中国科学院大学
University of Chinese Academy of Sciences

博士学位论文

神经网络加速器研究

作者姓名：杜子东

指导教师：陈云霁　研究员

中国科学院计算技术研究所

学位类别：工学博士

学科专业：计算机系统结构

研 究 所：中国科学院计算技术研究所

二〇一六年五月

Study of Hardware Neural Network Accelerators

A Dissertation Submitted to

The University of Chinese Academy of Sciences

in Partial Fulfillment of the Requirement

for the Degree of

Doctor of Philosophy

in

Computer Architecture

by

Zidong Du

Dissertation Supervisor: Professor Yunji Chen

Institute of Computing Technology

Chinese Academy of Sciences

May, 2016

声　　明

我声明本论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，本论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

作者签名：[signature]　　日期：2016.03.29

论文版权使用授权书

本人授权中国科学院计算技术研究所可以保留并向国家有关部门或机构送交本论文的复印件和电子文档，允许本论文被查阅和借阅，可以将本论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编本论文。

（保密论文在解密后适用本授权书）

作者签名：[signature]　导师签名：[signature]　日期：2016.4.1

摘 要

得益于神经网络算法研究的进步，神经网络在许多场景下的精度表现优异，被广泛应用在图像识别、语音识别、自然语言理解、广告推荐乃至计算机围棋中。神经网络本身具有数据密集和计算密集的特性，传统的处理平台上如通用CPU/GPU上难以高效处理神经网络。因此本文对新型的专用神经网络加速器结构进行了探索。具体主要包括以下三方面的贡献：

神经网络算法硬件化的比较：目前，神经网络加速器研究者所关注的神经网络算法来自两个截然不同的领域：机器学习领域和神经生物学。这两类神经网络算法具有极其不一样的特性，哪类神经网络算法更加适合实现成为硬件加速器也是一个存在争议的问题。我们首次从硬件角度系统地比较了这两类神经网络算法的研究，实现了两类神经网络算法的硬件逻辑设计，并完成后端布局布线工作，在此基础上比较这两类方法的硬件相关参数，如能耗、速度、面积开销、精度和功能性。我们的研究结果表明：在同样的精度需求下，神经生物学启发的神经网络（如SNN+STDP）的面积开销显著大于机器学习启发的神经网络（如MLP+BP）。另外，我们也发现，对于非常大规模的神经网络和对于精度要求不高的应用，SNN+STDP相较于MLP+BP的硬件开销更小。

CNN加速器：传统认为神经网络加速器的能效和性能受到内存访问的限制。为规避此限制，我们专门针对图像识别领域中处在领先地位的算法*卷积神经网络*(Convolutional Neural Networks, CNN)设计了一款加速器，名为ShiDianNao。卷积神经网络具有一个重要的性质：突触连接的权值被多个神经元共享，这大大的减少了神经网络的权值数量，从而大大的降低了卷积神经网络的内存占用。这使得片上SRAM能够容纳下整个CNN，从而消除掉所有对权重的SRAM访问。进一步，将ShiDianNao放置在图像传感器的旁边，也即能够消除剩余的所有由输入图像引起的内存访问。在65nm工艺下，我们完成单核设计直至后端进行评估，得到ShiDianNao的面积为5.94 mm^2，功耗为336 mW，比高端GPU（Nvidia K20M）快约30倍。对于实时处理具有更高的分辨率和帧率的视频流，我们进一步提出了多核ShiDianNao（ShiDianNao+）来提升性能。

非精确神经网络加速器：对于许多能够容忍一定结果误差的应用程序来说，***非精确计算（inexact computing）***被认为是降低能耗最有效的手段之一。但过去的研究中，非精确加速器应用范围狭窄且对不精确度的耐受性往往是固定的，缺乏灵活性。为应对此问题，我们提出将非精确计算与硬件神经网络相结合，从而设计出非精确神经网络加速器，从而扩大应用范围，提高错误恢复能力和资源节约的规模。得益于我们的*重训练方法*，我们的非精确神经网络加速器相比于精确的神经网络加速器，可以实

现42.82%-62.55%的能源消耗节约（延时和面积分别节省18.70%和31.51%），同时精度损失的代价则很小（均方误差MSE平均从0.14增加到0.20）。

关键词：神经网络；硬件加速器；神经生物学；机器学习；嵌入式系统，非精确计算

Study of Hardware Neural Network Accelerators

Zidong Du (Computer Architecture)
Directed by Yunji Chen

Due to the rapid process made in neural network algorithms which achieve the state-of-the-art performance on accuracy in many aspects of application, e.g., image recognition, voice recognition, natural language processing, ADs recommendation and even computer Go. Neural network algorithms are computation-intense and memory-intense, thus traditional CPUs/GPUs can not manage the processing highly efficiently. In this thesis, we focus on three main topics about hardware neural network accelerators:

Neuromorphic Accelerator: The mainly considered neural network algorithms are coming from two largely separate domains: machine-learning and neuroscience. The question about which approach should be considered as potential hardware implementation raised while these networks are very different and yet few researchers compare them directly from a hardware perspective. Here, we implement both approach down to layout and compare about the hardware characteristics, e.g., energy, speed, area cost, accuracy and functionality. In this study, for current neural networks inspired from machine-learning and neuroscience, with best effort at hardware implementation and same workload, our study helps dispel the notion that neuroscience inspired neural networks, such as SNN+STDP, are current comparable with neural networks inspired from machine-learning, such as MLP+BP: not only in accuracy, but also in hardware overhead. We report that for very large scale neural networks and application require only moderate accuracy, SNN+STDP can be more efficient than MLP+BP, thus be a more attractive option.

CNN accelerator ShiDianNao: It is shown that for a broad application scope neural networks accelerators can achieve both high performance and high efficiency. However, both performance and efficiency are still limited by memory accesses. In this study, we focus on the most important class of application in recognition and mining area, i.e., image applications. Convolutional neural networks (CNN) have been proven to be the state-of-the-art algorithm for these applications and they have a very important property that synaptic weights are shared by many neurons which reduce the total storage significantly. It allows to store a whole CNN within only on-chip SRAM while elimination most possible DRAM accesses. Further placing this accelerator closer to the sensor, it can eliminate remaining memory accesses (inputs and outputs). In this study, we propose ShiDianNao, a CNN accelerator which can be placed close

to a CMOS/CCD sensor. With all possible DRAM accesses eliminated, carefully designed optimized pattern of data accesses, ShiDianNao is a highly energy-efficient accelerator. With TSMC 65 nm technology, we implement a single-core design down to layout and report a 5.94 mm^2 modest footprint and only 336 mW in power, but still about 30× faster than high-end GPUs. For visual processing with higher resolution and frame-rate requirements, we further present ShiDianNao+, a multi-core implementation with elevated performance.

Inexact Neural Network Accelerator: For many applications which can tolerant a certain degree of error, *inexact computing* are drawing increasing attentions and regarded as the most promising approach for saving energy. By trade some accuracy for significant resources (energy, critical path delay, silicon area), this approach has been limited to ASICs with the driven of such principle. Current ASICs have very limited application scope and are crucial to error which determine the mount of potential resource savings. In this study, we achieve broad application scope, error resilience together with energy savings by combining inexact computing with hardware neural networks. For heterogeneous multi-core platforms, hardware neural networks are emerging as future promising accelerators with flexible error resilience achieved by *tranning*. With TSMC 65nm technology, we implement inexact hardware neural network down to layout and results show 42.82%-62.55% savings in energy consumption, 18.70% savings in corresponding delay and 31.51% savings in area respectively, with a small accuracy loss (MSE increases from 0.14 to 0.20 on average) when compared to baseline neural network implementation.

Keywords: Neural Networks; Hardware Accelerator; Neuroscience; Machine Learning; Embedded System; Inexact Computing

目 录

图目录

表目录

第一章 引言

1.1 硬件神经网络加速器的研究背景

1.1.1 硬件加速

自从数字芯片诞生以来，得益于制造工艺的不断提升，单个晶体管的面积和成本大幅度缩减，数字芯片得以在越来越小的尺寸单位（从微米到纳米）上实现，从而使得晶圆单位面积上的计算能力持续获得大幅度提升。芯片飞速发展的计算能力使得大规模处理数据的能力大幅提升。这不单单使得计算机学科和产业飞速发展，同时也大幅度推动其他学科和产业的发展。相应的，飞速发展的学科也不断对计算能力提出新的要求。

随着制造工艺制程的提升，工艺迈入深亚微米领域，通过提升主频获得更高的芯片处理性能的时代一去不复返。处理器芯片上的晶体管数目及峰值功耗随着摩尔定律发展不断增加，然而随着时钟频率的提升，泄漏电流增加，芯片的静态功耗急剧增加。简单来讲，电压和时钟主频具有正比关系，根据晶体管功耗的估算公式大致可以得出增加的功耗大大超出了获得的性能的增加。INTEL的研究[6]指出，大约3%的功耗增加才能带来1%的性能提升。同时，芯片单位面积上有限的散热能力也制约了单位面积的封装功耗，这使得芯片各个区域无法同时全速地进行计算，也即“暗硅”（dark silicon）[57,194]现象。目前晶体管制造工艺仍在快速发展，采用14nm制造工艺的嵌入式芯片已经全面商业化，而INTEL也在积极研究7nm锗基制造工艺，加之3D堆叠（3D-stack）等新兴技术的出现和发展，在可预计的将来，芯片的集成度仍将提升，功耗密度（Power Density）也将进一步增加，这使得性能和功耗的矛盾也将更加突出。

学术界和工业界很早就开始尝试不同的策略来使得芯片具有更高的性能功耗比，这其中最有代表性的当属采用多核多处理器的方案来提升处理的并行度。IBM在1980年发布的x86架构就已经包含多处理器系统，在2001年发布的双核RISC处理器POWER4即为多核处理器架构。HP、SUN、AMD和INTEL则在2004~2006年间分别发布多核处理器，并此后相继提出异构处理器架构。异构多核的架构实现包括采用现场可编程门阵列（Field Programmable Gate Array，FPGA）、图形加速器（Graphic Processing Unit, GPU）、数字信号处理器（Digital Signal Processor，DSP）、专用定制指令集处理器（Application Specific Instruction Set Processor，ASIP）和专用定制集成电路（Application Specific Integrated Circuits, ASIC）。目前学术界和工业界具有这样的共识：体系结构需要用加速器避免“暗硅”，即针对特定应用或算法定制电路[19]。针对细分的不同的应用场景，加速器或者能够极大地提升计算性能，如数据中心的应用或者科

学计算，有研究表明在某些应用上加速器能够带来高达50至1000倍的能效提升[76]；或者能极其高效地完成特定的任务功能，如移动嵌入式平台上的应用，有研究表明加速器相对通用处理器能够节省上千倍的功耗[50]。

1.1.2 硬件加速结构

现有的商业使用中的加速器最常见的是GPU、DSP、ASIP、FPGA（含CGRA，弹性粗粒度可配置电路，Elastic Coarse-Grained Recongurable Array）和ASIC五类。

GPU本质上是一种由大量能够处理简单任务的核心构成的阵列。相对于传统的通用核心，GPU的核心非常简单，一般不具备分支预测等复杂的程序流控制单元，因此对于分支较多、控制复杂的程序，效率会比较差。但GPU的核心数量大，因而能提供更高的峰值运算能力（目前NVIDIA的K40双精度浮点运算已经达到了1.4TFlops）。目前，NVIDIA和AMD等厂商都在大力推动基于GPU的应用加速方案，并已在科学计算和机器学习等领域取得了较好的效果。相关的并行编程开放标准包括CUDA、OpenCL和ATI STREAM等。

DSP是专门为数字信号处理所设计的较为专用的可编程器件，广泛用于通信与信息系统、信号与信息处理、自动控制、雷达、军事、航空航天、医疗、家用电器等领域。DSP一般采用哈佛结构，使用应用或领域专用的超长指令字指令集。DSP往往依赖程序员手工对指令进行精心排布，提高其程序效率。不同的程序员编写的DSP程序，其效率可能相差甚远[174]。

ASIP是面向领域定制的处理器核，其指令集为目标应用类进行缩减和扩充。ASIP可以权衡通用核的灵活性和ASIC的性能[121]，因而在无线通信、音视频处理等领域有着很广泛的应用。这些领域有很多标准（并且这些标准还在不断发展），对于一个产品高效地处理一大类标准的需求则需要兼具领域通用性和高性能的ASIP。

FPGA是一种用途较广的可配置电路，它包含大量的CLB（Configurable Logic Block，可配置逻辑模块），通过编程可决定CLB自身功能和相互间的连线方式，从而完成复杂的电路功能。对于很多应用，FPGA可以达到比通用核更高的效率。如江南计算所和国防科大合作实现了一种基于FPGA的共轭梯度求解器，相对于通用核性能可提高4.62-9.24倍[198]。国防科大的研究表明，对于LU分解这类常见应用，FPGA也能高效地实现[197]。但目前主流FPGA厂商如Xilinx和Altera提供的产品，其CLB粒度通常为位级，过度的灵活往往带来冗余的资源开销。国内外学术界为应对此问题进行了很多卓有成效的研究（如瑞士EPFL和中科院电子所合作提出的and-inverter锥技术[202]），然而还是难以彻底解决这个问题。

为降低可配置电路的资源开销，有很多研究者倡导利用字级的可配置单元构成粗粒度现场可编程阵列电路CGRA[45]，但如何从高级语言C直接生成CGRA的配置（即高层次综合）一直没有完美的解决方法[38]。Mishra等人提出的Tartan具有独特的异步

电路组成的分布式控制逻辑，可以通过他们提出的编译工具将C语言的一个子集映射到Tartan上[134]。Friedman等人提出了一种针对二维动态连接CGRA结构的可重新指向的编译工具SPR[67]。然而SPR不仅能力有限，只能处理一维循环代码，而且无法支持很多常见的CGRA结构。总而言之，将高级语言映射到CGRA的编译工具链还存在很多问题。

ASIC则是针对专门的、特定算法实现的硬件电路。这种面向特定需求、专注于解决特定问题的解决方案，通常具有体积更小、功耗更低、可靠性更高、性能更高等优点，同时也给予了设计人员更大的灵活性，从而使得ASIC成为很多面向特定领域的加速器设计的首选。近年来，很多研究者开始关注如何针对一类应用设计具有一定灵活性的可编程的ASIC[13,155]。中科院计算所的前期研究表明，面向机器学习类应用的可编程ASIC 可以达到通用核百倍以上的性能，而其面积和功耗远小于通用核[29]。国际上亦有研究者探索自动化的可配置的ASIC生成方法。如Swanson和Taylor提出的GreenDroid加速核[185]，即通过分析Android操作系统库中热函数后自动进行综合生成，具有一定的制造后再修改的能力。

1.1.3 神经网络

神经网络算法大致可以分为两类，生物神经网络（Biography Neural Network）和人工神经网络（Artificial Neural Network）。生物神经网络多由生物学家提出，算法重点在于模拟生物模型，从而更好的理解神经网络甚至大脑的工作机制。其代表性的网络为脉冲神经网络（Spiking Neural Network）[124]，也被称为第三代神经网络。2014年IBM在Nature上发布了TrueNorth[133]加速器，其结构即是脉冲神经网络；同时高通也透露了神经生物芯片Zerorth处理器的相关信息[101]。现代人工神经网络的起源则要追溯到1943年McCulloch和Pitts的工作[131]，其证明了采用其构建的神经元模型而构成的网络在原则上可以计算任何函数。1949年Hebb清楚地阐述了突触修正的生理学模型，而1958年Rosenblatt提出了第一个有监督学习感知器的模型，80年代Hopfield发表了引起巨大争论的论文[84]标志着一波神经网络的高速发展期的开端。近几年来，深度学习（Deep Learning）[106]则开始独领风骚，其相关的深度神经网络算法在许多应用程序上具有出色的表现（如网页搜索[85],图像分析[136]，语音识别[42]等），也引起了学术界和工业界的研究人员的极大热情，带来了神经网络的又一次发展高峰期。深度神经网络（Deep Neural Network，DNN）、多层感知器（Multi-Layer Perception，MLP）以及卷积神经网络（Convolutional Neural Network，CNN）[109]以及则成为研究的热点。

目前，神经网络算法在工业界和产业界得到了广泛的应用。近些年越来越多的云服务应用和移动终端应用成为热点，如网络搜索（谷歌Google，必应Bing，百度Baidu）、音乐或语音识别（如Shazam或Siri）、图像/视频分析（如自动文本标记，

如Picasa或Flickr)、在线导航（如谷歌地图、必应地图或百度地图）等。这些应用不单单对计算能力具有很高的需求，同时也对能效性具有相当高的要求。2005，INTEL综合当前的应用的发展趋势归纳出RMS (Recognition/识别, Mining/发掘, Synthesis/综合)应用[117]，并在之后联合Princeton推出了PARSE基础测试程序集[17]，其中大部分程序为RMS类应用，这些应用多基于分类（Classification)、聚合（Clustering)、近似（Approximation）或优化（Optimization）算法。在2012年Chen[28]等人将一半以上的PARSEC程序中占90%以上执行时间的部分用神经网络算法实现。这样的趋势也被不少体系结构研究人员注意到，进而开展了一系列相关的体系结构研究工作，如生物神经网络加速器（TrueNorth[133]，Zeroth[101]等等）和人工神经网络加速器（DianNao[29], Neuflow[153]等等)。

1.1.4 硬件神经网络：从期望到生产力

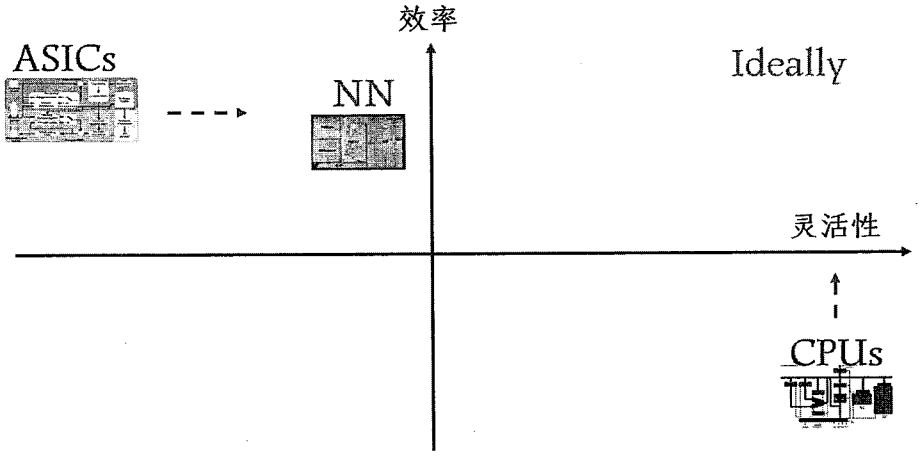

图 1.1 硬件神经网络加速器

得益于神经网络算法研究的进步使得其在精度上有着优异的表现，同时也由于神经网络算法本身具有较广泛的应用空间，加上近些年应用场景所发生的巨大变化，使得神经网络算法得到越来越广泛的应用。神经网络算法本身所具有数据密集和计算密集的特性，这使得采用硬件进行加速则成为再合适不过的选择。正如图 1.1中所示，传统ASIC设计的应用范畴受到限制，其通常不够灵活，但是具有较高的能效性；而通用处理器如CPU则通常足够灵活，能够应对绝大多数应用，但也因此效率低下。在灵活性（也即应用的适用范围）和效率的权衡取舍上，硬件神经网络具有天然的优势。硬件神经网络加速器也促进神经网络算法从单纯的学术研究中走出，减少了算法对于大规模服务器的依赖，从而大规模的应用到现实生活的方方面面。

1.2 主要研究内容及贡献

本文中的研究内容即是硬件神经网络加速器。

图 1.2 硬件神经网络加速器研究路线图

1.2.1 主要研究路线图

正如图 1.2所示，本文中的研究按照该研究路线图进行：（1）在神经网络算法本身适用场景不清晰的情况下，加上硬件设计的成本极高，贸然选择一种神经网络进行加速器设计并不是一个明智的选择，我们研究的第一个问题即是对于在什么场景下应该采用哪种神经网络进行硬件设计；（2）在回答了第一个问题后，我们选择机器学习激励而来的神经网络算法，针对之前加速器本身存在的访存开销过大的问题和嵌入式移动端的应用场景，我们设计实现了更加高效的ShiDiannao神经网络加速器，ShiDiannao最大的特性即是通过将整个网络存储在片上从而避免与内存进行交互，这使得ShiDiannao变得极其高效；（3）对神经网络算法来讲，其本身具有一定的错误容忍能力，而重训练更是可以通过训练算法调整网络本身的参数从而使得精度可以一定程度上恢复到原来的水平，我们也利用这样的特性在硬件神经网络加速器设计中引入非精确计算的概念，从而设计出更加高效的神经网络加速器。

本文中的主要研究内容和主要贡献为以下三个方面：

神经网络加速器比较

从工业机器人、自动驾驶汽车到智能手机，大量的设备需要对从现实世界中捕获的信息（如图像、声音、无线电波等等）进行越来越复杂的处理。值得注意的是，对于这些任务，硬件神经网络又渐渐成为极具吸引力的可选结构。这其中所实现的神经网络算法来自两个截然不同的领域：机器学习领域和神经生物学。这两类神经网络算法也因此具有极其不一样的特性，这也导致了对于这两类神经网络算法哪种更加适合实现成为硬件加速器的疑问。目前为止，几乎没有从硬件角度比较这两类神经网络的研究。在本问题研究中，我们把两类神经网络算法实现成为硬件电路，并完成后端布局布线工作，在此基础上比较这两类方法的硬件相关参数，如能耗，速度，面积开销，精度和功能性。

在本问题的研究中，对于目前的神经生物学所启发的神经网络和机器学习启发的神经网络，我们尽可能的实现最优的硬件加速器结构，并通过同样的应用测试从而进行比较，我们的研究结果有助于澄清一个观念：神经生物学启发的神经网络（如SNN+STDP）是和机器学习启发的神经网络（如MLP+BP）在精度和可实现的硬件开销上相媲美的算法。另外，我们也发现，对于非常大规模的神经网络和对于精度要求不严格的应用，SNN+STDP相较于MLP+BP具有更小的

硬件开销，也即是更具吸引力的选择。对于SNN+STDP在精度上劣势，我们也发现其原因并不单纯，相对于脉冲编码的信息丢失，更多的是源于STDP训练算法本身。最后，我们也发现对于需要永久在线训练且对精度要求不高的应用类别，SNN+STDP硬件加速器更加高效。相关内容参见章节 3。

卷积神经网络加速器

近年来，神经网络加速器的研究已经表明：对识别和数据挖掘类的广泛应用神经网络加速器既能够达到高能效同时也能够达到高性能。

尽管如此，此类加速器的能效和性能仍然受到内存访问的限制。在本问题研究中，我们专门针对于可以说是识别中最重要的一类应用，也就是图像应用和数据挖掘类应用。神经网络类算法中对于此类应用目前处在领先地位的算法是*卷积神经网络*(Convolutional Neural Networks, CNN)。卷积神经网络具有一个重要的性质：突触连接的权值被多个神经元共享，这大大的减少了神经网络的权值数量，从而大大的降低了卷积神经网络的内存占用。这点使得片上SRAM能够容纳下整个CNN，从而消除掉所有对权重的SRAM访问。进一步，将此类加速器放置在图像传感器的旁边，也即能够消除剩余的所有的DRAM访问，即输入和输出。

在本问题研究中，我们提出了这样一个CNN加速器，将其放置在一个CMOS或CCD传感器旁边。消除掉DRAM的访存，加上精心设计针对神经网络内部的数据访问模式，我们设计完成了一个高度节能的加速器ShiDianNao。在65nm工艺下，我们完成单核设计实现至后端，然后进行评估，得到该加速器面积为5.94 mm^2，功耗为336 mW，比高端GPU快约30倍。对于实时处理具有更高的分辨率和帧率的视频流，我们提出了多核ShiDianNao（ShiDianNao+）来提升性能。相关内容参见章节 4。

非精确神经网络加速器

近些年来，对于许多能够容忍一定程度非精确度的应用程序来说，**非精确计算（*inexact computing*）**越来越引人瞩目，被认为是降低能耗最有效的手段之一。通过牺牲可接受的精度来换取明显的资源节约（能耗，关键通路的延迟，硅面积等等）的方法和原则被限于使用在专用集成电路（ASIC）上。在目前的设计中，这些实现后的ASIC的应用范围狭窄且对不精确度的耐受性往往是固定的，而后者基本决定了我们能够实现的资源节约量。在本问题的研究中，我们提出将非精确计算与硬件神经网络相结合设计非精确神经网络网络加速器，从而扩大应用范围，提高错误恢复能力和资源节约规模。这些神经网络正迅速成为未来异构多核平台的热门备选加速器，并且能够被重训练的特性使得它们具有灵

活的错误恢复限制。我们在65纳米工艺下的实现结果表明，相比于已存在的硬件神经网络，非精确神经网络加速器可以实现42.82%-62.55%的能源消耗节约（延时和面积分别节省18.70%和31.51%），同时精度损失的代价则很小（均方误差MSE平均从0.14增加到0.20）。相关内容参见章节 5。

1.3 论文组织结构

本论文总共分为六个章节，并按照以下顺序组织。

在章节 2中介绍神经网络相关内容，硬件神经网络加速器相关问题的研究背景和相关工作。

在章节 3中介绍关于硬件神经网络比较的相关研究，比较神经生物学激励的神经网络算法和机器学习激励的神经网络算法的硬件相关问题。

在章节 4中介绍ShiDiannao加速器的相关研究，包括设计原则、结构设计、算法映射和性能评估，并且在本章节中介绍ShiDianNao的多核扩展设计以提高性能。

在章节 5中介绍非精确计算的实现方法和非精确硬件神经网络加速器的设计方法和性能评估。

在章节 6中总结本文中硬件神经网络的研究工作，并给出未来可能的研究方向和可能的技术突破点。

第二章　神经网络简介

2.1　神经网络

神经网络算法大致可以分为两类，生物神经网络（Biography Neural Network）和人工神经网络（Artificial Neural Network）。生物神经网络多由生物学家提出，算法重点在于模拟生物模型，从而更好的理解神经网络甚至大脑的工作机制。其代表性的网络为脉冲神经网络（Spiking Neural Network）[124]，也被称为第三代神经网络。现代人工神经网络的起源则要追溯到1943年McCulloch和Pitts的工作[131]。人工神经网络多注重模式分类或者目标识别类任务的应用，并不以模拟大脑为己任，而这其中最广为人知的则是多层感知器（Multi-Layer Perception，MLP）和深度神经网络（Deep Neural Network，DNN）。近几年深度神经网络算法独领风骚，在许多应用程序上具有出色的表现（如网页搜索[85],图像分析[136]，语音识别[42]等）。

2.1.1　机器学习激励的神经网络算法

相较于神经生物学模型，机器学习领域对于神经元模型的抽象层次更高，如图2.1中所示的一个抽象模型，神经元的轴突（axon）和树突（dendrite）之间连接已经被抽象成为权值连接，而之间的信息传递也是通过权值来表示。值得注意的是，神经元之间的连接不存在时间概念，也即神经元在同一时刻会集合所有来自外部的输入（输入和权值相乘，然后将所有的乘法结果相加），而其输出则通过线形函数或者非线性函数传递给后续神经元。

机器学习领域中使用了各种类型的人工神经网络。其中应用最广泛的一种神经网络类型是多层感知器（Multi-layer Preceptron, MLP）[79]；卷积神经网络（Convolutional Neural Network, CNN）[109]也非常流行，并且最近出现的另一种类型的人工神经网络被称为深度信念网络（Deep Belief Network, DBN），它已经超越支持向量机（Support Vector Machine,SVM）和其它的技术成为目前最先进的机器学习算法[99,106]。简单地说，DBN和MLP、CNN结构相似，区别只是DBN有更大更多的层。我们在这里主要介绍本文中所涉及的MLP和CNN。

2.1.1.1　MLP

典型的MLP是一个两层神经网络：一个输出层和一个隐含层，加上一个不含神经元的输入层，见图2.2。MLP是一种前馈神经网络，信息从输入层（$l = 0$）传播到输出层（$l = 2$）。输入层不包含神经元，并且输入通常为$n - bit$的值（如灰度图像中像素亮度，其值为8-bit）。输出层中的每个神经元连接至隐层中所有神经元，隐层中每个神经

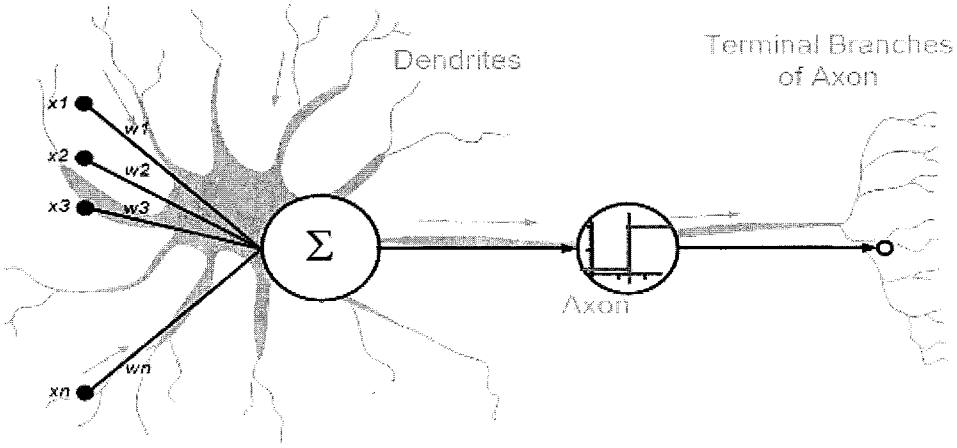

图 2.1　机器学习对生物神经元抽象的神经元模型

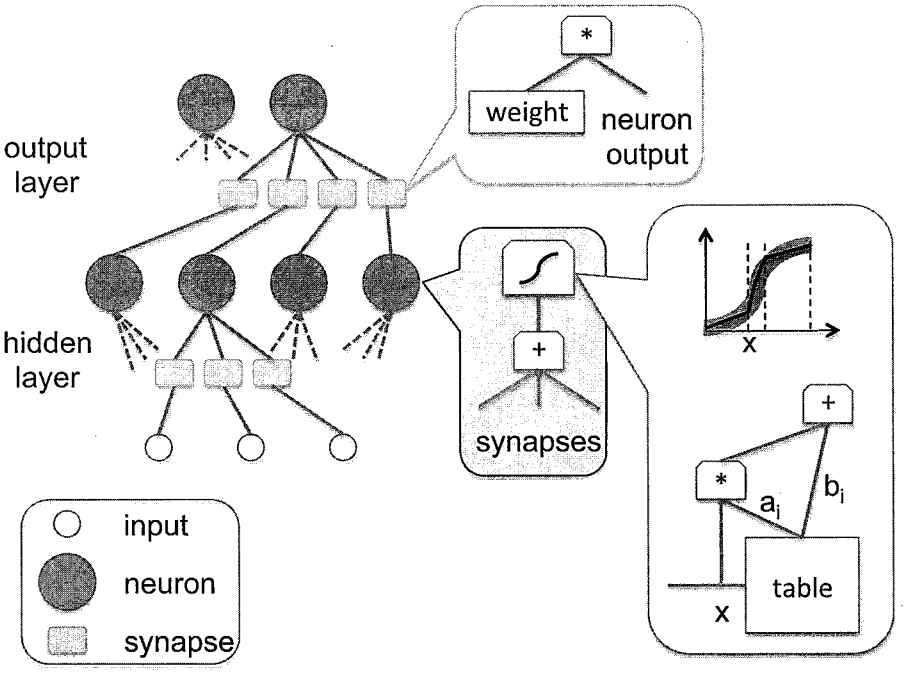

图 2.2　2层MLP的结构和相应逻辑运算符

元连接到所有输入。每个神经元与前一层的所有神经元相连（或是输入），l层的神经元j与$l-1$层的神经元i之间的连接带有突触权值w_{ji}。这个突触权值与神经元i的输出相乘，所有这一层的乘积在神经元j中相加，并且将所得的和值输入至一个“激活函数”f（通常为sigmoid函数$f(x)=\frac{1}{1+exp^{-x}}$）。

神经元和突触　层l中的一个神经元j完成计算：$y_j^l = f\left(s_j^l\right)$，其中$s_j^l = \sum_{i=0}^{N_{l-1}} w_{ji}^l y_i^{l-1}$，$w_{ji}$是层$l-1$中的神经元$i$和层$l$中的神经元$j$突触连接权值，$N_l$是层$l$中的神经元数目，$f$是激活函数。

激活函数　激活函数实现了人工神经元中的非线性行为。在硬件中，激活函数的最高效的实现方式是作为一个分段线性函数的近似，这需要一个查找表

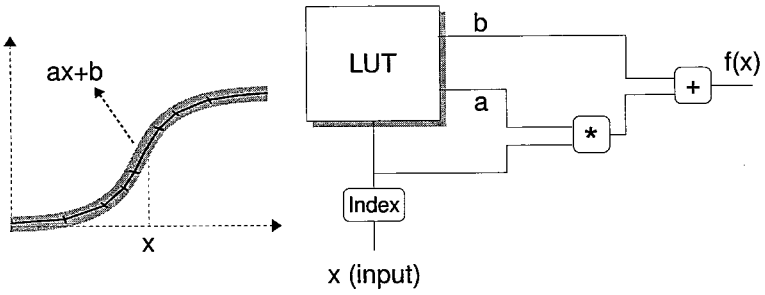

图 2.3 Sigmoid函数和线性分段拟合实现

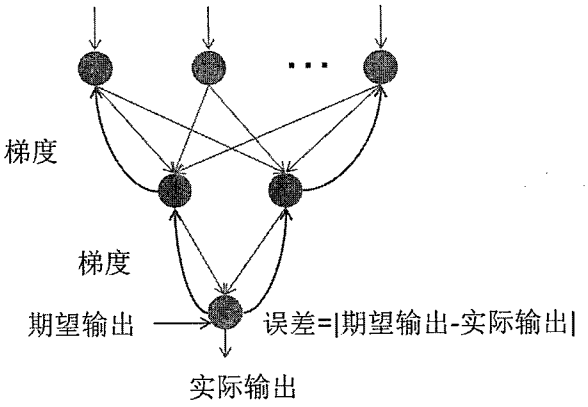

图 2.4 反向传播算法

$(x \rightarrow f(x) = a_i \times x + b_i)$,表格的每个条目包含一对系数（$a_i, b_i$），见图 2.3中sigmoid函数的分段线性近似实现。每一段i代表了输入值x的一个有限的范围。根据实验，我们发现采用16段线性函数近似激活函数就已经足够，相比于原来的sigmoid函数本身对精度的影响微乎其微，而且神经网络的精度并没有受到明显的影响。

前向通路 我们使用最流行的训练算法也就是反向传播（Back-Propagation）算法[79]，但是在硬件中并不实现这个算法：硬件神经网络仅仅包含前向通路。一个时常出现的误解是认为在线学习（*on-line* learning）对许多应用是必要的。然而对许多工业应用来说离线学习（*off-line* learning）就已经完全足够，神经网络可以在数据集上训练完成后再移送给顾客，例如训练网络识别手写数字、汽车牌照、人脸等其它物体。神经网络是可以在离线情况下反复地训练，在使用时则不需要永久的学习。例如用来作金融预测的神经网络就是以一个前向通路的模型运作的，比如将股票特征作为输入，股票价格预测作为输出，这样的网络可以每周或每天反复训练来学习到目前为止的未知的股票模式，但当做预测的时候，不需要在线去学习数据。这里需要将商用的硬件神经网络和仿生的神经网络区分开来，后者依赖持续不断的在线学习，例如脉冲时间相关可塑性（Spike timing dependent plasticity，STDP）[145]。

学习（反向传播） 反向传播是一种有监督学习方法，这里首先将输入提交

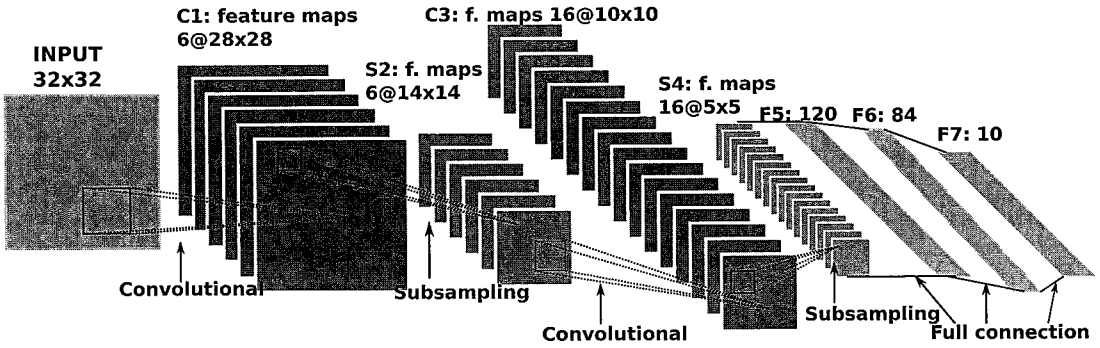

图 2.5 代表性的CNN结构：LeNet5 [109]。**C**：卷积层；**S**：采样层；**F**：分类层

给网络通过前向通路产生输出，然后将该输出和已知的输出之间的误差则通过网络反向传播回输入层从而更新所有的权重。反向传播的方法是迭代进行的：此过程重复多次，直到精度目标达成或者分配的学习时间已经过去。权值更新如下$w_{ji}^{l}(t+1) = w_{ji}^{l}(t) + \eta\delta_{j}^{l}(t)y_{i}^{l-1}(t)$，其中$t$是迭代索引，$\eta$是学习率，$\delta_{j}^{l}$是误差的梯度也就是错误的方向。在输出层，梯度通过以下公式进行计算：$\delta_{j}^{l}(t) = f'(s_{j}^{l}(t)) \times e_{j}^{l}(t)$，其中$e_{j}^{l}$是误差（神经网络的输出和期望输出之间的误差）；在隐层，梯度通过以下公式进行计算：$\delta_{j}^{l}(t) = f'(s_{j}^{l}(t)) \times \sum_{k=0}^{N_{l+1}} \delta_{k}^{l+1}(t)w_{kj}(t)$，其中$f'$是$f$的导数。

2.1.1.2 CNN

卷积神经网络（Convolutional Neural Networks，CNNs）[109]和深度神经网络（Deep Neural Networks，DNNs）[106]被认为是目前最先进的机器学习算法。这两个神经网络都属于多层感知神经网络（Multi-Layer Perceptrons，MLPs）[79]）的范畴，共同的特点是都包含4种不同的层：卷积层，采样层，归一化层和分类层。其中卷积层的执行过程中占据整个神经网络执行时间的绝大部分，然而这两种神经网络在卷积层上具有较大的区别。卷积神经网络中，突触权值可以被不同集合的神经元复用，但是在深度网络中没有这样的复用（参见后文中详述）。CNN中的数据复用的特点为硬件实现创造了有利条件，这是因为权值的复用降低了突触权值的存储开销，也使得将所有的权值同时存储在片上变得可能。

通用结构 图 2.5中展示了在文字识别方面应用广泛且具有代表性的CNN结构：LeNet-5 [109]。此网络包含两个卷积层（图 2.5中C1和C3），两个采样层（图 2.5中S2和S4）和3个分类层（图 2.5中F5、F6和F7）。此外，最近的研究建议在深度学习中增加使用归一化层 [89,100]。

卷积层 卷积层可以被认为是在输入特征图像（二维数据，来自输入像素或者前一层神经元）上检测特定特征的一组局部滤波器。每个局部滤波器有含$K_x \times K_y$个系数的卷积核（*kernel*），处理一个输入特征图上包含有$K_x \times K_y$个输入神经元的卷积窗口（或者是多个的输入特征图上多个同样的大小的卷积窗口）。一组二维的局部滤波器能

够产生一个输出特征图像，其中每一个滤波器对应一个卷积窗口得到一个输出神经元，而相邻的输出神经元的卷积窗口在同一输入特征图像上滑动步长为S_x（x-方向）和S_y（y-方向）。也即，在特征图像#mo上位置为(a,b)的输出神经元通过以下公式计算

$$O_{a,b}^{mo}=f\left(\sum_{mi\in A_{mo}}\left(\beta^{mi,mo}+\sum_{i=0}^{K_x-1}\sum_{j=0}^{K_y-1}\omega_{i,j}^{mi,mo}\times I_{aS_x+i,bS_y+j}^{mi}\right)\right), \tag{2-1}$$

其中$\omega^{mi,mo}$是输入特征图像#mi和输出特征图像#mo之间的卷积核$kernel$，$\beta^{mi,mo}$是该组输入输出特征图像的偏置（bias），A_{mo}所有连接到输出特征图像#mo上的所有输入特征图像，$f(\cdot)$是非线性激活函数（如$tanh$函数和$sigmoid$函数）。

采样层 采样层通常直接在输入特征图像上进行降采样，采样通过在不重叠的窗口中（也即，采样窗口，每个窗口内包含$K_x\times K_y$个输入神经元）选择具有神经元的最大值或者平均值。也即，在特征图像#mo上位置为(a,b)的输出神经元通过以下公式计算

$$O_{a,b}^{mo}=\max_{0\le i<Kx,0\le j<Ky}\left(I_{a+i,b+j}^{mi}\right), \tag{2-2}$$

其中$mo=mi$，这是因为输入输出特征图像是一一映射的。上述公式中所用的采样方法为选择具有最大值的神经元，而平均所有神经元的采样公式基本上与上述公式相同只是在采样窗口的的最大数选择操作（max）应被替换为平均数计算操作（avg）。通常，CNNs也会将采样结果输入至非线性的激活函数，然而最近的研究结果不再推荐使用这样的操作 [89,110]。

归一化层 归一化层在不同输入特征图像上相同位置的神经元之间引入竞争，这样能够进一步提升CNNs的识别精度。目前存在两种不同的归一化方法：*局部对比归一化方法*（Local Contrast Normalization，LCN [89]）和*局部响应归一化方法*(Local Response Normalization，LRN [100])。

在局部响应归一化层（LRN）中，在特征输出图像#mi上位置为(a,b)的神经元可以通过以下公式计算

$$O_{a,b}^{mi}=I_{a,b}^{mi}/\left(k+\alpha\times\sum_{j=\max(0,mi-M/2)}^{\min(Mi-1,mi+M/2)}(I_{a,b}^{j})^2\right)^{\beta}, \tag{2-3}$$

其中Mi是所有输入特征图像的数目，M是连接到一幅输出特征图像上最大的输入特征图像数目，α，β和k则是常数参数。

在局部对比归一化层（LCN）中，在特征输出图像#mi位置为(a,b)的神经元可以通过以下公式计算

$$O_{a,b}^{mi}=v_{a,b}^{mi}/\max\left(\mathrm{mean}(\delta_{a,b}),\delta_{a,b}\right), \tag{2-4}$$

其中$\delta_{a,b}$通过下面的公式计算

$$\delta_{a,b} = \sqrt{\sum_{mi,a,b} (v^{mi}_{a+p,b+q})^2}, \tag{2-5}$$

其中$v^{mi}_{a,b}$（减法归一化）通过下面公式计算

$$v^{mi}_{a,b} = \mathrm{I}^{mi}_{a,b} - \sum_{j,a,b} \omega_{a,b} \times \mathrm{I}^{j}_{a+p,b+q}, \tag{2-6}$$

其中$\omega_{a,b}$是一个正则化的高斯函数窗口，满足$\sum_{a,b} \omega_{a,b} = 1$，$\mathrm{I}^{mi}_{a,b}$是输入特征图像#*mi*在位置$(a,b)$的输入神经元。

分类层 串行地经过几层的计算后，CNNs通常在最后连接一个或者多个分类层来计算最后的分类结果。在传统的分类层中，输出神经元通过不同的权值全连接至所有的输入神经元。写成公式，输出神经元#*no*通过以下公式计算

$$\mathrm{O}^{no} = f\left(\beta^{no} + \sum_{ni} \omega^{ni,no} \times \mathrm{I}^{ni}\right), \tag{2-7}$$

其中$\omega^{ni,no}$是输入神经元#*ni*和输出神经元#*no*之前的权值，β^{no}是输出神经元#*no*的偏置，$f(\cdot)$是激活函数。

识别 vs. 训练 关于神经网络的一个常见的误解是，它们必须是在线训练以实现较高的识别率。事实上，对于视觉识别，离线训练（明确的拆分训练和识别阶段）被证明已经足够使用，这一事实也获得了机器学习研究人员的广泛认可 [29,63]。对于低开销的嵌入式传感器来说，由服务供应商进行离线训练是必不可少的，这是因为嵌入式设备只有有限的计算能力和功耗预算。我们这里自然再设计加速器上关注CNNs网络的识别阶段。

2.1.2 神经生物学激励的神经网络算法

生物神经网络的模型多由生物学家提出，很多算法重点在于模拟生物现象，从而更好的理解生物神经网络（如大脑）的工作机制，所以生物神经网络中的神经元模型抽象层次更接近真实的生物学所观察到的生物现象（生物神经元模型参见图 2.6），其信息传递是通过脉冲放电的形式，与生物学中的观察一致。脉冲放电是生物神经网络里最重要的概念之一，涉及到如何表示和传递信息。单个输入刺激所携带的信息编码成为一连串的脉冲序列传递到后续神经元，后续神经元通过解码这样的序列获得相应的信息。这样的编码方式通常分为两类，一类是通过脉冲的放电频率编码信息，即Rate Coding，这种工作方式已经被研究人员使用多年；另一类是强调单个脉冲放电的精确时间，通过脉冲的放电时刻和接收时刻来编码信息，即Temporal Coding，它的困难性

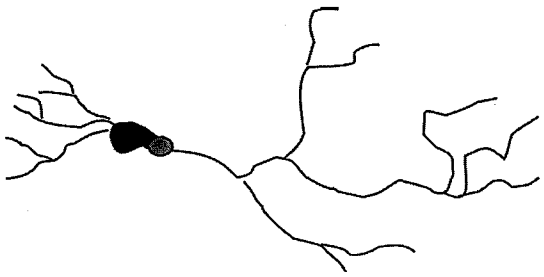

图 2.6 生物神经元

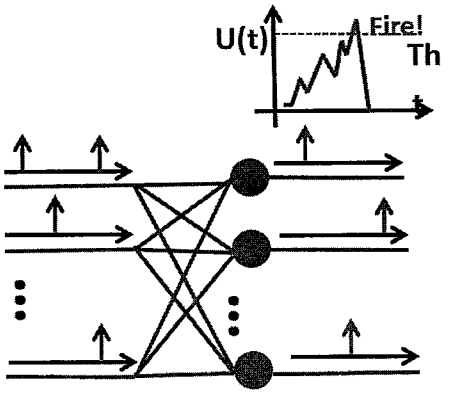

图 2.7 SNN网络拓扑结构

也是显而易见的，Gerstner[73]和Thorpe[188]在近些年也提出了一些相关的方法。关于究竟哪种编码方式更优的问题在学术界一直争论不断，也并没有定论。这一方面是因为我们尚不完全了解大脑的工作机制，另一方面是因为两者都不够理想，没有哪一种具有压倒性的优势。多个神经元输入组合的编码方式一般有Population Coding和Sparse Coding。生物神经网络的代表性网络为脉冲神经网络（Spiking Neural Network，SNN），也即本文中所设计的网络，我们在此进行介绍。

2.1.2.1 SNN

脉冲神经网络（Spiking Neural Network， SNN）是神经生物学启发而来的神经网络，该网络通常包含一层或者多层的脉冲神经元，层与层之间的神经元多为全连接，有的层内神经元之间存在连接，然而和机器学习类的神经网络不同的是信息的载体更多的是脉冲而不是权值大小。

拓扑 如图 2.7中所示结构，SNN的输入为脉冲，输入脉冲会发放到所有的脉冲神经元，单个脉冲神经根据输入脉冲和自身状况吸收脉冲。当其集成的电位超过阈值的时候就会向后续所有连接的神经元发放一个脉冲，同时其自身的电位回落。SNN层不仅通过可激活的突触连接到它的所有输入，其层内所有的神经元之间也存在互相抑制性的连接。本文中，我们考虑一个单层SNN。尽管这种选择相较于2层的MLP可能是不公平的，然而使用这样的SNNs在MNIST获得了目前SNNs中最佳的效果 [158]。事实上，横向抑制连接产生所谓WTA（赢家通吃，Winner-Takes-ALL）的动态连接则在事实上创建了某种形式上的recurrent网络：同层之间的神经元存在相互作用关系。这种类型的连接有效确保了紧凑性和理论上对输入数据的非线性映射，从而获得和MLP相比有竞争力的识别精度 [154]。

神经元和突触 神经元的输入和输出都是脉冲（Spikes），如图 3.4所示。每个神经元的输出可以根据固定的时间间隔的脉冲重新解读为脉冲串和激发频率 [13]。然而，

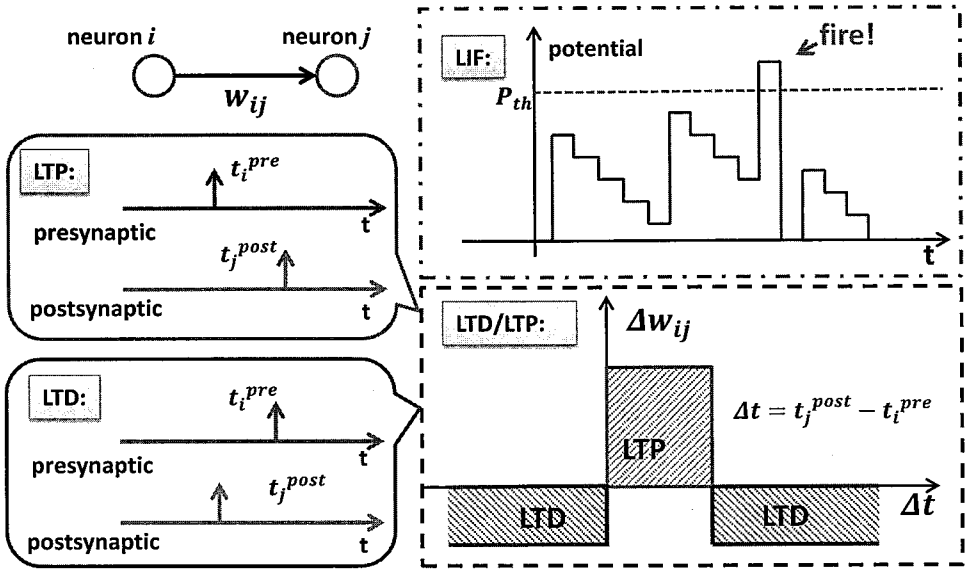

图 2.8 长时程增强（Long-Term Potentiation， LTP）和长时程抑制（Long-Term Depression，LTD）

更有效的读出方法是考虑基于WTA首个激发出脉冲的神经元。这种方法高效且快速，有利于高效的硬件实现，并且，目前为止在SNN上展现了一定的机器学习最擅长的能力 [130]。

本文中我们采用标准的LIF神经元模型（Leaky Integrated and Fire） [21]，其中神经元j的电位v_j通过计算下面微分方程的解得到：

$$\dot{v}_j(t) + \frac{v_j(t)}{T_{leak}} = \sum_{i=1}^{n} w_{ji} \times I_i(t)$$

该公式中，T_{leak}是泄露时间常数，I_i是输入i的值。通常情况下，该微分方程的解可以通过离散模拟（多个时间步长）获得，转化在硬件中就是在较短的时间间隔内进行重复计算。这样的方法既在时间上低效也在能耗方面比较低效。已知神经元的电位$v_j(t)$是两个输入脉冲间的微分方程的解是：$\dot{v}_j(t) + \frac{v_j(t)}{T_{leak}} = 0$ 不同于前面的微分方程，这个方程可以得到解析解。从而得到在两个连续的输入脉冲之间（分别在T_1和T_2时刻）神经元的电位可以通过以下方程得到：$v_j(T_2) = v_j(T_1) \times e^{-\frac{T_2 - T_1}{T_{leak}}}$. 这个方程可以进行更高效的硬件实现。

最后，注意当一个神经元激发后，它会抑制其他所有的神经元导致其他所有的神经元进入抑制期（inhibition period），从而模拟抑制性突触连接的存在；同时激发的神经元本身会进入一个固定时长的耐火期（refractory period），在耐火期，该神经元感知输入脉冲。此机制和以前其他高效的SNN模型 [130]一致。

学习（STDP） STDP的学习原则是非常不同于BP的：学习过程本身是非监督的，每个神经元本身会逐步向其本身接收到的最显著的信息模式收敛。学习过程的基

本准则是检测输入脉冲和输出脉冲间的因果关系：如果一个神经元在一个固定突触连接上接收到输入脉冲后很短的时间内就激发，则认为该突触连接在神经元激发过程中的作用极大，则这个突触连接应该被增强，这个概念被称为长时程增强（Long-Term Potentiation， LTP），参见图 2.8；相反的，如果一个神经元在一个固定突触连接上接收到输入脉冲后很长时间才激发或者在接收到脉冲前激发，则认为该突触连接在神经元的激发过程中没有影响的可能性很大，则这个突触连接应该被削弱，这个概念被成为长时程抑制（Long-Term Depression，LTD），参见图 2.8。注意，STDP只在输入到输出之间的可激活的突触上存在，而不在SNN层中神经元间的抑制性的连接上存在。

自稳态 为了平衡神经元之间的信息，调整其激发的阈值变得很关键：激发过于频繁的神经元受到抑制（增加它们的激发阈值）；激发过少的神经元则得到促进增强（降低其激发阈值）。这样的动态调整过程存在于生物神经元中，称为稳态过程 [127]。本章节中，我们也采用了这样的机制，保证了所有输出神经元都可以在学习过程中学习到不同的信息模式，我们观察到这使得SNN提高了约5%的精度。最后，对于任意一幅输入图像，只有一个神经元能够激发，这使得结果读出变得简单迅速。

和激发事件相比，自稳态过程在一个较大的时间尺度上发挥作用。这里我们定义一个时间长度*自稳态回合*，并在每个自稳态回合结束时我们更新所有神经元的激发阈值。更新过程是根据它已经发射了大于（或者小于）预先设定的自稳态的阈值，增加（或者减少）的神经元的阈值。阈值调整根据下面的表达式：

$$firing\_threshold+ = sign(activity - homeostasis\_threshold) * firing\_threshold * r$$

其中*activity*是自稳态回合期间的神经元已经激发的次数，r是正值的乘法常数 [157]。需要注意的是，整个过程是每个神经元完全在本地进行的操作（因此硬件层面的布线开销将是极小的）。

标记 因为STDP是一种无监督学习方法，有必要存在一个对应的互补步骤去根据相应的输出信息标记神经元。我们用如下所示的自我标记方法：我们用标签是已知的训练图像来标记神经元。每个神经元有和可能存在的标签一样多的计数器；当一个输出神经元激发，则根据输入图像给定的标签将该神经元中对应的标签计数器加一。当处理完所有的训练图像，每个神经元则被标记为具有最高得分的标签（分数从标签计数器的值推导出，即标签计数器结果除以与标签一致的输入图像的数量;这种方法考虑了输入图像中标签数目的可能差异）。

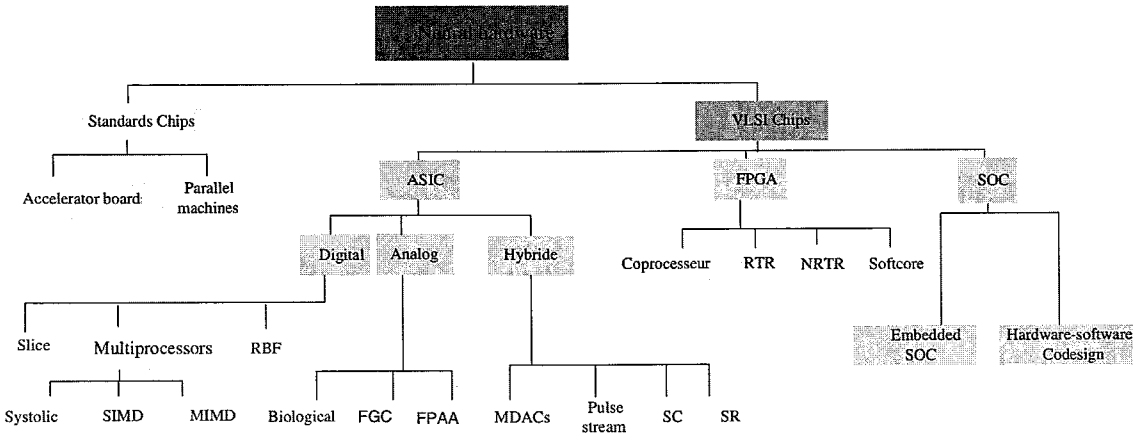

图 2.9　硬件神经网络的分类方法[88]

2.2　硬件神经网络加速器的研究现状

从神经网络提出以来，硬件发展也一直跟随着算法的更新而不断进步，这部分我们主要梳理下硬件神经网络加速的研究现状。

2.2.1　硬件神经网络加速器的分类标准

一方面由于神经网络算法和模型的种类在不断的增加，另一方面由于硬件技术也在不断的发展而出现了多种多样的加速平台，这使得硬件神经网络加速器的分类变得越来越困难，难以通过单纯的标准对硬件神经网络加速器进行具有实际意义的划分。研究人员一直未曾停止将硬件加速进行合理且有意义的划分。在1972年，Flynn等人 [66]提出了SISD（单指令流单数据流，Single Instruction Single Data），SIMD（单指令流多数据流，Single Instruction Multiple Data），MISD（多指令流单数据流，Multiple Instruction Single Data）和MIMD（多指令流多数据流，Multiple Instruction Multiple Data）的划分方法；1995年Paolo Ienne等人 [87]提出了基于灵活性和性能的串行并行分类方法；1996年Isik Aybay等人 [11]提出了多属性多维度的分类方法（片上/片外，数字/模拟/混合等等）；2014年Izeboudjen等人 [88]提出了偏向体系结构的分类标准；2004年Dias等人 [47]总结了商业上的硬件神经网络加速器；2010年Misra等人 [135]则总结了从1990年到现在这20年间的硬件神经网络加速器。在这里，我们将跟随Izeboudjen等人的分类标准来介绍现在的硬件神经网络加速器的发展现状，该分类方法参见图 2.9。注意，我们区分了机器学习启发而来和生物神经学启发而来的神经网络模型。

2.2.2　机器学习激励的硬件神经网络加速器

近些年来，尤其是深度学习的提出，神经网络的规模越来越大，模型中的参数也越来越多，例如较大的CNN有6千万权重 [100]，而最大的DNNs中则有10亿权重 [108]甚至100亿权重 [37]。现在机器学习激励的硬件加速器多集中在常用的网络，也

即MLP/CNN/DNN，这几种网络都是分层运算，具有良好的可并行性，而且这几种网络往往也具有良好的精度表现。根据我们在章节 2.2.1中所介绍的分类标准和章节中介绍的常用加速结构，我们主要介绍通用处理器（CPU）、图形计算加速器（GPU）、现场可编程门阵列（FPGA）和专用电路（ASIC）上的对神经网络的硬件加速。

2.2.2.1 CPU

CPU作为通用处理器，本身不用做任何改变就可以完成神经网络算法的计算，然而由于通常CPU的并行度低，本身的计算能力也有限。现在常用的方式是进行分布式计算，通过集合多个CPU从而提升计算的并行度。

Google在2012年提出了基于分布式机群运算的深度学习模型 [69,108,191]，该模型使用非常大的图像数据集（一千万幅图像，每幅图像大小为200x200），本身具有约10亿个权值参数。该分布式机群有1000台机器，每台机器具有16个核（总计16000个核），在ImageNet上识别2200个物体的精度经过3天训练相对提升了70%，达到了15.8%。

2.2.2.2 GPU

GPU具有极高的并行度，本身的发展和相应的编程平台的成熟，使得使用GPU做加速平台的工作变得相对容易，也促使这样的工作越来越多，不单单是神经网络，也包括更多其他方面的应用。

Oh等人 [147]在2004年提出了将向量内积运算编排成矩阵操作，从而有效的利用GPU加速神经网络计算。Coates等人 [36]在2009年的研究表明，GPU加速器可以达到90倍以上的加速比。近些年来，支持GPU加速的深度学习软件框架大量出现，这一方面是因为GPU本身巨大的加速能力，另一方面也是因为GPU的易用性。这些框架中比较流行的有Caffe[90]，Theano[12,14]，Torch[5]，cuda-convnet[1]，CXXNET[3]。Nvidia也应景地推出了自己的深度学习库cuDNN～[2]，最大可加速30%。

2.2.2.3 FPGA

FPGA，相较于GPU而言，是更加偏向底层硬件的加速平台，多用于硬件功能快速验证和性能评估，很多进行自定制电路ASIC设计的工作都需要在FPGA上进行流片前的验证。然而，也正是由于FPGA本身能通过可编程逻辑单元满足可定制的需求，使得其相较于ASIC仍然不够高效。

尽管这样，仍然有很多研究开发工作是基于FPGA的，例如[44,64,95,98,148,181,199]等等。2002年Yun等人 [201]提出ERNA结构（基于传统SIMD机构），并在FPGA上验证了具有256个输入神经元、32个隐层神经元和5个输出神经元的2层MLP和反向传播算法（Back Propagation）。Farabet等人[65]在2009年提出了卷积神经网络处理器CNP（Convolutional Network Processor），并在FPGA上实现，基于CNP的人脸检测系统处理512×384视频流的速度可达10帧每秒。2011年Farabet等人 [63]提出了一种运

行时可重构数据流处理器NeuFlow结构，并于2012年[153]在Xilinx Virtex 6 FPGA上实现。他们的实验表明，NeuFlow相较于CPU的加速比在100倍以上，功耗约为10W。2009年Sankaradas等人 [166]也在FPGA上实现了用于CNN算法的加速器，其中卷积运算作为卷积神经网络中运算量最大的部分成为了研究热点，专门实现了多个卷积核运算单元和非线性函数，同时使用了大的数据带宽保证运算单元的数据供应，然而却不能灵活的支持不同大小的卷积核。2010年Chakradhar等人 [25]提出了类似systolic的结构实现了卷积神经网络CNN的协处理器来加速处理VGA视频流（640×480，25-30的帧率），在FPGA上实现频率达到200MHz。2013年Peemen等人 [152]则利用CNN本身的计算访存特性，实现了以存储为中心的CNN卷积神经网络协处理器，其中存储采用传统的scratchpad memory，计算单元PE采用SIMD结构，通过MicroBlaze主控芯片控制FPGA上的操作，实验表明该结构减少了13x的资源消耗。

2.2.2.4 ASIC

ASIC作为自定制电路，本身完全依赖于设计人员的实现，也是所有加速器平台中赋予设计者最大自由的平台，从而也最有可能针对单独的特定应用实现最高效的电路，然而一旦设计完成，电路本身只适用于特定的应用，反而是所有加速器平台中最不具有灵活的。传统的ASIC受制于其有限的应用范围而不具有良好的扩展性，而最近的研究表明硬件神经网络算法能够具有较广泛的应用范围，这样推动了硬件神经网络ASIC加速器的进一步发展，凸显出ASIC本身在性能和功耗上的巨大优势。

NeuFlow的研究人员也在2012年 [153]将该结构在IBM 45nm SOI工艺下实现ASIC电路设计，结果表明NeuFlow的ASIC电路达到了490 GOPs/W的性能功耗比（FPGA上为14.7 GOPs/W，GPU上为1.8 GOPs/W），这样的结果也说明ASIC的巨大优势和潜力。

ASIC电路本身包含数字电路、模拟电路和数模混合电路。Intel早在1989年就提出了名为ETANN的模拟芯片，该芯片包含64个全连接的神经元和10240个权值连接，研究人员也尝试集合12个ETANN芯片来实时处理图像（Mod2 Neurocomputer[138]）。Liu等人 [123]则在2002年提出了采用Orbit 2um的N-well的CMOS芯片，其包含前向通路和实时错误处理模块。除此之外，早期还有大量工作也是采用了模拟电路来实现。而[115,129,169]等工作则是采用了数字和模拟混合电路来实现硬件神经网络，其中，或是通过模拟电路完成神经网络计算，或是直接接收外界输入的模拟信号。然而模拟电路一方面精度较难控制，另一方面通过电压或者电流表示的数值的值域有限，只能表示有限的范围。除此之外，模拟电路的实现也依赖设计人员本身。相较而言，数字电路具有完整可靠的流程，且本身具有良好的鲁棒性，所以目前大多数芯片都是数字芯片，尤其是计算需要高精度的领域，如科学计算。Micro Devices提出的MD1220是第一款真正商用的数字芯片 [39]。2010年Esmaeilzadeh等人提出了NnSP [59]，一个采用了近似计算的硬件MLP神经网络处理器。Temam则在2012提出了具有容错能力的硬件MLP神经

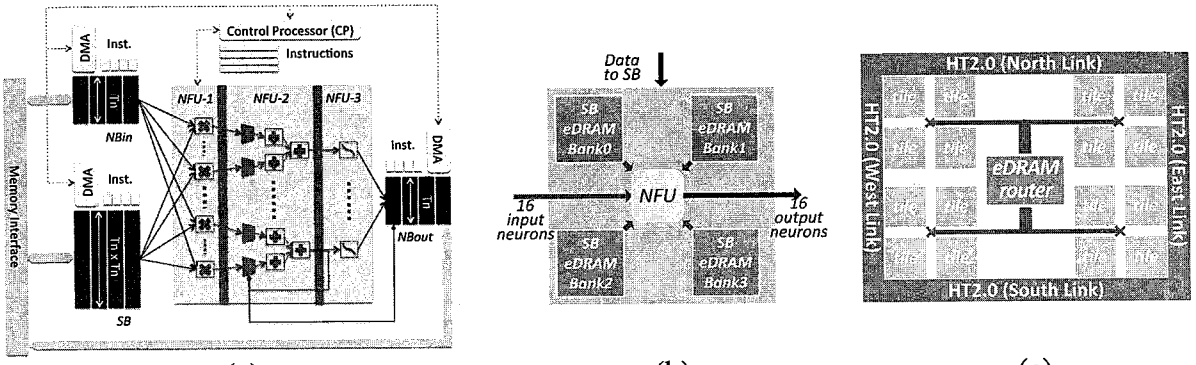

图 2.10 （a）DianNao：NFU结构[29]（b）DaDianNao：一个tile的结构[33]（c）DaDianNao：一个node的结构（包含16个tile）[33]

网络芯片 [187]。Zidong等人 [51]则在2014年研究了采用非精确计算的硬件MLP神经网络加速器。最近，Chen等人提出了DianNao系列神经网络加速器 [29,33,122]，该系列被认为是目前最优秀的神经网络加速器芯片，能够支持MLP/CNN/DNN等类型的神经网络算法。在这里，我们对DianNao进行简单的剖析介绍。

如图2.10所示的DianNao结构，DianNao中包含一个计算单元模块（Neural Functional Unit，NFU），一个指令控制器（Control Processor，CP），一个权值存储器（Synapse Buffer， SB），一个输入神经元存储器（Neuron Buffer for inputs，NBin），一个输出神经元存储器（Neuron Buffer for outputs，NBout）和外部存储接口（Memory Interface）。DianNao采用一个简化的控制处理器用来读取自定义的SIMD指令，从而控制所有的模块进行相应的操作。数据通过DMA流入流出处理单元。计算模块NFU共有Tn个单元，每个单元在同一个流水线阶段可以同时完成Tn个乘法操作，Tn个数相加的操作，以及一个激活函数操作（图2.10（a））。对于MLP/CNN/DNN来说，输入可以被所有的输出神经元共享，而权值通常不固定，DianNao每次读取Tn个输入神经元送给Tn个输出神经元，权值最多需要读取Tn×Tn个数据。采用Tn=16的DianNao在保持功耗为485mW，1GHz主频的同时，面积开销仅为3mm^2（TSMC65nm工艺），而相对于2GHz SIMD结构的CPU加速比达到了117x，功耗降低21倍。在这基础上将NFU与周围的局部存储组成一个title（图2.10（b）），多个tile（DaDianNao中是 16）通过局部互联组成一个node（图2.10（c）），多个node通过HT2.0连接组成并行计算能力超强的DaDianNao。意法半导体ST 28nm工艺下，64node系统相较于GPU加速比达到450x，功耗降低150x。同时值得注意的是每个节点的面积开销达到了68mm^2。

2.2.3 神经生物学启发的硬件神经网络

神经生物学启发而来的生物神经网络具有和机器学习启发而来的神经网络具有截然不同的特性，更接近生物学上的观察和发现，然而在机器学习任务上的精度不尽如人意，使得其使用受到了极大的限制。卷积神经网络CNN研究方向的专家LeCun就曾

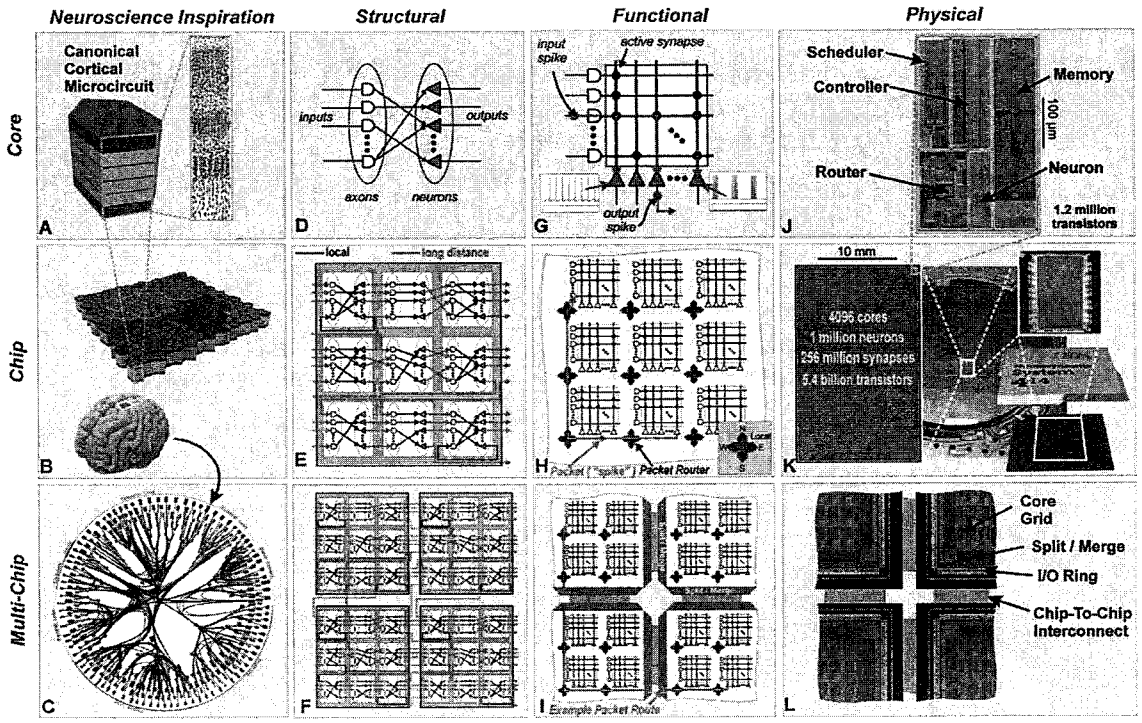

图 2.11　IBM TrueNorth结构图示[133]

表达过对于此类神经网络的悲观态度，他说，“在人工智能领域，他们寄希望于通过复制我们所知晓的神经元和神经突触的所有细节，然后在一台超级计算机上模拟规模庞大的神经网络这样的方式产生人工智能，这就是‘草包族科学’的人工智能。”[7]。然而，事实上很多的研究人员认为人工智能的基石是此类具有生物特性的神经网络。

目前，在硬件实现上兴起了一波采用新兴器件如忆阻器（Memristor）来搭建神经网络的研究浪潮，这其中包括 [143,156,158,179,186]，其中，在所有采用脉冲神经网络SNN和MNIST手写数字识别基准测试集的工作中，Querilioz等人 [157]的工作实现的精度则是最高的。Lee等人 [113]则采用了CMOS的CrossNet结构来实现神经网络；Eyrilmaz等人[55]则研究了采用非易失存储器下神经网络的学习问题；Smith则在2014年提出了如何用数字电路高效的实现不同的脉冲神经元模型方案 [177]；Vogelstein等人则比较了不同的脉冲神经元模型 [195]。

值得注意的是，作为BrainScale项目的研究子课题，Schemel等人则提出了基于Wafer-scale实现的脉冲神经网络模型 [167]。SpiNNaker[94]实现了10亿个神经元。作为项目SyNAPSE的一部分，IBM持续在生物神经网络芯片上研究多年。在阶段一中，Seo等人[170]提出了45nm的CMOS神经网络芯片，也包含学习训练部分；Cassidy等人[23]提出了针对生物神经模型的数字神经元实现方案；Merolla等人[132]提出了采用crossbar结构的数字芯片，单个脉冲的能耗为45pJ。在目前的阶段二中，IBM在2014年上半年发表了他们TrueNorth[133]的工作。单个芯片核是采用了Crossbar的结构自定制SRAM，将权值存储在这样的结构中的节点上。每个芯片核拥有256个输入

和256个输出神经元。每个芯片拥有4096个芯片核，总计一百万神经元和256百万个权值连接，共计54亿个晶体管。单个芯片平均放电频率为20Hz（功耗低至70mW），而单个神经元放电功耗低至26pJ，与普通处理器相比，节省能耗是176000x，远低于传统的数字芯片。IBM表示后期将单个芯片连接到一起组成多片网络互联结构，以提供更加强大的模拟能力。

无论是学术界还是工业界，TrueNorth自身充满了争议，业界许多研究人员诟病这样的芯片的用途，如卷积神经网络方向的研究专家LeCuun就表示“IBM的TrueNorth无法避免‘草包族科学’的帽子。尽管文章报告令人印象深刻，然而实际上没有实现任何有价值的东西” [7]。Neuflow的作者Culurciello则认为：“计算表明，Neuflow具有和TrueNorth相同数量级能耗比”；“从性能功耗比的角度上讲，与传统的数字芯片相比，TrueNorth并没有优势” [40]。不可否认的，这样的神经网络和芯片能够吸引更多的关注，进一步推动整个神经网络领域的发展。事实上，现在基本上很多国家地区公司都已经展开了类似的脑计划，欧盟开展了人脑计划（Human Brain Project），计划建造一种模拟神经元功能的芯片使其功能接近真正的生物神经元，然后将这样的芯片用于建造超级计算机，期待能够促进人工智能的发展；高通放出了关于Zeroth很少的消息[101]，只能肯定是生物神经网络模型的芯片；据悉，我们国家目前也出现了相关的脑计划项目。

第三章　神经网络加速器比较

3.1　引言

对于目前严峻的能耗限制，不论是嵌入式系统还是高性能服务器都开始寻求耦合加速器来高效地加速，也即对于某些特定的应用/任务采用自定制集成电路进行计算。相较于通用处理器，定制电路虽然灵活性低，但是却极其高效。FPGAs或者GPUs则是目前最通用的加速平台，Kuon等人 [102]的研究表明，ASICs比FPGAs节省约12倍的功耗，而GPU的功耗会更高 [26]。同时，除去极少的一些应用或者使用频繁的算法（如视频编解码），现状是针对单一算法定制的ASICs集成电路并没有很好的被异构多核系统或者嵌入式处理平台采用。对于加速器设计而言，确定合适的加速器是具有挑战性的任务，这需要该加速器应该具有比较广泛的应用场景，且加速器要能够高效的运行在这些场景中的应用和算法。

因此，机器学习成为特别吸引人的领域：机器学习算法的应用非常广泛，特别是用于处理真实世界输入信息（如图像和声音）的应用，例如图像和声音的分类识别应用。特别有趣的是，一些特别重要的关键算法，如深度神经网络（*Deep Neural Networks*，*DNN*），已经被证明在很多应用上是目前最好的算法 [99,106]。这些也为设计高效且具有广泛应用场景的加速器提供了极其适合的应用：只需要硬件化极少的算法，就可以通过该硬件完成广泛应用场景中不同的任务。不少研究人员已经确认这种趋势并提出了不同的硬件神经网络加速器 [63,187]。然而，深度神经网络DNN通常具有较大的规模，例如上千万的突触连接和几十万的神经元数目 [99]，这使得直接将DNNs映射成为硬件结构变得极其困难；也因此最近大多数研究工作提出的方案是将网络中少数神经元映射成为硬件神经元，如卷积神经网络（Convolutional Neural Networks，CNNs：CNNs也被认为是DNNs的一种变种） [63]，或者是全部神经元映射成为硬件神经元但是这只限于小规模的网络，如规模为90个输入神经元、10个输出神经元的多层感知机（Multi-Layer Perceptron，MLP） [187]。

同时伴随着这种趋势，另外一种新兴的硬件神经网络加速器也蓬勃发展：从神经生物学启发而来的神经网络算法。不同于机器学习启发而来的、与生物上的神经网络关系不那么紧密的神经网络算法，这类神经网络大多直接来自神经生物学上的生物模型 [43]，也即和生物事实（现象和观察）具有紧密的联系。对于此类神经网络算法来说，最主要的目标是模拟大规模的生物神经网络，从而进行生物学上的观察。有趣的是，研究人员也考虑了将此类神经网络算法应用到其他不同的任务上。这类神经网络最主要的进展来自两家商业公司，分别是IBM（TrueNorth [132]）和高通（Qualcomm，

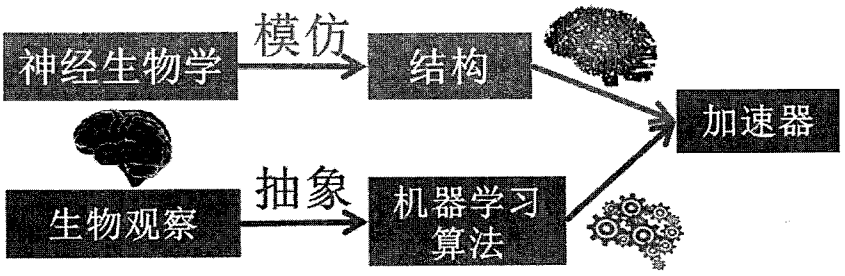

图 3.1 神经网络方法比较

Zeroth处理器 [101])。这两家公司都在推动此类神经网络的相关硬件实现，并最终有可能实现成为硬件化的商业产品。此类神经生物学启发的神经网络算法有以下几点重要的事实：（1）已经被成功的应用到复杂的机器学习类任务上，如人脸识别和手写字体识别 [130,158]，这意味着此类神经网络也可以成为可能的加速器设计选择；（2）由于采用了脉冲编码信息（脉冲神经网络，Spiking Neural Networks，也即SNNs），此类神经网络可以大量节约硬件开销，如面积、功耗 [13]；（3）神经生物学启发的学习算法STDP（Spike-Timing Dependent Plasticity）能够使得神经网络持续进行学习，从而具有极高的自适应性。

也正是因此导致了从面积开销、速度、精度和功能性的角度上，嵌入式系统中应该采用哪种神经网络算法成为了一个的问题。不幸的是，这两类神经网络来自两个极其不相同的领域：*机器学习*和*神经生物学*。他们具有极其不相同的特性。同时，目前也没有量化的研究去比较两类神经网络的精度和硬件开销。为了填补这项空白，我们对这两类神经网络进行了严格的量化比较。我们选取了每类神经网络中最知名的算法，分别是多层感知器MLP和脉冲神经网络SNN；采用反向传播算法（Back-Propagation，BP）训练MLP [109]，采用STDP [130]训练SNN；采用MNIST [109]手写数字识别作为驱动示例；同时，为了获得最好的网络参数，我们分别尽力搜索了神经网络算法设计空间①，然后在65nm工艺下将神经网络进行硬件化实现（完成后端设计②），从而获得神经网络硬件化的相关参数如面积、能耗、速度和精度。另外，为了验证我们从驱动示例得到的结论，我们也将比较应用到其他应用测试集上，如图像识别和语音识别：MPEG-7物体识别测试集 [4]和阿拉伯数字语音识别 [9]。

对于以下几个硬件神经网络加速器的相关问题，我们尝试通过本章的研究给出答案：

① 对MLP和SNN，我们对设计空间搜索了不同的拓扑结构和操作符类型。对于SNN，我们还尝试了（1）不同的编码方式，4中rate coding和2种temporal coding编码方式；（2）神经元参数（参见章节 3.5）。在尝试不同SNN结构的同时，我们尽力重新实现IBM的TrueNorth核 [132]用以比较(参见章节 3.5)。

② 与之前多数工作保持一致 [29,132,158], 加之大多数的应用可以兼容离线训练 [29,187]，我们也着重于神经网络的前向通路（用于测试或者使用）的硬件实现。但是，不同于BP，STDP学习算法可以进行在线训练，我们也研究了STDP和与之相关的硬件开销。

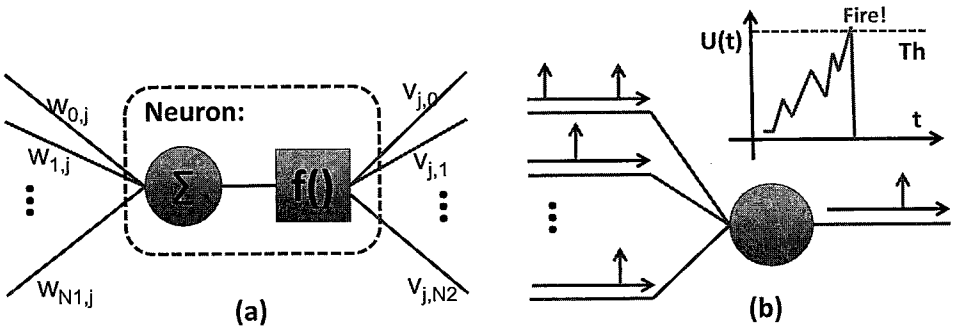

图 3.2 MLP (a) vs. SNN (b)

1. 在识别类的任务上，神经生物学启发的算法（如SNN+STDP）是否能够在精度上达到甚至超过机器学习启发的算法（如MLP+BP）?

2. SNN的硬件开销能否比MLP显著降低？

3. 在什么情况下应该采用SNN或MLP硬件神经网络加速器？

由于SNN和机器学习启发的神经网络算法的不断改进，以及可用于实施这些神经网络的硬件不断增多，对于这些问题我们无法提供明确的回答。然而，对于当前的神经网络算法和在当前的硬件技术下，在我们尽最大的努力完成硬件电路并使用有限的测试集（特别是MNIST，这是少数几个SNN和机器学习启发的神经网络共同使用的测试集中的一个）进行评估的情况下，我们可以对当前的两种方法提供一些初步的结论。

在研究的有限范围内，我们提供以下结论：

(1) 当考虑完全相同的识别任务（例如MNIST手写数字识别），从神经生物科学启发的算法如SNN+STDP的识别精度，甚至比不上简单的机器学习所启发的算法如MLP+ BP。我们还发现造成精度差别原因并不单一，相对于脉冲编码的信息丢失，精度损失更多的是因为STDP训练算法本身。同时在这项研究中，我们尝试去填补SNN+STDP和MLP+BP之间的精度鸿沟。

(2) 虽然在硬件开销上，全展开（每个逻辑神经元和突触分别映射到一个硬件神经元模块和一个硬件突触模块）的SNNs相对于MLPs有显著的优势（面积和功率），但是考虑合理可实现的硬件结构时（有特定开销限制时），MLPs却比SNNs更具硬件上的优势。采用简化的SNN神经元模型以及容忍一定的精度损失，我们可以缩小两种SNN模型硬件开销之间的差距。然而，MLPs仍然比简化的SNN算法在硬件上更加高效。

(3) 尽管需要永久在线学习的应用程序范围（相较于离线学习）可能会比较有限，我们观察到实施SNN学习算法STDP的硬件开销（面积和功率）很低。因

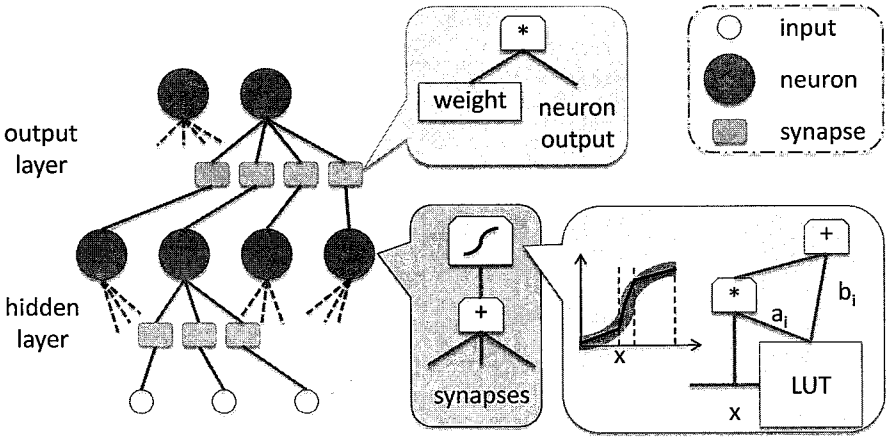

图 3.3　2层MLP中的操作符

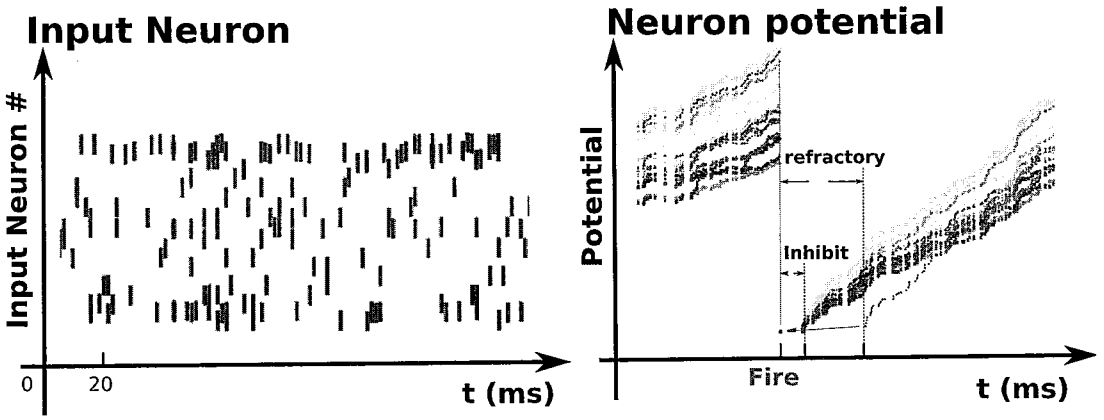

图 3.4　SNN中的脉冲编码：*(左) MNIST*测试中*300*个神经元的所有脉冲（每条线表示一个神经元）。*(右)* 不同神经元的电位（*Potential*）随着接收到脉冲不断增加（每条线表示一个神经元的电位），直到某个神经元激发（*Fire*）：激发的神经元会随后进入一段*20ms*的耐火期（*refractory*），其他神经元则进入*5ms*时长的抑制期（*inhibition*）。基于自稳态机制（*homeostasis*），所有神经元的激发阈值并不相同

此，需要永久在线学习和能容忍一定精度损失的应用程序是SNN+STDP加速器的绝佳选择。另外，快速和大规模的硬件实现（在空间上扩展的）也应该考虑SNN+STDP的设计，特别是如本文中所探讨一样假设精确性可以通过改变训练算法来提高。

本章节其他部分安排如下：章节3.2中我们介绍两种目标算法（MLP+BP和SNN+STDP）；章节3.3中我们比较两种算法的精度并尝试缩小之间的精度差距；章节3.4我们考虑不同的硬件实施方法（全展开模型和空间折叠模型），也包括一个简化模型的SNN算法和SNN的在线学习（STDP）；章节3.6我们介绍与本章节研究相关其他研究。

3.2　神经网络

在本节中，我们简要介绍两种目标算法（MLP+BP和SNN+STDP）包括前向通路

表 3.1 MLP和SNN的相关特征

MLP参数	范围	本文	说明
# N_{hidden}	10-1000	100	隐层神经元数目
# N_{output}	10	10	输出神经元数目
η	0.1-1	0.3	学习率
# epochs	10-200	50	训练回合
SNN参数	范围	本文	说明
# N	10-800	300	单层，神经元数目
T_{period}	100-800	500ms	图像展示时间
T_{leak}	10-800	500ms	泄露时间常数
$T_{inhibit}$	1-20	5ms	抑制期时间
T_{refrac}	5-50	20ms	耐火期时间
T_{LTP}	1-50	45ms	LTP阈值
T_{init}	w_{max}*70	17850	初始激发阈值
$Homeo_T$	$10 * T_{period} * (\#N)$	1,500,000	自稳态回合（ms）
$Homeo_{th}$	$\frac{3*Homeo_T}{T_{period}*(\#N)}$	30	自稳态阈值

和在线学习模式，用来作为硬件实现的指导，最后我们比较了在基准测试集MNIST上两种目标模型的精度。

3.3 精度比较

3.3.1 SNN+STDP vs. MLP+BP

为了比较两种算法在精度上的表现，我们选择广为人知的用于手写体数字识别的机器学习基准测试数据集MNIST [109]。对MNIST来说，ImageNet2012年竞赛获胜的网络（使用的是CNN）使用6000万突触连接65万神经元实现的精确度为99.21% [99]。然而这个网络的规模远远超出合理硬件的实现范围。目前最好的结果（99.77%）来自于一个更加复杂的网络MCDNN [34]。

这里针对MNIST，我们训练了相对小的MLP（隐层只含100个神经元，见表3.1），其精度为97.65%，比其他研究中使用MLP得到在MNIST上最好的精度结果略低了一点[175]（使用800个隐层神经元实现精度为98.4%）。值得注意的是，我们的精度结果是在网络中使用100个隐层神经元得到的，这是经过对隐层神经元数目进行遍历后的选择。我们测试了使用不同隐层神经元数目时（从10个隐层神经元的数目到1000个隐层神经元，同时探索了其他参数，如学习率）的网络精度，并发现当隐层神经元超过100个后，相对于增加的开销，精度增益微乎其微，但硬件的开销却显著增大（也即精度增加进入平台期）。这里还需要注意的是，我们使用的是全60000幅非扭曲的原始MNIST图像，并使用全部10000幅测试图像进行测试。

本问题研究中针对SNN，我们做如下训练。每个像素的亮度会转换成为一串泊

表 3.2 目前已知MNIST上的最好精度结果（非扭曲）

Type	Accuracy (%)
MLP+BP [175]	98.40
SNN+STDP [158]	93.50
SNN+STDP [48]	95.00
ImageNet [99]	99.21
MCDNN [34]	99.77

表 3.3 在MNIST上MLP和SNN的精度结果

Type	Accuracy (%)
SNN+STDP - LIF (SNNwt)	91.82
SNN+STDP - Simplified (SNNwot)	90.85
SNN+BP	95.40
MLP+BP	97.65

松脉冲序列，序列的频率与像素的亮度值成正比，例如，255的最大亮度对应于平均50ms的脉冲周期（20Hz）。泊松过程的参数λ值对应于下面的表达式：$\frac{1}{3*U-2*\frac{p(i,j)}{255}}$，其中$U$是一个预先定义的表示最大峰值脉冲频率的常数，$p(i,j)$是输入像素的亮度值。我们做了细粒度的探索以决定SNN的其他参数，包括SNN神经元个数（#neurons）、图像刺激持续时间（T_{period}）、泄漏时间常数（T_{leak}）等（参见表 3.1，其中列出了参数最后选择的数值）。需要注意的是，尽管SNN是从神经生物科学启发而来的算法，但是我们这里的研究主要目的是计算效率和准确度。因此，当涉及到校准参数时，我们不会为了与神经生物科学上的观测保持一致而特别严格的遵守特定数值，我们只选择那些能够产生最好的结果的参数数值，例如，我们的探索实验表明，最佳精度用的泄漏时间常数值为500ms，而已知的神经科学文献中这个常数值是在50ms左右。

我们在表 3.2和表 3.3中展示已知最好的精度结果和我们本问题研究中实验得到的精度结果。我们可以观察到，在MNIST上SNN+STDP的精度比MLP低5.83%，这一结果是和目前SNN+STDP上最好的结果一致，该结果比MLP的结果低1.68%（Querlioz等人得到的93.50% [158]。请注意，我们尝试的所得到的结果是通过仔细复制Querlioz等人所使用的算法，从而和Querlioz等人所用的算法保持一致，我们甚至与作者联系，但由于我们不能访问他们的软件所以不可能完全弥补这一差距。一言以蔽之，在MNIST使用SNN+STDP已有的最好结果（我们的91.82%或者最好结果93.50%），相比MLP有5.83%至4.15%的精度差距。尽管这一差别对于某些应用程序可能很致命，例如，造影分析癌症患者的生死决策，然而对于其他的一些应用这个精度差距是可以接受的，例如，使用智能手机识别标记物体的应用程序，如谷歌的Goggles。总而言之，SNN+STDP识别的精度明显不如MLP+BP，但对于许多应用其精度已经足够高。

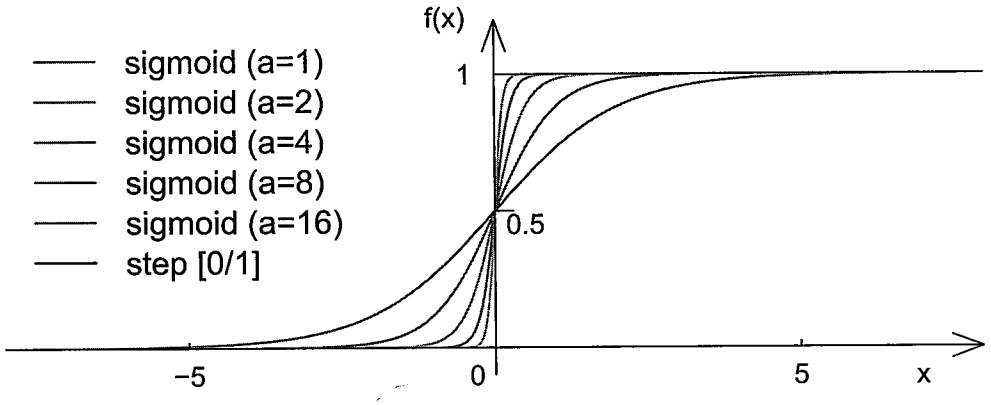

图 3.5 激活函数曲线（参数化sigmoid函数和阶跃函数）

请注意，Diehl等人 [48]使用类似于 [158]的SNN模型得到了95%的精度结果，但是这个结果是用6400个神经元得到的（400个神经元精度为82.9%，而我们使用300个神经元得到91.82%的精度结果）。

3.3.2 跨越SNN+STDP和MLP+BP之间的精度鸿沟

3.3.2.1 SNN+BP

这里我们要尽最大努力去进一步分析SNN+STDP和MLP+BP之间精度差异的来源，着重于两个学习算法本身之间的两个主要差异。首先，STDP是无监督学习算法，而BP是一个监督学习算法。STDP的原理 [128]完全依靠脉冲发生在突触前和突触后的神经元的时间，也即本地信息，这里并没有一个全局性的如同BP中的误差信号或者类似的概念；同时也没有错误沿着输入输出通路反向传播的概念。其次，STDP是一个在线学习算法，而BP是一个离线算法。在线学习规则如STDP存在一个问题，当新的信息被接受后如何保留早期记忆。在单个神经元模型中，研究表明学习规则对记忆保留时间有很大的影响 [18]。研究表明，尤其是当有足够的侧抑制能够稳定感受域，感受域的稳定性是记忆保持的时间跨度衡量 [18]。

为了评估学习算法的影响，我们用反向传播训练SNN。我们进行如下操作：在前馈正向模式下，我们使用的SNN与之前完全相同（峰值，泄漏时间常数，激发阈值等），但每个图像刺激后，我们计算输出错误，并用BP将其传播到使用的突触权重（梯度下降和权重更新）。我们在表 3.3中报告精度结果，可以看到，精度已经从91.82%升至95.40%，即只有2.25%的精度差距（SNN+BP和MLP+BP之间）。这突出表明了脉冲编码只造成了极小的精度损失，大部分的精度损失则是由于学习算法（STDP）。

3.3.2.2 阈值函数 vs. Sigmoid

我们试图进一步缩小SNN+BP和MLP+BP之间的精度差距。在这里，两者的主要区别在于脉冲编码和激活函数的使用。SNNs使用的阈值函数是一个简单的[0/1]阶跃（Step）函数（无脉冲/有脉冲），而MLP使用了Sigmoid作为激活函数，参见图 3.5。通过

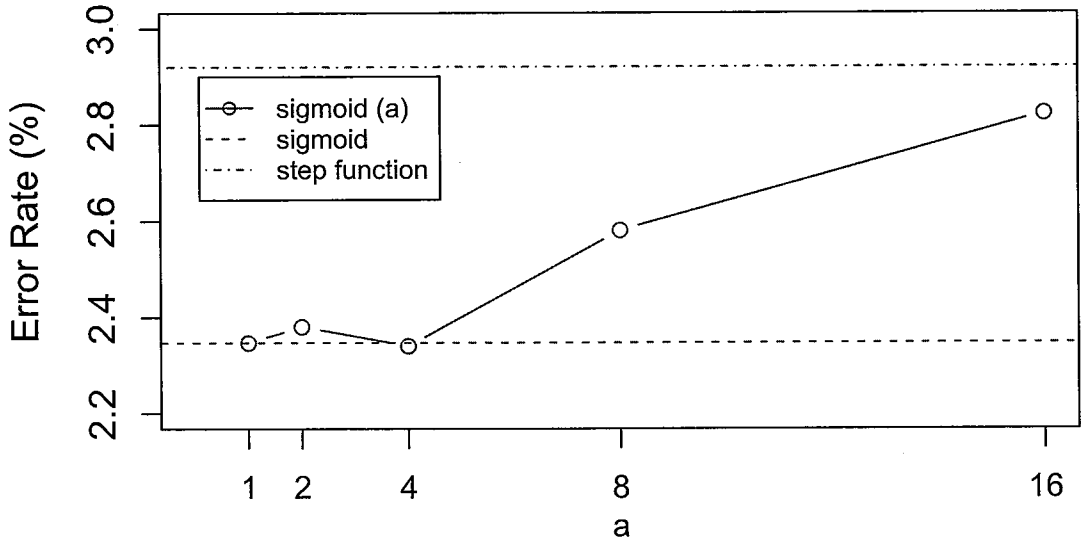

图 3.6　跨越sigmoid和step激活函数的精度差别

参数化的Sigmoid函数$f_a(x) = \frac{1}{1+e^{-ax}}$而不是$f(x) = \frac{1}{1+e^{-x}}$，我们能够逐步地改变Sigmoid的函数曲线，使其逐步地趋向于阶跃函数；其中a是关于坡度（斜率）的一个参数，a越大，则Sigmoid函数越接近阶跃函数，参见图3.5。我们训练并测试MLP+BP在a不同值下的精度结果，并在图 3.6显示出相应的误差结果。可以看到，当随着a增加，使用Sigmoid作为激活函数的MLP+BP误差变得越来越接近使用阶跃函数的误差，使得SNN+BP和MLP+BP之间的差距完全被弥补。

总体而言，我们观察到脉冲的编码对SNN+STDP和MLP+BP之间的精度差异只部分相关：而学习算法（STDP）则对于精度有着更大的影响，而唯一和脉冲相关的方面是使用了阈值函数来生成脉冲，而不是MLPs所使用的Sigmoid函数。虽然我们硬件实现的SNN是采用的阶梯函数产生脉冲（参见后文），这里的分析为进一步弥补SNNs和MLPs的精度差距提出了一个研究方向。

3.4　硬件开销对比

在本节中，我们研究MLPs和SNNs硬件实现的权衡取舍。正如引言中所述，我们重点放在硬件实现前向通路上，因为绝大部分应用程序采用离线学习就已足够。这里我们考虑两种设计：空间展开的方法（最直接的也是最昂贵的的方法）和空间折叠的方法。前一种方法中，硬件设计直接对应神经网络中的概念，即所有组件（神经元，突触）被映射到独立的硬件组件，也即空间展开的硬件实施方法。由于这种设计的成本十分昂贵，它们只能对应于小规模的神经网络，因此它们不适用于有着实际应用意义的场景。后一种方法，设计中硬件组件（尤其是神经元的输入）被多个逻辑器件采用时间共享的方法完成原始神经网络的计算，也即空间折叠的硬件实施方法。为了公平起见，我们尽量对MLPs和SNNs的实施做相似的选择。

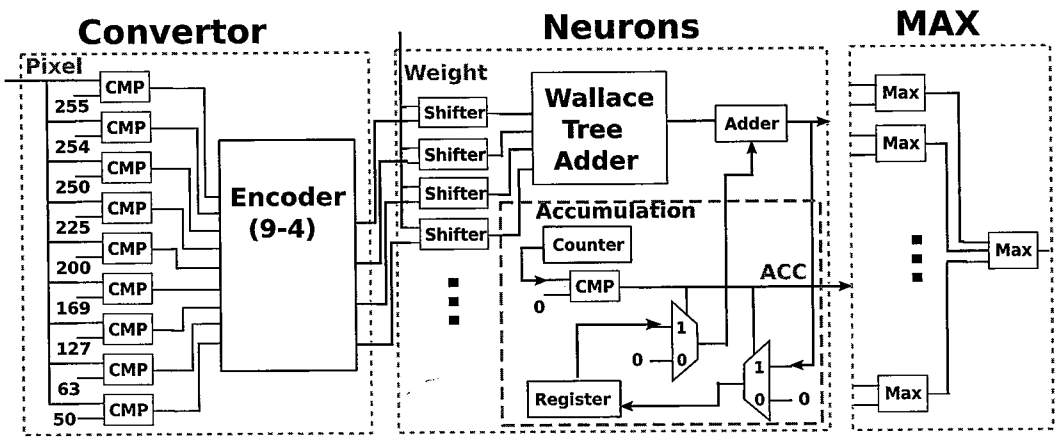

图 3.7 SNN硬件实现

3.4.1 方法论

我们在RTL级实现了前面章节中描述的不同的算法，然后综合和布局布线，得到后端结果。在某些例子中（空间展开版本），由于电路太大，我们基于单个神经元的布局和布线的结果去估计整个电路的面积。我们使用Synopsys公司的产品Design Compiler（TSMC 65nm GPlus VT工艺库）进行综合，IC Compiler进行布局布线。使用后端布局布线后的版图，我们使用Synopsys公司的VCS和Prime Time PX进行设计评估，以获得准确的功耗和能量。同样，本章中所有报告的面积、延迟、功耗和能源都是通过这些工具获得的。

对于精度的比较中，由于RTL级仿真的速度导致训练时间过长，我们对SNN+STDP和MLP+BP实现了两个C++模拟器完全模拟硬件行为。这里，我们用两种网络的RTL结果验证我们所实现的模拟器。

3.4.2 空间展开

3.4.2.1 MLP

对于MLPs，空间扩展的硬件设计是类似于图 3.3中所示的概念。硬件神经元是一组乘法器的实现，每个乘法器对应一个输入突触，随后通过加法树来累积所有的乘法结果。Sigmoid函数是采用16分段线性插值实现的，硬件上仅需要一个小的SRAM表来存储插补点的值（每点两个系数），一个加法器和一个乘法器 [187]。插入函数f在i分段x点的值由公式$f(x) = a_i \times x + b_i$得到。我们对运算符的最佳大小和Synaptic存储（通过在MNIST基准测试集上反复训练/测试）进行评估，最后我们发现，实现8位的定点运算符（乘法器，加法器，SRAM的宽度）的精度结果和采用浮点运算符的精度结果分别是96.65%和97.65%。

3.4.2.2 SNN

脉冲时间对常用于SNNs的生物现实STDP学习策略会产生根本性影响 [178]。然而在前向通路中，脉冲时间所蕴含的信息是极其重要的这点并不十分明显。忽略脉冲时

间能够避免需要仔细地模拟脉冲时间信息和同时地响应脉冲输入，从而能够对计算带来显著的加速比。

因此，我们首先评估在前向通路中时间信息的重要性和影响。我们考虑两个SNN版本：一个是传统的LIF模型，如部分呈现在章节 2.1.2.1中（我们称之为SNNwt，“含时间信息的SNN”）；第二个是简化的模型，其中所有的时序信息已被移除（我们称之为SNNwot，“不含时间信息的SNN”）。对于SNNwot，每个像素被转换成一组脉冲，采用和传统的LIF相同的原则，只有这里获得是脉冲尖峰的数目，而不是脉冲之间的时间；类似地，泄漏的作用在这里被忽略。我们报告了相应的精度结果（如表 3.3中所示）。我们观察到精度的差别是1.03%，虽然这个是不可忽视的，但是对于所获得的速度收益这点差距已经足够小到可以接受的了。我们现在开始描述各个版本的硬件实现。

不含时间信息的SNN（SNNwot）

脉冲的产生 我们首先需要将像素信息（这里，8位灰度）转换为脉冲；虽然有些传感器能够直接产生脉冲[172]，但这里我们做保守假设使用的仍然是传统的CMOS传感器。

忽略脉冲之间的时间间隔能够显著地减少硬件的复杂性并且提高执行速度：我们只需要计算给定的一个像素值所能生成的脉冲数。然而，我们必须生成和STDP学习中数目相同的脉冲个数，从而获得和前向通路一致的计算结果。像素到脉冲的转换如下：8-bit像素能产生多达10个脉冲（图像之间的延迟也即单幅图像的展示时间是500毫秒，且最小脉冲间隔为50ms），脉冲数目被直接转换为4-bit的值，参见图3.7，而不是STDP所需要的脉冲序列。

神经元 每个脉冲的到来都会使得神经元的电位升高（获得对应脉冲权值），如图5.8所示。对于一个给定的输入与突触权重W，权重的积累等于$N \times W$，其中N是脉冲数目。由于N的有限取值范围（$N \leq 10$，即脉冲数目不大于10），这里采用的高效硬件实现的乘法器包括4个移位器和4个加法器，用于计算$n_3 * 2^3 * W + n_2 * 2^2 * W + n_1 * 2^1 * W + n_0 * 2^0 * W$，其中$N = n_3 n_2 n_1 n_0$，参见图 3.7。所有的输入结果的累加运算都是通过一个华莱士树（Wallace Tree）加法器完成。

读出 当神经元电位达到阈值后，神经元将激发出一个输出脉冲。SNNs中一般有两种方法来确定对当前输入刺激最后获胜的神经元：第一种是最典型的，即在当面刺激下统计每个神经元输出脉冲的数量，脉冲数目最多的神经元获得竞争胜利；另一种方法是基于输出脉冲的激发速度，即将检测第一个激发输出脉冲的神经元[188]作为最

表 3.4 SNN和MLP空间展开的硬件实现

网络	运算符	面积/运算符 (μm^2)	#运算符	总运算符面积 (mm^2)	总面积无SRAM (mm^2)	SRAM面积 (mm^2)	总面积 (mm^2)
SNNwot	adder tree	89006	300	26.70	26.79	19.27	46.06
(28x28-300)	max	6081	16	0.10			
SNNwt	adder tree	60820	300	18.25	19.62	19.27	38.89
(28x28-300)	rand	1749	784	1.37			
MLP	adder tree	45436	100	4.54			
(28x28-100-10)	adder tree	5657	10	0.06	73.14	6.49	79.63
	multiplier	862	79510	68.54			
MLP	adder tree	45436	15	0.68			
(28x28-15-10)	adder tree	1131	10	0.01	10.98	1.35	12.33
	multiplier	862	11935	10.29			

后竞争的获胜者。这两种方法都依赖于为了效率而在硬件中移除的时间信息，并不适合硬件实现。然而，神经元电位和输出脉冲的数量是高度相关的，因此类似于第一种方法，我们将具有最高电位的神经元作为对当前输入刺激的竞争胜利者。

简而言之，在硬件中的实现是一个三级流水线的结构，一级流水用于的脉冲产生，一级流水用于加法树和一个用于最大电位读出的操作的流水级。

含时间信息的SNN（SNNwt） 为了使用脉冲时间信息，我们必须对每个输入像素随机产生脉冲数目和脉冲之间的间隔时间，如章节3.3.1中和图3.4所示。为了加快执行速度，同时最小化神经元随机数产生模块的数目，我们仅产生的脉冲的时间间隔，并且由于每幅图像会作为输入刺激对网络产生固定时长的刺激（在我们的例子中是500ms，硬件中用一个时钟周期模拟1ms），当图像刺激时长到达时我们停止产生脉冲，也即我们隐式的决定了脉冲产生的数目。

通常来说我们应该采用泊松Poisson随机数产生模块，如章节3.3.1所述，但是这样的随机数产生模块通常来讲开销比较大[111,116]。我们已经尝试了各种替代方法来产生脉冲时间信息，同时我们观察到，尽管Gaussian分布不如Poisson分布那么类似生物机制，但是精确度基本没有受到影响。幸运的是，高斯伪随机数发生器可以通过使用中心极限定理高效地实现[126]，其原理是可以加和从四个线性反馈移位寄存器（LFSR）生成的符合均匀分布的随机数，加和的结果即基本符合高斯分布。我们采用31位作为LFSR的长度和$x^{31}+x^3+1$作为本原多项式以避免获得循环使用过的LFSR数值。单高斯随机数发生器的面积开销为1749μm^2，而我们使用784个这样的发生器（和输入的数量一致）。这些随机数发生器产生毫秒级的脉冲之间的时间间隔（一个时钟周期为一毫秒），时间间隔被存入计数器，每个周期计数器递减，当计数器递减至0则新的脉冲被

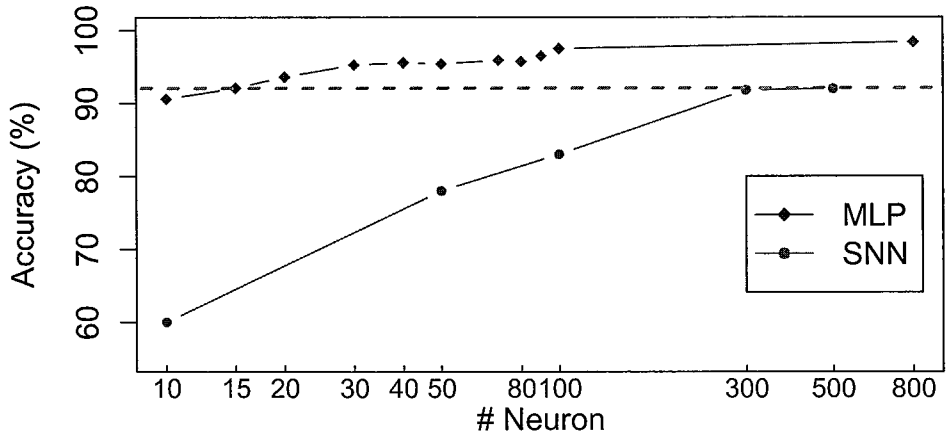

图 3.8　神经元数目对MLP和SNN精度的影响

激发。

3.4.2.3　比较

我们现在比较两个网络的空间扩展硬件实现的面积开销。每个网络具有28x28相同的8位灰度的输入（MNIST图像）。对于每一个网络，我们采用在文献中常见的结构：2层的MLP（一个隐层）[109]，1个包含侧向抑制的大层SNNs[159]。在MNIST基准测试集上，我们尝试搜索网络配置，从而使得两个神经网络的神经元个数达到最小。考虑图3.8，我们改变MLPs和SNNs中的神经元数目，我们可以观察到使用大约100个隐层神经元时MLP的精度变化已经变得很平坦，在使用300个神经元时SNN的精度变化变得平坦。

我们还探讨了神经元和突触的数据位宽，在目标为MNIST的精度相对使用浮点数变化在1%的约束下寻找最高效的实现。在MLP中我们最终使用8位运算符和权重。对于SNN、SNNwt使用8位权重，而SNNwot使用12位的权重（8位权重×脉冲数目）。这是神经网络学习算法的特性之一，即能够通过训练补偿这种低精确度。

MLP网络太大，所以在我们的服务器上无法进行后端设计（估计面积开销约73mm^2），这里我们通过如下方法进行评估：我们首先估计这两个网络中的基本运算器和存储器，然后基于单个运算器和存储器的面积开销去计算总的面积开销。同时我们还实现了两个小规模的神经网络。在表3.4中，我们报告了评估的整个神经网络面积开销，其中的基本运算器和和寄存器使用的是台积电TSMC的65nm GPlus HVT工艺库进行评估。我们注意到MLP版本面积开销远远大于（2.72x）具有更多神经元的SNN，这是因为在MLP使用有乘法器而其开销比较高，而SNN神经元则是使用加法器。

两个小规模网络的后端版图级比较　这些结果由完全实现两个小规模的神经网络设计进一步证实，参见表3.5，其中输入被限制为4×4，MLP采用了10个隐层的神经元

表 3.5 SNN（4x4-20）和MLP（4x4-10-10）硬件参数比较

Type	Area (mm^2)	Delay (ns)	Power (W)	Energy (nJ)
SNN	0.08	1.18	0.52	0.63
MLP	0.21	1.96	0.64	1.28

和10输出神经元，而SNN使用了20个的神经元。从表中可见，延迟和能源比率和面积比大致相同，而仅从功耗来比较来看两个网络是相似的，部分原因是因为时钟功耗在两个网络中不一致，在SNN中占据更大的份额（60%对MLP中20%）。

同精度下面积比较 我们还大致比较了实现类似精确度时MLP和SNN的面积成本。由于SNN相比于MLP具有较低的精度，我们减少MLP中隐层神经元的数量，直到其准确性几乎是同样的，也就是15个隐层神经元时MLP的精度为92.07%（我们在SNN上获得MNIST的最好精度为91.82%），见图3.8。然而，MLP的面积分别比SNNwt和SNNwot小68.30%和73.23%。通过这进一步的分析，我们证实MLP显著优于SNN。

3.4.3 空间折叠

对章节3.4.2中描述的空间展开神经网络版本的面积估计表明，这种网络实现的开销过大与低面积开销的嵌入式系统不太兼容。以前的一些工作实现MLP [187]具有大约几个mm^2的面积开销，然而该设计针对的是加州大学欧文分校的机器学习测试库UCI [9]，这些数据集通常只有很少的输入和输出；最终，他们所实现的MLP具有90个输入神经元，10个隐层神经元和10个输出神经元。嵌入式系统的应用中，输入中28x28像素的灰度图像已经是应用范围中很小的应用，因此可以公平地指出，空间扩展对于大多数设备来讲是过于昂贵的设计。

尽管空间扩展的神经网络设计通常可以提供最大的计算速度，但是采用空间折叠的神经网络设计是可能的，其原理是一些硬件的神经元分时计算目标神经网络中的多个逻辑神经元；空间折叠的硬件神经网络设计早在单晶圆片中晶体管数目都很少的年代就曾经出现过 [83]，在最近的卷积/深度神经网络加速器中的得到特别应用[29]。我们现在实现MLP和SNN设计并比较他们的空间开销，从而实现具有合理面积开销的神经网络加速器。

3.4.3.1 MLP

硬件神经元 由于神经元存在高连接性（大量的输入和突触权值相乘），空间展开的设计成本过高。因此，在空间折叠设计中，输入端被限制为单个神经元每次可以接收的最大输入数：这里我们用n_i表示；因为所有的神经元可以共享输入，这大大地降低了对输入带宽的需求。折叠设计中的每个硬件神经元必须累积$e \times C$的乘法结果，并

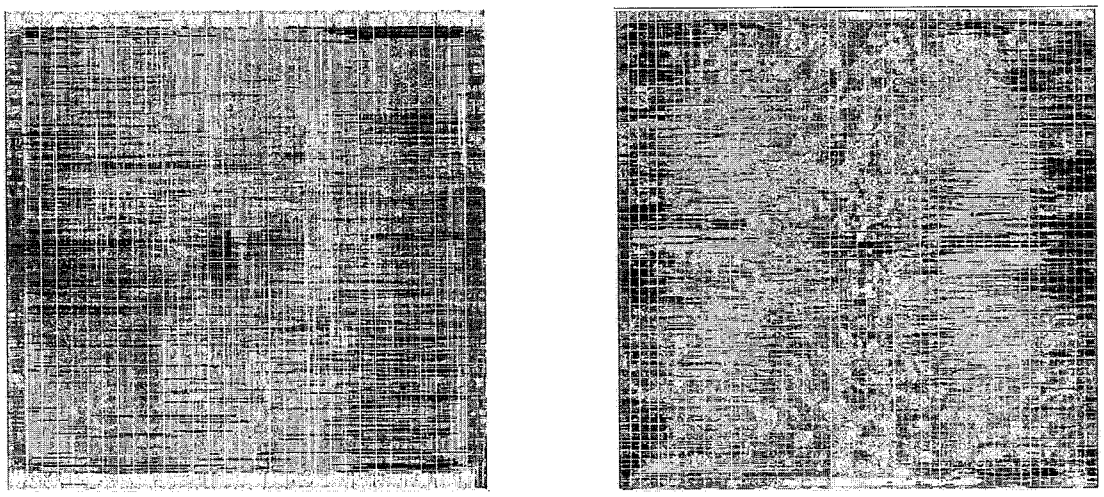

图 3.9　MLP版图（左, $n_i = 16$），SNN版图（右, $n_i = 16$）

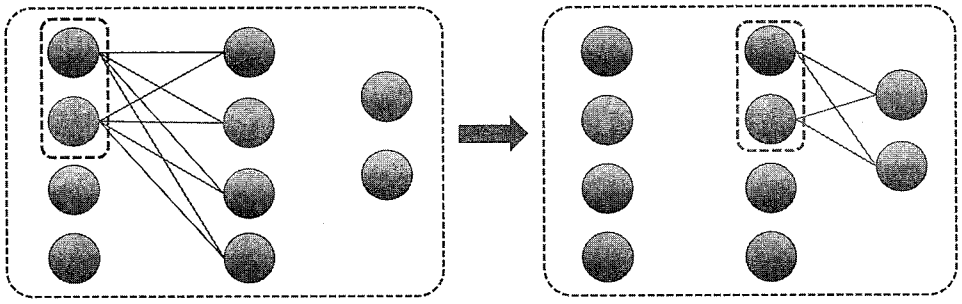

图 3.10　空间折叠设计中MLP的调度策略

且它必须包含一个寄存器来存储中间结果，请参见图 3.10中$n_i = 2$的例子。由于权重对于每个输入是独立的，它们必须每次对不同神经元载入一定数量的权值;所以神经元中还包含n_i个寄存器用来存储输入，SRAM存储模块用来存储神经元所有的相关权重.（SRAM为单端口，宽度为$n_i \times 8$位；它必须包含所有N_i的输入，其深度为$\frac{N_i}{n_i}$）。相应的硬件MLP神经元的设计在图 3.11中示出。

调度　由于空间折叠，硬件神经元现在需要$\lceil \frac{N_i}{n_i} \rceil + 1$个周期来计算其输出（"+1"这里是激活函数中的乘法器和加法器功能）。虽然这种多周期计算不利于如同空间展开设计中一样进行全流水线执行（即每个周期处理新的图像），它仍然是可能实现部分流水的，其中每个阶段都需要多个执行周期（如同处理器中计算大部分浮点运算）来完成计算。另一方面，相较于空间展开的设计，较少数量的输入可以允许实现更高的时钟的周期，见表 3.7，其中硬件神经元的关键路径延迟随着输入n_i数量的减少而减少。需要注意的是隐层并行的计算每个神经元的输出；结果存在输出寄存器中供输出层神经元使用它们作为输入，同样是每次载入n_i个数目的输入（相同的硬件神经元特性）。

评估　在表 3.7中,我们报告MLP空间折叠设计的结果，在MNIST上的面积、延迟、功耗和能耗（和空间展开设计中具有相同数量的硬件的神经元）。我们观察到，一个硬

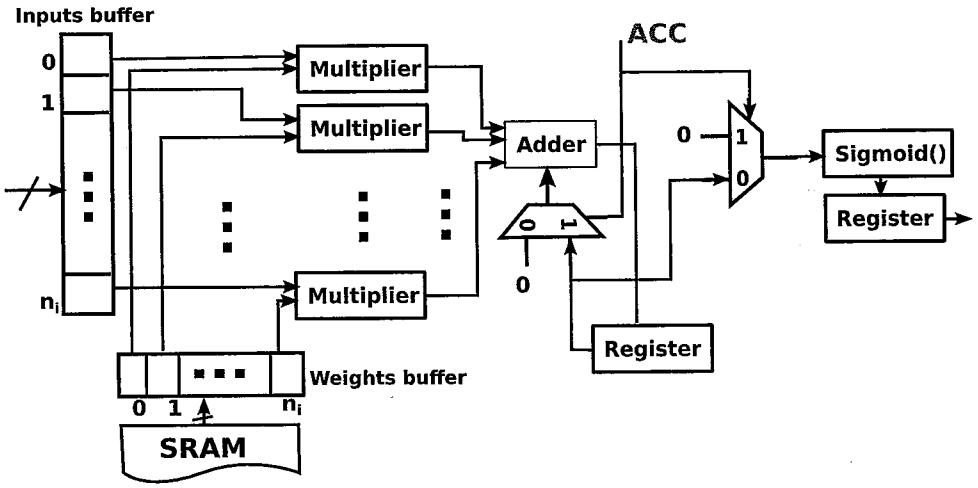

图 3.11 MLP神经元

件神经元采用n_i = 16输入（784个输入则为空间展开的神经元设计）而完成的整体设计，比空间展开设计的面积小38.84x，也更符合嵌入式平台，而处理操作一次MNIST输入仅需要57个周期（435MHz）。硬件神经元为采用n_i = 4输入而完成的设计相比空间展开设计的面积小117.76x，每个输入需要223个周期（同样435MHz）。另一方面，空间折叠设计的能耗是空间展开的5x大，然而空间展开的设计具有更高的功耗，这是因为其具有更多更大的运算模块，但一个图像在空间折叠设计中需要花费15.16 ns（4个周期，每个周期的时间为3.79ns），或是128.25ns（57个时钟周期，每周期2.25ns），或是1975.68ns（882个时钟周期，每周期为2.24ns）。

3.4.3.2 SNN

SNN神经元的空间折叠设计采用和MLP神经元一致的设计原则。输入的数量也同样被限制为n_i，权重存储在SRAM中（宽度为$n_i \times 8$位），每周期输入到n_i个寄存器中。和MLP折叠设计的神经元不同的是，SNN折叠神经元中包含一个加法器用来处理n_i个输入，而不是乘法器;在接收到各组n_i个脉冲输入，神经元电位增加。当所有输入都已经被处理后，所有神经元的电位传递给选取最大数的运算模块以得到最后的竞争胜利者。同样的，这选取最大数的运算模块也可以通过限制输入的数量进行折叠，就如同神经元折叠一样。但我们发现，一个两级的最大数选择树（15个最大数运算模块，每个有20个输入，后面是一个最多有15个输入的最大数运算符）只占最小的空间折叠实现的SNN（n_i = 1）面积开销的5.6%（178,448μm^2），所以我们决定这里继续采用全展开的最大数运算符设计。

这里我们报告SNNwt和SNNwot空间折叠实现的面积、延迟、能耗和运行周期数，参见表3.7。 SNNwt相较于SNNwot面积开销显著降低，这是因为SNNwot需要同时处理多个输入脉冲，因此需要运算模块能够同时接受比SNNwt可接受脉冲更多的输入（n_i×单个像素能够达到的最大脉冲数目）。另一方面，SNNwt必须模拟图像产生刺激的整个过程和相关的泄露反应过程，这里的泄露时间常数时间500ms；通过分

表 3.6 权值存储的SRAM相关参数

Design	$ni = 1$		$ni = 4$		$ni = 8$		$ni = 16$	
	SNN	MLP	SNN	MLP	SNN	MLP	SNN	MLP
SRAM width	128		128		128		128	
SRAM depth	784		200		128		128	
Read Energy (pJ)	44.41		33.05		32.46		32.46	
Area (μm^2)	108351		46002		40772		40772	
# Banks	19	8	75	28	150	55	300	110
Total Energy (nJ)	0.84	0.31	2.48	0.93	4.87	1.79	9.74	3.56
Total Area (mm^2)	2.06	0.76	3.45	1.29	6.12	2.24	12.23	4.48

表 3.7 空间折叠实现的SNN和MLP的硬件结果

网络	神经元输入数目	面积 (无SRAM) (mm^2)	总面积 (mm^2)	延时 (ns)	能耗 (uJ)	单幅图周期数
SNNwot (28x28-300)	$ni = 1$	1.11	3.17	1.24	1.03	791
	$ni = 4$	1.89	5.34	1.48	0.68	203
	$ni = 8$	2.79	8.91	1.76	0.67	105
	$ni = 16$	4.10	16.33	1.84	0.70	56
	expanded①	26.79	46.06	3.17	0.03	3
SNNwt (28x28-300)	$ni = 1$	0.48	2.56	1.15	471.58	791*500
	$ni = 4$	0.84	4.36	1.11	315.33	203*500
	$ni = 8$	1.19	7.45	1.18	307.09	105*500
	$ni = 16$	1.74	14.25	1.84	325.69	56*500
	expanded①	19.62	38.89	2.61	214.70	500
MLP (28x28-100 -10)	$ni = 1$	0.29	1.05	2.24	0.38	882
	$ni = 4$	0.62	1.91	2.24	0.29	223
	$ni = 8$	1.02	3.26	2.25	0.30	113
	$ni = 16$	1.88	6.36	2.25	0.29	57
	expanded①	73.14	79.63	3.79	0.06	4

① 估计数值，基于运算符的综合结果。

解序列到1毫秒（相对来说已经是粗粒度），SNNwt需要500个步长才能完成整个过程（即，500时钟周期）。通过折叠，这个数目需要进一步乘以输入的数目（n_i），即大约为$\frac{784}{n_i} \times 500$。这样做的结果是，SNNwt在硬件开销上有竞争力，但是在时间开销显著大于SNNwot。因此，我们将重点放在SNNwot和MLP之间的比较。

3.4.3.3 比较

MLP vs. SNN 对于空间展开版本的实现，SNN的开销比MLP低数倍，但这个结论在空间折叠版本上已经不再成立。即使在输入端有一个相当大的处理并行度，例如$n_i = 16$，空间折叠实现的MLP比空间折叠实现的SNNwot面积开销低2.57x（与空间折叠实现的SNNwt相比会更节省）。这里出现这种差距的主要原因是SNN所需要的Synaptic存储。在SNN中需要更多的神经元才能达到到最佳可能的精确度（然而精度仍然比MLP显著低），于是，突触权值的总数量显著增大（MLP的权重数量为$784 \times 100 + 100 \times 10 = 79400$，即隐层中的$784 \times 100$和输出层中的$100 \times 10$，SNN的权重数为输出层的与$784 \times 300 = 235200$）;作为一个参考，我们在表3.6中列出了对于这两个MLP和SNN的不同n_i值使用128位的SRAM bank时总的SRAM的存储成本。从能耗的数据看，MLP的高效打了几分折扣，这是因为乘法器的较高的成本和复杂性都会增加延迟和各个操作符的功耗；但是仍然，MLP相较于SNNwot来讲节约能耗2.41x（$n_i = 16$），甚至是2.71x（$n_i = 1$，因为乘法器的数量较少）。

总体而言，可以观察到，空间折叠MLP相对于SNNS加速器来说在成本、能耗及精度上是具有吸引力的。正如在章节3.3.2中所述，SNN+BP可以显著的跨越精度之间的鸿沟，但是不完全弥补，这使得MLP+BP仍然是一个更好的选择。然而，有一个领域，其中SNN+STDP相比于MLP+BP更具吸引力：这就是*在线学习*,学习过程中允许神经网络学习同时使用，这个特性对某些应用程序或许会很有用。在接下来的章节中，我们评估在硬件实施STDP的开销。

表 3.8 相对于GPU的加速比和能耗收益

		GPU	$ni = 1$	$ni = 16$	空间展开
Speedup	SNNwot	1	59.10	543.43	6086.46
	SNNwt	1	0.12	1.14	44.60
	MLP	1	40.44	626.03.04	5409.63
Energy Benefit	SNNwot	1	2799.72	4132.53	31542.31
	SNNwt	1	6.15	8.90	13.51
	MLP	1	12743.14	16365.61	79151.75

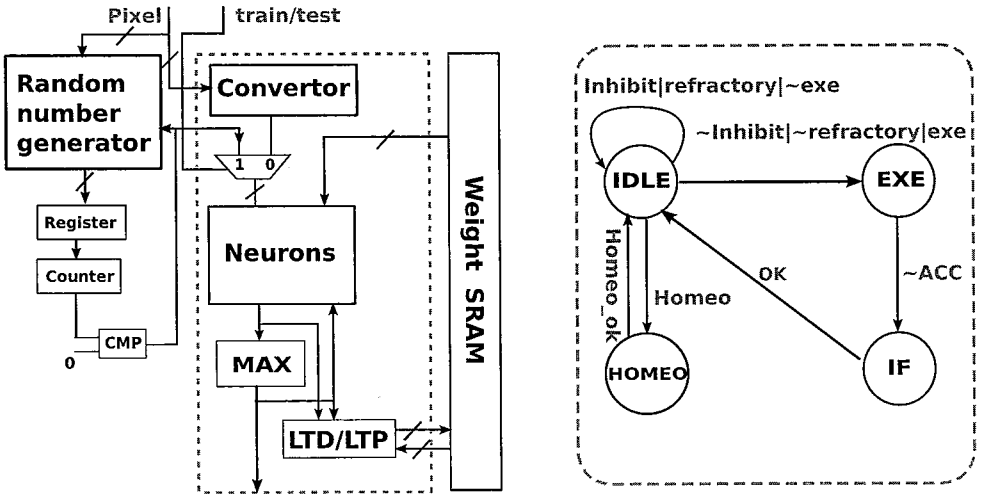

图 3.12 SNN的在线学习模块

MLP & SNN accelerators vs. GPU 虽然本章节研究的焦点是要理解哪些神经网络加速器方法是比较高效的，我们仍然提供相应算法在GPU上执行的软件对比作为参考。我们使用CUDA实现两个最高效算法的加速器版本（MLP和SNNwot），以及为了确保我们实现高质量的代码，我们使用了CUDA BLAS库（CUBLAS） [146]，特别是*sgmv*算法 [70,141]。我们使用最新的NVIDIA GPU K20M进行试验并在表3.8中比较不同n_i下加速器相对于GPU（对应于$n_i = 1$）的加速比。即便是$n_i = 1$，加速器所获得加速比也是显著地，这主要是由于三个原因：从global memory获取数据传输到计算单元时间；目标计算缺乏重用；数据结构的尺寸小（100到300元，784输入） [70,141]。

3.4.4 在线学习

神经元级别的STDP电路是通过一个简单的有限状态机来管理不同的信息，参见图3.12.(b)。首先，它记录了对于每一个神经元自从上次激发脉冲经过的时间；这些信息将被用来实施LTP和LTD，见章节2.1.2.1。同时，它还通过计数器管理耐火期和抑制期。每次神经元激发脉冲后耐火计数器就会被复位；同时当有任何一个神经元激发脉冲后抑制计数器也被复位，同时发出一个抑制信号；当任一计数器有效的时候，神经元忽略输入脉冲并且不改变它自身的神经元电位。为了实现LTP和LTD，神经元在内部也包含了计数器，每次激发脉冲后会被复位。在下次激发脉冲事件，它比较内部计数器和LTP延迟，并且如果内部计数器是更小或相等，相应的权重增加（LTP）一个恒定的增量（本文中为1），反之类似的相应权重降低（LTD）。

这样的结果是，由神经电路来确定哪些突触权重增加哪些突触权重减少;它不断的对突触权重进行更改（递增/递减恒定量，本文中也即1）。如章节2.1.2.1中所述，我们的泄露模型采用下面的表达式$v_j(T_2) = v_j(T_1) \times e^{-\frac{T_2-T_1}{T_{leak}}}$。在硬件实现上，我们采用分段线性插值来近似，该方法在硬件上极其高效。

表 3.9 硬件特性（SNN+在线学习）

Type		Area (no SRAM) (mm^2)	Total Area (mm^2)	Delay (ns)	Energy (mJ)
SNN	ni=1	2.55	4.92	1.23	0.71
	ni=4	3.33	7.10	1.48	0.37
	ni=8	4.26	10.70	1.81	0.32
	ni=16	6.44	19.06	1.88	0.33

自稳态神经元也记录了在一个稳态时期内激发脉冲的次数（恒定的时间周期，在此时间周期后，所有神经元调节激发阈值；这里我们用1500000ms，即每3000图像调整阈值一次）。一旦自稳态周期结束（它是由一个外部计数器管理的，所有的神经元共享该计数器），每个神经的激发脉冲的次数和预设的3ΔΦ<相比来调整神经元的放电阈值，如同章节2.1.2.1所述。

3.4.4.1 STDP硬件开销

在表 3.9中，我们列出了对不同n_i值带STDP的SNNwt神经网络的版图结果的参数。我们可以观察到的总面积（含SRAM）比单独的神经元SNNwt面积大1.34×（n_i = 16）到1.93×（n_i = 1），如表 3.7中所述。单周期的时间最多增加了7%，能耗开销与SNNwt神经元电路相比约为1.02×（n_i = 16）到1.50×（n_i = 1）倍。因此，可以观察到实施STDP的硬件开销是相当小的。因此，实施SNN+STDP加速器似乎是在任务具体要求永久（在线）学习上具有真正的价值。

3.4.5 在其他工作负载上进行验证

我们以MNIST基准测试集作为例来进行研究，选用MNIST是因为它是机器学习和神经生物科学领域两个领域研究人员少有的共同使用的基准测试集之一。此外，我们也想在其他的工作负载上验证我们的结论。我们选择另外两种重要的领域内的应

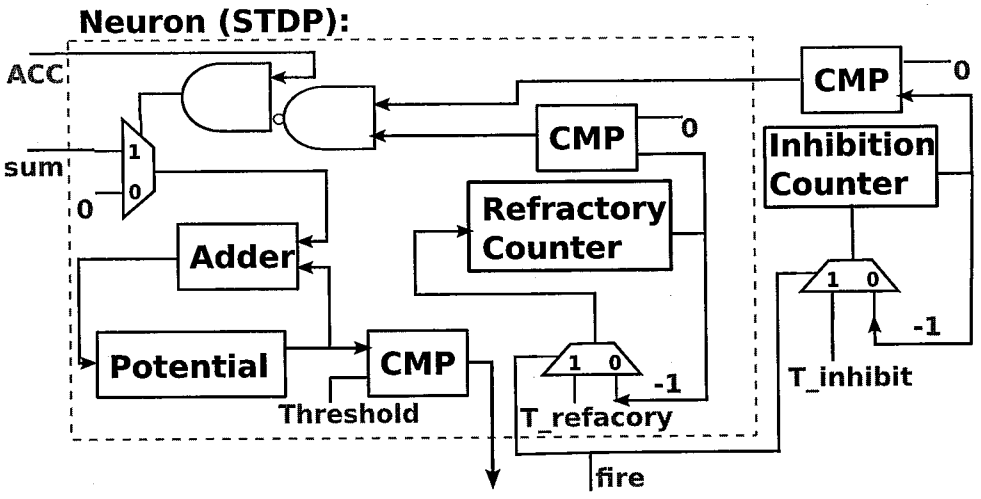

图 3.13 硬件实现STDP

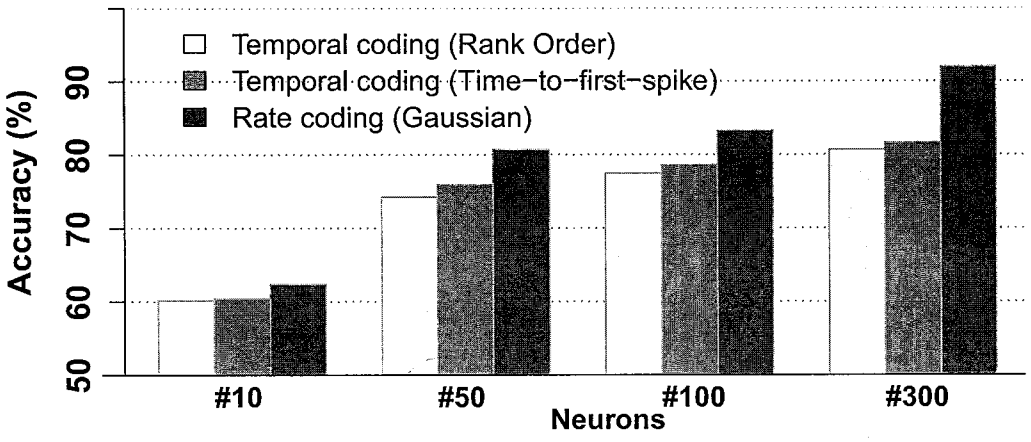

图 3.14 SNNs中不同得编码方式

用：语音识别，即MPEG-7 CE Shape-1 Part-B [4]基准测试集和口语阿拉伯数字口语识别（SAD）数据集[9]。

对于这两个工作负荷，我们进行了和MNIST完全相同的参数搜索来确定SNN和MLP的相关参数，即，确定了每个算法模型的超参数下的最优解，尤其是神经元个数，然后进行硬件实现确定其硬件成本。

对于MPEG-7中，优化的MLP（28x28-15-10）和SNN（28x28-90）能够分别实现99.7%和92%精度。当n_i从1变化到16时，在空间折叠设计上SNNwot相比MLP消耗3.81x-5.57x更多的面积开销，3.20x-5.08x更多的能耗开销。

对于SAD，优化的MLP（13x13-60-10）和SNN（13x13-90）能够分别实现91.35%和74.7%的精度。当n_i从1变化到16时，在空间折叠设计上SNNwot相比于MLP消耗1.27x-1.31x更多的面积开销，1.24x-1.26X更多的能耗开销。

总体概括地来看，这些基准测试上得到的结果与在MNIST得到的结果一致：SNN实现精度比MLP低，开销比MLP大。

3.5 讨论

编码方式 除了前面研究中所使用的Rate Coding编码方式，我们对SNN还尝试了其他两种不同Temporal Coding编码方式，Rank order Coding [188]和 time-to-first-spike coding [73,163]。Rate coding [68,176]和temporal coding [22]的编码方式已经被证明在生物上是合理且存在的，我们观察到在MNIST上SNN采用Temporal Coidng编码方式获得的精度比Rate Coding编码方式要显著的降低（82.14% vs. 91.82%），参见图 3.14，图中的结果来自采用相同网络拓扑结构的SNN。出于这样的原因，本章节中我们只讨论Rate Coding编码方式。

TrueNorth TrueNorth是一种多核心脉冲神经元数字芯片，由IBM [133]推出。而最近的文章 [133]中IBM并没有报告TrueNorth在机器学习基准测试集上准确性，如在MNIST、MPEG-7和SAD上等等，但是我们能够从他们以前发表的文章中提取出相关信息。例如，根据MEROLLA等人 [132]，一个TrueNorth核芯（具有相同的架构但规模大4倍 [133]）包含1024个输入，256个输出神经元，和1024 × 256个突触连接（9位精度），在采用IBM 45nm工艺下面积开销为4.2mm^2，频率为11MHz④，可以在MNIST实现89%的精度[60,133]。在这里，我们比较TrueNorth与SNNwot空间折叠实现（$ni = 1$），一个空间折叠实现的SNNwot具有和与TrueNorth核心加速器相似的特征（例如，在相同的时间的所有输出神经元都处理一个输入，见3.4.3.2）。由于IBM并未公布TrueNorth相关的资源，我们根据 [132]尽最大努力重新实现了TrueNorth的核芯并实现至后端版图（采用TSMC的65nm GPlus high VT标准库）。根据我们的重新实现，我们观察到SNNwot和TrueNorth相比面积更小（3.17 mm^2与3.30mm^2），速度更快（0.98us与1024us），能耗更低（1.03uJ与2.48uJ）和准确度更高（90.85%对89%）；然而，这样结果很可能是我们的重新实现并没有能够重现在文献中没有提到的IBM对TureNorth的优化。

3.6 相关工作

机器学习加速器不再赘述。对于神经生物学来说，很多研究并不使用MNIST全部的测试集，如所有的十个手写数字，所有的训练图像，所有的测试图像等等，使得我们选择算法进行比较变得比较困难。如，Netftci等人 [143]在MNIST使用RBM组成LIF神经元（824可见神经元和500隐藏神经元）报告了91.90%的精度（5-bit突触权值精度为89.40%），他们提出了一种事件驱动的对比差异采样的方法（contrastive divergence sampling）来训练STDP突触权值。标准的对比差异加随机单元的精度为93.6%。Arthur等人 [8]采用LIF神经元模型加RBM布局（256可见神经元和484隐藏神经元）在软件上的到精度94%（有限精度的硬件神经核精度下降为89%）。其他的研究结果精度如96.8% [20]和91.64% [16]，然而这些结果并不能用来比较，因为对于MNIST测试集部分使用或是不同的处理，不同的权值更新规则，突触模型，脉冲神经元模型等等。Querlioz等人在MNIST上用基于忆阻器模型的SNN网络结果实现。Diehl等人 [48]实现了较Querlioz等人更好的精度结果（95% vs. 93.50%）但是却使用了更多的神经元（6400 vs. 3300）：400个神经元下，精度结果只有82.9%。

值得注意的是，只有极少的工作是关注神经生物学激励的神经网络算法和机器学习激励的神经网络算法直接比较。例如，Serre等人 [173]提出了HMAX，一个从大脑视觉皮层启发而来的神经网络，表现出了和已有的机器学习算法（非神经网

④ TrueNorth才采用1MHz的的物理频率，从而最大可能的脉冲频率可以变得低于1KHz的，从而和神经生物学上观察保持一致 [68,176]

络）相当的结果。更近一些的是，Masquelier等人 [130]提出了一个和HMAX相较的基于SNN的算法（类似SNNwot），并在物体识别的任务上表现出了具有竞争力的结果；Nere等人 [144]提出了更加类似生物的SNN+STDP算法（接近SNNwt），并且搭建了一个类似HMAX的神经网络，然而这个搭建这个网络的目标本身并不是和HMAX相比。Farabet等人 [64]在FPGA上比较了SNN和CNN的算法，他们得出SNN相较于CNN具有较少计算需求的事实，然而他们对SNN和CNN都只进行了有限的硬件设计探索（空间展开设计）。Rast等人 [161]提出了不同的方案，他们将MLP映射在为SNN所设计的结构上（SpiNNaker [94]），然而这与两种算法硬件结果的比较并不相关，这只是提出了另外一种在硬件上实现MLP的方法。

3.7 本章总结

本章中我们主要探讨了对于两类基本上独立领域发展而来的神经网络算法作为加速器设计的比较，根本动机在于对目前持续增长的对于识别类任务的关注。

为了帮助理解两种不同算法的特性及哪种算法在目前的情况下更适合作为加速器硬件化的首选，我们比较了两个领域最广为人知的算法在MNIST基准测试集上的表现，并且进一步在物体识别和语音识别应用上验证我们所得到的结论。

当然我们的研究本身也有其局限性：都是当前的生物神经网络和机器学习神经网络和当前的硬件制造工艺。在这样的情况下，我们尽最大的努力并得到以下结论。尽管神经生物学激励来的算法更加接近生物上的事实和观察到的生物现象，我们注意到相应的几个平方毫米的硬件实现相较于机器学习激励来的经典的分类算法并不具有可比性，包括精度，面积，延时和能耗。

只有对于非常大型的应用，SNN应该在硬件参数上（面积、延时和能耗）上具有优势，然而精度依旧不如机器学习激励的神经网络算法。

我们也确认目前在SNN+STDP和MLP+BP之间的精度鸿沟是因为学习算法STDP本身和脉冲编码方式本身。

第四章　卷积神经网络加速器

4.1　简介

在过去的几年间，加速器因为其高能效性和高性能成为CPUs和GPUs的替代解决方案而越来越受到关注 [59,61,76,187,194,200]。过去一直认为，加速器的主要缺陷在于其有限的应用范围，但是最近工业界和学术界的研究都表明程序越来越向识别、数据挖掘和综合类收敛 [117]。同时存在这样的小部分的算法集合，也即基于神经网络的算法，能够实现很大部分上述的应用范畴内的任务 [42,85,136]。这使得设计出两全其美加速器成为可能：加速器既具有高性能和高效能也能够具有广阔的应用范畴。Chen等人 [29]利用这个事实并设计提出了神经网络加速器DianNao；然而，作者也同时在文章中确认，如同多数处理器结构一样，他们所设计的加速器的效率和可扩展性仍然受到内存带宽的严格约束。

本章研究的目标是目前表现最好的神经网络：*卷积神经网络（Convolutional Neural Networks，CNNs）* [109]和*深度神经网络（Deep Neural Networks (DNNs)）* [106,164]。两种神经网络目前都非常流行，同时DNNs也比CNNs更具有一般性：在CNNs中，在同一个特征图像中所有的神经元**共享**连接的权重，这使得在CNNs中的权重的总数目远小于DNNs中的权重数目。例如，目前较大的CNN有6千万权重 [100]，而最大的DNNs中则有10亿权重 [108]甚至100亿权重数目 [37]。CNNs中权重共享的特性则是直接源自于采用CNNs最多的应用，也就是图像识别类应用（输入为图像）：CNNs的神经元连接对应的权重集中抽取了神经元应该识别出的特征，而共享权重则是去获取图像中可能出现在任何地方的特征 [109]（如图像的平移不变性）的一个简单的方法。

这个简单的特性对结构设计产生了深刻的影响：众所周知，相较于计算而言，目前大部分能耗消耗与数据移动相关，特别是和内存的数据交互 [76,92]。权重存储比较小，则有可能将整个CNN存储在放置在紧挨着计算单元的片上的SRAM存储，即再也不需要通过内存DRAM去获取对应的每个输入的权值。最后剩下需要进行内存DRAM读取的数据则是输入图像。不幸的是，如果输入图像过大，这就不可避免的去通过内存DRAM去获取，从而使得这部分的功耗开销有可能仍然是是十分大的。

然而，CNNs多用于图像类应用，此类应用占据了识别类应用最大比重（其次是语音类）。在许多真实世界和嵌入式系统应用中，如智能手机、安保、自动驾驶汽车等，输入图像直接来自CMOS或者CCD传感器。在一个典型的图像设备中，图像由CMOS或者CCD图像传感器获得，然后传送至内存DRAM，而后利用CPUs和GPUs对从内存获取的图像数据进行后续处理，比如识别。CNN加速器具有较小的开销（计算

单元和SRAM存储权值），从而使得将CNN加速器放置在传感器旁边变得可能；同时只将识别后（通常为图像类处理）的结果，也即很少量的数据，传送到内存DRAM或者主处理器，这使得加速器几乎消除了全部和内存DRAM之间的数据交换，也即基本消除了全部相应的能耗开销。

在本章节中，我们提出了一个高能效的视觉识别加速器，名为ShiDianNao，可直接嵌入任何CMOS或CCD传感器，同时能够实时处理图像。我们的加速器利用CNN算法的特性，根据最终的实现结果，加速器比DianNao [29]节能54倍，这也是因为DianNao的定位是更广泛的神经网络算法。ShiDianNao的高效率不仅是因为消除DRAM存取（缓冲区计算，In-buffer Computing），也是因为通过仔细的最小化各个处理单元（处理单元的拓扑为半环拓扑二维处理阵列）之间的数据移动操作和传感器和SRAM存储有CNN的数据的SRAM片上存储之间的数据移动操作。在65 *nm*CMOS工艺下实现单核ShiDianNao到后端版图级结果，并将其扩展到一个多核心设计（ShiDianNao+）以实现更高的性能。单核心ShiDianNao的峰值性能是194 GOP/s（每秒十亿次定点运算），面积为5.94 mm^2，功耗为336.51 mW，运行频率为1 GHz，能够以11帧的速率处理640×480分辨的图片，扩展的多核版本ShiDianNao+（4核心）进一步实现了896 GOP/s的峰值性能，面积为8.50 mm^2，功耗为1440 mW，并能够以56帧速率在频率为1 GHz处理1280×720的图像。以上评估ShiDianNao和ShiDianNao+设计的结果是基于我们所选取的10个有代表性的CNN基准测试集（神经网络层）。我们相信，这样的加速器可以显著地降低复杂的视觉处理的硬件和能耗成本，从而有助于使他们广泛应用到更多的平台。

本章节其他部分安排如下：章节4.2中我们介绍了可能的系统集成方案；章节4.3中我们比较了不同的硬件映射的设计原则并着重分析了我们最终选择ShiDianNao方案的原因；章节4.4中我们考虑ShiDianNao加速器中间的计算部分，也即神经网络功能运算部件和算数逻辑运算单元；章节4.5中我们介绍了ShiDianNao加速器中的存储部分；章节4.6中我们介绍ShiDianNao加速器中间的控制模块，包括缓存控制器和指令控制器；章节4.7中我们给出完成CNN映射的过程，并通过简单的例子说明计算流程；章节4.8中我们介绍我们评估ShiDianNao加速器的实验设置；章节4.9中我们给出评估结果；章节4.10中即介绍了为了性能提升所完成的ShiDianNao+，ShiDianNao的多核扩展；章节4.11中我们给出相关工作；我们在章节4.12中给出本章总结。

4.2 系统集成方案

图 4.1显示了一个和常见相机整合的典型解决方案（非常类似组装STM芯片组[182,183]）：图像处理芯片连接到摄像机头（典型的智能手机中有两个摄像头），通过标准协议（*Camera Serial Interfaces (CSIs)*）传输数据。更先进的处理器已经包含有初步对象检测和跟踪功能，如人脸识别 [183]。图 4.1还显示了我们ShiDianNao加速器的大

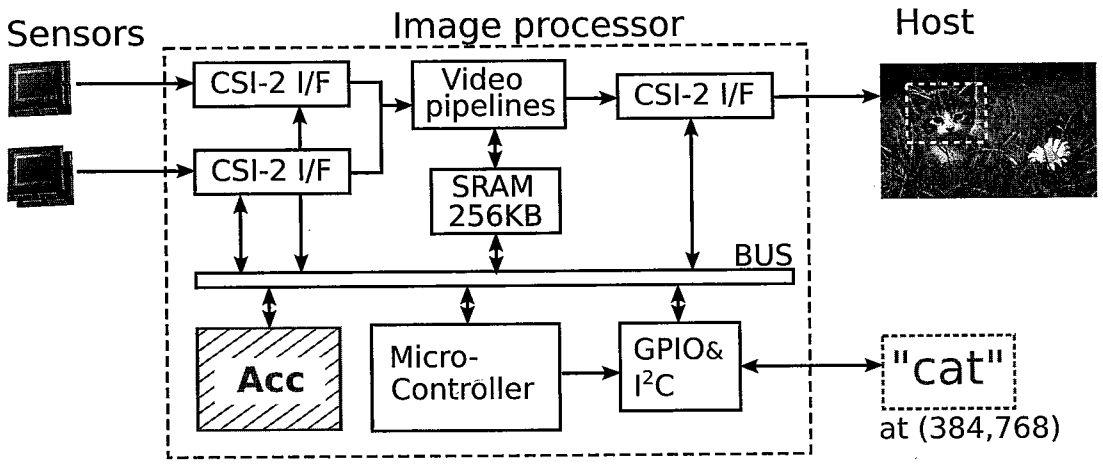

图 4.1 在商业的图像处理芯片上集成我们加速器的可能方案

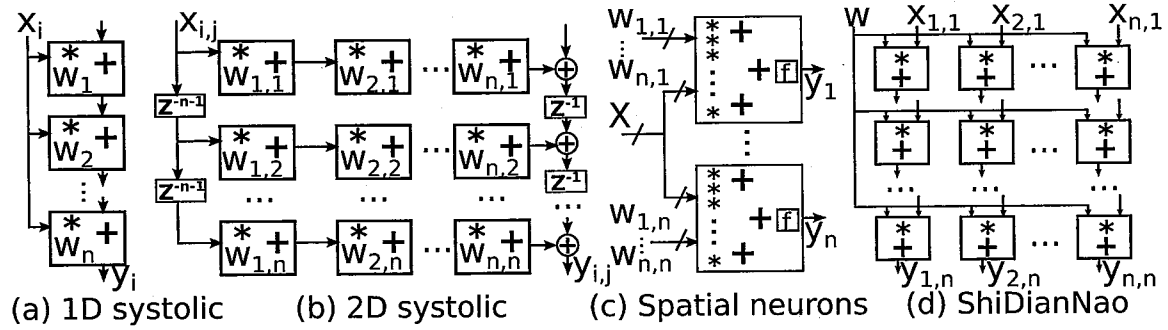

图 4.2 神经网络的典型设计方案

致设置：通过来自嵌入式微控制器的高级别控制信号进行控制，使用图像输入相同的存储缓冲器。不同于与这些视频流水线中的相当基本的操作，我们的加速器目标在于实现显著的更加高级的分类任务。应该注意到，为了控制成本和能耗开销，这种类型的商业图像的处理器技术发展了很久才避免缓冲整幅图像（对于800万像素的图像需要数MB的存储）：输入和系统的输出是通过串口通道，和外部DRAM存储没有任何接口，片上SRAM存储的容量是非常有限的（例如，256 KB [182]）。这些限制是我们在设计加速器中需要必须考虑的：我们的识别系统必须避免全帧存储，排除任何外部DRAM接口，串行地处理整幅图像中的一部分，这个过程如同数据从传感器到应用处理器一样。

4.3 设计原则

大致来说，一个纯粹的空间展开硬件实现的神经网络对于每个神经元将需要一个单独的累积单元，对于每个突触则需要单独的乘法器。在上世纪80年代和90年代的神经网络发展的早期，体系结构研究人员曾想到，一些大型应用将包含太多的神经元和突触，从而不容易将其实施成为简单的又深又宽的流水线。即使晶体管密度的发展已经取得了惊人的进展，然而目前实用的CNNs的大小显然仍然超过单纯的空间展开实施方案的能力 [187]，这驱使着体系结构研究人员寻找将CNN进行空间划分并且顺序映射

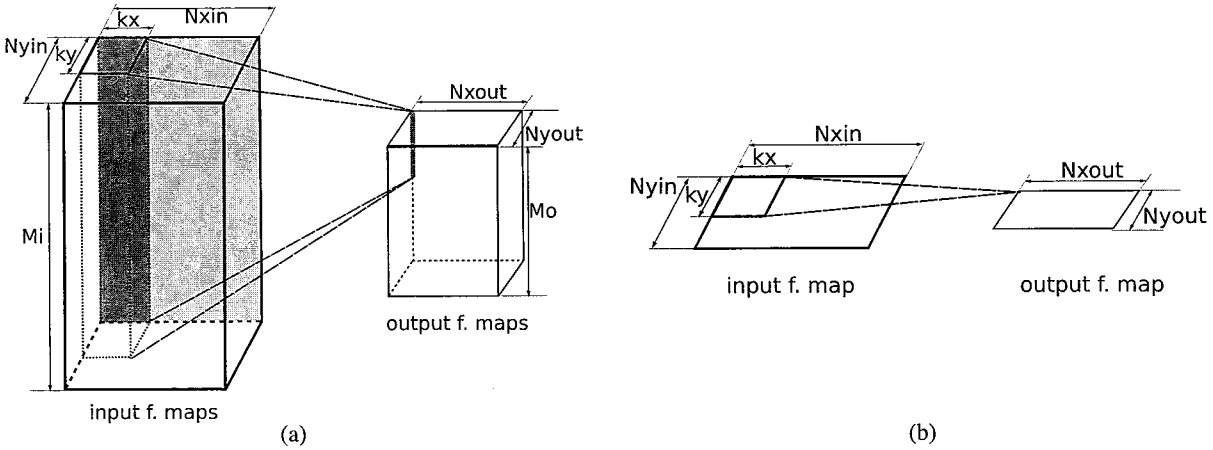

图 4.3　DianNao中卷积层的映射原则 [29]

到物理结构进行计算的方案。

不同的映射策略已经被尝试和研究（见Ienne等人早期的工作 [86]）：产品、原型、文献中已经很可能尝试了所有设计的可能性，包括每个神经元映射成为一个处理单元，每个synapes映射到一个处理单元，并采用systolic类似的方式传递kernels和输入特征图像这样的方案。其中的一些主要设计方案在图 4.2中展示。在本章节的工作中，我们很顺理成章地决定利用（1）我们需要处理数据（图像）的2D性质和（2）卷积核kernels的有限尺寸。

图 4.2展示出了四种可能最优的映射策略。在图 4.2 （a）中的第一个策略是*1D systolic*映射：然而它不能有效的利用2D图像数据，因为它无法利用二维卷积所有可能存在的数据重用。图 4.2 （b）中的第二个策略是*2D systolic*映射：尽管这种策略看来天然适合2D图像，但是它不能灵活地支持不同大小的卷积内核和其它CNN层如采样层和归一化层。在图 4.2 （c）中第三个策略已被最近的几项工作所采用，其中神经元被映射到硬件实现，这种映射通过在空间上扩展的方法 [187]或着通过空间折叠的方法[29]（也即DianNao）。这种策略编排所有的计算成为向量操作从而可以由多个矢量处理单元进行有效的处理。然而，它没有充分考虑了CNN性质，例如，各种内核/输入特征图像的大小，数据复用和复杂的输入数据的要求，从而导致对某些CNNs网络硬件资源有着显著的浪费。这里举一个说明性的例子，我们考虑一个目前流行的神经网络加速器DianNao中卷积层的映射策略，其中包含16个乘法-加法矢量处理单元。每个处理单元独立的处理不同输出特征图像上具有相同位置的输出神经元，使用共享神经元输入和单独的突触权重：输入是来自不同的输入特性图相同的位置的神经元，见图 4.3 （a）。然而，一旦特征图像的数目和矢量处理单元的数目的大小不完全匹配时，处理单元中的部分乘法器和加法器将被浪费。一种极端的情况是卷积层仅包含一个输入和一个输出特征图（见图 4.3 （B））。在这种情况下，在DianNao只有一个处理单元能够获得有效的数据从而进行处理，而其余处理单元则是空闲的，因此，真正的性能只有峰值性能的1/16。注意，这里DianNao为能够有效地处理各种尺寸的内核已经优化了

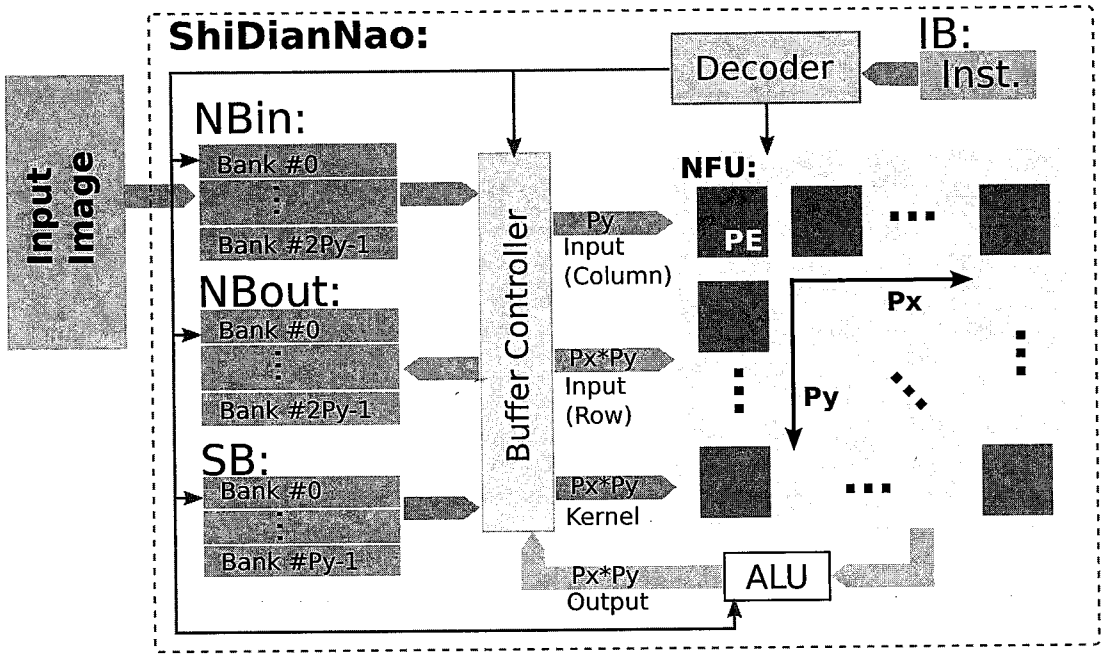

图 4.4 ShiDianNao整体结构框图

卷积层的映射策略。此外，在不考虑卷积层的2D性质和共享内核的重用，DianNao需要反复不断从片上存储的加载不同的内核和中间计算结果到处理单元，从而导致相当大的SRAM的功率消耗。例如，在DianNao中，SRAM的功耗的总片上功耗的62%。

总体而言，我们选择映射如图 4.2 （d）的策略：我们的处理单元（ i ）代表神经元，（ ii ）以二维网格的方式组织，（iii）接收，广播内核ω_{fhJ}，（iv）通过右-左和下-上的方式移动的输入特征图，最后（v）局部累计产生的输出特征图计算结果。

当然，映射的细节远远超出了图 4.2 （d）中的简单图示，我们在章节4.7中将致力于完整的展示如何将各个层和计算的阶段映射到我们的体系结构。然而，就目前而言，图 4.2的目的是给介绍的遵循我们架构的映射方法。

4.4 ShiDianNao结构：计算

正如图 4.4中所示的ShiDianNao的结构框图，ShiDianNao包含以下的主要组成模块：用于输入输出的两个缓存（NBin和NBout），用于权值存储的缓存（SB），神经功能单元（Neural Functional Unit，NFU），算术逻辑单元（Arithmetic Logic Unit, ALU），控制器（Decoder）和指令寄存器（IB）。在本章节剩下的部分，我们将首先介绍ShiDianNao中的计算部分，后面紧接着介绍存储和控制逻辑。

我们的加速器有两个功能单元，一个NFU用来处理基本的神经元计算（乘法，加法和比较操作），一个ALU用来计算非线性激活函数。我们在这两个计算模块中使用*16-bit*定点运算操作符而不是传统的32-bit浮点运算操作符。这样的原因首先是使用16-bit定点运算操作符对于网络精度的影响基本是可以忽略的，这点已经被以前的研究工作所证实 [29,51,187]。第二则是使用小的运算符能够显著的降低硬件开销，例

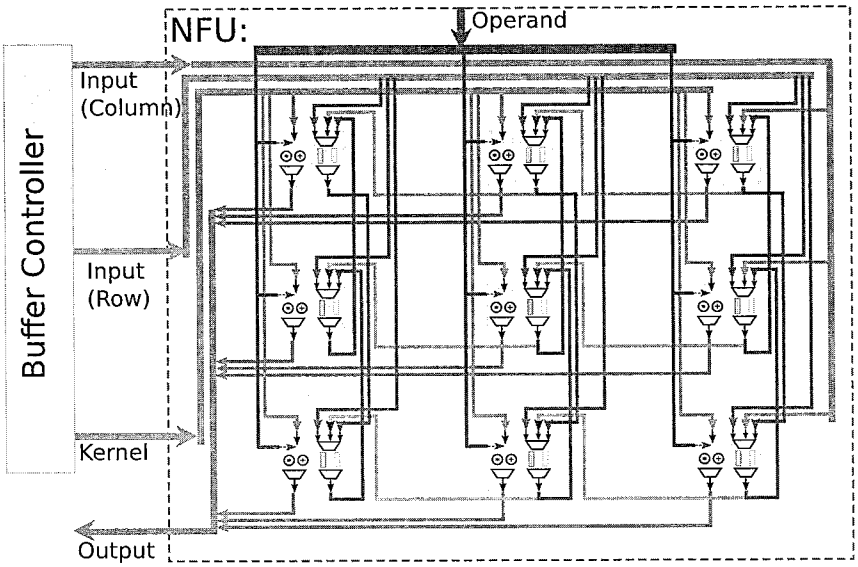

图 4.5　NFU结构图

如16-bit的定点乘法器在TSMC 65nm的工艺下比32-bit的浮点乘法器小6.10倍，功耗节约7.33倍 [29]。

4.4.1　神经功能单元（Neural Functional Unit，NFU）

由于我们的加速器主要处理二维数据（输入图像，特征图像），因此NFU也主要对处理二维数据（神经元/像素阵列）进行优化。DianNao [29]中的功能单元在这里并不十分高效，其主要原因为DianNao处理二维图像的方式与处理一维向量一样，并不能很好地利用场景中二位数据的局部性。与之相对的，本章节的加速器的功能单元NFU是一个大小为$P_x \times P_y$的二维处理单元阵列（Processing Elements，PE）（我们称之为半环拓扑二维处理单元阵列），为能够自然地处理二维数据的拓扑结构。

一个简单直觉性的处理神经元到PE映射的方案是把$K_x \times K_y$个PE划归成为一组来处理卷积窗口大小为$K_x \times K_y$的单个神经元，从而使得所有的$K_x \times K_y$个PE能够同时计算同一个神经元。此种映射方案有几方面缺陷：其一，需要复杂的逻辑（需要多个大型的MUX阵列）来使得不同的神经元共享数据；其二，为了高效的使用PE阵列，复杂度会进一步提升才能支持不同大小的卷积核。也因此，我们采用的高效的方案来替代：单个输出神经元被映射到单个PE，对于单个输出神经元（PE），特征图像输入或者输入图像串行输入PE用于计算相同的输出神经元，而不同的PE间则可以共享相同的输入。在图 4.5中，我们完整的展示整个NFU的结构。NFU能够分别从NBin/NBout和SB中同时读取输入神经元和权值然后分配到不同的PE中。另外，NFU中每个PF包含局部存储，用于在PE间传播输入神经元，具体参见章节 4.4.1中关于*PE内部数据传播*。在完成计算后，NFU从不同的PE单元中收集计算结果并把结果相应地送至NBout/NBin或者ALU。

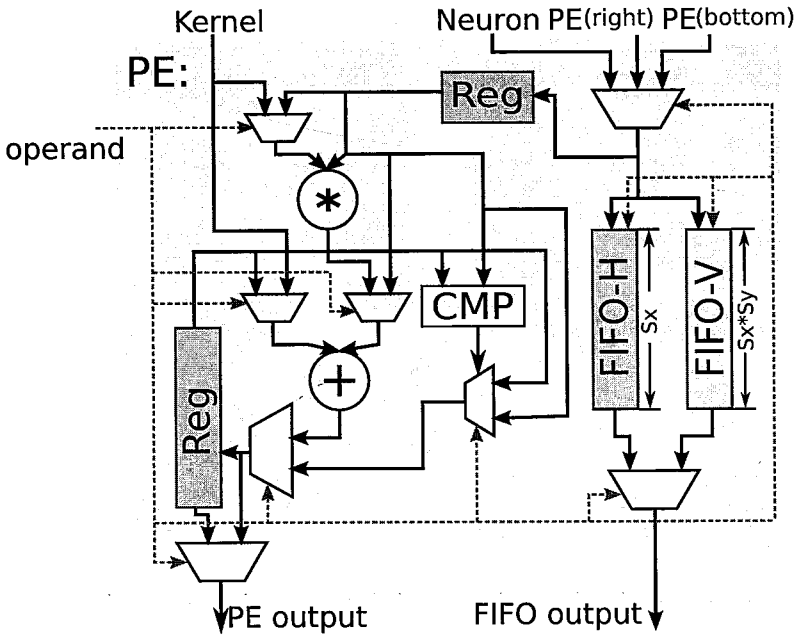

图 4.6 PE结构图

处理单元PE 在每个周期，每个PE能够完成卷积层、分类层或者归一化层运算中的一次乘法和一次加法，或是平均采样层中的一次加法，或是最大数采样层中的比较操作（参见图 4.6）。我们用$PE_{i,j}$来表示NFU中位于第i行，第j列的PE处理单元。$PE_{i,j}$拥有三个输入：一个用于接收控制信号；一个用于读取突触权值（如卷积层运算中的卷积核）；一个用于从$PE_{i+1,j}$（右方），或者$PE_{i,j+1}$（下方）读取输入神经元。$PE_{i,j}$有2个输出：一个用于将结果写回存储介质NBout/NBin；另一用于将本地存储的神经元的值传播至临近的PE处理单元，从而使得临近的PE单元能够高效的取得相应的数据（复用片上数据）。在计算CNN的一个层时，直到当前的输出神经元计算完成后，每个PE才会开始计算新的输出神经元（具体神经元到PE的映射方案参见章节 4.7）。

PE间数据传播 在卷积神经网络CNN中的卷积层，采样层和归一化层中，每层中的输出神经元需要从输入图像内获得一个矩形窗口内的神经元。对于相邻的输出神经元，这样的窗口一般有比较大的重叠（参见章节 4.7）。虽然所有所需的数据都可从NBin/NBout获得，然而从缓冲区到不同的PE反复的读取需要很高的带宽。我们使用LeNet-5 [109]中具有代表性的卷积层（32×32个输入特征图像，5×5的卷积内核）来估计的片上存储和NFU的内部带宽要求（参见图 4.7）。我们观察到，对于只有25只个PE的NFU需要大于52 GB/s的带宽。如此大的带宽的需求可能导致连线开销过大，或显著的性能损失（如果我们限制连线开销）。

为了支持高效的数据重用，我们允许PE内部进行数据传播，其中每个PE可以将本地存储的输入神经元传播给其左侧和上方的PE邻居。我们通过在每个PE中使用两个FIFO（水平和垂直：FIFO-H和FIFO-V）存储它接收到的输入值。FIFO-H缓冲器缓存

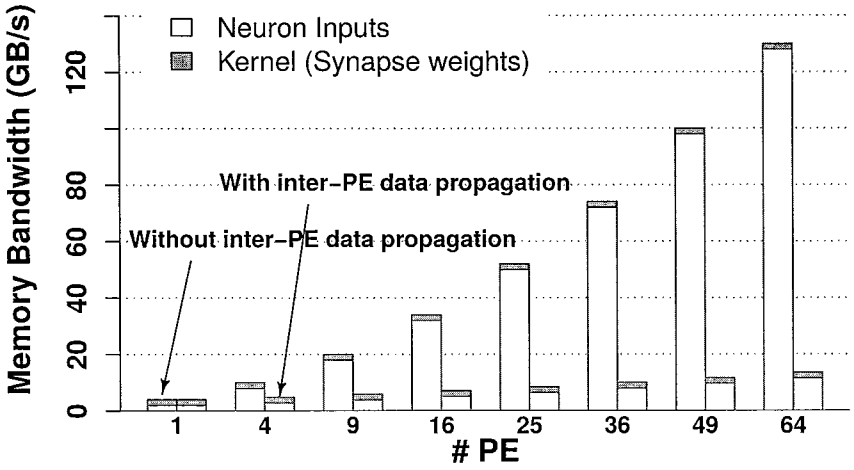

图 4.7　片上存储（输入神经元和突触权值）至NFU内部带宽需求

来自NBin/NBout，或是右侧相邻PE的数据；同时此数据也将传播到左侧相邻的PE以便重用。FIFO-V缓冲器缓存来此NBin/NBout，或是从上方相邻PE的数据；这样的数据将稍后传播到下方相邻的PE以便重用。使用PE内部数据传播，内部带宽要求可以显著降低（见图 4.7）。另外，在我们的半环拓扑二维处理单元阵列中，最下方的PE也能够数据传递给最上方的PE，构成环路；而水平方向则不能够如此构成环路传递数据。

4.4.2　算术逻辑运算单元 (ALU)

在加速器中，NFU的PE阵列并不完成CNN中所有运算，也因此，我们使用轻量级的算术逻辑运算单元ALU来完成其他剩下的计算。在ALU中，我们使用16-bit的定点算术逻辑运算单元，包括除法（用于平均采样层和归一化层），非线性激活函数如*tanh*()和*sigmoid*()（用于卷积层和采样层）。我们使用分段线性拟合的方法来实现激活函数（也即，$f(x) = a_i x + b_i$，其中$x \in [x_i, x_{i+1}]$，$i = 0, \ldots, 15$）。线性拟合的方案被证明对于CNNs的精度只有微乎其微的影响 [29,105]。分段拟合函数中的系数被事先存储在片上存储，从而使得该计算可以通过一个乘法器和一个加法器高效的完成。

4.5　ShiDianNao结构: 存储

我们使用片上SRAM同时存储CNN的所有数据（如突触权值）和CNN的所有指令。从体系结构来看，计算过程中所有需要的数据和计算产生的数据全部都存储在片上SRAM的缓冲区内，我们称之为缓冲区计算（In-buffer Computing）。从机器学习和体系结构的角度来看，最近的一些研究中使用有限的参数仍然使CNN具有很好的精度。这意味着5.36 KB–263.11 KB片上存储对于许多CNNs来讲已经足够将所有的数据存储在片上。我们使用了264 KB的片上SRAM存储，加速器已经能够将所有的数据存储在片上从而不去访问片外存储，同时也能保持高精度。在我们的加速器设计中，我们总共

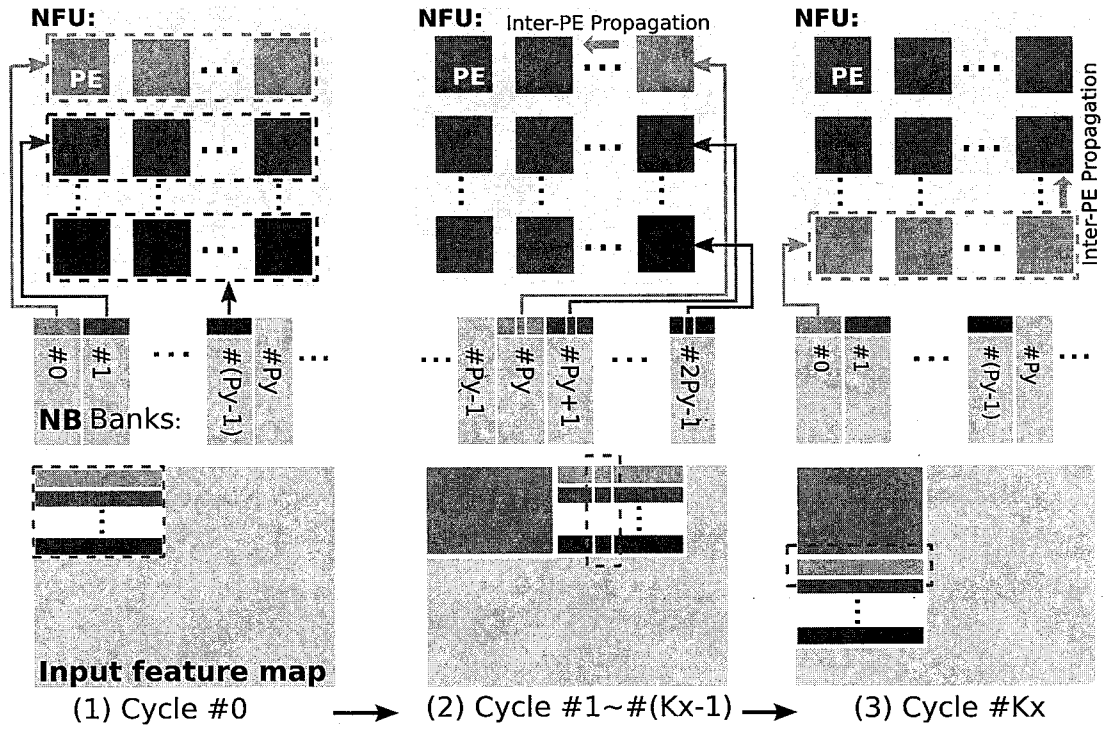

图 4.8 典型卷积层中的数据流（这里我们考虑最复杂的情况：卷积核的大小大于NFU的大小，即$K_x > P_x$且$K_y > P_y$）

使用了460 KB片上存储，这对于我们所选的10个实际应用中的CNNs（参见表 4.1）已经足够。460 KB的SRAM的开销是适中的：TSMC 65nm工艺下，面积为2.95 mm^2，每次读取数据的能耗为0.57 nJ。

更进一步的，我们将片上SRAM存储划分为独立的不同的缓存器以对应不同的数据（也即NBin，NBout，SB）。独立的缓存器可以允许我们对不同的数据类型选择不同的读取宽度，从而最优化每次读写的开销和延迟。特别的，NBin和NBout分别用来存储输入和输出神经元，只有当本层所有的输出神经元已经计算完成并且成为下

表 4.1 CNNs

CNN	最大层大小 (KB)	权值大小 (KB)	总存储开销 (KB)	精度 (%)
CNP [65]	15.19	28.17	56.38	97.00
MPCNN [140]	30.63	136.52	197.78	96.77
Face Recogn. [107]	21.33	61.14	103.8	96.20
LeNet-5 [109]	9.19	118.30	136.11	99.05
Simple conv. [175]	2.44	258.54	263.42	99.60
CFF [71]	7.00	1.72	18.49	—
NEO [142]	4.50	3.63	16.03	96.92
ConvNN [46]	45.00	4.35	87.53	96.73
Gabor [103]	2.00	0.82	5.36	87.50
Face align. [54]	15.63	29.27	56.39	—

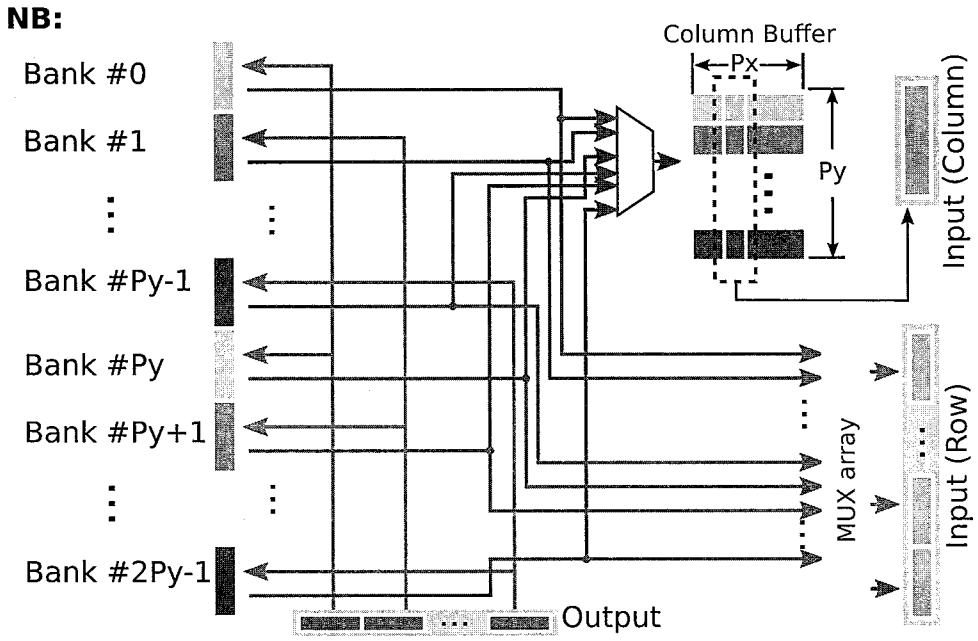

图 4.9 NB控制器结构

一层的输入神经元时，两者的功能才会交换，也即对于下一层NBout和NBin分别用来存储输入和输出。为了支持SRAM到PE的数据搬运，也为了支持PE内部数据传播，NBin和NBout都有$2 \times P_y$个banks（参见图 4.8）。每个bank的宽度为$P_x \times 2$ bytes。这其中，NBin和NBout都必须足够大，从而能够存下每一层的所有神经元数据。与之不同的，SB只有P_y个banks，用来存储CNN的所有突触权值。

4.6 ShiDianNao结构：控制

4.6.1 缓存控制器

片上缓存的控制器支持高效数据重用和在NFU中的计算。这里，我们详细介绍NB控制器（NBin和NBout共同使用）并以此为例来介绍所有的缓存控制器的设计相关，这里我们省略其他缓存控制器的详细介绍（和NB控制器类似但是更简单）。NB控制器的结构如图 4.9所示：该控制器能有效地支持6个读模式和一个写模式。

不失一般性，这里我们假设NBin存储了计算某一层的所有输入神经元，NBout用于存储当前层所有的输出神经元。注意，NBin和NBout都有$2 \times P_y$个banks且每个bank的宽度为$P_x \times 2$ bytes（即, P_x个16-bit神经元）。图 4.10说明了NB缓存控制器的6个读模式：

(a) 读取多个banks（#0到#$P_y - 1$）。

(b) 读取多个banks（#P_y到#$2P_y - 1$）。

(c) 读取一个bank。

(d) 读取一个神经元。

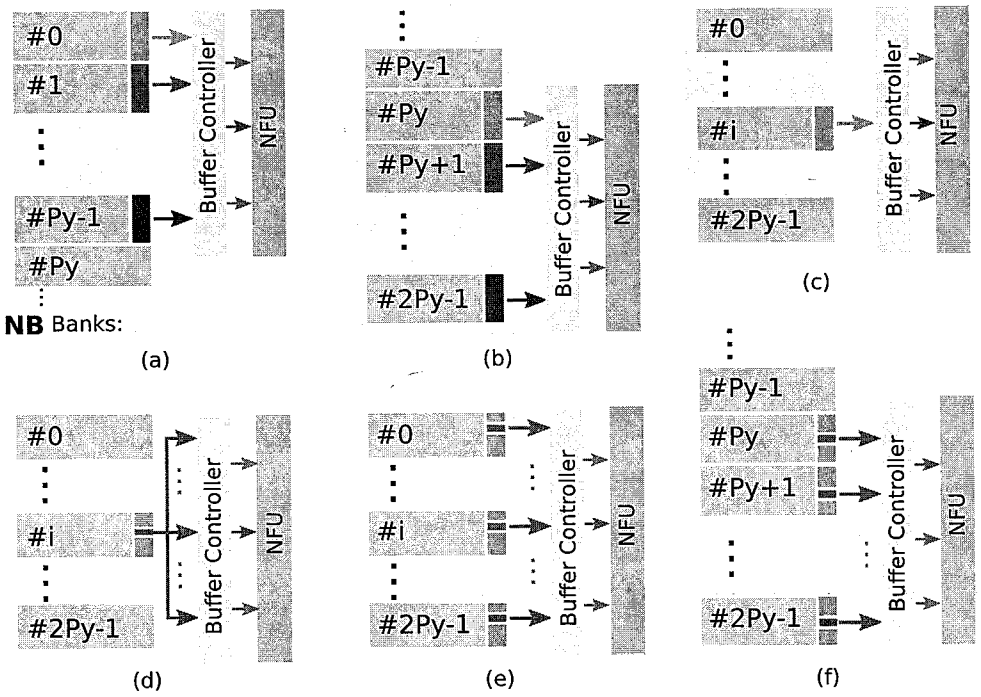

图 4.10 NB控制器所支持的读模式

(e) 读取给定步长的所有神经元。

(f) 读取每个bank中的单个神经元（#0到#P_y-1或者#P_y到#$2P_y-1$）。

对于不同的层，我们选取这6个读取模式的一个子集即可。对于卷积层，我们选取模式（a）（b）来从NBin的banks#0到#P_y-1或者#P_y到#$2P_y-1$读取$P_x \times P_y$个神经元（图4.8（1））；模式（e）来处理很少但是存在的一种情况，即卷积窗口在输入特征图像的滑动步长不为1；模式（c）从单个NBin bank读取P_x个神经元（图 4.8（3））；模式（f）从NB banks#P_y到#$2P_y-1$或者#0到#P_y-1中读取#P_y个神经元（图 4.8（2））。对于采样层，我们同样适用模式（a），（b），（c），（e）和（f），这是因为采样层有着类似卷积窗口的滑动采样窗口（对于输入神经元）。对于归一化层，我们仍然适用模式（a），（b），（c），（e）和（f），这是因为归一化层被分解成不同的类似卷积、采样的子层（具体参加章节 4.7）。对于分类层，我们使用模（d）来对所有的输出神经元读取同样的输入神经元。NB缓存控制器中的写回模式则相对直接简单了许多。对于计算一个卷积层，一旦PE将当前输出神经元的所有计算都已经完成，结果会暂时存储在NB缓存控制器中的一个寄存器序列（图 4.9中的*output*）。当所有$P_x \times P_y$个PE的结果都已经被缓存器收集，NB缓存器会把结果一次性的写入NBout。为了使得每层的映射保持一致，$P_x \times P_y$个输出神经元会按照有P_y行的数据块进行组织，其中每行的宽度为$P_x \times$ 2-byte（对应NB的一个bank的宽度）。当数据块中的输出神经元对应输出特征图像上的列$2kP_x, \ldots, ((2k+1)P_x - 1)$（$k = 0, 1, \ldots$），也即图 4.11中特征图像中的蓝色的列，数据块会被写回到NB的前P_y个bank。否则，即位于图 4.11中特征图像中的灰色的

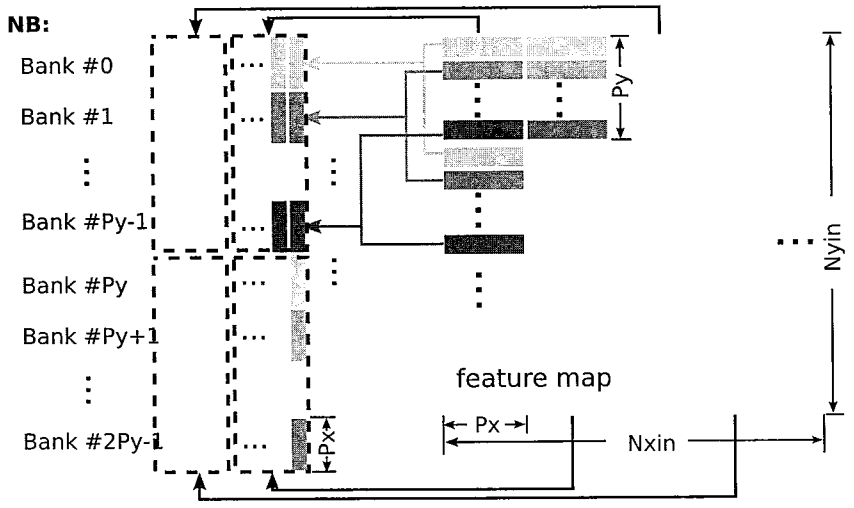

图 4.11　NB中的数据组织

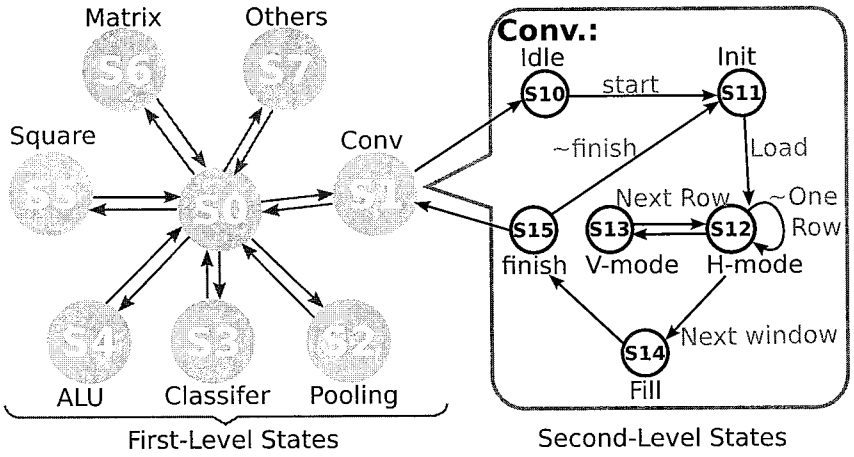

图 4.12　分级控制的有限状态机

列，数据块会被写回NB的后P_y个bank。

4.6.2　控制指令

我们使用控制指令来灵活的支持加速器计算CNN中的不同层。一个直觉性且直观的方法是每周期采用控制指令来直接控制加速器中的所有模块进行操作（每周期需要97-bit控制信号）。然而，64个PE的ShiDianNao加速器来计算一个典型的CNN可能需要多达5万个周期，将所有的指令都保存在片上这大概会需要600 KB（97 × 50K）的SRAM来仅仅存储指令，这对于嵌入式传感器是不现实的。

我们选择一个高效的替代方案，充分考虑了CNN各层的特性从而找到紧凑且精确的信号表示：我们定义了一个2层的分层控制有限状态机（Hierarchical Finite State Machine，（HFSM））来描述加速器中各层的执行控制逻辑（参见图 4.12）。在HFSM中，*第一层状态*描述了高度抽象的ShiDianNao中的处理任务，如CNN中不同的层，ALU的操作等。和第一层相结合，*第二层状态*则对应相应抽象任务的执行细节。例如，对应卷积操作（*Conv*）的第二层状态则对应一对输入输出特征图像的计算。这里我们不提

供所有状态的细节。总之，第一层状态和第二层状态的结合已足以应对CNN中不同任务（即不同层）的执行，这些任务的执行需要花费一定的周期数。另外，我们同样也可以根据HFSM的状态的跳转规则在执行过程中通过HFSM推测加速器下一步的执行任务。我们使用61-bit的指令来表示HFSM中的每一个状态和相关的参数（如特征图像大小），这61-bit的指令随后被解析成为针对每个模块详细的控制信号。采用这样的控制方法，在不失灵活性的基础上，实际应用中，前面所提到的5万个周期执行的CNN现在则只需要1 KB的指令存储模块和一个轻量级的解码器，这个解码器在65nm的工艺下只占用0.03 mm^2的面积开销（是前面提到的600 KB的SRAM面积开销的0.37%）。

4.7 CNN映射

在本章节中，我们将介绍如何将CNN中不同的层映射到我们的加速器设计上。

4.7.1 卷积层（Convolutional Layer）

卷积神经网中的卷积层中包含多个输出特征图和多个输入特征图。当执行卷积层时，加速器连续的进行的同一幅输出特征图像的计算，直到当前地输出特征图像已经完全计算完成才会移动到下一个输出特征图进行计算。当计算每个输出特征图时，加速器的每个PE连续的进行同一个输出神经元的计算，直到当前的神经元已经计算才会切换到另一个输出神经进行计算。

在图 4.13中，我们用一个简单的例子来说明同一幅输出特征图像上的不同神经元是如何在同一时间进行计算的。不失一般性的，我们采用一个有2×2个PEs（$PE_{0,0}$，$PE_{1,0}$, $PE_{0,1}$和$PE_{1,1}$）的结构来计算一个具有3×3卷积核、1×1卷积步长的卷积层作为例子。图 4.13中，我们只描述了前四个周期中数据的流动，这四个周期中的描述已经足够代表整个卷积层的计算。

Cycle #0: 所有的PE从NBbin中读取（模式（a））各自相关卷积窗口中的第一个输入神经元（也即$x_{0,0}$, $x_{1,0}$, $x_{0,1}$和$x_{1,1}$），从SB中读取各自相关的权值（也即$k_{0,0}$）。每个PE首先计算当前输入神经元和输入权值的乘积然后将当前计算结果存储在PE内部的寄存器当中，且每个PE将输入神经元存储在FIFO-H和FIFO-V中用于PE间的数据传播。

Cycle #1: $PE_{0,0}$和$PE_{0,1}$在本周期分别从$PE_{1,0}$和$PE_{1,1}$的FIFO-Hs中读取相应的输入（也即输入神经元$x_{1,0}$、$x_{1,1}$），即为水平方向的PE间的数据传播。$PE_{1,0}$和$PE_{1,1}$则从NBin中读取（模式（f））所需要的输入神经元（也即$x_{2,0}$，$x_{2,1}$），同样的，将输入神经元存储在FIFO-H和FIFO-V中用于PE间的数据传播。所有的PE此时共享从SB中读取的权值，也即$k_{1,0}$。

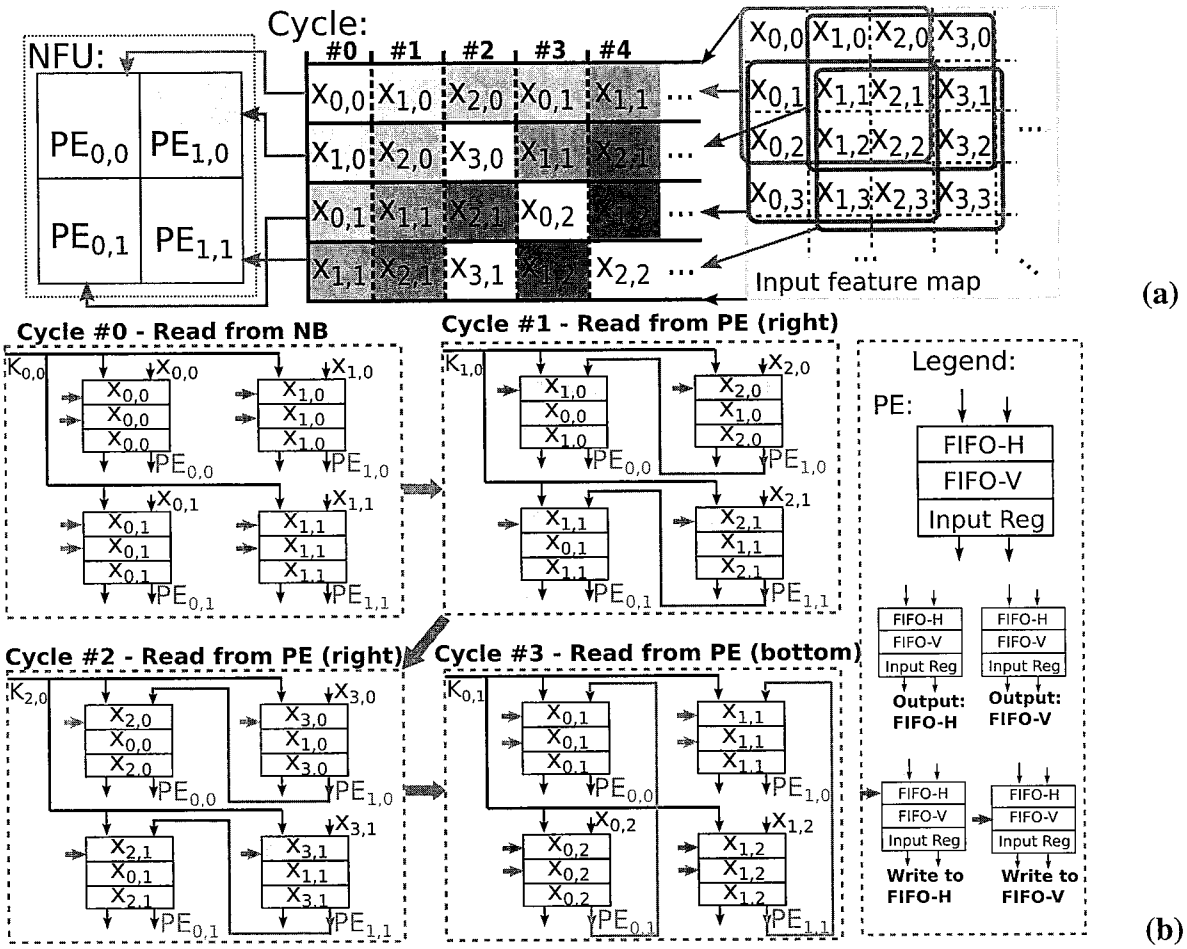

图 4.13　在NFU（2×2个PEs）上卷积层的硬件映射（卷积窗口：3×3；卷积步长：1×1）

Cycle #2: 和第一个周期类似，$PE_{0,0}$和$PE_{0,1}$在本周期分别从$PE_{1,0}$和$PE_{1,1}$的FIFO-Hs中读取相应的输入（也即输入神经元$x_{2,0}$、$x_{2,1}$），同样为水平方向的PE间数据传播。$PE_{1,0}$和$PE_{1,1}$则从NBin中读取（模式（f））所需要的输入神经元（也即$x_{3,0}$，$x_{3,1}$），将输入神经元存储在FIFO-H和FIFO-V中用于PE间的数据传播。所有的PE此时共享从SB中读取的权值，也即$k_{2,0}$。目前为止，所有的PE都已经处理了各自卷积窗口中第一行的计算，从下周期开始则开始计算卷积窗口的第二行。

Cycle #3: $PE_{0,0}$和$PE_{1,0}$在本周期分别从$PE_{0,1}$和$PE_{1,1}$的FIFO-Vs中读取相应的输入（也即输入神经元$x_{0,1}$、$x_{1,1}$），即为垂直方向的PE间的数据传播。$PE_{0,1}$和$PE_{1,1}$则从NBin中读取（模式（c））读取所需要的输入神经元（也即$x_{0,2}$，$x_{1,2}$），同样的，将输入神经元存储在FIFO-H和FIFO-V中用于PE间的数据传播。所有的PE此时共享从SB中读取的权值，也即$k_{0,1}$。

在上述的简单的例子中，对于4个输出神经元的计算中，PE间数据传播降低了44.4%的NBin的读取要求，从而有效的降低了NFU和NBin之间的带宽需求。在实际应用中，PE的数目和卷积核的大小通常要比例子中的大，因此相应的效率的提升也更

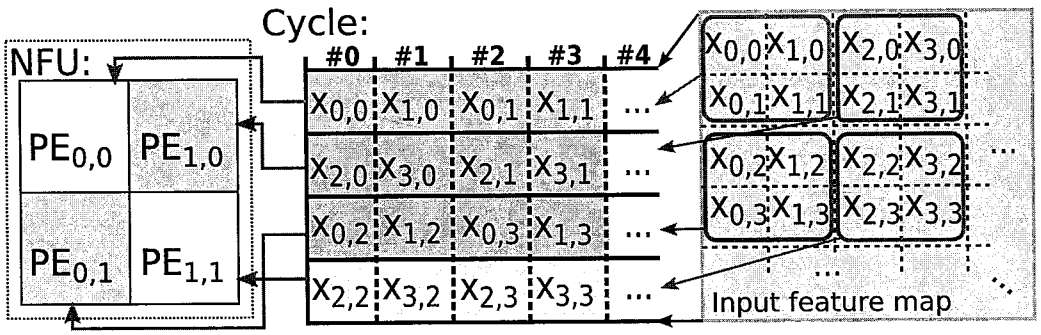

图 4.14 在NFU（2×2个PEs）上采样层的硬件映射（采样窗口：2×2；采样步长：2×2）

加显著。例如，在有64个PE的NFU加速器中LeNet-5中的典型卷积层C1（卷积核：5×5；卷积步长：1×1) [109]，PE间数据传播降低了73.88%的NFU和NBin之间的带宽需求，参见图 4.7。

4.7.2 采样层（Pooling Layer）

采样层通过最大数或者平局数采样的方法从输入特征图像构建输出特征图像。类似于卷积层，每个输出神经元是通过输入特征图像上的一个窗口内（采样窗口）的数据计算完成。当执行采样层时，加速器连续进行的当前输出特征图像的计算，直到当前的输出特征图像已经计算完成才会移动到下一个输出特征图进行计算。当计算每幅输出特征图时，加速器的每个PE连续的进行当前单个输出神经元的计算，直到当前的神经元已经完成计算才会切换到另一个输出神经元。

在典型的采样层中，相邻输出神经元的采样窗口相邻但是不重叠，也即采样步长和采样窗口一样大小。在图 4.14中，我们采用一个有2×2个PEs（$PE_{0,0}$，$PE_{1,0}$，$PE_{0,1}$和$PE_{1,1}$）的结构来计算一个具有2×2采样窗口、2×2采样步长的采样层作为例子来说明加速器执行采样层的流程。每个周期，每个PE每个神经元从NBin中读取（模式（e））一个采样窗口中的输入神经元（行优先，从左至右）用于计算。此时PE间不相互传播数据，这是因为PE相应的采样窗口不存在数据复用。

另外，也有相邻输出神经元的采样窗口相邻且重叠的情况，也即采样步长小于采样窗口大小。对于这样的情况，我们采用类似于卷积层的映射方法，不同的是此时采样层中没有权值。

4.7.3 分类层（Classifier Layer）

在CNN中，卷积层允许不同的输入-输出神经元之间的突触连接共享权值（卷积核）而采样中则不存在突触权值。与卷积层不同的是，分类的输入输出神经元之间是全连接的，也即在不同的输入输出神经元的突触连接不存在这样的共享的权值。这样也使得分类层通常具有较多的权值，需要较大的存储空间（例如，LeNet-5 [109]中分类层占总权值的97.5%）。在图 4.15中，我们展示分类层的一般映射规则。当计算一个分类层的时候，加速器的每个PE连续的进行当前单个输出神经元的计算，直到当前

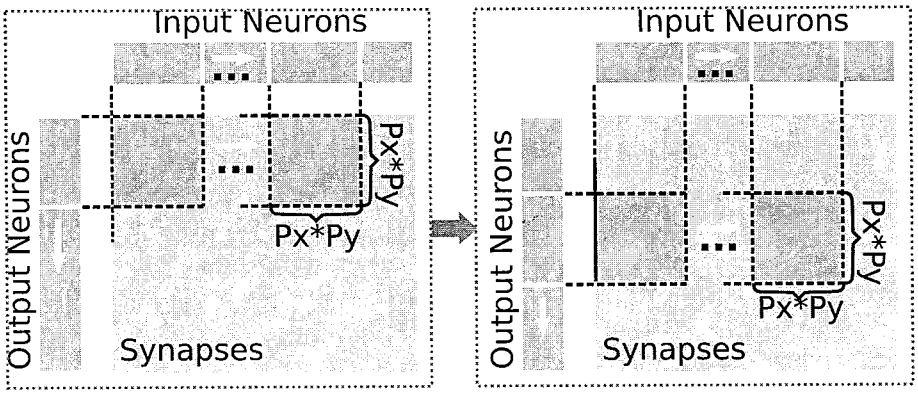

图 4.15　分类层的映射方法

的神经元已经完成计算才会切换到另一个输出神经元。每个周期中，对于NFU中所有的$P_x \times P_y$个PE来说，卷积层运算需要读一个突触权值和$P_x \times P_y$个不同的输入神经元，分类层需要$P_x \times P_y$个不同的权值和一个输入神经元。每个PE把当前的突触权值和输入神经元相乘并把结算结果累加存储在PE内部的存储中。在一定的周期数后，当与当前输出神经元相关的点乘计算（输入神经元和突触权值相乘）都已经完成后，结果会被送至算术逻辑单元ALU中用于计算激活函数。

4.7.4　归一化层（Normalization Layers）

为了使得归一化层也能在我们的加速器上运行，我们将归一化层分解成由成若干子层和基本的计算原语。在图 4.16和图 4.17中，我们详细说明了如何将不同的归一化层分解。LRN层分解成分类子层，一个矩阵元素平方，一个矩阵加法，指数函数和除法；LCN层分解成两个卷积的子层，一个采样子层，一个分类子层，二个矩阵加法，一个矩阵元素平方和除法。按以上分解，其中卷积，采样和分类子层可以与在前小节描

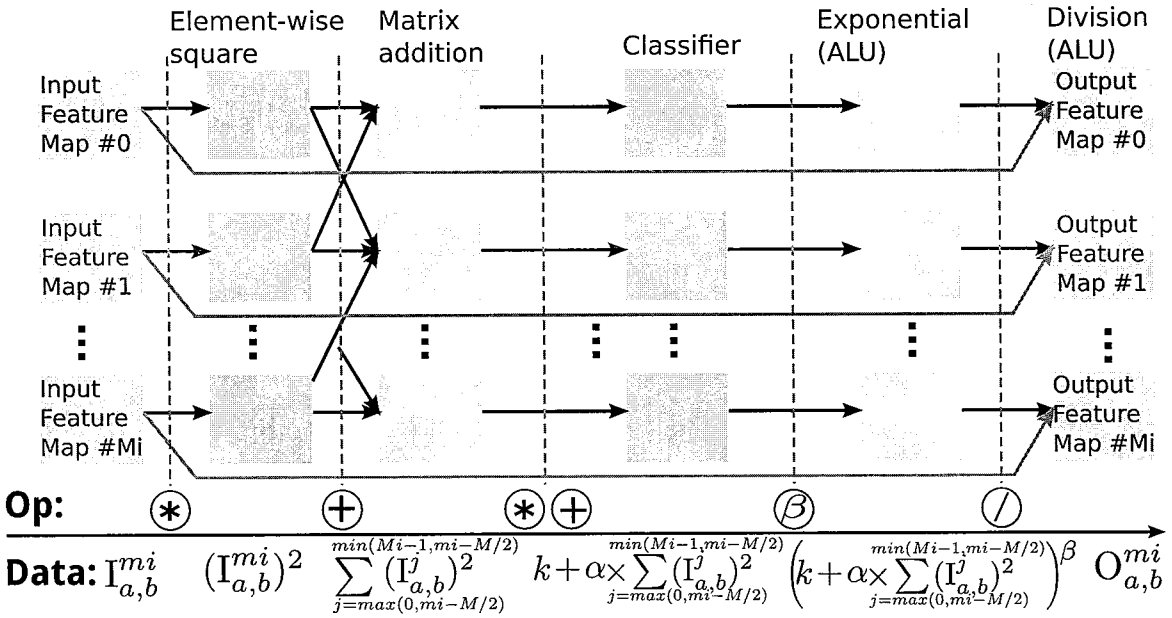

图 4.16　LRN层的分解

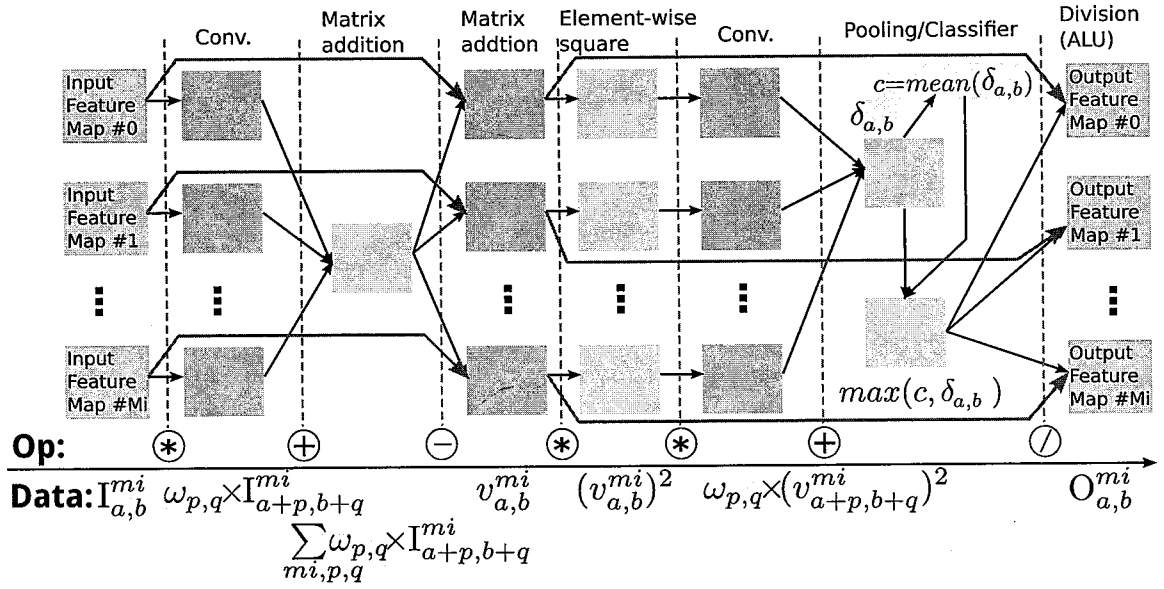

图 4.17 LCN层的分解

述的规则来处理，指数函数和除法由算术逻辑单元ALU完成。其余的计算元语，包括矩阵元素平法和矩阵除法则由NFU进行计算。为了支持这两种元语，在每个周期，每个PE完成一个矩阵元素的计算输出其乘法或者加法的功能，所有$P_x \times P_y$个PEs的结果在稍后被写回至NBout。其中规则在章节 4.6.1中详述。

4.8 实验方法

评估方法 我们用硬件设计语言Verilog完成设计，采用TSMC 65 *nm* Gplus High VT library工艺库，采用Synopsys Design Compiler进行综合，后采用Synopsys IC Compiler进行布局布线后段设计。我们使用CACTI 6.0来评估DRAM的能耗 [139]。为了详细的评估，我们将ShiDianNao和以下四个设计进行比较：

CPU CPU我们采用一个256-bit SIMD机器(Intel Xeon E7-8830, 2.13 GHz, 1 TB memory)。对于所有的测试集，我们采用GCC和编译选项"-O3 -lm -march=native"进行编译，同时打开SIMD指令支持，如MMX、SSE、SSE2、SSE4.1和SSE4.2。

GPU GPU我们采用当前高端的计算显卡NVIDIA K20M（K20M, 5 GB GDDR5, 峰值性能3.52 TFlops，28 *nm*工艺）；我们采用Caffe库，因为Caffe目前被广泛的认为是CNN上最快的GPU库 [15]。

加速器 为了做一个公平的比较，我们重新实现了之前的工作，也即DianNao [29]，使得DianNao和我们的加速器ShiDianNao拥有相同的算术单元——我们实现了一个8 × 8大小的DianNao NFU（8个硬件神经元，每个每周期可以计算8个输入神经元和8个权值，带宽为62.5 GB/s的内存模型；原来的DianNao为16 ×

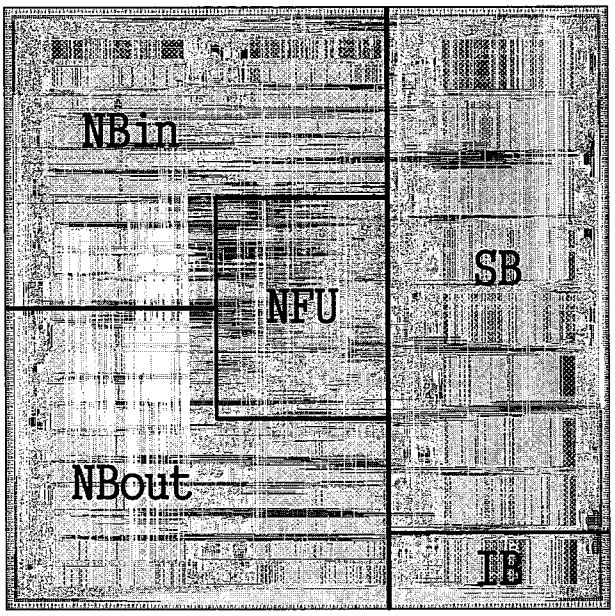

图 4.18　ShiDianNao版图(65 *nm*)

16的NFU， 250 GB/s带宽的内存模型，这样的设计在一个视觉感知传感器上过大)。相应地，我们也缩减了片上的存储以适应相应的设计：1 KB NBin/NBout，16 KB SB。我们验证了我们的设计并且这样的实现也和以前的设计一致，例如重新实现的DianNao面积开销为1.38 mm^2（原始DianNao面积开销为3.02 mm^2 [29])。

测试集　我们选择了10个具有代表性的视觉应用作为我们的测试集(CNP [65], MPCNN [140], Face Recog. [107], LeNet-5 [109], Simple Conv [175], CFF [71], NEO [142], ConvNN [46], Gabor [103]和Face Align. [54])。在所有的测试集中输入神经元最多占用45 KB，权值最大为258.54 KB，都不超过我们设计中的SRAM的大小（参见表 4.3)。

4.9　实验结果

4.9.1　后端布局布线结果

在表 4.3和表 4.4中，我们示出了当前ShiDianNao设计的相关参数和后端布局布线(见图 4.18)的结果。当前设计中，ShiDianNao有8 × 8 (64)个PEs，64 KB NBin，64 KB NBout，300 KB SB和32 KB的IB. ShiDianNao中SRAM总大小为460 KB（大于DianNao中所有存储17.7倍），是为了能够同时将所有针对CNN计算一幅输入图像的数据和指令存储在片上。结果表明，ShiDianNao的总面积开销是DianNao的4.30倍（5.94 mm^2 vs. 1.38 mm^2）.

表 4.2 测试集(*C*：卷积层，*S*：采样层，*F*：分类层)

Layer	Kernel Size #@*size*	Layer Size #@*size*	Layer	Kernel Size #@*size*	Layer Size #@*size*	Layer	Kernel Size #@*size*	Layer Size #@*size*
CNP [65]			MPCNN [140]			FaceRecog. [107]		
Input		1@42*x*42	Input		1@32*x*32	Input		1@23*x*28
C1	6@7*x*7	6@36*x*36	C1	20@5*x*5	20@28*x*28	C1	20@3*x*3	20@21*x*26
S2	6@2*x*2	6@18*x*18	S2	20@2*x*2	20@14*x*14	S2	20@2*x*2	20@11*x*13
C3	61@7*x*7	16@12*x*12	C3	400@5*x*5	20@10*x*10	C3	125@3*x*3	25@9*x*11
S4	16@2*x*2	16@6*x*6	S4	20@2*x*2	20@5*x*5	S4	25@2*x*2	25@5*x*6
C5	305@6*x*6	80@1*x*1	C5	400@3*x*3	20@3*x*3	F5	1000@5*x*6	40@1*x*1
F6	160@1*x*1	2@1*x*1	F6	6000@3*x*3	300@1*x*1			
			F7	1800@1*x*1	6@1*x*1			
LeNet-5 [109]			Simple Conv [175]			CFF [71]		
Input		1@32*x*32	Input		1@29*x*29	Input		1@32*x*36
C1	6@5*x*5	6@28*x*28	C1	5@5*x*5	5@13*x*13	C1	4@5*x*5	4@28*x*32
S2	6@2*x*2	6@14*x*14	C2	250@5*x*5	50@5*x*5	S2	4@2*x*2	4@14*x*16
C3	60@5*x*5	16@10*x*10	F3	5000@5*x*5	100@1*x*1	C3	20@3*x*3	14@12*x*14
S4	16@2*x*2	16@5*x*5	F4	1000@1*x*1	10@1*x*1	S4	14@2*x*2	14@6*x*7
F5	1920@5*x*5	120@1*x*1				F5	14@6*x*7	14@1*x*1
F6	10080@1*x*1	84@1*x*1				F6	14@1*x*1	1@1*x*1
F7	840@1*x*1	10@1*x*1						
NEO [142]			ConvNN [46]			Gabor [103]		
Input		1@24*x*24	Input		3@64*x*36	Input		1@20*x*20
C1	4@5*x*5	1@24*x*24	C1	12@5*x*5	12@60*x*32	C1	4@5*x*5	4@16*x*16
S2	6@3*x*3	4@12*x*12	S2	12@2*x*2	12@30*x*16	S2	4@2*x*2	4@8*x*8
C3	14@5*x*5	4@12*x*12	C3	60@3*x*3	14@28*x*14	C3	20@3*x*3	14@6*x*6
S4	60@3*x*3	16@6*x*6	S4	14@2*x*2	14@14*x*7	S4	14@2*x*2	14@3*x*3
F5	160@6*x*7	10@1*x*1	F5	14@14*x*7	14@1*x*1	F5	14@3*x*3	14@1*x*1
			F6	14@1*x*1	1@1*x*1	F6	14@1*x*1	1@1*x*1
Face Align. [54]								
Input		1@46*x*56						
C1	4@7*x*7	4@40*x*50						
S2	4@2*x*2	4@20*x*25						
C3	6@5*x*5	3@16*x*21						
S4	3@2*x*2	3@8*x*10						
F5	180@8*x*10	60@1*x*1						
F6	240@1*x*1	4@1*x*1						

表 4.3 ShiDianNao和DianNao的参数设置

	ShiDianNao	DianNao
数据位宽	16-bit	16-bit
# multipliers	64	64
NBin SRAM大小	64 KB	1 KB
NBout SRAM大小	64 KB	1 KB
SB SRAM大小	300 KB	16 KB
Inst. SRAM大小	32 KB	8 KB

表 4.4 ShiDianNao硬件结果（1GHz，功耗和能耗为10个测试集上的平均据结果）

Accelerator	面积（mm^2）	功耗（mW）	能耗（nJ）
Total	5.94 (100%)	336.51 (100%)	6770.13 (100%)
NFU+ALU	0.66 (11.11%)	268.82 (79.88%)	5515.87 (81.47%)
NBin	1.12 (18.86%)	38.18 (11.34%)	518.73 (7.66%)
NBout	1.12 (18.86%)	6.60 (1.96%)	86.62 (1.28%)
SB	2.96 (49.83%)	20.53 (6.10%)	266.29 (3.93%)
IB	0.31 (5.21%)	2.38 (0.71%)	36.21 (0.53%)

4.9.2 性能

在本章节中，我们使用章节 4.8中所有的10个测试集将ShiDianNao和CPU、GPU还有重新实现的DianNao加速器进行比较，在图 4.19中展示了相关的结果。不出意料，ShiDianNao在性能方面能够显著的优于通用计算平台：比CPU快46.38×，比GPU快29.94×。特别的是，因为表 4.1视觉处理任务中的小的计算核并不能很好的映射到GPU的2496个线程上，GPU并不能完全的被利用起来。

更有趣的是，和DianNao相比，ShiDianNao在10个测试集中的9个都有加速比（在10个测试集的平均加速比为1.87×）。原因主要是两方面：首先，和DianNao相比，ShiDianNao由于片上有足够大的存储（相应的开销也大），从而可以免去片外的存储访问；其次，ShiDianNao能够通过其专门设计的控制器和PE间数据传递利用2D数据的局部性，从而也更高效，反之DianNao则不曾考虑这样的问题。

在测试集*Simple Conv*上ShiDianNao略慢于DianNao。其原因在于ShiDianNao一次计算一幅输出特征图像，每个PE同时在计算同一幅特征图像上不同的输出神经元，也因此当一个应用中包含不同寻常小的输出特征图像，其上的是输出神经元数目甚至小于PE的个数（如*Simple Conv*中特征图像大小为5×5的C2层和本加速器设计中8×8的PE数目）时，一些PE将会被空置而不进行运算。尽管我们也曾采用了增加复杂的控制逻辑的方法，来使得不同的PE可以同时进行不同的输出特征图像的计算，从而避免这种PE浪费，然而这样使得编程的逻辑极其复杂化，最终我们放弃了这样的方法。

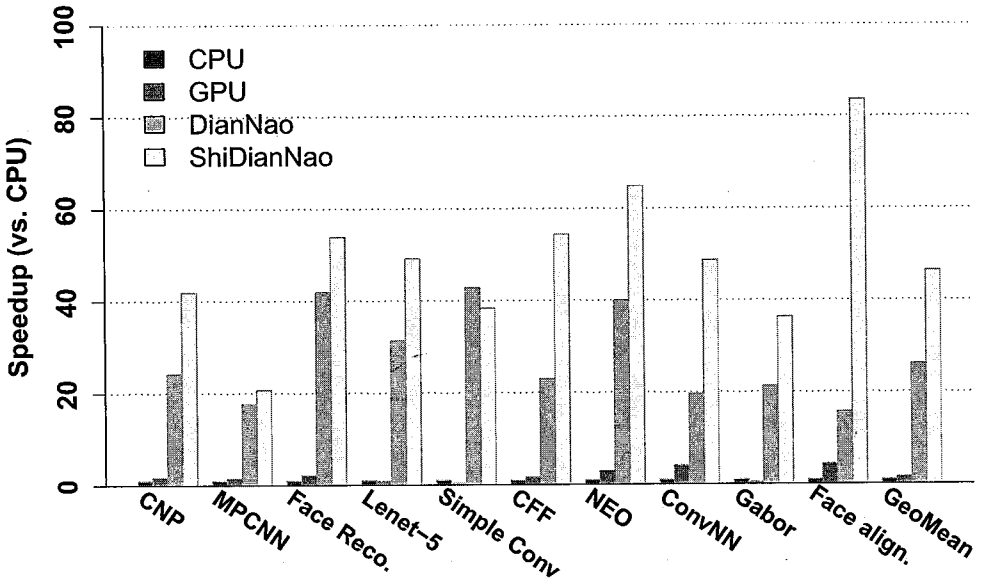

图 4.19 相较于CPU，GPU，DianNao和ShiDianNao的加速比

考虑在实时系统中直接通过传感器获得图像数据并采用简单的窗口扫描方法使用ShiDianNao，处理640x480大小的视频流中的一帧的最长时间是测试集*MPCNN*：0.079 ms处理一个32×32的像素区域。每帧图像包含$\lceil(640-16)/16\rceil \times \lceil(480-16)/16\rceil = 1131$这样的像素区域（假设扫描的步长是像素区域大小的一半，即16个像素），一帧图像则需要约89 ms来处理，也即对于最费时的测试集来说实时处理的帧率为11帧。大部分的商业传感器能够流式地以一个可观的速率传送数据从而传送速率也能跟得上处理的速率，视频流中一帧图像的部分需要被缓存下来以提供重叠区域的复用。这样的图像规模大小是十几个像素大小的像素行，能够绰绰有余的存储在256 KB商业图像处理器中。尽管分辨率有些小，然而640×480这样的分辨率符合目前常用的图像缩放的范围[63,81,108,160]。

4.9.3 能耗

在图 4.20中，我们报告了GPU、DianNao和ShiDianNao的能耗数据，这部分数据包含从主存获得输入图像数据的能耗。尽管ShiDianNao的本意是不和主存发生交互，我们还是保守的包含了这部分功耗来进行比较。ShiDianNao相较于GPU和DianNao则分别节约能耗3734.60×和54.46×。我们同样评估了理想的DianNao模型，这里指的是假设和主存交互这部分的能耗为0（DianNao-FreeMem，参见图 4.20）。有趣的是，我们注意到ShiDianNao仍然比DianNao-FreeMem节约1.41×的能耗。更多的是，当ShiDianNao集成在视觉传感器内部时，视频帧能够直接被写入加速器的NBin中，ShiDianNao的优越性得到更一步的展示：ShiDianNao相较于DianNao和DianNao-FreeMem分别节约能耗72.56×和1.88×。

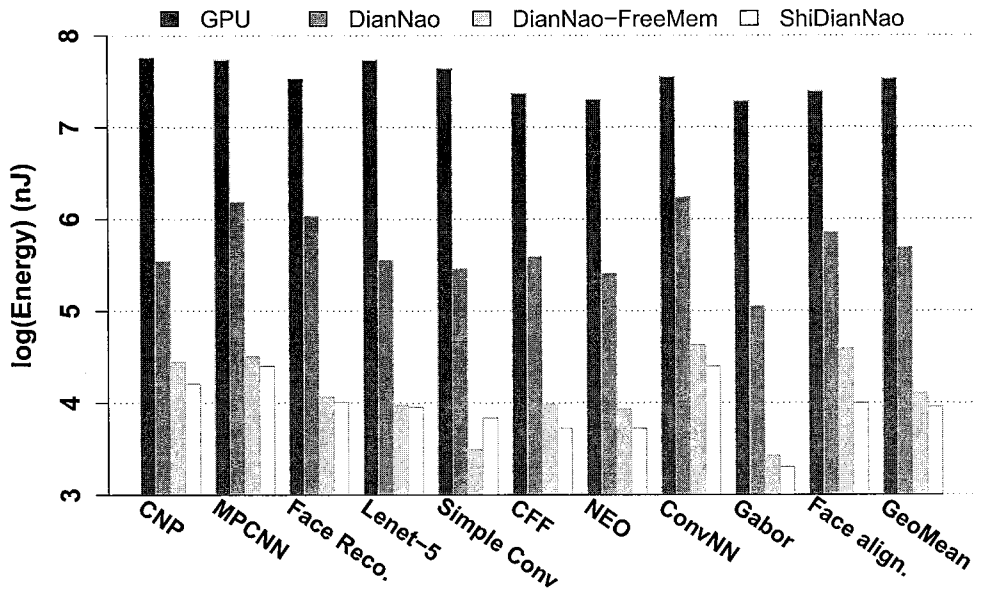

图 4.20 GPU, DianNao，和ShiDianNao的能耗

在表 4.4中，我们显示了加速器中能耗的组成部分。我们注意到在整个能耗中，SRAM的能耗占有全部的13.61%，其余的则全部是逻辑电路（81.47%）。这点和Chen等人从DianNao [29]得到的结论截然不同，在DianNao中，多达95%以上的能耗是花费在DRAM上。

4.9.4 应用ShiDianNao到传统的视频处理任务

通过将视觉处理的位置移动至离传感器更近的位置，ShiDianNao能够消除绝大部分的DRAM存储访问，从而能够大幅度地提升视觉识别任务的效率。同样的，ShiDianNao也能够被用到传统的视频处理任务中，如去马赛克（Demosaic），Bayer重建（Bayer Construction），运动估计（Motion Estimation） [196]等等。这是因为上述的应用能够自然的在ShiDianNao中进行运算。在本章节接下来的部分，我们以视频编码中最为耗时的部分（~90% [30,76]）运动估计为例，来说明ShiDianNao如何完成类似任务的计算。

特别的，在视频编码标准（如H.264）中，每帧图像会被划分成为图像块，图像块稍后会被划分成为图像子块。对于当前帧中的每个图像块或者图像子块，运动估计的目的就是在参考帧中寻找到与这些图像块或者图像子块最佳匹配的图像块或者图像子块。运动估计中匹配过程首先在原始图像像素（Integer Pixels）上进行，稍后通过上采样对原始像素进行差值获得分数像素（Fractional Pixels），然后在分数像素上进行再次匹配。以上的两个步骤被分别称为整数运动估计（Integer Motion Estimation，IMF）和分数运动估计（Fractional Motion Estimation，FME），参见图 4.21。整个匹配的过程可以规范成为参考帧和当前帧的卷积操作，类似卷积神经网络中的卷积层，从而可以高效的通过ShiDianNao完成计算。更多的是，从整数像素获得分数像素的上采样过程可

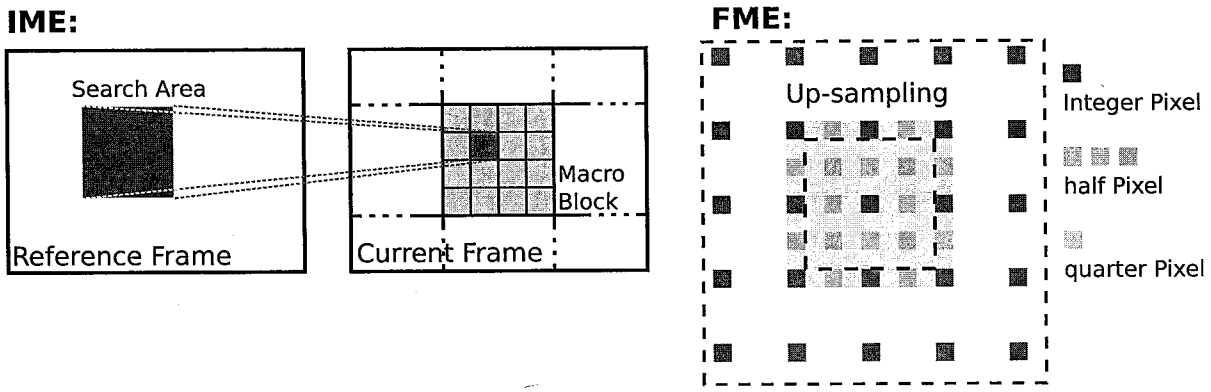

图 4.21 IME：图像字块搜索。FME：上采样

以规范成为单纯的卷积，从而也可以在ShiDianNao上有效地进行计算。

在表 4.5和表 4.6中，我们在五个流行的1080p（1920×1080）的视频序列上（*Basketball, Rush_Hour, Cactus, Pedestrian, Sunflower, Riverbed*）上，比较ShiDianNao和一个在65nm工艺的视频编解码专用加速器，也即ME_ACC [190]。我们注意到ShiDianNao在面积、功耗上并不能超过ME_ACC这个专用的加速器，这个结果和Qadeer [155]得到的结论一致。我们也注意到和标准的视频压缩算法（典型的32×32搜索区域）相比，ShiDianNao有0.1% PNSR损失和1.7% BitRate的增加。在本例子中，我们说明了ShiDianNao能够应用到传统的视频处理应用中，并不仅仅局限于神经网络算法。

4.10 ShiDianNao+：ShiDianNao的多核扩展

随着传感器能力的提升，在移动端和可穿戴设别上视频流具有越来越高的分辨率和帧率，也因此使得实时处理视频流变得愈发困难。尽管目前ShiDianNao的设计能够以11帧每秒（frame-per-second，fps）处理最耗时的任务*MPCNN*，但是这仍不足以处理分辨率比640x480更高或者帧率大于11帧每秒的视频流。

表 4.5 硬件结果比较

	ShiDianNao	ME_ACC [190]
Area (mm^2)	5.94	0.67
Power (mW)	336.51.10	360.70
Freq. (GHz)	1.00	0.98

表 4.6 实验结果（6个测试集上，默认设置）

Parameters	Value
Video	1920x1080
Search range	32x32
Max FPS	62
PSNR Loss (%)	0.1
BitRate Increase (%)	1.7

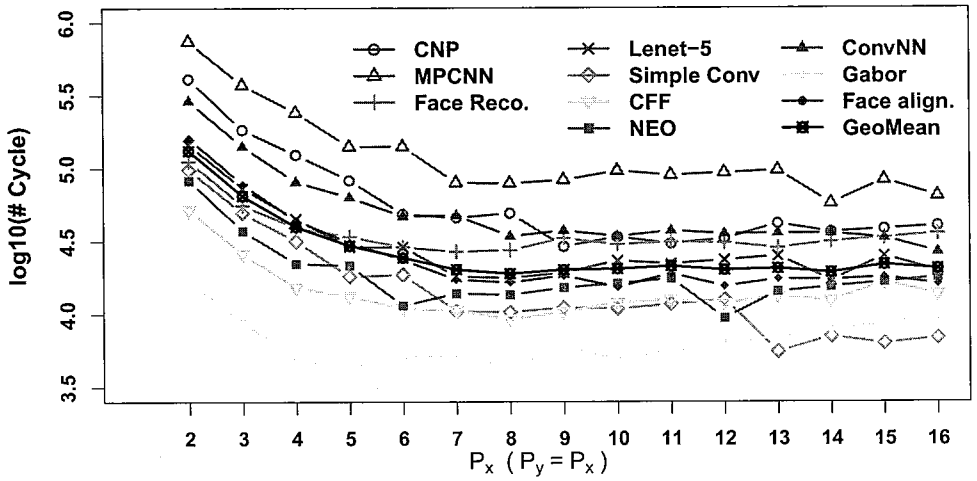

图 4.22　不同# PE(# PE = $P_x \times P_y$)所需的执行周期数

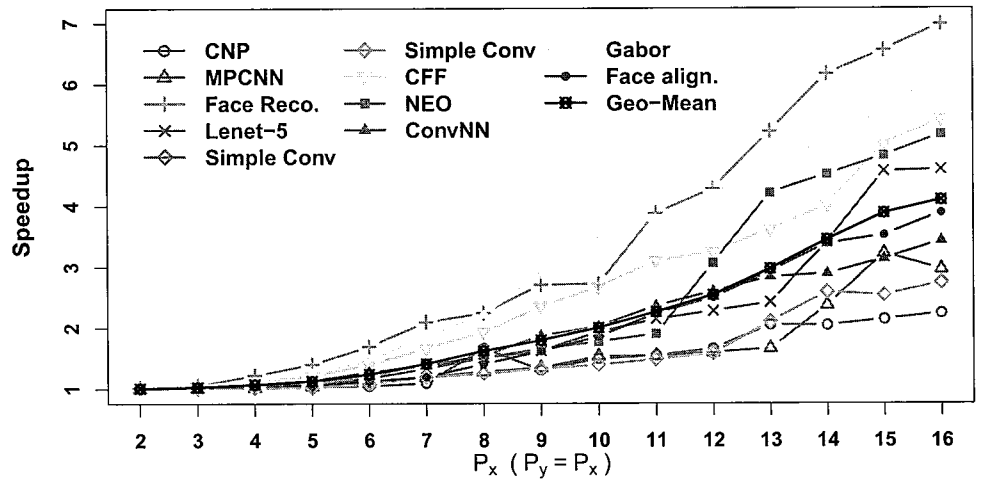

图 4.23　# P_y ALU vs. # 1 ALU时ShiDianNao的加速比

4.10.1　增加PE的数目：直观的方法

一个直观且简单的解决方法是增加ShiDianNao的PE数据，从而提升ShiDianNao处理的并行度。这里我们变换ShiDianNao中的PE数目（4到256）而不改变ShiDianNao结构中的其他部分来观察PE数目对于性能的影响，我们仍然在章节 4.8中所涉及的10个测试集进行评估。在图 4.22中我们报告性能的结果。我们注意到每个测试集的执行周期数并按比例随着PE数目的增加而减少。这可能的原因有两个。首先，尽管程序中的大部分运算可以并行的计算，其中的非线性运算因为加速器中只有一个ALU所以仍然需要串行的处理。其次，增加PE的数目并不能总是提升性能。例如，当PE的数目大于输出特征图像的大小后，执行时间就不会随着PE数目的增加而减少，而是保持不变，也即这时PE会存在空转浪费的情况。

4.10.2 增加ALU的数目

为了提升ShiDianNao的性能，我们针对上面所发现的问题增加ALU的数目来促进非线性运算的速度。我们对PE从2x2到16x16同时ALU从2到16的变化的ShiDianNao设计，同样采用章节4.8中涉及到的10个测试集进行测试并在图4.23中报告性能结果。单个ALU的面积开销为0.015 mm^2（为NFU面积开销的2.3%）。我们观察到对于采用P_y个ALU的ShiDianNao设计在10个测试集上的性能提升平均为1.01x到4.09x：特别是8×8PE的ShiDianNao设计有1.62x性能提升。然而，这样的性能提升仍然不能满足一些需求，如10帧每秒用$MPCNN$识别1280x720的视频流。总之，通过简单的增加ShiDianNao中PE的数目或者ALU的数目并不是一个令人满意的提升性能的解决方案。

4.10.3 ShiDianNao+：结构、映射和评估

在本章节中，我们提出了ShiDianNao的多核扩展ShiDianNao+来适应视频流中的高分辨率和高帧率。

4.10.3.1 ShiDianNao+：结构

在图4.24中我们展示ShiDianNao+的结构图。ShiDianNao+是ShiDianNao的多核扩展，通过在NFU中引入多个核（每个核有$P_x \times P_y$个PEs和P_y个ALU）实现。ShiDianNao+能够在视觉处理任务上获得比ShiDianNao更好的性能是由于：（1）ShiDianNao+中的多核允许多个不同的输出特征图并行计算；（2）每个核中的多个ALU允许以高并行度计算算法当中的非线性运算。我们选择和ShiDianNao中一样的数据带宽以避免在NFU和存储间引入过大的数据带宽开销，同时也使得核数目和数据带宽需求脱离，从而使得核心数目具有良好的可扩展性。另外，我们也修改了控制逻辑使得ShiDianNao+能够选择核来执行。

4.10.3.2 ShiDianNao+中的映射

在图4.25 (a)中所示的典型的卷积层，N_c个输出特征图像能够在ShiDianNao+中的N_c个核上并行运算，而输入为同一幅输入特征图像。同一幅输出特征图像上神经元被映射到同一个核上进行预算，其运算过程如同1-core的ShiDianNao。每个核会一直计算自己当前的输出特征图像，直到当前的输出特征图像被计算完成才会开始计算新的输出特征图像。如图4.25 (b)所示，从NBin中获得的一幅输入特征图像会广播至所有N_c个核用于计算，同时每个核会单独获取自己相关的输入权值，即卷积核。在大多数情况下，卷积核的大小小于输入输出特征图像，并且在一对输入输出特征图像只存在一个卷积核。也因此，对于计算一对输入输出特征图像卷积核只需要载入一次用于计算使得读取的开销可以忽略不计。

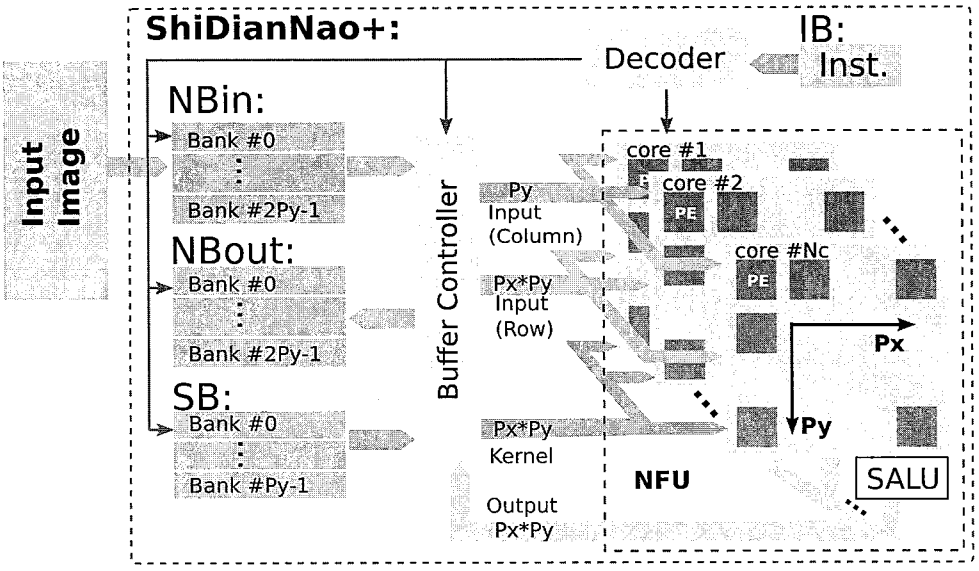

图 4.24　ShiDianNao+结构

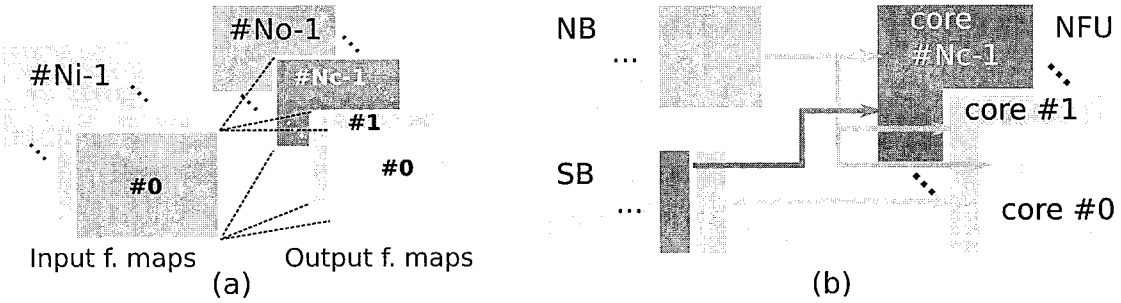

图 4.25　ShiDianNao+上调度映射卷积层

不同于卷积层，我们仍然将采样层和分类层映射到单一的核上以保持这种相较于卷积层计算压力小的层控制逻辑的简单性。ShiDianNao+仍然能够在不同的核数目下获得性能收益这是因为卷积层通常是CNN中运算量最大的部分。因此，在这里，我们对上述的层只使用一个核计算。

4.10.3.3　ShiDianNao+评估

我们实现了有1到4个核（每个核有8 × 8个PE）的ShiDianNao+的设计，同时保持了NBin、NBout和SB的大小分别为64KB、64KB和300KB。我们用ShiDianNao为基

表 4.7　ShiDianNao+硬件结果（功耗和能耗为10个测试集上的平均结果）

Accelerator	Area (mm^2)	Power (mW)	Energy (nJ)
ShiDianNao	5.94	336.51	6770.13
1-core	6.03	459.73	5405.89
2-core	6.83	776.29	5529.11
3-core	7.65	1102.80	5961.48
4-core	8.50	1440.04	5622.66

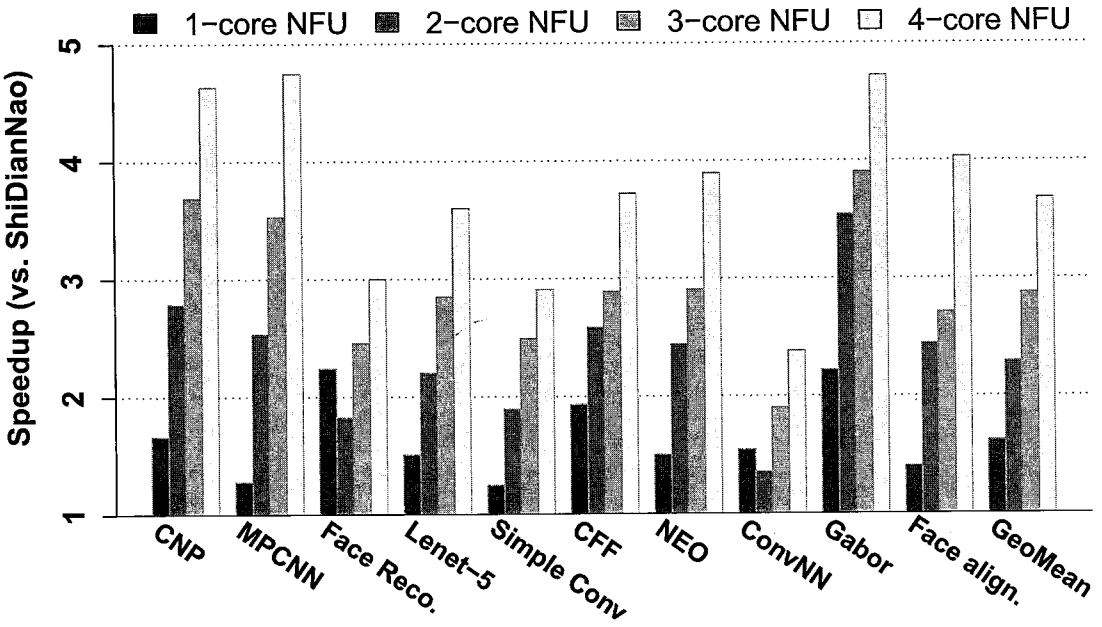

图 4.26 ShiDianNao+性能提升加速比

准，将ShiDianNao+的设计与之相比，在表 4.7和表 4.4中报告了实验结果。我们注意到ShiDianNao中NFU占有11%面积但是却占有81%的能耗开销，从而使得多核扩展的面积开销也不大。ShiDianNao+的1-core, 2-core, 3-core和4-core版本相较于ShiDianNao面积开销分别多1.51%, 14.98%, 28.79%和43.10%。

性能 我们在图 4.26中报告1-core, 2-core, 3-core和4-core版本的ShiDianNao+相较于ShiDianNao在前述的10个测试集上的性能表现：ShiDianNao+分别获得1.62x, 2.29x, 2.87x和3.68x性能提升加速比。我们注意到1-core的ShiDianNao+（也即多ALU版本的ShiDianNao，8个ALU）在测试集*Face Reco.*和*ConvNN*比2-core的ShiDianNao性能更好。这是由于我们的调度映射策略所导致的。不失一般性的，我们对待卷积层的连接为全连接，也即卷积核存在于任何输入特征图像和输出特征图像，也即每个输入特征图像对于输出特征图像的结果都有贡献（卷积核卷积结果）：原来不存在的连接也被补上卷积核全是0的连接。更多的是这样使得ShiDianNao+的调度映射策略在任何数目下都保持一致。然而特别的是在1-core情况下，ShiDianNao+却可以利用卷积层中不存在的连接，只计算那些存在的连接。也因此，在*Face Reco.*的C3层，1-core ShiDianNao+只需要计算其存在的125个连接（20幅输入特征图像，25幅输出特征图像）而2-core ShiDianNao+则需要计算全部500（25×20）个连接。考虑到视频流的实时处理，1-core, 2-core, 3-core和4-core版本的ShiDianNao+能够在10个测试集上平均实现1280x720视频25, 35, 44, 56 fps的处理速度。注意，这还是通过采用简单重叠分割的方法来实现识别方法下的速度，这样的处理能力能够保证ShiDianNao+能够处理更加复杂

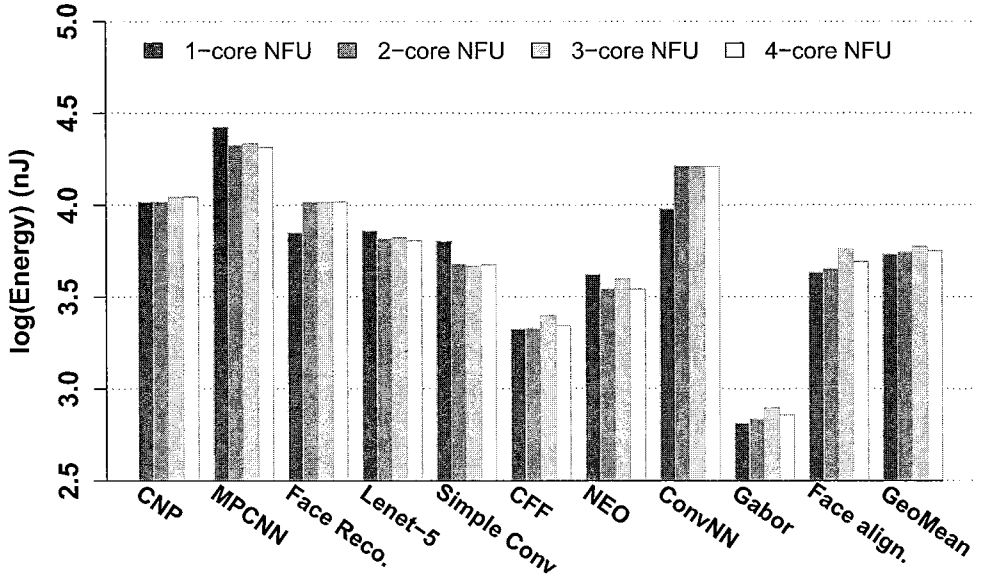

图 4.27　10个测试集上的能耗

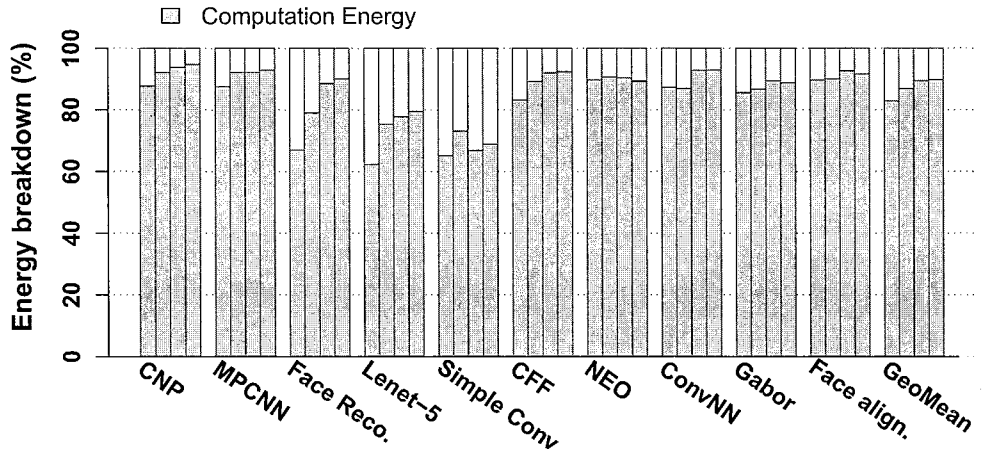

图 4.28　ShiDianNao+能耗分析（每簇中从左至右：1-core NFU, 2-core NFU, 3-core NFU, 4-core NFU）

的视频流实时处理需求。

能耗　采用和ShiDianNao一样的评估方法，在图 4.27和表 4.7中我们报告ShiDianNao+的能耗开销。我们注意到随着核心的数目增加ShiDianNao+的能耗开销并不减少，这主要是因为增加的核心的能耗开销能够很好被性能提升中和去。当继承了更多的核心数目，NFU上的能耗开销增加但是读取输入特征图像的能耗减少（读取次数减少）同时读取时间也减少。总的来说，1-core, 2-core, 3-core和4-core ShiDianNao+相较于ShiDianNao能够分别实现能耗节约20.15%, 18.33%, 11.94%和16.95%。在10个测试集上SRAM的平均能耗开销占总能耗的比例随着核心数目增加而减少：相较于ShiDianNao的13.61%，ShiDianNao+中SRAM分别占总能耗15.82%, 11.68%, 9.38% and

8.84%，参见图 4.28。总而言之，ShiDianNao+为高性能目的的计算提供了一个高效能的可扩性设计。

4.11 相关工作

视觉传感器和处理 基于集成电路和传感器技术的迅速发展，视觉传感器的尺寸和成本大幅度下降的同时能力大幅度提升，使得移动终端中和可穿戴设别中配备高分辨率视觉传感器成为可能，例如，Google Glass [180]和Samsung Gear [165]。在新兴的应用场景如图像识别/搜索 [75,180]中，由于有限的计算能力和能耗限制，终端并不在本地完成这些计算密集的视觉处理。相反的，计算密集型视觉处理算法如CNN [71,91,103,107]多在服务器端完成，给服务器带来巨大的压力从而大大的限制了服务的质量和服务的用户数量。我们的研究通过将视觉处理搬移至距传感器更近的地方来平衡这样的差距。

神经网络加速器 通常神经网络是在计算能力强大的CPU [25,191]上或者GPU [35,62,168]上进行运算。然而这些计算平台尽管具有相当的灵活性能够适用于不同的任务，但这样灵活性的代价是复杂的硬件支持，严重的影响了执行特定应用（如CNN）时的性能和能耗。在上世纪最后几年中，曾经出现过不少特定神经网络应用专用的结构 [86]，或实现在FPGA [65,166,171]上或是ASIC [29,63,187]实现。对于CNN，Farabet等人 [63]提出一种systolic类似的结构成为NeuFlow。Chakradhar等人是实现了一个systolic类似的协处理器 [25]。尽管能够高效的处理信号处理中的二维卷积 [80,112,114,189]，systolic结构不能够灵活且高效的支持不同的CNN配置（如CNN中的卷积窗口大小，卷积步长）以及相应的高带宽需求 [25,44,63,166]。也有一些神经网络加速器结构采用了SIMD类似的结构。Esmaeilzadeh等人提出了神经网络流处理核（Neural Network stream processing core，NnSP） [56]，其包含PE处理单元的阵列，然而NnSP是为MLP所设计的。Peemen等人 [152]采用FPGA和一个主处理器去加速器CNN，尽管该加速器配备了一个专用的为CNN优化过的存储系统，系统却需要主处理器参与使得能效性大打折扣。Gokhale等人 [74]为移动设备设计了用于CNN和DNN的协处理器。以上的研究都没有把内存访问看作很重要的因素，甚至直接通过DMA和内存连接在一起。最近，Chen等人 [29]设计实现了适用多种神经网络（如CNN和DNN）的加速器DianNao，并为之设计了片上SRAM以减少内存访问。然而为了灵活的支持多种不同的神经网络算法，DianNao并没有专门为CNN优化，并不包含专门针对CNN中数据多具有的二维特性的优化（反之DianNao把2D数据当做1D数据进行计算）。也因此，在执行CNN时，DianNao仍然需要频繁的访问内存，从而使得整个设计相较于ShiDianNao不够高效（参见章节 4.9中的实验结果）。最近DianNao家族中的其他加速器成员 [33,122]是专门为大规模神经网络和机器学习算法进行优化，然而他们都不是为嵌入式终端设计，也

和ShiDianNao的结构截然不同。

相比较之前的设计，我们的ShiDianNao主要在两个方面大不相同。首先，不同于之前的加速器设计需要从内存中获取数据，执行CNN过程中，ShiDianNao并不需要访问内存。第二，不同于之前systolic类的设计支持特定参数和层数的CNN [25,44,63,166]，ShiDianNao能够灵活的适用于不同参数的不同层数的CNN。正是由于这些引入注意的特点，ShiDianNao相较于以前为CNN加速的设计具有更好的能耗性，特别是对嵌入式系统中的视觉处理任务。

4.12 本章总结

在本章中，我们为目前前沿的视觉处理算法设计并实现了一个灵活的加速器。在10个代表性的测试集上，我们的设计相较于CPU、GPU和重实现的DianNao [29]快50×，30×和1.87×。ShiDianNao的能耗是GPU和DianNao的3700分之一和54分之一。ShiDianNao在65nm工艺下的面积开销为5.94 mm^2，频率为1GHz，功耗为336.51 mW。正是由于高性能，低能耗，低面积开销，ShiDianNao能够适应在移动终端和可穿戴设别上的视觉处理应用。更进一步的，我们提出了ShiDianNao+在保持高能效性时提高性能。ShiDianNao适合集成在传感器上从而可以高效的本地处理相应的任务。这样能够大幅度的降低服务器端的工作负载，提高视觉处理的服务质量，最终能够为无处不在的视觉处理奠定基石。

第五章　非精确神经网络

5.1　引言

由于越来越严峻的能源制约，研究人员已经开始认同节约能源需要整体性的方法，它涉及计算系统的各个方面，从电路到应用。一种理想的节约能源的方法是通过牺牲实际应用的精度来换取能源的节约，这种方法通常被称为*非精确计算* [150]或是*近似计算* [58](inexact or approximate computing)。驱动这一方法的背后原理是某一些应用不需要现如今的电路和编程方法实现的那么高的精度。例如视频解码可以容忍相当程度的不准确性，因为图像的变化对人眼来说可能是不可见的。有些研究开始利用这一发现来减少逻辑（logic），即逻辑运算符的精度 [118]。这些技术提出可以通过删去无意义的组成部分 [118]或是转化为能耗更低的相似的逻辑 [10,41]来减低逻辑单元的密度。

虽然这个理念很吸引人，但实际上它大多被用在专用集成电路（ASICs)上来利用目标应用程序中特定通路中的逻辑运算符的特定连接 [120]。这些ASIC（或是在其上运行的程序）的错误容忍能力受限于已经设计的非精确计算，也限制了可以实现的资源节约的程度。

我们探索结构和应用的两大趋势来寻求克服这一限制的方法。结构上的趋势来源于日益增加的能源限制：根据预测，多核结构（嵌入式或者高性能）将越来越多地依赖能够处理密集计算任务的*加速器*（异构多核），因为缺少电压调节技术，还有尤其是“暗硅”带来的影响 [137]。在传统的加速解决办法如GPU、FPGA以及ASIC之外，研究人员越来越多地探究具有适中应用范围（包含许多但不是所有应用）的加速器，目的是想获得比GPU和FPGA更高的能源效益[184,193]。多个这样的加速器可以在同一个芯片上实现，这样这些加速器在一起的应用范围将会非常广。第二个趋势与计算密集型的应用的本质有关。几年前Intel让体系结构领域注意到它的参考基准的设计是有问题的，并且将识别、挖掘和综合应用看作兴起的高性能应用的代表 [53]，这也推动了之后的Intel和Princeton大学的合作成果PARSEC参考基准测试程序集[17]的提出。

一些最近的研究 [78,187]指出结合这两种趋势，我们将得到一种非典型的但极具吸引力的可选加速器方案：*硬件神经网络*。最近，Chen等人 [27]指出对于半数或更多的PARSEC参考基准，90%甚至更多的执行时间是可能通过这样一个硬件神经网络加速器上进行加速。因此，神经网络更像是在其上能够实现大范围应用的核。虽然也存在着其它的能够应付测试基准的加速器选择，但是固有的容忍错误的能力使得神经网络尤其有吸引力。例如，最近的研究表明一个硬件神经网络加速器能够容忍电晶体级别的错误 [187]。

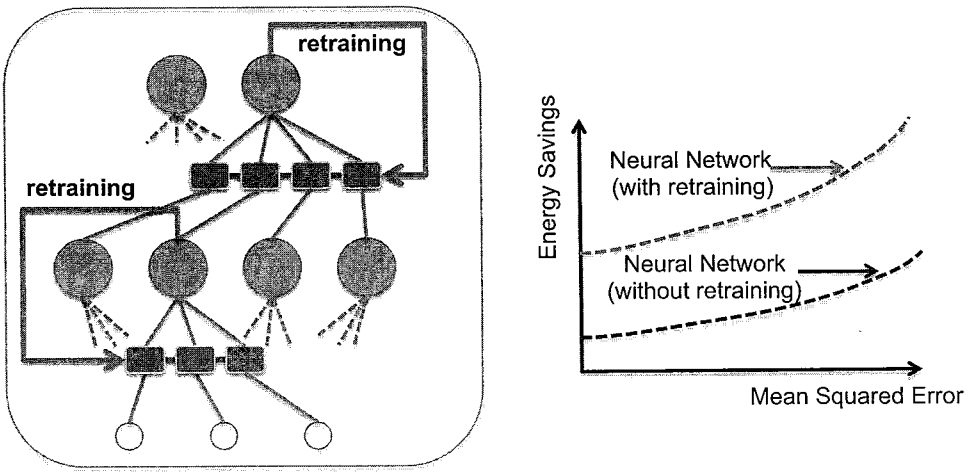

图 5.1　神经网络通过重训练恢复精度或者实现更高的能耗节约

在这篇文章中，我们从以下发现开始：1）与普通认知不同，硬件神经网络加速器的应用范围非常广，考虑到越来越多的计算密集型应用的出现；2）尽管有较大的应用范围，一个硬件神经网络就是一个核中的巨大的数据通路电路，没有复杂的控制，这就使得它能够直接受益于计算电路的优化；3）它实现了一个特有的能够在重新训练后容忍错误的算法。

我们在非精确计算中利用这三个发现。首先，在硬件神经网络上使用非精确计算，这样所有利用这个加速器的应用的能源节约量都会增加，就不再受限于ASIC了。然后我们利用神经网络的错误容忍能力来最大化可容忍的不精确度，也就最大化了能源节约量，这个能力来源于神经网络的学习算法（参见图 5.1）。使用一个带有非精确逻辑运算器的硬件神经网络加速器，在一系列University of California, Irvine (UCI) machine-learning repository中的机器学习任务上，我们证明了相比于一个有标准逻辑运算器的硬件神经网络加速器，平均而言可以实现54.45%的能源节约，31.44%的面积减少，代价只有0.06的误差（Mean Squared Error，MSE）。之后我们也将这个发现应用到最先进的神经网络加速器DianNao [29]上，发现尽管节约了42%的面积和50%的能源，非精确的NFU仍然能够实现97.10%的精度，精度损失不超过1%。

在章节 5.2，我们描述了非精确计算和硬件神经网络的主要概念，并且指出这两者是可以相结合的。在章节 5.3中，我们探讨非精确硬件加速器形成的设计空间，潜在的能源、延迟和面积的节约，以及它们对任务精度的影响。最后我们在章节 5.4 中描述了相关的工作，在章节 5.5中提供了结论。

5.2　非精确硬件神经网络

在这一节，我们先对非精确计算和神经网络做一个简单的认识，然后再论述它们如何相互结合，来提过硬件神经网络的能效，同时不会带来明显的精度损失。

5.2.1 非精确计算介绍

考虑一个计算特定的布尔函数$\mathcal{F}$: $B^n \rightarrow B^m$的电路，将n输入的布尔向量$\mathbf{x} = [x_1, x_2, \ldots x_n]$映射成为m维的输出布尔向量$\mathbf{y} = [y_1, y_2, \ldots y_m]$，其相关的硬件开销函数为$\mathbf{C}_{\mathcal{F}}$。非精确逻辑最小化的目标就是找到一个布尔函数$\mathcal{F}'$: $B^{n'} \rightarrow B^{m'}$，其中$n' \leq n$且$m' \leq m$, 使得其相应的硬件开销函数$\mathbf{C}_{\mathcal{F}'}$在以下条件下最小：

$$\sum_{\forall \vec{i} \in \mathbf{I}} \frac{|\mathcal{F}(\vec{i}) - \mathcal{F}'(\vec{i})|}{L} \leq Er_{th} \tag{5-1}$$

其中$\mathbf{I}$电路相应的L个测试向量，Er_{th}是电路的错误恢复上限（这里表示平均错误）。

在本研究中，我们将使用概率逻辑最小化（Probabilistic Logic Minimization，PLM）[10] 的一个修正形式来设计非精确电路。这个技术涉及到对布尔函数逻辑层次的操作，在不同的专用背景下特意去做位上的翻转（即对给定的输入特意使输出从0 → 1或1 → 0来帮助接下来的逻辑最小化）。经过精细的操作，对一个输出$y_i \in \mathbf{y}$,我们定义函数$\mathcal{F}$的两个集合**on-set**和**off-set**，$\mathbf{x}^{\mathbf{ON}} \subseteq B^n$使得$\mathcal{F}(\mathbf{x}^{\mathbf{ON}})=1$，$\mathbf{x}^{\mathbf{OFF}} \subseteq B^n$使得$\mathcal{F}(\mathbf{x}^{\mathbf{OFF}})=0$。PLM技术依赖于在这两个集合上使用位翻转来利用三次方展开进而降低逻辑复杂度。一个例子就是将其应用到一个大多数计算电路都有的基本块的构建上，即一个3输入2输出的全加单元如图 5.2所示。

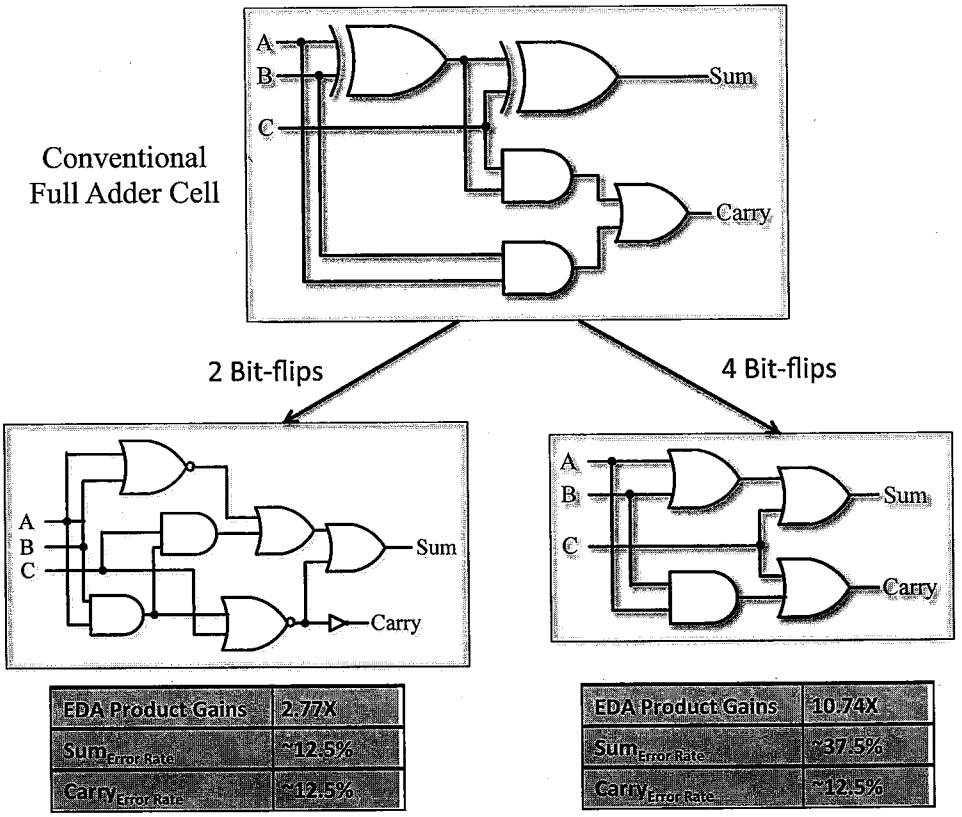

图 5.2 非精确逻辑最小化实现的非精确设计

然而，利用位翻转技术来探索一个完备设计的空间从计算上来说非常棘手，因为在给定的精度限制下解空间将会产生组合爆炸。举个例子，对一个n输入m输出的函数，在最坏的情况下（比如以异或函数为主的数据通路电路）,对每个输出我们需要评估$\sum_{i=0}^{k/2} C(2^k, i)$种可能的逻辑函数，$k = 2^n$是输入向量组合的基数。这将导致一个输入向量基数的双重指数级的复杂度$\mathbf{O}(2^{m.2^n})$。因此，围绕这个棘手的问题，我们使用一个与统计搜索相结合的、以重要性为导向的启发式贪婪方法，来减小搜索空间使其变得计算可行。

简单的说，对电路主要输出的误差单元利用灵敏度分析可以得到每个结点的能够传入的误差总量，在这个基础上给电路中每个结点分配一个重要性或是等级 [93]。我们把这样一个等级数组当作指引传入每个结点的误差量的模板。在数据通路电路的设计中，我们做一个受输出重要性驱动的工作，即给输出重要性较高（二元）的结点分配较高的等级，相应地被分配了较少的非精确结构。由于解空间是非常庞大的，与其做全空间搜索去发现全局最小值，倒不如使用一个基于启发式的设计空间的搜索，见图5.3中的算法，这样仍然会有好的设计。

我们使用这样的基于启发式的搜索来寻找神经网络中最重要的那些运算器，并且给它们建立了库。我们在章节 5.2.3中描述了这些运算器的特性，库的描述见章节 5.3.1。

5.2.2 结合非精确计算和神经网络

在章节 5.2.1中我们描述了非精确逻辑最小化被应用到单独的运算器级别上比如加法器和乘法器。这项技术将分两步应用到神经网络上。首先将神经网络看作是一个巨大的数据通路电路，将标准的逻辑运算器替换为非精确逻辑运算器，这样就最小化了输出误差。再考虑图 3.3发现，一个神经网络的数据通路仅仅是由加法器和乘法器组成，乘法器计算突触权值和低一层的神经元输出的乘积，加法器计算这些乘积的加和，同时激活函数也可以用一个乘法器和一个加法器实现（$f(x) = a_i \times x + b_i$)。上述的步骤可以说是应用非精确计算的标准方法，即搜索出那些传入最小误差的非精确逻辑运算器。

第二步，这是神经网络独有的，即重新训练神经网络来减少非精确逻辑运算器带来的影响。结果得到的最后的神经网络将会是完全相同的硬件结构但是有不同的权值集合。重新训练的最大意义在于自动地平衡突触权值使得那些产生最多误差的神经元沉寂或是使它们的影响得到降低。最终，对一个给定的目标误差，一个经过重新训练的神经网络能够比未经过重训练的神经网络容忍更多的逻辑最小化，并且不会减小加速器的应用范围。

INEXACTLOGICMINIMIZATION (Graph $\mathcal{G}$, Significance Array $\mathcal{S}$, Error Resilience Er_{th})

Step I: $\mathbf{k} \leftarrow$ NODECLUSTER ($\mathcal{G}$);

// Input nodes between 3-5 and output nodes between 2-3 were found sufficient for most of building blocks in the datapath circuits

Step II: *foreach* $k_i \in \mathbf{k}$ do

$\mathbf{ErCfg}_i \leftarrow \{\phi\}$

while ($k_i \neq$ wire) do

$\mathbf{ErCfg}_i \leftarrow$ GREEDYCUBESWAP(k_i)

// An example of this exploration done on a (3 input, 2 output) full adder node is shown in Figure 5.2 and $|\mathbf{ErCfg}| = 6$.

Step III: $\mathcal{G}_{min} \leftarrow \mathcal{G}$

while *iteration* < bound **repeat**

$\mathcal{G}' \leftarrow$ RANDOMERRCFG ($\mathbf{k}$, $\mathbf{ErCfg}$, $\mathcal{S}$)

s.t. if $s_m > s_n$, *then* $ErCfg_m < ErCfg_n, \forall$ *nodes* $m, n \in \mathbf{k}$

if COST($\mathcal{G}'$) < COST($\mathcal{G}_{min}$) && $Er(\mathcal{G}') < Er_{th}$

$\mathcal{G}_{min} \leftarrow \mathcal{G}'$

return $\mathcal{G}_{min}$

图 5.3 非精确基本单元的设计算法

5.2.3 设计空间搜索的敏感性和复杂度

全部的非精确神经网络的搜索空间是巨大的。我们需要考虑的所有的参数会产生1.07×10^{68}种不同的网络结构。考虑到我们需要训练每个网络结构来评估精度，评估每个设计点是极耗时的，平均大约要100s。因此不仅彻底的搜索是不可能的，即使随机地评估一个极小部分例如设计空间的2.95×10^{-61}%都需要一年。同时随意地将搜索范围限制到设计空间的一个极小子集上将要面临出现非最优解决方案的风险。

我们通过分析神经网络的特性来解决这个问题。我们之前及图 3.3中提到过，所有的网络运算器都可以分为加法器和乘法器。尽管任意的加法器或乘法器都有可能合适做非精确逻辑最小化，但它们对代价（能源、面积）和精度的影响是不同的。

在每个神经元中，将权值与输出得到的乘积求和的加法器（见图 3.3）比任何一个单独的权值乘法器对误差更敏感，因为一个非精确乘法器只是加和的许多项之一。同时那个加法器的开销通常远小于一个乘法器。结果，最小化这个加法器后得到的面积

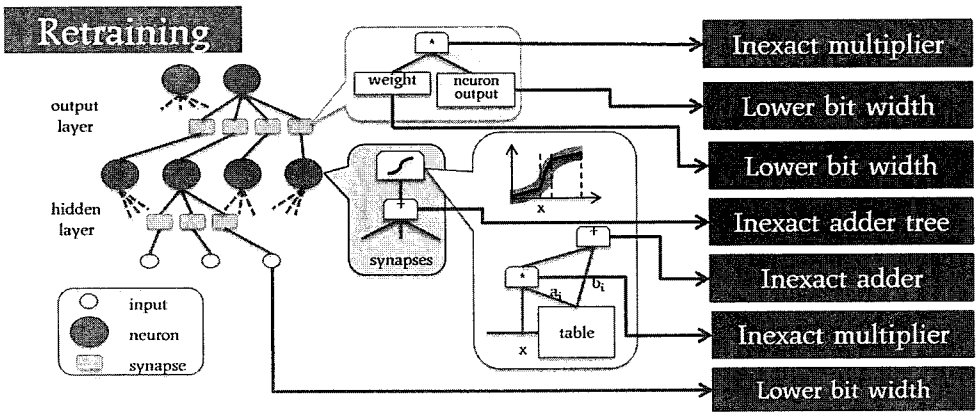

图 5.4　非精确神经网络设计空间

和能源效益很小，但它对神经元结果的影响非常大。类似地，激活函数的加法器和乘法器也是敏感的，而且能够提供的能源节约不超过一个乘法器。因此，非精确逻辑最小化的效益最大的目标是突触权值的乘法器。

通过更多地关注隐层而不是输出层使得搜索被进一步减少。尽管训练重新决定了每个输出的地位从而提供了一定程度的误差恢复力，但是由于输出层的神经元之后没有突触权值，想要通过训练算法移除这些神经元的误差就更加困难了。幸运的是，大部分神经网络需要巨大的隐含层，而输出层的神经元数目通常都很小（如果用来近似函数就为1，否则就是分类问题的类的数目），所以我们又把搜索偏向隐含层。总的来说，神经网络的最敏感部分幸好是那些最不可能带来最大能源节约的部分。

我们在图 5.5中描述了我们的搜索算法。位宽和隐含层中的非精确形式的乘法器的非精确类型是在搜索中主要考虑的结构参数。对于小的设计空间（例如对称搜索），我们使用全搜索，而对于巨大的设计空间，我们使用模拟退火方法去寻找一个近似的最优结构。

5.3　性能评估

在这一节，我们会描述方法、搜索的设计空间以及怎样利用非精确硬件神经网络实现能源和精度的折中。

5.3.1　方法

工具链　我们利用章节 5.2.1中的准则，在标准的带有校正常数的n位截断乘法器的基础上，探索出了7个非精确乘法器构成的集合 [97]；一个n位的截断乘法器有n位的输入和n位的输出，并且它在一些输出值非常大的情况下呈现出了少量的误差见表格 5.1中的16位截断乘法器。表格 5.1中展示了这7个非精确乘法器的特征，它们对精度和能效有不同的权衡结果。在这些乘法器中用到的全加器的一个子集和它们的特性在之前的图 5.2中已经展示了。

```
INEXACTCONFIGURATIONSEXPLORATION (Multiplier M, Inexact Type I, Bitwidth
B, Error Resilience Er_th)
//Multiplier m_{i,j}: multiplier type I_{i,j}, Bitwidth B_{i,j})

//full exploration or Simulated Anneanling
function: select()
Case: Full Exploration
    Add m_{i,j} ∈ M to M'
Case: Simulated Anneanling
    for selected numbers < Bound
        Add m_{i,j} ∈ M to M'
function: generate() //change to new configuration
foreach m_{i,j} ∈ M':
    Assign new value to i_{i,j} and b_{i,j} of m_{i,j}

exploration: retrain and test
    Case: Full Exploration
        while configuration < Bound
            M'=select(M); generate(M',M)
    Case: Simulated Anneanling
        while Erget < Erth and k < K repeat:
            M_new=generate(select(M),M)
            if Evaluate(M_new, M) > random() && test(M_new) < Erth
            M=M_new
```

图 5.5 非精确配置设计空间搜索算法

这些非精确乘法器已经被实现成了插入C++神经网络模型的C++子程序。这个模型是基于NNlib神经网络库建立的，这个库同时附带了BenchNN参考基准 [28]。NNlib实现了各种类型的神经网络，比如CNN、Hopfield网络和MLP，这些都可用反向传播算法进行训练。对于神经元的输入和突触权值的乘法运算，为了实现对某一个非精确乘法器的访问替代对标准乘法器的访问，我们对MLP的实现做了修改。修改后的神经网络软件模型用来评估重训练前后非精确乘法器对神经网络精度的影响。

我们也用可配置的Verilog硬件描述性语言实现了这些非精确乘法器和整个神经网络，目的是为了评估神经网络的耗能、延迟和面积。我们用Synopsys Design

表 5.1 非精确16-bit乘法器

Type	Rel.Err (%)	Area (um^2)	Power (uW)	Delay (ns)
#0 (裁剪)	0.09	2068.20	931.41	11.47
#1 (非精确)	0.99	1883.52	725.50	10.46
#2 (非精确)	5.23	1642.32	558.70	8.48
#3 (非精确)	10.70	1537.92	494.59	7.99
#4 (非精确)	17.16	1520.64	468.02	8.84
#5 (非精确)	27.51	1384.92	380.86	7.03
#6 (非精确)	41.65	1298.88	370.81	7.03
#7 (非精确)	98.94	1249.92	364.10	7.03

Compiler和Taiwan Semiconductor Manufacturing Company (TSMC) 65nm standard-V_{th}工艺库来综合神经网络，用Synopsys IC compiler来做布局布线的工作，用Synopsys VCS模拟并且用PrimeTime PX来评估功耗。

测试集和神经网络规模 参考基准和神经网络尺寸：我们用UCI机器学习库中的任务来测试神经网络的精度 [9]。学习库中的任务案例是由不同领域的研究员和工程师提供的，用来促进他们对机器学习应用的研究，所以它们的特性和尺寸随着实际应用的不同，差异也非常大。在 [187]中展示了一个包含90个输入、10个隐层神经元和输出的神经网络可以完成UCI库中90%的任务并且一样能达到目前最好的分类结果。我们实现了同样结构的神经网络并且采用了 [187]中的10个参考基准中的7个，舍弃了训练时间过长的3个数据集。我们使用的参考基准测试集包括：*glass, ionosphere, iris, robot, sonar, vehicle, wine*。我们在一个256位单指令多数据流的机器（(Intel Xeon E7-8830, 2.13 GHz, 1 TB memory）上训练这些基准数据集。我们用GNU compiler collection 4.4.7和参数"-O3 -lm -march=native"来编译，同时使用SIMD指令例如MMX、streaming SIMD extensions (SSE)、SSE2、SSE4.1和SSE4.2。实验结果表明同一个结构下对不同的基准数据集的训练时间从几毫秒（*iris*）到10min（*optdigits*）长短不一（几何平均值大约是100s）。注意到探索*optdigits*的10000个配置需要大约70天。

神经网络的误差通常用分类的MSE来表示，分类错误率是错误分类的输入的数目与所有的输入的数目之比。在本章的研究中我们使用MSE作为误差的度量方式。

神经网络加速器 我们针对UCI库实现了一个包含90个输入神经元、10个隐含层神经元和10个输出神经元的神经网络。这个实现很直接，一个物理神经元就是当作一个逻辑神经元来计算。如果神经网络的大小和加速器的大小不匹配，那么没有用到的神经元将失效并且对之后的神经元没有作用。

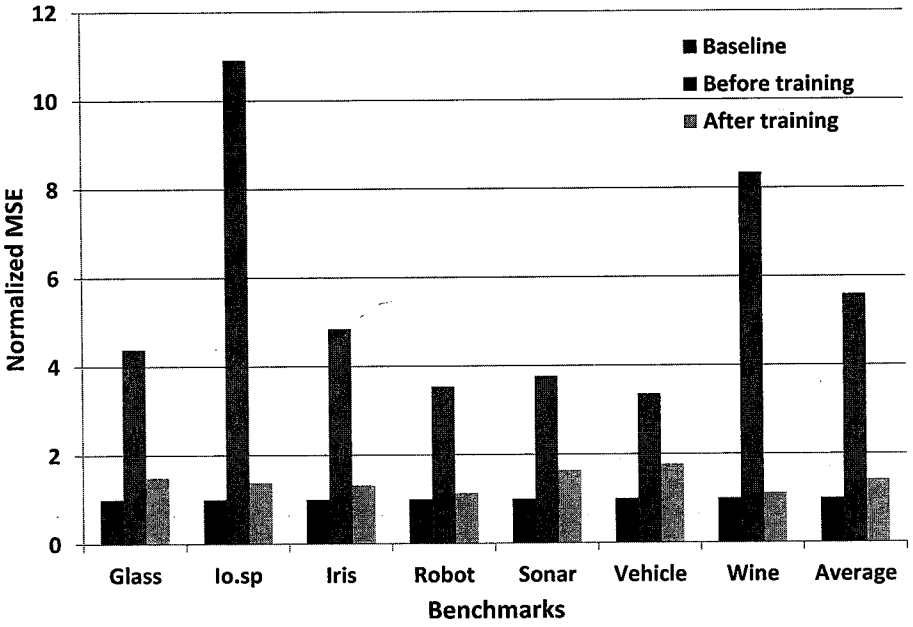

图 5.6 非精确运算符对误差的影响

5.3.2 非精确硬件神经网络的能耗，面积和MSE

尽管我们将会在章节 5.3.6中详细讨论我们解决方案的探索过程和特性，这里我们还是要从能耗、面积和MSE三个方面比较一下最后探索得到的最好的硬件结构和基准的神经网络，即一个使用16位定点运算器的精确神经网络。

在图 5.6中，我们比较了基准结构和训练前后的最好的结构（“训练前”指的是突触权值用的是基准结构）的MSE。训练前的MSE说明非精确运算器使得精度损失非常大。这么大的精度损失对许多其它的ASIC来说是不可接受的。神经网络的一大特性就是它们能够训练，因此像之前解释过的那样可以补偿由非精确运算器带来的误差。训练后的MSE(0.20)和基准的MSE(0.14)非常接近，即非精确运算器对精度的影响几乎不可见。

同时，经过非精确方案的配置和规划后的能耗和面积的节约量的结果明显，见图 5.7中的*exact NN pipelined*和*inexact NN pipelined*，non-pipelined的版本会在后续的段落中讨论。相比这7个参考基准，能耗节约量至少有43.82%，最大有62.55%（平均有54.54%）。需要的电路面积从4.23mm^2降低到2.90mm^2，即降低了31.51%的面积开销。

流水的开销和收益 到目前为止展示的都是神经网络加速器的流水线版本的结果。每一层（隐含层和输出层）对应一个流水线阶段。尽管神经网络可以用一个纯粹的组合电路实现，在将来的SoC和高性能异构多核的集成中吞吐率是非常重要的。即使前面提到的流水线的程度不低，由于高度并行的设计，神经网络加速器能够用少于10ns

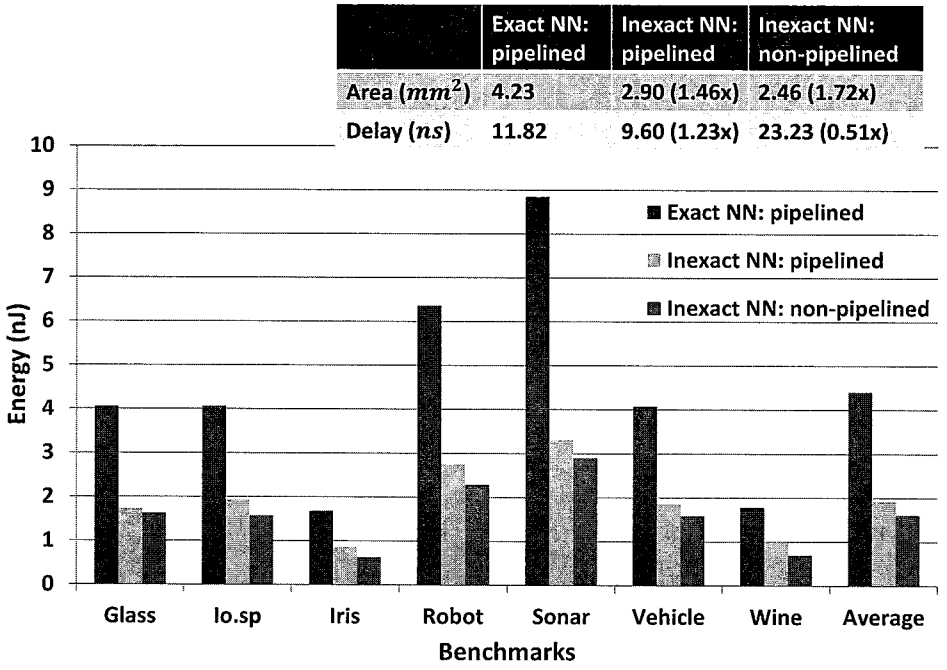

图 5.7 非精确神经网络（非流水设计和流水设计，non-pipelined and pipelined）和精确神经网络（流水设计，pipelined）的能耗、延时和面积比较

（9.60 ns）处理更密集的层。例如，在隐含层中每个神经元（10个这样的神经元并行地计算）在乘法输出之后并行地执行90个乘法计算，得到的结果传给sigmoid(一个加法和乘法)。由于每个算术运算器能够自己流水（乘法器和加法器），如果需要的话即使是达到更高的吞吐率也是可能的。

在图 5.7中，我们也比较了流水线版本和非流水线版本的非精确硬件神经网络。我们发现流水技术只增加了0.18倍的面积，能耗从1.06倍增加到1.42倍，但同时非流水线延迟是其2.42倍（从23.23 ns到9.60 ns)。在流水线版本中，时钟和寄存器承担了总能耗的7%-24%（平均14%），大部分的能耗还是由组合逻辑产生的。

5.3.3 非精确逻辑最小化 vs. 截断

在这一节我们来分析在更标准的位宽截断方法下的非精确乘法器的影响。在我们的探索中，位宽截断主要体现在降低运算器的大小，从16位减小到8位。

在图 5.2中，我们报告了每一种结构的结果，总共有3种：（1）由探索出的解决方案提供的最优*(best)*的结构，具体将在第三节中详细描述；（2）只是用精确乘法器的位宽*(bit-width)*结构，但运算器的位宽在标准的搜索过程中可以自由改变；（3）基准结构*(baseline)*，即一个采用16位定点运算器的精确神经网络；（4）对称式*(symmetric)*的结构将在之后的第三节中讨论。

我们发现改变运算器的位宽相比于基准在能耗和面积上的提升很小。加入非精确乘法器带来了更大的提升，且对精度的影响很小。

表 5.2 基准结构（baseline），最优结构（best），位宽结构（bit-width）和对称式结构（symmetric）的硬件特性

-	Symmetric	Best	Bit-Width	Baseline
Area (mm^2)	1.81	1.73	2.24	2.61
Delay (ns)	24.35	24.97	26.99	29.89
Power (W)	1.04	1.02	1.38	1.48
Energy (nJ)	28.19	25.40	37.15	44.34
Average MSE	0.17	0.20	0.15	0.14

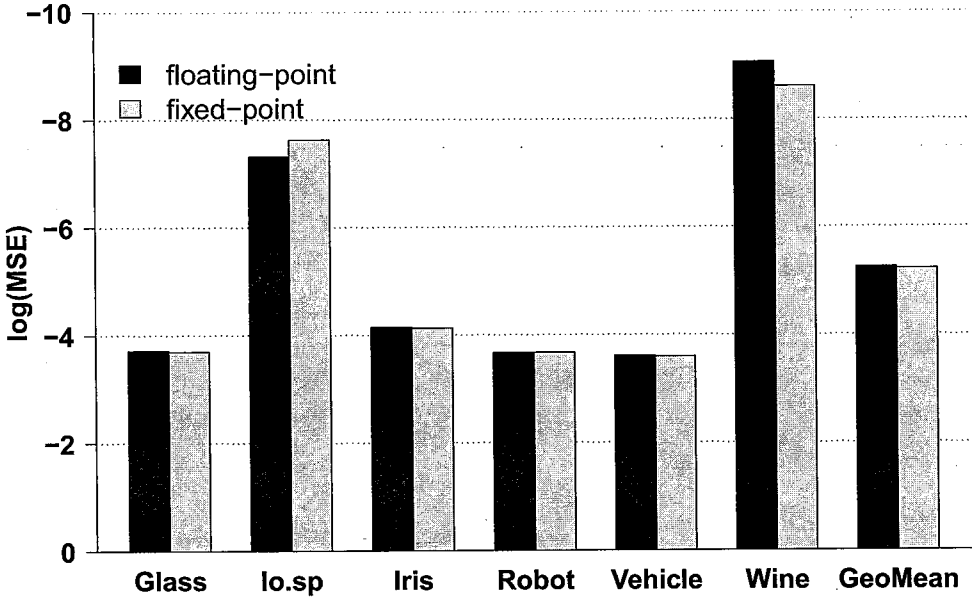

图 5.8 32-bit浮点数 vs. 16-bit定点数在UCI基准测试集上的精度

5.3.4 神经网络中的浮点运算器与定点运算器相比较

表 5.3 乘法器特性

Type	Area (μm^2)	Power (μW)
16-bit截断阵列乘法器	2068.20	931.41
32-bit浮点乘法器	7997.76	4229.60

在这一节中，我们将证明用定点运算器代替浮点运算器几乎没什么影响。虽然结论很惊人，但在学术著作中已经有充足的证据表明即使是更小的逻辑运算器对神经网络的精度也几乎没有影响[49,82,104]。在UCI机器学习库的神经网络的训练和测试中，我们使用32位浮点运算器或是16位定点运算器，并且采用10折交叉验证。32位的浮点数据形式是IEEE Standard Floating-Point Arithmetic (IEEE 754)。对于定点运算器，6位用来作为整数部分，10位作为小数部分。结果见图 5.8，可以确定这样设计对精度的影响很小。我们重点关注能带来最多节约量并且对错误率影响最小的乘法器（见

章节5.2.3)。我们在表格 5.3中展示了乘法器的硬件特性。相比于一个32位的浮点乘法器，截断的16位序列乘法器的面积为其25.86%，功耗为其22.02%。在章节 5.3.1中可以看到CAD工具方法。

5.3.5 位宽的影响

表 5.4 神经网络中位宽参数

Parameters	Minimal	Maximum	Step
Sigmoid	8	16	4
Input	8	18	4
Weight (8 bits)	0%	100%	20%
Weight (12 bits)	0%	100%	20%
Weight (16 bits)	0%	100%	20%

现在我们尽可能多地减小运算器的位宽，评估它们对精度的影响。如图 5.9所示，我们探究了不同的“配置”。这里的一个配置指的是神经网络的不同部分的位宽：每一层的输入神经元、突触权值等。我们按字典顺序逐一审视这些不同的位宽值，例如在某一层中增加突触权值的宽度同时保持其余所有的参数不变，直到达到了最大值，增大另外一个参数并重新设置那一层的突触权值宽度到最小值，这样就得到了图 5.9中的波状图。在表格 5.4中，我们展示了一个结构的确定形式和每个参数的变化范围。表格5.4中的各个参数：sigmoid指的是每一层中每个神经元的位宽，它包含了章节2.1.1.1中提到的一个加法器和一个乘法器；input指的是输入层中每个神经元的输入长度；weight with percentage指的是在所有的权值中这些宽度长度所占的比例。对表 5.4中列出的参数的每一种可能组合，我们为下一步随机地产生了100个固定的网络结构。我们在UCI基准集上重新训练每一个结构。结果与之前的神经网络对运算器和存储位宽的敏感性的发现一致[104]。

表 5.5 重训练后的MSE

Apps	Floating.Max	Floating.Avg	Fixed.Max	Fixed.Avg
GLASS	2.403	1.289	2.567	1.271
IO.SP	329.982	56.993	413.421	53.573
IRIS	4.843	2.021	5.151	1.972
ROBOT	3.095	1.539	3.148	1.547
VEHICLE	1.573	1.167	1.432	1.148
WINE	23.592	7.174	32.8285	5.996

位宽和重训练 我们现在想展示重训练对精度的影响。基于这一目的，我们重训练和测试了章节 5.3.5中探索出的所有结构，结果见表 5.5和图 5.10。不同位宽结构的重

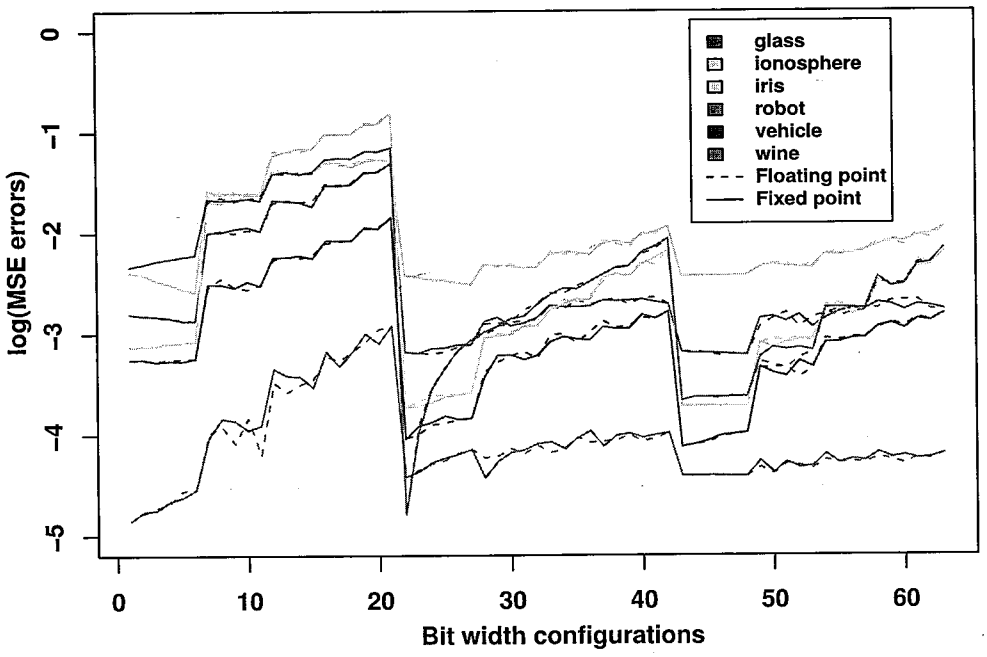

图 5.9 浮点运算符和定点运算符的位宽探索

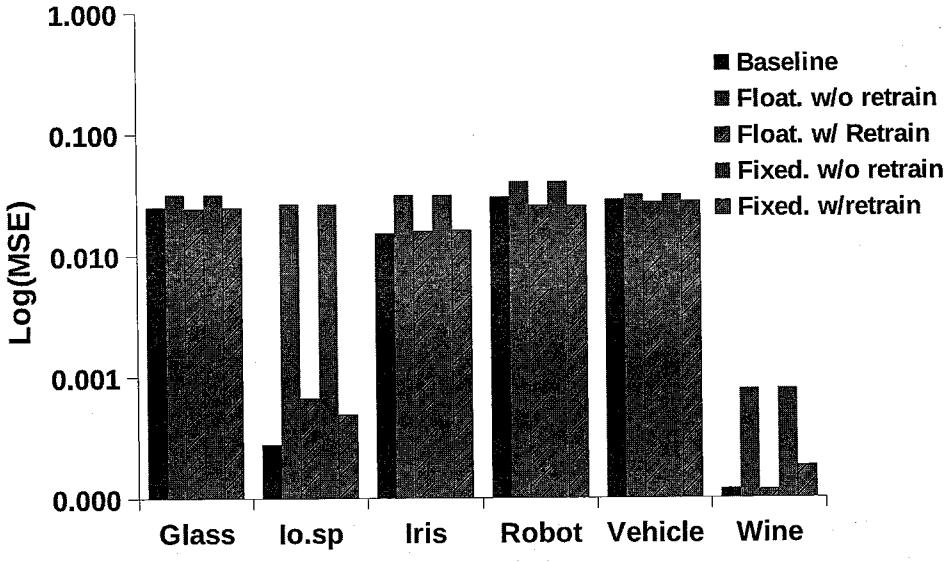

图 5.10 位宽的影响和重训练的效果

训练的平均结果如图 5.10所示。我们把使用浮点运算器的精确神经网络作为参考基准。可以发现不管我们是否减少定浮点运算器的宽度，重训练后的MSE与参考基准非常接近。另一方面在没有重训练的情况下，定点运算器的MSE增加到1.17倍到56.99倍（最高有329.98倍），浮点运算器增加到1.15倍到53.57倍（最高有413.42倍）(见表格 5.5)。表格 5.5中通过给出最大的和平均的MSE的减少量来比较重训练前后影响的特征。结果体现出了神经网络强大的错误恢复能力。注意到重训练过程只用了非精确硬件中（前馈通路），如果有实现好的精确训练电路它仍然能够在在线学习时有很好的效果。

5.3.6 设计空间搜索

在表格 5.6中我们描述了探索所有参数的范围和步骤。我们改变隐层和输出层中所有乘法器的位宽和形式，突触和sigmoid都改变，但我们只改变隐层的突触乘法器的结构，具体分析见章节5.2.3。尽管基准结构用的是16位的运算器，我们仍然要考虑的位宽分别为8、10、12、14、16，位宽低于8的会产生很差的MSE。根据我们在章节5.3.1和章节5.2.3中的解释，网络中的任意乘法器都可以用8种乘法器（7个非精确和1个精确）替换。因此我们总共探索了24500种结构。

表 5.6 设计空间相关参数

Parameters	Minimal	Maximum	Step
b/w of synapses	8	16	2
b/w of sigmoid	8	16	2
multiplier type	0	7	1

对每一种结构，我们在所有的7个基准集上用C++神经网络模拟器测试MSE。我们分别地在每个基准集上用10折交叉验证法训练神经网络（输入集合被分为10个子集合，在其中9个上训练用剩下的集合测试，重复这样的过程直到10种可能的集合组合都训练完）。对每个基准集，我们重复实验5次来规避统计的不确定性（神经网络的初始权值是随机确定的），并且对5次的MSE结果做平均。

我们接着保留那些MSE比精确神经网络小4倍以上的结构，通过单独的运算器（乘法器、加法器、寄存器等）的设置和规划的结果来得到面积、延迟和功耗的一个近似估计。然后我们通过设置和规划整个硬件神经网络来确定最优方案的面积、功耗和延迟。

在图 5.18中，我们展示了探索出的方案的一个子集合（它们的MSE比基准MSE低两倍以上），纵观所有的基准集它们有最低的MSE和面积，所以是最好的。X轴代表的是面积，y轴代表的是相对MSE$\left(\frac{\text{MSE}-\text{MSE}_{\text{exact}}}{\text{MSE}_{\text{exact}}}\right)$。显然有很多不同的方案集合，并且有低MSE、低面积的好的权衡结果。从图 5.11到图 5.17，我们说明了对所有基准集的探索的过程。我们发现对每个单独的基准集来说，例如*robot*和*vehicle*，都有比平均的最优方案更好的方案，这表明具有更专一应用的神经网络加速器的效果更好。

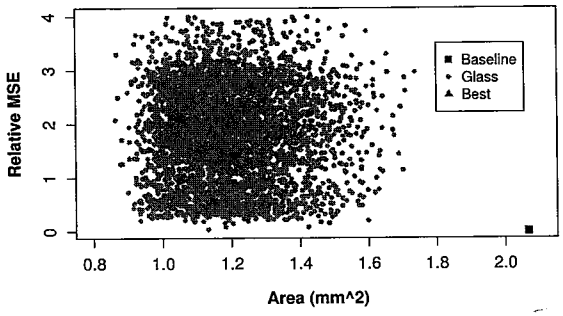

图 5.11 数据集*glass*配置搜索

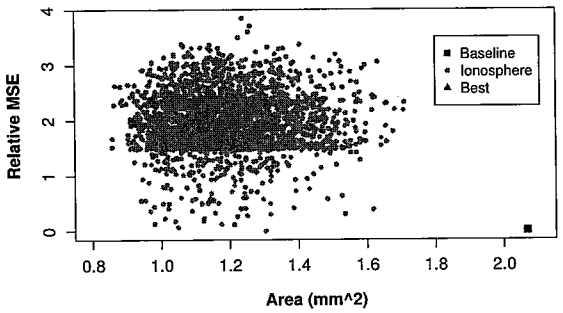

图 5.12 数据集*ionosphere*配置搜索

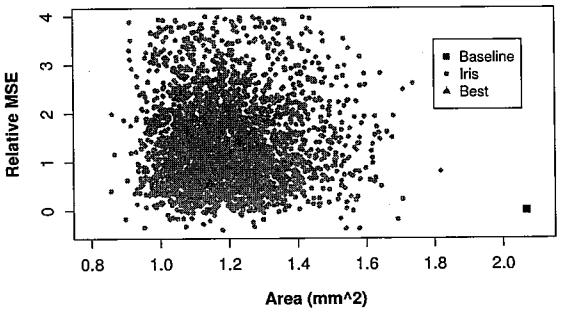

图 5.13 数据集*iris*配置搜索

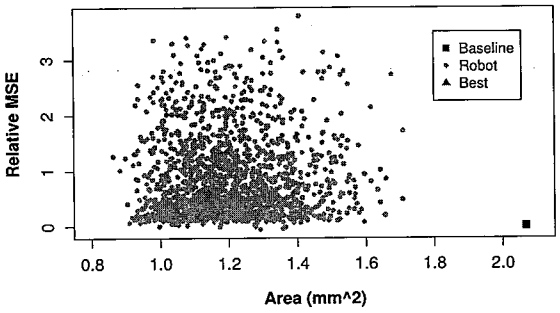

图 5.14 数据集*robot*配置搜索

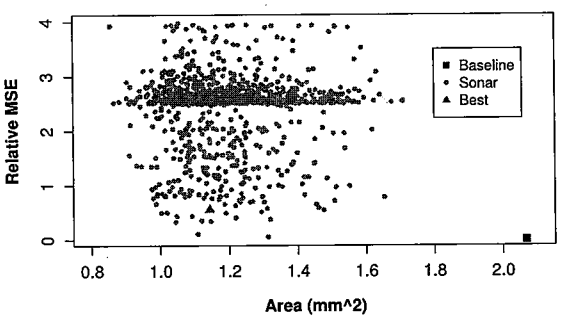

图 5.15 数据集*sonar*配置搜索

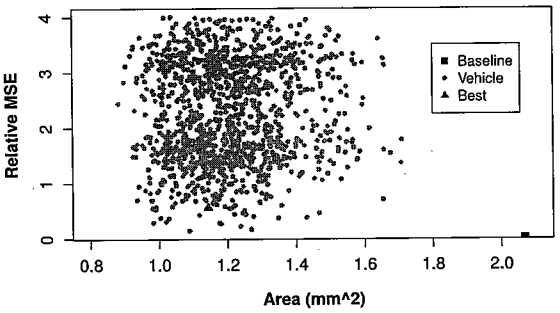

图 5.16 数据集*vehicle*配置搜索

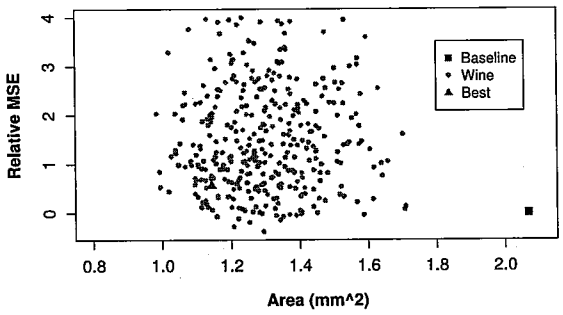

图 5.17 数据集*wine*配置搜索

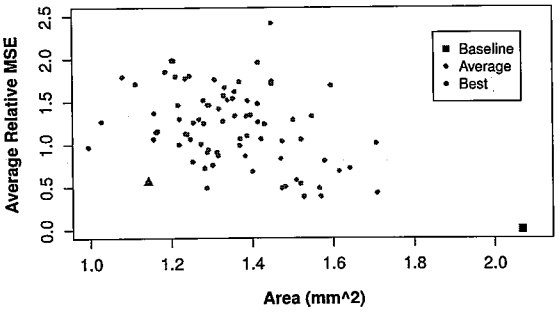

图 5.18 所有数据上的配置搜索

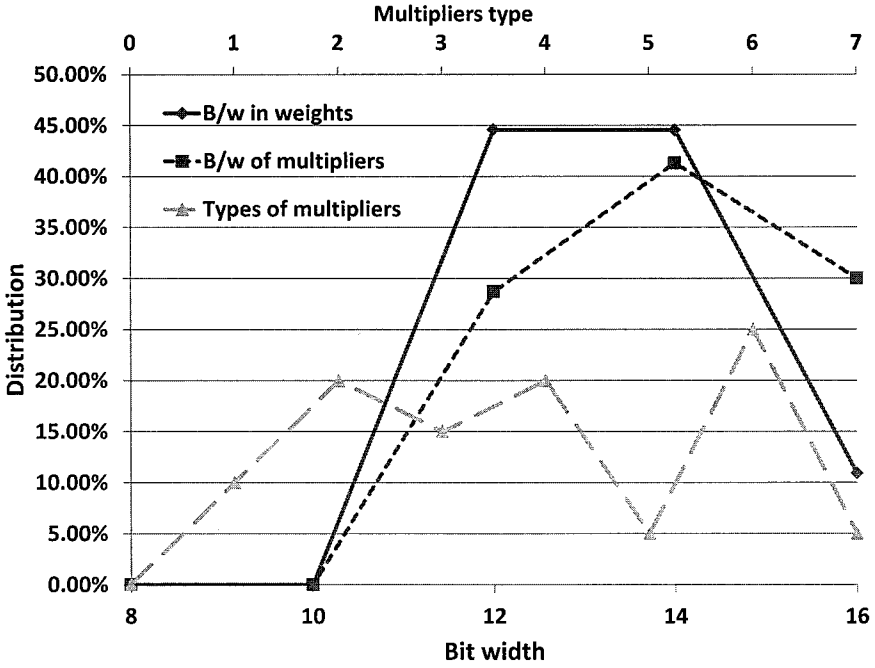

图 5.19　最终配置的位宽和乘法器类型分布

我们找出方案中按面积与MSE的帕累托最优标记为最优解决方案（最低的MSE,最小的面积）。我们选择最接近面积与MSE折中的结构。通常地，我们可以根据加速器的目标面积或MSE来选择其它的结构，即更小面积或更低MSE的帕累托最优。

在图 5.19中，我们更详细地分析了最优权衡的结构，我们提供了在神经网络结构中发现的突触寄存器位宽、乘法器位宽和乘法器种类的分布。其中没有8位和10位的权值和乘法器，44.56%的权值位宽为12位，还有44.56%用的是14位，剩下的是16位的权值。乘法器中，41.30%用的是14位，28.70%用的是12位，30%用的是16位。所有种类的非精确乘法器都在最后的结构中用到了，尤其是种类2、4和6，但种类0（标准截断乘法器）没被用到。有趣的是，输出层中所有的乘法器都是16位，符合我们在章节5.2.3中的分析，即尽管训练重新决定了每个输出的地位从而提供了一定程度的误差恢复力，但是由于输出层的神经元之后没有突触权值，想要通过训练算法移除这些神经元的误差就更加困难了。

非对称式分配非精确运算符　到目前为止，我们都是允许每个突触乘法器使用一个不同种类的乘法器。为了更大程度上地减少搜索的代价，我们这里探究一下乘法器的对称分配方案，即当所有的突触乘法器都是独立的，是否会很明显地影响探索得到的方案的质量。我们将我们的最优方案与对称搜索的最优方案相比较，见表格5.2中的*symmetric*。我们发现对称性的方案的质量和最优方案非常接近，这表明可以通过只考虑对称结构（所有的乘法器是独立的）来极大地降低设计空间的大小。

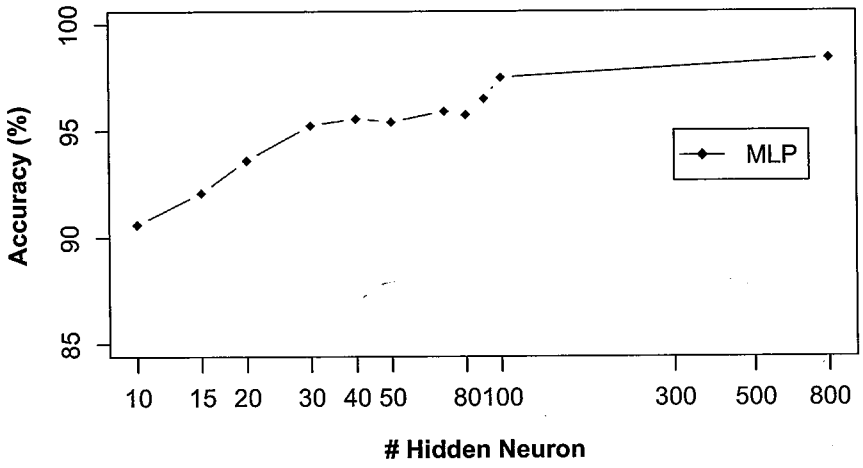

图 5.20 MNIST数据上不同隐层神经元数目对精度的影响

5.3.7 在可编程式神经网络加速器上的应用

上面所有的分析都是基于固定功用的空间扩展设计，即每个物理神经元硬件对应一个逻辑神经元。然而有一些非常大的神经网络，因其规模太大而不能进行空间扩展。对目标是MNIST数据集 [109]的784-100-10神经网络来说，据我们估计需要消耗74mm^2（使用TSMC 65nm技术），在许多嵌入式场景中这几乎是不现实的。因此我们之前的成果，一个可编程的神经网络加速器DianNao [29]，它使用了空间折叠使得可以在很小的神经元功能部件（NFU）上容纳任意大的神经网络。在这一节中我们探究非精确方法能否成功地被应用在这样一个可编程的加速器上。考虑到在大规模神经网络上过长的训练时间，我们决定做对称性搜索，由于非精确运算器的对称分配的质量和我们探索的最优结果非常接近，这一点之前也提过了。

神经网络拓扑结构 对MNIST数据集，最好的结果是由一个多列的深度神经网络（MCDNN [34]）得到的精度为99.77%。然而，考虑到训练和重训练的时间代价，我们选择一个适当大小的MLP就已经能得到很好的识别结果了，参见图 5.20。我们探索不同的神经元数目来寻找隐层神经元的合适数目。在之后的试验中我们选择使用100个隐层神经元。

神经功能单元（Neural Functional Unit，NFU） 在本章的研究中，我们重点关注DianNao的计算硬件即NFU的错误恢复力。DianNao的NFU在乘法、加法和激活函数上分别有一个三级流水线。乘法器和加法器用16个物理硬件神经元（$Tn = 16$）构成,每个神经元同时计算16个输入（$Ti = 16$）,即有256个乘法器和16个加法树。注意到流水线是交错的，因为只有前两级的所有输入的计算结束，才会使用第三级（即激活函

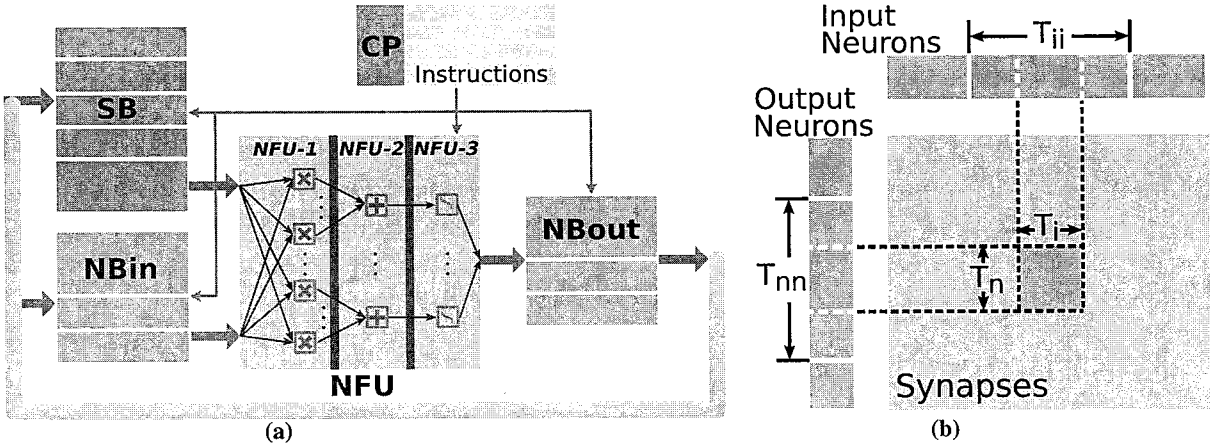

图 5.21 （a）DianNao中的NFU （b）DianNao上调度运行MLP

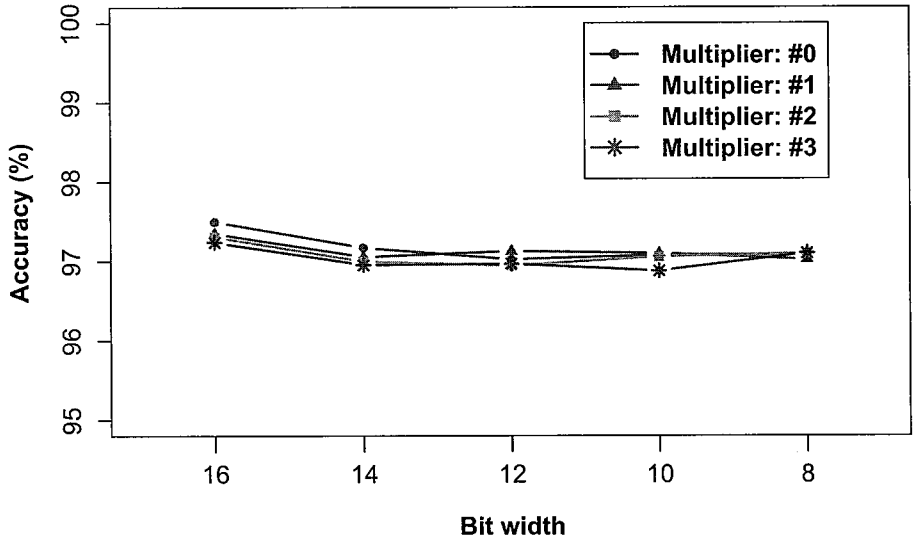

图 5.22 不同非精确配置下MNIST的精度

数）。具体的NFU结构和MLP调度方案如图 5.21所示。注意到神经网络的每一层将被映射到同一个NFU（芯片上实现的唯一的NFU）。我们基于65nm技术实现了精确和非精确版本的DianNao。精确与非精确NFU的面积、功耗和功率数据如表格 5.1所示。非精确NFU的具体结构是在对设计空间彻底地搜索后选择的，像之前在表格 5.6中提到过的那样。这里，我们使用了一种对称性方法（即将所有的乘法器限定为一样的）来减小设计空间。

精度 对每个非精确NFU的结构，我们分别训练5次神经网络，并且将5次试验的精度结果取平均，我们在图 5.22中报告了一些结构的精度。当结构是16位到8位的宽度，类型#0~#3的乘法器，非精确NFU的识别精度损失总是不超过1%，这几乎是可以忽略的[①]。即使是8位、类型3乘法器的结构，非精确NFU仍然能达到97.10%的精度（只

① 我们注意到使用少于8-bit的位宽或者类型#4~#7的非精确乘法器会使得精度大幅度下降。

比精确NFU的精度97.65%低了0.55%)。这么理想的结果证明了非精确神经网络在可编程加速器上应用的可行性。

表 5.7 精确NFU和非精确NFU的硬件特性

	Exact NFU	Inexact NFU
Area (mm^2)	0.66	0.28
Power (mW)	66.37	32.86
Energy (nJ)	56.91	28.18

硬件特性 精确、非精确NFU的硬件特性参见表格 5.7，它们都限制在2.5ns的时钟上（400MHz)。非精确NFU采用8位、类型3的乘法器实现，其它的部分（例如加法树）对应地做最优化设计。如表 5.7所示，非精确NFU面积是精确NFU的41.99%，功耗是其49.51%。

总的来说，我们的实验结果表明，一个非精确NFU可以在实现42%的面积和50%能耗节约的同时仍然达到97.10%的精度（与精确NFU相比精度损失不超过1%)。这理想的结果证明了非精确神经网络应用在能够承载大规模神经网络的可编程的加速器上的可行性。

5.4 相关工作

用牺牲精度的方式换取电路的高能效的理论基础在十年前就已经被提出 [151]，从那以后，关于应用这种方式到更多的应用程序中，尤其是那些输出质量是由人类主观认识进行判断的应用程序的研究就已经层出不穷 [150]。大多数此类的研究都与ASIC电路相关，也有扩展到通用处理器的尝试。然而，处理器中的控制电路十分敏感，容不得半点错误，使得这样的扩展受到了极大的限制 [58]。Mahdiani等人[125]尝试用非精确的加法器和乘法器实现fuzzy逻辑和神经网络，然而他们串行的在单个硬件神经元上执行相关的运算，这样不单单是丧失了高性能，也限制了神经网络本身能够获得的收益，而且使得电路也更加脆弱。

早期关于错误容忍的研究多数集中在相关物理参数上（如电压）[24,31,32,72]，并将这样的参数调整作为手段来控制精度能耗的取舍。然而，最近在电路层面上的创新成为一个最具潜力的实现方式 [118,119,149]，这主要是因为电路层面上的创新不带来任何硬件开销。这点不同于电压调整中需要增加改变电压的模块，支持多个电压域下工作的电源，以及低电压下电路可能存在的亚稳态 [118,149]。

另外，从神经网络算法结构方面进行非精确设计的工作在近些年也逐渐增多，尤其是在深度神经网络中。Han等人[77]通过去除神经网络中小于特定阈值的权值连接从而大幅度降低网络中的权值数量，最高压缩率可以达到13倍。然而这样的

网络并没有对硬件实现带来收益，甚至为了利用修剪后网络的稀疏性，硬件需要添加专门的模块。Kim等人[96]提出了一种用于前向通路的MLP网络Bitwise Neural Networks（BNN），其中输入和权值都采用至多2bit表示，然而BNN需要使用更多的神经元，如对MNIST使用具有三层隐层的MLP（1568-1024-1024-1024-10），神经元数目是[52]中的28倍，权值数目是两千万倍（即使[52]中采用16bit定点数，权值存储大小仍有268万倍差距）。Rastegari等人[162,164]提出了二值化近似的卷积神经网络Binary-Weight-Networks和XNOR-networks，可以通过二值化表示权值从而大幅度的降低前向通路计算时的权值存储大小。这两种网络都可以大幅度的降低前向通路的硬件开销（硬件神经元中的运算可以被位操作代替），然而目前无论是Binary-Weight-Networks还是XNOR网络，实现的分类精度仍有较大差距，需要进一步提升。

同时发展于电路层面上进行错误容忍方面的研究，体系结构方面基于神经网络的相关研究也变得越来越吸引人 [59,187,192]。在异构多核情景下，硬件神经网络处于十字交叉路口：对于错误容忍和能效需求，对于拥有较广泛应用场景的需求[27]和对于目前机器学习领域内算法的进展[106]。这样的趋势也被IBM这样的公司确认，他们也正在研究硬件神经网络加速器[132]。

5.5 本章总结

本章节中，我们提出了应用非精确计算到拥有广泛应用范围的神经网络。特别的，我们利用神经网络以下3个方面：（1）目前许多高性能的应用可以完全用神经网络实现；（2）神经网络本身具有很高的容错性，这是基于其训练算法的特性；（3）非精确计算可以应用到电路结构层级使得其易于实现。在机器学习基准测试集上，我们说明如何应用非精确计算到神经网络来设计硬件加速器。加速器实现在65nm工艺下，相较于精确的神经网络加速器，能够在能耗方面节省42.82%-62.55%（延时和面积分别节省18.70%和31.51%）。另外，我们也将非精确计算应用到DianNao中并且得到非精确DianNao设计，获得了58.01%的面积节约和50.49%的能耗节约。另外，我们也提出了如何缩减设计非精确加速器中的设计空间，即通过对称式来设计非精确运算符。

第六章　结束语

6.1　本文总结

本文首先研究了来自不同领域神经网络算法硬件化的比较。从工业机器人、自动驾驶汽车到智能手机，大量的设备需要对现实世界捕获的信息（如图像、声音、无线电波等等）进行越来越复杂的处理。对于这些任务，硬件神经网络渐渐成为极具吸引力的备选结构。而这其中所实现的神经网络算法来自两个截然不同的领域，机器学习领域和神经生物学。这两类神经网络算法也因此具有极其不一样的特性，哪类神经网络算法更加适合实现成为硬件加速器也存在争议。目前为止，几乎没有从硬件角度比较这两类神经网络的研究。在本问题研究中，我们把两类神经网络算法实现成为硬件电路，并完成后端布局布线工作，然后比较这两类方法硬件相关的参数，如能耗，速度，面积开销，精度和功能性。对于目前的神经生物学所启发的神经网络和机器学习启发的神经网络，我们实现硬件加速器结构，并采用同样的应用进行比较。我们的研究结果表明，同样精度的需求下，神经生物学激励的神经网络（如SNN+STDP）的硬件开销显著大于机器学习启发的神经网络（如MLP+BP）。另外，我们也发现，对于具有非常大规模的神经网络和对于精度要求不高的应用，SNN+STDP相较于MLP+BP具有更少的硬件开销，即是更具吸引力的选择。同样的，对于SNN+STDP在精度上劣势，我们发现相对于脉冲编码的信息丢失，更多的原因是源于STDP训练算法本身。最后，我们也发现对于需要永久在线训练且对精度要求不高的应用，SNN+STDP硬件加速器更加高效。

本文接着提出了面向嵌入式和移动终端的CNN加速器。近年来，对识别和数据挖掘类的广泛应用，神经网络加速器已经证明既能够达到高能效，同时也能够达到高性能。尽管如此，此类加速器的能效和性能仍然受到内存访问的限制。在本问题研究中，我们专注于识别中最重要的应用类，即图像类应用和数据挖掘类应用。神经网络类算法中对于这些应用目前处在领先地位是卷积神经网络(Convolutional Neural Networks, CNN)。卷积神经网络具有一个重要的性质：突触连接的权重被许多神经元共享，这大大的减少了神经网络的内存占用。此属性允许片上SRAM容纳下整个CNN，从而消除掉所有对权重的内存DRAM访问。进一步，将此类加速器放置在图像传感器的旁边，它能够消除剩余的所有的内存DRAM访问，即输入和输出。在本问题研究中，我们提出了这样一个CNN加速器，并将其放置在一个CMOS或CCD传感器旁边从而消除掉内存DRAM的访存，加上精心设计的神经网络内部的数据访问模式，最终我们设计实现了一个高度节能的加速器，也即ShiDianNao。在10个代表性的测试

集上，ShiDianNao设计相较于CPU、GPU和重实现的DianNao [29]快50×，30×和1.87×。ShiDianNao的能耗是GPU和DianNao的3700分之一和54分之一。ShiDianNao在65nm工艺下的面积开销为5.94 mm^2，频率为1GHz，功耗为336.51 mW。正是由于高性能，低能耗，低面积开销，ShiDianNao能够适应在移动终端和可穿戴设别上的视觉处理应用。更进一步的，我们提出了ShiDianNao+在保持高能效性时提高性能。ShiDianNao适合集成在传感器上从而可以高效的本地处理相应的任务。这样能够大幅度的降低服务器端的工作负载，提高视觉处理的服务质量，最终能够为无处不在的视觉处理奠定基石。

本文最后研究了非精确电路的设计相关问题，并将非精确计算应用到硬件神经网络中。近些年来，对于许多能够容忍一定非精确度的应用程序来说，非精确计算（*inexact computing*）越来越引人瞩目，也被认为是降低能耗最有效的手段之一。通过牺牲可接受的实验精度来换取明显的资源节约(能耗，关键通路的延迟，硅面积)。在这一原则的驱使下，这种方法被限制在专用集成电路（ASIC)上。按照目前的设计，这些ASIC针对的应用范围有限，且对不精确度的耐受性往往是固定的不够灵活，然而后者决定了我们能够实现的硬件资源节约量。在这个问题的研究中，我们提出将非精确计算与硬件神经网络相结合来扩大应用范围、提高错误恢复能力和提高能源节约程度。这些神经网络正迅速成为未来异构多核平台的热门备选加速器，并且能够被重训练，这使得它们具有灵活的错误恢复程度。我们在65纳米工艺下的结果表明，相比于已有的神经网络实现基准，该非精确神经网络加速器能够在能耗方面节省42.82%-62.55%（延时和面积分别节省18.70%和31.51%），而精度损失的很小（均方误差MSE平均从0.14增加到0.20)。另外，我们也将非精确计算应用到DianNao中并且得到非精确DianNao设计，获得了58.01%的面积节约和50.49%的能耗节约。我们也提出了如何缩减设计非精确加速器中的设计空间，即通过对称式来设计非精确运算符。

6.2 未来

神经网络仍在如火如荼的发展着，进一步的研究工作可以从以下三个方面进行。首先，脉冲神经网络目前虽然在精度上并不具有和机器学习算法一样的水准，这一方面是由于学习算法，另一方面是由于信息编码，这既体现我们本身对于生物的认知不够深入，也体现出现在机器学习算法的强大。然而脉冲神经网络是更接近生物学上的现象和观察的模型，考虑到即使是果蝇这样只拥有几千个神经元的生物也能表现出绝对的智能性，长远来看我认为未来对于人工智能的突破应是从脉冲神经网络突破，而短期来看脉冲神经网络算法本身亟需进一步研究，若算法的进步促使脉冲神经网络具有和机器学习类神经网络一致的精度，则其在硬件方面的优势则得到凸显。其次，深度学习仍在不断的进步，目前网络的规模开始朝着更深但是参数更少的方向发展，如微软提出的残差网络和Standford提出的稀疏神经网络。这也体现出现在深度神经网络中参数的冗余性，可以预见的是后面的算法研究会进一步压缩冗余参数的存在空间，

从而网络可能具有更高的精度但是却拥有更少的参数，从而降低计算量。如何设计面向这样的深度神经网络加速器也成为一个具有挑战性的课题。最后，算法的进步不断的对加速器设计提出更多的挑战，不同的神经网络可能具有完全不同的特性，也存在组合不同神经网络搭建一个完整系统的可能性。为了尽可能的高效需要加速器设计尽可能专用，然而如何在同一个架构下设计支持不同神经网络的加速器也许最后会成一个不得不面对的挑战；或是需要容纳不同特性的技术来设计尽可能高效高性能的加速器，这样技术例如非精确计算、稀疏网络、权值压缩编码、自压缩等等。未来的发展仍然是充满不同的机遇和挑战，也是大有可为的。本文中的研究也是在这个方向进行了小小的探索，也为未来的发展奠定了一定的基础。