摘 要

随着众核处理器的发展，功耗密度还在不断上升，并且使负载不均衡问题更加严重，形成了严重的芯片高温和局部高温点问题。高温热问题已经成为阻碍处理器发展的重要因素。为应对芯片热问题，本文做了深入研究。研究主要集中在动态温度管理方法。针对最近的动态温度管理方法做了详细研究分析。对动态温度管理常用的操作技术，动态电压频率调整技术和任务迁移技术，做了深入的分析。对动态温度管理操作需要的引导控制技术也进行了分析，尤其是模型预测控制方法。 HotSpot热模型是一种简洁准确的热建模方法，本文提出的方法中用其来对芯片做热模型，并结合模型预测控制方法进行温度计算。引入模型预测控制方法之后就可以将任务迁移策略的决定化作任务分配问题，用二部图匹配方法来解决该问题。

因为二部图匹配算法复杂度较高，随着核数的增长算法计算时间显著增加，所以在众核处理器高性能运行时进行热管理是一个巨大的难题。本文提出分层的动态温度管理的解决方法来克服这个问题。新的动态温度管理方法依然采用传统的动态电压频率调整技术和任务迁移，采用模型预测控制方法做理论依据来实现平滑的任务控制，减少牺牲计算性能。为了扩展到众核系统，分层控制方法设计为两层。在低层，核在空间上被分割成块，块内用现有的功耗分布和用模型预测控制方法优化得到的功耗分布进行匹配。在高层，对低层没有匹配上的功耗进行全局任务迁移。如果高层的功耗数量很大，本文采用改进的迭代最小割算法来辅助实现任务迁移的策略。任务迁移操作完成之后动态电压频率调整技术用来调节最终都没有匹配上的功耗。最后仿真实现该算法，并将其与其他算法进行比较，着重比较算法运行时间，处理器性能和芯片热平衡性。结果显示，新的方法比现有的其他方法有很大优势，而且在众核处理器中只有很少的性能损耗。

关键词：众核，动态热管理，模型预测控制，任务迁移，动态电压频率调整

ABSTRACT

With the development of many-core processors, power density is still rising, and the load imbalance problem is more serious plus, become a serious chip high-temperature and local hot spots issues. High-temperature heat has become an important factors that hinder the development of the processor. In response to chip thermal problem, we made a deep research. Research focuses on the dynamic thermal management(DTM). For recent DTM, there is a detailed analysis. Dynamic thermal management commonly used technique are dynamic voltage frequency scaling (DVFS) technique and task migration technique, there is a thorough analysis in those technique. Guidance control technology needed by dynamic thermal management is also analyzed, especially the model predictive control (MPC). HotSpot is a compact and accurate thermal modeling methods, the proposed method with which to make a thermal model of the chip, combined with MPC to do temperature calculation. After the introduction of MPC, policy decision for task migration is turned into task allocation issue, the bipartite graph matching methods can be used to resolve the problem.

Because of high complexity of bipartite graph matching algorithm, with the number of cores increase, the computation time become large, so runtime DTM is a huge problem in high-performance many-core processor. In this paper , a hierarchical DTM is proposed to overcome this problem. The new DTM use traditional DVFS technique and task migration, combine with MPC to do a theoretical basis achieving a smooth control and reducing the cost of computing performance. To extend to many-core systems, hierarchical control method designed to two level. In the low-level cores spatially divided into blocks, there is a match between the existing power distribution within the block and optimized power distribution form MPC. It the top-level, unmatch powers in low-level are collected to do task migration globally. If the number of high-level powers is large, we use improved minimal cut iterative algorithm to assist migration strategy. Finally, DVFS is used to adjust the final unmatch power. The new method is implemented, and compared with other methods. The results show that the new method with little performance loss has a great advantage over other existing methods.

Keywords: many-core, DTM, MPC, task migration, DVFS

目 录

第一章 绪论

1.1 研究工作的背景与意义

根据摩尔定律芯片上的晶体管数量每18个月翻一番。随着这些数量空前庞大的晶体管集成在一个芯片上，当前的多核技术很快就会发展成上百核上千核的时代[1]。已经有几十核的芯片投入生产，包括Tilera的64核处理器和intel的至强融核处理器。2012年Intel就发布了名为 Knights Corner 的至强融核协处理器，中国天河二号超级计算机就装了48000个Knights Corner芯片。 Knights Corner芯片上最多可容 61个内核。 Intel下一代众核处理器 Knights Landing 可容下更多的核心。

多核和众核技术带来了极大的性能提升，但是我们不得不面对随之而来的功耗和热问题。 CMOS 技术的不断发展，功耗密度不断增长，就出现了高温热问题。不仅高的操作温度对微处理器的可靠性有负面影响，还有新的问题。历史上，对芯片和封装设计做热建模，热工程师只需要使用芯片的总功耗，和一个温度值就能构建这个芯片的模型。虽然这种方法在低功耗集成电路中仍然使用，但是对于高性能或者电源受限的设计中这是完全不适用的。芯片上不断增长的非均匀功耗导致硅芯片上出现局部热点和高的温度梯度。比如，在英特尔 90 nm 安腾处理器上，即使经过严格的热管理，局部温度仍然可以高达 80°C，同时芯片是其他部分温度相对较低（61°C）[2]。在高性能多核和众核微处理器中，这种情况更加严重。过高的高温点必然导致可靠性问题，性能下降。局部高温点问题是技术发展带来的一个副作用，这给热工程师，电路设计工程师，计算机设计师带来新的挑战。本来高功耗密度的冷却成本就已经很高，考虑到处理局部高温点，散热成本会更加的高，传统的散热方式比如风扇散热冷却方法已经无法满足需求。所以找到经济和有效地方法去解决热问题同时提高多核和众核芯片的性能和可靠性仍然是一个挑战 [3]。

解决热问题可以从多方面来考虑，比如改进散热系统，改进芯片功能单元布局规划或者采用动态温度管理（DTM）技术。传统空气散热方法因为其简单和低成本仍然被广泛应用。热问题更加严重，研究者也开始寻找更有效的冷却方法，已经有研究者开始研究液体冷却技术，因为水的比热较大。

针对处理器芯片热问题尤其是高性能芯片，在微体系结构级进行动态温度管理还是有巨大优势。动态温度管理利用能使芯片自主修改任务的执行和功耗特性的技术，以使低开销的冷却方法也能保证芯片在安全温度以内。动态温度管理控制器在系统运行时监控系统信息，并采取相应的热管理措施。以最小的性能损耗，

尽可能地把系统温度保持在安全阈值以下，尽量平滑地修正热分布，有效提升系统的可靠性。本文也主要针对动态温度管理进行讨论。

1.2 动态热管理技术的发展

动态温度管理方法是一个提升芯片热相关性能的有效技术 [4]，是随微处理器的发展而产生的技术。管理问题的目的是寻找最优的资源管理策略，来有效控制芯片峰值温度，高温点数量和芯片热梯度。现有的热管理研究可以主要分成两类：

1.离线方法： 离线技术通常针对具有可预测负载的特定嵌入式系统，在设计阶段和编译阶段就解决温度感知的资源管理问题。

2.在线方法： 在线技术针对设计阶段负载信息未知的更通用的平台，依靠有效的学习与控制技术，自适应管理硬件和软件来控制温度。

很多不同的动态热管理操作都被研究过。这些操作包括时钟门控，动态电压频率调整，计算迁移等等。或者是混合方法，就是上面两种或两种以上技术的结合。虽然不同的技术使用不同的机制，并且是用在不同的计算环境中的。但是他们有一个相同的关键思想，就是修改计算系统功耗特性，使热产生量更少和温度分布更平滑。

两种最常用的动态温度管理操作就是任务迁移和动态电压频率调整：

1.任务迁移： 任务迁移方法通过交换多核或者众核处理器上任务来降低芯片的最高温度 [5–10]，而且也可以降低多核系统的能量损耗 [11]。所有核都在最大速度运行，所以任务迁移方法能使处理器得到更高的性能。但是如果没有其他动态温度管理方法的话，这个可能还是会有局部温度太高的问题。

2.动态电压频率调整（DVFS）：动态电压频率调整（DVFS）[12–14] 控制电压和操作频率来调整芯片的温度。最近，DVFS 也应用于暗硅领域 [15, 16]，它随温度限制来改变电压和频率水平。 DVFS 可以保证芯片的温度安全，但是因为频率的降低，芯片的计算性能就要下降。

为了使DVFS和任务迁移更有效，控制方案通常采用基于经验的动态温度管理方法。用控制器分析热模型，热传感信息等等，然后为动态温度管理操作做引导行为。例如，DVFS应该被调整到多少频率，任务迁移到哪一个核。最近的研究中提出了有很多混合方法。许多动态温度管理方法是基于传统控制方法的 [17]，但是这些方法并不太适合多核和众核热系统，因为系统的复杂性 [18]。 [19]中提出了一种基于经验的结合 DVFS 和任务迁移的方法，但是并没有包含支持集成方法的

理论。 [20, 21] 中提出的方法是都是温度预测结合任务迁移，通过温度预测模型，进行任务迁移策略。当预测出一个核上的任务将要超出温度阈值的时候，就将该任务移出该核，移到预测温度最低的核。这种方法中虽然温度预测比较准确，能够降低芯片的平均温度，减少局部高温点，但是这种方法还是不能保证芯片温度在安全温度以下，而且对于高性能众核处理器来说效果将会变差。研究方向也多样，有针对性能的优化的混合方法 [22]，有针对能耗的优化的混合方法 [23, 24]。

最近，模型预测方法（MPC）被引入动态温度管理 [18, 25, 26]。MPC利用芯片的热模型输出功率上的管理建议。因为这个方法在热行为上进行预测来得到更加有效的控制，所以 MPC 可以提供更有效和更精确的管理建议。对比于传统的方法，应用 MPC 有明显的性能提高 [18]。很多研究方法结合 MPC 和 DVFS [18, 25, 26]，只用 DVFS 不能将处理器性能发挥到最大。

DVFS和任务迁移结合MPC或许可以得到这三种方法的优点。MPC有高质量的预测控制，任务迁移提升更高的性能，DVFS保证温度的安全。然而， MPC 结合任务迁移要比 MPC 结合 DVFS 更难。 [22] 提出了一种类似于 MPC 的优化性能的混合方法，但是这种方法为了近似解决非线性优化问题，将核与核之间的热导忽略。

1.3 本论文的主要工作

在这篇文章中，针对高性能众核微处理器提出了新的动态温度管理方法。主要包括两部分内容，第一部分是提出的基于模型预测控制结合任务迁移和 DVFS 的动态温度管理方法。第二部分是针对第一部分中任务迁移决策的扩展性问题，提出了分层的动态温度管理方法。新方法的第一部分利用模型预测控制以设定的目标温度来计算期望的输入功耗分布，然后引导任务迁移和 DVFS 来调整当前的功耗分布去匹配期望的功耗分布。任务迁移由二部图匹配做引导，DVFS只对有限数量的核进行电压频率调整，以保证CPU在热限制下的最大速度运行。这个方法可以最大化处理器性能，最小化核之间温度差，而且核温度可以追踪目标温度。新方法第二部分为了解决众核系统的执行集成 MPC 和任务迁移的可扩展性问题，将任务迁移部分分成两层，在第一层，相邻的核被分成一块，核功率的二部图匹配在块内执行来进行块内任务迁移。没有匹配的核收集起来做第二层的任务迁移计算。在第二层的迁移结果计算中引入改进的迭代最小割算法来提速升计算速度。新的分层方法对众核处理器来说，只需要很少的开销，而且高度可扩展既能保证高的处理器性能，又能保证温度不超过限制。本论文方法的创新性贡献包括：

1.在动态温度管理中，首次将模型预测控制方法与任务迁移和 DVFS 相结合，将任务迁移决策化为任务分配问题处理。

2.针对任务迁移在众核处理器中的扩展性问题，创新性地将任务分配问题分层处理，并引入最小割算法辅助计算。

3.实验证明，新提出的方法比 [25] 具有更大的性能优势，比 [22] 能进行更有效可靠的热控制，而且在开销和扩展性上具有极大的优势。

1.4 本论文的结构安排

本论文的章节结构如下：

第一章绪论，首先介绍当前众核处理器发展，接着说明众核处理器面临的热问题。然后展示了动态温度管理技术的发展和研究现状，最后说明本文主要工作，本论文的主要贡献与创新。

第二章对动态温度管理相关工作进行分析，首先分析对处理器芯片温度不受管理的负面影响。其次，详细介绍动态温度管理操作技术：动态电压频率调整技术和任务迁移技术。最后，介绍对动态温度管理操作进行引导的预测控制方法，包括本文中用到的模型预测控制（MPC）方法。

第三章介绍芯片热建模方法，先说明热传导理论，再主要介绍本文用到的 HotSpot 简洁热建模方法。

第四章详细介绍本文新方法的第一部分即未分层方法，该方法基于模型预测控制结合任务迁移和 DVFS 的动态温度管理方法。主要对微处理器热模型进行说明，用模型预测控制方法计算期望功耗。最后说明基于期望功耗怎样进行任务迁移。

第五章详细介绍本文新动态温度管理方法中的第二部分，即分层方法。主要介绍将任务迁移决策分层计算，并且说明最后的 DVFS 调整。

第六章介绍本论文方法的实现，与结果比较。主要在瞬态温度，算法执行时间，和性能方面与其他方法进行比较，说明本论文提出的方法在众核芯片动态热管理中的优势。

第七章总结，对本文涉及到的内容和研究的方法进行总结。

第二章 动态温度管理相关工作分析

在这一章我们将介绍动态热管理相关的知识。首先 2.1 节将介绍芯片温度不受管理、处于高温状态或有高温点存在时，对芯片的可靠性、性能、功耗和散热成本不利影响。 2.2 节将介绍具体的动态温度管理技术，包括动态电压频率调整、任务迁移等，详细分析其调整机制和降低平衡芯片温度的原理。 2.3 节介绍预测控制方法，仅仅有动态温度管理技术是不够的，这些技术需要有完整的引导控制才能发挥最大的作用，不进行合理的引导，不仅损害处理器性能，而且可能使处理器温度处于更加危险的地步。

2.1 高温对芯片的影响

因为我们持续减小芯片大小和要求高功耗下的性能，增长的芯片复杂性和功耗密度提高了芯片的峰值温度，也使温度梯度更加不均衡。上升的峰值温度缩短芯片寿命，降低芯片性能，影响可靠性也增加散热成本 [27]。上升的温度和静态功耗之间有正反馈的关系，有可能造成热失控。在多核或者众核系统中，不同的应用负载或许引起核之间功耗和温度的不平衡。温度在时间和空间上的变化产生的芯片局部温度最大值叫做高温点 [4]。过多的空间上的温度差也就是热梯度增加时钟抖动降低性能和可靠性。上升的温度需要更多的散热能力去冷却处理器，一个典型的散热风扇会消耗服务器高达 51% 的功耗 [28, 29]。

2.1.1 温度对系统可靠性的影响

高功耗的一个最明显的结果就是上升的芯片温度，高温对芯片最严重的的后果就是损害芯片的可靠性。在高温状态下，载流子迁移率降低，会导致当代 CMOS 技术下的器件变慢。而且还可能导致器件失效，下面是温度相关的半导体器件失效机理 [30]：

电迁移(EM)：由于传导电子和扩散金属原子之间的动量交换，金属线中离子会逐步移动，这样会导致金属线形变。它甚至会引起金属线断开导致器件失效。

应力迁移（SM）：金属原子在机械应力梯度下会发生移动导致金属线变形。不同材料的热膨胀率不同会产生应力。这个也会引起金属线断开导致器件失效。

介质击穿（DB)：当介质中形成一个导电通路，电路的阴阳极短路时，介质失效。

这些机理引起的失效时间（TF）可以被表示为下式：

$$TF = A \cdot e^{E_a/(k \cdot T)} \tag{2-1}$$

其中A是一个常数，E_a活化能量（eV），k是玻尔兹曼常数（8.62×10^{-5} eV/K）。这些都是正常数，T是器件的操作温度（K）。所以，TF是温度的递减函数。当温度上升时，TF会指数下降。这就意味着器件将极快速失效。

互连电阻率也随温度升高而升高，会引起更长的互连 RC 延迟，导致性能损失和时序噪声分析复杂化。高温环境还严重缩短互连和器件的寿命。

温度高不是系统可靠性问题的唯一原因。另外还有两个热现象即热循环和热梯度也会严重影响器件的可靠性。热循环是操作温度的暂时波动，可能是由器件的正常功率上升或者下降引起。另外也可能因为任务负载的变化或者设备的功耗管理策略，比如设备的部件在几种功耗模式下很频繁的切换。热循环可以会削弱材料，引起介质裂开或者焊料疲劳等的不同失效。热循环的失效率可以表示为：

$$\lambda \propto \Delta T^q \tag{2-2}$$

其中 ΔT 是热循环的幅度，q 是热循环的频率。温度波动越大或者越频繁，器件失效率就越大。

热梯度是整个芯片温度在空间上的不平衡，热梯度和局部高温点严重影响封装可靠性。因为现在片上系统（SoC）和多核众核芯片上负载不均衡，这样温度就会不平衡，引起大的温度梯度。大的温度梯度最重要的影响是互连电阻不均衡，这会导致时钟偏差。这个是同步数字电路中很不希望发生的，会引起时序冲突。

2.1.2 温度对静态功耗的影响

温度不仅是功耗的结果，在某种程度上功耗和温度互为因果。这是因为，静态功耗很大程度上取决于操作温度。当前静态功耗已经是系统总功耗的很明显的一部分，而且会随技术继续增长。温度和静态功耗之间的关系已经被广泛的研究过 [31]，漏电流可以表示为：

$$I_{leak} = I_s \cdot A \cdot T^2 e^{\frac{\alpha + \beta V_{dd}}{T}} + B \tag{2-3}$$

其中，对于指定的技术，A、B、α、β 是温度无关的正常数。V_{dd} 是供电电压，I_s 是在指定温度和供电电压下的基准漏电流。我们可以看出温度对漏电流的影响是 $T^2 e^{1/T}$ 。

2.1.3 温度对冷却系统的影响

上升的温度另一个明显的坏处就是冷却成本变得更高。为了保证芯片正常运行，芯片不得不配备成本更高的封装来散热。对于常规散热风扇型的冷却方法，散热风扇不得不在一个更高的速度上运行来散去芯片产生的额外的热量。这是因为散热器的散热能力是由热阻R_{h2a}（h2a表示从散热器到外部环境）决定，热阻R_{h2a}又是由散热风扇速度决定的，如下式：

$$R_{h2a} = (h_1 v_f (1 - e^{\frac{h_2 v_f^{h_3} + h_4}{h_1 v_f}}))^{-1} \tag{2-4}$$

在式(2-4)中，h_1 、 h_2 、 h_3 、 h_4是芯片的特定物理系数，v_f 是散热风扇速度。风扇功耗 P_{fan}正比于风扇转速的立方，即 $P_{fan} \propto v_f^3$。这意味着高的温度不仅是风扇功耗上升，同时也降低了风扇的使用寿命。

2.2 动态温度管理相关技术

前面已经说明，温度不受管理将会带来严重的可靠性，性能，功耗和成本的问题。为应对热问题，这方面的研究已经很多，静态热管理技术也很早被研究过。比如，热感知的版图布局规划。在设计阶段将芯片的最热的功能单元之间留出空间，以便该单元的热量能更多地散热到周围。更进一步的布局规划是通过独特的布局规划将这些功能单元分解成更小的组成部分。但是这种方法并不适用于现在更加广泛复杂的应用。动态温度管理（DTM）技术就是用来解决上述问题，来控制芯片温度和功耗。随着温度被调整，系统的可靠性就可以提升。适当的降低电子设备的温度 10°C ～15°C 可以使设备寿命延长两倍。对于金属结构，热循环幅度减少 10°C 可以使平均失效时间增长16倍。温度降低时静态功耗也会显著减少[32]。温度每降低 9°C ，静态功耗降低 50% [33] 。这对于将来的片上系统设计非常重要。因为静态功耗估计将会超过总功耗的 50% 。调控温度不仅能保证芯片可靠性和降低静态功耗，而且还能提升性能。在低温时，晶体管的开关速度更快 [34] 。平衡的空间热梯度可以显著减轻时钟偏移问题。

动态温度管理需要能使芯片自主修改任务的执行和功耗特性的技术，以使低开销的冷却方法也能保证芯片在安全温度以内。动态温度管理控制器在系统运行时监控系统信息，并采取相应的热管理措施。以最小的性能损耗，尽可能地把系统温度保持在安全阈值以下，尽量平滑地修正热分布。

对一个计算系统执行动态热管理，需要的最重要的系统信息是芯片温度。这个信息可以从片上温度传感器的到，或者用热模型估计。一些最先进的动态温度管理技术还需要温度信息，应用程序特性，任务功耗等等。

2.2.1 动态温度管理和动态功耗管理的不同

尽管温度基本上是由功耗引起的，动态温度管理也需要修正功耗特性，甚至动态功耗管理和动态温度管理都用相同的措施，像动态电压频率调整（DVFS）和任务迁移，但是在动态功耗管理和动态温度管理上有几个明显的不同点。第一，系统温度分布不仅仅只跟功耗分布相关。因为功耗可以瞬间改变，但是温度是功耗的积累，在时间和空间上改变都很慢。打个比方，功耗就像 RC 电路中的电流源，温度就像各个节点的电压。所以温度的行为就像一个低通滤波器，滤掉了功耗变化中的高频部分。第二，温度正比于功耗密度， $T \propto P/A$，这里 A 是面积。所以即使不能降低功耗，我们可以分配功耗到更大的面积上，这样仍然可以降低温度。比如所有进程都合并在一个核上，芯片上这一小部分的负载很重，最好从散热的角度将他们分配到多个核上。第三，比起温度管理策略，功耗管理策略可能有冲突的目标，有产生不期望的温度分布的可能。比如，功耗管理策略可能为了节省功耗非常频繁的将器件切换到低功耗状态。 2.1.1节中提到过这个调整会产生大幅度和频繁的热循环，加速封装疲劳。为了实现低功耗，功耗管理策略可能关掉一些部件整合计算，这会产生局部高温点和大的温度梯度。

2.2.2 动态电压频率调整技术

过去动态电压频率调整技术（DVFS）被广泛应用于功耗和能量优化中，随着芯片热问题的凸显，DVFS 也被广泛应用在处理器热管理中。 DVFS 成为一种被广泛应用的动态温度管理技术。DVFS 技术是调整处理器的时钟频率和电源电压，当处理器时钟频率降低时，电源电压也可以相应地降低。这样可以降低功耗，甚至于节省能量 [35]。 DVFS 技术在计算系统中应用广泛，从嵌入式，到平板，桌面系统，以至于到高性能服务器系统。

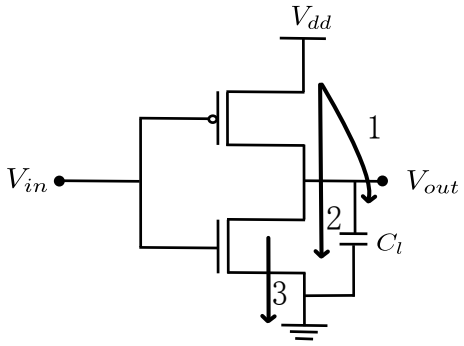


图 2-1 简单CMOS反相器功耗示意图

现在大多数数字电路都是 CMOS 电路，尤其是在处理器方面。所以我们简要分析一下 CMOS 电路的功耗，找出功率，电压，频率之间的关系。 CMOS 电路的功耗是动态功耗，静态功耗和电路短路功耗的综合，这些功耗如图 2-1所示：

1. P_d 表示动态功耗，源于电容的充电和放电（1）。

2. P_s 表示静态功耗，源于晶体管反向偏置（2）。

3. P_{sh} 表示电路短路电流，源于开关时 V_{dd} 到 GND 的直接通路（3）。

CMOS电路总功耗表示为：

$$P_{cmos} = P_d + P_s + P_{sh} \tag{2-5}$$

动态功耗是 CMOS 电路功耗的很大一部分，2.1.2中已经提到近年来静态功耗也占据总功耗的很大一部分，其可以表示如下：

$$\begin{aligned} P_d &= C f V_{dd}^2 \\ P_s &= I_l V_{dd} \end{aligned} \tag{2-6}$$

其中， C 是晶体管栅极电容（取决于它的特征尺寸）， f 是操作频率，V_{dd} 是电源电压， I_l 为漏电流。

动态电压频率调整技术降低电压可以有效地减少功耗。动态功耗可以显著减少，因为上面显示 $P_d \propto V^2$ 。静态功耗也可以减少。降低电源电压会限制操作频率 [36]，其关系可表示为：

$$f \propto \frac{(V_{dd} - V_t)^2}{V_{dd}} \tag{2-7}$$

其中 V_t 表示 CMOS 阈值电压。这也就是说，频率的降低可以伴随着电压也做降低调整。时钟频率 f 降低一半时，能降低处理器功耗，任务完成时间会延长，但是总的能量并没有少。程序运行时间增加会显著影响处理器性能，这也是 DVFS 的缺点之一。当电源电压也降低一半时，功耗将会继续降低，但是任务完成时间并没有继续延长，这样能量也被减少了。如图 2-2所示。

为了在多频处理器上执行 DVFS 操作，操作系统需要先预测或者估计将来的任务负载（比如通过 MPC 方法，后面会介绍），然后将其转换为频率值 f_{des}。这个值被用于调整处理器的时钟频率 f 和电源电压 V_{dd}。 [25] 中的方法就是用 MPC 方法做预测，用 DVFS 做调整，保证芯片在安全温度以下。因为DVFS通过降低处理器的操作频率来降低功耗会增加程序运行时间，这会显著影响处理器性能。所以很多方法中 DVFS 并不是单独使用，往往结合任务迁移来降低对处理器性能的影响（下面将详细介绍任务迁移）。

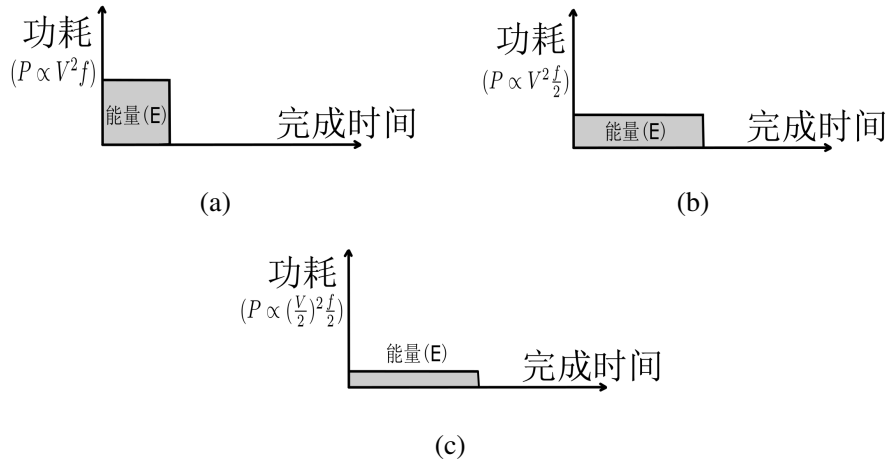


图 2-2 任务的功耗与能耗

(a)最大电压频率运行时的功耗与能耗；(b)频率减半时的功耗与能耗；(c)电压频率都减半时的功耗与能耗

2.2.3 任务迁移技术

对于多核或者众核处理器来说，每个核运行不同的任务，或者每个核上的任务数量不同。核与核之间的负载不同，各核的功耗不同，导致温度不平衡。其实即使负载功耗相同，各核的散热参数也不同，也会导致温度不平衡。这对多核处理器还不明显，因为核数少各核的散热参数基本相同，但是对于众核处理器这就是一个很明显的问题了。在 2.1 中已经说明温度梯度会严重影响芯片的可靠性，静态功耗等。所以解决温度不平衡非常关键，任务迁移是一个很好的解决方法。

任务迁移是任务管理的一种特殊形式，就是将任务从一个计算环境迁移到另一个计算环境中。这本来是一种用在分布式计算中的技术，但现在随着多核众核处理器的出现，其应用更加广泛了。在多核处理器中，任务迁移是任务调度的一个标准部分，它的过程很简单，就是将一个任务从一个核迁移到另个核上去运行。

下面介绍一个例子来说明任务迁移，如图 2-3 。双核处理器上运行着三个任务（A、B、C），每个核可以独立设置其时钟频率和电源电压来降低功耗，去满足当前负载的需求。任务负载由全速等效负载（FSE）表示，全速等效负载是一个任务在核上以最大频率运行时的负载。核 1 上运行任务 A 和 B ，分别有 50% 和 40% 的等效 FSE ；核 2 上运行任务 C ，有 40% 的等效 FSE 。在这个例子中，核 1 在理想状态下，可以将频率调整为最大频率的 90% ，核 2 可以将频率调整为最大频率的 40%。这样是最能减少功耗的。但是这样仅仅用 DVFS 方法并不能平衡芯

片的温度。在这个例子中，因为负载不同，核 1 的温度将会高于核 2。因此，处于对温度平衡的考虑，可以将任务 B 在两个核上周期地迁移。这样两个核的负载就会平均（40%＋50%＝65%）。这是一个双核简单任务迁移的例子。对于更多核的处理器，任务迁移策略不会这么简单，需要由理论上的算法去计算如何迁移。

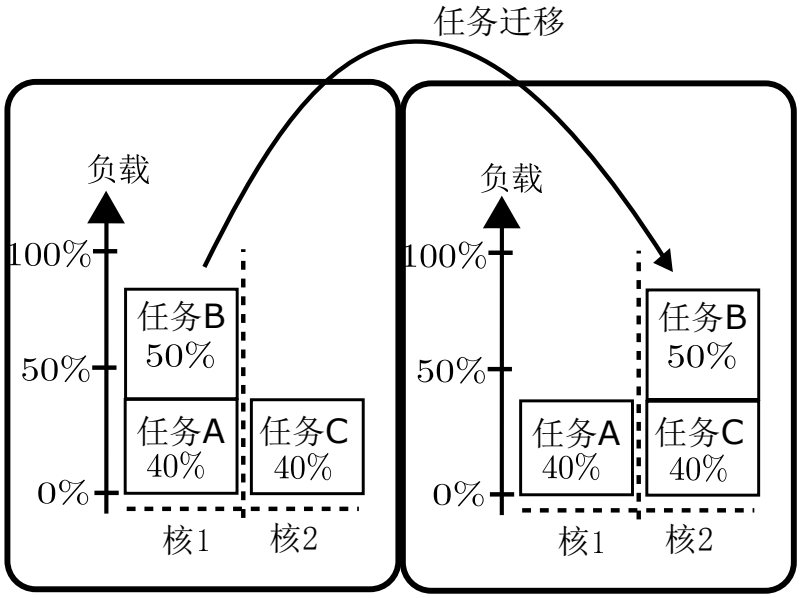


图 2-3 简单的任务迁移例子

2.3 预测控制方法

上面介绍了动态温度管理技术，然而这些技术是实际的操作。如果只应用这些技术，依靠温度传感器的温度数据，是无法避免高温事件的。因为那只能在超过安全阈值之后才能采取操作，动态温度管理操作严重延时。这样不仅管理效果不好，而且还可能引起更大的问题，比如某核上任务负载较小温度较低，将要将其与高温核上任务交换。但是延时之后该任务负载加重，这样就会使高温核上出现更高的温度。针对这问题，对动态温度管理技术提供指导性调整意见就很重要了。这里我们介绍一些预测控制方法。

2.3.1 芯片温度预测方法

对于预测控制，最简单的就是预测温度。温度预测技术是控制决策的基础，只要预测是准确的，提前采取适当的操作就可以避免高温事件。温度预测技术方面的研究已经很多，过去常用的温度预测控制方法是，直接对将来芯片的温度，和负载功耗进行预测。根据预测出的下一时刻的温度，反过来对现有的负载功耗进行调整。这里我们简要介绍两种温度预测方法，它们的主要区别是温度预测

模型不同，基于递归最小二乘法（RLS）的温度预测 [20] 和基于自回归移动平均值(ARMA)的温度预测 [21] 。

在 [20]中，通过分析，影响芯片温度变化快慢有两个因素，一个是当前温度与稳态温度的差值，另一个是应用程序本身。第一个因素是长期的温度行为，第二个因素是短期的温度行为。基于这个分析，基于 RLS 的温度预测由两部分组成：基于应用程序的温度模型（ABTM）做短期温度预测，基于核的温度模型（CBTM）做长期温度预测。 ABTM 通过观察应用程序近期的温度行为，然后将这一信息纳入递归最小二乘回归模型来预测将来的温度，如式 (2-8) 所示。

$$y = \theta_1 u_1 + \theta_2 u_2 + \cdots + \theta_n u_n \tag{2-8}$$

其中， y 是将要预测的温度， u_i 是最近的 n 个时刻的温度，θ_i 是 RLS 模型将要估计的系数。 CBTM 模型考虑核的长期热行为。它忽略应用的短期功耗变化，假设应用在一个稳定的功耗运行。 CBTM 的温度变化公式如式 (2-9) 所示。

$$T(t) = T_{ss} - (T_{ss} - T_{init}) \times e^{-bt} \tag{2-9}$$

其中，T_{ss} 是稳态温度， T_{init} 是初始温度， b 是温度常数。T_{ss} 对每个应用是预先计算的， b 是线下计算的，由芯片热特性觉定，所以对所有只需要计算一次。用式 (2-9) 就可以预测时间 t 之后的处理器温度。这样就得到了两个预测的温度，最终的温度预测是这两个温度的权重和，如式 (2-10)所示。

$$T_{predict} = w_s T_{ABTM} + w_l T_{CBTM} \tag{2-10}$$

这样芯片将来的温度就可以预测，根据这个温度，来做动态温度管理操作，将芯片温度控制在安全温度以下。这就是 [20]的温度预测控制方法。

在 [21]中， ARMA 预测模型的原理是，当负载不变时，通过回归过去的测量，可以准确地估计温度。式 (2-11) 就是 ARMA 模型，这与RLS模型相似，因为他们都是基于线性回归的模型。

$$y_t + \sum_{i=1}^{q} a_i y_{t-i} = e_t + \sum_{i=1}^{q} c_i e_{t-i} \tag{2-11}$$

其中，y_t 是时刻 t 的温度， e_i 叫做预测误差或者残留噪声。与 RLS 模型相似，系数 a_i 和 c_i 是 ARMA 模型通过计算确定的。该方法也是根据预测的温度来进行动态温度管理操作。

但是通过这两种温度预测控制方法，我们可以看出，即使温度的预测是准确，也并不能对动态温度管理操作提供指导性意见。比如我们要采用 DVFS 操作降低功耗来降低温度，但是我们并不知道将电压和频率降低到什么水平才合适。假如

采用任务迁移方法，我们也并不知道具体多大功耗的任务适合迁移到最低温度的核上，因为高功耗任务迁移到低温核上可能引起更高的高温点。

2.3.2 模型预测控制方法

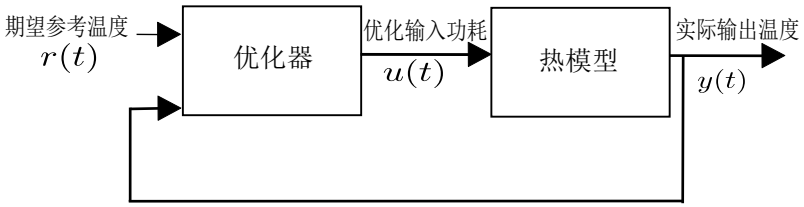


图 2-4 模型预测控制基本结构

最近，针对 2.3.1 节中的问题，即仅仅预测温度并不能很好的给动态温度管理操作提供很好的指导性意见，模型预测控制（MPC）方法 [37] 被引入到动态温度管理中 [25]，该方法可以给动态温度管理操作提供良好的指导性意见。下面我们介绍一下模型预测控制方法。

图 2-4是模型预测控制方法的概念结构。 MPC 用在动态温度管理中源于这样的想法，采用用于控制处理器的热模型做预测，预测给定期望温度下的所需的功耗。注意，该预测方法与 2.3.1 节中的温度预测方法不同， MPC 并不是预测温度，而是根据设定的温度和处理器的热模型来预测输入功耗为多大。这种预测能力允许在线解决优化控制问题，在未来的预测时间内，对应的优化输入和输出可以使跟踪误差最小化。跟踪误差就是期望的温度和预测的温度的差值。

优化的结果根据滚动优化策略来采用 [38]，如图 2-5所示。N_p 表示预测范围或者输出范围长度，N_m 表示控制范围或者输入范围长度。在时刻 t 进行预测时，设定N_p 长度范围的期望输出温度，即图中虚线所示。这样就可以预测出可以操纵的输入功耗 u 和预测出的输出温度 y 。在预测出的优化输入功耗序列（$u(t)$ 到 $u(t+N_m-1)$）中，只应用第一个 $u(t)$ 。优化序列中的其余值被丢弃，在时刻 $t+1$ 进行新一轮的预测。这就是滚动优化策略。

2.4 本章小结

本章首先分析了温度不受管理的情况下，会严重影响芯片可靠性，寿命和性能。高温对功耗也有互为因果的关系。对于常规散热片，散热成本也是急剧增加。然后，对动态温度管理相关技术进行介绍，详细分析动态电压频率调整技术通过调整电压频率对功耗的影响。对任务迁移技术也进行详细说明。动态热管理技术需要好的控制方法做引导，最后介绍温度预测方法和模型预测控制方法，完整的

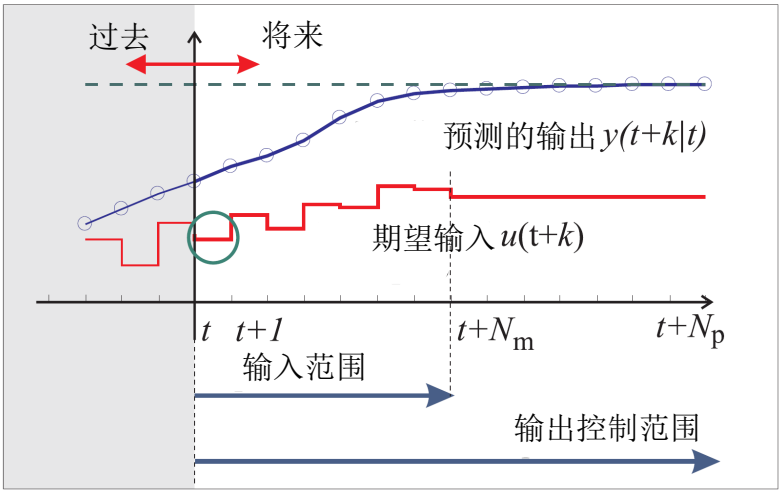


图 2-5 滚动优化策略

引导控制策略才能让动态热管理技术发挥最大作用。本论文的方法就是将模型预测控制方法与电压频率调整技术和任务迁移技术相结合。

第三章 芯片热建模方法

为了应对超大规模集成电路功耗和的增长，在设计阶段做热分析变得越来越重要，做热分析需要用到热模型。超大规模集成电路早期设计阶段，就是在还没有版图信息时就需要做热分析，现在体系结构层面的热管理技术也利用简洁热模型做温度预测。要做热建模的根本原因是，温度的波动不是简单正比于功耗，也不是功耗密度。在空间和时间上还有很多其他因素显著影响温度分布，都是要考虑进去的。这些因素包括热扩散和时空温度滤波效应。因此，为了执行精确的热分析，温度必须直接建模。

热量从一个小的界面传到大的界面是发生热扩散。在时域长热时间常数的硅片和封装中，温度过滤往往会滤掉功率和功率密度的快速变化（高频分量）。功率和功率密度在一个很小的尺寸上变化也会发生空间温度过滤（高空间频率）。

这一章主要介绍一下热模型基本知识尤其是热系统与电路系统的对偶关系，然后详细介绍 HotSpot 简洁的热建模方法。本论文中提出的分层动态温度管理方法实现就是采用的 HotSpot 热模型。

3.1 热传导理论

所有的集成电路产生的热必须被移除或传送到周围环境当中，不然的话，温度积累会越来越高。根据热力学第一定律，能量守恒定律，从热区域传出的热能和冷却液传入的热能是相等的。热力学第二定律是，热一定是由热的区域传到冷的区域。

热传导控制方程是傅里叶定律：

$$q = -k\frac{dT}{dx} \tag{3-1}$$

式 (3-1) 是傅里叶定律的一维形式。其中，q 是热通量（W/m^2），即单位面积单位时间的热流。 k 是材料的热导率（$W/(m \cdot K)$）。式 (3-1) 表明在介质的一点上，热通量 q 是和这一点上的温度梯度成正比关系的。减号表示热流是向温度降低的方向。 参考图 3-1，这里 $q = Q/A$ ，其中 Q 是热传输率，就是单位时间上的热生成或者消散的热量，通常情况下等于功耗 P，也就是能量消耗率。 A 是热传导面积。式 (3-1) 就变成式 (3-2)。

$$Q = -kA\frac{T_2 - T_1}{L} \tag{3-2}$$

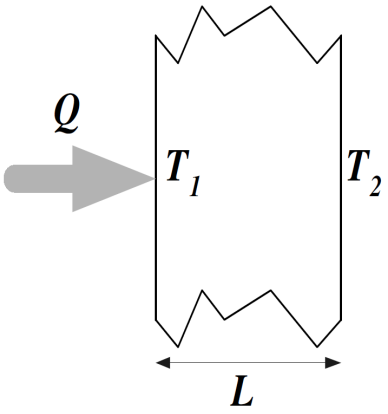


图 3-1 一维热传导

如果定义热阻 $R_{th}=(T_1-T_2)/Q$，温度下降值除以热传输率，我们可以得到

$$R_{th}=(T_1-T_2)/Q=\frac{1}{k}\frac{L}{A} \tag{3-3}$$

可以发现式 (3-1) 和大家熟知的电路理论中的欧姆定律相似：

$$R=(V_1-V_2)/I=\rho\frac{L}{A} \tag{3-4}$$

因此电路和热系统有一个有趣的对偶 ——热阻率（$1/k$）和电阻率（ρ）；温度差（ΔT）和电压差（ΔV）；热传输率（Q 或者功耗 P ）和电流（I）；热阻（R_{th}）和电阻（R）。

热传导也是一个瞬态过程，更加通用的带时间的热扩散方程为：

$$\rho c_p\frac{\partial T(x,y,z,t)}{\partial t}=\nabla\cdot[k(x,y,z,T)\nabla T(x,y,z,t)]+g(x,y,z,t) \tag{3-5}$$

其中，ρ 是材料的密度（kg/m^3），不是电阻率；g 是热源的体积功耗密度（W/m^3）；c_p 是比热（$J/(kg\cdot^\circ C)$）。

对于稳态情况，就是式 (3-5) 中 $\partial T/\partial t$ 项为 0。可以验证，稳态下热扩散方程的一维形式可以简化成式 (3-1) 的傅里叶定律。

假设 g 和 k 是常数，将式 (3-5)写成一维形式，两遍从 0 到 L积分变量 x，$g\cdot L=Q/A=q$，即热通量，就得到了如下形式：

$$(\rho c_pAL)\frac{dT(t)}{dt}=kA\frac{\Delta T(t)}{L}+Q \tag{3-6}$$

式 (3-6) 的右边第一项是通过热阻 R_{th}的热量，与式 (3-3)相似。注意$\Delta T=T_2-T_1$。该式可以变换为

$$C_{th}\frac{dT(t)}{dt}+\frac{T_1-T_2}{R_{th}}=Q \tag{3-7}$$

其中，$C_{th} = \rho c_p AL = \rho c_p V$ 被定义为热容，V 是材料的体积。

根据电路理论，$C\frac{dV(t)}{dt} = i_c(t)$ ，表示通过电容的电流量等于它的电容值与它两端电压差的一阶导的乘积。这正类似于式 (3-7) 中的第一项。这也正是定义C_{th}为热容的原因。热容表示材料的热吸收能力，正如电容表示材料吸收积累电荷的能力。式 (3-7) 表示通过热容的热流（AC 部分）加上通过热阻的热流（DC 部分）等于通过该材料的总热流。

表 3-1 中总结了热系统和电系统的对偶现象，HotSpot简洁热建模方法构建热容热阻网络将会用到这个对偶关系。

表 3-1 热系统与电系统的对偶关系

热系统	单位	电系统	单位
P，功耗	W	I，电流	A
T，温度	K	V，电压	V
R_{th}，热阻	K/W	R，电阻	Ω
C_{th}，热容	J/K	C，电容	F

3.2 HotSpot热建模方法

HotSpot 热建模方法提供了一种精确有效构建简洁热 RC 网络方法，可以简化热扩散方程。

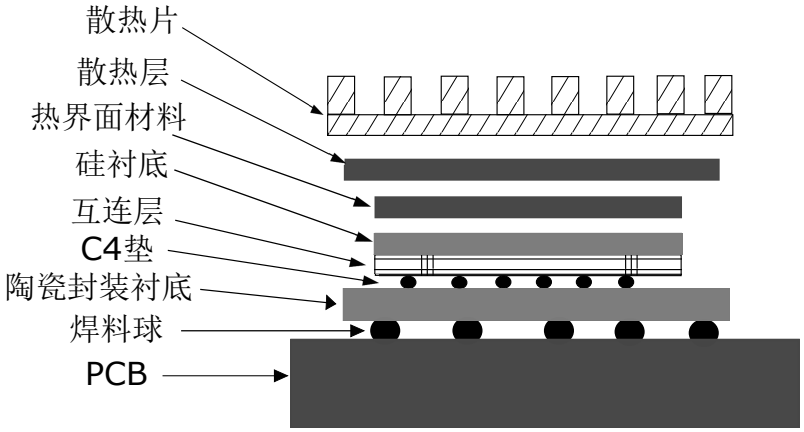


图 3-2 典型CBGA封装的堆叠层

大多现在VLSI系统的封装有若干不同材料的叠层组成[39]，如图 3-2。HotSpot热模型就以此为例来说明。典型的层有：散热片，散热层，热界面材料（导热膏），硅衬底，互连层，C4 垫，陶瓷封装衬底，焊料球等。堆叠芯片封装（SCP）和3D IC设计也是堆叠分层结构，可以作为通用堆叠结构的扩展很容易建模，如图

3-3。硅器件层产生的热有两条主要传输路径如图 3-3，上面是主要热流路径，从硅片传到热界面材料，散热层，散热片，最后对流传到周围空气中；下面是次要热流路径，从硅片传到互连层，C4 垫，陶瓷封装衬底，焊料球，到印刷电路板。HotSpot 简洁热模型包含这两条热流路径的所有层，并特别强调主路经和片上互连层。这些部分的对详细温度分布是很重要的，准确的温度分布才是需要的。在模型中也考虑各层的横向热流来实现更准确的温度估计。

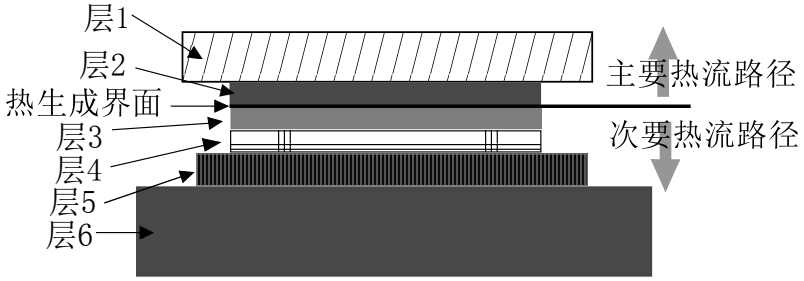


图 3-3 HotSpot中用的抽象堆叠层结构

3.2.1 主要热流路径

首先介绍一下主热流路径模型。构建 HotSpot 热模型时，不同层的位置是首先要被确定的。然后每一层被划分为若干块。例如图 3-4 (c) 中，可以根据不同的设计需求，将硅衬底按照结构单元或者规则网格划分。为简单起见，图 3-4 (c) 只显示了三块。其他影响整个芯片温度分布的层可以做类似于硅衬底层的建模。

对于并不需要太详细温度信息的层，我们可以如图 3-4 (a) 所示简单划分该层。图 3-4 (a) 层中心遮挡部分是被另一个相邻曾覆盖，如3-4 (c) 所示的那样。中心部分可以类似相邻层有相同数量的节点，也可以把这些节点合并成更少数量的节点，这取决于计算精度和速度。图 3-4 (a) 中层周边部分，被分成了四个梯形块，每一块分配一个节点。

每一层每一块或者网格单元都有一个垂直热阻连接到下一层，和几个横向的热阻连接到同一层的相邻块或网格单元。图 3-4 (b) 就是一个层的侧视图，表示出了横向和垂直热阻。垂直热阻表示为$R_{vertical} = t/(k \cdot A)$，其中 t 是该层厚度， k 是该层材料的热导率， A 是这块横截面的面积。每一层不在划分成多个薄层，即这个方法不是完全 3D 的。这是早期设计阶段的一个合理近似，因为每一层相对较薄（$1mm$ 或者更薄），垂直方向更进一步的分化会引入更多计算量但不会明显提高精度。

横向热阻的计算不像垂直热阻那样简单，因为横向热扩散必须考虑。块一侧的横向热阻可以被看做层内相邻部分到这个特定块的热阻。横向热阻通常比垂直

热阻大得多，因为横向传热界面通常比垂直界面小得多。为了阐明扩散热阻如何计算，考虑图 3-5中的相邻两块，块 1 和块 2。长度分别为 $L1$ 和 $L2$ ，芯片厚度为 t 。现在我们计算横向热阻 R_{21} ，就是从块 2 中心到块 1 块 2交界边的热阻。考虑热量通过 L_1t 和 L_2t定义的表面区域从块1传到块2。接收热量的硅本体区域是 L_2t，本体厚度是 $W_2/2$。有这些量之后，就可以通过 [40] 中的公式来计算扩散热阻。

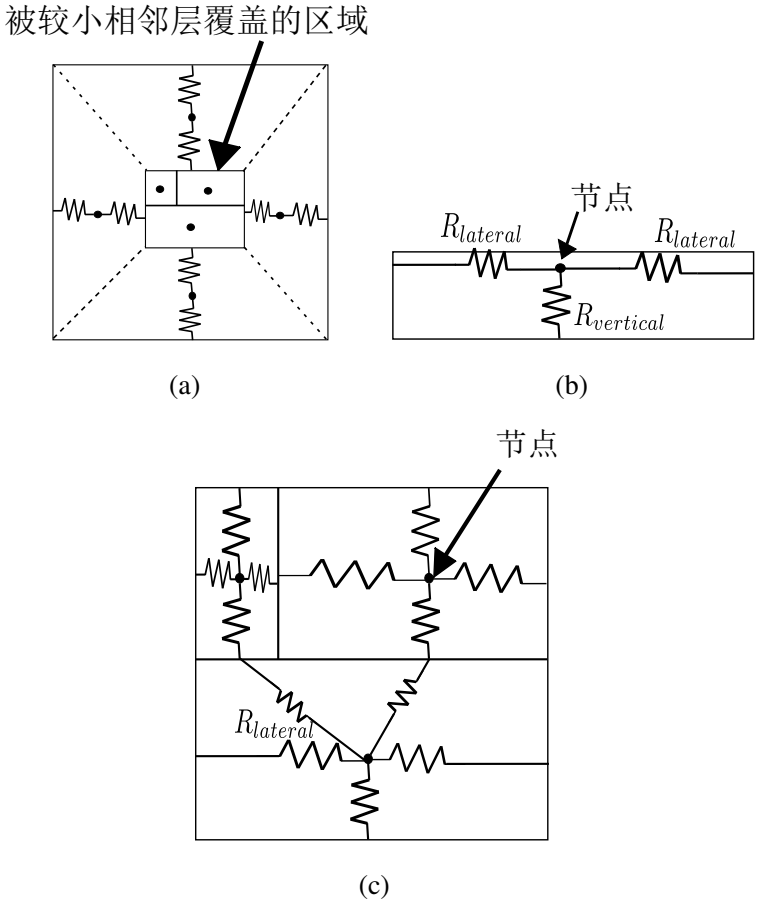


图 3-4 层划分和等效热阻示意图

(a)大面积层的划分（顶视图）；(b)块中的水平和垂直热阻（侧视图）；(c)一层可被划分成任意数量的块（顶视图）

每个节点也都有一个连接到地的热容 $C_{th} = \alpha c_p \rho t A$ ，其中 c_p 和 ρ 分别是材料的比热和密度。因子 $\alpha \approx 0.5$ 是集中分布占热 RC 时间常数的比例因子。

最后，扩散到空气的对流热阻建模为 $R_c = 1/(h \cdot A_c)$ ，其中 A_c 是对流表面积， h 是热传输系数，这是边界条件依赖的。对于不同对流条件下典型散热片的典型 h 值可以在散热参数表中找到。

这样主热流路径的建模就完成了，图 3-2 中封装的从硅片到散热片到周围空气的主热流路径模型如图 3-6 。图 3-6 的例子中，是图3-2中主热流路径倒置结构，硅芯片被划分为 3×3 网格单元。为了提升精度，热界面材料层划分与硅芯片相同。散热层中热界面材料层正下方的部分的划分与热界面材料层相同，周围部分被划分为四个梯形。散热片层分成五块：一块是散热层正下方对应部分，其余四块为周边梯形。每一个网格单元映射热电路中的一个节点，由横向和垂直热阻连接它们。每一个节点都有一个连接周围环境的热容。每一个硅片网格单元消耗的功耗被建模成“电流源”，连接到相应的节点。

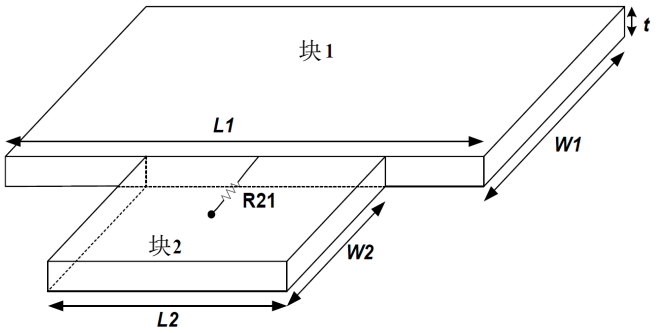


图 3-5 扩散热阻示意图

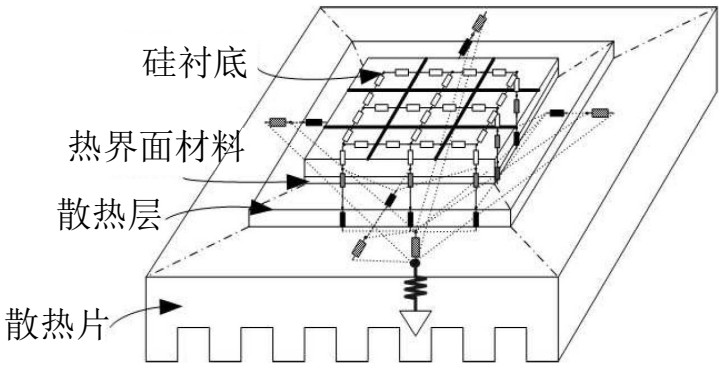


图 3-6 芯片3×3网格单元简洁热模型示例

3.2.2 次要热流路径

上一节解释了 HotSpot 主要热流路径的建模。次要传热路径，即硅衬底，芯片互连层，C4 垫，陶瓷封装衬底，焊料球，印制电路板。次要热流路径通常散去不可忽略的热量（高达30%）。忽略次要热流路径会导致温度预测不准确。本节中

次要热流路径模型分成两部分，一部分对应互连层，另一部分就是从C4 垫到印制电路板。

互连层热模型有两个方面，第一个是金属线的自热功率，$P_{self}=I^2\cdot R$，这里 I 是导线中的电路，$R=\rho_m\cdot l/A_m$ 是导线电阻，ρ_m 是金属电导率（与温度相关），l 和 A_m 是金属线长度和截面积。这里通常是估计各金属层上导线的平均长度和平均电流。详细的方法见 [41]。第二就是每条金属线和它周围的层间介质的等效热阻，通孔在不同金属层之间的热传递也起到很重要的作用，也对其进行建模。

为构建模型，以图 3-7 中的两条相邻互连线（线1 和线2）为例。顶上是其相邻金属层的正交互连线。所有的线周围都是层间介质。互连线和周围层间介质的等效热阻等效热阻 R_0 是从线1 到其上方 $d/2$的区域。其中 d 是两个金属层之间层间介质的厚度。d 的另一半属于线1 上面的金属层，计算那层等效热阻的时候才会考虑。假设同一金属层中的所有信号线都是相同的，线1 和线2 是在同一时间消耗相同的功耗，具有相同的温度。图中外虚线区域为近似等温面，用于计算 R_0，它不与相邻层等温面重叠。热传导角度为 $\theta=2\cdot tan^{-1}(D/(d+H))$，如图所示。每条互连线对应的等效热阻为

$$R_0=ln(\frac{d+2r}{2r})/(\theta\cdot k_{ins}\cdot l) \tag{3-8}$$

其中，$r=\sqrt{WH/\pi}$，k_{ins} 是层间介质的热导率，l 是互连线长度。

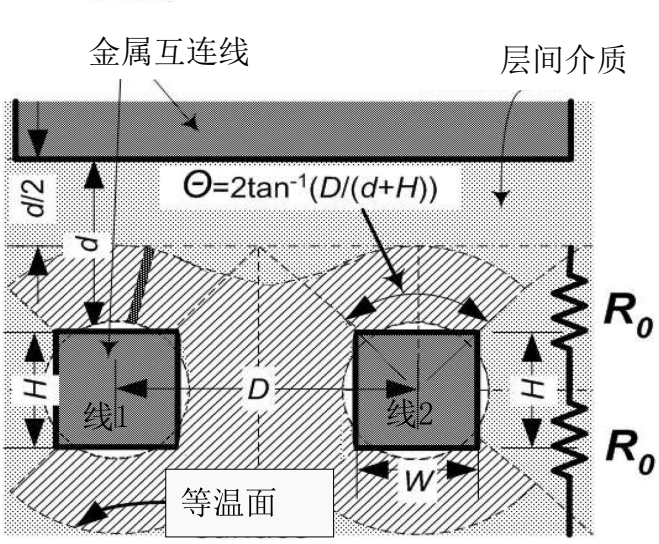


图 3-7 计算互连线和周围介质等效热阻示意图

线 1 和线 2 之间也有横向热阻 R_{lat}，但是因为线 1 和线 2 相同而且有相同的温度，这里没有热传输，所以 R_{lat} 被去除。

层间热也通过通孔传导。用于信号互连的通孔数量的简单近似是假定每条互连线有两个通孔，一个连接到上层金属层，另一个连接到下层金属层。每个通孔的热阻近似计算为 $R_{via} = t_v/(k_v A_v)$，这里 k_v 是通孔填充材料热导率，t_v 和 A_v 是通孔厚度和截面积。

两金属层之间的所有互连线和通孔的热阻可以看成是并联。所以组合两金属层之间所有热阻就能得到两金属层间的总等效热阻。

对于每个金属层和层间介质的热容，可以根据其尺寸和材料特性用 3.2.1 节中的类似公式计算。

最后从 C4 垫到印制电路板的热模型也是一系列的热阻热容对，分别代表垫凸点/片下填充，陶瓷基板，球阵列和 PCB 对流。热容和热阻的计算用 3.2.1 节中主热流路径上各层的类似方法。垫凸点热阻一端连接到互连层模型，另一端连接到代表陶瓷衬底的热容热阻，等等。

3.3 热建模在动态温度管理中的应用

传统的动态温度管理是在芯片级进行的。芯片级动态温度管理通过功耗控制技术减少热产生密度，来达到控制温度的目的。芯片级动态温度管理技术显著降低散热成本，对典型应用可以允许最大性能，但是对于超出设计温度点的应用程序，性能会显著降低。另一方面微体系结构动态温度管理技术能提供更好的性能温度折中，因为具有调用芯片上不同单元的运行信息做处理的独特能力，从而实现对芯片热行为更精细的控制。 HotSpot 简洁热模型在微体系结构级上的应用是成功的。从 HotSpot 得到的各单元瞬态和静态温度估计可以被送到一个精确的处理器仿真器中，在那里动态温度管理技术被实现和模拟。反过来，精确的处理器仿真器提供微处理器的运行信息到体系结构级功耗模型，比如 wattch [42] 。功耗模型的输出功耗又作为 HotSpot 的输入做仿真，更新运行温度信息。这样形成一个可以使处理器进行温度感知操作的控制回路。该回路在图 3-8 表示出来。

很多研究显示采用体系结构级的动态温度管理技术要比芯片级技术有显著的性能提升。比如对大部分基准程序，微体系结构级动态温度管理技术至少能减少 10% 的性能损失。 HotSpot简洁热模型在温度感知微体系结构设计中起关键作用，已经被广泛应用在计算机体系结构研究上。

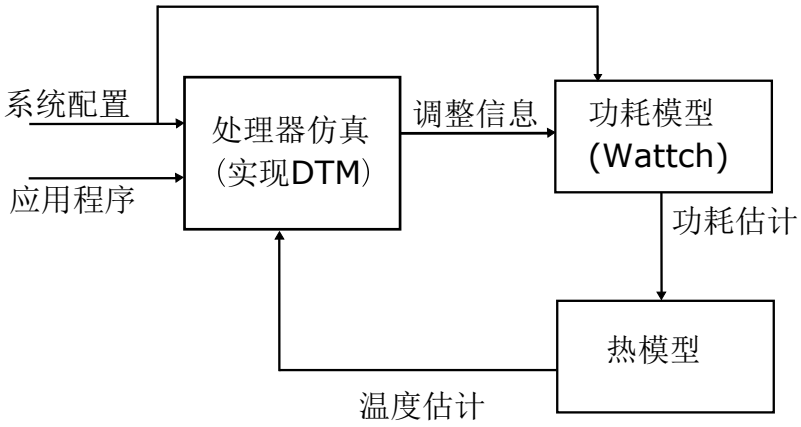


图 3-8 动态温度管理仿真环境

3.4 本章小结

本章针对热建模方法进行了详细讨论分析。首先说明了热系统与电路系统的对偶性，可以用类似电路分析的方法表示热系统。对 HotSpot 简洁热建模方法进行了说明，对处理器的主要热流路径和次要热流路径都进行了建模分析。最后是介绍热模型在动态温度管理中的应用。热建模本来是在芯片设计阶段进行热分析的，但是很早就被引入动态温度管理方法中，只要已知处理器上的功耗分布就能够利用热模型准确计算芯片的温度分布。这对众核动态温度管理相当重要。本论文的方法就是用的 HotSpot 热模型做温度计算。 HotSpot 热模型在本文方法中的作用就是在 3.3 节所描述那样。

第四章 基于模型预测控制的动态温度管理

本章详细介绍新动态温度管理方法的第一部分内容，即基于模型预测控制方法结合任务迁移和 DVFS 的方法。在前面的章节中，分别介绍了动态温度管理技术，模型预测控制方法，和 HotSpot 热模型。新的动态温度管理方法首先是将处理器 HotSpot 热模型和模型预测控制（MPC）方法结合起来，最一般的用途就是进行温度计算。模型预测控制方法通过设定期望温度，来预测期望功耗，本章详细介绍其推导过程。然后就是知道期望功耗之后，有了引导就需要有效控制策略进行动态温度管理操作。 4.1 先介绍新的动态温度管理方法基本流程，说明新的方法需要哪些技术方法，要进行哪些计算，最终要得到的是什么样的结果。 4.2 节中介绍结合模型预测控制 MPC 的热模型， 4.3 节介绍怎么用 MPC 计算期望的用于引导动态温度管理方法功耗，最后 4.4 节说明怎么样用 MPC 计算得到的功耗来引导任务迁移和 DVFS。

4.1 新的动态温度管理方法基本流程

现在开始介绍新的动态温度管理方法，首先我们先说明一下新方法的简要流程，如图 4-1 所示。首先是要建立微处理器的热模型，一方面微处理器的热模型用于温度计算，即利用已知功耗分布轨迹就可以计算出温度分布轨迹。另一方面用于给模型预测控制方法进行期望功耗分布的计算。然后当前功耗分布就要根据期望的功耗分布进行重新规划分布和调整。这也就是要进行任务迁移和 DVFS 操作。但是操作如何进行就需要在新的动态温度管理方法中给出。期望的功耗分布已经给出了一个很好的引导。首先如果直接进行 DVFS 操作，就是当前任务所在核通过调整电压和频率将功耗调整到对应的期望功耗上。但是问题是，一般处理器都是在最大电压和频率上运行的，可以将功耗往更小的功耗调整，不能将功耗调整到更高的水平。这样虽然能保证芯片温度不超过安全温度，但是会极大影响处理器性能。但是如果一个核上的期望功耗与该核上的当前功耗差别太大，而与另外一个核上的当前功耗近似相等，这样就可以考虑进行任务迁移。将这个问题优化为一个任务迁移问题就可以通过二部图匹配来解决该问题。但是匹配算法在处理器核数较多时遇到扩展性问题，在后面的章节中将介绍如何解决该问题。能够匹配的任务尽量全部匹配，剩下少量不能匹配的任务，只能调用 DVFS 来处理，如果对应期望功耗更小就只能对该核降频降压使其满足功耗需求，如果对应期望功耗更大，现在也不能做任何操作，只能全频全压运行。最终的结果，新的温度

管理方法的最终输出是一个优化的功耗分布，也就是确定任务如何迁移（要迁移到哪个核上）， DVFS 对需要调整的核改调整到什么水平。下面分别介绍新的动态温度管理方法的各个部分。

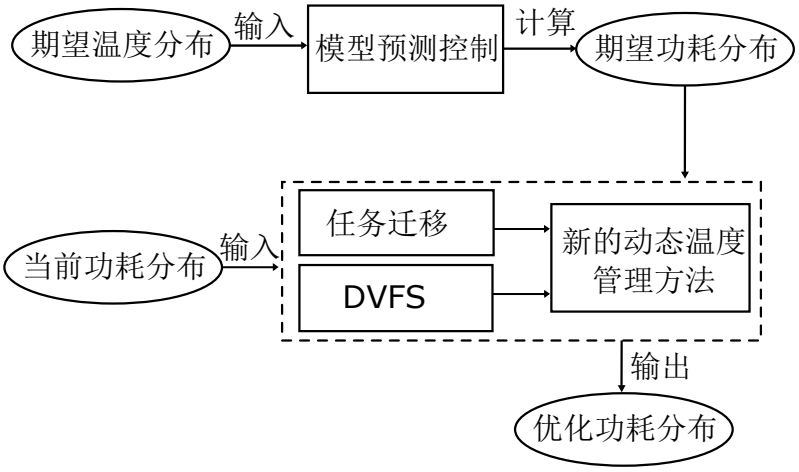


图 4-1 新的动态温度管理方法简化流程

4.2 微处理器热模型

在第三章中已经介绍过，热系统和电路系统是相似的，我们可以用热阻，热容，和等效的热电流电压源来建立微处理器的热模型。在硅片层，这里假设核按规则网格排列，类似于图 4-2所示 9核处理器核分布。这样对硅片层的划分就可以按照核划分为规则的网格单元。类似于电路系统，l 核的微处理器热模型可以被表达为常微分方程 [43]，即式 (4-1)。

$$\begin{aligned} GT(t)+C\dot{T}(t) &= B_cP(t) \\ Y(t) &= LT(t) \end{aligned} \tag{4-1}$$

其中，$T(t)\in\mathbb{R}^n$ 是表示处理器被划分成的 n 块的温度的向量，包括 l个核($l<n$)，封装部分的节点等； $G\in\mathbb{R}^{n\times n}$ 包含热阻信息； $C\in\mathbb{R}^{n\times n}$ 包含热容信息； $B_c\in\mathbb{R}^{n\times l}$ 包括功率输入的拓扑信息； $P(t)\in\mathbb{R}^l$ 是 l 个核在时刻 t 的功耗向量，这就是模型的输入； $Y(t)$ 是 l 个核的温度信息向量，这就是模型的输出； $L\in\mathbb{R}^{l\times n}$ 是输出选择矩阵，从 $T(t)$ 中选择 l 个核的温度。

为了分析热系统，用欧拉方法或者其他数值积分方法将连续常微分方程 (4-1) 离散化为式 (4-2) 的差分方程。

$$\begin{aligned} T(k+1) &= AT(k)+B_dP(k) \\ Y(k) &= LT(k) \end{aligned} \tag{4-2}$$

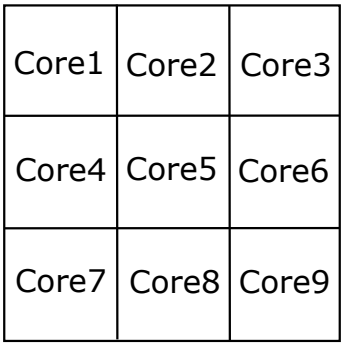


图 4-2 9核处理器核分布

其中，变量 $T(k)$ 、 $P(k)$ 和 $Y(k)$ 是公式 (4-1)中 $T(t)$ 、 $P(t)$ 和 $Y(t)$ 的离散形式，A 和 B_d 是由 G、 C、和 B_c 根据离散(4-1)的特定的数值积分方法得到的。

对于一般用途，(4-2) 中的热模型是用于用芯片上的各单元功耗（即输入$P(k)$）来计算芯片上核的温度（即输出的 $Y(k)$）。

4.3 用模型预测控制方法计算期望的功耗

4.2 节中已经说明，用热模型 (4-2)，可以由给定的功耗输入 $P(k)$ 来计算芯片上的核的温度 $Y(k)$，这足以进行热估计和仿真。对于动态温度管理问题，由给定的功耗来计算期望的功耗也是很重要的，因为动态温度管理方法需要操作功耗方面来管理温度。有的时候为了简化，可以利用静态热模型由给定的温度信息来计算功耗，静态热模型可以由模型 (4-1) 去掉热容项来得到。然而，基于静态热模型的动态温度管理方法会忽略掉当前热状态，但是当前的热状态在做管理决策的时候非常重要。这个方法也假设温度和功耗大致温度，这样会影响动态温度管理效用。为了减轻这个问题，一些反馈控制方案或者优化设计用(4-1)中的瞬态热模型（或者(4-2)中的离散形式）在动态温度管理决策时进行更好的功耗计算。尽管这个方法考虑了当前温度状态而且处理了热与功率的影响，但是这种方法不能得到一个平滑的温度控制。主要是因为这种方法缺乏未来预测能力，而且只能为温度控制获得当前步的优化功耗。在这篇文章中，我们用了基于模型预测控制功率计算方法，这个方法将(4-2) 中的瞬态热模型扩展成预测形式，它具备为平滑精确热管理计算未来期望功耗的能力。模型预测控制方法利用(4-2) 中的系统模型可以计算得到输入的调整需求，这样就能满足设计者定义的输出。为了最大化处理器性能，处理器每个核允许的最高温度称作顶温度 Y_{max}， Y_{max} 通常当做设计者定义的输出来被趋近。顶温度可以根据现实中的不同应用来被调整，它可以稍微低于处理器允许的最高温度以保证绝对的安全。

首先，我们定义状态和温度变量的差：

$$\begin{aligned}\Delta T(k) &= T(k) - T(k-1)\\ \Delta P(k) &= P(k) - P(k-1)\end{aligned} \tag{4-3}$$

取(4-2) 的相邻两步的差，这里有

$$\begin{aligned}\Delta T(k+1) &= A\Delta T(k) + B_d\Delta P(k)\\ Y(k+1) - Y(k) &= LA\Delta T(k) + LB_d\Delta P(k)\end{aligned} \tag{4-4}$$

引入一个新的变量

$$\hat{T}(k) = \begin{bmatrix} \Delta T(k) \\ Y(k) \end{bmatrix}$$

将 (4-4) 重写成下面改进的模型

$$\begin{aligned}\hat{T}(k+1) &= \hat{A}\hat{T}(k) + \hat{B}\Delta P(k)\\ Y(k) &= \hat{L}\hat{T}(k)\end{aligned} \tag{4-5}$$

其中

$$\hat{A} = \begin{bmatrix} A & 0_m \\ LA & I \end{bmatrix} \qquad \hat{B} = \begin{bmatrix} B_d \\ LB_d \end{bmatrix}$$

$$\hat{L} = \begin{bmatrix} 0_m & I \end{bmatrix} \qquad \hat{T}(k) = \begin{bmatrix} \Delta T(k) \\ Y(k) \end{bmatrix}$$

0_m 是一个合适维度的全零矩阵。

到这里我们已经从(4-5)中得到了输入功耗差和输出核的温度之间的关系。下面，需要确定输入功耗的差来满足核期望的顶温度。假设核未来几个时间步长的顶温度已经给出，写成下面向量的形式。

$$Y_{ceil} = [Y_{max}^T, Y_{max}^T, ..., Y_{max}^T]^T \in \mathbb{R}^{lN_p \times 1}$$

在这个向量中，$Y_{max} \in \mathbb{R}^{l\times 1}$ 包含每一个核的顶温度。这里我们假设顶温度不变，这个也符合实际情况，并不是新方法的限制。 N_p 表示从当前到未来N_p 步的时间帧，称作预测域。为了使核的温度在时间域内趋近于顶温度，将来的控制轨迹（并不知道，需要计算）表示为（当前时刻为 k ）

$$\Delta P_k = [\Delta P(k), \Delta P(k+1), \ldots, \Delta P(k+N_c-1)]^T$$

其中，N_c 称作控制域。核的预测温度定义为

$$Y_k = [Y(k+1|k)^T, Y(k+2|k)^T, \ldots, Y(k+N_p|k)^T]^T$$

其中， $Y(k+j|k)$ 利用当前时刻k的信息预测出来的核在时刻 $k+j$ 的温度。如果ΔP_k已知，Y_k 就可以用下面的公式计算出来。

$$Y_k = V\hat{T}(k) + \Phi\Delta P_k \tag{4-6}$$

其中

$$V = \begin{bmatrix} \hat{L}\hat{A} \\ \hat{L}\hat{A}^2 \\ \vdots \\ \hat{L}\hat{A}^{N_p} \end{bmatrix} \Phi = \begin{bmatrix} \hat{L}\hat{B} & 0 & 0 & \cdots & 0 \\ \hat{L}\hat{A}\hat{B} & \hat{L}\hat{B} & 0 & \cdots & 0 \\ \hat{L}\hat{A}^2\hat{B} & \hat{L}\hat{A}\hat{B} & \hat{L}\hat{B} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \hat{L}\hat{A}^{N_p-1}\hat{B} & \hat{L}\hat{A}^{N_p-2}\hat{B} & \hat{L}\hat{A}^{N_p-3}\hat{B} & \cdots & \hat{L}\hat{A}^{N_p-N_c}\hat{B} \end{bmatrix}$$

接下来，我们想计算功耗，使利用这个功耗计算出的核的温度Y_k 和设计者定义的期望的顶温度Y_{ceil} 之间的差最小。我们首先将这个差的测量值表示为$(Y_{ceil} - Y_k)^T(Y_{ceil} - Y_k)$ ，最优的功耗分布是使$Y_k = Y_{ceil}$ 的功耗。此外，对于实际的考虑，我们优先选择功耗分布不进行急剧变化。所以额外的调整项 $\Delta P_k^T R\Delta P_k$ 也加到$(Y_{ceil} - Y_k)^T(Y_{ceil} - Y_k)$ 上，这就形成

$$F = (Y_{ceil} - Y_k)^T(Y_{ceil} - Y_k) + \Delta P_k^T R\Delta P_k \tag{4-7}$$

作为变量ΔP_k 的最终函数，我们的目标就是使这个函数最小化。 $R = rI_{N_c\times N_c}$ 是一个调整矩阵，r 是调整参数，这决定该函数两项之间的权重。对于不同的核数通过实验可以的到合适的值。这里要注意，Y_k 也是未知变量ΔP_k 的函数。

下一步，对式(4-7)求一阶导数，使它等于零，就可以得到优化的最小值。ΔP_k的解是

$$\Delta P_k = (\Phi^T\Phi + R)^{-1}\Phi^T(Y_{ceil} - V\hat{T}(k)) \tag{4-8}$$

在每个模型预测控制MPC时刻 k，我们只需要从(4-8)计算得到的控制信号$\Delta P(k)$，然后更新功耗分布。

$$\bar{P}(k) \leftarrow P(k) + \Delta P(k) \tag{4-9}$$

其中$\bar{P}(k)$ 是更新后的功耗分布。结果就是，更新后功耗输入使核的温度 $Y(k)$ 趋近于期望顶温度。换句话说，更新后的功耗是在没有温度要求冲突下能达到的最高温度。

4.4 基于期望功耗的任务迁移和动态电压频率调整

式 (4-9) 中模型预测控制方法提供的期望的功耗分布可以用于执行动态热管理。动态电压频率调整（DVFS）可以很容易的和模型预测控制（MPC）结合，仅仅需要调整每一个核的电压和频率去匹配由 MPC 得到的期望功耗分布。然而，DVFS 可能导致处理器性能急剧下降。其根本原因是如果一个核已经在最高的频率和电压水平，那 DVFS 只能降地这个核的功耗，不能提升它的功耗。例如，如果一个负载正在核 i 上运行，消耗功耗为 p_i ，而且此时核 i 已经是最高电压水平和最高频率，这个时候 MPC 建议核的功耗 $\bar{p}_i$（$\bar{p}_i > p_i$）。这种情况下，核 i 不能做任何调整。但是此时可能存在一个核 j ，其功耗 $p_j \approx \bar{p}_i$，而且正好 DVFS 为满足热限制要调整该核到一个较低的功耗（比如，等于 p_i ）。这样核 j 性能会降低，导致较低的吞吐量。

实际上，显然在这个例子中如果我们交换核 j 和核 i 的负载，就不需要DVFS来调整电压频率，处理器性能也不会受到损害。所以，可以先执行任务迁移，任务迁移就是将 P 和 $\bar{P}$ 中的相近的元素匹配成对。根据匹配对，来进行任务迁移操作。这个匹配过程是一个任务分配问题，但是可以看出这个问题与传统的 n 个人分配 n 个任务不同。这个问题中如果 p_j 和 $\bar{p}_i$ 相差太多是不能够匹配的。所以该问题修正为 n 个核分配 m 个任务，其中 n 可能等于 m，也可能不等于 m，甚至 n 可以大于 m，也可以小于 m 。$|p_j - \bar{p}_i|$ 小于一定的阈值，也就是两个功率相差值在一定的阈值之下时，才可以做任务迁移。假如$n > m$，也就是说期望功耗值与现有功耗值近似的个数较少。我们也需要在这些功耗值中找出近似程度最好的匹配对，也就是以匹配对两功耗差值绝对值为权重，找出的 m 个匹配对的权重和最小。这个问题仍然可以构建成一个带权重的二部图，当成一个二部图匹配问题来处理，需要找出的是最小权重匹配。

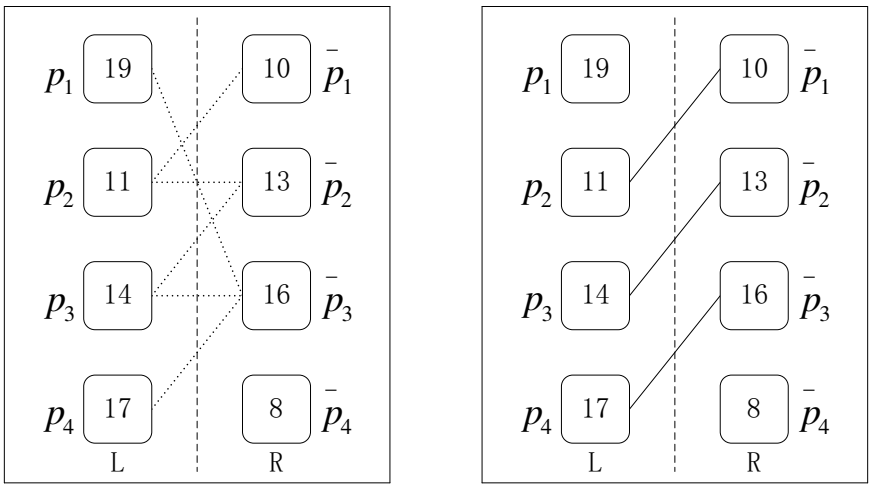
相应的二部图可以表示为 $\mathcal{G} = (L, R, E)$，这里 L 和 R 分别表示现有功耗和 MPC 预测出来的期望功耗的集合。$L = \{p_1, p_2, \cdots, p_l\}$， $R = \{\bar{p}_1, \bar{p}_2, \cdots, \bar{p}_l\}$， E 包含 L 和 R 之间的部分边：我们定义一个阈值 e_{th}，只有边 $(p_i, \bar{p}_j)$ 满足 $|p_i - \bar{p}_j| < e_{th}$ 才将该边放到 E 中，其权重为 $e_{ij} = |p_i - \bar{p}_j|$。

图 4-3 展示了一个二部图匹配的例子，这个例子中带权重的二部图阈值为 $e_{th} = 3$。这里就成了一个 3 个核分配 4 个任务的问题，所以最终只能有 3 个匹

配对。解决匹配问题最简单的方法就是设定一个权重矩阵 W 如下。

$$W = \begin{bmatrix} INF & INF & 3 \\ 1 & 2 & INF \\ INF & 1 & 2 \\ INF & INF & 1 \end{bmatrix} \tag{4-10}$$

其中，W 中第一行代表 p_1 分别于 $\bar{p}_1$， $\bar{p}_2$， $\bar{p}_3$ 之间差的绝对值。INF 表示其差值超过阈值直接被设定为无限大，不能匹配。这个简单的例子我们可以直接从矩阵中看出匹配对 $(p_2,\bar{p}_1)$， $(p_3,\bar{p}_2)$ 和 $(p_4,\bar{p}_3)$，如图 4-3 (b)是表示出的。但是对于数量庞大的功耗上，和在算法的实现上还是需要复杂的过程。



(a) (b)

图 4-3 带权重二部图匹配的例子

(a)阈值 $e_{th}=3$的带权重二部图；(b)二部图匹配结果。匹配对用实线连接

最简单的方法就是生成矩阵 W 上的所有匹配独立集合。计算每种分配独立集合的总权值，搜索找到最小的权值集合。这个方法的复杂性由矩阵W 独立分配可能性的个数决定，第一个核（期望功耗）有 4 中选择，第二个就有 3 个选择，第三个就有 2 个选择。总共是 $4\times3\times2=24$ 中分配可能。那么对于 $n\times m$ 矩阵（$n>m$），分配可能数为 $n!/(n-m)!$。因此这个方法是至少指数运行时间复杂度的算法。

事实上存在更好的算法解决这个问题，1950年代詹姆士·芒克勒斯就将匹配问题简化成多项式复杂度的算法，叫做芒克勒斯算法或者有时成为匈牙利算法 [44]。这个二部图匹配问题可以用匈牙利算法解决，匈牙利算法的第一步就是将权重矩

阵 W 每一行中的最小值找出，每一行各元素都减去该行最小值，得到矩阵 W_r。

$$W_r = \begin{bmatrix} INF & INF & 0 \\ 0 & 1 & INF \\ INF & 0 & 1 \\ INF & INF & 0 \end{bmatrix} \tag{4-11}$$

矩阵中每一行至少有一个元素 0。如果对整个矩阵每一行并且每一列都最多只有一个 0 元素，那么这些 0 元素所在的位置就代表了匹配对。在这个例子中显然第三列就有两个 0 元素，并不符合。接下来可以对矩阵 W 选取每列中的最小元素，每列都减去该列的最小元素得到矩阵 W_v

$$W_v = \begin{bmatrix} INF & INF & 2 \\ 0 & 1 & INF \\ INF & 0 & 1 \\ INF & INF & 0 \end{bmatrix} \tag{4-12}$$

在矩阵 W_v 中，每一行每一列都最多只有一个 0 元素，这样就找出了匹配对，即 $(p_2,\bar{p}_1)$， $(p_3,\bar{p}_2)$ 和 $(p_4,\bar{p}_3)$。这是一个简单的例子，利用匈牙利算法到这里就可以找出最佳匹配对。但是对于一般情况，到这一步每一行或者每一列可能都不止一个 0 元素（比如正好一行中有两个相等的最小值）。完整的匈牙利算法比较复杂，参见 [44] 中的描述。我们在这里不再详细描述。总之匈牙利算法的复杂度已经优化为多项式级别，已经极大改善了原始算法的执行时间复杂度。

二部图利用匈牙利算法发现匹配对，就意味着可以根据这些匹配对进行任务迁移，如果 $(p_i,\bar{p}_j)$ 是其中一个匹配对，那么核 i上的负载就要被迁移到核j上。进行匹配过后，可能留下没有匹配上的功耗元素如图 4-3 (b) 中的 p_1 和 $\bar{P}_4$，这些将由DVFS进行处理，后面将会说明。

这里要说明一下，e_{th} 的值的大小决定二部图匹配后匹配不上的功耗元素的数量的多少，而且还控制着温度过高的风险。大的 e_{th} 值导致较少的匹配不上的功耗，更少的 DVFS 操作，温度过高的风险和程度会比较大。但是芯片性能较高。合适的 e_{th} 值应该被设定在可以接受的过热风险和程度上。对于具有不同数量处理器核的不同的处理器，e_{th} 的值是不一样的。对不同的负载，这个值并不需要实时改变。 e_{th} 还有另外一个重要的功能，就是当这里有太多低功耗任务的时候，可以消除不必要的任务负载。想象一下极端的情况，所有的任务都是低功耗任务，那样搜有的核温度都很低。在这种情况下，我们不需要执行任务迁移或者DVFS。如果我们设定了合适的 e_{th}，那么二部图就根本不会有边，就不会执行任务迁移（这样是正确的）。因为P中的元素全都是很小的值，而 $\bar{P}$ 中的值全都是很大的值

（ MPC 为了趋近于顶温度计算出来的），e_{th} 在这里就可以阻止他们相互连接，这样就避免了不必要的任务迁移。

4.5 本章小结

本章开始构建本论文新的动态温度管理方法的第一部分，首先介绍了新方法的基本流程，需要采用模型预测控制方法，任务迁移， DVFS 等，并说明了各部分的作用。并且介绍该方法中要进行的算法设计，最终要得出的是优化的功耗分布。然后介绍了结合模型预测控制的处理器热模型。前面的章节已经说明，动态温度管理操作需要引导控制策略。新方法是采用 MPC 进行预测控制，这里详细推导了处理器热模型进行温度计算和用模型预测控制方法进行期望功耗预测。这样由 MPC 预测的期望功耗和现有的功耗构成任务分配问题。本章 4.4 节对任务分配问题进行了详细分析，说明了问题构成和利用匈牙利算法解决该问题。解决了这个问题就解决了任务迁移的引导问题，该问题被看做是处理任务迁移决策。但是这里有个问题就是匈牙利算法复杂度较高，较少核数的多核系统依此来解决问题是没有问题的，并不会占用太多的时间。但是对于核数巨大的众核系统，该算法时间急剧增加，会超出可以承受范围，使该方法完全失效。后面我们将讨论如何解决这个扩展性问题。

第五章 分层的动态温度管理方法

在这一章，介绍新方法的第二部分内容，即分层的动态温度管理方法。第二部分是在第一部分的基础上进行的，即基于模型预测控制而且采用了任务迁移和DVFS。众核处理器上执行结合任务迁移的 MPC 是一个挑战，因为但处理器处理器核数非常的大，用复杂度为$O(n^3)$的匈牙利算法计算任务迁移的决策需要花费大量的时间，这个时间可以看成是二部图匹配的时间。为了扩展到具有大量处理器核的众核处理器，新的分层方法将处理器分成块，在两个层次上进行任务迁移决策：块内（低层）和块间（高层）。首先在低层，也就是块内执行当前功耗和期望功耗组成的二部图匹配。在低层内没有匹配上的功耗元素，收集起来形成高层，也就是块间。在高层，用改进的迭代最小割算法 [45] 将高层分成“优化”块，在每个“优化”块执行二部图匹配。高层最后没有匹配的功耗元素用 DVFS 来处理，以保证绝对的温度安全。分层算法通过减小二部图匹配规模来降低计算开销，而且匹配是并行执行的，大大减少了计算时间，所以该算法可以扩展到众核系统。

5.1 低层块内任务迁移

在进行任务迁移决策时，我们将众核处理器分割成块。作为第一步，我们可以简单根据核的位置分割处理器。就是在空间上直接划分处理器成块。这一步分割不需要任何开销。我们通常将方形的块叫普通块，在边缘可能会出现矩形或者小的方形块，把这些称作边缘块。

块内匹配就是执行 4.4 节中说明了二部图匹配的过程，这个叫做低层匹配。对每一个块，指定一个块内的核执行匹配的计算。从整个芯片来看，低层匹配的计算是并行执行的。所以底层匹配引入的延迟时间只是一个普通块低层匹配的时间（注意，边缘块比常规快要小，也就是说它们的计算时间也并不计算在内）。

可以调整块内的核数以实现整个算法更小的延迟时间：如果核数很大，低层的功耗匹配将会占用更多的时间，但是可以发现更多的匹配对，将会剩余更少的未匹配功耗到高层，这样高层匹配处理时间就会减少。

图 5-1 (a)是一个简单的低层划分的例子。这是一个 100 核微处理器，被划分为四个 16 核普通块（标记为A、B、D、E），五个核数 4 到 8 核的边缘块（标记为C、F、G、H、I）。低层的功耗匹配将在每一个块内执行。图 4-3 展示的正是图 5-1 (a)中块I内的二部图匹配。

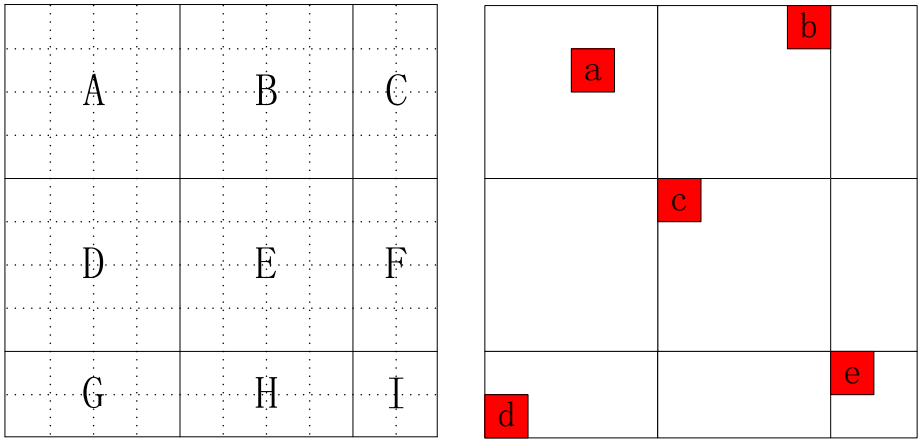


图 5-1 100核微处理器的垂直视图（作为分层算法的一个例子）

(a)微处理器分割成块准备进行低层二部图匹配；(b)低层二部图匹配之后留下的未匹配核（红色）

显然，只执行低层的块内功耗匹配不足以找到所有的匹配对。例如，在图 5-1 (a) 的块 I 中，图 4-3 (b) 已经表示出功耗 p_1 和 $\bar{p}_4$ 不能匹配。对低层匹配阶段块内未匹配功耗直接执行 DVFS 并不是好办法，因为一个块内的未匹配功耗可能在其他块找到很好匹配的期望功耗。这样就可以避免太多不必要的 DVFS 行为，将性能损失最小化。所以，我们可以收集所有块内低层匹配时未匹配的功耗，形成高层。图 5-1 (b) 中表示出第一次底层匹配后未匹配的功耗。

5.2 高层块间任务迁移

在上一节中，我们已经将所有处理器核划分成块，完成了块内低层二部图匹配。将所有低层块中的未匹配上的功耗收集起来，为高层匹配做准备。

我们希望在高层匹配中找到所有的匹配功耗对，只在那些最后匹配不上的功耗上执行 DVFS。似乎我们可以像划分低层块一样，继续按照核的位置来将高层核划分成更大的块。然后我们在新的块内进行二部图匹配，用未匹配上的再形成更大的块。块的大小在迭代中变大，知道整个芯片最后变成一个块。然而，实验表明，在高层匹配中，只有很少比例（低于 25% ）的功耗可以被最终匹配上。相比之下，在底层功耗匹配中比例较大（高于 60% ）。这样小的比例将会使块大小在迭代中增长非常缓慢，甚至有可能块的大小永远不会增长到整个芯片那么大，因为有可能有太多功耗不能最终匹配。所以在块的大小增加到整个芯片那么大之

前，块内收集的未匹配功耗就可能太多而不能处理，因为这需要巨大的计算开销。

为了使高层任务迁移决策更加有效率，我们用另外一种方法将高层功耗划分成块，即最小割算法。首先我们用高层功耗生成一个图（后面将会详细介绍）。然后，我们用最小割算法将图分成两组，每一组是一个新的块（高层的块中的核已经不像低层块一样空间上相邻了）。如果这个新块的大小太大（对二部图匹配算法而言），对这个块在进行一次最小割算法，将其大小减半。在最小割之后，每一个新块内执行二部图匹配。最小割的一个重要特性就是不同的新块之间的元素关系非常弱。新块内的元素关系非常强。这意味着每个新块内部匹配率最大。如果有功耗元素在新块内二部图匹配仍未匹配上，那它也很难与其他新块内的元素匹配上。所以，高层未匹配功耗再收集起来做更高层次的匹配是没有必要的了。

通常情况下，精确的最小割算法开销太大不能实时执行。幸运的是，我们这里并不需要最优的割，我们采用迭代近似算法改进形式，该算法曾被用在网络分割上 [45] 。首先我们用所有高层未匹配功耗建立一个新图 $\mathcal{G}_p=(V_p,E_p)$，其中 $V_p=\{p_1,\bar{p}_1,p_2,\bar{p}_2,\cdots,p_m,\bar{p}_m\}$， E_p 包含 V_p 中所有元素的带权重的边。权重这样定义：对所有的 i 和 j， $w(p_i,\bar{p}_j)=1/|p_i-\bar{p}_j|$， $w(p_i,p_j)=0$， $w(\bar{p}_i,\bar{p}_j)=0$ 。图 5-2 (a) 展示了 $\mathcal{G}_p$ 的一个例子。我们这里需要解决的是图 $\mathcal{G}_p$ 上的最小割问题。

作为一个迭代算法，第一步是做一个初始割，将 V_p 分成两个子集 V_{p1} 和 V_{p2}。定义割的权值就是割集的权重的和。图 5-2 (b)给出了一个例子，其中初始割生成了两个子集 $V_{p1}=\{P_a,P_b,P_c,\bar{P}_a,\bar{P}_b,\bar{P}_c\}$ 和 $V_{p2}=\{P_d,P_e,\bar{P}_d,\bar{P}_e\}$。

然后，为了将正确的元素从一个子集移到另一个子集进行迭代，我们需要测定一个移动操作导致的割的权值变化。对每一个元素 v，我们首先将元素 v 和相同子集中元素的边集定义为 $I(v)$，类似地，将元素 v 和不同子集中的元素的边集定义为 $E(v)$ 。下面将图 5-2 (b)中的元素 P_a 做一个例子。

$$\begin{aligned} I(P_a)&=\{(P_a,\bar{P}_a),(P_a,\bar{P}_b),(P_a,\bar{P}_c)\}\\ E(P_a)&=\{(P_a,\bar{P}_d),(P_a,\bar{P}_e,\} \end{aligned} \tag{5-1}$$

注意这里我们将权重为 0 的边忽略掉，例如 (P_a,P_b) 等。下面是增益函数 $f(v)$。

$$f(v)=\sum_{n_i\in E(v)}w(n_i)-\sum_{n_j\in I(v)}w(n_j) \tag{5-2}$$

$f(v)$ 测定的是如果 v 移动到另一个子集，割的权值产生的减少量。在这个例子中 $f(P_a)$ 是这样计算的， $f(P_a)=(1/|P_a-\bar{P}_d|+1/|P_a-\bar{P}_e|)-(1/|P_a-\bar{P}_a|+1/|P_a-\bar{P}_b|+1/|P_a-\bar{P}_c|)=-1.40$。 $f(P_a)$ 是负值表示移动 $f(P_a)$ 到另一个子集的操作会增加割的权值，这就表示 $f(P_a)$ 不应该被移动。类似地，其他元素的增益也

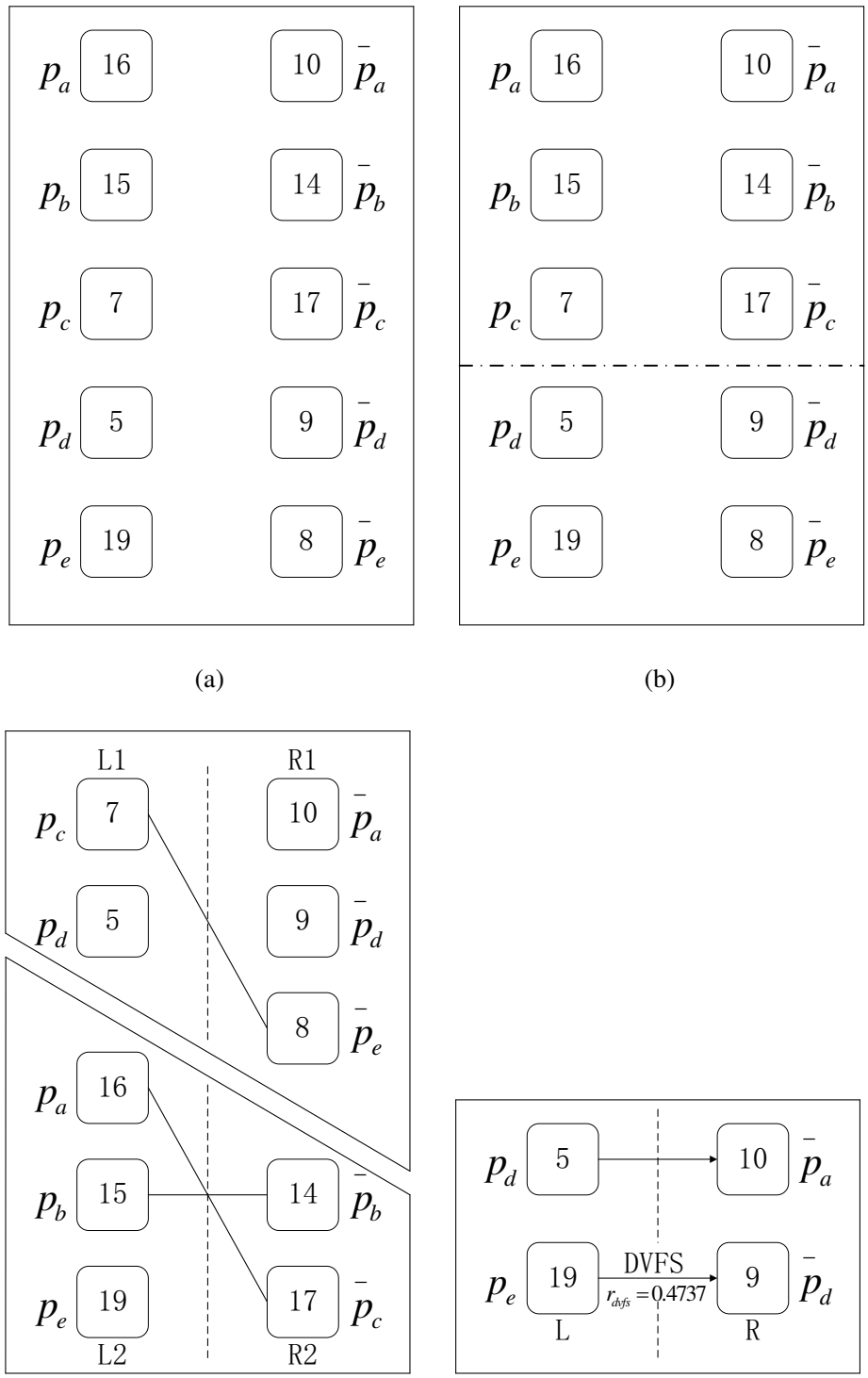


图 5-2 分层算法在100核微处理器上的高层处理过程示例

(a)图 $\mathcal{G}_p$ （为简化并未表示出边和权重）；(b)$\mathcal{G}_p$上的初始割（改进迭代最小割算法的第一步）；(c)$\mathcal{G}_p$上的最小割，每个高层块中的二部图匹配；(d)对未匹配的功耗用DVFS做最后调整

都被计算了， $f(P_b)=-1.39$， $f(P_c)=0.92$， $f(P_d)=-0.19$， $f(P_e)=0.62$，$f(\bar{P}_a)=-0.39$， $f(\bar{P}_b)=-1.33$， $f(\bar{P}_c)=-1.02$， $f(\bar{P}_d)=0.46$， $f(\bar{P}_e)=0.84$。对第 i 次迭代，最合适的元素记作 v_i 。 v_i 移动过去之后，将被锁定在那个子集中，不能再移动出来。它相邻的元素的增益都要更新。在这个例子中，最合适的元素是 P_c ， P_c 有最大的增益 0.92 ，它将被移动到下面的子集然后锁定。在这个简单的例子中，我们可以很容易从图中验证，将 P_c 从上面的子集移动到下面的子集，增强了功耗匹配质量： P_c 在下面子集中的 $\bar{P}_d$ 和 $\bar{P}_e$ 有一个更好的匹配候选，比上面子集的任何一个功耗元素都要好。然而这里有一个问题，在这个例子中如果我们直接执行这个移动操作，每次迭代中都移动最合适的元素可能会导致一个极端不平衡的结果：一个自己含有很多元素（功耗），另一个子集几乎没有元素。对我们的方法这个会引起问题，因为如果最小割之后，其中一个块仍然很大，那这个块执行二部图匹配会产生很大的延迟。所以我们在迭代最小割算法中引入一个平衡阈值：如果将合适的元素从 A 移动到 B 会超出阈值，那它将不被移动。相反的从 B 中选择最合适的元素移动到 A 并且锁定。所有元素都被锁定之后，生成了一个序列， $\mathcal{F}=\{f(v_1),f(v_2),\cdots,f(v_{2m})\}$ 。在这个阶段，我们已经移动了所有的元素到对应的另一个子集，这看起来似乎很奇怪，但这不是没有意义的。实际的情况是，所有的移动操作目的都是为了分析，来确定哪些元素要实际移动，下面步骤继续说明。

在序列 $\mathcal{F}$ 中，假设所有前面的元素 $v_1,v_2,\ldots,v_{i-1}$ 的移动操作都已经完成，$f(v_i)$ 表示 v_i 的单步移动操作的增益（割权值的减少）。所以，为了得出移动 $v_1,v_2,\ldots,v_i$ 的总增益（记作 $\tilde{f}(v_i)$ ），我们需要取和：

$$\tilde{f}(v_i)=\sum_{j=1}^{i}f(v_j) \tag{5-3}$$

对 $i=1,2,\ldots,2m$ 分别计算 $\tilde{f}(v_i)$ ，我们形成一个累积和序列

$$\tilde{\mathcal{F}}=\{\tilde{f}(v_1),\tilde{f}(v_2),\cdots,\tilde{f}(v_{2m})\} \tag{5-4}$$

其中 $\tilde{f}(v_i)$ 是前面讨论过的元素 v_1 到 v_i 移动的总增益。假设 $\tilde{f}(v_k)$ 是 $\tilde{F}$ 中的最大元素，这意味着移动 $v_1,v_2,\ldots,v_k$ 可以得到最大的增益（即最大减小割权值）。这样，我们就执行实际的操作，移动 $\{v_1,v_2,\cdots,v_k\}$ 去它对应的另一个子集。所有上面的程序记为一个移动动作。

尽管移动动作可以重复执行，但是以前在电路分割上的研究已经表明2到4次的移动动作已经足够实现最小化 [45, 46] 。对我们这个情况，实验表明执行一次移动动作已经足够了。

最小割之后，我们将所有剩余功耗分割成块。然后二部图匹配可以在每块内部执行。因为最小割已经将相关的功耗划到了同一块中，执行二部图匹配之后剩下的未匹配的功耗在其他块中也很难找到匹配了。所以，不在执行更进一步的二部图匹配了，我们可以由之前的低层和高层的二部图匹配结果直接确定任务迁移操作。最后没有匹配上的功耗，可以简单的用 DVFS 进行处理。

图 5-2 (a)， (b) 和(c) 展示了高层的匹配的一个例子。在图 5-2 (a) 中，低层匹配（图 5-1 (b) 表示已表示出）之后未匹配上的功耗全部收集形成图 $\mathcal{G}_p$ 。注意，图 $\mathcal{G}_p$ 中所有的元素都有带权重的边连接，图 5-2 中为简化并没有表示出。图 $\mathcal{G}_p$ 的初始割由图 5-2 (b) 上的虚线表示。然后执行迭代最小割算法，图 5-2 (c) 表示 $\mathcal{G}_p$ 的最终割， $\mathcal{G}_p$ 中的功耗元素被分割成了两个块，然后，每个新块的形成一个二部图，高层二部图匹配在每个新块执行。

5.3 DVFS最终调整

经过前面二部图匹配算法后，留下的还未匹配上的负载功耗引入 DVFS 方法来做最后调整。任务迁移根据低层和高层的二部图匹配结果进行操作，已经匹配上的负载功耗就迁移到了正确的核上，这些就符合 MPC 预测出的功耗分布（在这些位置上功耗可能会有些小的差值）。迁移操作过后，那些没有匹配上的负载功耗，就被迁移到一个新的位置，因为它们原来的位置可能被匹配的那一个占据了。

假设一个功耗为 p_i 未匹配上的负载被迁移到核 j 上，MPC预测核 j 需要的功耗为 $\bar{p}_j$ 。 p_i 与 $\bar{p}_j$ 并不匹配，它们并不相等。如果 $p_i < \bar{p}_j$ ，表明如果我们保持现在的状态，核 j 的温度将会低于我们设定的顶温度。因为这个并不影响芯片的可靠性，我们可以保持这个功耗不变。对另外一种情况 $p_i > \bar{p}_j$ ，我们在该核上执行DVFS操作，使功耗变化比率为 $r_{dvfs} = \bar{p}_j / p_i$ 。这是在没有超过温度限制的情况下，核 j 可以实现的最大性能。这里也要注意，DVFS 进行的是离散的电压频率调整，所以在实际的应用中， r_{dvfs} 被确定在低于 $\bar{p}_j / p_i$ 的最近水平上。

图 5-2 (d)为 100 核微处理器示例的最后调整步骤。 $p_d < \bar{p}_a$ ，所以没有 DVFS，p_d 仅被简单的移动到了新的位置上。因为 $p_e > \bar{p}_d$ ，所以 DVFS 在 p_e 上有操作。

我们知道在 DVFS 方法中，DC-DC 转换器是用来调整电压水平的，这个会在芯片设计上引入开销。在众核处理器上这是一个严重的问题，尤其是对于每个核都可以进行 DVFS 的情况。为了减小 DC-DC 转换器实现上的开销，引入 DVFS 块是一种很实用的方法，这个是在空间上相邻的几个核共用一个 DC-DC 转换器。在这种情况下， DVFS 的决策就要依据对应块上几个核之中的最低电压水平，会

影响吞吐量和性能。这个折中是众核架构中一个普遍也很重要的问题，需要在未来的工作上做进一步的研究。

这里总结一下这个新的分层动态温度管理方法，从利用 MPC 确定动态温度管理策略到执行温度管理操作整个流程总结如下：

1.用模型预测控制方法根据设定的温度限制求期望的功耗分布$\bar{P}(k)$。

2.划分相邻核成为低层块。

3.对每一个块，用对应的$P(k)$， $\bar{P}(k)$，和阈值 e_{th}建立这个块自己的二部图 $\mathcal{G} = (L, R, E)$ 。

4.在每个低层块上执行二部图匹配，记录匹配对。

5.收集搜有低层块未匹配上的功耗，建立一个新图 $\mathcal{G}_p = (V_p, E_p)$ ，为最小割做准备。

6.用改进的迭代最小割方法分割图$\mathcal{G}_p$生成高层块。

7.在第6步生成的每一个块上执行二部图匹配，记录匹配对。

8.根据记录的低层和高层的匹配对确定所有负载的新位置。

9.对仍然没有匹配上的负载，按5.3节所述执行DVFS操作。

5.4 本章小结

上一章说到用期望功耗与现有功耗构成的任务分配问题，在扩展到众核系统时需要超出可以忍受的计算时间，会使算法失效。本章主要解决这个问题，解决方法是将众核系统整个芯片上的所有核划分成多块，各自分别计算。这样就能极大降低算法的计算时间。具体实现是将划分分成两个层次，低层划分按照物理位置划分，高层划分按照功耗元素关系划分。这样的划分在一定程度上使计算并行化，极大降低计算时间，使算法更加有效。解决了任务分配问题后，还剩下没有匹配上的任务，由 DVFS 来做最后的处理，保证芯片在安全温度以下。

第六章 新的动态热管理方法的实现和结果比较

前面的章节已经介绍了众核处理器动态温度管理方法的基础知识，也详细说明了新提出的动态温度管理方法。接下来就要实现新的算法，并验证该算法比其他方法更有优势。 6.1 主要介绍算法实现的详细细节。 6.2 说明第四章描述的未分层的动态温度管理方法的实验结果及比较。 6.3 说明第五章描述的分层方法的实验结果及比较。

6.1 新的动态热管理方法的实现

本论文中新的动态温度管理方法的实现是在一个具有两个 8 核 16 线程 CPU 的 linux 服务器上进行的，每个 CPU 主频为 2.90GHZ，服务器内存 64GB。新的动态热管理方法主要用 MATLAB 实现，主要分为两部分，第一部分为第四章描述的未分层方法，和第二部分第五章描述的分层方法。对未分层方法我们构建了四个不同核数量的处理器，从 16 核（4×4）到 49 核（7×7）。针对分层方法我们也构建了四个不同处理器核数量配置的众核处理器，从 100 核（10×10）到 625 核（25×25）。这些处理器的热模型由 HotSpot 生成，HotSpot 已经有完整的工具可以直接调用。环境温度设定为 20°C。在这个实验中众核处理器由完全一致的Alpha 21264核组成。所有芯片的大小都是 $10mm\times10mm\times0.15mm$。这里我们假设众核处理器中任务运行时相互之间没有通讯也不需要同步。任务的功耗由 Wattch 通过运行 SPEC 基准程序 [47] 生成，初始任务分配为，一个任务随机指定给一个核运行。接下来的任务分配和调度有动态温度管理方法确定。我们有 9 个 SPEC 基准程序的实时功耗信息。对于不同的处理器的功耗信息，我们重复这 9 个功耗信息得到处理器需要的 16 个， 25个实时功耗信息等等。

因为核的大小会随核的数量增长变化，这会超过所谓的“功耗墙”或者“利用率墙”导致不现实的功耗密度 [48] ，产生极高的温度。为解决这个问题，提出了很多解决方法。一个解决方案是灰硅，放缩每个核的功耗 [48, 49]。另一个更广泛的方法是完全关掉一些核[48, 50]。在我们的研究中，我们采用灰硅技术，放缩功耗的大小以保证所有处理器都有相似的功耗密度，这样温度分布就类似于现在的多核芯片。这个可以通过以一定的比例调整操作频率和电压来实现，放缩比例在表 6-1 中给出。在我们的这个研究中，我们并不考虑完全关掉一部分核的策略，这会是我们将来的研究方向。

表 6-1 对不同核数处理器在新分层方法的参数

处理器核数	放缩比例	e_{th}	r
16 核 (4×4)	1.5	1.0	20
25 核 (5×5)	0.85	0.4	50
36 核 (6×6)	0.58	0.1	50
49 核 (7×7)	0.48	0.1	50
100 核 (10×10)	0.21	0.06	500
256 核 (16×16)	0.08	0.05	2100
400 核 (20×20)	0.052	0.04	3000
625 核 (25×25)	0.033	0.03	3000

对于不同核数的处理器，为保证温度能有效地趋近顶温度，任务迁移过程中的阈值 e_{th} 和式 (4-7) 中 MPC 调整参数 r 需要手动调整。这里注意，最优的 e_{th} 和 r 值与微处理器核的数量和结构高度相关，理论上没有必要去计算这些参数在实际的。在实际的应用中，针对一个确定的众核微处理器，很容易通过实验来调整这些参数。表 6-1 给出了这些参数的值。我们可以看出 e_{th} 随功耗模型的大小（在我们的情况中也是核的大小）变化。如果核的大小相对较大， e_{th} 也需要指定一个相对较大的值（即表 6-1 中 16 核的情况）。反之亦然（即表 6-1 中 625 核的情况）。这是因为较大的功耗有较大的芯片空间供功耗分布，所以对相同的温度容忍限制，允许有更大的功耗差值。另一个现象是核数越大，需要较大的 r 值。这是因为在式 (4-7) 中，当核数增长的时候 $Y_{ceil}-Y_k$ 中的每个元素并没有变化太大，但是 ΔP_k 中的每个元素会变得小很多（因为每个核的功耗缩小）。为了使 ΔP_k 有更小的解， r 的值需要变大。这里注意625核的情况 r 值仍然是 3000，与 400 核的情况相同，因为我们发现到这个程度再继续增大 r 值没有明显的影响。

对于分层方法，在低层划分上我们划分每 25 (5×5) 个相邻核为一块。在高层处理中，如果图 $\mathcal{G}_p$ 中的元素数目超过 240 就用改进迭代最小割算法进行分割。

为了最小化任务迁移的开销，任务迁移和 DVFS 的启动周期设定为 20s。任务迁移的开销来自于计算核迁移操作。通常处理器核数较小时，用较小的迁移周期。因为核数少时，计算开销和迁移操作开销（关系到核与核之间的通信）都很小。对中核处理器情况，因为核数很大，频繁的任务迁移是很难执行的，所以迁移周期延长。迁移间隔内一个核的负载可能会增长很多，这样可能引起超出温度限制。在这种情况下，我们在迁移间隔内需要保证安全的时候只能执行 DVFS。

在实验中，除了任务迁移决策计算的开销，任务迁移操作的开销也要考虑进去。正常的任务迁移操纵开销大概是 10^6 个时钟周期，对 1 GHZ的处理器大概是1ms [51]。因为在众核处理器中核数很大，核之间通信时间很长，这样的开销会很大。所以我们设定上百核处理器的任务迁移时间为 100ms。在实验中我们要考虑任务迁移的功耗。这个功耗由迁移时间内核的通信产生，在迁移时对应的两个核不处理任何任务。所以我们假设任务迁移时间的功耗小于任务处理时间的功耗。为保证安全，我们将任务迁移时的功耗设定为之前处理任务时的平均功耗，用这样的功耗提供给MPC做预测。

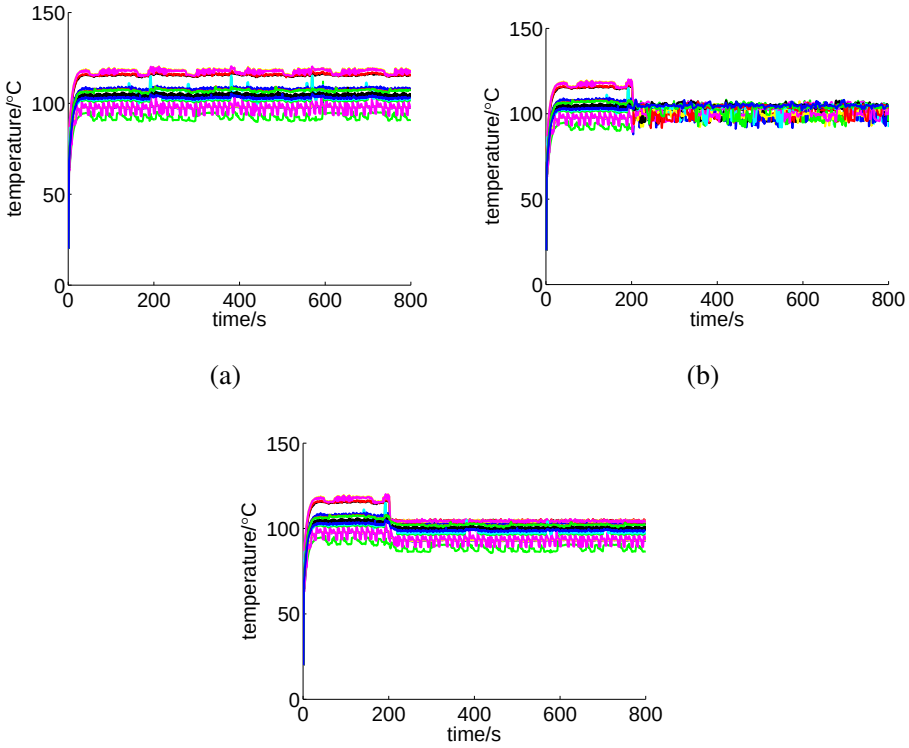


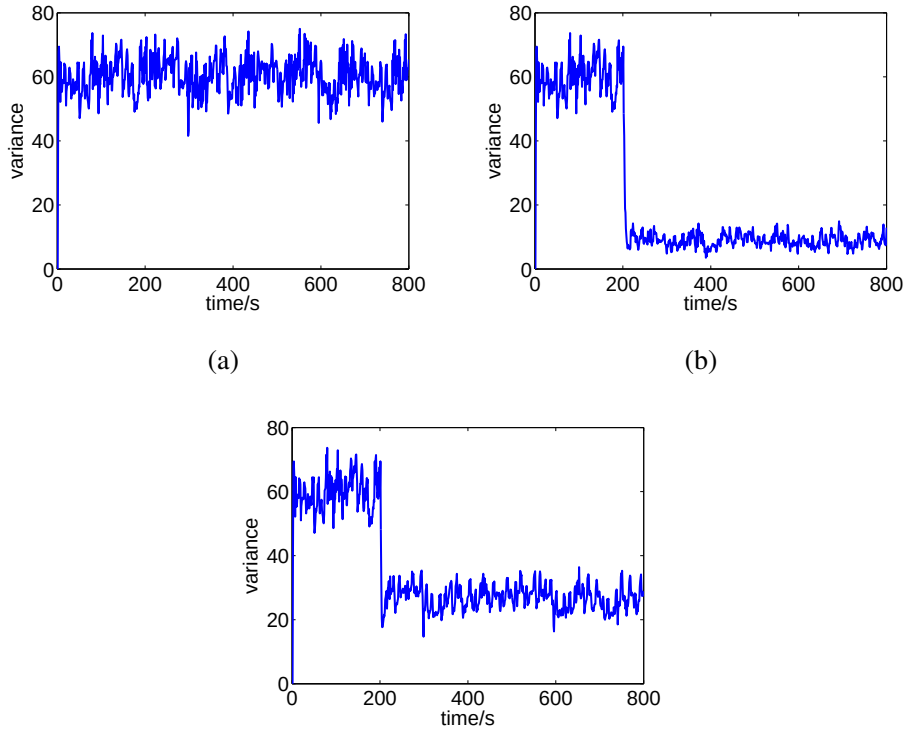
图 6-1 不同DTM方法下的16核CPU瞬态温度

(a)不采用任何DTM方法；(b)采用未分层DTM方法；(c)采用[25]中的DTM方法

作为未分层方法的比较，我们实现了另外一种基于MPC结合DVFS的方法 [25]。针对分层方法，我们既将其与未分层方法比较也与方法 [25]进行比较，还选择了[22]中的动态温度管理方法作为对比，它和我们的研究有相同的目标，即改性能处理器有温度限制时最大化性能（吞吐量）。我们在实验中实现了 [22] 的作者提供的开源项目 MAGMA V2。MAGMA只能给出100核处理器的结果，对于更大

的核数情况会有“超出内存”错误。在这里所有的方法都用相同的激活周期，功耗信息和顶温度。

6.2 未分层方法与其他方法的结果比较



(a)　　(b)

(c)

图 6-2 不同DTM方法下的16核CPU核间瞬态温度方差

(a)不采用任何DTM方法；(b)采用未分层DTM方法；(c)采用[25]中的DTM方法

首先我们实现16核处理器不采用温度管理时的瞬态温度测量,如图 6-1 (a)。核的温度大概在 90°C 到 120°C。温度在 120°C 左右会影响芯片的可靠性。其核之间温度的方差，如图 6-2 (a)。可以看出核之间温度方差很大。

接下来我们测试未分层方法，将安全顶温度设置为 105°C。温度管理在第200秒时被激活，激活周期为 1s。采用未分层方法后的瞬态温度如图 6-1 (b)。可以看出温度管理激活后所有核的温度都趋近于给定的 105°C 顶温度。我们可以注意到这里有一些尖峰超过顶温度，这些尖峰由温度管理周期内的快速改变的温度响应引起，所以这个不能避免，所有的温度管理方法都会出现这个问题。为了保证芯片的安全性，我们设置顶温度比实际安全温度稍微小一点。为方便分析我们也计算

了核之间的温度方差如图 6-2 (b) 所示，显然采用未分层方法后，核之间温度方差被极大降低了。

然后我们实现[25]中的动态温度管理方法。同样的，我们在第200秒时激活这个温度管理方法，为每个核设置同样的顶温度 105°C。 16核处理器对应的瞬态温度如图 6-1 (c)，对应的核间温度方差如图6-2 (c)。从图 6-1 (c) 可以看出，这个动态温度管理方法实际上非常有效，能够有效保证温度在顶温度以下（尖峰除外）。但是，通过比较图 6-1 (c) 和图 6-1 (b)， [25]中的动态温度管理算法很明显导致了一个较大的核间温度方差，我们的新方法在这方面要好得多。这是因为 [25] 只用了DVFS，有些核的温度不调整电压和频率也本来就在顶温度以下。在这种情况下，[25] 对这些核没有采取任何措施，所以导致一些核的温度在顶温度以下。这个表示，处理器没有在最高性能运行。

我们已经展示了两种动态温度管理方法在16核处理器上的详细性能比较。我们也在不同处理器核数不同激活周期也比较了两种方法。不同激活周期的核间温度方差的比较结果记录在表 6-2。其中 var_o 表示不采用动态温度管理状态下的平均方差， var_d 表示 [25]中的动态温度管理方法下的平均核间瞬态温度方差，var_n 表示我们的未分层方法下的平均核间瞬态温度方差。从这个表中我们可以看出两种动态温度管理方法都很大程度上降低了核间温度方差，也就是平衡和芯片温度。我们的未分层方法要明显比 [25] 中的方法好很多。更小的温度方差能够使芯片可靠性更好。激活周期延长方差也仅仅稍微增大一点。激活周期 40s 时，我们新方法的方差仍然明显小于方法 [25]。后面将会介绍，大的激活周期可以使计算开销更小。

除了可靠性方面的考虑，处理器的计算性能也是同等重要的。我们也比较了采用这两种方法的是处理器的性能。处理器性能可以由它的吞吐量来表示，就是单位时间任务的完成量。在这个实验中，我们通过测量核的平均每秒指令数（IPS）代表吞吐量来估计处理器性能。也就是更高的 IPS 表示有更高的性能。新

表 6-2 未分层方法与其他方法的核间温度方差比较

核数	var_o	周期 1s		周期 10s		周期 20s		周期 40s	
		var_d	var_n	var_d	var_n	var_d	var_n	var_d	var_n
16 核	60.6	27.0	9.2	25.5	11.1	25.5	11.5	25.3	13.2
25 核	79.0	29.8	15.5	31.2	12.5	31.1	12.8	30.9	16.1
36 核	66.0	27.7	11.4	26.8	11.5	25.5	12.9	25.6	12.8
49 核	60.4	25.7	6.4	23.1	6.7	22.9	6.8	23.2	7.8

表 6-3 未分层方法与其他方法的IPS比较

核数	$MIPS_o$	周期 1s		周期 10s		周期 20s		周期 40s	
		$MIPS_d$	$MIPS_n$	$MIPS_d$	$MIPS_n$	$MIPS_d$	$MIPS_n$	$MIPS_d$	$MIPS_n$
16 核	651.5	630.6	643.8	628.9	642.1	628.8	642.2	628.4	641.8
25 核	530.2	508.0	520.4	508.9	519.7	508.6	519.2	508.2	518.8
36 核	477.1	460.4	471.4	459.6	470.9	458.1	470.5	457.5	470.4
49 核	442.6	426.4	439.1	423.8	438.0	423.3	438.1	422.7	437.5

的未分层方法和[25]中方法的每核平均IPS 记录在表 6-3 中。 $MIPS_o$ 代表不采用动态温度管理时的核平均每秒百万指令数（MIPS）， $MIPS_d$ 表示 [25]中方法的核平均 MIPS， $MIPS_n$ 表示我们新的未分层方法的核平均 MIPS。可以看出，我们的新方法比 [25] 中的方法有很大提升。对理想状态性能$MIPS_o$显著减少。

最后我们来比较，测量了我们新方法在不同激活周期下的计算时间。由于未分层方法主要由 MPC计算和二部图匹配组成，我们将算法计算时间分成 MPC时间（t_p）和匹配时间（t_m）。这个时间记录在表 6-4 中。可以看出，MPC和匹配的计算时间随核数的增长而增加。MPC的计算时间比匹配要小很多。对激活周期是1s的情况，49核微处理器需要的匹配计算时间为0.04s，这个被看做一个比较大的延迟，但是考虑到这是49核中的一个核的计算，整体的吞吐量损耗又可以忽略。如果增加激活周期，计算时间将会减少，伴随着一点可靠性的损失。即其温度方差会增加（比对表 6-2 和表 6-3）。 [25]中的方法，执行时间只有MPC时间（t_p）。我们的新方法虽然计算时间比 [25] 要稍微多一点，但是整体来说，在性能和可靠性上还是有很大提升的。

总体来说，未分层方法比较适合于核数不是太多的多核微处理器，对于核数超过100的众核处理器超出了未分层方法的能力。因为匹配算法复杂度较高，随核数增加计算时间会显著增加。在核数较少时并不明显，当处理器核数超过100时将会显著增加，后面会做介绍。下面的分层方法主要来进行众核方面的比较。

表 6-4 未分层方法与其他方法的计算时间比较

核数	周期1s		周期10s		周期20s		周期40s	
	$t_p(10^{-3}\text{s})$	$t_m(10^{-3}\text{s})$	$t_p(10^{-3}\text{s})$	$t_m(10^{-3}\text{s})$	$t_p(10^{-3}\text{s})$	$t_m(10^{-3}\text{s})$	$t_p(10^{-3}\text{s})$	$t_m(10^{-3}\text{s})$
16 核	0.29	4	0.031	0.3	0.019	0.1	0.010	0.10
25 核	0.38	8	0.053	0.7	0.035	0.3	0.022	0.16
36 核	0.50	19	0.063	1.7	0.044	0.9	0.032	0.43
49 核	0.61	41	0.078	4.1	0.059	1.9	0.041	0.83

6.3 分层方法与其他方法的结果比较

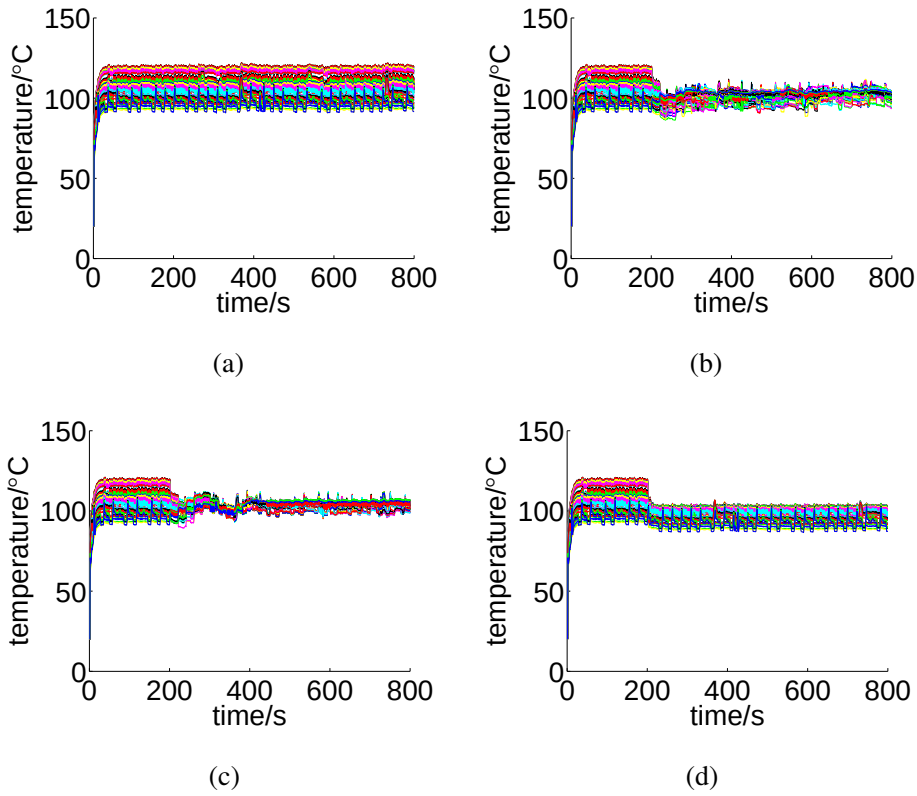


图 6-3 不同DTM方法下的100核CPU瞬态温度

(a)不采用任何DTM方法；(b)采用分层DTM方法；(c)采用未分层方法；(d)采用[25]中的DTM方法

首先我们不进行任何温度管理让100核处理器在最大速度上运行。图 6-3 (a)就是这个处理器的瞬态温度。我们可以看出，核的温度大概是从 90°C 到 120°C。我们也测量了温度的方差，见图 6-4 (a)。可以看出，没有任何温度管理的情况下，核之间的温度差也很大。

接下来我们把顶温度设定在 105°C，测试新的分层动态温度管理方法在100核处理器上的效果。分层动态温度管理方法在第200秒时被激活，激活周期为 20s。对应的瞬态温度就是图 6-3 (b)，对应的温度方差就是图 6-4 (b)。分层动态温度管理方法被激活之后，所有核的温度开始趋近于设定的顶温度 105°C，核之间的温度差也减小了很多。

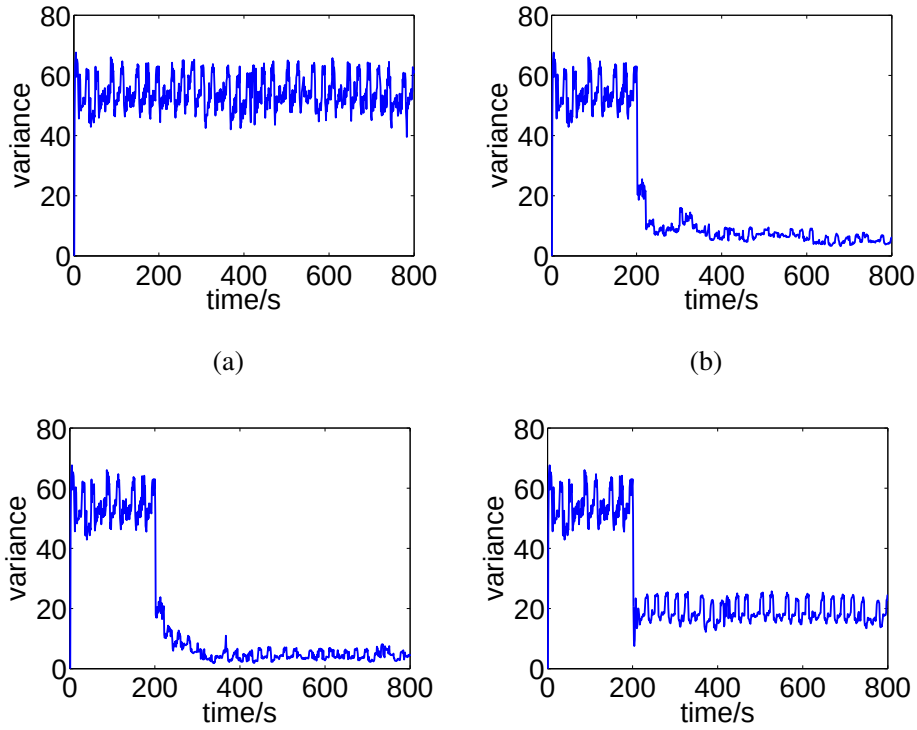


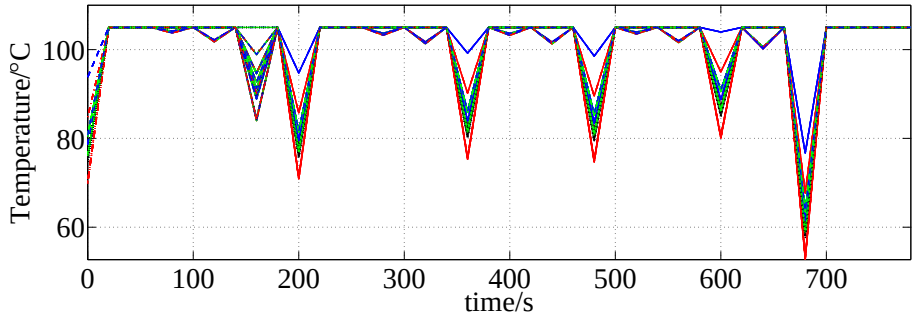
图 6-4 不同DTM方法下的100核CPU核间瞬态温度方差

(a)不采用任何DTM方法；(b)采用分层DTM方法；(c)采用未分层方法；(d)采用[25]中的DTM方法

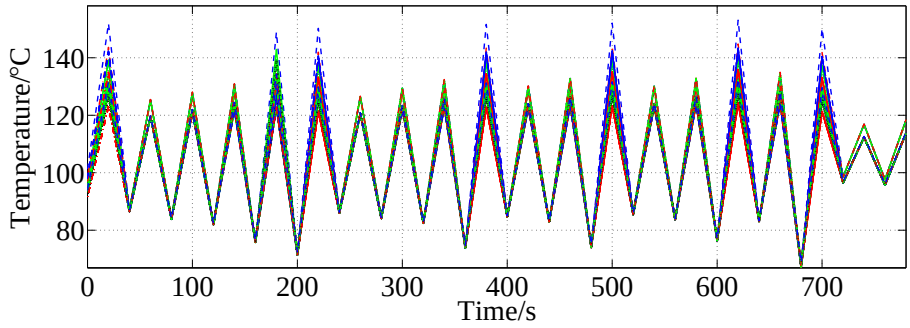
作为对比，我们的未分层方法和 [25] 中基于 MPC 结合 DVFS 的方法也进行了测试。顶温度同样被设定为 105°C，所有的方法仍然在相同的时间点被激活，即第 200 秒，激活周期仍然是 20s。图 6-3 (c) (d) 和图 6-4 (c) (d) 就是这两种方法对应的瞬态温度，和核之间温度方差。新的分层动态热管理方法和未分层方法有相似的瞬态温度，比未分层方法的核之间温度方差稍微大一点点，说明新方法并不能最优地平衡整个芯片的温度（后面会介绍，新分层方法在开销和扩展性上比未分层方法要好得多）。 [25] 中的动态温度管理方法在瞬态温度和核之间温度差上要比前两种方法差很多。

下面我们对比 [22] 中的方法。这个方法也是执行 DVFS 和任务迁移，来使核的温度趋近于顶温度。不过，这个方法为了减小计算开销，在做动态温度管理决策的时候假设每个核都核其他核热隔离。这种假设在核数较少的多核芯片上或许可以，因为核数较少时，核之间有大面积的缓存区域阻隔了核间热交换。但是在众核情况下，因为每个核面积较小，热交换很明显，并且不能被忽略。该方法的

100 核处理器温度即图 6-5 (a) 所示。因为这里的仿真步长较大，所以温度波形是直的线段。依据这样的温度来做动态温度管理决策，温度大部分时间都是趋近顶温度的。但是因为核与核之间的零热交换，所以这样的温度并不是实际的温度。我们修改了MAGMA 程序，画出了考虑核间热交换的实际温度，如图 6-5 (b) 所示。我们很清楚地看到这个方法的动态温度管理决策不是最优的，会明显的超出实际温度的限制。核与核之间的热交换对众核处理器的温度具有极大的影响，导致严重超出 105°C 的顶温度。从图 6-5 我们可以看出，温度控制有一个延时调整问题。这是因为在 [22] 的方法中，当当前温度并不完全等于顶温度的时候，核或者被关掉（如果当前温度高于顶温度），或者全速运行（如果当前温度低于顶温度）。但是在图 6-3 中，所有基于 MPC 的动态温度管理方法都没有这个问题，因为它们可以提前预测来做决策，这样就会是比较平滑的温度控制。



(a)



(b)

图 6-5 100核CPU执行[22]中的DTM方法的瞬态温度

(a)假设核间没有热交换的瞬态温度；(b)有核间热交换的真实瞬态温度

表 6-5 记录了不同核数的处理器的核之间温度方差。表中 var_o 是没用任何动态温度管理方法时核之间温度方差， var_d 是 [25] 中动态温度管理方法的核之间温度方差， var_n 是未分层方法的核之间温度方差， var_h 是新的分层动态温度管

理方法的核之间温度方差， var_g 是 [22] 中动态温度管理方法的核之间温度方差。不同核数的结果类似，未分层方法稍微优于我们新的分层动态温度管理方法，基于 MPC 只结合 DVFS 的方法明显有较大的核间温度方差。 [22] 中的方法只完成了 100 核的实验，其核之间温度方差与未分层方法接近。

表 6-5 分层方法与其他方法核间温度方差比较

核数	var_o	var_d	var_n	var_h	var_g
100 核 (10×10)	54.2	18.8	5.5	7.6	5.9
256 核 (16×16)	50.8	19.3	3.4	6.5	N/A
400 核 (20×20)	52.1	16.7	2.5	4.1	N/A
625 核 (25×25)	50.5	15.9	2.3	4.1	N/A

从前面的比较可以看出，我们新提出的分层动态温度管理方法与未分层方法在瞬态温度和核之间温度方差上的相差不大。在众核处理器上与未分层方法、[22]相比，我们的新方法真正的亮点是可扩展性和小开销。我们测定这几种算法的平均每秒执行时间。我们的新方法主要由 MPC，二部图匹配（任务迁移），最小割划分组成。为更好分析，我们将算法的总执行时间分为 MPC 时间 t_p ，匹配时间 t_m，最小割划分时间 t_a。匹配操作的执行时间一部分为低层匹配时间，另一部分为高层匹配时间。对于低层匹配时间，如果这里有多个匹配操作在并行执行，我们只计最长的一个，这个时间才是主导延时的时间。未分层方法也有 MPC 时间 t_p 和匹配时间 t_m ，但是没有最小割划分时间 t_a 。 [25] 中的方法只有 MPC 时间 t_p。尽管 [22] 中的方法只完成了100核的情况，但是我们还是可以测试它的任务迁移时间 t_m 。我们可以用正确维度的随机矩阵输入给对应的函数。表 6-6 记录了各算法在不同核数处理器上的执行时间。很显然随核数增长，未分层方法开销明显上升。从 400 核的情况开始，每秒的平均管理决策时间要超过1s，这是完

表 6-6 分层方法与其他方法的平均计算时间比较

	分层方法				未分层方法			[25]		[22]	
核数	t_p $(10^{-4}s)$	t_m $(10^{-4}s)$	t_a $(10^{-4}s)$	t_{all} $(10^{-4}s)$	t_p $(10^{-4}s)$	t_m (s)	t_{all} (s)	t_p $(10^{-4}s)$	t_{all} $(10^{-4}s)$	t_m (s)	t_{all} (s)
100	1.58	1.63	0	3.21	1.49	0.01	0.01	1.60	1.60	0.01	0.01
256	2.85	19.26	1.58	23.7	2.93	0.45	0.45	2.80	2.80	0.09	0.09
400	4.88	7.77	3.27	15.9	5.38	1.90	1.90	5.29	5.29	0.34	0.34
625	9.09	12.27	17.49	38.8	8.27	8.63	8.63	8.87	8.87	0.99	0.99

全不能接受的。[22] 中的方法在任务迁移计算上的时间也随核数很快增长，使它不能扩展到众核应用中。根据这篇论文，我们找出任务迁移算法的时间复杂度为$O(nq)^3$（跟未分层方法的任务迁移算法接近），其中n是核数，q是任务数量。这篇论文中假设 $n = q$，那么当核数增长额时候，$O(n^6)$ 的复杂度使任务迁移占用了太多的时间。相对地，我们新的分层算法的计算时间增长缓慢。即使是625核的情况（这已经是很大的核数了），我们的方法平均每秒消耗 4 ms 做管理决策。这仅仅是 0.4% 的计算时间花费在少数几个核上，因此这个可以被忽略。当然，[25] 中的方法需要最少的时间计算开销，但是我们的新方法只在任务迁移上花费很少的开销就使性能显著提成，下面将继续介绍。

最后我们测定不同众核处理器采用各种动态温度管理方法的性能。每秒执行的指令数（IPS）是衡量处理器性能一个标准。因为未分层方法和 [22] 中的方法有很大的开销，不能完成众核处理器的管理决策，所以在性能对比中并没有考虑这两个方法。表 6-7 记录了采用新分层方法和仅采用DVFS的方法时的平均 IPS。在表中，$MIPS_o$ 表示不采用任何动态温度管理方法时单个处理器核的平均IPS（以百万计就是MIPS），$MIPS_d$ 表示采用 [25] 中方法时单个处理器核的平均IPS，$MIPS_h$ 表示采用我们的新分层方法时单个处理器核的平均IPS。$MIPS_o$ 是理想性能，是在处理器没有任何温度限制的情况下取得的。$MIPS_h$ 只比 $MIPS_o$ 小一点，说明在确定管理决策时，我们新的分层算法是很有效的。从表中可以看出，新的分层算法在性能上优于 [25] 中的算法。尽管新算法有稍微大一点的开销，但是消耗在任务迁移决策的时间减少了 DVFS 激活的次数，带来的是性能收益。我们注意到，新方法相比于 [25] 中的方法在吞吐量上的提升和运行的应用有很大关系。比如，如果这里有很低温度的核（甚至是闲置无任务的核），新方法就会比 [25] 中方法有更明显的吞吐量提升，因为低温核可以被任务迁移充分利用，减少DVFS操作。

表 6-7 分层方法与其他方法的每核平均IPS比较

核数	$MIPS_o$	$MIPS_d$	$MIPS_h$
100 核 (10×10)	290.8	279.6	281.5
256 核 (16×16)	210.2	202.9	207.1
400 核 (20×20)	182.1	174.7	178.8
625 核 (25×25)	156.5	150.2	154.4

6.4 本章小结

本章实现了新提出了未分层和分层动态温度管理方法，并且实现了与其他方法来做比较。通过比较，未分层方法能最优地平衡芯片温度，提升处理器性能，在核数较少的多核芯片上有很大优势，但是在扩展性上并不如分层方法。虽然在瞬态温度方差上新的分层方法并不是最好的结果，并不能最优地平衡芯片温度，但是也与未分层方法相差不大。在众核系统中新的分层方法与未分层方法、[22]相比是在可扩展性和小开销上具有巨大的优势。在计算开销上新的分层算法要比 [25] 中的差，但是新的方法提升了处理器性能。所以总体来说，新的动态温度管理方法比其他方法有很大优势。

第七章 总结

近年来，处理器技术持续发展，已经由多核发展到众核领域。随着众核处理器带来的性能提升，也带来了一定的负面影响。尤其是功耗密度增加，出现了热问题。不仅有高温问题，众核处理工作负载不均衡，导致处理器温度梯度加大，极可能出现局部高温点。高温和高的温度梯度都对处理器有极大的影响，会导致处理器性能和可靠性下降，影响芯片寿命，冷却成本也急剧上升。针对这一问题，本文做了动态温度管理方面的研究，提出了一种针对高性能众核系统的新的分层动态温度管理方法。动态温度管理方法通过调整任务的执行改变功耗特性，平衡芯片负载，降低峰值温度和局部高温点数量，能有效改善热问题。

本文首先分析了芯片温度不受管理对芯片的负面影响，温度过高对芯片可靠性，给芯片性能带来极大损害。过高的温度也会加剧静态功耗增加，静态功耗的增加反过来又会加剧温度上升。现在的冷却系统对于处理现有的高温问题已经相当困难，需要极高的冷却成本。对于动态温度管理方法，虽然是对任务或者处理器进行调整来修正功耗特性，但是这不仅仅与功耗相关，与芯片封装和散热参数等等都有很大的关联。现今动态温度管理方法最常用的操作动态电压频率调整和任务迁移技术，在修正功耗或功耗分布方面很有作用。对这些方法如何能调整修正温度也进行了详细说明。仅仅有动态温度管理操作是不能够有效控制芯片温度的，这些操作需要一个良好的引导控制。所以引导控制需要的温度预测方法和模型预测控制方法也在这里进行了介绍。对于众核处理器进行模型预测控制，首先需要对芯片进行热建模。 HotSpot 是一种简洁精确的热建模方法。热系统与电路系统有相似的对偶关系，对热传导理论进行说明。为方便理解处理器热模型，对 HotSpot 热建模方法进行了介绍，并讨论了热模型在动态温度管理方法中的作用。

本文提出了针对高性能多核众核处理器的动态热管理算法，对其进行了详细介绍，新方法基于模型预测控制，结合了任务迁移和动态电压频率调整技术，能有效降低性能损耗，提高芯片可靠性。做任务迁移决策时用到的复杂度较高的匈牙利算法做二部图匹配，直接扩展到众核系统需要的计算开销很大。为了扩展到众核系统，在两个层次上做基于二部图匹配的任务迁移决策：块内的低层，块间的高层。改进的最小割算法用于辅助高层的任务迁移决策过程。在实验中建模实现该算法，并在不同核数的众核处理器上验证测试，并与其它动态温度管理方法进行比较。新方法能够把芯片温度控制在安全范围内，能有效解决众核芯片热问题，并且能使处理器得到更高的计算性能，比现有的方法具有很大优势。

最后本文方法也有一定的局限性，在仿真实现中做了一些理想化的假设，比如动态电压频率调整时，可以将频率和电压连续地调整到任意水平。实际可以调整频率电源电压的处理器一般只设定几个离散的频率水平。所以这方面需要在未来的工作中改进。还有文中提到的 DVFS 要用到的 DC-DC 转换器，为减少转换器的开销，可以考虑几个核共用一个转换器，这样就要改进 DVFS 策略。同样的这部分改进也需要在未来工作中做进一步研究。

致 谢

马上就要毕业，在研究生学习期间，尤其是最近学位论文写作期间，老师和同学们给予了很多帮助和意见。在这里尤其要感谢我的指导老师-王海老师。王老师对该领域认识理解深刻透彻，为人和蔼，治学严谨。跟随王老师我得到了科研水平的提升。在论文撰写中，王老师也给予了深刻的指导，提出了宝贵的意见。

另外也要感谢教研室全体同学，在平时研究学习中给予学术上的帮助，论文撰写过程中也给予了我很大的帮助，提出了很多改进与修改意见。

在此向帮助和指导过我的老师和同学表示最衷心的感谢。

参考文献

[1] S. Borkar. Thousand core chips: a technology perspective[C]. Proceedings of the 44th annual Design Automation Conference, 2007, 746–749

[2] R. McGowen, C. A. Poirier, C. Bostak, et al. Power and temperature control on a 90-nm Itanium family processor[J]. IEEE Journal of Solid-State Circuits, 2006, 41(1):229–237

[3] D. Brooks, R. Dick, R. Joseph, et al. Power, Thermal, and Reliability Modeling in Nanometer-Scale Microprocessors[J]. IEEE Micro, 2007, 27(3):49–62

[4] J. Donald, M. Martonosi. Techniques for Multicore Thermal Management: Classification and New Exploration[C]. Proceedings of International Symposium on Computer Architecture (ISCA), Boston, MA, USA, 2006, 78–88

[5] M. Powell, M. Gomaa, T. Vijaykumar. Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the Operating System[C]. Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2004, 260–270

[6] Y. Ge, P. Malani, Q. Qiu. Distributed task migration for thermal management in many-core systems[C]. Proceedings of Design Automation Conference (DAC), San Diego, CA, USA, 2010, 579–584

[7] T. Chantem, S. Hu, R. Dick. Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2011, 19(10):1884–1897

[8] G. Liu, M. Fan, G. Quan. Neigbor-Aware Dynamic Thermal Management for Multi-core Platform[C]. Proceedings of European Design and Test Conference (DATE), Dresden, Germany, 2012, 187–192

[9] R. Ayoub, T. Rosing. Predict and Act: Dynamic Thermal Management for Multi-Core Processor[C]. Proceedings of International Symposium on Low Power Electronics and Design (ISLPED), San Francisco, CA, USA, 2009, 99–104

[10] T. Ebi, M. Al Faruque, J. Henkel. TAPE: Thermal-aware agent-based power economy for multi/many-core architectures[C]. Proceedings of International Conference on Computer Aided Design (ICCAD), San Jose, CA, USA, 2009, 302–309

[11] J. Cong, B. Yuan. Energy-Efficient Scheduling on Heterogeneous Multi-Core Architectures[C]. Proceedings of International Symposium on Low Power Electronics and Design (ISLPED), 2012, 345–350

[12] K. Skadron, M. Stan, W. Huang, et al. Temperature-Aware Microarchitecture[C]. Proceedings of International Symposium on Computer Architecture (ISCA), 2003, 2–13

[13] R. Jayaseelan, T. Mitra. A hybrid local-global approach for multi-core thermal management[C]. Proceedings of International Conference on Computer Aided Design (ICCAD), San Jose, CA, USA, 2009, 314–320

[14] A. Mutapcic, S. Boyd, S. Murali, et al. Processor Speed Control with Thermal Constraints[J]. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 2009, 56(9):1994–2007

[15] H. Khdr, S. Pagani, M. Shafique, et al. Thermal Constrained Resource Management for Mixed ILP-TLP Workloads in Dark Silicon Chips[C]. Proceedings of Design Automation Conference (DAC), 2015, 1–6

[16] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, et al. Hierarchical Power Management for Asymmetric Multi-Core in Dark Silicon Era[C]. Proceedings of Design Automation Conference (DAC), 2013, 1–9

[17] M. Kadin, S. Reda, A. Uht. Central versus Distributed Dynamic Thermal Management for Multi-Core Processors: Which One Is Better?[C]. Proceedings of IEEE/ACM International Great Lakes Symposium on VLSI (GLSVLSI), 2009, 137–140

[18] A. Bartolini, M. Cacciari, A. Tilli, et al. Thermal and Energy Management of High-Performance Multicores: Distributed and Self-Calibrating Model-Predictive Controller[J]. IEEE Transactions on Parallel and Distributed Systems, 2013, 24(1):170–183

[19] D. Brooks, M. Martonosi. Dynamic thermal management for high-performance microprocessors[C]. Proceedings IEEE International Symposium on High-Performance Computer Architecture (HPCA), Monterrey, Mexico, 2001, 171–182

[20] I. Yeo, C. C. Liu, E. J. Kim. Predictive dynamic thermal management for multicore systems[C]. Proceedings of the 45th annual Design Automation Conference, 2008, 734–739

[21] A. K. Coskun, T. S. Rosing, K. G. C. Gross. Proactive temperature balancing for low cost thermal management in MPSoCs[C]. Proceedings of International Conference on Computer Aided Design (ICCAD), 2008, 250–257

[22] V. Hanumaiah, S. Vrudhula, K. Chatha. Performance optimal online DVFS and task migration techniques for thermally constrained multi-core processors[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2011, 30(11):1677–1690

[23] V. Hanumaiah, S. Vrudhula. Energy-efficient operation of multicore processors by DVFS, task migration, and active cooling[J]. IEEE Transactions on Computers, 2014, 63(2):349–360

[24] C. Tan, T. Muthukaruppan, T. Mitra, et al. Approximation-Aware Scheduling on Heterogeneous Multi-core Architectures[C]. Proceedings of Asia South Pacific Design Automation Conference (ASPDAC), 2015, 618–623

[25] F. Zanini, D. Atienza, L. Benini, et al. Multicore Thermal Management with Model Predictive Control[C]. Proc. 19th European Conference on Cicuit Theory and Design, Antalya, Turkey, 2009, 90–95

[26] Y. Wang, K. Ma, X. Wang. Temperature-Constrained Power Control for Chip Multiprocessors with Online Model Estimation[C]. Proceedings of International Symposium on Computer Architecture (ISCA), 2009, 314–324

[27] K. Skadron, M. R. Stan, K. Sankaranarayanan, et al. Temperature-aware microarchitecture: Modeling and implementation[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2004, 1(1):94–125

[28] C. Lefurgy, K. Rajamani, F. Rawson, et al. Energy management for commercial servers[J]. Computer, 2003, 36(12):39–48

[29] R. Ayoub, S. Sharifi, T. S. Rosing. Gentlecool: Cooling aware proactive workload scheduling in multi-machine systems[C]. Proceedings of the Conference on Design, Automation and Test in Europe, 2010, 295–298

[30] J. S. S. T. Association, et al. Failure mechanisms and models for semiconductor devices[J]. JEDEC Publication JEP122-B, 2003

[31] W. Liao, L. He, K. M. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2005, 24(7):1042–1053

[32] E. Kursun, C.-Y. Cher, A. Buyuktosunoglu, et al. Investigating the effects of task scheduling on thermal behavior[C]. Third Workshop on Temperature-Aware Computer Systems (TACS’06), 2006

[33] Y. Liu, R. P. Dick, L. Shang, et al. Accurate temperature-dependent integrated circuit leakage power estimation is easy[C]. Proceedings of the conference on Design, automation and test in Europe, 2007, 1526–1531

[34] V. K. Pamula, K. Chakrabarty. Cooling of integrated circuits using droplet-based microfluidics[C]. Proceedings of the 13th ACM Great Lakes symposium on VLSI, 2003, 84–87

[35] E. Le Sueur, G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns[C]. Proceedings of the 2010 international conference on Power aware computing and systems, 2010, 1–8

[36] D. Suleiman, M. A. Ibrahim, I. Hamarash. Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction[C]. 4th International Conference on Electrical and Electronics Engineering, 2005

[37] L. Wang. Model Predictive Control System Design and Implementation Using MATLAB[M]. Springer, 2009

[38] A. Bemporad, M. Morari. Robust model predictive control: A survey[M]. Springer, 1999, 207–226

[39] W. Huang, S. Ghosh, S. Velusamy, et al. HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2006, 14(5):501–513

[40] S. Song, V. Au, K. P. Moran. Constriction/spreading resistance model for electronics packaging[C]. Proceedings of the 4th ASME/JSME thermal engineering joint conference, 1995, 199–206

[41] W. Huang, M. R. Stan, K. Skadron, et al. Compact thermal modeling for temperature-aware design[C]. Proceedings of the 41st annual Design Automation Conference, 2004, 878–883

[42] D. Brooks, V. Tiwari, M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations[C]. Proceedings of International Symposium on Computer Architecture (ISCA), Vancouver, BC, Canada, 2000, 83–94

[43] H. Wang, S. Tan, D. Li, et al. Composable Thermal Modeling and Simulation for Architecture-Level Thermal Designs of Multi-core Microprocessors[J]. ACM Transactions on Design Automation of Electronics Systems, 2013, 18(2):28:1–28:27

[44] J. Munkres. Algorithms for the assignment and transportation problems[J]. Journal of the Society for Industrial and Applied Mathematics, 1957, 5(1):32–38

[45] C. Fidducia, R. Mattheyses. A linear-time heuristic for improving network partitions[C]. Proceedings of Design Automation Conference (DAC), 1982, 175–181

[46] S. Dutt, W. Deng. A probability-based approach to VLSI circuit partitioning[C]. Proceedings of Design Automation Conference (DAC), 1996, 100–105

[47] J. L. Henning. SPEC CPU 2000: Measuring CPU Performance in the New Millennium[J]. IEEE computer, 2000, 1(7):28–35

[48] M. Taylor. A Landscape of the New Dark Silicon Design Regime[J]. IEEE Micro, 2013, 33(5):8–19

[49] W. Huang, K. Rajamani, M. Stan, et al. Scaling with Design Constraints: Predicting the Future of Big Chips[J]. IEEE Micro, 2011, 31(4):16–29

[50] M. Shafique, S. Garg, J. Henkel, et al. The EDA challenges in the dark silicon era[C]. Proceedings of Design Automation Conference (DAC), 2014, 1–6

[51] D. Cuesta, J. Ayala, J. Hidalgo, et al. Adaptive Task Migration Policies for Thermal control in MPSoCs[C]. IEEE Annual Symposium on VLSI, 2010, 110–115

攻硕期间取得的研究成果

[1] J. Ma, H. Wang, S. Tan, et al. Hybrid Dynamic Thermal Management Method with Model Predictive Control[C]. IEEE Asia Pacific Conference on Circuits and Systems, Okinawa, Japan, 2014, 743–746

[2] H. Wang, J. Ma, S. Tan, et al. Hierarchical Dynamic Thermal Management Method for High-Performance Many-Core Microprocessors[J]. ACM Transactions on Design Automation of Electronic Systems (TODAES), 2016