# PEPON：针对基于NoC的多核的性能感知分级功率预算

Akbar Shariﬁ, Asit K. Mishra, Shekhar Srikantaiah, Mahmut Kandemir and Chita R. Das Department of CSE The Pennsylvania State University University Park, PA 16802, USA {akbar, amishra, srikanta, kandemir, [das}@cse.psu.edu](mailto:das%7d@cse.psu.edu)

## 抽象

针对基于NoC的多核，我们提出了一种称为PEPON的两级功耗预算分配机制，其中第一级将多核系统的整体功耗预算分配到各种类型的片上资源（如内核，缓存和NoC）中， 第二级确定对每种类型资源的单独实例的权力分配。 这两种分布都以最大化工作负载性能为目标，而不超过指定的功耗预算。 使用全系统仿真和详细的功率模型对所提出的配电方案进行广泛的实验评估强调了两个层面上功率预算划分的重要性。 具体而言，我们的结果表明，与无功耗预算相比，所提出的方案可以提供高达29％的性能改进，并且在相同的芯片级功耗上限下，性能比竞争方案提高13％。

## 类别和主题描述

C.1.2 [计算机系统组织]：多处理器; 互连架构; C.4 [系统的性能]：设计研究

## 一般条款

设计，实验，管理，性能

## 关键词

功率预算，基于NoC的多核，性能

## 引言

最近在多核系统中给定功率预算下的功率预算和性能优化近来受到了相当的关注[19,20,24,32]。这些解决方案中的大多数专注于处理器级别的功耗预算，并且它们用来控制多核处理器功率的唯一旋钮是内核的电压/频率级别。在调制电压/频率时

**\*本研究部分受NSF拨款＃1213052，＃1152479，＃1147388，＃1139023，＃1017882，＃0963839，＃0811687和微软公司的授权支持**。

允许将个人或课堂使用的全部或部分作品的数字化或硬拷贝免费授予，前提是复制品不是为了获得利益或商业利益而制作或发行的，并且副本在第一页上包含本通知和完整引用。要复制，重新发布，发布到服务器或重新发布到列表，需要事先特定的许可和/或费用。 PACT'12，2012年9月19 - 23日，美国明尼苏达州明尼阿波利斯。版权所有2012 ACM 978-1-4503-1182-3 / 12/09 ... $ 15.00。

多核处理器中的核心频率水平有可能控制处理器的功耗，但它无法直接控制处理器的效能（每瓦性能）。这是因为应用程序的性能不受核心频率的控制（尽管它具有重要影响），但也受到其他片上组件的影响;特别是最后一级缓存和片上网络（NoC）。事实上，非核心组件功耗的重要性正在增加，并预计将迅速与核心本身的功耗相媲美，如果不是更大的话[11,12,23]。我们的实验研究也强化了文献中的这些观察结果。例如，我们在具有8MB末级缓存和4×4基于网格的NoC的16核机器上进行的实验表明，非核心组件平均消耗约33％的功率（分别为15％和18％由最后一级缓存库和NoC分别）。

英特尔最近基于多核的网络[12]分别报告了全功率和低功率工作模式下31％和79％的非核心功率贡献。 尽管高速缓存和NoC的功耗与内核的功耗相当，但据我们所知，迄今为止，还没有在所有这些资源中分配功耗预算以最大化性能，同时保持相同的性能 系统总体功耗预算。 我们相信，通过基于工作负载对核心/ NoC频率和缓存分配的敏感度，在资源之间智能分配总体功耗，可以实现显着的性能改进。

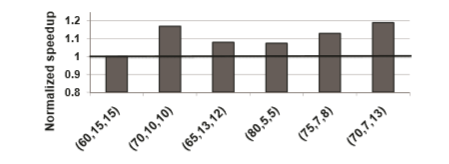


图1：在同一功率预算（90W）的不同分布下，16核机器上16个应用混合的性能（归一化加权加速）。 所有块被规范化到第一个块。 （x，y，z）分别表示特定的内核，缓存和NoC功耗预算（单位为瓦特）

为了说明本文利用的关键洞察力，整个系统功率的不同分布可能导致不同的系统性能（尽管保持总体功率预算相同），我们进行了一个简单的实验。我们将不同百分比的固定芯片功率上限（90W）分配给系统中的不同资源。核心频率被调整，以便核心的整体功率分配得到遵守。类似地，调整集合关联最后级高速缓存（LLC）中的活动高速缓存路的数量，使得总体高速缓存功率保持在分配的预算内，并且NoC频率被设置为使得NoC功率预算也受到尊重。图1显示了16个核心4×4系统上的16个应用程序的多程序工作负载组合，其性能由在相同总功耗预算的各种分布下的归一化加权加速度测量得到。从图中我们可以看到在相同的功率预算下，不同的功率预算分布（x轴）会导致显着不同的系统性能。请注意，图1中显示的结果仅与不同资源之间的总体功率的静态分布有关，这不会对工作负载行为的变化做出反应。此外，对特定资源进行功率分配的任何更改都需要跟踪实际（现在有限）的资源分配情况。因此，功率预算的动态分配可能实现更高的整体性能，并且跟踪应用程序之间分配的功率的机制也可以确保始终遵守功率预算，尽管工作负载行为发生变化。

为此，在本文中，我们提出了一种功率预算分配机制（称为PEPON），该机制将核心，缓存和NoC等各种关键资源之间的芯片级功耗预算进行分配，以最大化性能，同时尊重分配的预算 在相应的资源中，从而尊重整体功率预算。 具体而言，本文的主要贡献可概括如下：

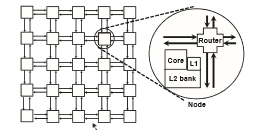
•我们提出了一种新颖的两级配电方案，通过仔细管理分配的功率以获得给定工作负载的最佳性能，在不同的资源之间分配特定的功率预算。 该方案采用不同功率分配级别的不同型号，并使用反馈控制器来执行核心和NoC功率分配。

•使用全系统仿真和详细的功耗/性能模型对建议技术进行广泛的评估表明，在所有片上资源中仔细分配功耗预算不仅可以确保处理器上的功率上限得到尊重，还可以优化系统 性能。 我们的实验分析表明，与未实施配电的情况相比，整体工作负载性能可提高多达29％。 此外，对于最近提出的仅考虑核心功率重新分配的功率预算方案[20]，我们的方案在同样的芯片级功耗预算下平均实现了13％的性能提升。

## 多核系统目标

图2显示了一个基于NoC的多核架构。 它是基于共享内存的二维网格结构，其中每个节点包含一个处理器核心，私有L1数据或指令高速缓存，共享L2高速缓存库以及通过它连接到相邻节点的路由器。 虽然所有节点共享相同的逻辑片上L2高速缓存空间，但数据访问延迟取决于请求核心与保存请求数据的高速缓存组之间的距离。 在访问某个其他节点的L2中的数据元素期间使用的链路集由路由策略确定，该路由策略在本文中为XY路由[17]。

1注意，虽然（对于给定的工作负载/配置），可以找到接近动态分区的静态分区，但需要进行彻底搜索才能实现此目的（实践中不可能）。 此外，在其他情况下，甚至可能找不到可比的静态分区（因为静态分区不考虑工作负载行为中的动态调制）



通讯链接

图2：基于NoC的多核架构。

设计这种多核架构的关键问题之一是功耗，主要来自三个来源：内核，片上高速缓存组件（共享L2 bank）和NoC。为了降低内核的功耗，可以采用文献中提出的基于动态电压/频率比例（DVFS）的方法[4,12,14]。 DVFS的想法是通过在处理器轻负载时降低电压/频率并在处理器高度利用时提高电压/频率来利用处理器利用率的变化。与处理器内核一样，取决于应用程序的通信/数据共享模式，片内存储器中的高速缓存线利用率以及基于NoC的多核架构中的链路利用率也存在很大差异。事实上，通过准确关闭未使用的缓存路径或在链路使用减少时降低链路的频率和电压，可以节省高达10倍的功耗[16]。我们利用DVFS来改变内核和NoC的功率分配。

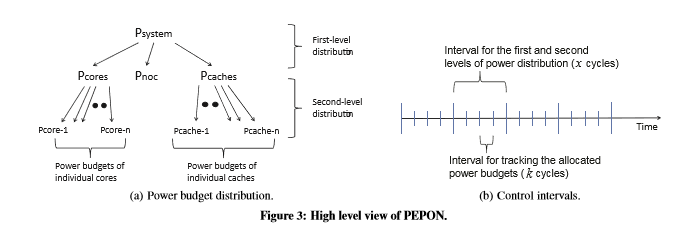
## 3.我们的建议方法

### 3.1PEPON的高级视图

针对图2所示的多核架构，我们在这项工作中的目标是在全芯片功耗预算约束下最大化工作负载性能。 更具体地说，考虑到多核功率预算，我们希望在不同的硬件组件上分配/重新分配，以便（i）不超过给定的功率预算，并且（ii）性能最大化。 因此，我们对最大化工作负载性能但违反指定功耗预算的解决方案不感兴趣。 此外，在满足功率预算约束的所有解决方案中，我们有兴趣寻找最大化性能的解决方案。

图3（a）说明了我们提出的两级功率预算分配策略（PEPON）的高级操作。 在这个策略的第一个层面，整体功耗预算被划分在核心，NoC和L2缓存之间。 在第二级中，分配给内核的功率预算进一步划分到各个内核之间，同样，分配给L2缓存的功率预算进一步划分到各个L2缓存中。 虽然也有可能将NoC划分为电压岛[26]，并将分配给NoC的功率分配到这些岛上，但我们没有在这项工作中寻求这个方向，因为这种划分可能对不同的应用产生非确定性影响。 相反，在这项工作中，我们将NoC视为一个单一的单片实体，它们的所有分量在给定时间具有相同的电压/频率水平（即，当应用电压/频率缩放时，它应用于整个NoC）。

在这个配电策略的每个层面，都采用了不同的方法。 具体而言，对于第一级分布，我们采用基于回归的性能模型，该模型指导我们决定最有效的性能分配。 另一方面，对于二级分发，我们对高速缓存和内核使用不同的策略。 分配给核心的功率预算使用控制理论方法进行分配。 另一方面，对于不同L2银行的电力预算分配，我们采用基于效用的模型。 接下来将介绍这些型号的技术细节以及我们的两级功率分配策略和功率控制旋钮的细节。



### 3.2 PEPON的设计细节

对于第一级功率分配，系统功率预算（Psystem）在不同类型的组件（即，核心（Pcores），缓存库（Pcaches）和NoC（Pnoc））之间进行划分。也就是说，我们有：Psystem = Pcores + Pcaches + Pnoc。在第二级，Pcores分布在核心之间，Pcaches分布在高速缓冲存储区中。每第x百万个周期都会执行第一级和第二级分配。此外，每个感兴趣的组件（每个核心，缓存库和NoC）都有一个与其关联的控制器，它跟踪分配的预算。如图3（b）所示，较长的历元（x百万个周期）被划分为若干个子历元，并且与每个元件相关的控制器每调用一千万个周期（子历元）来跟踪其功率预算，在每x百万次循环中确定。虽然我们用不同的x和k参数值进行了实验，但我们大多数实验中使用的默认值分别为100和20。

#### 3.2.1一级配电

在这个级别上，我们使用了一个基于回归的估计器：

Perf = + a1 × Pcores + a2 × Pcaches + a3 × Pnoc. (1)

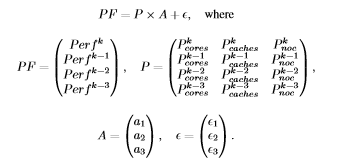
在这个表达式中，Perf是总体工作负载性能，ai（1≤i≤3）表示模型系数， 捕获模型误差值。 当总功率预算划分为该时期的Pcores，Pcaches和Pnoc值时，该模型给出了预计在一个时期内实现的总体性能（Perf）。 在我们的模型中，我们将每个时间间隔的整体工作负载性能（Perf）定义为Pi IPSk i Avg\_IPSi，其中IPSk i是第i个应用在第k个时间间隔的指令每秒（IPS），Avg\_IPSi是平均测量的IPS 该应用程序在执行时没有任何功率限制。 可以通过在具有最高可能频率的核心上运行该应用并且不施加任何功率预算约束来为每个应用获得Avg\_IPSi。

在每个时间间隔内，首先根据测量的性能和先前分配的功率预算对系数进行更新，然后我们的方案增加具有最大系数的组件的功率预算，并降低具有最小组件的功率预算COEF网络cient。例如，如果a1> a2> a3，那么Pcores和Pnoc分别增加δ和减少δ-α1（δ是增量/减量单位，而δ1是一个很小的数字，有助于回归的计算）。



表1：功率预算分配的示例窗口（表格）和相应的性能数字（第一列）。

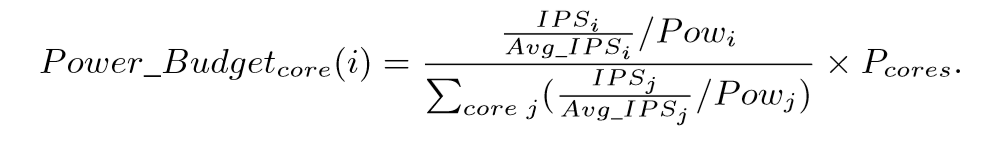
为了更新模型，我们保留一个窗口（表格），其中有四个条目，用于存储功率预算分配和最近四个时期的相应测量性能数字。本表中的每个条目都是（Perf，Pcores，Pcaches，Pnoc）的形式;表1给出了一个示例表。在每个预算分布时期（图3（b）中显示的时间间隔中较长的一个），首先将表中最旧的条目替换为新条目（Perfk，Pkcores，Pkcaches， Pknoc），其中Perfk是上一个历元和Pkcore测量的整体系统性能，Pkcaches和Pknoc分别是分配给内核，缓存和NoC的最近功耗预算。表1中存储的数据和模型系数可以堆叠在一起，写成如下形式：



在更新回归历史之后，我们使用最小二乘法来估计a1，a2和a3值。 在最小二乘法中，回归系数计算如下：A =（PT×P）-1PT×PF。 然后PEPON以最大的系数增加功率预算，并以最小的系数降低功率预算，因为这些值反映了改变相应组件的功率预算对总体性能改进的影响。

#### 3.2.2二级配电

在第二级，我们针对内核和高速缓存采用不同的配电策略。 为了在多核系统中的不同内核上分配Pcores，我们使用[20]中提出的方法。 具体而言，我们根据其在上一个历元上的IPS瓦特（每瓦性能）（IPS瓦特=每秒指令功耗）为核心I分配功率预算。 分配的预算与此参数成比例。 以数学术语来说，如果IPSi和Powi是测量的IPS和在最后一个时期在核心i上运行的应用的估计功耗，并且Avg\_IPSi是该应用在没有任何功率限制的情况下运行时的平均IPS，则功率 分配给核心i的预算将按如下计算：



被用来代替IPSi，因为即使在没有任何功率预算限制的情况下它们被执行时，一些应用固有地具有低IPS号码。我们可以通过在频率设置为最高可能值的核心上运行该应用程序并且不强加任何功率预算约束来获得每个应用程序的Avg\_IPS。请注意，这个限制对每个应用程序执行一次（不适用于工作负载）。以前的工作，如[31]，也使用平均IPS值作为基准。在4.2节中，我们给出了实验结果，表明仅在核心层面的分配能力（如[20]的情况）可能不足以最大化性能。一旦确定了每个核心的功率预算，与每个核心相关联的控制器就会尝试通过调整核心的频率/电压来跟踪预算。如前所述，我们使用DVFS作为我们的旋钮来控制核心的功耗。更多细节在本节后面给出。

为了在不同的缓存库中分配Pcache，我们采用基于效用的策略。更具体地说，每家银行的权力份额与其在上一纪元的效用成正比。我们的目标是为缓存银行分配更多的功耗预算，这可能会为整个系统的性能带来更多的好处。可以使用不同的策略来计算缓存库的效用。作为一个例子，银行的效用可以计算为在上一个时期对该银行的访问总数。然而，这对于功耗预算分配来说不是一个好的指标，因为这些访问可能有一个小的缓存足迹，并且可能不会有更高的效用为银行分配更多的功率预算（请注意，更多的预算意味着更大的银行）。这项工作采用的方法是基于缓存脚印。在这种方法中，我们考虑作为该缓存库的实用程序在最后时期在缓存库中命中的唯一缓存线的数量。为了实现这个方案，一个小比特向量与每个缓存库相关联。该位向量被用作散列表。一旦高速缓存存储区中发生未命中，其位向量中的一位将被设置为'1'。该位使用新数据的标签作为关键字进行索引（该关键字使用高效的硬件功能进行散列）。同时，被驱逐数据的相应位被设置为'0'。因此，在一段时间内，高速缓存存储体的位向量中的'1'表示已被访问的唯一活动高速缓存行的数量。由于高速缓存行在第一次访问后可能不会长时间访问，因此我们在每个时间间隔开始时重置位向量。在PEPON中，每个高速缓冲存储体的功率预算份额与其在最后一个历元上的效用（即，位向量中'1的数量）成比例。

请注意，在我们的系统中，每个核心都运行单一应用程序，而NoC和缓存库是共享的。 因此，纯粹的以应用程序为中心的预算分区对于NoC或高速缓存来说是不可能的。 这就是我们采取以组件为中心的分区策略的原因。

#### 3.2.3功率控制器的设计

在本小节中，我们解释如何分配功率预算

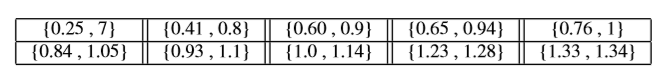


表2：针对核心考虑的{频率（GHz），电压（V）}水平。

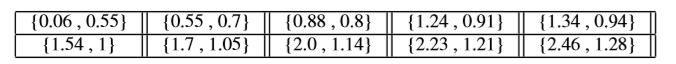


表3：针对NoC考虑的{频率（GHz），电压（V）}水平。

由PEPON进行跟踪由与不同硬件组件相关的控制器进行跟踪。

核心和NoC控制器。如前所述，我们使用内核的频率/电压和NoC作为我们的旋钮来控制这些模块（我们采用DVFS）的功耗。表2和表3给出了我们在DVFS方案中考虑的频率/电压对。一个简单的P（比例）反馈控制器被配置并与每个核心以及NoC相关联来调制频率水平，目标是跟踪分配的功率预算。图4说明了我们的控制器如何操作的高层次观点。我们在PEPON中使用了P控制器，主要有两个原因：（i）由于控制输出是控制误差乘以一个恒定的值，并且（ii）他们很快将系统输出转换为所需的值（我们需要PEPON中的这个属性，因为控制器只有有限的时间间隔以达到所需的功率预算）。在本节中，根据我们的系统模型，我们证明只采用PID控制器的P分量可以提供一个稳态误差为零的快速稳定控制器。

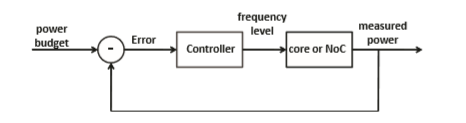


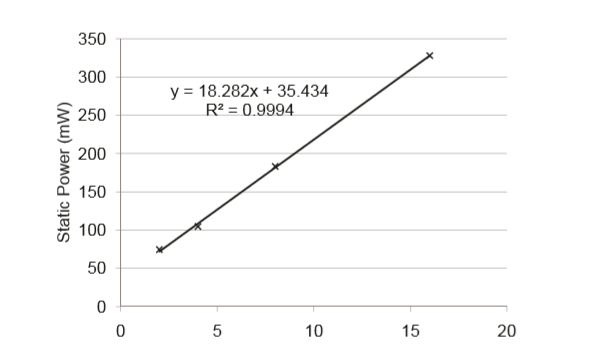
图4：用于调节核心/ NoC功率的反馈控制环路。

在每个时期，控制器比较当前功率预算和最后一个历元上的测量功率，并基于该比较的结果调整（增加或减少）频率水平以最小化测量的误差值（这是 功率预算和实际功率）。 PEPON中的每个控制器执行以下规则来调节频率级别：

f(T + 1) = f(T) + K × (Pbudget − Pcurrent), (2)

其中f（T）是第T个时间间隔的频率，K是一个常数值，Pbudget和Pcurrent分别是指定的功率预算和当前功率。 控制器首先计算f（T + 1），然后将当前频率水平设置为与f（T + 1）最接近的水平。 控制器确保其输出为表2和表3中的可用频率水平之一。从该控制规则可以看出，如果测量功率大于参考功率，控制器会降低频率水平以降低功耗 下一个时代。

配置控制器的下一步是确定K的值。该值是根据系统功率特性计算的。2 P控制器[10]是一种控制器，其中控制器的输出与误差值成比例， 这是目标值和当前系统输出之间的差异。 尽管我们大多数的实验都是使用P控制器进行的，但我们稍后也会使用PID控制器报告结果。



缓存方式的数量

图5：静态功耗与活动（L2）缓存方式的数量关系。（核心/ NoC），当频率水平变化时。 如[24]所示，改变频率对功耗的影响可以线性估计如下：

P2 = P1 + a × ∆F,

其中P1是旧频率消耗的功率，P2是频率增加ΔF后的功耗。 通过对不同的应用和频率进行实验并对所获得的α值进行平均，可以估计α的值。 通过结合等式（2）和（3），我们观察到，如果我们在控制器中将K值设置为1，则频率改变为达到参考（目标）功率预算（在我们的K值为60M 方案）。

控制系统分析。 表征控制系统的主要方法之一是在频域中对其进行分析[10]。 在这种方法中，系统由一个传递函数来表示，该函数显示了频域中输入和输出之间的相关性（z变换用于离散时间系统）。 我们的系统由公式（3）建模，可以表示为Y（z）X（z）= az-1，其中Y（z）是系统的功耗，X（z） z域中的工作频率。 由于我们使用了一个P控制器，我们的闭环传递函数可以通过下式得到：其中R（z）是功率预算目标。

在形式控制理论中，如果闭环传递函数的极点放置在z平面中的单位圆内，则控制系统是稳定的并且在稳定状态下跟踪目标。 通过求解特征方程（将传递函数的分子设置为零）获得传递函数的极点。 因此，可以看出，我们的频率控制系统在z = 1 - K×a处有一个极点。 此外，如前所述，K参数的值被设置为1a，这确保了系统的稳定性。

稳态误差是控制系统的另一个重要指标。该误差被定义为当控制器达到其稳定状态时输入参考值和实际系统输出之间的差值。可以看出，对于步进参考输入，稳态误差等于limz→0 1 1 + H（z），其中H（z）是闭环传递函数。在我们的控制系统中，参考输入是阶跃信号，因为它们立即在两个不同的频率水平之间变化。如果用我们的系统传递函数代替H（z），可以观察到基于数学分析的稳定误差为零。请注意，尽管我们的分析显示了一个精确的控制系统，但在跟踪参考目标时可能仍存在一些不准确之处，这将在我们的实验评估中讨论。这些不精确性可能会发生，因为我们使用线性估计来建模系统（参见公式（3）），并且它不能理想地捕获共同运行程序应用程序的动态行为。

缓存库控制器。 随着技术规模的缩小，高速缓存的静态（泄漏）功耗成为整体功耗的重要组成部分。 静态功耗高度取决于缓存的大小。 举例来说，图5显示了当“活动”缓存方式的数量变化时，用45纳米技术制造的512KB，16路组关联缓存的泄漏功率，该图清楚地表明，数字之间存在线性相关 通过这种观察，在PEPON中，通过关闭缓存路径的“开”或“关”来控制电源在高速缓冲存储器组中，特定地，在每个时期，与每个高速缓存存储器相关联的控制器 比较当前功率和参考功率预算，并且使用图5中的数据决定打开或关闭若干缓存路径，以使实际功率与参考功率之间的差异最小化。

#### 3.2.4实施开支

在硬件支持下，PEPON在操作系统级别的软件中实现。 PEPON施加的其中一项开销是计算个别功率预算的过程。这包括简单的数学和矩阵运算来划分第一和第二层的功率。这些开销很小，因为该方案是用软件实现的，并且每隔1亿个周期执行一次。相比之下，控制器以更高的频率被调用以追踪功率预算。但是，应该注意的是，我们的控制器执行简单的操作（减法和乘法），因为它们被实现为P控制器。因此，我们的控制器造成的额外开销也不是很高。下一节介绍的功率和性能数字包括PEPON产生的所有开销。除了这些软件级开销之外，PEPON还假定底层系统支持DVFS，而DVFS是当今几乎所有处理器的标准。我们对DVFS开销做了与[20]（20us DVFS驱动开销）相同的假设。如果电压/频率值每隔2000万周期（我们的默认值）执行，并且系统以500MHz运行，则产生的DVFS开销（20us）将约为0.05％。此外，PEPON还假定处理器支持软件级别控制，以重新配置（打开/关闭）L2缓存中的缓存路径，并调整NoC的工作频率。

## 4.实验评估

在本节中，我们通过实验评估PEPON，并将其与数量上与备用电源预算策略进行比较。 我们使用SIMICS [21]作为我们的仿真框架，它是一个用于多核的全系统模拟器。 对于我们的实验，SIMICS通过详细的延迟模型以及来自CACTI [29]，Orion [30]和Wattch [3]的功耗模型进行了增强，以估计目标多核架构中不同组件的功耗。

### 4.1功率估计模型

核心的功率模型。 我们使用[24]中的相同模型来估计内核的功耗。 核心消耗的功率几乎线性地取决于核心的效用。 换言之，一旦应用程序开始执行，核心的平均功耗可以基于以下等式估算：Pi = k0×Ui + k1，其中Pi是核心i的功耗，Ui是其利用率，并且 k0和k1是常数值。 请注意，对于不同的应用，k0和k1的值可能会有所不同。 在这项工作中，我们通过独立运行每个应用程序并使用Wattch [3]来监视这些常量的值，以监视执行过程中的功耗。

高速缓存的电源模型。 我们使用从CACTI获得的数字[29]来估计高速缓存的功耗。 每个缓存库的功率可以估算为：



其中泄漏功率线性取决于高速缓存大小。 由于在我们的方案中，我们使用活动缓存路的数量作为旋钮来控制缓存库的功耗，所以可以考虑执行期间每个时间间隔处的活动缓存大小来估计泄漏功率。

NoC的功率模型。 在时间间隔T内NoC的平均功耗估计如下：



其中NHnoc是网络中所有数据包在T上经过的总跳数，Enoc是每跳能量。 在这个表达式中，“每跳能量”是数据包通过路由器时消耗的能量，我们使用从Orion [30]获得的能量值来估计NoC的总功耗。

|  |  |
| --- | --- |
| 处理器 | 具有私有L1数据和指令缓存的6个内核 |
| NoC架构 | 4 × 4 |
| 处理器模型 | 4路问题超标量 |
| 私有L1数据和Instr缓存 | 直接映射，32KB，64字节块大小，3个周期访问延迟 |
| L2高速缓存库的数量 | 16（分布在网络上） |
| L2高速缓存 | 64字节的块大小，10个周期的访问延迟 |
| L2高速缓存行大小 | 每核心512KB |
| 记忆 | 4GB，200个周期的片外访问延迟 |

|  |  |
| --- | --- |
| 功率预算控制执行间隔 | 1亿次循环 |
| 跟踪控制执行间隔 | 20 Million cycles |
| δ | 1 |

表4：基线配置]

表4给出了我们在实验评估中使用的基线配置。在我们的基准配置中，PEPON分配给每个组件的功率预算每隔1亿次计算并执行一次。此外，分配给每个组件的控制器有机会跟踪其超过5千万个​​周期持续时间的5个功率目标。我们使用SPEC CPU 2006基准测试套件[1]构建了六个应用程序混合（工作负载），用于我们的实验评估。我们的每个应用程序混合包含16个应用程序。我们考虑一对一映射，也就是说，在每个实验中，混合的16个应用程序在我们系统的16个不同内核上映射和执行（每个内核一个应用程序）。表5中给出了这六种工作负载。在形成这些工作负载时，我们试图在内存访问行为方面具有多样性。对于每个应用，我们跳过第一个十亿个预热周期，然后仿真继续进行下一个10亿个周期（100个功率预算执行间隔）。

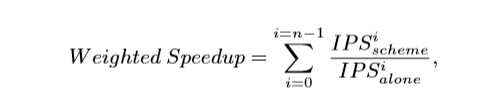
#### 4.1.1绩效评估方案

表6列出了我们在实验评估中考虑的与PEPON进行比较的五种不同方案。在此表中，“动态”表示相应的功率预算随执行而变化。从表6可以看出，对于PEPON，所有条目都显示为“动态”，表明功率预算随着预算划分的第一和第二层的执行而变化。然而，在其他方案（方案1通过4）中，功耗预算在运行时的第一级分区固定（即Pcores，Pcaches和Pnoc固定在表6中给出）。该表还给出了实际的功率预算值（对于高功率和低功率模式），但是在方案1中，核心和缓存库的总功率预算仍然在各个组件之间动态分配，不同组件的所有功率预算在方案2中固定，核心和高速缓存的总功耗预算在它们之间均匀分布;而在方案3和4中，另一方面，在第二级中仅启用一个功率预算分配机制（对于Scheme中的核心3和方案4中的缓存）。请注意，方案3模拟了最近提出的多内核功率预算分区方法[20]。除了这些方案之外，我们还实施了另一种方案，其中第一级的总功率预算是动态分配的，第二级方案中，核心和缓存库的功率预算 - 从第一级开始 - 统计分区均匀地）在各个组件（核心和缓存）之间。我们观察到PEPON的平均性能比最后一种方案提高了约8％。由于空间问题，该方案的详细结果被忽略。

### 4.2结果

#### 4.2.1性能结果

我们在两种功率模式下进行了实验：（i）低功率模式。 在这种情况下，芯片的总功率预算设置为40瓦，（ii）高功率模式。 在这种情况下，总功率预算设置为90瓦。 这些功率预算在执行过程中由PEPON分为两个级别。 对于表6中的其他方案（方案1-4），功率预算划分的第一级在运行时固定的总功率预算分别为每种情况下最大功率值的约80％和40％ 如表6所示。图6（ab）绘出了采用表6中列出的方案时对不同混合物所实现的归一化加权加速值。对于n个应用的加权加速计算如下：



哪里是所采用的应用程序i的方案实现的每秒（IPS）指令，而IPSialone是在没有任何功率限制的情况下单独执行应用程序时测得的IPS。 图6中的加权加速值被归一化到表6中的方案2（假设为基本方案），其中功率预算在执行中不被调制（即，所有功率分布在整个执行过程中被固定）。 请注意，我们考虑IPS而不是每个周期的指令（IPC）来计算加速值，因为我们使用工作频率作为旋钮来控制内核的功耗，并且IPC值不会通过改变内核频率而改变。

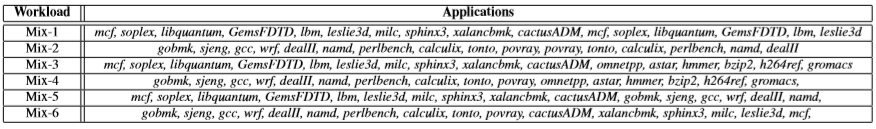


表5：为我们的多程序工作负载考虑的各种应用混合

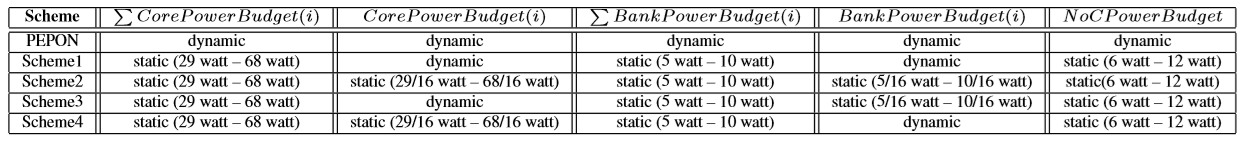


表6：具有固定功率预算值的各种实验方案。

图6（a）和（b）显示，PEPON在低功耗和高功耗模式下均实现了显着的性能（工作负载加速）改进（低功耗情况下超过25％，高功率情况下超过20％）。此外，如图6（c）和6（d）所示，总功耗与功率预算非常接近。这些图表显示，PEPON在第一级别划分总功耗预算对于在某个功耗预算下实现更好的性能起着重要作用。例如，从图6（a）可以看出，虽然通过在Mix-2（方案1,3和4）中的核心和高速缓存存储体之间划分第二级功率预算来改进性能，但它是低于PEPON所实现的改善，因为预算在所有这些计划中都是第一级固定的。我们还注意到，PEPON与方案3（模仿最近提出的方案[20]）之间存在显着差异，进一步强调了两个层面的权力预算。

图6（a）和（b）也显示了不同方案在所有工作负载中实现的平均加速值。 人们可以观察到，PEPON相比于在高功率模式和低功率模式下测试的其他方案（大约21％在低功率模式和17％在高功率模式中），总体工作负载性能显着提高。 另外，可以看出，方案1实现的性能改进高于其他方案（方案2,3和4）。 这是因为这个方案利用了在第二级核心和高速缓存间划分功率预算的优势，而在所有其他被测试的方案（除我们之外）中，功耗预算（核心或高速缓存或者两者）都固定在 第二级。

#### 4.2.2 PEPON的动力学

接下来我们将重点介绍PEPON在执行过程中如何动态划分总功耗预算，以实现图6所示的性能增强。在此，我们将介绍我们的一个工作负载（Mix1）的结果。 剩余的工作量表现出类似的行为。

图7（a） - （c）绘出了当PEPON在Mix-1运行时分配给内核，缓存和NoC的功率预算，当总功耗预算设置为40瓦时。这些图也绘制了不同组件的测量功耗。可以看出，PEPON首先分配第一级的总预算，然后在每个组成功跟踪预算目标。图7（d）绘制了内核，缓存和NoC的累计功耗（该图表显示了总功耗，而不是总功耗预算）。需要注意的是，使用PEPON时，功率预算违规行为非常低（不到2％）;功率预算违规将在稍后进一步讨论。图7（a） - （c）中还有两点需要注意：（i）核心和NoC（图7（a）和（c））的功耗在目标功耗预算周围振荡，在执行过程中预算略有违反，而高速缓存的功耗总是低于预算，并且（ii）高速缓存的功率预算被更快地跟踪。例如，如图7（a）和（c）所示，在时间'0'时，内核和NoC的功率预算与其功耗不同，并且功率预算在一定时间间隔后被跟踪。这两个区别的主要原因是用于高速缓存的控制器的类型与与核心和NoC相关的控制器的类型不同。如前所述，在高速缓存中，功耗主要由静态功耗决定，静态功耗精确模型是已知的（活动方式数量与功耗之间的关系）。因此，与高速缓存关联的控制器能够在短时间内基于模型选择正确数量的活动高速缓存路径，从而不会违反功率预算。另一方面，我们对核心和NoC使用P控制器，其中确切的模型是未知的，并且基于误差值确定频率，导致小的振荡

图8绘出了Mix-1的第一级功率预算分布（类似于图7），但是当我们的总预算为90瓦（高功率模式）时。 正如人们可以观察到的，当我们在高功率模式下运行时，功率预算也被成功分区和跟踪。 我们在功率预算的第一级观察到与我们其他混合体相同的动态行为。 请注意，这些实验中的NoC功率直接通过调制NoC频率/电压电平来控制。 但是，核心和高速缓存的功耗预算不是直接控制的，而是在各个组件之间进一步分配。 因此，跟踪第一级核心和高速缓存的总功耗预算（如该组数据所示）意味着每个单独的组件（核心和高速缓存）已成功跟踪其自身的预算。

图9说明了在第一级确定的功率预算如何在Mix-1执行期间在第二级单独组件（核心和高速缓存）之间进行划分。从这些图中可以看出，总功率预算被划分为16个部分，每部分的大小表示分配给每个部件的功率预算量。由于总预算是根据缓存的效用或最近时间间隔内核的性能功率比（我们在第二级使用的方案）分配的，因此不同的组件会从功耗预算中获得不同的份额。为每个硬件组件配置的控制器分别通过调整核心和高速缓存的电压频率和活动高速缓存路径的数量来跟踪目标功耗预算。为了更好地理解这些功率分布曲线，我们在图10中绘制了核心的选定频率和Mix-1的缓存路数。我们看到这些旋钮的值在特定时间对于不同的组件也不同。例如，在时间10，分配给某些核心的频率与其他核心相比较低。这是因为，不同内核的功耗预算不尽相同。还要注意的是，即使预算相同，不同的内核在执行不同的应用程序时也会消耗不同的能量，因为它们以相同的频率运行。

控制器选择。 正如前面所讨论的，我们采用简单和快速的P控制器来调制核心和频率的频率

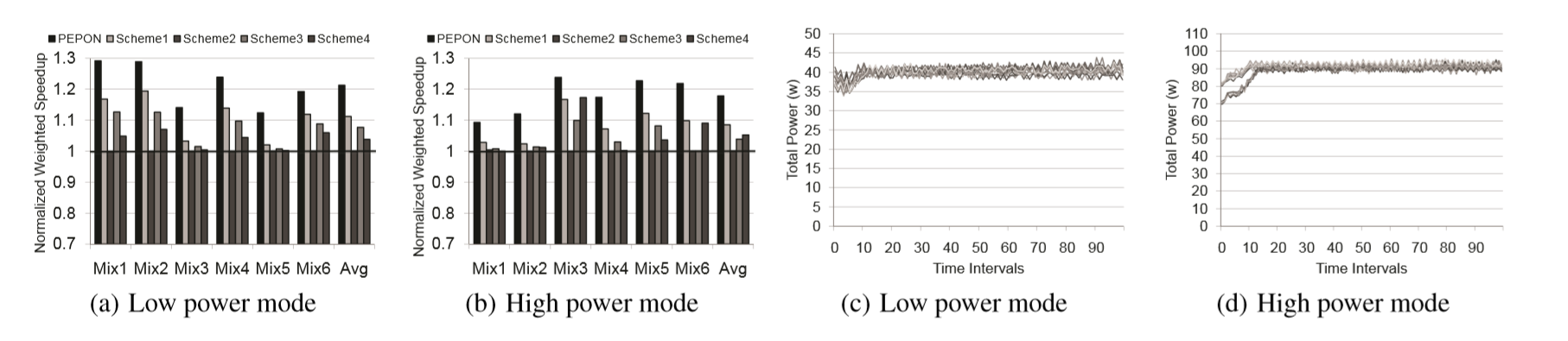


图6：（a-b）在一定功率预算下通过不同方案实现的归一化加权加速值，以及（c-d）总功率预算的动态跟踪。 请注意，（c）和（d）都将所有混合在一起。 这里的要点是，在低功耗情况下，我们的混合芯片的功耗接近40W，同样，在高功率情况下，我们混合芯片的芯片功耗接近90W。

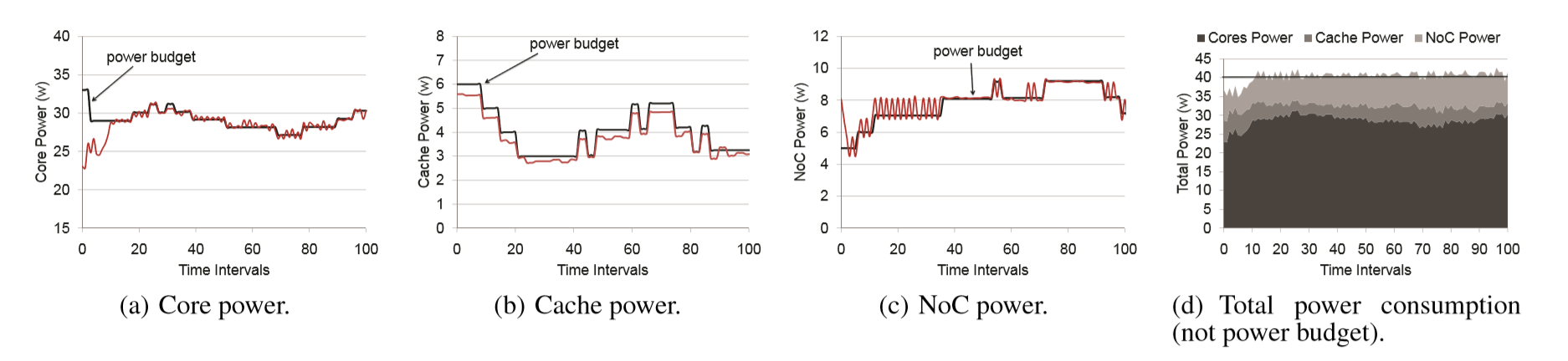


图7：PEPON第一级Mix-1（低功耗模式）的功率预算分区和跟踪。

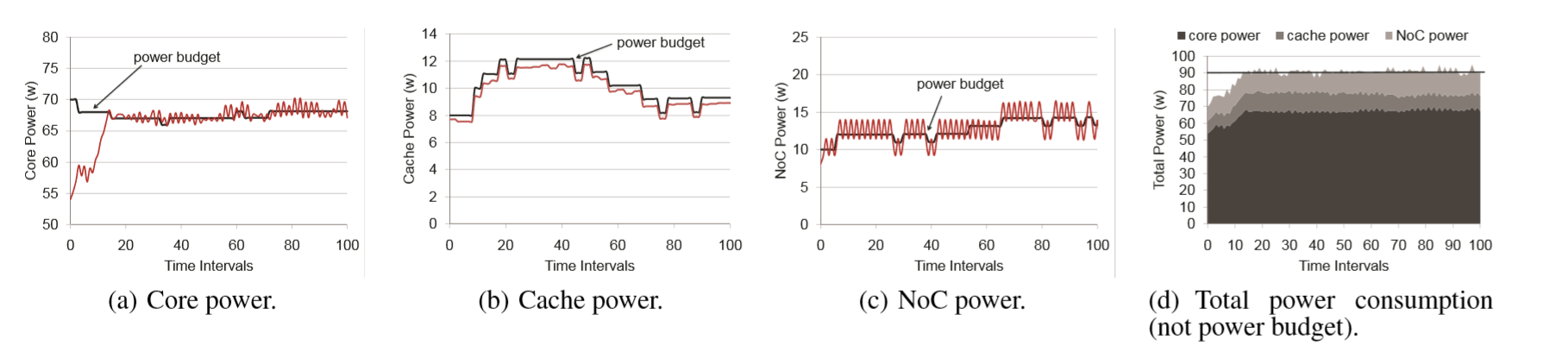


图8：PEPON第一级Mix-1（高功率模式）的功率预算分区和跟踪。

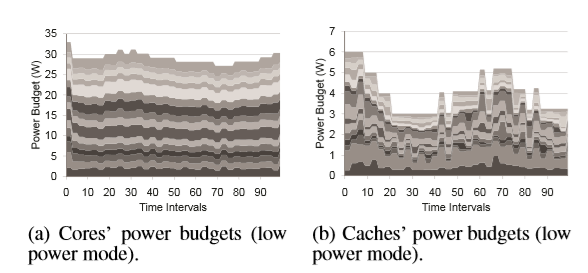


图9：第二级功率预算划分（Mix-1）。 请注意，每个图都绘制了全部16个核心（或缓存）组件。

NoC跟踪指定的功率预算。 可以建议使用更复杂的PID控制器来进行频率调制，而不是P控制器。 采用PID控制器的主要优点之一是它们可以将稳态误差驱动为零。 如前所述，在我们的系统中，使用P控制器时此误差已为零。 图11绘出了P和PID控制器在执行过程中如何跟踪功率预算（分配给Mix-1实验中的一个内核）。 从图中可以看出，P控制器能够跟踪目标以及PID控制器。 然而，PID控制器在某些时间间隔（例如，在第30个时间间隔）稍微减少振荡次数。 在流程侧，P控制器能够更快地响应目标变化（例如，在第70个时间间隔）。 另外，与PID控制器相比，P控制器的开销更小。

#### 4.2.3灵敏度实验

我们还测量了Mix-1（在低功率模式下）对不同参数的灵敏度。 如前所述，在每个时期，功率预算分别在功率预算分区树的第一级增加和减少δ和δ - Δ1。 回顾表4，在我们的实验评估中δ的默认值是1。 图12绘出了当δ值设置为2（为便于比较而重现值为1的结果）时Mix-1实现的加速。 我们发现增加δ值对PEPON实现的加速有负面影响。 这是因为，功率预算可能会围绕最优值振荡，而不会通过具有更大的δ而达到最佳值。

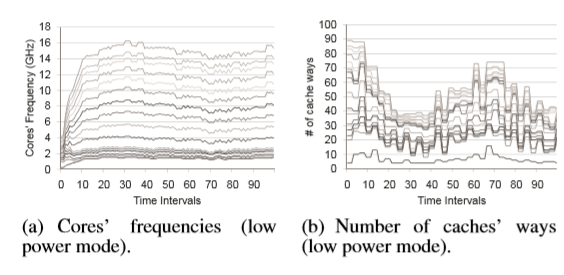
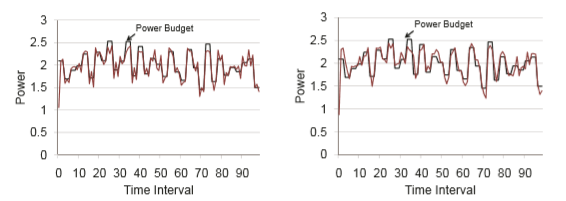


图10：核心频率的调制和执行中活动缓存路的数量（Mix-1）。 请注意，曲线堆叠在一起以获得更好的可视性。 还要注意，激活的（缓存）路径的累积数量低的时间间隔对应于缓存未被分配多少功率的时间段。



（a）P控制器 （b）PID控制器

图11：采用P和PID控制器

我们对不同高速缓存大小的实验表明，我们的加速对高速缓存区大小不是很敏感，因为在这种情况下，由于分配给高速缓存区的预算较低，即使有更多的高速缓存空间可用，也只有一部分是 在运行时使用。 影响PEPON实现性能改进的参数之一是制定和执行控制决策的频率（我们的时期大小）。 我们的实验表明，对于小纪元尺寸所实现的加速通常较高。 这是因为，在这种情况下，PEPON被更频繁地调用，并且正在运行的应用程序的行为以更高的频率被监视; 因此，我们最终可能会获得更好的性能改进。 但是，这也意味着更多的开销。

### 4.3功率预算违规

从图7（d），8（d）可以看出，总体功率预算不能始终以100％的准确率跟踪，并且在执行过程中被违反（尽管这种违规通常非常低）。此外，在某些情况下，预算没有得到充分利用。这是因为分区树底部各个组件的功耗（见图3（a））并不总是与其功耗预算完全相同。造成这种情况的原因有两个：（i）控制器使用的旋钮只能从一组离散的水平取值，不能连续变化。例如，核心的频率/电压只能设置为从表2中选择的一个离散值。因此，控制器选择的频率可能不是达到功率预算所需的确切频率; （ii）当控制器的目标功率预算被调制时，控制器追踪新目标需要时间，并且在此期间输出功率可能波动。即使预算是固定的，控制者也需要补偿潜在的行为变化

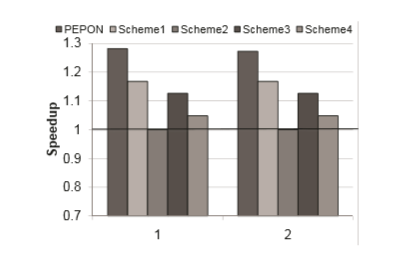


图12：通过改变配置参数的默认值实现了Mix-1的性能改进。

运行应用程序 可以处理可用功率预算的这些最低利用率的方法之一是通过动态地改变参考预算。 例如，如果我们利用前一个时间间隔的可用预算x％，则可以将参考预算值增加x％。 调查这个和类似的补偿计划是我们未来的研究议程。

## 5.相关工作

作为电源管理工具的控制理论最近获得了普及，但大多数先前的以控制为中心的工作并未考虑芯片级电源管理，该管理考虑了内核，片上高速缓存和底层NoC。 我们已经将我们的工作与[20]进行了定量比较，并表明我们的建议更好，因为它是一种更全面的方法，可以考虑片上内核，缓存和NoC。 在本节中，我们将讨论与我们提案最相关的以前的作品。

### 5.1基于反馈控制的电源管理

[24]提出了一种多时钟域架构中的电源管理控制理论框架，其中每个域都由一个单独的时钟控制。在这项工作中，提出了两层电源管理架构，第一层为每个本地岛分配一个目标预算，第二层控制电源。在这种情况下，解决方案变得更简单，因为它隐含地假定域中的所有内核（及其高速缓存）都将运行类似的应用程序，即特定域中的所有内核都将承载CPU密集型应用程序或内存密集型应用程序。这允许使用每个域时钟的整个域的DVFS。与此体系结构相比，我们的解决方案针对实际的CMP，包括核心，共享缓存和NoC，其中域级时钟管理功能非常强大。为了解决这个问题，我们的建议有两级功率控制：第一级在核心，共享缓存和NoC之间分配功率预算，第二级在不同核心之间重新分配核心功率，以及不同高速缓存之间的缓存功率。其他有关电力和热管理控制理论的相关着作包括[6,15,31,32,33]。在[6]中，提出了一种基于PI的核心热控制器，并使用外部控制回路来决定热突发事件的过程迁移。 [15]中的提案扩展了[33]，并提出了该方案的分布式版本，将正式方法的基本机制应用于中医。在文献[31]中，提出了一种分级控制体系结构，用于控制大型数据中心的总功耗，使其保持在配电容量所限制的范围内。但是，这项工作仅在处理器内核模型中进行电源管理。在[32]中，Wang et al。提出了一种用于CMP温度管理的多输入多输出解决方案。我们的建议与[32]中的工作之间的一个主要区别是，尽管我们的方法简单灵活（实际上灵活性策略是我们提出的方法背后的主要设计时间考虑之一），[32]使用复杂的控制器，其复杂性与芯片中的内核数量成正比。因此，很难将其应用于真正的大型CMP。此外，与[20]中的工作相比，我们的建议针对CMP中的所有主要组件（核心，缓存和NoC），而[20]仅针对核心中的功率控制。

### 5.2 Ad-hoc电源和散热管理

有许多关于电源和热管理方案设计的先前建议[5,9,13,18,22,25,28]。这些作品突出了在电力和热力约束下适应工作负荷行为的重要性。但是，大多数这些作品在操作系统中采用了独立的本地方案组合或调度决策。 Meng等人。 [22]提出了一种适用于多核电源管理的多优化策略，并通过可配置处理器内核的运行时适应解决了满足全局芯片级功耗预算的问题。 Isci等人[13]也使用了一个类似于我们的全局电源管理器，它提供了一种机制来定期检测芯片的每核功率和性能状态;它然后设置每个核心的工作功率级别或模式，以强制遵守已知的芯片级功耗预算。然而，用于执行这些功率预算的本地监视基于开环控制，并且不提供与我们工作中的正式反馈控制相同的稳健性。此外，对于可实现性能的限制并不像我们在工作中那样执行每核心DVFS而不是核心，缓存和NoC。像[2，8，27]之前的建议通过动态重新配置高速缓存层次结构或在多个应用程序之间对共享高速缓存进行分区，仅考虑了缓存的节能机制。像[7]这样的工作已经调查了存储器单元尺寸以获得缓存能量效率。我们的工作也使用高速缓存重新配置进行能源管理，但我们这样做是为了管理整个CMP的能耗，而不是单独的片上高速缓存。

## 6.结束语

本文的主要贡献是针对在特定功率预算下最大化系统性能的两级配电策略。 从使用从30个应用程序中选择的16个应用程序形成的6个工作负载的实验中收集的结果清楚地表明了两个层面上配电的重要性。 具体而言，我们的实验结果表明，使用PEPON时，与不采用配电时相比，整体性能可提高多达29％。 此外，我们的结果还表明，仅在核心层面分配权力（如最近的研究[20]所做）可能不足以实现绩效最大化。

## 7.参考

[1] http://www.spec.org/cpu2006/.

[2] R. Balasubramonian, et al. Memory hierarchy reconﬁguration for energy and performance in general-purpose processor architectures. In MICRO’00.

[3] D. Brooks, et al. Wattch: a framework for architectural-level power analysis and optimizations. In ISCA’00.

[4] T. Burd and R. Brodersen. Design issues for dynamic voltage scaling. In ISLPED’00.

[5] J. D. Davis, J. Laudon, and K. Olukotun. Maximizing CMP throughput with mediocre cores. In PACT’05.

[6] J. Donald and M. Martonosi. Techniques for multicore thermal management: classiﬁcation and new exploration. In ISCA’06.

[7] R. G. Dreslinski, et al. Reconﬁgurable energy efﬁcient near threshold cache architectures. In MICRO’08.

[8] S. Dropsho, et al. Integrating adaptive on-chip storage structures for reduced dynamic power. In PACT’02.

[9] M. Gomaa, et al. Heat-and-run: leveraging smt and CMP to manage power density through the operating system. In ASPLOS’04.

[10] J. L. Hellerstein, et al. Feedback Control of Computing Systems. 2004.

[11] Y. Hoskote, et al. A 5-GHz mesh interconnect for a teraﬂops processor. IEEE Micro, 27:51–61, September 2007.

[12] J. Howard, et al. A 48-Core IA-32 Message-passing processor with DVFS in 45nm CMOS. In ISSCC’10.

[13] C. Isci, et al. An analysis of efﬁcient multi-core global power management policies: Maximizing performance for a given power budget. In MICRO’06.

[14] N. K. Jha. Low power system scheduling and synthesis. In ICCAD’01.

[15] P. Juang, Q. Wu, L.-S. Peh, M. Martonosi, and D. W. Clark. Coordinated, distributed, formal energy management of chip multiprocessors. In ISLPED ’05.

[16] J. Kim and M. Horowitz. Adaptive supply serial links with sub-1v operation and per-pin clock recovery. In ISSCC’02.

[17] J. F. Kurose and K. W. Ross. Computer Networking: A Top-Down Approach Featuring the Internet. Addison Wesley, 2003.

[18] J. Li and J. F. Martínez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In HPCA’06.

[19] S.-H. Lim, et al. Migration, assignment, and scheduling of jobs in virtualized environment. In ATC’11.

[20] K. Ma, et al. Scalable power control for many-core architectures running multi-threaded applications. In ISCA’11.

[21] P. S. Magnusson, et al. Simics: A full system simulation platform. Computer, 2002.

[22] K. Meng, et al. Multi-optimization power management for chip multiprocessors. In PACT’08.

[23] Y. Meng, et al. On the limits of leakage power reduction in caches. In HPCA’05.

[24] A. K. Mishra, et al. CPM in CMPs: Coordinated power management in chip-multiprocessors. In SC’10.

[25] M. Monchiero, et al. Power/performance/thermal design-space exploration for multicore architectures. IEEE Trans. Parallel Distrib. Syst., 19:666–681, May 2008.

[26] U. Y. Ogras, et al. Voltage-frequency island partitioning for GALS-based networks-on-chip. In DAC’07.

[27] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In MICRO’06.

[28] J. Sharkey, et al. Evaluating design tradeoffs in on-chip power management for cmps. In ISLPED ’07.

[29] S. Thoziyoor, et al. CACTI 5.1. Technical Report, Hewlett-Packard Company, 2009.

[30] H. Wang, et al. Orion: A power-performance simulator for interconnection networks. In MICRO’02.

[31] X. Wang, et al. SHIP: Scalable hierarchical power control for large-scale data centers. In PACT’09.

[32] Y. Wang, et al. Temperature-constrained power control for chip multiprocessors with online model estimation. In ISCA’09.

[33] Q. Wu, et al. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. SIGARCH Comput. Archit. News, 32:248–259, October 2004.