第一章 绪论

1.1 研究背景和意义

1.1.1 集成电路简介

集成电路（Integrated Circuit，简称IC）是一种微型电子器件或部件。通过采用一定的工艺，把一个电路中所需的晶体管、电阻、电容和电感等元件及布线互连在一起，制作在一小块或几小块半导体晶片或介质基片上，然后封装在一个管壳内，成为具有特定电路功能的微型结构。其中所有元件在结构上已组成一个整体，使电子元件向着微小型化、低功耗、智能化和高可靠性方面迈进了一大步。

集成电路具有体积小、重量轻，引出线和焊接点少，寿命长，可靠性高，性能好等优点。同时成本低，便于大规模生产。时至今日，集成电路已经在各行各业中得到了广泛的应用，我们日常用的智能洗衣机、冰箱、电脑、空调等家电设备，再到军事、通讯、建筑等行业用到的几乎所有电子设备都用到了集成电路。可以说，集成电路是现代信息社会的基石[1]。

1.1.2 摩尔定律

自1958年美国德克萨斯仪器公司发明全球第一块集成电路后，随着硅平面技术的发展，20世纪60年代先后发明了双极型和MOS型两种重要电路，使得集成电路产业一直呈现飞速增长的态势。针对集成电路的发展情况，英特尔创始人之一戈登摩尔在1965年首次提出摩尔定律[2]，预言半导体芯片上集成的晶体管和电阻数量将每年增加一倍。后来，摩尔在1975年根据当时的实际情况对摩尔定律进行了修正，也就是现在常说的摩尔定律：集成电路芯片上所集成的电路的数目，每隔18到24个月就翻一倍，性能也将提升一倍。这代表着信息技术和集成电路产业发展的速度。

1.1.3 Dennard Scaling

登纳德缩放比例定律（Dennard Scaling，又叫MOSFET Scaling）是1974年由Robert H. Dennard等人提出来的。它的主要内容是：当集成电路中晶体管变得越来越小，单位面积的晶体管数随尺寸缩小按接近平方关系增长，但它的功耗密度（单

表 1.1 登纳德缩放比例定律

器件或电路参数	缩放因子
器件尺寸（T_{ox}, L, W）	$1/k$
掺杂浓度（Na）	k
电压（V）	$1/k$
电流（I）	$1/k$
电容（eA/t）	$1/k$
时间常数（VC/I）	$1/k$
功耗（VI）	$1/k^2$
功耗密度（VI/A）	1

位面积功耗）不变，原因是晶体管中的电压和电流随着晶体管尺寸一起缩放。如表1.1所示[3]。

登纳德缩放比例定律和摩尔定律一起指导了集成电路的发展近 30 年，在这 30 年中，芯片性能随工艺稳定上升，芯片面积越来越小，单位面积功耗又一直不变，芯片产业一片繁荣昌盛。直到 2006 年左右，登纳德缩放比例定律提前终结了。原因主要是因为当晶体管缩小到一定尺寸时，晶体管的动态功耗足够小，而此时的泄漏电流将不可忽略，由此导致的静态功耗（Leakage Power）占总功耗的比例增大。而静态功耗由于晶体管阈值电压的限制，不随着晶体管尺寸的缩放而同比减小，最终导致单位面积的功耗持续上升。

1.1.4 Dark Silicon

登纳德缩放比例定律的终结，导致单位面积的功耗持续上升，给芯片的散热带来了巨大的挑战。芯片单位面积功耗过高将导致芯片工作性能不稳定，影响芯片寿命甚至直接烧毁芯片。

为了满足芯片散热需求，现在的集成电路芯片在正常工作时，我们通常使其一部分电路关闭，只让一部分电路工作以减少芯片功耗来满足芯片散热需求。这部分关闭的电路区域就叫做暗硅（Dark Silicon）。不同研究表明，在 8nm 工艺节点，暗硅区域所占比重，根据处理器结构、冷却技术和工作负载不同，甚至可以达到 50% 到 80%[4]。也就是说由于功耗和散热方面的限制，芯片的性能将大打折扣。这也告

诉我们，芯片性能的提升不再像过去那样通过缩小晶体管尺寸增加集成度就行，我们要更多地考虑功耗带来的限制和瓶颈。

1.1.5 功耗管理技术

芯片功耗过高除了带来大面积的Dark Silicon区域影响性能外，还带来很多其他问题。概括起来主要有以下几点[5]：

1. 电路的可靠性。就像前面说的，过多的热量导致芯片工作温度升高，严重影响系统的可靠性。过高的温度使连线电阻增大，从而加大了线上延时，可能超出时序的要求。温度升高还会导致漏电流增大，导致电路工作状态改变。

2. 芯片封装成本。芯片功耗过高使芯片温度升高，不能再采用成本较低的塑料封装，必须采用高成本的陶瓷封装来使得芯片不会被烧毁，增加了封装成本。

3. 芯片测试成本。芯片在测试期间会消耗巨大的功耗（如进行老化测试），为了保证在测试时不会烧坏芯片，会通过昂贵的封装和散热装置来降低芯片温度。很显然，功耗过高的芯片为了达到测试覆盖率会增加测试成本。

4. 移动设备电池容量的限制。对于移动设备来说，如手机，笔记本电脑，都是依靠电池供电的，电池的容量有限，芯片功耗过高会直接降低整个移动设备的供电时间。

为了应对芯片功耗过高带来的种种问题，一个常用的办法是对芯片进行功耗管理。功耗管理的目的是在满足芯片性能需求的同时，尽可能地降低功耗。在移动电子设备中，功耗管理技术显得尤为重要。现如今，已经有越来越多的功耗技术问世。其中，应用最广泛的有时钟门控技术（Clock Gating），电源门控技术（Power Gating）和动态电压频率调节技术（Dynamic Voltage and Frequency Scaling，简称DVFS技术)。其中，时钟门控技术通过在电路中增加额外的逻辑单元、优化时钟树结构来节省能量[6]。电源门控技术通过关掉当前电路中空闲的模块，从而节省能量[7]。DVFS技术通过动态调节当前芯片的电压（V）和频率（F）来节省能量。通常在一个系统中，我们结合使用多种低功耗技术来降低功耗。

本文主要采用动态电压频率调节技术（DVFS）来对系统中的用电大户CPU芯片进行功耗管理。为了在节省能量的同时兼顾性能，我们以能量和延迟的乘积（Energy Delay Product，简称EDP）作为优化目标，通过gem5和McPAT仿真工具的

结合使用得到相关数据，引入机器学习领域的支持向量机（SVM）来训练得到决策模型（DVFS 策略），最终将决策模型应用到 gem5 的 DVFS 功能中，实现功耗管理的目的。

1.2 相关研究

我们的研究目的是建立合理的 DVFS 策略进行 CPU 功耗管理，在减少 CPU 能耗的同时兼顾性能。DVFS 策略关注的是 CPU 在什么时间以什么速度运行负载。早期的研究采用基于时间片的策略。Weiser 等人[8]最早提出把时间分为固定长度的时间片，在各个时间片设置执行速度保证在时间片末尾完成任务。稍后 Chan 等人[9]改进了 Weiser 的策略，把它分为两部分，即预测和设置。一个时间片开始时，预测部分预测这个时间片 CPU 的工作量，然后设置部分利用这些信息设置该时间片内的速度。但是其共同缺陷是要求知道未来的 CPU 利用率，以此得出最优的调度，所以只有理论上的意义。稍后 Pering 等人[10]及 Grunwald 等人[11] 都指出了这个缺点，并提出了他们的预测模型和设置策略。他们比较了自己的以及另外的一些策略，得出的结论是悲观的，认为不可能做到既节约能量又不损害性能[12]。

Pering 等人及 Grunwald 等人在比较上述策略时提出了时限的想法，认为若一个任务的时限能得到满足，那么运行快慢无关紧要，从而引出了基于任务的 DVFS 策略。它的关键在于怎样获取各个任务的时限与周期需求，从而求得各个任务运行中的最佳电压和频率设置。Flautner 等人[13, 14]、Lorch 等人[15-17] 和 Yuan 等人[4]分别在解决这些问题时提出了自己的策略，其主要不同体现在对任务分类与动态周期需求探测所用的方法不同[18]。

基于任务的 DVFS 策略需要获取各个任务的时限和周期需求。南加州大学的 Kihwan Choi 等人[19, 20]提出将 CPU 的工作任务分为片上（on chip）和片下（off chip）两部分，片上任务指的是 CPU 需工作的时钟周期数，片下任务指的是外部内存操作所需的时钟周期数。通过比较两部分任务量来制定 DVFS 策略。更多的实验结果表明，DVFS 技术实质上是在 CPU 较空闲的时候（此时可能的情况是任务少或者进行访存密集型任务）降低 CPU 的工作电压和频率以达到节省电量的目的。IBM 公司 Waston 研究中心的 Canturk 等人通过利用 Sherwood 等人提出的 Phase [21-23] 概念，将程序按照某些特征（特征用机器学习的 K-means 方法提取）分为多个 Phase，给每个 Phase 选择最佳的电压和运行频率，通过预测下个 Phase 的种类来判断 CPU 下阶段的频率和电压[24]。

1.3 本文的工作

本文主要研究基于任务的DVFS策略。将程序按固定指令数平均划分为程序段，对每个程序段，使用gem5仿真工具以多种频率运行，通过gem5和McPAT仿真工具的结合使用得到各频率运行时CPU的相关统计数据和功耗情况。在多种频率中，选择EDP最小的一组数据所对应的频率作为最佳频率。这组数据构成一个用来后续学习的样本。多个程序段可以获得多个这样的样本，一起组成样本集合（样本集合中每组数据对应的频率选择都是最佳的）。得到样本集合后，引入机器学习领域的支持向量机（SVM）来对样本数据集合进行学习和训练，得到DVFS的决策模型（该模型能选择下阶段最佳运行频率），最终将此决策模型集成到gem5的DVFS功能中，验证其优越性。

本文的贡献主要有两点。一是平台的创新。前人的研究都是在实际的硬件平台上，提出的DVFS策略大多是平台相关的，可移植性差。本文中借助gem5仿真工具，可以模拟配置各种CPU结构和计算机构架（ARM，X86，Alpha），训练得到的决策模型直接通过文件形式集成到gem5的DVFS功能中，并可以通过参数配置进行多种平台的实验验证，可移植性强。二是方法的创新。本文引入了机器学习领域的支持向量机，通过机器学习的方法建立预测模型，获得了高的预测准确度。

1.4 本文组织结构

本论文后续部分的组织结构如下：第二章介绍了DVFS技术的原理，并对DVFS技术决策模型、延迟和调节粒度，技术实现等进行了说明。第三章详细介绍了研究过程中用到的支持向量机，包括其原理，应用和实现。第四章是本文的重点，讲述了研究的方法及具体流程。第五章是实验及结果分析。第六章是是总结与展望。

第二章 DVFS技术

2.1 DVFS 技术原理

在芯片设计中，功耗一直是备受关注。CPU 芯片的功耗计算有如下关系式，

$$P = P_{Dynamic} + P_{Static} = \alpha * C * f * V_{dd}^2 + I * V_{dd} \tag{2.1}$$

其中，$P_{Dynamic}$ 是动态功耗，P_{Static} 是静态功耗，α 是活动因子（Activity Factor），C 是电路中所有晶体管的等效电容，f 是电路时钟频率，V_{dd} 是电路的供电电压，I 是静态电流。处理器的功耗 P 分别与频率 f 和电压 V_{dd} 的二次方成正相关。而处理器的运行频率 f 越快，需要提供的电压 V_{dd} 也越高，两者也可看成是正相关关系。通过降低处理器的运行频率 f 和电压 V_{dd}，处理器的总功耗将以大约三次方的同比速度降低。DVFS 技术就是通过调节处理器的频率 f 和供电电压 V_{dd} 来达到节省功耗的目的。

2.2 DVFS 调节流程

DVFS 技术调节电压和频率的流程如下：

1. 采集与系统负载相关的信号，计算当前的系统负载。

2. 根据当前的系统负载，预测系统在下一时间段需要的性能。

3. 将预测的性能转换成需要的频率，从而调整芯片的工作时钟。

4. 根据新的频率计算相应的电压。通知电源管理模块调整给 CPU 的电压。

在调节频率和电压时，频率和电压一般为一一对应的关系。同时要注意电压和频率调节的顺序。当频率由高到低调整时，应该先降频，再降电压；相反，当升高频率时，应该先升电压，再升频率。因为，在具体的电路实现中，时钟频率的升高需要高电压保证电容翻转时间的缩短（电流增大，充电时间变短），以此减少周期时间，提升频率。反之，需要先降频，减轻频率对电压的需求，才能再降低电压。

2.3 DVFS 决策模型

DVFS 技术的合理应用能显著减少 CPU 的整体功耗。但是，降低处理器频率（f）和电压（V）势必会降低处理器的运算性能，导致运算时间变长。能耗是功耗与时间的乘积，不合理的 DVFS 调节有时候可能导致更多的能耗，这是我们所不想看到的。所以，如何合理运用 DVFS 技术，准确预测下阶段应选择的电压（V）和频率（f）成为了问题的关键。也就是说我们需要找到合理的 DVFS 决策模型来预测得到下阶段应该选择的电压和频率。

在 Linux 系统中，内核提供了名叫 cpufreq 的模块通过 sysfs 提供的用户接口动态调节 CPU 运行频率。现在的 Linux 版本中默认提供五种决策模型供用户使用，它们分别是 powersave，performance，userspace，conservative 和 ondemand。其中，powersave 模式会将 CPU 工作频率固定在其支持的最低频率上，来最大程度地节省功耗。performance 模式会将 CPU 工作频率固定在其支持的最高频率上，使得处理器性能最佳。这两种模式都属于静态调节。选择 userspace 模式时，系统将变频策略的决策权交给了用户应用程序，并提供了相应的接口供用户应用程序调节 CPU 运行频率使用。conservative 模式和 ondemand 模式都是通过实时观察系统运行负载的变化，动态调整 CPU 频率，只是两者对频率的调整策略略有不同，ondemand 模式更加激进，conservative 模式趋于保守。

虽然 Linux 系统中已经有默认的 DVFS 决策模型了，但是基于各厂商生产的 CPU 硬件的不同，各 CPU 支持的频率不同以及系统间的不兼容性等，任何 DVFS 决策模型都不能通用到所有的 CPU 中，所以需要一个仿真平台针对不同 CPU 类型进行仿真实验探索得到其最佳 DVFS 决策模型。

2.4 DVFS 延迟时间

DVFS 技术通过调节电压和频率来节省能量。在实际电路中通常是通过 VR（Voltage Regulator）执行实际的电压调节操作，很显然，这个过程中会需要一段时间才能完成调节操作，我们把这段时间称作 DVFS 延迟时间。DVFS 延迟时间的长短与电路结构有关系（一般为几十微妙[25]），它的存在会在一定程度上影响 DVFS 策略的制定：如果 DVFS 延迟时间很短，我们每次调节所需的代价小，可以进行更频繁地检测和调节；而如果 DVFS 延迟时间很长，每次调节的代价相对较大，我们做出 DVFS 调节的门槛可能需要变高，只有在确认有足够大的收益时才进行 DVFS 调

节。另外，也有很多电路方向的研究人员在不断改善电路结构，改进VR的实现方式，减少DVFS延迟时间。所以，在制定DVFS策略时，应结合实际的电路情况考虑DVFS调整时间间隔。

2.5 DVFS调节颗粒度

DVFS调节颗粒度分为时间和空间两个方面。

时间上的调节颗粒度指的是DVFS调节的频繁程度，多少个时钟周期进行一次检测和调节。一般细颗粒度的调节能够取得更好的节省能耗的效果，但是太过频繁的检测和调节会造成系统额外的负担，开销增大，而这种开销和本来的程序执行并没有什么直接帮助，造成资源浪费。同时应该考虑DVFS的延迟时间，选择适宜的调节颗粒度。

空间上的调节颗粒度指的是DVFS技术调节的精细度。比较粗糙的调节方式是整个系统共用一个时钟系统，进行统一调节。比较细颗粒的是实现每个核的DVFS调节，每个核采用单独的时钟系统。更细颗粒的调节是将核中的各部件分开，每个部件进行单独的DVFS调节。现阶段，英特尔公司已经在新一代处理器构架Haswell上实现了每个核的DVFS调节技术，取得了更好的功耗控制效果。但是要知道，实现空间上细颗粒度的调节，需要额外的时钟系统和VR等硬件开销，还增加了布线的复杂度，需要解决各部件时钟不同步的问题等，很显然增加了整个芯片设计的复杂度和硬件成本。

2.6 DVFS技术实现

研究中，我们选择了计算机体系结构领域的gem5 仿真器作为仿真工具来仿真模拟实现DVFS技术。gem5仿真器结合了M5和GEMS中最优秀的部分，是一款高度可配置，集成多种ISA和多种CPU模型的体系结构模拟器[26]。

gem5支持两种不同的系统模式：SE（System Emulation）和FS（Full System）模式。其中SE模式能够仿真大部分操作系统级服务，能够取得很好的功能模拟加速比。FS模式模拟完整的全系统，包括操作系统，能够精确模拟系统各种开销。通过gem5仿真器，我们可以配置各种CPU模型和具体参数，用配置好的CPU模拟真实CPU的执行过程，通过运行编译好的程序得到输出结果报告。结果报告中将包含CPU运行时间，时钟频率，各级Cache访问数，Cache miss数，ipc，运行指令数等

信息。

gem5 仿真器是开源的项目工具，个人可以在源码基础上修改添加新的功能。本研究在 2015年4月份发布的 gem5 稳定版本（stable_2015_04_15）中进行修改和实验。gem5 在新的版本中的 FS 模式下加入了 DVFS 功能[27]，但由于研究需要，我们是在 SE 模式下进行实验。通过修改源代码，我们实现了 SE 模式下的 DVFS 功能，并将训练得到的新的 DVFS 决策模型通过 python 接口集成到 gem5 中，具体过程将在第四 章中详细说明。

2.7 本章小结

本章介绍了 DVFS 技术的原理及其决策模型的重要性，探讨了 DVFS 延迟时间和调节颗粒度对决策模型的影响，最终简单介绍了研究过程中实现 DVFS 技术的方法。

第三章　支持向量机

支持向量机（Support Vector Machine，简称SVM）是Corinna Cortes和Vapnik等于1995年首先提出来的[28]，是机器学习领域十大算法之一，被广泛运用到模式识别等各种分类问题中。早期的工作可追溯到1963年，Vapnik在解决模式识别问题时提出了支持向量方法，这种方法从训练集中选择一组特征子集，使得对特征子集的划分等价于对整个数据集的划分，这组特征子集就被称为支持向量（Support Vector），这也是支持向量机名字的由来。

简单地说，支持向量机是一个有监督的学习模型，通常用来进行模式识别、分类、以及回归分析。它在解决小样本，非线性及高纬模式识别中表现出许多特有的优势，并能够推广应用到函数拟合等其他机器学习问题中。

3.1　支持向量机的基本原理

支持向量机首次提出来是为了解决模式识别问题的，是一个分类方法，所以我们先从简单的模式识别问题入手对其加以说明，此处主要参考了文献[29, 30]。

我们先从最简单的二分类问题开始。假设现有的数据集$\{x_1, x_2, ..., x_p\} \subset R^n$中的$x_i$由两类组成，分别标记为1和-1。从中取出$m$个样本作为训练集:

$$(x_i, y_i), \quad i = 1, 2, ..., m. \quad (m \leq p) \tag{3.1}$$

这里$y_i = 1$或-1，代表此数据点的标签类。我们的目的是依据这个训练集的数据，构造一个判别函数，使其能够正确判定出数据点所在的类别。分三种情况讨论。

3.1.1　线性可分

对于线性可分的数据集（见图3.1），直观上很容易理解。它的定义是，如果存在超平面$\omega^T x + b = 0$，使得

$$\begin{aligned} \omega^T x + b \geq 1, \quad y_i = 1 \\ \omega^T x + b \leq -1, \quad y_i = -1 \end{aligned} \tag{3.2}$$

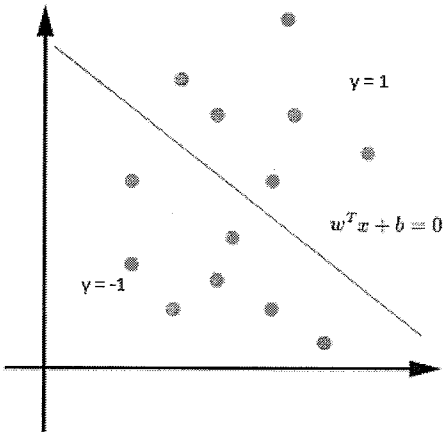


图 3.1 SVM 线性可分示例图

我们称数据集 $\{x_1, x_2, ..., x_p\} \subset R^n$ 是线性可分的。这个超平面也就是我们要找的判别函数。可以把上述不等式合并，写成

$$y_i\,(\omega^T x + b\,) \geq 1, \quad i = 1, 2, ...m. \tag{3.3}$$

要注意的是，对一个固定的超平面，参数（ω, b）不是唯一确定的（可以乘以一个常数因子来缩放，如 $x + y + z = 1$ 和 $2x + 2y + 2z = 2$ 表示的其实是一个平面）。在这个基础上，我们总可以找到一对（ω, b），使得

$$\min_{1 \leq i \leq m} |\,\omega^T x_i + b\,| = 1 \tag{3.4}$$

这时，我们可以定义数据集样本到该超平面的最小几何距离为

$$\gamma = 1/\|\omega\| \tag{3.5}$$

对线性可分的数据集，我们可能可以找出很多分离超平面满足上述(3.3) 式的要求。这时候，需要找出最优的分离超平面。我们希望数据样本到最优超平面的最小几何距离 γ 最大（见图3.2）。

于是求最优超平面可以转化为求下面问题的解

$$\begin{aligned} &\min_{\omega,b} \ \frac{1}{2}\|\omega\|^2 \\ &s.t. \quad y_i(\,\omega^T x_i + b) \geq 1, \ \ i = 1, 2, ..., m. \end{aligned} \tag{3.6}$$

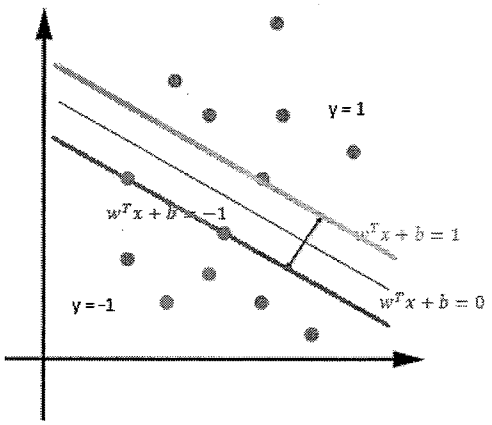


图 3.2 SVM 寻找最优分离超平面

这是个凸二次优化问题，有唯一的极小点，也就是说可以找到一个最优超平面，此时 γ 最大。解这个二次规划问题（QP）除了常规的方法外，还可以通过求解对偶问题得到最优解，这就是支持向量机线性可分情况下的对偶算法。引入对偶算法的优点在于：一方面对偶问题更容易求解，另一方面可以自然地引入核函数，进而推广到非线性分类问题。

我们给每一个约束条件加上一个拉格朗日乘子 α（Lagrange multiplier），便可以通过拉格朗日函数将约束条件融合到目标函数中去，于是式(3.6)引入拉格朗日乘子后变成拉格朗日函数

$$L(\omega, b, \alpha) = \frac{1}{2}\|\omega\|^2 - \sum_{i=1}^{m} \alpha_i \{y_i(\omega^T x_i + b) - 1\} \quad (\alpha_i \geqslant 0, \ i = 1, 2..., m) \tag{3.7}$$

对其求偏导后，由 Lagrange 函数可得原问题的 Karush-Kuhn-Tucker（KKT）条件

$$\begin{aligned}
&\frac{\partial L}{\partial \omega_v} = \omega_v - \sum_{i=1}^{m} y_i \alpha_i x_{iv} = 0, \quad v = 1, ..., n. \\
&\frac{\partial L}{\partial b} = -\sum_{i=1}^{m} y_i \alpha_i = 0 \\
&y_i \left(\omega^T x_i + b\right) - 1 \geq 0, \quad i = 1, ..., m. \\
&\alpha_i \geq 0, \quad i = 1, ..., m. \\
&\alpha_i \{y_i(\omega^T x_i + b) - 1\} = 0, \quad i = 1, ..., m.
\end{aligned} \tag{3.8}$$

根据经典的优化理论，（ω, b）是原问题的解当且仅当（ω, b, α）满足 KKT 条

件。由此也可导出原问题的等效对偶表达式

$$\begin{aligned} &\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j x_i^T x_j \\ &s.t. \quad \alpha_i \geq 0, \quad i = 1, ..., m. \\ &\sum_{i=1}^{m} y_i \alpha_i = 0 \end{aligned} \tag{3.9}$$

且原问题的解

$$\omega = \sum_{i=1}^{m} y_i \alpha_i x_i \tag{3.10}$$

以上原问题（式(3.7)），KKT条件（式(3.8)）和对偶问题（式(3.9)）是同一问题的不同描述方式，可以根据具体问题来选择其中一种描述形式。

在对偶问题和 KKT 条件中，每个训练数据 x_i 都对应一个拉格朗日乘子 $\alpha_i \geq 0$，其中与 $\alpha_i > 0$ 对应的数据称为支持向量（此时 $y_i\,(\,\omega^T x_i + b\,) = 1$）。

在上述几种描述中求得 ω 后，利用任一支持向量和 KKT 条件 $\alpha_i\{y_i\,(\omega^T x_i + b\,) - 1\} = 0$，可求出

$$b = y_i - \omega^T x_i \tag{3.11}$$

在数值计算中，通常求出多个b值，取平均[31]。

3.1.2　引入松弛变量后的线性可分

如果数据集并不像图3.1所示的理想化，如图3.3所示两类有所交错，不能找到一个超平面将其完全分开。实际情况中也确实存在这种情况，如数据受到噪声污染，或者一些特殊样本的干扰。面对这种情况，数据集还是可能线性可分，此时需要排除噪声等带来的干扰。

另外一种情况是，虽然存在线性分割平面满足要求，但是线性分割的效果很不好，如图3.4 所示。此时数据集经过干扰后虽然仍可以线性分割，但是明显可以看出舍弃极个别干扰点噪声，就能得到极佳的分割结果。这时候，我们应当做出舍弃。

对于诸如图3.3和图3.4的情况，我们可以引入非负松弛变量 ξ，将原约束条件改为

$$\begin{aligned} &\omega^T x_i + b \geq 1 - \xi_i, \quad y_i = 1 \\ &\omega^T x_i + b \leq -1 + \xi_i, \quad y_i = -1 \\ &\xi_i \geq 0, \quad i = 1, ..., m. \end{aligned} \tag{3.12}$$

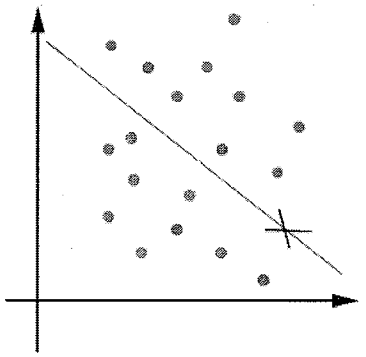


图 3.3 SVM 两类交错示例图

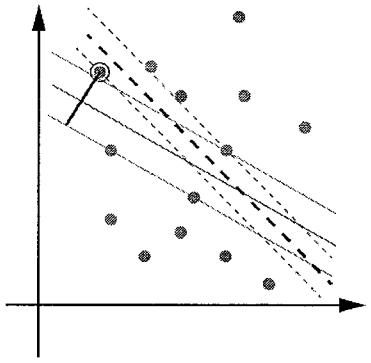


图 3.4 SVM 舍弃干扰示例图

其中，ξ_i 可以看作训练样本关于分离超平面的偏差容忍度。$\xi_i = 0$ 时，问题变回了前面讨论过的线性可分情况，此时不需要偏差容忍。$\sum \xi_i$ 是训练样本偏差的总数。为了控制偏差总数，原目标函数改为

$$\frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m}\xi_i \tag{3.13}$$

得到修改后的优化问题为

$$\begin{aligned} &\min_{\omega,\xi} \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m}\xi_i \\ &s.t. \quad y_i(\,\omega^T x_i + b\,) \geq 1 - \xi_i \\ &\xi_i \geq 0, \quad i = 1, ..., m. \end{aligned} \tag{3.14}$$

其中 $C > 0$ 是自定义参数，称为惩罚系数。设定 C 越大，错误容忍度越小，但泛化能力下降，设定 C 越小则情况相反。用户可以通过调节 C 的大小在泛化能力与误差之间取得平衡。C 的选取具体与训练集本身有关。

根据(3.14) 式引入拉格朗日乘子得到新的拉格朗日函数为

$$L=\frac{1}{2}\|\omega\|^2+C\sum_i\xi_i-\sum_i\alpha_i\{y_i(\omega^Tx_i+b)-1+\xi_i\}-\sum_i\mu_i\xi_i \tag{3.15}$$

其对偶形式为

$$\begin{aligned}&\max_{\alpha}\quad\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_jy_iy_jx_i^Tx_j\\&s.t.\quad 0\le\alpha_i\le C,\quad i=1,...,m.\\&\sum_{i=1}^{m}y_i\alpha_i=0\end{aligned} \tag{3.16}$$

问题的解为

$$\omega=\sum_{i=1}^{m}y_i\alpha_ix_i \tag{3.17}$$

同样可写出其 KKT 条件

$$\begin{aligned}&\frac{\partial L}{\partial\omega_v}=\omega_v-\sum_{i=1}^{m}y_i\alpha_ix_{iv}=0,\quad v=1,...,n.\\&\frac{\partial L}{\partial b}=-\sum_{i=1}^{m}y_i\alpha_i=0\\&\frac{\partial L}{\partial\xi_i}=C-\alpha_i-\mu_i=0\\&y_i(\omega^Tx_i+b)-1+\xi_i\ge0\\&\xi_i\ge0,\quad\alpha_i\ge0,\quad\mu_i\ge0\\&\alpha_i\{y_i(\omega^Tx_i+b)-1+\xi_i\}=0\\&\mu_i\xi_i=0,\quad i=1,...,m.\end{aligned} \tag{3.18}$$

若 $\alpha_i>0$，称相应的 x_i 为支持向量（Support Vector），若 $0<\alpha_i<C$，称 x_i 为边缘（Margin）支持向量，若 $\alpha_i=C$，称 x_i 为偏差（Bias）支持向量。

利用任一边缘支持向量及 KKT 条件联立方程

$$\begin{aligned}&\alpha_i\{y_i(\omega^Tx_i+b)-1+\xi_i\}=0\\&\frac{\partial L}{\partial\xi_i}=C-\alpha_i-\mu_i=0\\&\mu_i\xi_i=0\end{aligned} \tag{3.19}$$

可求出

$$b=y_i-\omega^Tx_i \tag{3.20}$$

数值计算时通常取平均值。将 ω,b 代入分割平面 $\omega^T x + b = 0$ 得

$$\sum_{x_i \in SV} y_i \alpha_i x_i^T x + b = 0 \tag{3.21}$$

于是判别函数为

$$y = sgn(\sum_{x_i \in SV} y_i \alpha_i x_i^T x + b) \tag{3.22}$$

3.1.3 非线性可分

线性分类是最简单的分类算法。但是如果数据集本身呈现如图3.5所示，此时若利用线性分类器，即使加入松弛变量，效果也会非常差。所以，需要构造非线性分类器。

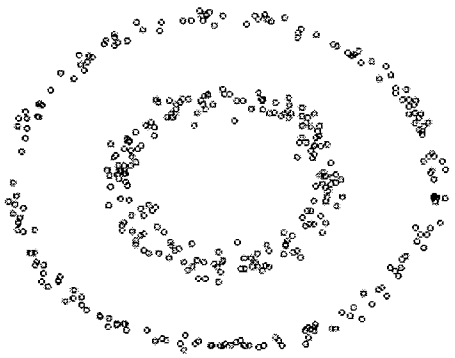


图 3.5 SVM 线性不可分示例图

非线性分类器主要有两种构造方法，一是利用神经网络方法，二是把原有数据集映射到高维特征空间，然后在特征空间中构造线性分类器。SVM 算法采用了第二种实现方法。

如图3.5所示，假设输入空间是 R^2，在此空间维度上，我们无法构建一个线性分类器来很好地做出分类。我们可以将其做一个映射 ϕ，把 R^2 上的点映射到 R^5 中，

$$\phi:\ x = (x_1, x_2) \rightarrow\ (x_1, x_2, x_1 x_2, x_1^2, x_2^2) \tag{3.23}$$

然后在 R^5 中，我们做线性分割

$$\omega^T \phi(x) + b = \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_1 x_2 + \omega_4 x_1^2 + \omega_5 x_2^2 + b \tag{3.24}$$

这样我们就可以在 R^5 中很好地解决这个问题。判决函数为

$$\begin{aligned} f(x) =& sgn(\omega^T \phi(x) + b) \\ =& sgn(\omega_1 x_1 + \omega_2 x_2 + \omega_3 x_1 x_2 + \omega_4 x_1^2 + \omega_5 x_2^2 + b) \end{aligned} \tag{3.25}$$

这里存在一个问题。当输入空间的维数很高时，这种非线性映射方法使得特征空间的维数呈指数增长，计算代价太大。这时候，就要用到前面所说的对偶式了。

在式(3.16)中，我们只要把 x，x_i 分别变换成 $\phi(x)$，$\phi(x_i)$，就得到了非线性映射到高维特征空间后的对偶式，

$$\begin{aligned} &\max_{\alpha} \quad \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \phi(x_i),\ \phi(x_j) \rangle \\ &s.t. \quad 0 \le \alpha \le C, \quad \forall i \\ &\sum_{i=1}^{m} y_i \alpha_i = 0 \quad (i = 1, ...m) \end{aligned} \tag{3.26}$$

分离超平面的方程是

$$\sum_{x_i \in SV} y_i \alpha_i \langle \phi(x_i),\ \phi(x) \rangle + b = 0 \tag{3.27}$$

这样，我们就得到了非线性可分情况下，将原数据集非线性映射到高维特征空间后的分离超平面表达式。在高维特征空间（如R^5）看来，式(3.27) 是线性分割平面，在原数据集维度（如R^2）看来（将式(3.27)的映射函数代入展开），最终的分离超平面方程是不规则的曲面（见图3.6)。

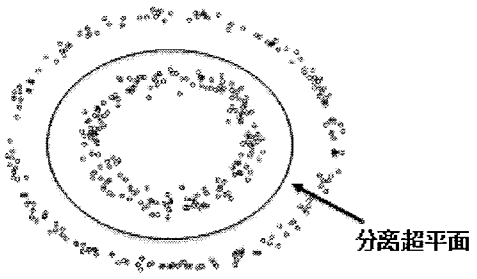


图 3.6 SVM 线性不可分情况分离曲面

3.1.4 核函数

从上节(3.26)式和(3.27)中可以发现，在非线性可分问题中，我们将原数据集做非线性映射到高维特征空间，数据将以内积的形式出现在最优超平面方程中。我们把形如 $\langle\phi(x),\phi(y)\rangle$ 内积形式的表达式单独提取出来，称之为核函数（Kernel Function），即

$$\kappa(x,y)=\langle\phi(x),\phi(y)\rangle \tag{3.28}$$

把核函数代入对偶表达式(3.26)，得

$$\begin{aligned}&\max_{\alpha}\quad\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j\kappa(x_i,x_j)\\&s.t.\quad 0\le\alpha\le C,\quad\forall i\\&\sum_{i=1}^{m}y_i\alpha_i=0\quad(i=1,...m)\end{aligned} \tag{3.29}$$

分割面方程为

$$\sum_{x_i\in SV}y_i\alpha_i\kappa(x_i,x)+b=0 \tag{3.30}$$

判别函数

$$y=sgn(\sum_{x_i\in SV}y_i\alpha_i\kappa(x_i,x)+b) \tag{3.31}$$

其中，关于 b 的计算有点复杂，具体请参考文献[32]。

不难发现，核函数的本质是将输入数据集空间映射到一个高维特征空间，从而将输入空间的非线性问题转化为特征空间内的线性问题[33]。通过引入核函数，高维特征空间的内积运算就可以通过原空间的一个核函数来隐含地进行运算，算法的复杂性并没有随着维数的增加而增加[34]。

那么，怎么确保一定能将非线性问题转换为特征空间的线性问题呢？这就要求我们找到合理的核函数。常用的核函数有以下四类[35]：

1. 线性核（Linear）：$\kappa(x,y)=\langle x,\ y\rangle$
2. 多项式核（Polinomial）：$\kappa(x,y)=(\gamma\langle x,\ y\rangle+c)^n$
3. 径向基核（Radial Basis Function）：$\kappa(x,y)=\exp(-\gamma\|x-y\|^2)$
4. Sigmoid核（两层神经网络）：$\kappa(x,y)=\tanh(\gamma\langle x,\ y\rangle+c)$

支持向量机的性能很大程度上取决于核函数，合理选择核函数以及其参数是解决问题的关键。目前的选取方法主要有：经验选择法、实验试凑法、梯度下降法、

交叉验证法、贝叶斯法等。还有些研究通过对核函数进行适当修改来加强支持向量机的性能[36]。实际上，对支持向量机的具体求解过程，就是选取核函数及其参数的过程。

3.2　支持向量机的应用

前面简单介绍了支持向量机的基本原理。作为机器学习十大算法之一，支持向量机在模式识别[37, 38]（字符识别[39]、文本自动分类[40]、人脸检测[41-43]、头的姿态识别[44])、函数逼近[29, 45]、数据挖掘[46]和非线性系统控制[47] 中均有很好的应用[48]，被多方面推广。相比于神经网络（Neural Network，简称NN），支持向量机有以下几个优点：

1. 支持向量机的理论基础比 NN 更坚实。

2. 支持向量机不需要太多的经验支撑，而 NN 更加依赖于工程技巧。

3. NN 不能控制经验风险和置信范围，支持向量机能。

相信随着支持向量机技术的发展，将来会有更多的应用前景。

3.3　支持向量机实现

支持向量机（SVM）作为当下研究热点，诞生了很多的相关工具来实现其算法。如 SVM light[49]，bsvm[50]，LIBSVM[51]，mySVM[52]和 Matlab 自带的 toolbox [53]。当然，在清楚原理的基础上，也可以自己代码实现相关算法。本文采用了 LIBSVM 工具作为 SVM 实验的工具。

3.4　本章小结

本章介绍了支持向量机的基本概念、原理及几种常见情况，介绍了核函数的作用及其重要性。还介绍了支持向量机的应用及其优点（与神经网络相比）。最终，介绍了几种实现支持向量机的相关工具。

第四章 研究方法

本文采用了 gem5 和 McPAT 的组合作为实验仿真的主要工具。其中，gem5 是计算机体系结构领域的常用仿真工具，学术界用它来模拟程序在不同平台上的运行情况。McPAT （见第4.3.2节）是我们在 gem5 运行完仿真程序后，通过提取 gem5 仿真结果，用来计算整个系统各部件功耗情况的（包括静态功耗和动态功耗）。这样，gem5 + McPAT 工具的组合使得我们能够做系统级别的功耗研究。

4.1 主要思想

基于任务的 DVFS 策略需要准确预测当前 CPU 任务量，由于程序执行过程具有时间和空间连续性，当调节周期比较短时，可以把前一周期的 CPU 任务量作为当前周期 CPU 任务量的参考，以此估计当前 CPU 的任务量。更进一步，可以把计算得到的前一周期的 CPU 最佳执行频率作为当前周期 CPU 的执行频率。这样虽然有一个周期的延时，但是实验过程中发现，设置调节周期比较短时，大多数程序运行显现出阶段性特征，需要十几或者几十个调节周期才会进行一次变频操作，一个周期的延时带来的影响微乎其微。

于是，预测下个周期 CPU 的执行频率变成了当前周期的 CPU 最佳频率判断问题。在当前周期运行完毕后，我们通过 CPU 各部件的统计数据来判断其最佳执行频率。显然，可以把它当做一个分类问题处理。

假设 CPU 进行 DVFS 调节时可选的频率有 L 种（L 类），把 CPU 各部件的统计数据写成向量形式，称为 Parameter vector，对任一 Parameter vector，有一种频率 $f_i\,(i = 1, 2, ..., L)$ 使得其 EDP 最小，这组 Parameter vector 就被认为是属于第 i 类，该类对应的频率即为 CPU 的最佳执行频率。这样，DVFS 最佳频率选择变成了一个分类和模式识别的问题，我们引入机器学习领域的支持向量机来构建模型。

需要注意的是，CPU 各部件的统计数据非常多，Parameter vector 作为判别模型的输入，内部参数的选择应该以能反映程序特征为标准，例如 ipc，Cache miss 数，Memory access 数等。另外，在满足判断准确性的前提下，应适当删除 Parameter vector 中的元素来缩减长度，减少运算时间。

4.2　流程概览

整个研究主要分为如下几个大的步骤：

1. 仿真。利用 gem5 + McPAT 组合获得程序在不同频率下运行得到的仿真结果和完整的功耗数据。

2. 获得样本数据。将第 1 步中得到的所有初始数据做出处理，得到可用于 SVM（参见第三章）建模的数据集样本。

3. 使用支持向量机建立 DVFS 决策模型。

4. 集成验证。

研究流程见图4.1。流程图中，我们把上面第 1 到 3 步称为训练阶段，第 4 步称为测试运行阶段。

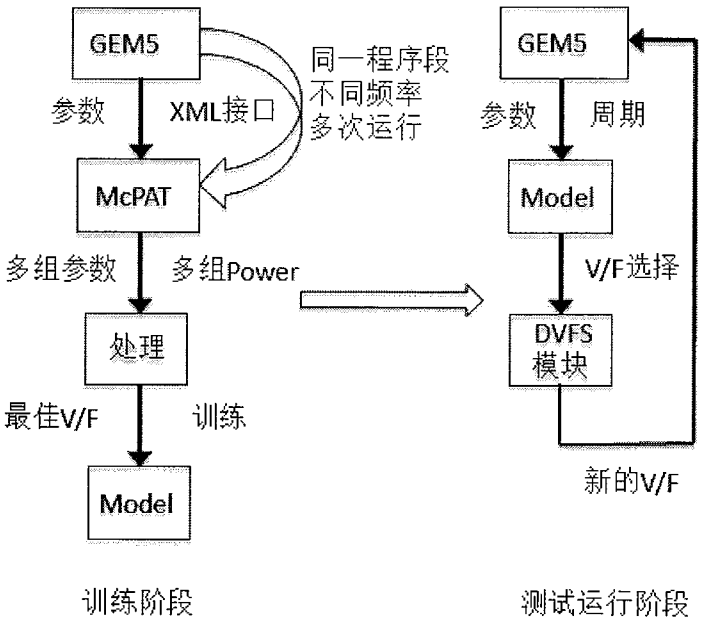


图 4.1 研究流程图

在训练阶段，首先要得到用于训练的数据。我们将要仿真的程序按固定指令数平均划分为 n 段，对每个程序段，利用 gem5 和 McPAT 工具可以得到一组形如<Parameter vector，F，EDP>形式的数据。其中 Parameter vector 需要自己选取，

例如ipc，各级Cache hit/miss数，Memory access数等。F为CPU运行频率，EDP为执行此程序段所消耗的能量和延迟的乘积。以不同频率执行此程序段，得到多组这样的数据。在处理环节中，选择出拥有最低EDP的一组数据，我们认为它所对应的执行频率F是此程序段的最佳运行频率（电压V跟随F变化），并将这组数据选入数据样本中。运行不同程序段，得到更多样本，组成我们用于训练的样本数据集合。

接着，我们将样本数据中的<Parameter vector，F，EDP>按频率分类（频率F相同的作为一类）。采用机器学习领域的支持向量机（见三 章）对样本数据加以学习和训练，得到决策模型。此决策模型的目标是，给定一组Parameter vector，能判断出其所属的F类，此F类即该程序段运行的最佳频率。

在测试运行阶段，我们将训练阶段得到的决策模型作为一个模块（Model）集成到gem5中。gem5执行程序过程中，每隔一段固定的时间将当前周期运行程序段的Parameter vector统计结果传递到这个决策模型中，决策模型经过判断后向DVFS执行模块输出最佳运行电压和频率（V/F），DVFS模块将执行这个切换过程（相当于实现DVFS的调频调电压功能），切换完毕后，系统将以新的V/F运行下个程序段。循环此过程直到整个程序运行完毕，得到实验结果进行验证分析。

以上就是整个流程的简要概括。本章作为研究的重点，将详细讲述每个步骤，包括实验工具，方法和结果等。

4.3 仿真

本小节主要是仿真平台搭建和数据收集过程。我们采用gem5+McPAT组合获得各程序段在不同频率下运行得到的仿真结果和完整的功耗数据。要完成这个过程，涉及对gem5和McPAT这两个主要工具的修改与应用。

4.3.1 gem5相关

前面在第二章DVFS技术实现中已经对gem5工具的功能做了简要说明。实际上，gem5仿真器是一个用于进行计算机微体系结构研究的模块化平台。作为我们实验的主要工具，需要对gem5有更全面的了解。

4.3.1.1 gem5特性

gem5的特性主要有以下几点：

1. 多种可互换的CPU模型。gem5提供了四种可互换的CPU模型：一种简单的每周期执行一条指令的CPU模型（simple one-CPI CPU，简称simple CPU），一种精细的按序执行CPU模型（detailed model of in-order CPU），一种精细的乱序执行CPU模型（detailed model of out-of-order CPU，简称O3 CPU）。这些CPU模型使用同一种高级ISA描述（如可均为X86架构）。另外，gem5的突出点是拥有基于内核虚拟的CPU模型，可以通过利用硬件虚拟化来加速仿真。

2. Nomali GPU模型。gem5支持集成NoMali GPU模型，并且Nomali GPU模型与Linux和安卓GPU驱动兼容，这样免除了软件渲染的需求。

3. 事件驱动的存储系统。gem5拥有一个精细的，事件驱动的存储系统，包括caches，crossbar，snoop filters和快速准确的DRAM控制模块，来测试当前内存的效果，例如LPDDR3/4，DDR3/4，HBM1/2，HMC，WideIO1/2等。这些组件可以灵活安排，例如可以构建一个多级专用的cache层级结构和异构的主存。

4. 同构或者异构多核。CPU模型和Cache结构可以任意配置组合来组建同构或者异构的多核系统。通过MOESI snooping cache一致性协议保证Cache的一致性。

5. 全系统模式兼容多种架构：

   (a) Alpha：gem5详细模拟了DEC Tsunami系统来引导原版的Linux 2.4/2.6，FreeBSD和L4Ka::Pistachio。

   (b) ARM：gem5可以在ARM平台上模拟最多64个异构的核，结合in-order CPU或者O3 CPU引导原版的Linux和安卓内核。ARM的实现支持32位和64位的内核和应用程序。

   (c) SPARC：gem5仿真器可以精细模拟单核的UltraSPARC T1处理器引导Solaris系统。

   (d) X86：gem5仿真器支持标准的个人电脑平台X86结构。

6. 多系统兼容。gem5中各组件（CPUs，crossbars，caches等）都表示为objects，gem5允许灵活调用这些组件来描述复杂的系统结构。因为一个完整的系统是

多个objects的集合，所以一次仿真可以实例化多个系统。结合全系统模拟，这个特征允许完全的client-server网络仿真模式。

7. 功耗和能量建模。gem5的objects都拥有操作系统可见的功耗和时钟域，由此能进行一系列的关于功耗和能量的实验。在由外界操作系统控制支持的DVFS技术下，gen5可以提供一个完整的平台用于将来的能耗研究。

8. 和SystemC进行协同仿真。gem5可以包含于SystemC仿真中，作为一个线程在SystemC事件内核中运行，保持事件（Events）和时间线（timelines）同步。这项功能使得gem5组件可以和很多SOC组件模型交互操作，例如互连线，设备和加速器等。提供了SystemC传输线矩阵的封装。

4.3.1.2 gem5源代码结构

gem5主要由Python和C++语言编写，是一款高度模块化的仿真器（如包含CPU，L2 Cache，Memory，Memory Control模块等）。其中，Python代码用于初始化、设置、组织和配置各模块。而模块的具体实现由C++语言编写。实际上，gem5利用了swig工具[54]来封装和导出C++模块接口，使其被Python组织调用。

编译安装好gem5后，利用GDB和PDB工具调试发现gem5 SE模式启动过程图如图4.2所示。

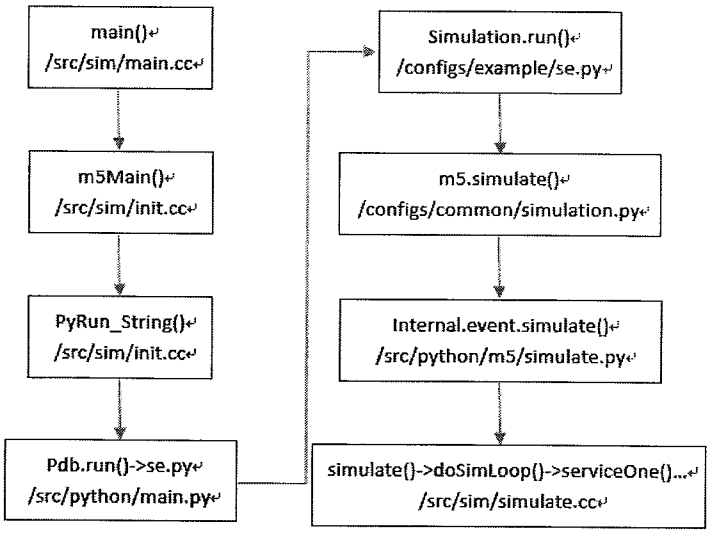


图4.2 gem5 SE模式程序执行流程图

前面的一列流程均是初始化操作，包括系统各模块的建立，连接，配置，设置各种路径等。其中，se.py 文件是整个系统配置的重点，包括系统各模块的初始化，它们的从属及连接关系等。Simulation.py 文件主要涉及对仿真过程的控制，包括设置断点，恢复执行，快进至断点，转换 CPU （switch cpu）执行，输出结果，重置计数器等。后续的 simulate 相关文件就是仿真执行的具体过程了。

gem5 是事件（Event）队列驱动的仿真系统，仿真运行的程序（Benchmark）会产生事件加入到事件队列中，仿真器从事件队列中取出事件执行。所以，源程序在 simulate.cc 的 doSimLoop 函数中设置了一个循环，每一次循环都调用 serviceOne 函数，该函数负责服务或者说处理本次从事件队列取出来的事件，每个事件的处理过程具体会涉及到 CPU 和 Cache 等系统各模块的运作，一切模拟真实计算机系统运行程序时的状态，并加入很多计数参量作为输出结果，如执行指令数，Cache miss 数，ipc，cycle 数等。当要仿真的程序产生的事件队列被执行完毕了，仿真任务也就完成了。系统各组件如 CPU ，Cache 等在此过程中的运行状态通过计数参数输出到 m5out 文件夹中的 stats.txt 文件中作为整个仿真的输出结果。

4.3.1.3　gem5 源代码修改目的

我们选择在 gem5 SE模式下，运行 CPU SPEC2006 [55] 中的程序 (Benchmark)。在此之前，我们需要对 gem5 源代码做一些修改，修改 gem5 源代码的主要目的是实现 gem5 对仿真程序的程序段（interval）划分，使 gem5 每执行完一个程序段都输出当前程序段仿真结果。

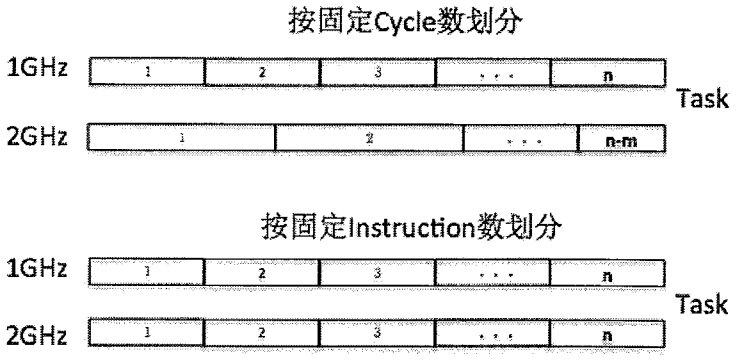


图 4.3 程序段划分

和大多数按固定 Cycle 数划分程序段[56]的方法不同，我们按固定 Instruction 数

划分程序段。这样做的好处如图4.3所示。

对一个固定程序，它的任务量（Task）是恒定的。在仿真过程中，可以发现它的指令数是恒定的。但是在不同运行频率下，其运行的总 Cycle 数会不一致。所以，当将程序按照固定 Instruction 数划分时，每个程序段是对齐的；而当将程序按照固定 Cycle 数划分时，不同频率下执行的程序段每次都会发生一段偏移，越往后累计偏移越多，程序在 1GHz 频率下执行的第 i 段代码和 2GHz 频率下执行的第 i 段代码已经完全不是同一段代码了。显然，我们的目的是要针对同一段程序代码在不同频率下运行情况进行研究，所以我们选择按固定 Instruction 数划分程序段。

4.3.1.4 gem5 源代码修改

基于以上讨论结果和目的，我们来修改 gem5 源代码。我们要实现程序段的划分，需要对程序的运行进行控制。由于 Python 代码用于初始化、设置、组织和配置各模块，所以我们在 Python 代码部分进行修改能更方便地达到目标。基于4.3.1.2节中对 gem5 源代码结构的讨论，我们应该在 Simulation.py 文件中进行修改。主要用到三个函数：m5.simulate()，m5.stats.dump() 和 m5.stats.reset()。

m5.simulate() 函数是运行仿真的命令，括号内的参数代表仿真运行的 tick 数（tick 是 gem5 中的计时单位，默认一个 tick 为 1ps），如 m5.simulate(1000) 代表程序仿真运行 1000 个 ticks。通常，gem5 也用 tick 来描述时钟频率，如设定 CPU 时钟频率为 1GHz，相应的周期为 1ns，表示为 1000 ticks，m5.simulate(1000) 也就是仿真一个周期。

m5.stats.dump() 函数和 m5.stats.reset() 函数是一对对结果输出进行控制的函数，他们总是成对出现。其中，dump()函数会向指定的输出文件中（默认为 stats.txt）打印当前计数结果，reset() 函数会重置各计数变量为零，通常用在 dump() 函数后，用于开始新一轮的计数。

另外要注意的是，m5.simulate() 函数只能指定运行固定 tick 数，而我们的目的是要执行固定指令数后 dump 输出结果。当然，我们可以 cycle by cycle 地仿真执行，每次检测已执行指令数是否符合要求。但是，这样细精度的仿真执行速度太慢，执行一小段程序可能需要几个礼拜的时间。所以，我们采用了近似的办法。

譬如我们要按一千万个指令划分程序段。我们可以每次运行一万个周期，运行完后通过变量“system.cpu.committedInsts”的值（gem5 中用来累计已执行指令数）来判断是否已执行超过一千万个指令，若达到则执行 dump 操作和 reset 操作，进行

下个程序段的划分，若没达到，则继续执行一万个周期同时累计已执行指令数。

这里可以发现，执行 dump 操作时，已执行的指令数（system.cpu.committedInsts）可能已经超过一千万，最大可能到一千零一万多（ipc 值一般在 0.2 到 1.6 左右，且大部分情况为 1 左右），相对于一千万的总指令数来说一万左右的误差在可接受范围内。或者我们可以进一步调小步伐，每次运行一千甚至一百个周期就做一次判定，这样虽然精度可以很高，但是仿真花费的时间也会相应更多，可根据精度需求折衷处理。修改的代码关键部分如下：

```
#Simulation.py
print "**** REAL SIMULATION ****"
cycle_num = 10000
exit_cause = "simulate() limit reached"
while(exit_cause == "simulate() limit reached"):
  exit_event = m5.simulate(cycle_num * testsys.cpu_clk_domain._ccParams.clock[0])
    exit_cause = exit_event.getCause()
    maxtick = maxtick - cycle_num * testsys.cpu_clk_domain._ccParams.clock[0]
  if(maxtick<=0):  exit_cause = "Max simulate() limit reached"
    if((m5.stats.stats_dict['system.cpu.committedInsts'].total()>10000000)
    &(exit_cause=="simulate() limit reached")):
        m5.stats.dump()
        m5.stats.reset()
print 'Exiting @ tick %i because %s' % (m5.curTick(), exit_event.getCause())
```

值得注意的是，代码中的变量名字并不是固定不变的，如 system.cpu.committedInsts 在 option 选项中有断点设置需要 switch cpu 时，可能 cpu 名字变为了 *switch_cpus*，所以前面要加入条件判断使其通用化。

4.3.1.5　gem5 运行

gem5 支持多种体系架构，集成了多种 CPU 模型，还有 SE 和 FS 两种不同的仿真模式。研究中，我们在 Ubuntu 平台下安装运行 gem5，选用 SE 模式，O3 CPU，X86 架构，默认的 Cache 大小和结构，按照不同频率分别执行程序（Benchmark）。如以 1GHz 频率执行编译好的 gcc 程序，命令为 build/X86/gem5.opt configs/example/se.py -c gcc –cpu-type=detailed –cpu-clock=1GHz –caches –l2cache -i cccp.i，其中 cccp.i 为程序 gcc 执行时的输入文件，具体各选项（如 -c，-i 等意义请参见 gem5 说明文档，或者在命令行中直接输入 build/X86/gem5.opt –help 命令查看），执行完毕后，即可在 /m5out 文件夹中的 stats.txt 文件中查看仿真输出结果。

4.3.2 McPAT相关

McPAT是我们要用到的另外一个主要工具，我们用它来计算CPU各部件的功耗情况，包括静态功耗和动态功耗，它提供了我们最原始的功耗建模数据。

4.3.2.1 McPAT简介

McPAT（Multicore Power, Area, and Timing）是一种新的功耗，面积和时序建模框架。它使用XML文件作为输入，按照输入文件通过配置优化CPU结构进行面积，功耗和时序的建模。结合gem5仿真器，我们常常用它来进行功耗方面的研究和探索。McPAT模拟的不止是动态功耗，还有静态和短路功耗。并且，McPAT在输出报告中将给出系统各部件的详细功耗情况[57]。

4.3.2.2 McPAT使用

我们利用gem5得到仿真输出结果后，需要利用McPAT来计算每个程序段不同频率下执行的功耗情况。在上节中，我们得到了程序的仿真结果（stats.txt），通过批处理脚本，可以分割stats.txt文件得到每个程序段的仿真输出结果，然后通过perl脚本提取各结果中的参量信息填充到XML模板中，计算得到功耗报告。

要从仿真结果（stats.txt）文件中提取的参量数据具体见附录6.2。通过将这些参量信息提取到McPAT提供的XML文件接口中，再按照配置输入config相关参数，就能计算得到功耗报告。

值得注意的是，McPAT计算功耗分为两个步骤，第一步配置和优化CPU结构，第二步在配置好的CPU结构上进行计算。XML文件中参数的变更可能导致第一步中按照功耗、时序、面积为目标优化得到的CPU不同。我们需要设置McPAT仅作一次CPU配置，以后均在此CPU结构上进行功耗计算，这样既保证了我们是在同一款CPU上不同输入情况下进行功耗计算，又节省了每次配置CPU（第一步）的时间。

4.4 获得样本数据

我们利用gem5给程序分段后，以不同频率运行各程序段，得到仿真输出结果stats.txt文件，并利用McPAT工具计算各程序段不同频率下运行的功耗。现在，我们的目的是在这些原始数据的基础上获得能够用于支持向量机训练的样本数据。

在实验仿真过程中，每个程序段在不同频率下运行获得相应的<Parameter vector，F，EDP>形式的数据，针对某个具体程序段，我们认为其 Parameter vector 在误差范围内一致，体现了这段程序的特性。我们要找出运行这段程序的最佳频率F，通过比较 EDP，选择 EDP 最小的那组数据，其对应的 F 即为所求。我们引入整数线性规划（Integer Linear Programming，简称 ILP）来解决这个问题。

4.4.1　整数线性规划

整数线性规划（ILP）问题的标准形式是

$$\begin{aligned} &\min Z = \sum_{i=1}^{n} C_i X_i \\ &\sum_{i=1}^{n} A_i X_i = b \\ &X_i \geq 0, \quad i = 1, 2, ..., n. \\ &X_i \in I, \quad i \in J \subset \{1, 2, ..., n\} \end{aligned} \tag{4.1}$$

其中 A 为 $m \times n$ 矩阵，X_i 为 $n \times 1$ 向量，是要优化的变量，I 是一整数集合。若 $I = \{0, 1\}$，则问题为 0-1 规划问题。若 J 是{1,2,...,n}的非空真子集，则该问题为混合整数线性规划问题。若 $J = \{1, 2, ...n\}$，则该问题为纯整数线性规划问题。

本研究中我们要解的方程式如(4.2)式所示：

$$\begin{aligned} &\min \sum_{j=1}^{L} EDP_{i,j} X_{i,j} \\ &\sum_{j=1}^{L} X_{i,j} = 1 \\ &X_{i,j} \in \{0, 1\} \end{aligned} \tag{4.2}$$

其中，$EDP_{i,j}$表示第 i 段程序在第 j 种频率下运行时的 EDP，L 表示可选的频率数（$L = 1$ 表示只有一种频率供选择）。$X_{i,j} = 1$ 代表诸多频率中选择第 j 种频率运行第 i 段程序是最优的。很显然，这是一个 0-1 规划问题。解这个方程式得到$X_{i,j}$的值，就能得到第 i 段程序的最佳执行频率。稍作改变（如式(4.3)）即可批量求得各程

序段(1-N)的最佳执行频率。

$$\begin{aligned} &\min \sum_{i=1}^{N}\sum_{j=1}^{L} EDP_{i,j}X_{i,j} \\ &\sum_{i=1}^{N}\sum_{j=1}^{L} X_{i,j} = N \\ &X_{i,j} \in \{0,1\} \end{aligned} \tag{4.3}$$

引入ILP求解的好处是能用现成的数学工具包求解。另外，在(4.3)式上稍加修改，就能探索CPU时间换能量的的转换关系，譬如一段程序已知在1GHz频率下执行花费时间为t，现在多出10%的时间，探索应该采用何种频率执行，相比1GHz频率最多节省多少能耗，如参考文献[56]所做工作。这同样是一个整数线性规划的问题，可用数据工具包求解。本文限于篇幅，不再这方面进行过多探索。

我们引入Gurobi Optimization工具包解这个ILP问题，最终得到每个程序段的最佳执行频率，获得样本（Sample）数据集合。

4.4.2 Gurobi Optimization 工具包求解整数线性规划

Gurobi Optimization是由美国Gurobi公司开发的新一代大规模数学规划优化器，在第三方优化器评估中，展示出更快的优化速度和精度，成为了优化器领域的新翘楚。

Gurobi的特点包括[58]：

（1）采用最新的优化技术，充分利用多核处理器优势。

（2）提供了方便轻巧的接口，支持c++，Java，Python，.Net开发。

（3）支持多种平台，包括Windows，Linux，Mac OS。

（4）和Matlab有便捷接口。

我们采用Gurobi Optimization工具包来求解整数线性规划问题。我们采用Python接口在Ubuntu平台上进行求解。描述(4.3)式的主要语句为：

```
for i in Iset:
  for j in Jset:
        x[i,j]=m.addVar(vtype=GRB.BINARY,obj=EEnergy[i,j]]*x[i,j]*TT[i,j],name='
                                                    x_%s_%s' %(i,j))
for i in Iset:
     m.addConstr((quicksum(x[i,j] for j in Jset) == 1),'eqn2_%s_%s' %(i,j))
```

上面一段程序段设定了(4.3)式中的两个约束条件，GRB.BINARY 类型表示这是一个 0-1 规划问题，最后再设置目标函数如下：

```
m.setObjective(quicksum(EEnergy[i,j]*x[i,j]*TT[i,j]*100000 for i in Iset for j in
                                        Jset),GRB.MINIMIZE)
```

运行 m.update()，m.optimize() 即可打印查看输出结果中 x[i,j] 的值。

4.5 支持向量机建立 DVFS 决策模型

我们将获得的样本数据集合按照频率分为 L 类（L 为可供选择的频率数），相同频率的样本（Sample）作为一类。用支持向量机来建立分类模型，需要用到 LIBSVM 工具。

4.5.1 LIBSVM 简介

LIBSVM 是台湾大学林智仁（Chih-Jen Lin）教授等开发设计的一个简单、易于使用和快速有效的 SVM 模式识别与回归的软件包，它不但提供了编译好的可执行文件，还提供了源代码，方便改进、修改以及在其它操作系统上应用。另外，该软件提供了很多的默认参数，参数调节相对较少。该软件可以解决 C-SVM、v-SVM、ε-SVR 和 v-SVR 等问题，包括基于一对一算法的多类识别问题。

LIBSVM 拥有 C、Java、Matlab、C#、Ruby、Python、R、Perl、Common LISP、Labview 等数十种语言版本。源代码中通常直接提供 C、Java、Matlab、Python 接口。由于要将训练得到决策模型集成到 gem5 上，我们选择 Python 接口进行实验。

4.5.2 LIBSVM 使用

LIBSVM 使用的一般步骤是[59]：

（1）按照 LIBSVM 软件包所要求的格式准备数据集；

（2）对数据进行简单的缩放操作；

（3）考虑选用核函数；

（4）采用交叉验证选择最佳参数 C 与 γ；

（5）采用最佳参数 C 与 γ 对整个训练集进行训练获取支持向量机模型；

（6）利用获取的模型进行测试与预测。

我们在获得样本数据集后，通过脚本程序将数据整理成要求的格式：

$$< label > < index1 >:< value1 > < index2 >:< value2 > ...$$

其中，<label>是训练数据集的标签，例如我们把频率为1.2GHz的一类标记为-1，1.8GHz的一类标记为1。<index>是以1开始的整数，可以是不连续的。<value>为实数，是我们常说的自变量，也就是上面Parameter Vector中的元素值。

LIBSVM中Python接口提供了svm_read_problem()，svm_train()，svm_predict()，svm_save_model()等函数，可以通过参数选项选定核函数并调节C和γ的值，利用这些函数和合理的选项训练得到满意的分类模型，并保存为文件作为DVFS决策模型。

关于各函数的具体用法，以最重要的训练函数svm_train()为例，它的用法是：

$$svm-train\ \ [options]\ \ training\_set\_file\ \ [model\_file]$$

其中，$training_set_file$是按照固定格式整理的用于训练的数据文件，$model_file$是训练生成的模型，由自己命名。另外，options选项内容包括SVM类型选择，核函数选择，各参数值选择，交叉验证等。

4.6 集成验证

通过支持向量机得到DVFS决策模型后，我们需要将其集成到gem5中。这部分主要是代码修改工作。由于涉及的代码部分有点多，此处不再展示，只做简单介绍。

修改的代码中主要用到了LIBSVM中Python接口提供的svm_load_model()函数将训练好的决策模型文件读入，再从gem5中读入Parameter vector的各参数值，利用svm_predict()函数判断下阶段运行频率F，利用gem5中DVFS模块提供的perfLevel()函数改变运行频率（此处需要在python文件中调用C++实现的perfLevel()函数，通过swig接口）。集成后，发现gem5仿真时确实按照DVFS决策模型给出的频率执行程序，功能正确。

4.7 本章小结

本章介绍了研究的方法和流程，包括gem5、McPAT和LIBSVM等工具的使用，还介绍了gem5集成DVFS决策模型时代码修改的主要思想和用到的函数。按照这个流程，我们搭建起了一个仿真平台，可以随时验证不同DVFS决策模型的性能。

第五章 实验

5.1 参数配置

实验按照表5.1所示的参数配置进行仿真。在gem5的SE模式下，选择SPEC CPU2006中的程序作为仿真对象（benchmark）。

表5.1 仿真参数配置

参量	值
ISA	X86
Core Number	1
CPU	O3 CPU
L1 I-Cache Size	32KB
L1 D-Cache Size	64KB
L2 Cache Size	2MB

5.2 程序划分

用gem5执行仿真程序，我们按照一千万个指令划分程序段，部分程序的程序段划分结果如表5.2所示。

表5.2 程序段划分结果

程序	程序段数量
bzip2	97
gcc	708
mcf	675
astar	6345
lbm	2206
omnetpp	407

可以发现，不同的程序在按照相同的指令数划分时，其划分得到的程序段数量很不一致，其中 astar 程序划分成了 6345 段，而 bzip2 程序只有 97 段，这和程序本身有关系，同时和程序执行的输入文件有关系。同样的程序执行不同输入时，划分得到的程序段会不一致。我们让程序在不同频率下仿真执行，需要保证程序执行的任务是一致的。

图5.1是 bzip2 程序在 1.8GHz 频率下运行的结果。bzip2 程序总共被划分为 97 段，每运行一段输出一次仿真结果。在这些结果中，我们提取了 L2 Cache miss，ipc，仿真时间，功耗，能量这几个参量。从图中可以看出，程序执行过程各参量变化是密切相关的。其中，L2 Cache miss 升高，ipc 降低，仿真时间（sim_seconds）变长，动态功耗（Runtime Dynamic）降低，能量（Energy）升高，反之则相反。

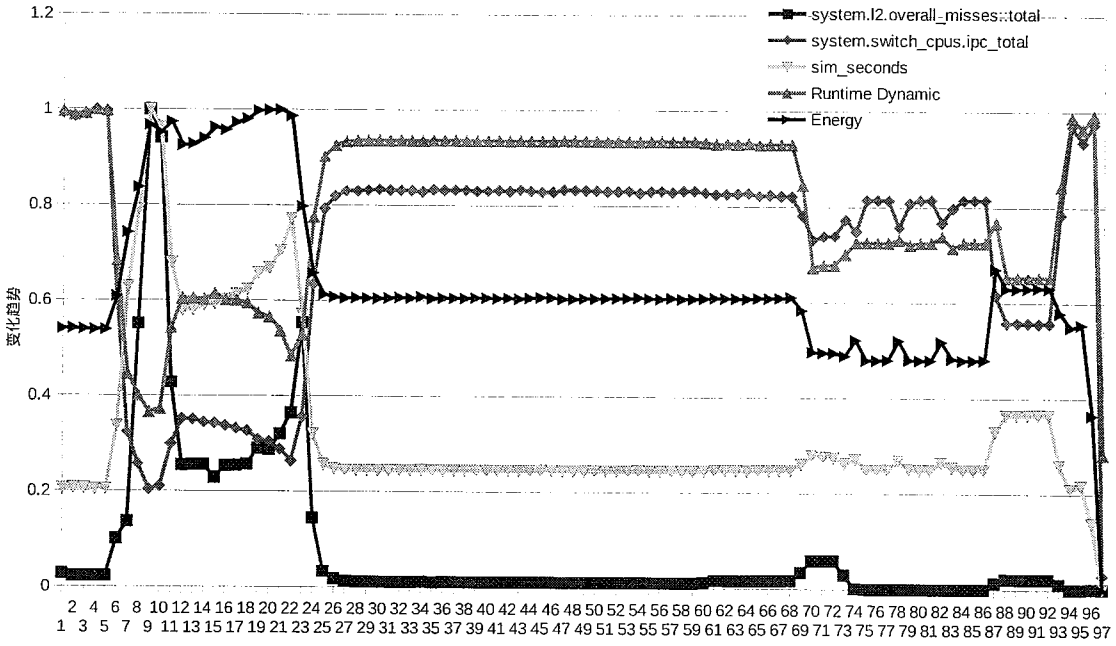


图 5.1 bzip2 程序 1.8GHz 频率下执行各参量变化趋势图

分析内在原因，L2 Cache miss 升高时，此时 CPU 需要频繁地从内存中读取数据，由于内存读取操作比 CPU 执行速度慢很多，造成 CPU 空闲等待时间（stall time）增多，执行指令速度变慢，ipc（Instruction per Cycle）降低，平均动态功耗降低。每个程序段都需要执行一千万条指令，ipc 降低导致执行时间变长。同时，综合效应导致执行需要的能量增多（主要原因是执行时间变长）。可以发现，执行相同任务量（一千万条指令），当发生很多 Cache miss 时，整个执行的速度变慢，能耗增

加。在这个过程中，因为CPU处于等待状态的时间比较长，我们通过DVFS技术降低处于等待状态的CPU的电压和频率来节省能量。

5.3 一些发现和结论

我们用四种不同频率（1.2GHZ到1.8GHz）分别执行仿真程序，其对应的电压如表5.3所示，这也是我们进行DVFS调节时所能选择的四个F/V档位。前面计算功耗的时候也是按照此表格频率对应的电压进行计算的。

表5.3 DVFS频率电压档位

频率(F)	电压(V)
1.2GHz	1.05V
1.4GHz	1.10V
1.6GHz	1.15V
1.8GHz	1.20V

如图5.2和图5.3所示。我们发现不同频率下执行相同代码段，它的参数如L2 Cache miss，ipc等高度重合。在误差允许范围内，我们认为其一致，即第四章中提到的针对每个程序段，我们认为其Parameter vector在误差范围内一致，体现了这段程序的特性。

L2 Cache miss和ipc能很好地反映程序的运行状态，在将程序分段后，可以得到各程序（benchmark）L2 Cache miss和ipc趋势图。图5.4(a)和图5.4(b)分别是bzip2程序各程序段的L2 Cache miss和ipc的连线图。除了上节讲到的两者变化趋势相反外，在具体数值上，bzip2各程序段执行的ipc基本在0.5以上，大部分在1.8左右，且绝大部分程序段的L2 Cache miss数很小，几乎可以忽略不计。

同样的，图5.5(a)和图5.5(b)分别是mcf程序各程序段的L2 Cache miss和ipc的连线图。此处和bzip2程序有很大不同。从图中可以发现，mcf程序的L2 Cache miss数绝大部分在20000左右，远远高于bzip2程序。而ipc数值绝大部分在0.2左右，远远低于bzip2的1.8。

实际上，我们把类似mcf的程序称为访存密集型（Memory Bound）程序，这类程序发生Cache miss很频繁，需要频繁访问主存读取数据，CPU常常处在等待状态，单位周期执行的指令数（ipc）少。我们把类似bzip2的程序称为计算密集型

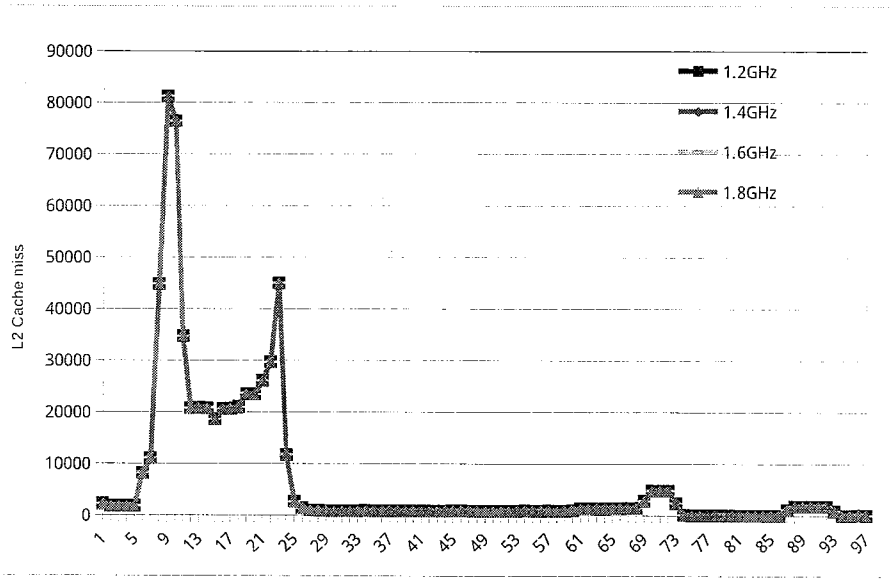


图 5.2 程序 bzip2 在不同频率下运行时 L2 Cache miss 变化趋势

（Compute Bound）程序，这类程序比较少发生 Cache miss，需要大量的计算操作，CPU 基本处于忙碌状态，单位周期执行的指令数（ipc）多。

除了上述的访存密集型和计算密集型的分类，我们在实验过程中选择了几种具有代表类型的程序。如图5.6、图5.7和图5.8所示。

其中，图5.6是 lbm 程序执行过程中参量的变化趋势，可以发现参量 L2 Cache miss 和 ipc 呈周期性变化。图5.7中 omnetpp 程序的参量在程序执行过程中波动非常小，基本维持在原状。而图5.8中 gcc 程序的参量在执行过程中波动特别大且基本无规律可循。

前面文献调研中，在相关研究中介绍的 Sherwood 等人提出的 Phase 概念就是基于类似 lbm 程序提出来的。这类程序有明显的周期规律，所以整个程序可以划分几种不同的 Phase。通过预测当前程序的 Phase 类别来确定 CPU 执行电压和频率。这种周期规律很可能是因为程序代码中的循环反复执行的结果。

相比而言，omnetpp 程序显得更加理想化，但是在最初选择好最佳执行频率后，不再需要 DVFS 技术更改 CPU 频率和电压，此时 DVFS 的调节颗粒度不再重要（因为只需要一次调节过程）。但是，类似这样的程序不具有普遍性。更加一般的情况是类似 gcc 程序那样，看起来杂乱无章，此时就需要更加复杂的调节策略了。

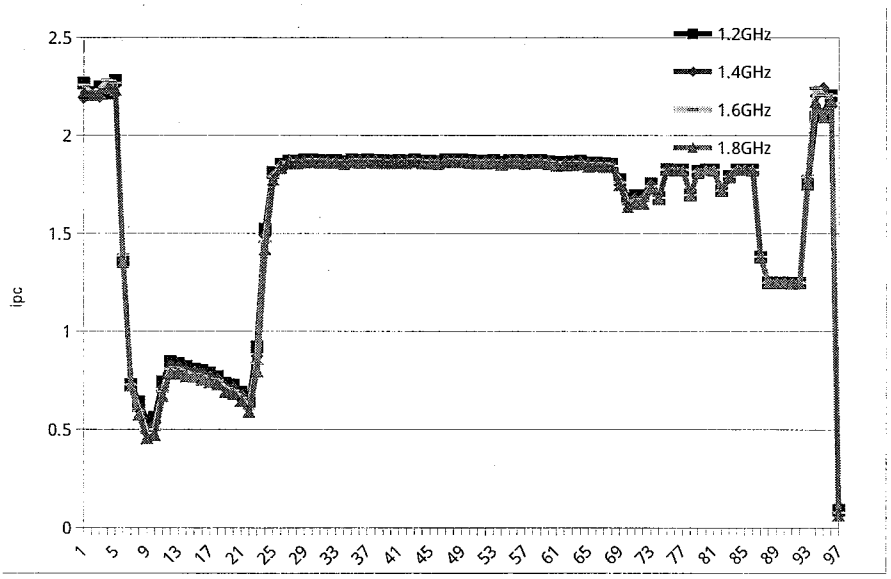
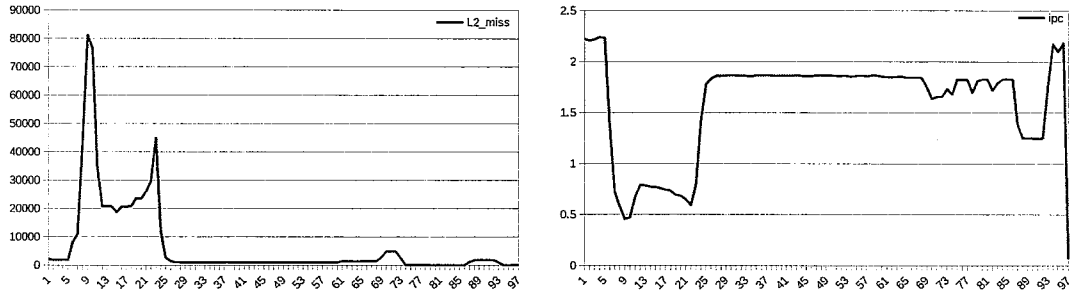


图 5.3 程序 bzip2 在不同频率下运行时 ipc 变化趋势

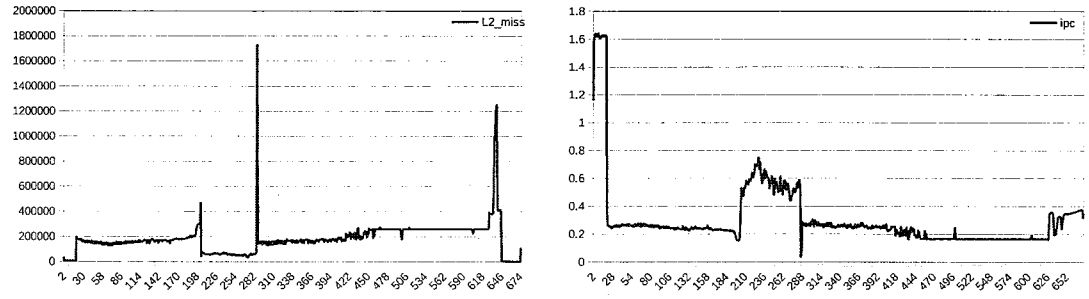


(a) L2 Cache miss　　(b) ipc

图 5.4 bzip2 程序 L2 Cache miss 和 ipc 参量变化趋势图

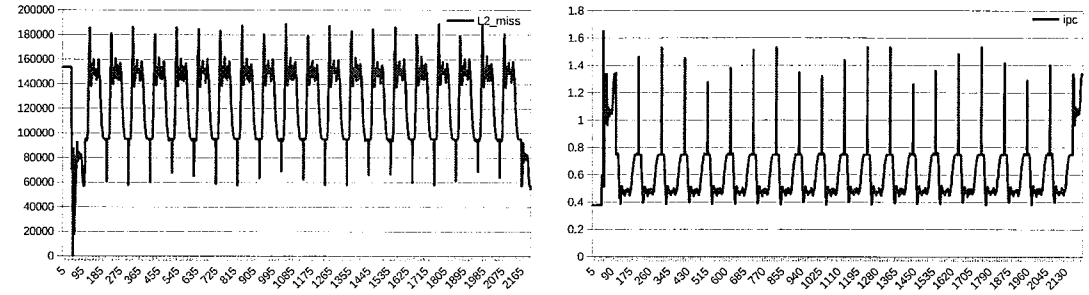
5.4 参量及核函数选择

在程序划分过程中，得到了一万左右的程序段（见表5.2），每个程序段包含一千万条指令。我们在实验中主要选择了 L2 Cache miss，ipc 等数据作为它的特征参量组成 Parameter vector。这样共获得了一万左右的样本。通过第四 章中介绍的 LIBSVM 工具用法，我们将样本分成训练集和测试集，训练得到了 DVFS 决策模型。构建过程中，我们发现使用线性核函数分类就能获得很好的分类预测结果，准确率高达 90% 以上。



(a) L2 Cache miss (b) ipc

图 5.5 mcf 程序 L2 Cache miss 和 ipc 参量变化趋势图



(a) L2 Cache miss (b) ipc

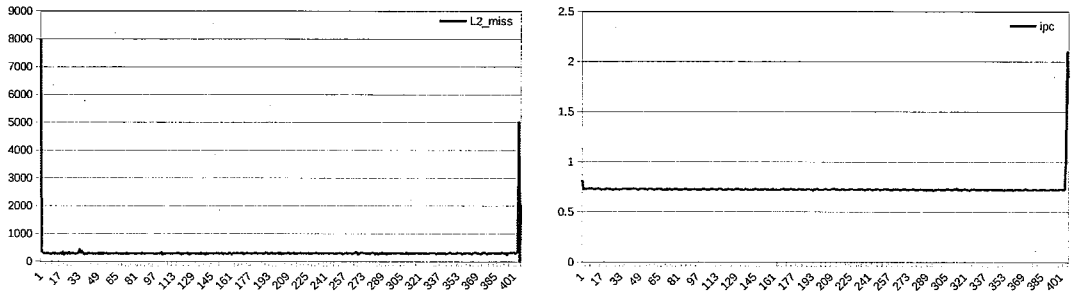
图 5.6 lbm 程序 L2 Cache miss 和 ipc 参量变化趋势图

5.5 实验结果

我们将决策模型集成到 gem5 中实现 DVFS 功能，得到实验结果。研究中分别尝试了两频率 DVFS 和四频率 DVFS，两频率 DVFS 可供选择的频率为 1.2GHz 和 1.8GHz。相比没有 DVFS 直接执行程序，DVFS 技术最多能节省将近 20% 的 EDP。

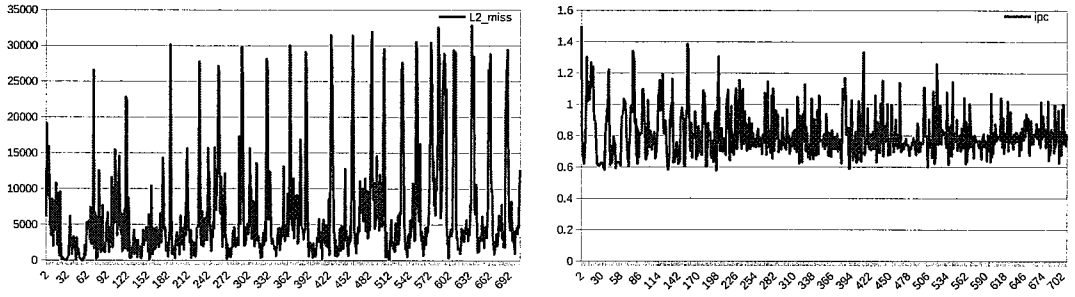
图5.9为程序 mcf 在 1.2GHz，1.8GHz 频率下单独运行消耗 EDP 和进行 DVFS 运行时消耗 EDP 的对比，纵坐标是消耗的总的 EDP 量，单位是焦耳秒（J*s）。因为 mcf 是典型的访存密集型（Memory Bound）程序，执行过程中会发生很多的 Cache miss，CPU 空闲等待时间长，所以更适合用较低频率 1.2GHz 来执行（消耗的 EDP 远少于 1.8GHz）。采用 DVFS 技术，利用 DVFS 决策模型做指导，发现其大部分程序段都选择用 1.2GHz 频率，极少部分程序段（此部分程序段的 ipc 较高，L2 Cache miss数少）采用了 1.8GHz频率。从图中可以看出，DVFS 消耗的 EDP 最少。

同上，图5.10为程序 bzip2 在 1.2GHz，1.8GHz 频率下单独运行消耗 EDP 和进



(a) L2 Cache miss　　(b) ipc

图 5.7 omnetpp 程序 L2 Cache miss 和 ipc 参量变化趋势图



(a) L2 Cache miss　　(b) ipc

图 5.8 gcc 程序 L2 Cache miss 和 ipc 参量变化趋势图

行 DVFS 运行时消耗 EDP 的对比。不同的是，bzip2 是典型的计算密集型（Compute Bound）程序，CPU 有大量的计算任务，大部分时间处在忙碌状态，也就是说对性能有很大的需求。这时候，需要更多地采用较高频率 1.8GHz 执行。实验结果也表明，在 DVFS 调节时，几乎所有的程序段都采用了 1.8GHz 的频率，所以图中 DVFS 执行消耗的 EDP 和 1.8GHz 执行消耗的 EDP 很接近。

更多程序的执行结果将在下面以表格形式给出。不管是访存密集型程序还是计算密集型程序，DVFS 技术总是能选择最佳的频率来执行程序，节省 EDP。内部其实是需要 DVFS 在执行调节时，做出最好的决策，有高的预测准确度。

5.5.1 两频率 DVFS 实验结果

研究采用支持向量机训练得到两频率 DVFS 决策模型。实验结果如表5.4 所示。其中第二列预测准确度表示构建的 DVFS 决策模型对下阶段电压频率（V/F）选择

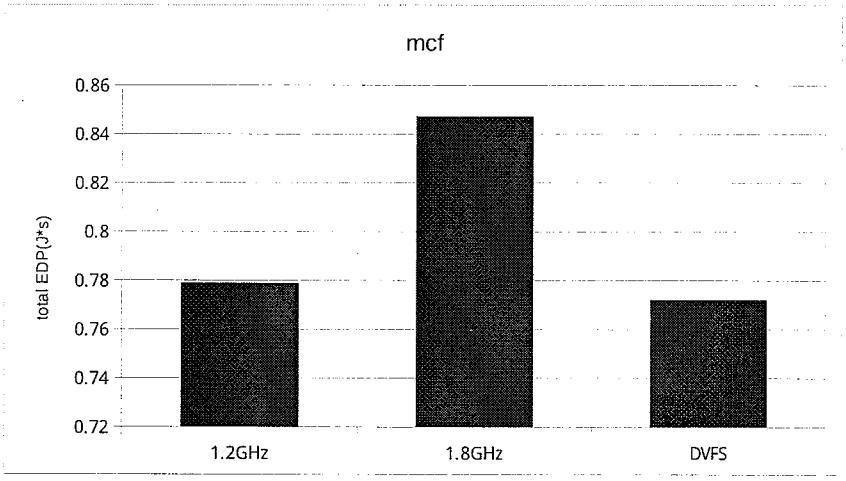


图 5.9 DVFS 执行 Memory Bound 程序 mcf

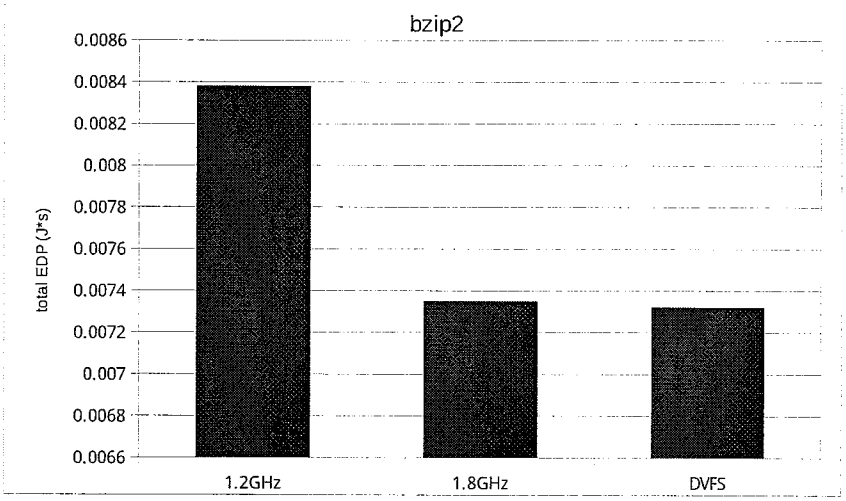


图 5.10 DVFS 执行 Compute Bound 程序 bzip2

的判断准确率，基本都在 95% 以上，依据程序不同，基本能够节省 10% 到 20% 的 EDP。

5.5.2 四频率 DVFS 实验结果

同样获得四频率 DVFS 实验结果如表5.5所示。相比两频率 DVFS，四频率 DVFS 中间多出了 1.4GHz 和 1.6GHz 两个频率选择，意味着更精细的调控。所以整体而言，四频率的 DVFS 能节省的 EDP 一般比两频率的多。但是，由于四频率 DVFS 在 SVM 训练得到决策模型时需要进行四分类预测，对分类预测的要求变高，分类准确性有所降低。

表 5.4 两频率 DVFS 实验结果

Benchmark	预测准确度	最多节省 EDP
bzip2	95.88%	12.64%
gcc	100%	16.26%
mcf	99.70%	8.87%
omnetpp	100%	17.54%
astar	100%	16.83%

表 5.5 四频率 DVFS 实验结果

Benchmark	预测准确度	最多节省 EDP
bzip2	95.83%	12.72%
gcc	99.01%	16.27%
mcf	88.15%	18.79%
omnetpp	100%	17.55%
astar	99.95%	16.83%

5.6 本章小结

本章介绍了实验的平台，参数配置，及最终的实验结果。并对实验过程中的数据和结果进行了分析。实验证明，支持向量机训练得到的 DVFS 决策模型平均预测准确度高达 90% 以上，根据执行程序不同最多可以节省将近 20% 的 EDP。

第六章 总结与展望

6.1 总结

随着集成电路的发展，时至今日，芯片功耗过高带来的挑战严重阻碍着整个行业的发展，各种低功耗技术纷纷诞生，其中，动态电压频率调节（DVFS）技术由于其良好的性能受到了广泛的关注和研究。

本文通过采用动态电压频率调节技术（DVFS）来对系统中的用电大户CPU芯片进行功耗管理。为了在节省能量的同时兼顾性能，我们以能量和延迟的内积（Energy Delay Product，简称EDP）作为优化目标，通过gem5和McPAT仿真工具的结合使用得到相关数据，引入机器学习领域的支持向量机（SVM）来训练得到DVFS决策模型，最终将决策模型应用到gem5的DVFS功能中，实现功耗管理的目的。实验证明，支持向量机训练得到的DVFS决策模型平均预测准确定要高达90%以上，根据执行程序不同最多可以节省将近20%的EDP。

不得不说的是，研究中存在如下几个问题有待提高：

1. DVFS延迟时间和程序段划分精度的问题。实验过程中，我们将程序段按照一千万个指令划分，由于一般DVFS延迟时间在几十个微妙，相比于程序段执行时间，延迟时间可以忽略不计。所以我们没有考虑DVFS延迟时间给DVFS策略制定带来的影响。但是，更好的办法应该是结合DVFS延迟时间考虑程序段划分精细度，做更精细的DVFS调节。

2. 研究过程中将每个程序段看成是独立的，忽略了机器当前状态对程序执行时的影响。机器不同状态下执行同一段代码，产生的输出会有一些不同。

3. 研究中将上一程序段的最佳频率选择作为下一个程序段的输出结果，虽然程序有连续性，但是在频率变化非常频繁或者在比较精细度的程序段划分情况下将带来误差。

4. 静态功耗。McPAT工具计算静态功耗不太准确，最终静态功耗采用了动态功耗的10%进行估计。而随着晶体管越来越小，静态功耗所占的比重越来越大，模型更加精确需要有新的静态功耗数据。

5. 只考虑了单核的情况。文章中只考虑了单核的 DVFS，在多核情况下，因为核与核之间可能发生交互，更好的方法是综合各个核的信息做全局处理，DVFS 策略可以更加复杂，譬如可以结合任务调度进行 DVFS 调节等。

6.2 展望

总体看来，随着集成电路的密度不断增大，DVFS 技术可以生存的时间越来越短。因为当工艺降低到一定程度后，电压的调整范围已经很小了，DVFS 的效果也会下降。面对当前的功耗困境，为了继续提升芯片性能，更好的突破方向是多核和高能效的微架构。

我们知道处理器性能是主频和 ipc（Instruction Per Cycle）的乘积决定的，其中主频是处理器运行频率，代表处理器单位时间运行计算的次数，ipc 代表每个周期 CPU 能能执行的指令数。在第二章中 DVFS 的原理介绍中，我们已经知道功耗差不多和频率的三次方成正相关，增加频率会让功耗急剧增加，现在已经遇到了瓶颈。所以考虑增加 ipc。

要提高 ipc，可以通过提高指令执行的并行度来实现。具体分为两个方面：一是多核技术，通过集成多个 CPU 或 CPU + GPU 的方式实现指令的并行执行。二是修改处理器的微架构，通过改进流水线结构，引进多线程技术，或者增加专门的硬件结构（如加速器）增加指令执行的并行程度。两者对功耗的需求都比提高频率少得多。

虽然，DVFS 技术治标不治本，但是在一定时间范围内，它能很好地缓解功耗过高带来的压力，不失为一味好的镇痛剂。同时我也相信未来会有更好的解决方法，继续提高芯片的性能。

参考文献

[1] 朱正涌. 半导体集成电路[M]. 北京：清华大学出版社有限公司, 2001.

[2] MOORE G E. Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp. 114 ff.[J]. IEEE Solid-State Circuits Newsletter, 2006, 3(20): 33–35.

[3] DENNARD R H, RIDEOUT V, BASSOUS E, et al. Design of ion-implanted MOSFET's with very small physical dimensions[J]. Solid-State Circuits, IEEE Journal of, 1974, 9(5): 256–268.

[4] ESMAEILZADEH H, BLEM E, AMANT R S, et al. Dark silicon and the end of multicore scaling[J]. IEEE Micro, 2012(3): 122–134.

[5] 陈静华. SOC芯片低功耗设计[D]. [S.l.]：万方数据资源系统, 2005.

[6] WU Q, PEDRAM M, WU X. Clock-gating and its application to low power design of sequential circuits[J]. Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, 2000, 47(3): 415–420.

[7] HU Z, BUYUKTOSUNOGLU A, SRINIVASAN V, et al. Microarchitectural Techniques for Power Gating of Execution Units[J], 2004.

[8] WEISER M, WELCH B, DEMERS A, et al. Scheduling for reduced CPU energy[G] // Mobile Computing. [S.l.]: Springer, 1994: 449–471.

[9] GOVIL K, CHAN E, WASSERMAN H. Comparing algorithm for dynamic speed-setting of a low-power CPU[C] // Proceedings of the 1st annual international conference on Mobile computing and networking. 1995: 13–25.

[10] PERING T, BURD T, BRODERSEN R. The simulation and evaluation of dynamic voltage scaling algorithms[C] // Proceedings of the 1998 international symposium on Low power electronics and design. 1998: 76–81.

[11] GRUNWALD D, MORREY III C B, LEVIS P, et al. Policies for dynamic clock scheduling[C] // Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4. 2000 : 6 – 6.

[12] KALIBER A. The Technology Behind CrusoeTM Processors[J]. Transmeta Corporation WhitePaper, 2000.

[13] FLAUTNER K, MUDGE T. Vertigo: Automatic performance-setting for linux[J]. ACM SIGOPS Operating Systems Review, 2002, 36(SI) : 105 – 116.

[14] FLAUTNER K, REINHARDT S, MUDGE T. Automatic performance setting for dynamic voltage scaling[C] // Proceedings of the 7th annual international conference on Mobile computing and networking. 2001 : 260 – 271.

[15] LORCH J R, SMITH A J. Improving dynamic voltage scaling algorithms with PACE[C] // ACM SIGMETRICS Performance Evaluation Review : Vol 29. 2001 : 50 – 61.

[16] LORCH J R, SMITH A J. Using user interface event information in dynamic voltage scaling algorithms[C] // Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on. 2003 : 46 – 55.

[17] LORCH J R, SMITH A J. Operating system modifications for task-based speed and voltage[C] // Proceedings of the 1st international conference on Mobile systems, applications and services. 2003 : 215 – 229.

[18] 王彪, 王小鸽. 功耗管理中的动态电压调整综述[J]. 计算机应用研究, 2007, 24(8) : 8 – 12.

[19] CHOI K, SOMA R, PEDRAM M. Dynamic voltage and frequency scaling based on workload decomposition[C] // Proceedings of the 2004 international symposium on Low power electronics and design. 2004 : 174 – 179.

[20] CHOI K, SOMA R, PEDRAM M. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access

to on-chip computation times[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2005, 24(1): 18–28.

[21] SHERWOOD T, PERELMAN E, HAMERLY G, et al. Discovering and exploiting program phases[J]. Micro, IEEE, 2003, 23(6): 84–93.

[22] SHERWOOD T, SAIR S, CALDER B. Phase tracking and prediction[C] //ACM SIGARCH Computer Architecture News: Vol 31. 2003: 336–349.

[23] LAU J, SCHOENMACKERS S, CALDER B. Transition phase classification and prediction[C] //High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on. 2005: 278–289.

[24] ISCI C, BUYUKTOSUNOGLU A, MARTONOSI M. Long-term workload phases: Duration predictions and applications to DVFS[J]. Micro, IEEE, 2005, 25(5): 39–51.

[25] PARK J, SHIN D, CHANG N, et al. Accurate modeling and calculation of delay and energy overheads of dynamic voltage scaling in modern high-performance microprocessors[C] //Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design. 2010: 419–424.

[26] BINKERT N, BECKMANN B, BLACK G, et al. The gem5 simulator[J]. ACM SIGARCH Computer Architecture News, 2011, 39(2): 1–7.

[27] SPILIOPOULOS V, BAGDIA A, HANSSON A, et al. Introducing DVFS-management in a full-system simulator[C] //Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on. 2013: 535–545.

[28] CORTES C, VAPNIK V. Support-vector networks[J]. Machine learning, 1995, 20(3): 273–297.

[29] SMOLA A J, SCHÖLKOPF B. A tutorial on support vector regression[J]. Statistics and computing, 2004, 14(3): 199–222.

[30] GUNN S R, OTHERS. Support vector machines for classification and regression[J]. ISIS technical report, 1998, 14.

[31] 王国胜. 支持向量机的理论与算法研究[D]. 北京：北京邮电大学, 2007.

[32] CRISP C J B D J. Uniqueness of the SVM solution[C] // Advances in Neural Information Processing Systems 12: Proceedings of the 1999 Conference: Vol 12. 2000: 223.

[33] 荣海娜, 张葛祥, 金炜东. 系统辨识中支持向量机核函数及其参数的研究[J]. 系统仿真学报, 2006, 18(11): 3204－3208.

[34] 郭丽娟, 孙世宇, 段修生. 支持向量机及核函数研究[J]. 科学技术与工程, 2008, 8(2): 487－490.

[35] 奉国和. SVM 分类核函数及参数选择比较[J]. 计算机工程与应用, 2011, 47(3): 123－124.

[36] AMARI S-I, WU S. Improving support vector machine classifiers by modifying kernel functions[J]. Neural Networks, 1999, 12(6): 783－789.

[37] BURGES C J. A tutorial on support vector machines for pattern recognition[J]. Data mining and knowledge discovery, 1998, 2(2): 121－167.

[38] ROOBAERT D, VAN HULLE M M. View-based 3d object recognition with support vector machines[C] // Neural Networks for Signal Processing IX, 1999. Proceedings of the 1999 IEEE Signal Processing Society Workshop.. 1999: 77－84.

[39] SCHÖLKOPF B, SUNG K-K, BURGES C J, et al. Comparing support vector machines with Gaussian kernels to radial basis function classifiers[J]. Signal Processing, IEEE Transactions on, 1997, 45(11): 2758－2765.

[40] 卢增祥, 李衍达. 交互支持向量机学习算法及其应用[J]. 清华大学学报: 自然科学版, 1999, 39(7): 93－97.

[41] OSUNA E, FREUND R, GIROSI F. Training support vector machines: an application to face detection[C] // Computer vision and pattern recognition, 1997. Proceedings., 1997 IEEE computer society conference on. 1997: 130－136.

[42] ZHANG S, QIAO H. Face recognition with support vector machine[C] // Robotics, Intelligent Systems and Signal Processing, 2003. Proceedings. 2003 IEEE International Conference on: Vol 2. 2003: 726–730.

[43] KUMAR V P, POGGIO T. Learning-based approach to real time tracking and analysis of faces[C] // Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on. 2000: 96–101.

[44] HUANG J, SHAO X, WECHSLER H. Face pose discrimination using support vector machines (SVM)[C] // Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on: Vol 1. 1998: 154–156.

[45] 田盛丰, 黄厚宽. 基于支持向量机的数据库学习算法[J]. 计算机研究与发展, 2000, 37(1): 17–22.

[46] BRADLEY P S. Mathematical programming approaches to machine learning and data mining[D]. [S.l.]: UNIVERSITY OF WISCONSIN, 1998.

[47] SUYKENS J A, VANDEWALLE J, DE MOOR B. Optimal control by least squares support vector machines[J]. Neural Networks, 2001, 14(1): 23–35.

[48] 刘江华, 程君实, 陈佳品. 支持向量机训练算法综述[J]. 信息与控制, 2002, 31(1): 45–50.

[49] JOACHIMS T. Svmlight: Support vector machine[J]. SVM-Light Support Vector Machine http://svmlight. joachims. org/, University of Dortmund, 1999, 19(4).

[50] HSU C-W, LIN C-J. BSVM, 2006[J]. Software available at http://www. csie. ntu. edu. tw/cjlin/bsvm, .

[51] CHANG C-C, LIN C-J. LIBSVM: a library for support vector machines[J]. ACM Transactions on Intelligent Systems and Technology (TIST), 2011, 2(3): 27.

[52] RÜPING S. mysvm–a support vector machine[J]. University of Dortmund, Computer Science, URL http://www-ai. cs. uni-dortmund. de/SOFTWARE/MYSVM/index. html, 2004.

[53] DUIN R, JUSZCZAK P, PACLIK P, et al. A matlab toolbox for pattern recognition[J]. PRTools version, 2000, 3.

[54] BEAZLEY D M, OTHERS. SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++.[C] // Tcl/Tk Workshop. 1996.

[55] HENNING J L. SPEC CPU2006 benchmark descriptions[J]. ACM SIGARCH Computer Architecture News, 2006, 34(4): 1 – 17.

[56] CHADHA A R, PANDEY A, PRAKASH A. ECE 752-Advanced Computer Architecture I[J], 2014.

[57] LI S, AHN J H, STRONG R D, et al. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures[C] // Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture. 2009: 469 – 480.

[58] OPTIMIZATION G, OTHERS. Gurobi optimizer reference manual[J]. URL: http://www. gurobi. com, 2012.

[59] HSU C-W, CHANG C-C, LIN C-J, et al. A practical guide to support vector classification[J], 2003.

数学公式

$\sum(\cdot)$	求和
$\|(\cdot)\|$	绝对值
$(\cdot)^n$	n次方
$\in$	属于
$\forall$	任意
$\subset$	包含于
$\{\cdot\}$	集合
$\min(\cdot)$	最小值
$\|\|(\cdot)\|\|$	向量求模
$\max(\cdot)$	最大值
$\exp(\cdot)$	自然指数运算
$s.t.$	限制条件
$\frac{\partial(\cdot)}{\partial(\cdot)}$	求偏导
$sgn(\cdot)$	符号函数
SV	支持向量
$\kappa(\cdot)$	核函数
$\langle\cdot\rangle$	内积
$\tanh(\cdot)$	双曲正切
$(\cdot)^T$	向量转置

McPAT需从stats.txt中提取的参量数据

system.cpu_clk_domain.clock

system.cpu.numCycles

system.cpu.idleCycles

system.cpu.iq.iqInstsIssued

system.cpu.iq.FU_type_0::No_OpClass

system.cpu.iq.FU_type_0::IntAlu

system.cpu.iq.FU_type_0::IntMult

system.cpu.iq.FU_type_0::IntDiv

system.cpu.iq.FU_type_0::IprAccess

system.cpu.iq.FU_type_0::FloatAdd

system.cpu.iq.FU_type_0::FloatCmp

system.cpu.iq.FU_type_0::FloatCvt

system.cpu.iq.FU_type_0::FloatMult

system.cpu.iq.FU_type_0::FloatDiv

system.cpu.iq.FU_type_0::FloatSqrt

system.cpu.branchPred.condPredicted

system.cpu.branchPred.condIncorrect

system.cpu.iq.FU_type_0::MemRead

system.cpu.iq.FU_type_0::InstPrefetch

system.cpu.iq.FU_type_0::MemWrite

system.cpu.commit.committedInsts

system.cpu.commit.int_insts

system.cpu.commit.fp_insts

system.cpu.rob.rob_reads

system.cpu.rob.rob_writes

system.cpu.rename.int_rename_lookups

system.cpu.rename.RenamedOperands

system.cpu.rename.RenameLookups

system.cpu.rename.fp_rename_lookups

system.cpu.iq.int_inst_queue_reads

system.cpu.iq.int_inst_queue_writes

system.cpu.iq.int_inst_queue_wakeup_accesses

system.cpu.iq.fp_inst_queue_reads

system.cpu.iq.fp_inst_queue_writes

system.cpu.iq.fp_inst_queue_wakeup_accesses

system.cpu.int_regfile_reads

system.cpu.fp_regfile_reads

system.cpu.int_regfile_writes

system.cpu.fp_regfile_writes

system.cpu.commit.function_calls

system.cpu.workload.num_syscalls

system.cpu.iq.int_alu_accesses

system.cpu.iq.fp_alu_accesses

system.cpu.itb_walker_cache.tags.tag_accesses

system.cpu.itb_walker_cache.no_allocate_misses

system.cpu.icache.ReadReq_accesses::cpu.inst

system.cpu.icache.ReadReq_misses::cpu.inst

system.cpu.icache.tags.replacements

system.cpu.dtb_walker_cache.tags.data_accesses

system.cpu.dtb_walker_cache.no_allocate_misses

system.cpu.dcache.ReadReq_accesses::cpu.data

system.cpu.dcache.WriteReq_accesses::cpu.data

system.cpu.dcache.ReadReq_misses::cpu.data

system.cpu.dcache.WriteReq_misses::cpu.data

system.cpu.dcache.tags.replacements

system.cpu.branchPred.BTBLookups

system.cpu.commit.branches

system.l2.ReadReq_accesses::total

system.l2.ReadExReq_accesses::total

system.l2.ReadReq_misses::total

system.l2.ReadExReq_misses::total

system.l2.tags.replacements

system.l2.demand_accesses::total

system.l2.overall_misses::total

简　历

基本情况

廖凯，男，湖南省邵东县人，1990年12月出生，中国科学院上海微系统与信息技术研究所在读硕士研究生。

教育状况

2009年9月至2013年7月，浙江大学信息与电子工程学系，本科，专业：电子科学与技术。

2013年9月至2016年6月，中国科学院上海微系统与信息技术研究所，硕士研究生，专业：微电子与固体电子学。

工作经历

2012年6月至2012年8月，浙江通信产业服务有限公司（实习），职位：软件工程师

2012年11月至2013年5月，三星半导体杭州研究所（实习），职位：SQA部门测试工程师

研究兴趣

计算机体系结构，芯片功耗管理

研究成果

电子设计工程　“支持向量机（SVM）算法用于计算机CPU功耗管理”（待刊登）

联系方式

通讯地址：上海市徐汇区岳阳路319号八号楼104室

邮编：200030

E-mail: liaokai901218@163.com