和调度算法的最终效果,ReLeTA 最终选择 Q-Learning 作为框架中的学习算法.Q-Learning 算法是一个基于价值的离轨策略算法,$Q$ 值指的是智能体在特定状态下执行特定动作所能够获得奖励的期望值[25],算法通过建立一个以状态(state)为行、动作(action)为列的Q表来存储和更新 $Q$ 值,$Q$ 值的更新如式(1)所示:

$$Q(S_t,A_t)\leftarrow Q(S_t,A_t)+\alpha[R_{t+1}+\gamma\max Q(S_{t+1},A_{t+1})-Q(S_t,A_t)] \tag{1}$$

其中,$\alpha$,$\gamma$分别表示学习率和折扣系数.$\alpha$用于控制每次 $Q$ 值更新的幅度,决定了算法收敛的速度以及最终算法的收敛效果,学习率较大则收敛较快但可能导致最终的性能不稳定.为了节省时间和保证最终收敛的效果,一般将学习率初始化为较大的值,然后逐渐缩小.$\gamma$是数值在0~1之间的折扣因子,强化学习的目标是为了能获得更多的长期收益,智能体会对未来动作的收益进行考虑.但是未来动作的选择具有不确定性,且离当前时间点越远对当前动作的影响越小,所以在 $Q$ 值的计算中,将未来可能执行动作的 $Q$ 值进行折扣.$\gamma$值为0,则只考虑当前的奖励.$\gamma$值越大,则表示对未来考虑的越多.Q-Learning 中智能体的决策使用了$\varepsilon$-greedy 策略:智能体的决策包括两个部分,其中一个是应用(exploit),另一个是探索(explore).智能体以 $1-\varepsilon(0<\varepsilon<1)$的概率选择目前 $Q$ 值最大的动作,这就是对当前学习到的策略的应用.而探索是指智能体以$\varepsilon$的概率随机选择动作来探索可能存在的更优的策略,这样做能防止智能体学到次优策略.

Q-Learning 算法通过图2所示的步骤实现:首先构建一个 $m\times n$ 的Q表(其中,$m$ 为状态数,$n$ 为动作数),并将表格内的值初始化为0.智能体根据当前的状态来计算所有动作的 $Q$ 值,结合 $Q$ 值和$\varepsilon$-greedy 策略来选择动作,决策后会获得新的状态和一个回报值,然后使用式(1)来更新Q表,在反复的迭代更新Q表后,智能体会学习到较

好的策略.

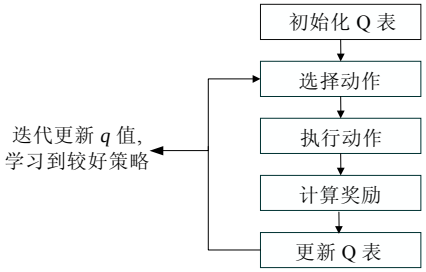



Fig.2 Flow of the Q-Learning algorithm

图 2 Q-Learning 算法的流程

## 3 动机案例

本节将讨论当前流行的两种基于强化学习的温度管理算法,并通过在真实系统上运行来评估这两种方法存在的问题,以此作为 ReLeTA 方法的动机.

### 3.1 LTB

文献[19]与 ReLeTA 的调度方式相似,本节首先对该文献进行讨论.为了方便后文进行讨论,将文献[19]中的算法命名为 LTB.算法 LTB 通过温度传感器读取所有处理器的温度,而将这些温度用作强化学习中的环境状态模型,智能体将程序分配到不同的处理器上运行.假定有 CPU 有 $n$ 个处理器核心,则动作空间与处理器核心数一致都为 $n$.在 LTB 中,奖励函数 $r$ 定义如下:

$$r=T_{em}-T_{\max} \tag{2}$$

其中,$T_{em}$ 为内核的最高阈值温度,$T_{\max}$ 为当前所有内核的最高温度值.式(2)中,奖励函数存在奖励和智能体动作之间关联度较低的缺陷.当系统中运行多个任务时,当前分配任务的内核不一定就是温度最高的核.这样导致奖励函数与动作之间相关性不够高,根据奖励函数优化的温度感知调度算法就很难达到一个最优的效果.

为了展示 LTB 方法存在的缺陷,我们在真实的系统上评估了其性能.实验平台设置如下:处理器为 Inter Core i7-4790 8 处理器,其最大频率为 3.6GHz;实验操作系统为 Ubuntu 18.04 LTS 的 4 核计算机上使用 Parsec benchmark[26]中的 facesim 应用程序来进行实验验证(本文将该任务设为单任务组合,在第 5 节实验配置部分说明).重复执行该程序 1 000 次,并记录系统温度变化数据.为了避免温度波动影响图像的展示,将每 5 个单位数据取平均值绘制成图,因此,1 000 次程序执行总共采样温度 200 次.图中 *step* 指采样的次数,*y* 轴是系统的峰值温度.作为参考方法,本组实验选用了本文所提出的状态模型和奖励函数作为对比.当使用相同的奖励函数和不同的环境状态模型(即本文所提出的环境状态模型)时,如图 3 所示,使用 LTB 环境状态模型的多核系统温度管理方式与本文提出的方式相比,处理器中内核的最高温度提高了平均 1.3°C.这是由于 LTB 所设计的环境状态模型仅仅考虑了所有内核的当前温度,该方式太过简单,无法准确地展现处理器内核的负载变化以及温度的变化趋势,因此对温度的控制效果略有不足.

图 4 展示了使用相同的环境状态模型和不同的奖励函数(即本文所提出的奖励函数)时,处理器内核最高温度的变化情况.使用 LTB 奖励函数下的内核的最高温度增加了平均 2.0°C,这是由于该奖励函数考虑的是某一状态下所有内核的最高温度值 $T_{\max}$ 与阈值温度的差值,而在真实的系统中,不同的动作(任务分配)可能会产生相同的 $T_{\max}$,导致模型不能很好地区分不同动作的效果,使得该奖励函数不能准确地帮助模型进行策略选择.

这里可以看出,LTB 的状态模型没能有效地反映系统温度变化,其奖励函数与任务分配关联度较低,导致了其效果不佳.而本文所提出的环境状态模型同时兼顾了系统负载信息和温度以及温度变化趋势(详细信息见第 4 节),能较为全面地反映当前的系统状态.同时,新提出的奖励函数相比于 LTB 中的奖励函数不仅考虑到了整体的温度,也提高了动作与奖励的相关性,从而产生更优异的温度控制效果.

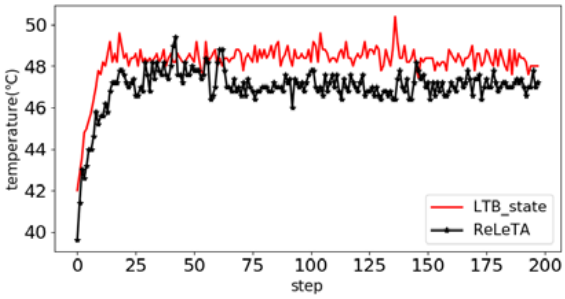



Fig.3 Comparison of the same reward function under different environmental state models

图 3 相同奖励函数,不同环境状态模型下的对比

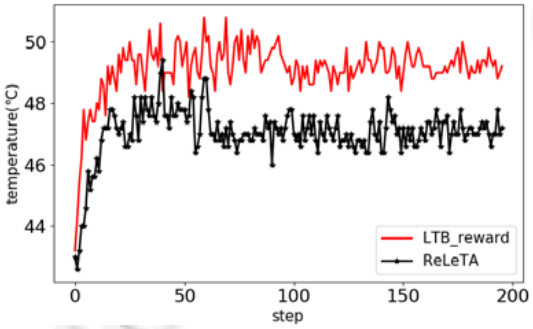



Fig.4 Comparison of the same environment state model and different reward functions

图 4 相同环境状态模型,不同奖励函数下的对比

### 3.2 DSM

本文还研究了当前另一种基于强化学习的温度管理算法[20],后文中简称该方法为 DSM.在 DSM 中,作者使用两个指标:热应力(由于温度变化使得芯片在各种约束下所有的应力反应)和老化程度作为最小化温度的环境状态,并将这两个指标加入到奖励函数的计算结果中.该方法采用了一个较为复杂的动作空间,同时对任务进行调度并相应地主动调节系统频率.在奖励函数中,DSM 将延迟考虑进去,通过加入延迟约束来同时优化温度和保证延迟.然而通过实验发现,此方法并不能在这温度和性能两者之间达到一个很好的平衡.使用之前相同的计算平台,采用 DSM 方法调度程序 facesim 进行实验,选择 *test* 为程序的输入集,因 DSM 需要加入一个约束延迟,通过实验测试 facesim 在本次实验环境下的最低运行时间为 4.0s,最高运行时间为 19.0s.我们选择将 DSM 的延迟设置为 5s,反复调用 1 200 次.

图 5 展示了第 850 次~第 885 次的运行实验结果,包含了程序运行时间和系统温度.

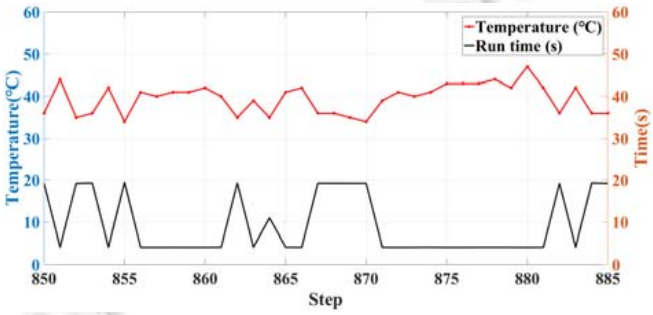



Fig.5 Peak temperature of the system and the running time of facesim under the DSM method

图 5 DSM 方法下系统的峰值温度和 facesim 的运行时间

根据图 5 所示的结果,在该程序执行 850 次后,DSM 方法在训练很多次之后仍然不能够达到一个较为稳定的结果,出现了很多延迟低但温度很高和延迟高却温度很低的情况.这是因为该方法的奖励函数中虽然同时考虑了温度和性能,但是并未对二者进行有效的权衡,造成了系统的温度波动以及程序的程序执行时间变化,并不能够满足制定的延迟约束条件.

从本节对现有方法的讨论可以看出,现有的基于强化学习的温度感知调度算法仍然存在一些问题.为了改善现有方法的缺陷,本文提出了全新的温度感知调度算法 ReLeTA,通过全新的建模方式来实现高效的温度感知多核处理器调度.

## 4 ReLeTA 温度感知调度方法

本节将对提出的算法进行全面的讨论,包括对算法的总体设计以及建模方法都进行了详细描述.

### 4.1 模型概述

图 6 展示了 ReLeTA 的整体框架,目前,ReLeTA 的调度算法是在用户空间(user space)实现,在未来考虑将其实现于操作系统内核中.作为一个多核系统的任务调度器,当一个新的应用程序到达后,ReLeTA 从底层硬件获取状态信息,随后与操作系统进行交互,通过 CPU_affinity 将程序分配到对应的处理核心上.系统完成程序分配之后返回奖励值,并根据奖励值来更新策略.其中,底层硬件是处理器的内核,现代 CPU 在每个计算内核上都设有传感器,可以直接读取处理器当前的运行温度.在工作过程中,操作系统将所获取的处理器频率、利用率以及温度信息传递给 ReLeTA 来优化调度策略.整个 ReLeTA 算法的系统开销较小,通常在 1ms 之内,详细的系统开销实验结果可见第 5 节表 8.在下面的章节中,将详细介绍 ReLeTA 如何使用从底层获取的系统信息.

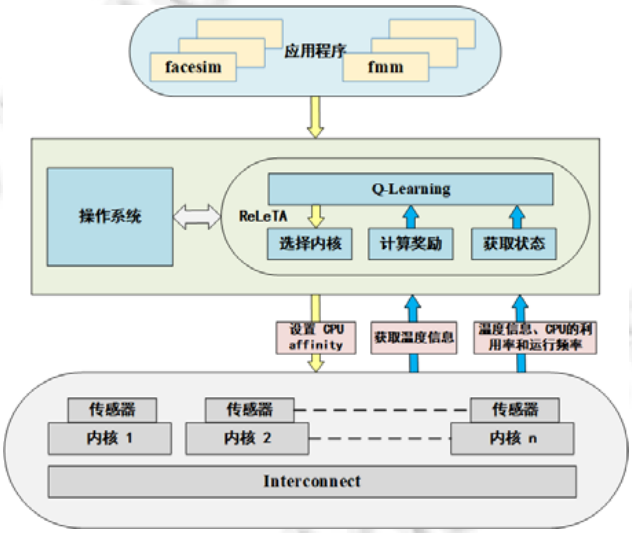



Fig.6 ReLeTA framework model

图 6 ReLeTA 框架模型

### 4.2 温度感知的强化学习建模

对 Q-Learning 来说,最重要的部分就是设置有效的环境状态模型、动作空间及奖励函数,使智能体也就是温度感知调度器可以快速地学习到一个策略来有效降低系统的温度.本节将详细介绍 ReLeTA 如何对这几个方面进行建模.

(1) 环境状态模型

强化学习需要一个精确的环境模型,可以准确地反映出系统变化和奖励以及智能体动作之间的关系,从而帮助智能体快速学习到一个最优的策略.因此,环境状态模型是否能精确地体现当前系统的状态,是任务调度器

至关重要的一部分.一个准确、全面的环境状态模型更有助于找到最优的任务分配策略,也能够促进任务调度器快速地收敛到最优的策略.

温度感知任务调度器的最终目的是要实现系统温度的优化,因此在建模系统状态时,首先对系统温度相关的因素进行分析.目前,大部分基于Linux的操作系统都使用“ondemand”调频模式,在此模式下,系统根据CPU当前的利用率来动态地调节运行频率[27]:系统负载较高,则提高频率来加速任务的运行;负载降低后,系统降低运行频率达到节能的目的.为了观察影响系统温度的不同要素,我们采用了与第4节相同的实验环境,在系统中反复调用parsec测试集中的facesim程序,同时采集任务所在内核的运行频率、利用率和温度,采集的系统数据绘制在图7中.通过观察可以看到:系统的利用率和运行频率均与温度有很高的相关性,当利用率和运行频率大幅度上升时,温度也会大幅度上升;反之,温度也随之下降.因此,我们将每个处理器的运行频率和利用率作为强化学习中状态的一部分.然而,将所有处理器核心的运行频率和利用率加入到系统状态中会导致状态的维度很大,当系统处理核心增加之后,不利于算法的可扩展性,也会导致强化学习的系统开销增加.因此,我们将同一个内核的利用率和运行频率以相加的形式合并到一个变量中.然而在不同的调频模式下,频率和利用率对温度的影响不一致,因此我们对它们进行加权求和,公式如下:

$$L_i^t = F_i^t \times w_f + U_i^t \times w_u \tag{3}$$

其中,$w_f$,$w_u$分别为频率和利用率对应的权重值,且两个参数满足$w_f$+$w_u$=1.通过实验结果发现,在“ondemand”调频模式下的最佳权重为:$w_f$=0.3,$w_u$=0.7.$U_i^t$为$t$时刻核$i$的利用率,计算机中内核利用率是介于0和1之间的值,频率则一般处于200MHz至4GHz之间,两个指标之间数值相差较大.当指标间的水平相差很大时,直接使用原始指标值作为状态,就会突出数值较高的指标对系统状态的影响,同时削弱数值较低指标的作用.因此,为了确保频率和利用率处于同一数值范围内,对频率进行归一化处理[28],式(3)中$F_i^t$表示为$t$时刻核$i$的归一化运行频率.运行频率的归一化处理方式如下:

$$F_i^t = F_{currently\_i} / F_{\max} \tag{4}$$

其中,$F_{currently_i}$为核$i$的当前频率,$F_{\max}$为核的最大运行频率.

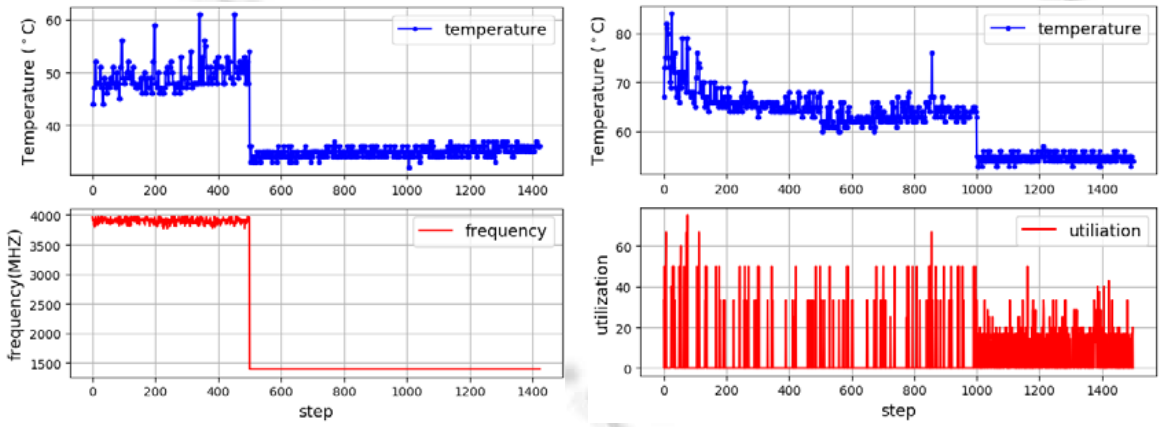



Fig.7 Relationship between operating frequency and core utilization with core temperature

图7 运行频率和内核利用率与内核温度之间的关系

我们在真实系统中对比了合并利用率和频率作为系统状态和单独使用利用率和频率作为状态下的实验效果,发现合并状态后的效果并未差于将两个状态分开表示.

为了全面反映当前系统的状态,除了系统利用率和频率信息,还需要考虑系统的温度信息.现有方法如LTB,直接使用所有内核的当前温度作为系统状态.然而系统温度变化很快,仅用内核的当前温度并不能较好地反映系统的温度变化趋势.在温度管理的文献中,通常同时使用当前温度和温度梯度来表示当前系统的温度变化趋

势[18],然而,直接用一个 $2n$ 维的数据来表示系统的温度状态会带来巨大的空间和时间开销.从当前温度和温度变化以及降维的角度出发,我们采用了文献[29]中所提出的方法来表示当前内核的温度状态:

$$T_i^t=(T_i(t)-T_i(t-\Delta t)\times e^{-b\Delta t})/(1-e^{-b\Delta t}) \tag{5}$$

其中,$T_i(t)$为核 $i$ 在 $t$ 时刻的温度,$T_i(t-\Delta t)$为核 $i$ 在 $t-\Delta t$ 时刻的温度,$b$ 是由处理器决定的常数,$\Delta t$ 表示间隔时间.这些参数在文献[29]中进行了详细的介绍,在此处不做描述.式(5)涵盖了系统的当前温度和温度变化趋势,相比于 LTB 能更好地反映系统温度信息,而相比于文献[18],则实现了有效降维.为了使温度状态与合并的系统状态 $L_i^t$ 处于相同的数值范围,我们使用系统的最大阈值温度 $T_{threshold}$ 对当前温度状态进行归一化:

$$\hat{T}_i^t=T_i^t/T_{threshold} \tag{6}$$

最终,ReLeTA 中单个内核的状态信息由下式表达:

$$s_i^t=(L_i^t,\hat{T}_i^t) \tag{7}$$

而所有内核的当前状态组合起来表示当前的系统状态,系统状态表示如下:

$$S_t=\{s_1^t,s_2^t,...,s_n^t\} \tag{8}$$

(2) 奖励函数

强化学习是通过智能体不断地执行操作和评估环境所反馈的奖励值来改善其学习策略,因此,奖励函数的设置决定了任务调度器学习到最优策略的时间和学习策略的最终效果.现有文献的奖励函数存在一定缺陷:LTB 的奖励函数与动作之间相关性不够高,DSM 的奖励函数复杂度高且很难实现温度和性能的均衡.这两种情况均不利于智能体学习到最优的任务调度策略.因此,我们提出了一种新的奖励函数来高效地指导调度器学习到最优的温度感知调度策略.新的奖励函数公式如下:

$$R(t)=\overline{T}(t-1)-T_{a_i}(t) \tag{9}$$

其中,$\overline{T}(t-1)$为 $t-1$ 时刻所分配的任务运行结束后,所有内核的平均温度;$T_{a_i}(t)$ 为 $t$ 时刻新分配任务运行结束后,任务所在内核的温度.新的奖励函数有两个优点:(1) 相比于 LTB 的奖励函数仅仅使用静态的系统温度最大阈值,新的奖励函数使用一个动态的系统平均温度 $\overline{T}(t-1)$ 作为参考温度,动态的系统平均温度能更准确地反映系统的温度状态;(2) 直接将被分配任务内核的温度加入到奖励函数中,能提高动作与奖励的相关性.而相对 DSM 中奖励函数,尽管同时考虑了性能和温度,但正如图 5 所示,其奖励函数很难实现对性能和温度的有效权衡;而新的奖励函数则相对简单且能有效降低温度.

(3) 动作空间

强化学习中的动作空间指智能体可以执行的所有动作的集合,ReLeTA 是一个温度感知的调度器,其目标就是将程序调度到不同的处理器上,从而达到降低系统温度的目的.因此,我们将 ReLeTA 的动作空间定义如下:

$$A=\{a_1,a_2,a_3,\ldots,a_n\} \tag{10}$$

其中,$a_i$ 表示为把当前任务分配到核 $i$ 上运行,$n$ 是系统所拥有的内核数.

### 4.3 $Q$值近似

ReLeTA 使用 Q-Learning 算法来训练调度策略,传统的 Q-Learning 算法使用 $Q$ 值表格来存储不同状态、动作下所获得的 $Q$ 值,然而随着状态空间和动作空间的增加,会导致存储 $Q$ 值所需的内存空间呈指数形增长.为了缓解这一问题,通常采用神经网络来近似估计不同状态动作组合下的 $Q$ 值,将 Q 表的更新问题变成一个函数拟合问题.在 ReLeTA 中,使用同样的方法来处理存储 Q 表带来的内存开销问题.

为了设计一个简单且准确的神经网络来拟合 $Q$ 值,我们对网络的层数和神经元的个数对模型的影响进行了评估.使用大量不同的神经网络进行了多组测试后,发现“输入层-隐藏层-输出层”这种 3 层结构的神经网络,隐藏层的神经元个数设置为 $2\times n$,能使算法达到较好的效果且开销较低,而增加网络层数或隐藏层神经元个数都不能使得效果有明显提升.因此综合考虑效果和系统开销,我们选择 3 层神经网络来拟合 $Q$ 值:

$$Q(s,a)=\theta_1S_1+\theta_2S_2+\ldots+\theta_nS_n+b \tag{11}$$

其中,$\theta=\{\theta_1,\theta_2,\ldots,\theta_n\}$表示神经网络的参数,$b$ 表示偏置,$S_i$ 是内核 $i$ 的系统状态,如式(8)所示.针对此神经网络,我

们使用梯度下降和以下损失函数来更新其参数$\theta$.

$$L(\theta)=E[(r+\gamma\max Q(s',a',\theta)-Q(s,a,\theta))^2] \tag{12}$$

其中,$r$ 为状态 $s$ 下执行动作 $a$ 的奖励,$r+\gamma\max Q(s',a',\theta)$为目标 $Q$ 值,而 $Q(s,a,\theta)$是真实的 $Q$ 值.Q-Learning 通过目标值和真实值之间的误差期望值建立损失函数.

### 4.4 Q-Learning参数设置

(1) 学习率

Q-Learning 中,学习率表示了 $Q$ 值更新的幅度,决定了算法的收敛速度和最终的收敛效果.学习率较大时,算法收敛较快,但一定程度上会影响性能[30].所以通常初始化一个较大的学习率,随着算法运行次数的增加,逐步缩小学习率,从而使算法收敛速度较快且最终性能较好.在 ReLeTA 中,我们通过实验来经验性地确定最佳学习率.实验平台与第 4 节中的一致.通过反复调度同一个任务,来观察不同的学习率下算法的收敛效率和对温度的控制效果,实验结果如图 8 所示.需注意,采用不同的程序不会影响图 8 所获得的实验结果.可以看出,学习率初始化为 0.8 的情况下,算法收敛速度最快且温度控制较好;其他方法最终也能达到类似的温度降低效果,但所需的时间更长.因此我们采用 0.8 作为算法的初始学习率来更新神经网络权重.

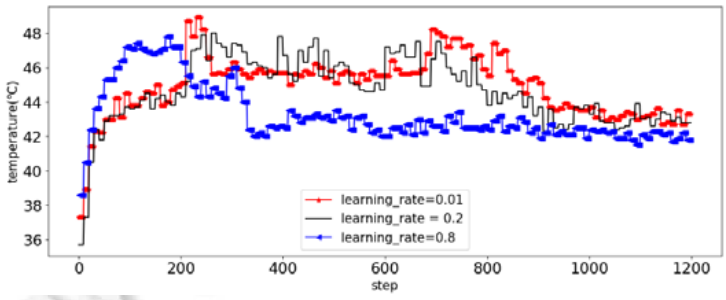



Fig.8 Convergence of temperature under different learning rates

图 8 不同学习率下温度的收敛情况

(2) 贪婪策略($\varepsilon$-greedy)

Q-Learning 在决策学习过程中,如果每次都选择当前状态下 $Q$ 值最大的动作,容易导致所学策略收敛到局部最优的结果.因此,强化学习算法通常使用贪婪策略($\varepsilon$-greedy)来进行一定比例的随机探索,从而避免策略收敛到局部最优.具体来说,贪婪策略就是在每次动作选择时,以概率$\varepsilon$来随机选择一个动作,而不采用当前最大 $Q$ 值的动作.在 ReLeTA 中同样使用了贪婪策略:为了学习到最优的任务调度策略,在任务调度初期将探索的概率$\varepsilon$设置为 0.4;随着任务调度次数的增加,逐步缩小$\varepsilon$,最终保持在 0.03.这样,算法初期可以尝试更多的策略,而后期减少探索偏向于使用优化过的策略.

### 4.5 算法实现

ReLeTA 的伪代码如下.

**算法**. ReLaTA.

Input:新的任务(应用程序).

Result:任务所分配的处理器.

```
1  获取系统状态信息 S_t
2  if 概率<ε then
3     A_t(在动作空间中随机选择动作)
4  else
5     A_t=argmaxQ(S_t,a_t,θ)(选择 Q 值最大的动作)
6  使用 CPU_affinity 将任务分配到处理器 A_t 上
```

7 通过式(9)计算回报值

8 对式(12)进行梯度下降来更新参数$\theta$

算法的输入为新的任务(即应用程序),输出结果为任务所分配的处理器.基本流程如下:

- 当一个新的应用程序进入系统,ReLeTA 首先获取系统状态;
- 随后,使用$\varepsilon$-greedy(第 2 行~第 5 行)策略来确定任务分配结果:根据获取的当前系统状态计算所有可能执行的动作的 $Q$ 值,以$\varepsilon$的概率随机选择动作,以 $1-\varepsilon$的概率选择当前 $Q$ 值最大的动作;
- 然后,智能体根据所选动作来设置任务的 CPU_affinity,将任务绑定到对应的内核;
- 最后更新状态,并更新神经网络中的参数值.

## 5 实 验

为了验证 ReLeTA 的性能,我们通过大量基于真实硬件平台的实验对所提出方法进行了全方位的评估,并与现有的几种方法进行了对比.相比于之前的很多方法采用系统模拟和数值模拟的实验方式,在真实系统上的评估更能反映出各个方法的实际效果.

### 5.1 实验配置

(1) 实验环境

为了验证所提出方法的有效性和可扩展性,本次实验采用了 3 种不同的计算机硬件平台进行实验,表 1 列出了 3 种实验平台的配置.本次实验中使用的操作系统均为 Ubuntu 18.04,内核版本为 4.15.0.

**Table 1** Computer configuration information

**表 1** 计算机配置信息

| 平台 | 平台 1 | 平台 2 | 平台 3 |
|---|---|---|---|
| CPU | Intel Core i5-3230M | Intel Core i7-4790 | Inter(R) Xeon(R)Silver 4210 |
| 内核数量 | 2 | 4 | 10×2 |
| 最大运行频率(GHz) | 3.2 | 3.6 | 2.2 |
| Memory(GB) | 4 | 8 | 126 |

(2) 测试程序

本次实验采用了 Parsec 程序集[26],Parsec 程序集中的程序种类多样,能充分代表计算机中常见的各种不同类型的程序,因此被广泛地用于计算机系统性能测试.为了验证算法在不同程序集下的效果,我们使用表 2 中的不同程序以及不同的程序输入来组成更加多样化的测试集,随机生成 3 任务组合、5 任务组合、8 任务组合、15 任务组合这 4 种.为了保证实验的公平性,对每个任务组我们通过随机生成一个时间间隔(时间间隔介于 0.1s~1.6s 之间)来生成一个程序调度序列,然后对不同的方法使用相同的序列.对于不同的对比实验,我们针对实验需求采用了不同的温度采集方法,在后面的实验会进行具体说明.实验中所用到的测试任务信息和输入信息展示在表 2,与 Parsec 程序集相关的输入信息不在此处进行详细地描述.

**Table 2** Test tasks and inputs

**表 2** 测试任务和输入

| 编号 | 任务名称 | 输入 |
|---|---|---|
| 0 | blackscholes | test,simsmall,simmedium,simlarge |
| 1 | canneal | test,simlarge |
| 2 | debup | Test,simlarge |
| 3 | facesim | test,simlarge |
| 4 | ferret | test,simlarge |
| 5 | fluidanimate | test,simlarge |
| 6 | fremine | test,simlarge |
| 7 | splash2x.fmm | test |
| 8 | splash2x.radiosity | test,simmedium |
| 9 | splash2x.water_spatial | test,simmedium |
| 10 | X264 | simmedium |

### 5.2 实验结果

(1) ReLeTA 与 LTB

现有的基于温度感知的调度算法中,DSM 同时考虑了性能和温度,每个任务都有特定的性能约束,在进行任务分配的同时,还进行系统内核频率调节.而 ReLeTA 和 LTB 都仅通过任务调度来优化系统温度,不主动进行频率调节.两种方法的动作空间及优化目标一致,所以本节首先对两种方法进行全面的实验对比,通过在不同的平台下调度不同的任务集,来对比两种方法对系统温度的管理效果.

任务调度过程中,两种调度方法均使用 Linux 系统中的“ondemand”频率调节方式.

第 1 组实验选取程序 fluidanimate(输入为 simdev)在 2 核的平台上运行,在系统中反复调用 1 400 次,每次任务运行结束采集一次系统峰值温度,结果如图 9 所示.

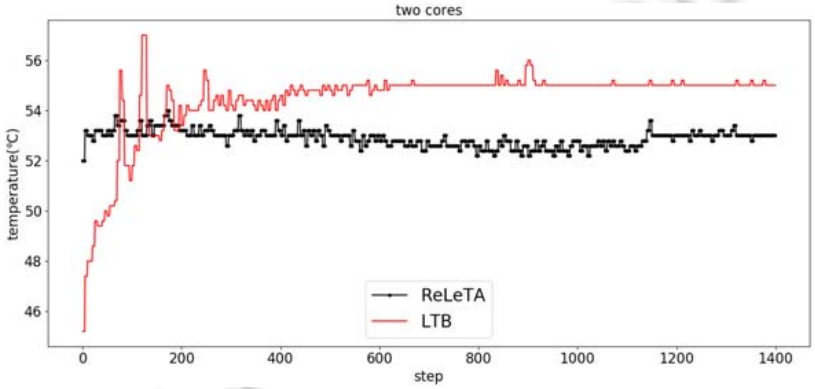



Fig.9 Peak system temperature when LTB and ReLeTA run a single task in the 2-core platform

图 9 LTB 和 ReLeTA 在 2 核平台中运行单任务时系统峰值温度

从图中可以看出,在任务反复调度 200 次后,我们方法的峰值温度明显低于 LTB,在整个任务执行周期的峰值平均温度比 LTB 降低 1.8°C.值得注意的是,两种方法下任务运行的平均时间均为 0.68s.

为了进一步增加硬件的内核数和程序的多样性,我们在 4 核平台上的展开实验,结果如图 10 所示.

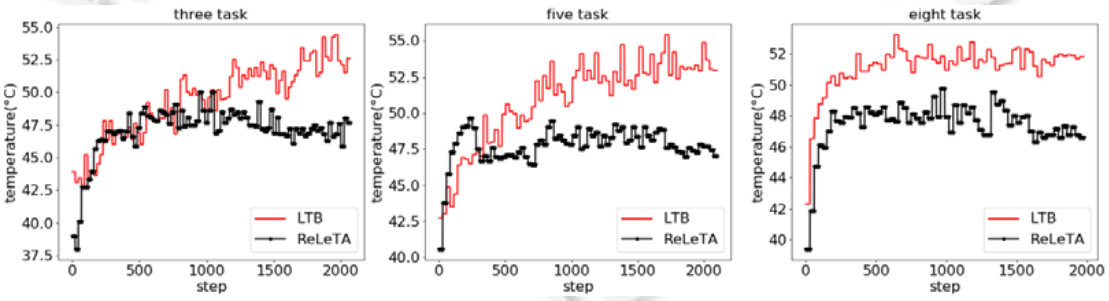



Fig.10 Peak temperature of the system under different task combinations of LTB and ReLeTA

图 10 LTB 与 ReLeTA 在不同任务组合下系统的峰值温度

实验分别使用 3 任务组合、5 任务组合、8 任务组合和 15 任务组这 4 个任务集对两种方法进行了对比实验.每个实验中,将任务集中的程序随机调度 2 000 次,并记录每次程序运行结束时的系统峰值温度.*X* 轴表示程序执行的次数,*Y* 轴表示为峰值温度.

从实验结果来看,ReLeTA 相比于 LTB 能进一步降低系统温度.表 3 总结了实验的温度差异数据,从该数据可以看出,在最好的情况下,ReLeTA 在 8 任务组合下平均温度降低了 4°C,在 5 任务组合下两种方法的最高温度差达到了 13°C.从图 10 和表 3 中可以看出,当执行 3 任务组合和 5 任务组合时,运行初期 LTB 的性能要略微优

于本文方法.这是因为 LTB 的状态简单,当测试程序数量较少时,LTB 可以更快地学习到有效的任务调度策略;但是随着程序运行次数的增加,ReLeTA 的温度管理效果逐渐优于 LTB.同时可以看到:随着测试程序种类的增加, ReLeTA 展现出更优越的温度管理效果,说明 ReLeTA 面对多样的程序时具有更强的适应性.

**Table 3** Temperature difference of the system under different task groups of LTB and ReLeTA

**表 3** LTB 和 ReLeTA 不同任务组下系统的温度差

| 任务组合 | 平均温度差(°C) | 最大温度差(°C) |
|---|---|---|
| 3 任务组 | 2.6 | 10 |
| 5 任务组 | 3.7 | 13 |
| 8 任务组 | 4 | 8 |

为了进一步对比两种方法的可扩展性,我们在 20 核的实验平台上进行了对比实验.实验中使用由 15 个任务构成的任务组合,按顺序将这 15 个任务各执行 1 000 次.为了减少数据采集所导致的系统开销,实验每 4s 采集一次系统的温度信息.图 11 显示了两种方法在整个执行周期的峰值温度变化,可以看出,ReLeTA 的峰值温度绝大多数情况下优于 LTB,峰值温度平均降低了 1.35°C.除了峰值温度,我们还采集了系统的平均温度和任务所运行内核的温度,具体温度变化情况如图 12 所示,具体温度数据总结见表 4.可以看出,ReLeTA 平均峰值温度比 LTB 降低了 1.59°C,任务所调度内核的温度相比于 LTB 则大大降低,平均降低了 5.26°C.

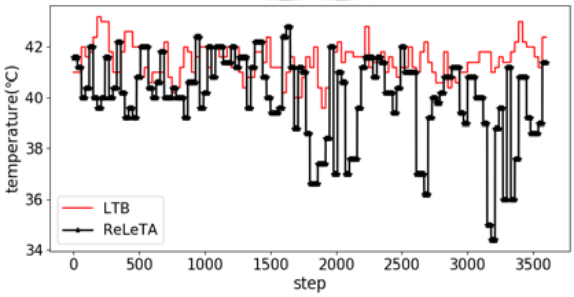



Fig.11 Peak temperature of the system when LTB and ReLeTA run 15 tasks in the 20-core system

图 11 LTB 和 ReLeTA 在 20 核系统中运行 15 任务组合时系统的峰值温度

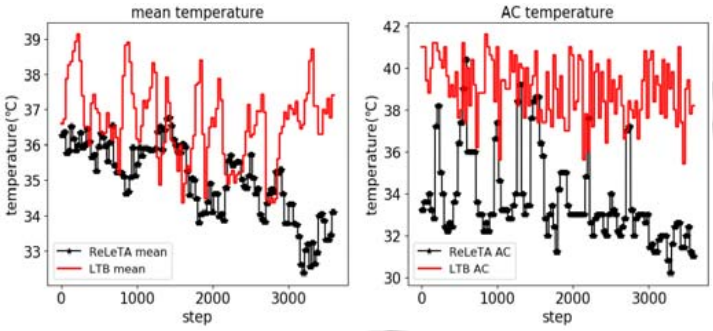



Fig.12 Average temperature of the system and the temperature of the core where the task is running when LTB and ReLeTA run 15 tasks in the 20-core system

图 12 LTB 和 ReLeTA 在 20 核系统下运行 15 任务组合时的系统平均温度和任务运行所在内核的温度

**Table 4** System temperature of LTB and ReLeTA running 15 tasks in the 20-core system

**表 4** LTB 和 ReLeTA 在 20 核系统中运行 15 任务组的系统温度情况

| 方法 | 平均峰值温度(°C) | 平均温度(°C) | 运行任务所在核温度(°C) |
|---|---|---|---|
| LTB | 41.45 | 36.61 | 38.96 |
| ReLeTA | 40.05 | 35.02 | 33.70 |

我们在 20 核的平台使用 Hyper-Threading 技术将逻辑内核数扩展到 40 核开展进一步的测试.因篇幅有限,我们在此处不展示详细的温度变化图,统计的实验结果总结在表 5 中.可以看出,在两种方法下,峰值平均温度相差 0.9°C,平均温度相差了 1.4°C,运行任务所在内核的平均温差较大达到了 3.5°C.在 40 核的情况下,峰值平均温度差相比于 20 核有所下降.这是因为在 40 核下,我们使用了与 20 核实验中相同的任务集,相比于 20 核的运行环境出现了更多的空闲时间,导致系统峰值温度和平均温度降低.

**Table 5** System temperature of LTB and ReLeTA running 15 tasks in the 40-core system

**表 5** LTB 和 ReLeTA 在 40 核系统中运行 15 任务组的系统温度情况

| 方法 | 平均峰值温度(°C) | 平均温度(°C) | 运行任务所在核温度(°C) |
|---|---|---|---|
| LTB | 41.4 | 35.0 | 35.3 |
| ReLeTA | 40.5 | 33.6 | 31.8 |

(2) ReLeTA 与 DSM、LTB 和 Linux 之间的比较

在第 2 个实验中,加入了 DSM 和 Linux 的 CFS 调度方式[31]来进行对比.本次实验采用了 4 核实验平台,选用了 3 个运行时间存在一定差异的程序:canneal、dedup、facesim 来进行实验.因为 DSM 需要指定任务的延迟约束,通过在系统上进行测试,将 3 个程序的约束依次设置为:2ms,43ms,5s.每个任务各执行 400 次,执行过程中,我们采集了所有程序的运行时间以及每次程序运行结束时的系统温度.针对 ReLeTA、LTB 和 Linux,直接使用 Linux 默认的“ondemand”调频模式来控制系统频率.

实验结果展示在图 13 中,可以看出,整个实验过程中,DSM 运行温度较低.这是因为 DSM 使用了主动调频,使用较低的运行频率来控制温度,但同时导致单个程序运行时间更长.该方法下,运行完所有程序所需要的时间大约为 3 000s;而其他方法以较高的频率运行程序,运行完所有程序的时间为 1 500s 左右.运行完所有程序后,系统进入空闲状态从而降低系统温度.我们以 DSM 的执行时间为周期,计算了在这段时间间隔内 4 种方法的平均系统温度,结果汇总在表 6 中.在整个任务执行周期,本文所提算法相对于 DSM 仅高出 0.6°C;而相对于其他两种方法,我们的方法降低的平均峰值温度分别为 5.8°C 和 2.5°C.

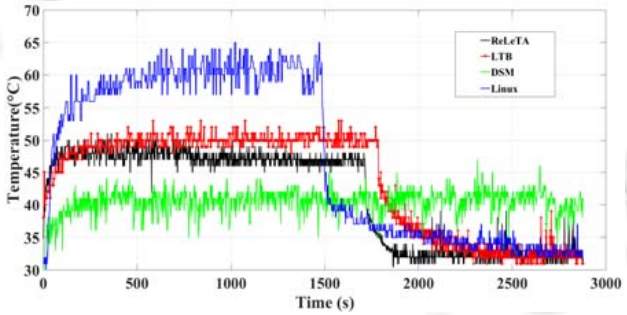



Fig.13 Experimental results of ReLeTA, LTB, DSM, and Linux default scheduling methods

图 13 针对 ReLeTA、LTB、DSM、Linux 默认调度方式的实验结果

**Table 6** Comparison of various methods with DSM

**表 6** 多种方法与 DSM 对比

| 方法 | Linux | ReLeTA | LTB | DSM |
|---|---|---|---|---|
| 平均温度(°C) | 46.4 | 41.2 | 43.7 | 40.6 |
| VS DSM (°C) | 5.8° | 0.6 | 2.5 | – |

DSM 的目标是在满足性能约束的前提下进行系统温度优化,然而正如我们在图 5 所展示,DSM 复杂的奖励函数很难实现温度和性能的平衡.表 7 对所有方法中不满足性能约束的情况进行了总结,DSM 方法下,任务 canneal、dedup、facesim 分别有 41%,5%和 4%的情况下不满足延迟约束,而本文所提方法只有极少量的情况下出现约束不满足的情况.综合温度和性能的考虑,相比于其他两种方法,ReLeTA 能在保证性能的情况下,将系统

温度保持在较低的状态.

**Table 7** Proportion of running three tasks under four methods that do not meet the time constraint (%)

**表 7** 4 种方法下运行 3 种任务不满足时间约束的占比(%)

| 任务 | Linux | ReLeTA | LTB | DSM |
|---|---|---|---|---|
| canneal | 0 | 2 | 0 | 41 |
| dedup | 0 | 3 | 14 | 5 |
| facesim | 0 | 0 | 0 | 4 |

除温度和性能,本文对 3 种方法下的时间开销进行了统计评估,总结见表 8.此处的时间开销是指各个方法从系统读取状态到最终完成任务调度的时间间隔,3 种方法的平均开销都在保证在 1ms 以内,但是 DSM 运行任务过程中的最大开销是其他两种方法的 3 倍左右.因为 DSM 读取了温度循环再进行了一系列的计算得到真正的状态,再计算各个动作下的 $Q$ 值,中间的计算量较大,所以时间开销最大(并且 DSM 动作包括了任务映射和调频).本文方法的开销高于 LTB,主要由于本文状态模型更为复杂.

**Table 8** Time overhead under the three methods

**表 8** 3 种方法下的时间开销

| 方法 | 平均时间开销(ms) | 最大时间开销(ms) |
|---|---|---|
| DSM | 0.776 | 3.76 |
| LTB | 0.296 | 1.04 |
| ReLeTA | 0.540 | 1.3 |

## 6 总 结

多核系统的温度管理已经成为多核系统领域一个重要的研究课题,由于机器学习方法在各个方面的应用均取得巨大突破,受到了关注.其中,强化学习作为灵活性最高的机器学习算法,被广泛运用于各种复杂动态决策问题.目前已经有很多基于强化学习的系统温度管理研究工作,然而当前存在的基于强化学习的系统温度管理方法在状态和奖励函数建模中均存在一些问题,使得算法很难实现性能、温度和复杂度得较好平衡.通过对当前相关工作的总结,本文提出了全新的状态建模和奖励函数建模方法,并且在不同真实的硬件平台上使用不同的任务集进行了全面的实验评估.相比于现有的两种方法,本文所提的 ReLeTA 方法可以实现更好的温度管理,降低系统的峰值温度和平均温度.ReLeTA 在本文中仅仅考虑了任务的调度,为了能够实现一个全方位的温度管理系统,需要进一步考虑主动控制系统的频率.在未来的工作中,我们将进一步将如何有效主动控制系统频率考虑到 ReLeTA 中.

**References**:

[1] Rudi A, Bartolini A, Lodi A, *et al*. Optimum: Thermal-aware task allocation for heterogeneous many-core devices. In: Proc. of the 2014 Int'l Conf. on High Performance Computing & Simulation (HPCS). IEEE, 2014. 82−87.

[2] Saito H, Yoneda T, Nakamura Y. An ILP-based multiple task allocation method for fault tolerance in networks-on-chip. In: Proc. of the 2012 IEEE 6th Int'l Symp. on Embedded Multicore SoCs. IEEE, 2012. 100−106.

[3] Tang H, Feng X. Train running time allocation algorithm based on dynamic programming. In: Proc. of the 32nd Chinese Control Conf. IEEE, 2013. 8157−8160.

[4] Rowlings M, Tyrrell AM, Trefzer MA. Social-insect-inspired adaptive task allocation for many-core systems. In: Proc. of the 2016 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2016. 911−918.

[5] Rathore V, Chaturvedi V, Singh AK, *et al*. Life guard: A reinforcement learning-based task mapping strategy for performance-centric aging management. In: Proc. of the 2019 56th ACM/IEEE Design Automation Conf. (DAC). IEEE, 2019. 1−6.

[6] Mitchell TM. Machine Learning. McGraw-Hill, 2003.

[7] Pagani S, Manoj PDS, Jantsch A, *et al*. Machine learning for power, energy, and thermal management on multicore processors: A survey. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2020,39(1):101−116.

[8] Chen KCJ, Liao YH. Online machine learning-based temperature prediction for thermal-aware NoC system. In: Proc. of the 2019 Int'l SoC Design Conf. (ISOCC). IEEE, 2019. 65−66.

[9] Yang S, Shafik RA, Merrett GV, *et al*. Adaptive energy minimization of embedded heterogeneous systems using regression-based learning. In: Proc. of the 2015 25th Int'l Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS). IEEE, 2015. 103−110.

[10] Donald J, Martonosi M. Techniques for multicore thermal management: Classification and new exploration. ACM SIGARCH Computer Architecture News, 2006,34(2):78−88.

[11] Mnih V, Kavukcuoglu K, Silver D, *et al*. Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[12] Silver D, Hubert T, Schrittwieser J, *et al*. Mastering chess and Shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815, 2017.

[13] Ukhov I, Bao M, Eles P, *et al*. Steady-state dynamic temperature analysis and reliability optimization for embedded multiprocessor systems. In: Proc. of the 49th Annual Design Automation Conf. 2012. 197−204.

[14] Chung EY, Benini L, De Micheli G. Dynamic power management using adaptive learning tree. In: Proc. of the 1999 IEEE/ACM Int'l Conf. on Computer-Aided Design. IEEE, 1999. 274−279.

[15] Jung H, Pedram M. Supervised learning based power management for multicore processors. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2010,29(9):1395−1408.

[16] Lee W, Patel K, Pedram M. GOP-level dynamic thermal management in MPEG-2 decoding. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 2008,16(6):662−672.

[17] Jayaseelan R, Mitra T. Dynamic thermal management via architectural adaptation. In: Proc. of the 2009 46th ACM/IEEE Design Automation Conf. IEEE, 2009. 484−489.

[18] Iranfar A, Shahsavani SN, Kamal M, *et al*. A heuristic machine learning-based algorithm for power and thermal management of heterogeneous MPSoCs. In: Proc. of the 2015 IEEE/ACM Int'l Symp. on Low Power Electronics and Design (ISLPED). IEEE, 2015. 291−296.

[19] Lu S, Tessier R, Burleson W. Reinforcement learning for thermal-aware many-core task allocation. In: Proc. of the 25th Edition on Great Lakes Symp. on VLSI. 2015. 379−384.

[20] Das A, Shafik RA, Merrett GV, *et al*. Reinforcement learning-based inter-and intra-application thermal optimization for lifetime improvement of multicore systems. In: Proc. of the 51st Annual Design Automation Conf. 2014. 1−6.

[21] Benini L, Bogliolo A, De Micheli G. A survey of design techniques for system-level dynamic power management. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 2000,8(3):299−316.

[22] Durand S, Lesecq S. Nonlinear and asymmetric thermal-aware DVFS control. In: Proc. of the 2013 European Control Conf. (ECC). IEEE, 2013. 3240−3245.

[23] Mnih V, Badia AP, Mirza M, *et al*. Asynchronous methods for deep reinforcement learning. In: Proc. of the Int'l Conf. on Machine Learning. 2016. 1928−1937.

[24] Lillicrap TP, Hunt JJ, Pritzel A, *et al*. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[25] Watkins CJCH , Dayan P. Q-learning. Machine Learning, 1992,8(3-4):279−292.

[26] Bienia C, Kumar S, Singh JP, *et al*. The PARSEC benchmark suite: Characterization and architectural implications. In: Proc. of the 17th Int'l Conf. on Parallel Architectures and Compilation Techniques. 2008. 72−81.

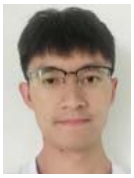
[27] Pallipadi V, Starikovskiy A. The ondemand governor: Past, present and future. In: Proc. of the Linux Symp., Vol.2. 2006. 223−238.

[28] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.

[29] Yeo I, Liu CC, Kim EJ. Predictive dynamic thermal management for multicore systems. In: Proc. of the 45th Annual Design Automation Conf. 2008. 734−739.

[30] Sutton RS, Barto AG. Reinforcement Learning: An Introduction. MIT Press, 2018.

[31] Ishkov N. A complete guide to Linux process scheduling [MS. Thesis]. Tampere: University of Tampere, 2015.



**杨世贵**(1995－),男,硕士生,CCF 学生会员,主要研究领域为基于强化学习的多核任务调度.

**王媛媛**(1998－),女,硕士生,主要研究领域为网络空间安全恶意文档检测.

**刘韦辰**(1982－),男,博士,助理教授,博士生导师,CCF 专业会员,主要研究领域为 Network-on-chip, Hardware-software codesign.

**姜徐**(1987－),男,博士,副教授,博士生导师,CCF 专业会员,主要研究领域为嵌入式实时系统.

**赵明雄**(1988－),男,博士,副教授,CCF 专业会员,主要研究领域为移动边缘计算,协作通信.

**方卉**(1998－),女,硕士生,主要研究领域为神经架构搜索.

**杨宇**(1997－),男,硕士生,主要研究领域为基于强化学习的可调节网络.

**刘迪**(1984－),男,博士,讲师,CCF 专业会员,主要研究领域为智能边缘系统.