

摘要

J. H. Wilkinson^[?]建立了非奇异矩阵的逆是矩阵元素的连续函数的理论。G. W. Stewart^[?]推出了矩阵的广义逆的连续性。为了得到Drazin逆的连续性，本文先给出了 M -矩阵、 H -矩阵类的逆的连续性。Campbell和Meyer^[?]也给出了Drazin逆的连续性性质，但没有给出明显的边界。

Drazin逆对扰动是很不稳定的。然而，在某种特定的扰动条件下，矩阵 $(A + E)^D$ 与 A^D 的接近程度能够得到量化且也能得到明显的相对误差边界。基于Drazin逆的不同形式，很多科学家和学者从事这一方面的研究。U. G. Rothblum 给出的Drazin逆的以下的表达式：

$$A^D = (A - H)^{-1}(I - H) = (I - H)(A - H)^{-1}$$

其中 $H = I - AA^D = I - A^D A$. 基于这个表达式，我们在本文中也给出了 $\|(A + E)^D - A^D\|_2 / \|A^D\|$ 和 $\|(A + E)^\# - A^D\|_2 / \|A^D\|_2$ 的范数估计，并与前人的成果进行了比较。

关键词： M -矩阵， H -矩阵，Drazin逆，Pseudo-Drazin逆，条件数

ABSTRACT

The theory that the inverse of a nonsingular matrix is continuous function of the elements of the matrix was established by J. H. Wilkinson^[?]. The continuity of the generalized inverse A^+ of a matrix A was introduced by G. W. Stewart^[?]. In this paper, at first, the continuity of the special matrices inverse, such that M -matrices and H -matrices, respectively, are provided. Campbell and Meyer^[?] also established the continuity properties of Drazin inverse, but the explicit bound was not given.

The Drazin inverse is unstable with respect to perturbation. However, under some specific perturbation , the closeness of the matrices $(A + E)^D$ and A^D can be proved and the explicit bound the relation error can also be obtained. Based on the different representations of Drazin inverse, many scientists and scholars have worked it research. U. G. Rothblum gave the following representation of Drazin inverse:

$$A^D = (A - H)^{-1}(I - H) = (I - H)(A - H)^{-1}$$

where $H = I - AA^D = I - A^D A$. Based on the representation, we also obtain the norm estimate of $\|(A + E)^D - A^D\|_2/\|A^D\|$ and $\|(A + E)^\# - A^D\|_2/\|A^D\|_2$ and compare with the precedent results.

Keywords: M -matrices, H -matrices, Drazin inverse, Pseudo-Drazin inverse, Condition number

目 录

第一章 绪论	1
1.1 研究工作的背景与意义	1
1.2 动态热管理技术的发展	1
1.3 本论文的主要工作	2
第二章 动态温度管理相关工作分析	3
2.1 温度不受管理的影响	3
2.1.1 温度对系统可靠性的影响	3
2.1.2 温度对静态功耗的影响	4
2.1.3 温度对冷却系统的影响	4
2.2 动态温度管理技术	5
2.2.1 动态温度管理和动态功耗管理的不同	6
2.2.2 动态电压频率调整技术	6
2.2.3 任务迁移技术	8
2.3 预测控制方法	10
2.3.1 芯片温度预测方法	10
2.3.2 模型预测控制方法	11
第三章 芯片热建模方法	13
3.1 热传导理论	13
3.2 HotSpot热建模方法	15
3.2.1 主要热流路径	15
3.2.2 次要热流路径	18
第四章 基于动态温度管理的模型预测控制	20
4.1 微处理器热模型	20
4.2 用模型预测控制方法计算期望的功耗	20
4.3 基于期望功耗的任务迁移和动态电压频率调整	23
第五章 分层的动态温度管理方法	25
5.1 低层块内任务迁移	25
5.2 高层块间任务迁移	26
5.3 DVFS最终调整	30
第六章 实验结果	32

目录

6.1 与其他方法的瞬态温度比较	34
6.2 与其他方法的算法执行时间比较	37
6.3 与其他方法的性能比较	38
致 谢	40
参考文献	41
附录 A 附录章	46
A.1 附录节	46
附录 B 附录另一章	47
B.1 附录另一章的一节	47
攻硕期间取得的研究成果	48

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的问题。高的操作温度对微处理器的可靠性有负面影响。在新一代的高性能多核和众核微处理器中，正在增长的功耗密度和空间上的功耗差距带来了局部热问题，结果导致性能下降，散热开销大，和严重的可靠性问题。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 [2]。

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 [3]。在处理器在高性能水平运行时，动态温度管理通过控制处理器的运行行为，来保证处理器在安全温度以内。动态温度管理一般有两个方法，任务迁移和动态电压频率调整 (DVFS)，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 [4–9]，而且也可以降低多核系统的能量损耗 [10]。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。但是如果沒有其他动态温度管理方法的话，这个可能会有平均温度太高的问题。

动态电压频率调整 (DVFS) [11–13] 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 [14, 15]，它随温度限制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。控制器分析热模型，热传感信息等等，然后为动态温度管理技术输出引导行为，例如，DVFS应该被调整到多少频率。许多动态温度管理方法是基于传统控制方法的。但是这些方法并不太适合多核和众核热系统，因为他的复杂性。最近，模型预测方法 (MPC) 被引入动态温度管理。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。

DVFS和任务迁移结合MPC或许可以得到这三种方法的优点。MPC有高质量的控制，任务迁移提升更高的性能，DVFS保证温度的安全。在这方面有许多研究，大都是结合这三种方法中的两种。有的研究基于经验结合任务迁移和DVFS，有的研究结合MPC和DVFS。然而，结合MPC和任务迁移要比MPC结合DVFS更难。最近有研究结合了这三种方法。但是这个方法只能应用于多核微处理器，因为在众核处理器中集成MPC和任务迁移的将引入很大的开销。

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一个新的分层动态温度管理办法。新的方法用模型预测控制来引导包含任务迁移和DVFS的管理过程。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，新的方法将任务迁移分成两层，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。

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 温度不受管理的影响

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平衡。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[16]。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[3]。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高达服务器 51% 的功耗[17][18]。

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后果就是损害芯片的可靠性。下面是温度相关的半导体器件失效机理[19]：

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:

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

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。

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:

$$\lambda \propto \Delta T^q \quad (2-2)$$

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。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。大的温度梯度最主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[20], 漏电流可以表示为:

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

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 $T^2 e^{1/T}$ 。图 2-1 是漏电流对温度的曲线。

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

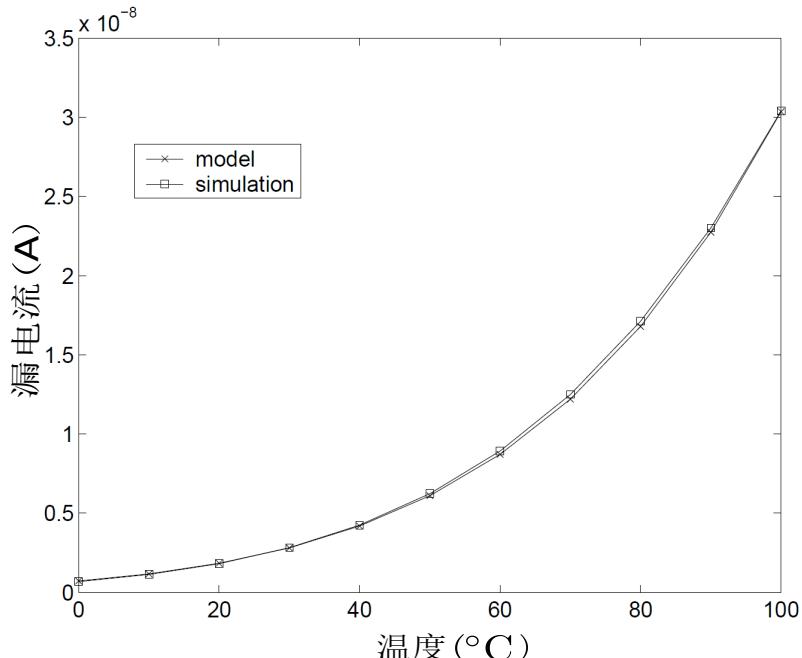


图 2-1 温度与漏电流的关系[21]

因为散热器的散热能力是由热阻 R_{h2a} （R2a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：

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

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。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 $10^{\circ}\text{C} \sim 15^{\circ}\text{C}$ 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[22]。温度每降低 9°C ，静态功耗降低50%[23]。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[24]。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想动态电压频率调整（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。首先，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， $T \propto T/A$ ，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 [25]。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。

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 2-2所示：

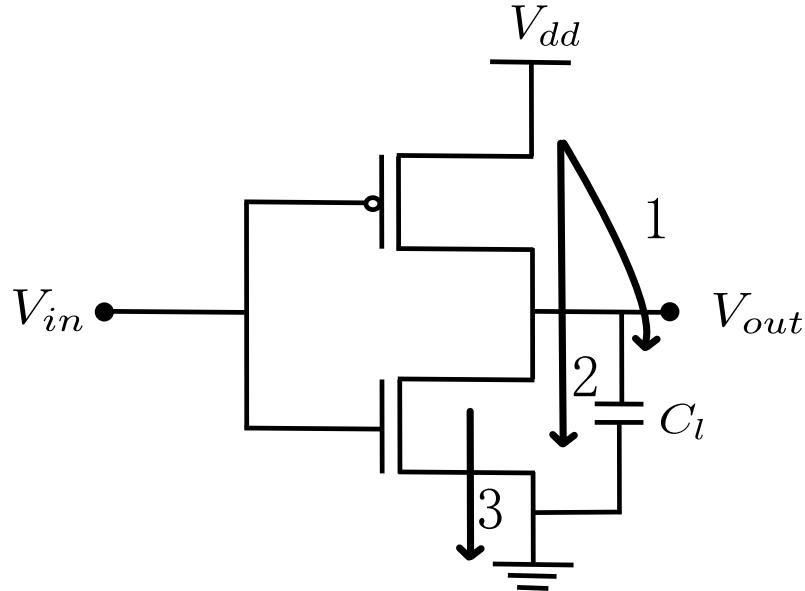


图 2-2 简单CMOS反相器功耗示意图

- P_d 表示动态功耗，源于电容的充电和放电（1）。
- P_s 表示静态功耗，源于晶体管反向偏置（2）。
- P_{sh} 表示电路短路电流，源于开关时 V_{dd} 到 GND 的直接通路。

CMOS 电路总功耗表示为：

$$P_{cmos} = P_d + P_s + P_{sh} \quad (2-5)$$

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：

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

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 [26]，其关系可表示为：

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

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 2-3 所示。

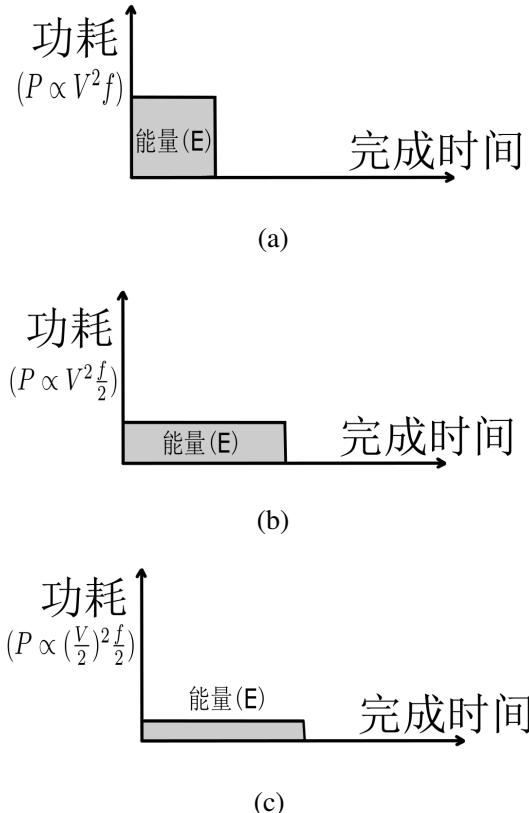


图 2-3 任务的功耗与能耗

(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

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} 。[27] 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）。

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。这样的话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 2-4。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，出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。

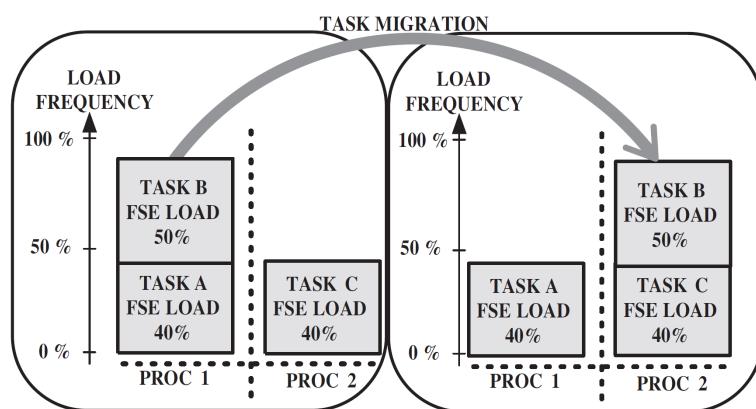


图 2-4 简单的任务迁移例子 [28]

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，提前才去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 [29] 和基于自回归移动平均值(ARMA)的温度预测 [30]。

在 [29]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。

$$y = \theta_1 u_1 + \theta_2 u_2 + \cdots + \theta_n u_n \quad (2-8)$$

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。

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

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，由芯片热特性决定，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。

$$T_{predict} = w_s T_{ABTM} + w_l T_{CBTM} \quad (2-10)$$

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[29]的温度预测控制方法。

在[30]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。

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

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

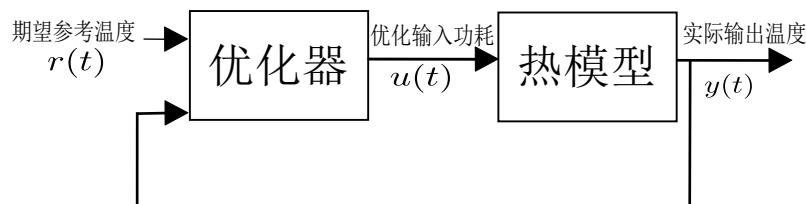


图 2-5 模型预测控制基本结构

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)方法被引入到动态温度管理中[27]，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 2-5 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 [31]，如图 2-6 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。

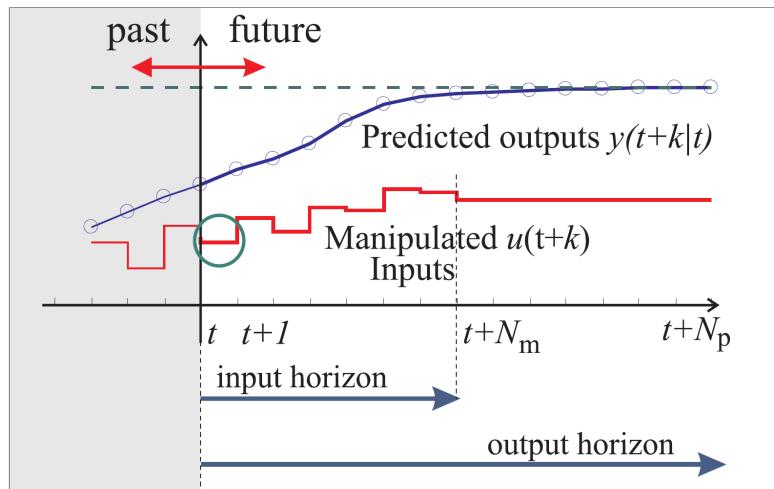


图 2-6 滚动优化策略（只有预测得到的第一步 u 被采用） [31]

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。这一章主要介绍一下热模型基本知识，尤其介绍一种简洁的热建模方法HotSpot。

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：

$$q = -k \frac{dT}{dx} \quad (3-1)$$

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

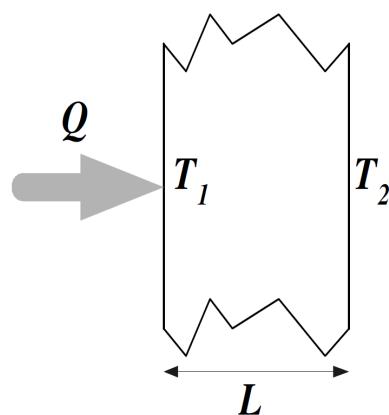


图 3-1 一维热传导

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)。

$$Q = -kA \frac{T_2 - T_1}{L} \quad (3-2)$$

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

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

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：

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

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)；热传导率 (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：

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

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：

$$(\rho c_p A L) \frac{dT(t)}{dt} = k A \frac{\Delta T(t)}{L} + Q \quad (3-6)$$

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

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

其中， $C_{th} = \rho c_p A L = \rho c_p V$ 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。

3.2 HotSpot热建模方法

大多现在VLSI系统的封装有若干不同材料的叠层组成，如图 3-2。HotSpot热

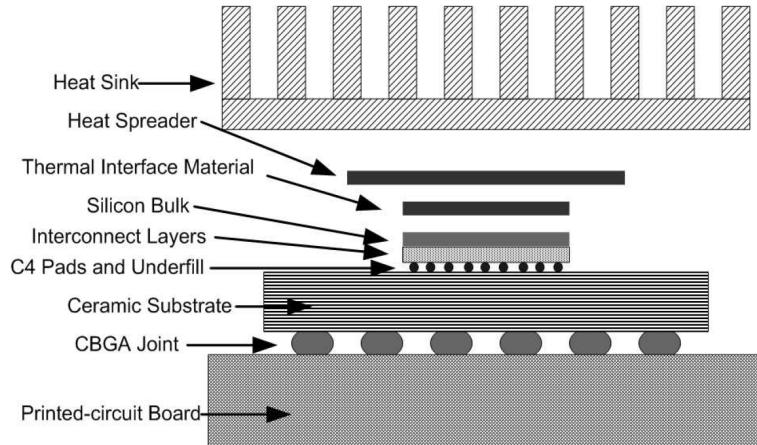


图 3-2 典型CBGA封装的堆叠层

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

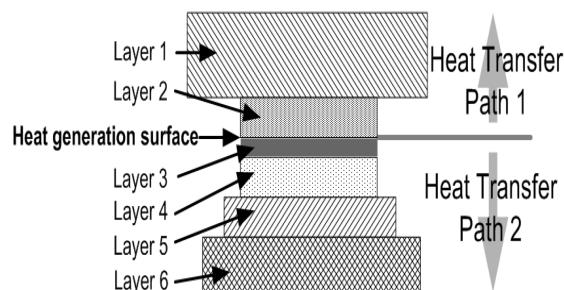


图 3-3 HotSpot中用的抽象堆叠层结构

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。

3.2.1 主要热流路径

首先我们来介绍主热流路径模型。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)

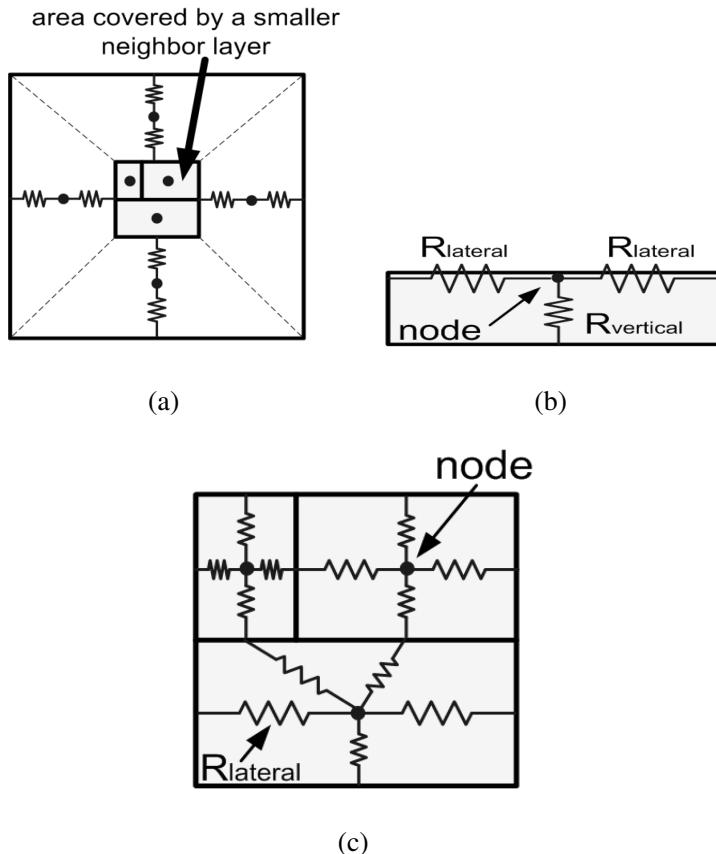


图 3-4 层划分和等效热阻示意图

(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)

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层覆盖，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 L_1 和 L_2 ，芯片厚度

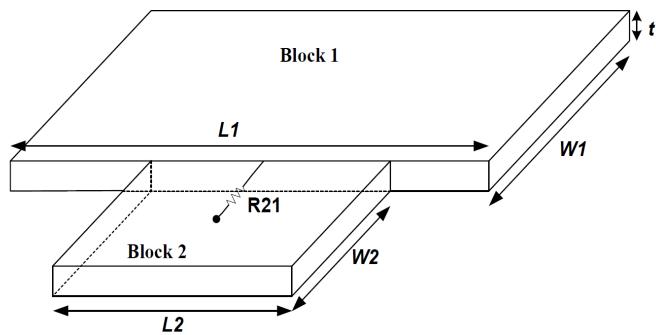


图 3-5 扩散热阻示意图

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 [32] 中的公式来计算扩散热阻。

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，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”，连接到响应的节点。

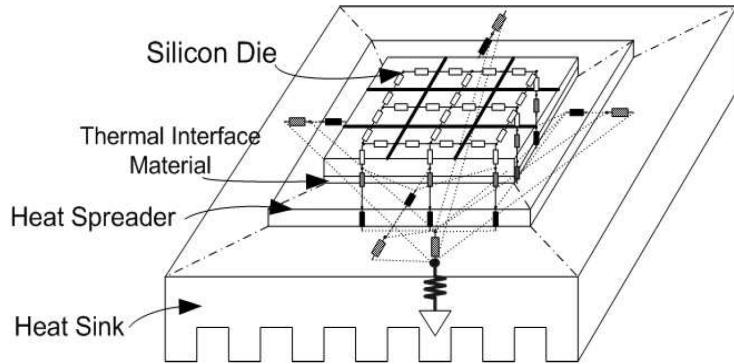


图 3-6 芯片 3×3 网格单元简洁热模型示例（图3-2中主热流路径倒置结构）

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 是导线中的电流， $R = \rho_m \cdot l / A_m$ 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 [33]。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。所有的先周围都是层间介质。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。

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。

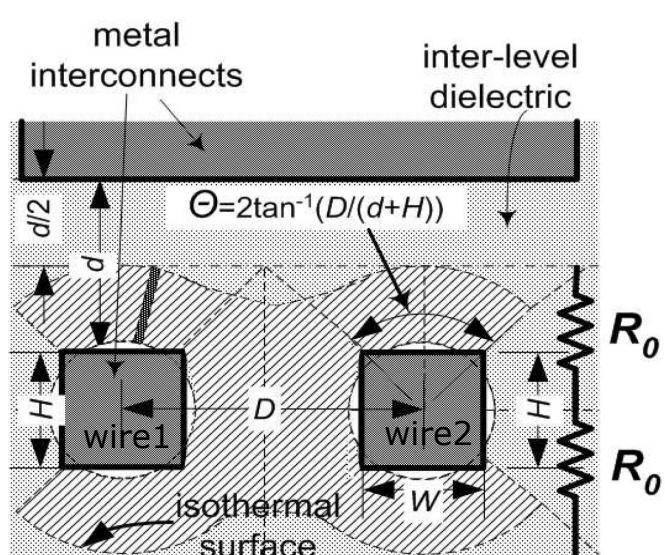


图 3-7 计算互连线和周围介质等效热阻示意图

第四章 基于动态温度管理的模型预测控制

在这一章我们将介绍基于模型预测控制方法的动态温度管理方法。4.1节中介紹结合模型预测控制MPC的热模型，4.2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4.3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 微处理器热模型

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。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，

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

其中， $T(t) \in \mathbb{R}^n$ 是表示处理器 n 块温度的向量，包括 l 个核($l < n$)，边界节点和封装部分的节点； $G \in \mathbb{R}^{n \times n}$ 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)离散化为下面的差分方程

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

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 $Y(k)$ ）。

4.2 用模型预测控制方法计算期望的功耗

4.1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

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

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

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

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

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 ）

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

其中， N_c 称作控制域。核的预测温度定义为

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

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

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

其中

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

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

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

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。 $R = r I_{N_c \times N_c}$ 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得到合适的值。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

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

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

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

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。

4.3 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只能降低这个核的功耗，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，可以将这个构建成一个二部图，当成一个二部图匹配问题来处理。相应的二部图可以表示为 $\mathcal{G} = (L, R, E)$ ，这里 $L =$

$\{p_1, p_2, \dots, p_l\}$, $R = \{\bar{p}_1, \bar{p}_2, \dots, \bar{p}_l\}$, 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 $e_{ij} = |p_i - \bar{p}_j|$ 。这个二部图匹配问题可以用匈牙利算法解决, 能发现匹配对, 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, 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。

图 4-1 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 $e_{th} = 3$, 图 4-1 (b) 是匹配结果, 表示出了匹配对 (p_2, \bar{p}_1) , (p_3, \bar{p}_2) 和 (p_4, \bar{p}_3) 。

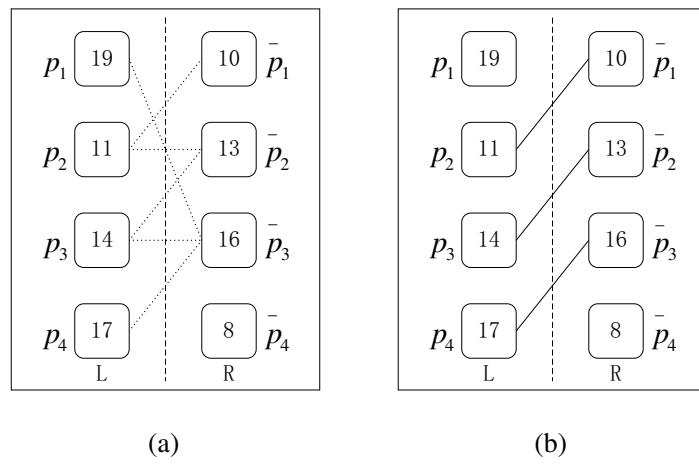


图 4-1 带权重二部图匹配的例子

(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

第五章 分层的动态温度管理方法

在这一章，针对高性能众核微处理器，提出了新的分层动态温度管理方法。新方法是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

首先，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4.3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 4-1 展示的正是图 5-1 (a)中块I内的二部图匹配。

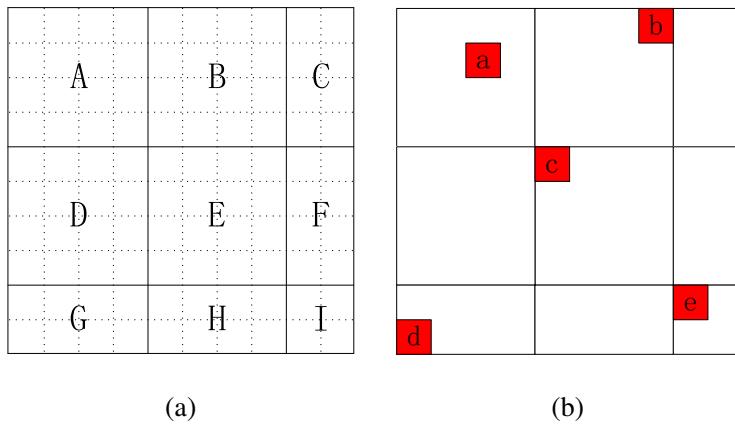


图 5-1 100核微处理器的垂直视图（作为分层算法的一个例子）

(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 4-1 (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。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 $V_p = \{p_1, \bar{p}_1, p_2, \bar{p}_2, \dots, p_m, \bar{p}_m\}$ ， 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 $E(v)$ 。将图5-2(b)中的元素 P_a 做一个例子， $I(P_a) = \{(P_a, \bar{P}_a), (P_a, \bar{P}_b), (P_a, \bar{P}_c)\}$ ， $E(P_a) = \{(P_a, \bar{P}_d), (P_a, \bar{P}_e)\}$ 。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 $f(v)$ 。

$$f(v) = \sum_{n_i \in E(v)} w(n_i) - \sum_{n_j \in I(v)} w(n_j). \quad (5-1)$$

$f(v)$ 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。

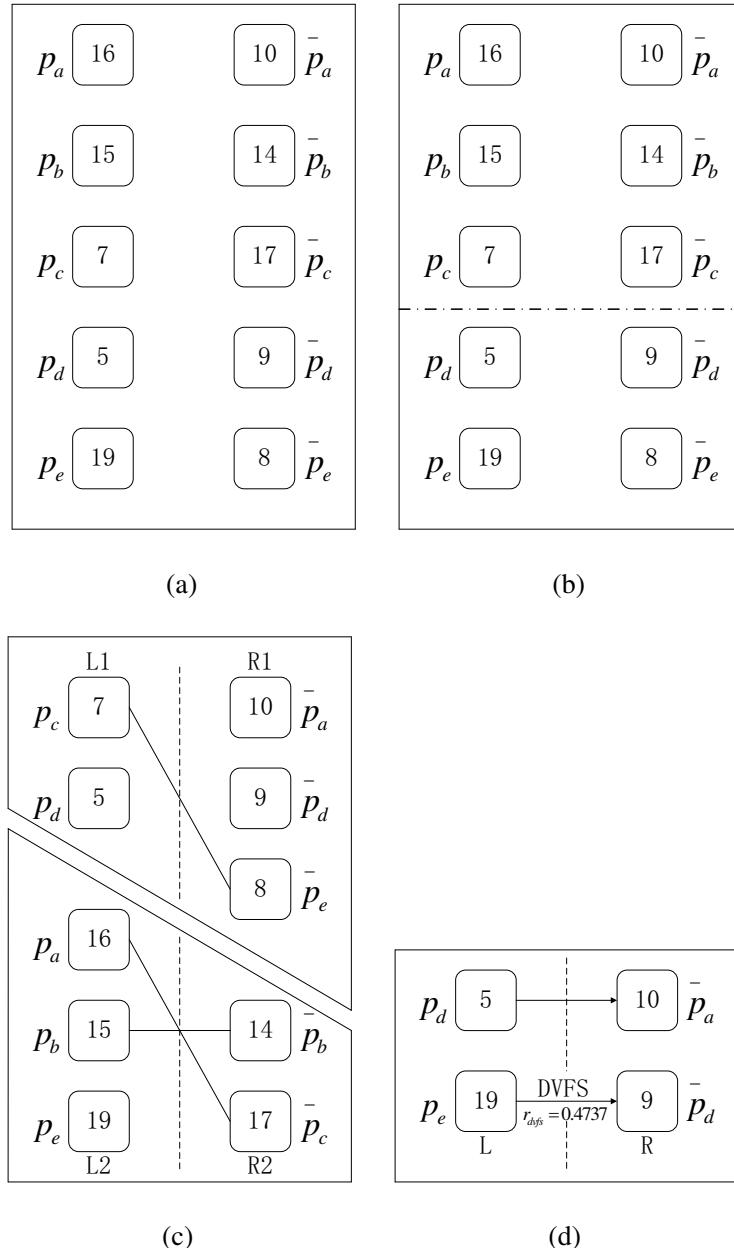


图 5-2 分层算法在100核微处理器上的高层处理过程示例

(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 (作为改进迭代最小割算法的第一步); (c) \mathcal{G}_p 上的最小割, 然后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

它相邻的元素的增益都要更新。在这个例子中，最合适元素是 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， $\mathcal{F} = \{f(v_1), f(v_2), \dots, f(v_{2m})\}$ 。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, \dots, 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, \dots, v_i 的总增益（记作 $\tilde{f}(v_i)$ ），我们需要取和：

$$\tilde{f}(v_i) = \sum_{j=1}^i f(v_j). \quad (5-2)$$

对 $i = 1, 2, \dots, 2m$ 分别计算 $\tilde{f}(v_i)$ ，我们形成一个累积和序列， $\tilde{\mathcal{F}} = \{\tilde{f}(v_1), \tilde{f}(v_2), \dots, \tilde{f}(v_{2m})\}$ ，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 $\tilde{\mathcal{F}}$ 中的最大元素，这意味着移动 v_1, v_2, \dots, 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 $\{v_1, v_2, \dots, v_k\}$ 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。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，需要在未来的工上做进一步的研究。

这里总结一下这个新算法，我们将整个流程总结如下。

1 分层动态温度管理算法

- 1: 用模型预测控制（MPC）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。
-

第六章 实验结果

这个实验是在一个具有两个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。新的分层动态热管理方法主要用MATLAB实现。我们分别构建了四个不同的处理器核配置的众核处理器，从100核（ 10×10 ）到625核（ 25×25 ）。这些处理器的热模型由HotSpot生成。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 $10mm \times 10mm \times 0.15mm$ 。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100个实时功耗信息，256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，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。另一个更广泛的方法是完全关掉一些核[34, 35]。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
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100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。

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1GHZ的处理器大概是1ms[36]。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。

作为比较，我们实现了另外两种基于MPC的方法，众核处理器基于MPC混合任务迁移和DVFS的动态温度管理方法[37]和基于MPC只用DVFS的方法。我们也选择了[38]中的动态温度管理方法作为对比，因为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开源项目MAGMA V2，这个由[38]的作者提供。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。

6.1 与其他方法的瞬态温度比较

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 6-1 (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。温度在120°C 左右会影响芯片的可靠性。我们也测量了温度的方差，见图 6-2 (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。

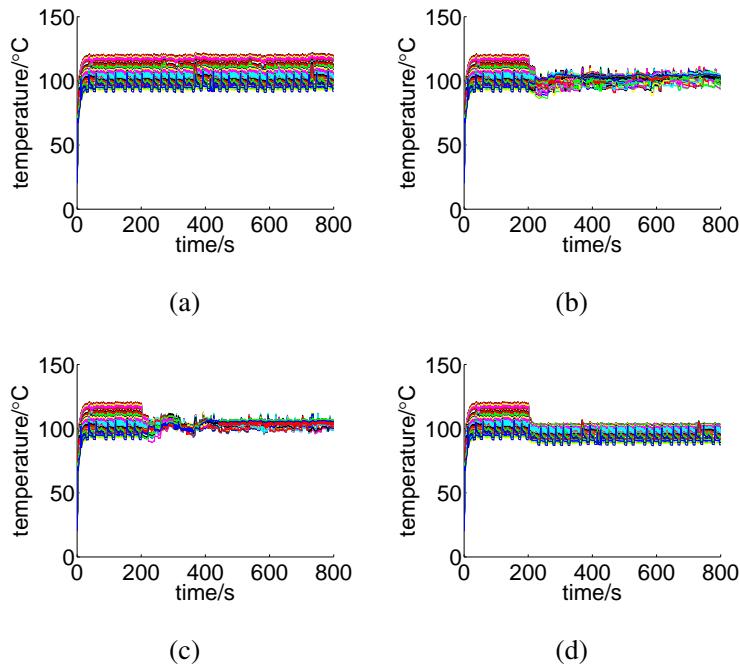


图 6-1 Transient temperature traces of the 100-core CPU with different DTM methods. Lines with different colors represents temperature traces of different cores. The new method in (b) and [37] in (C) both successfully tracks the ceiling temperature. But the DVFS only method in (d) has many low temperature cores, which means the chip performance is lower.

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 6-1 (b)，对应的温度方差就是图 6-2 (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。

作为对比，[37]中的基于MPC的结合任务迁移和DVFS的动态温度管理方法和基于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 6-1 (c) (d) 和图 6-2 (c) (d) 就是这两种方法对应的瞬态温度，和核之间温度方差。我们提出的方法和[37]中的方法有相似的瞬态温度，比[37]中的核之间温度方差稍微大一点点

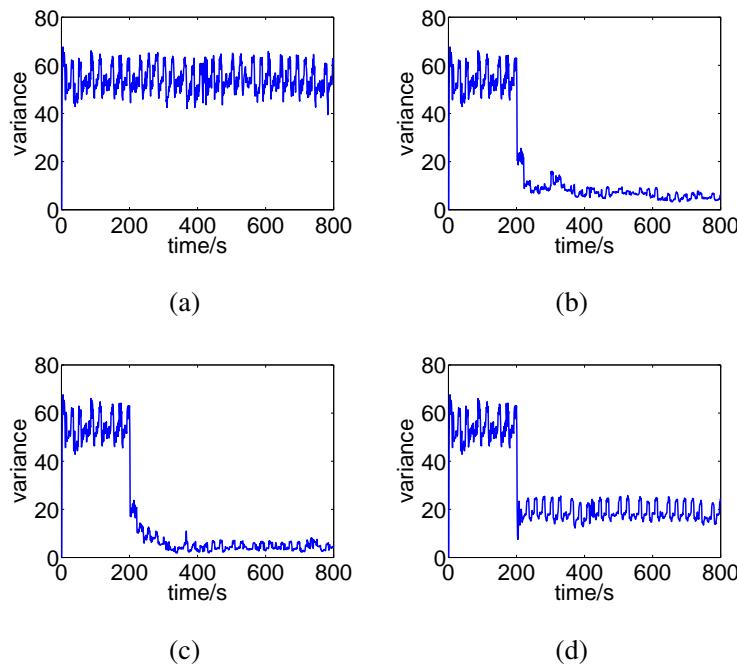


图 6-2 Transient variances among cores in the 100-core CPU with different DTM methods. The temperature variance of the new method is slightly higher than that of the method in [37], but much lower than that of the DVFS only method.

(后面会介绍，新方法在开销和扩展性上比[37]要好得多）。基于MPC结合DVFS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[38]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 6-3 (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 6-3 (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优化的，会明显的超出实际温度的限制。从图 6-3我们可以看出，温度控制有一个延时调整问题。这是因为在[38]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 6-1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。

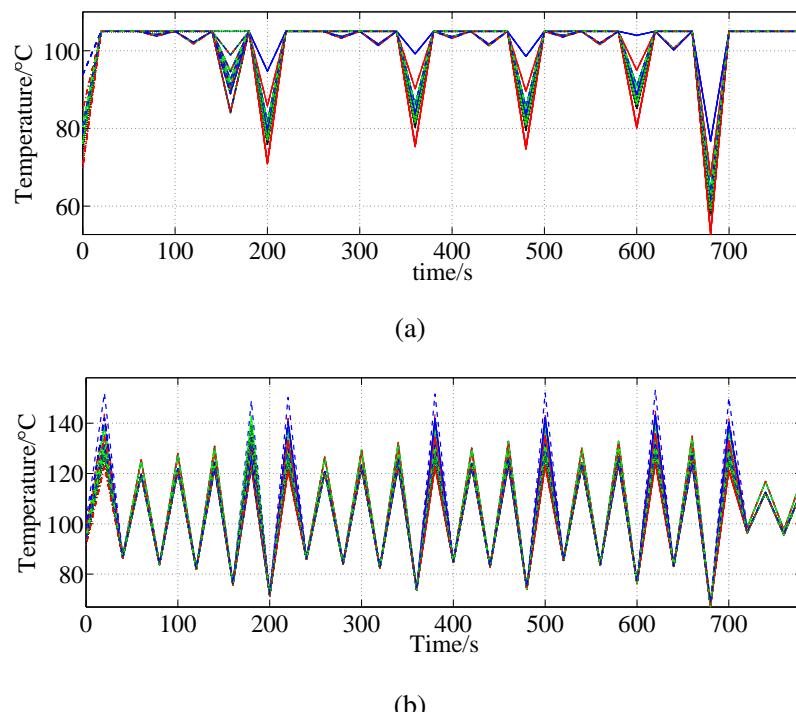


图 6-3 Transient temperature traces of cores by performing DTM in [38] on the 100-core microprocessor. Core to core heat exchange has huge impact on temperatures for many-core microprocessors, causing the 105°C ceiling temperature to be seriously violated. Overshoot problem is also significant comparing to the MPC based techniques.

表 6-2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 是基于MPC只结合DVFS的动态温度管理方法的核之间温度方差， var_c 是[37]中动态温度管理方法的核之间温度方差， var_n 是我们的分层温度管理方法的核之间温度方差， var_h 是[38]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，[37]中方法稍微优于我们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。[38]中的方法只完成了100核的实验，其核之间温度方差与[37]中的方法接近。

6.2 与其他方法的算法执行时间比较

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[37]中的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 [37]、[38]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。[37]中的方法也有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 。只用DVFS的方法只有MPC时间 t_p 。尽管[38]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 6-3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，[37]中的方法开销明显上升。从400核的情况开始，每秒的平均管理决策时间要超过1s，这是完全不能接受的。[38]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

表 6-2 The temperature variance among cores in CPUs with different core configuration. var_o is the variance without any DTM method, var_d is variance with DTM using DVFS only, var_c is variance with DTM in [37], and var_n is variance with our new hierarchical method, and var_h is the variance with method in [38].

Configuration	var_o	var_d	var_c	var_n	var_h
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

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[37]中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，仅仅用DVFS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。

表 6-3 The average runtime per second of the DTM methods on CPUs with different core configurations. t_p represents MPC time, t_m means task migration matching time, and t_a denotes for minimum cut partitioning time.

核数	分层方法				[37]			仅DVFS方法		[38]	
	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

表 6-4 The average number of instructions per second of one core on CPUs with different configurations. $MIPS_o$ represents the IPS in million (MIPS) of the core without any DTM method, $MIPS_d$ is for MIPS of DTM with DVFS only, and $MIPS_n$ denotes MIPS of our new hierarchical method.

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

6.3 与其他方法的性能比较

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[37] 和 [38]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 6-4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DVFS方法时单个处理器核的平均IPS， $MIPS_n$ 表示采用我们的新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。 $MIPS_n$ 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，我们的新算法在性能上优于只采用DVFS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DVFS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DVFS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。

致 谢

历时将近两个月的时间终于将这篇论文写完，在论文的写作过程中遇到了无数的困难和障碍，都在同学和老师的帮助下度过了。尤其要强烈感谢我的论文指导老师—XX老师，她对我进行了无私的指导和帮助，不厌其烦的帮助进行论文的修改和改进。另外，在校图书馆查找资料的时候，图书馆的老师也给我提供了很多方面的支持与帮助。在此向帮助和指导过我的各位老师表示最中心的感谢！

感谢这篇论文所涉及到的各位学者。本文引用了数位学者的研究文献，如果没有各位学者的研究成果的帮助和启发，我将很难完成本篇论文的写作。

感谢我的同学和朋友，在我写论文的过程中给予我了很多你间素材，还在论文的撰写和排版灯过程中提供热情的帮助。由于我的学术水平有限，所写论文难免有不足之处，恳请各位老师和学友批评和指正！

参考文献

- [1] S. Borkar. Thousand core chips: a technology perspective[M]. 2007, 746–749
- [2] 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
- [3] J. Donald, M. Martonosi. Techniques for Multicore Thermal Management: Classification and New Exploration[M]. 2006, 78–88
- [4] M. Powell, M. Gomaa, T. Vijaykumar. Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the Operating System[M]. 2004, 260–270
- [5] Y. Ge, P. Malani, Q. Qiu. Distributed task migration for thermal management in many-core systems[M]. 2010, 579–584
- [6] T. Chantem, S. Hu, R. Dick. Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs[J]. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 2011, 19(10):1884–1897
- [7] G. Liu, M. Fan, G. Quan. Neigbor-Aware Dynamic Thermal Management for Multi-core Platform[M]. 2012, 187–192
- [8] R. Ayoub, T. Rosing. Predict and Act: Dynamic Thermal Management for Multi-Core Processor[M]. 2009, 99–104
- [9] T. Ebi, M. Al Faruque, J. Henkel. TAPE: Thermal-aware agent-based power economy for multi/many-core architectures[M]. 2009, 302–309
- [10] J. Cong, B. Yuan. Energy-Efficient Scheduling on Heterogeneous Multi-Core Architectures[M]. 2012, 345–350
- [11] K. Skadron, M. Stan, W. Huang, et al. Temperature-Aware Microarchitecture[M]. 2003, 2–13
- [12] R. Jayaseelan, T. Mitra. A hybrid local-global approach for multi-core thermal management[M]. 2009, 314–320
- [13] A. Mutapcic, S. Boyd, S. Murali, et al. Processor Speed Control with Thermal Constraints[J]. IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications, 2009, 56(9):1994–2007
- [14] H. Khdr, S. Pagani, M. Shafique, et al. Thermal Constrained Resource Management for Mixed ILP-TLP Workloads in Dark Silicon Chips[M]. 2015, 1–6
- [15] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, et al. Hierarchical Power Management for Asymmetric Multi-Core in Dark Silicon Era[M]. 2013, 1–9

- [16] 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
- [17] C. Lefurgy, K. Rajamani, F. Rawson, et al. Energy management for commercial servers[J]. Computer, 2003, 36(12):39–48
- [18] R. Ayoub, S. Sharifi, T. S. Rosing. Gentlecool: Cooling aware proactive workload scheduling in multi-machine systems[M]. 2010, 295–298
- [19] J. S. S. T. Association, et al. Failure mechanisms and models for semiconductor devices[J]. JEDEC Publication JEP122-B, 2003
- [20] W. Liao, L. He, K. M. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2005, 24(7):1042–1053
- [21] Y. Zhang, D. Parikh, K. Sankaranarayanan, et al. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects[J]. University of Virginia Dept of Computer Science Tech Report CS-2003, 2003, 5
- [22] E. Kursun, C.-Y. Cher, A. Buyuktosunoglu, et al. Investigating the effects of task scheduling on thermal behavior[M]. 2006
- [23] Y. Liu, R. P. Dick, L. Shang, et al. Accurate temperature-dependent integrated circuit leakage power estimation is easy[M]. 2007, 1526–1531
- [24] V. K. Pamula, K. Chakrabarty. Cooling of integrated circuits using droplet-based microfluidics[M]. 2003, 84–87
- [25] E. Le Sueur, G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns[M]. 2010, 1–8
- [26] D. Suleiman, M. A. Ibrahim, I. Hamarash. Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction[M]. 2005
- [27] F. Zanini, D. Atienza, L. Benini, et al. Multicore Thermal Management with Model Predictive Control[M]. Piscataway, NJ, USA: IEEE Press, 2009, 90–95
- [28] F. Mulas, M. Pittau, M. Buttu, et al. Thermal balancing policy for streaming computing on multiprocessor architectures[M]. 2008, 734–739
- [29] I. Yeo, C. C. Liu, E. J. Kim. Predictive dynamic thermal management for multicore systems[M]. 2008, 734–739
- [30] A. K. Coskun, T. S. Rosing, K. G. C. Gross. Proactive temperature balancing for low cost thermal management in MPSoCs[M]. 2008, 250–257

- [31] A. Bemporad, M. Morari. Robust model predictive control: A survey[M]. Springer, 1999, 207–226
- [32] S. Song, V. Au, K. P. Moran. Constriction/spreading resistance model for electronics packaging[M]. 1995, 199–206
- [33] W. Huang, M. R. Stan, K. Skadron, et al. Compact thermal modeling for temperature-aware design[M]. 2004, 878–883
- [34] M. Taylor. A Landscape of the New Dark Silicon Design Regime[J]. IEEE Micro, 2013, 33(5):8–19
- [35] M. Shafique, S. Garg, J. Henkel, et al. The EDA challenges in the dark silicon era[M]. 2014, 1–6
- [36] D. Cuesta, J. Ayala, J. Hidalgo, et al. Adaptive Task Migration Policies for Thermal control in MPSoCs[M]. 2010, 110–115
- [37] 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, 2014
- [38] V. Hanumaiah, S. Vrudhula, K. Chatha. Performance optimal online DVFS and task migration techniques for thermally constrained multi-core processors[J]. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2011, 30(11):1677–1690
- [39] H. Wang, S. Tan, D. Li, et al. Composable Thermal Modeling and Simulation for Architecture-Level Thermal Designs of Multi-core Microprocessors[J]. ACM Trans. on Design Automation of Electronics Systems, 2013, 18(2):28:1–28:27
- [40] H. Wang, S. Tan, G. Liao, et al. Full-chip runtime error-tolerant thermal estimation and prediction for practical thermal management[C]. Proc. Int. Conf. on Computer Aided Design (ICCAD), 2011, 716–723
- [41] D. Brooks, M. Martonosi. Dynamic thermal management for high-performance microprocessors[M]. 2001, 171–182
- [42] V. Hanumaiah, S. Vrudhula. Energy-efficient operation of multicore processors by DVFS, task migration, and active cooling[J]. IEEE Trans. on Computers, 2014, 63(2):349–360
- [43] L. Wang. Model Predictive Control System Design and Implementation Using MATLAB[M]. Springer, 2009
- [44] Y. Wang, K. Ma, X. Wang. Temperature-Constrained Power Control for Chip Multiprocessors with Online Model Estimation[M]. 2009, 314–324
- [45] 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

- [46] A. Bartolini, M. Lombardi, M. Milano, et al. Optimization and Controlled Systems: A Case Study on Thermal Aware Workload Dispatching[M]. 2012, 427–433
- [47] D.-S. Chiou, S.-H. Chen, S.-C. Chang, et al. Timing Driven Power Gating[M]. 2006, 121–124
- [48] H. Kuhn. The Hungarian method for the assignment problem[J]. Naval Research Logistics Quarterly, 1955, 2:83–97
- [49] J. Munkres. Algorithms for the assignment and transportation problems[J]. Journal of the Society for Industrial and Applied Mathematics, 1957, 5(1):32–38
- [50] W. Huang, S. Ghosh, S. Velusamy, et al. HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design[J]. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 2006, 14(5):501–513
- [51] D. Brooks, V. Tiwari, M. Martonosi. Wattech: A Framework for Architectural-Level Power Analysis and Optimizations[M]. 2000, 83–94
- [52] J. L. Henning. SPEC CPU 2000: Measuring CPU Performance in the New Millennium[J]. IEEE computer, 2000, 1(7):28–35
- [53] S. Dutt, W. Deng. A probability-based approach to VLSI circuit partitioning[M]. 1996, 100–105
- [54] M. Kadin, S. Reda, A. Uht. Central versus Distributed Dynamic Thermal Management for Multi-Core Processors: Which One Is Better?[M]. 2009, 137–140
- [55] F. Zanini, D. Atienza, L. Benini, et al. Thermal-Aware System-Level Modeling and Management for Multi-Processor Systems-On-Chip[M]. 2011, 2481–2484
- [56] C. Fiduccia, R. Mattheyses. A linear-time heuristic for improving network partitions[M]. 1982, 175–181
- [57] S. Boyd, L. Vandenberghe. Convex Optimization[M]. the United States of America by Cambridge University Press, New York, 2006
- [58] S. Pagani, H. Khdr, W. Munawar, et al. TSP: Thermal Safe Power - Efficient power budgeting for many-core systems in dark silicon[M]. 2014, 1–10
- [59] J. H. Lau, T. G. Yue. Thermal Management of 3D IC Integration with TSV (Through Silicon Via)[M]. 2009, 635–640
- [60] K. H. Lu, X. Zhang, S.-K. Ryu, et al. Thermo-mechanical reliability of 3-D ICs containing through silicon vias[M]. 2009, 630–634
- [61] D. Xin-hui, T. Feng, T. Shao-Qiong. Study of power system short-term load forecast based on artificial neural network and genetic algorithm[M]. 2010, 725–728
- [62] S. K. Marella, S. S. Sapatnekar. A Holistic Analysis of Circuit Performance Variations in 3-D ICs With Thermal and TSV-Induced Stress Considerations[J]. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 2014, 23:1308–1321

- [63] Z. Xiong, Y. Zhang, L. Ou, et al. Two-phases Parallel Neural Network Algorithm Based on RPROP[M]. 2005, 1–6
- [64] Q. Zou, T. Zhang, E. Kursun, et al. Thermomechanical stress-aware management for 3D IC designs[M]. 2013, 1255–1258
- [65] T. Chen, D. M. Liao, J. X. Zhou. Numerical Simulation of Casting Thermal Stress and Deformation Based on Finite Difference Method[J]. Materials Science Forum (MSF), 2013, 762:224–229
- [66] H. Esmaeilzadeh, E. Blem, R. S. Amant, et al. Dark silicon and the end of multicore scaling[J]. IEEE Micro, 2012, 32(3):122–134
- [67] F. Li, C. Nicopoulos, T. Richardson, et al. Design and Management of 3D Chip Multiprocessors Using Network-in-Memory[M]. 2006, 130–141
- [68] A. Sridhar, A. Vincenzi, M. Ruggiero, et al. 3D-ICE: Fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling[M]. 2010, 463–470
- [69] R. S. Patti. Three-Dimensional Integrated Circuits and the Future of System-on-Chip Designs[J]. Proc. of the IEEE, 2006, 94(6):1214–1224
- [70] M. Jung, J. Mitra, D. Z. Pan, et al. TSV Stress-Aware Full-Chip Mechanical Reliability Analysis and Optimization for 3-D IC[J]. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2012, 31(8):1194–1207
- [71] M. Jung, J. Mitra, D. Z. Pan, et al. TSV Stress-Aware Full-Chip Mechanical Reliability Analysis and Optimization for 3D IC[J]. Communications of the ACM, 2014, 57(1):107–115
- [72] C. Tan, T. Muthukaruppan, T. Mitra, et al. Approximation-Aware Scheduling on Heterogeneous Multi-core Architectures[M]. 2015, 618–623

附录 A 附录章

如果将appendix.tex中所有内容删除，最后的论文将不会出现附录。

A.1 附录节

附录 B 附录另一章

B.1 附录另一章的一节

攻硕期间取得的研究成果

- [37] 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