03 智能体系架构与芯片

清华大学张悠慧：软硬件去耦合的类脑计算系统设计与思考

整理：智源社区 王光华

张悠慧，清华大学计算机系研究员，博导，主要从事计算机体系结构，包括处理器设计、类脑计算芯片与基础软件、高性能计算等方向的研究； 曾担任多个国际 / 国内学术会议程序委员会委员及期刊编委； 在 NATURE、ASPLOS、MICRO、NIPS、DAC 等期刊 / 会议上发表论文近 70 篇； 是国家科技进步二等奖、国家级教学成果二等奖、教育部科技进步一等奖获得者。

软硬件去耦合是计算机系统结构中非常重要的设计方法论。简单来理解，即软件研发人员不需要考虑底层硬件如何设计； 而硬件开发人员则只需要遵循一定指令集规范，并不用担心兼容性，也不用考虑上层软件开发问题。多年以来，软硬件去耦合确保了通用计算机体系结构的快速发展。而之所以能够做到软硬件去耦合，其背后的原理便是“图灵机与计算理论”。

那么类脑计算，要超越甚至替代冯 · 诺依曼模型和传统计算机，是否也需要做到软硬件去耦合？该用什么方法实现？这些问题对类脑计算的研究和开发意义重大。

针对这些问题，在第二届北京智源大会“智能体系架构和芯片”专题论坛上，清华大学张悠慧做了题为“软硬件去耦合的类脑计算系统设计与思考”的专题报告。

张悠慧在演讲中首先介绍了类脑计算的起源、应用与发展等研究背景，并分析了以类脑芯片为核心的软硬件和基于“通用”类脑计算开发框架（语言）系统两种类脑计算研究的现状，以及探讨了如何从传统计算机的发展中汲取经验，采用软硬件去耦合、软硬件协同设计等多种方法论，来探索符合类脑计算与神经形态器件特色的设计方法论，最后，张悠慧简述了通用类脑计算当前研究的最新进展和下步工作重点。

一、研究背景：类脑计算的起源、应用与发展

类脑计算或者神经形态计算的历史比较悠远。神经形态计算的鼻祖之一，加州理工的 Carver Mead 教授在 1990 年提出“神经形态计算”这个词，并将其定义为“采用以模拟器件仿真生物神经系统的 VLSI 来实现大规模并行的自适应计算系统”[1]。

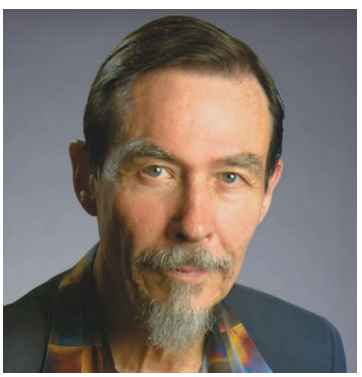


图 1：加州理工 Carver Mead 教授

但这个概念提出的初衷，并不是为了做 AI 或神经网络计算等，主要是为了追求计算效能。因为传统数字架构，一条指令执行时，需要经过译码、数据传输等一系列操作再进行计算，其“耗能不单是这个计算本身的，而是执行整个计算流程所消耗的总能量”，效能不高。而生物系统的计算效能比数字计算系统计算效能高几个数量级。所以，当时的有识之士，便开始借鉴生物神经网思想，用以提升计算效能。

这里主要包括两点： 一是使用基本的物理现象作为计算的原语，类似于模拟计算的方法。二是，采用类似 Spiking 这样的模拟相对值而不是数字的绝对值来进行信息表述。这两点是生物系统能效优势的根本原因，对于提升计算效能具有非常重要的启示。

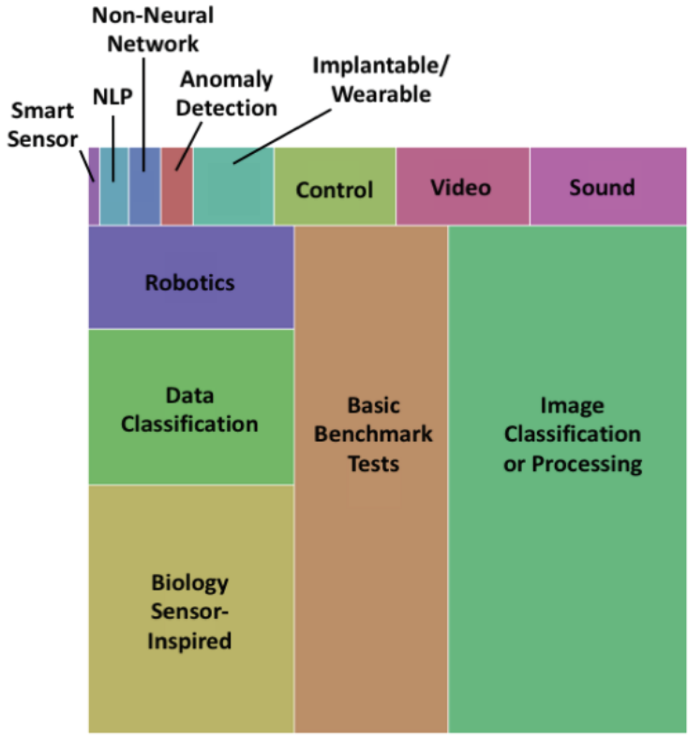


图 2：类脑计算所支持的各类应用 [2]

张悠慧介绍，类脑计算应用目前比较侧重于 AI。根据综述文章 [2]，类脑计算目前所支持的主要应用是神经网络形式的，但是也有越来越多的工作在探索利用类脑计算实现非神经网络应用。主要包括两种，一种是以神经网络的形式解决图计算、计算优化等非 AI 的问题； 另一种是问题形式、表述形式、解决形式等非神经网络形式的，比如利用脉冲神经网络构建通用计算框架。

但 2016 年的《Nature》文章提到： 类脑计算作为后摩尔时代极具潜力的发展方向之一，不能把类脑计算应用只局限于 AI，要放宽一点。

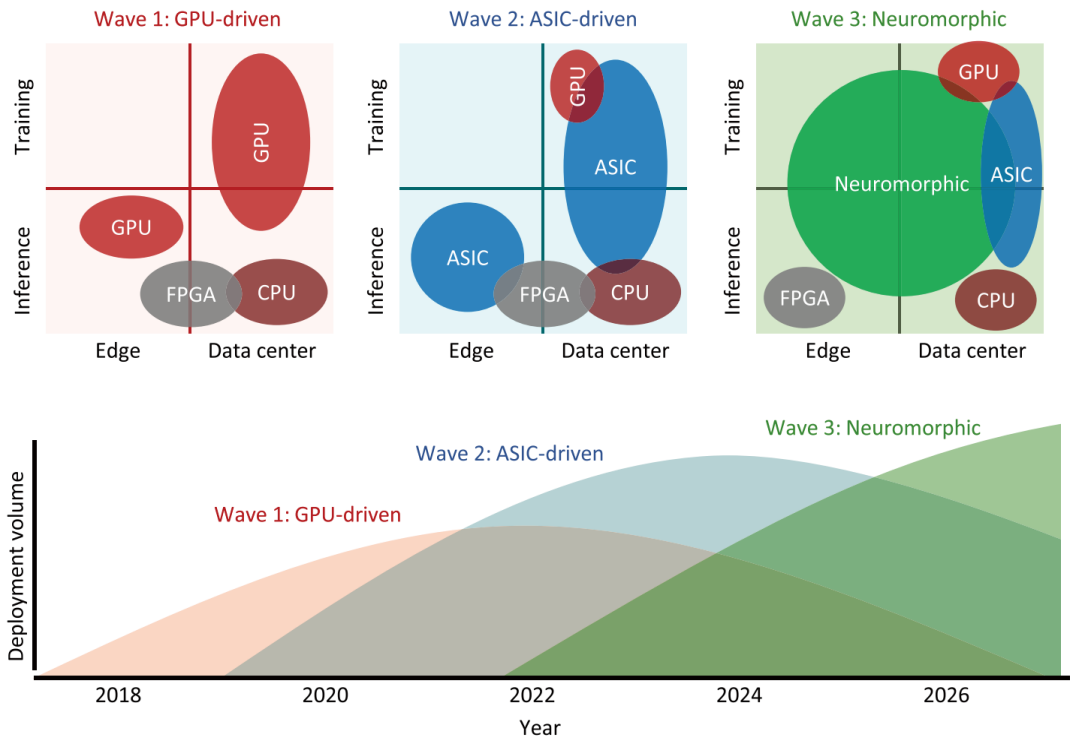


图 3：引领类脑计算的技术浪潮 [5]

那么类脑计算的未来会怎么样？ 2020 年初有一篇文章 [5] 指出，类脑计算可能引领下一波 Computer Engineering 的浪潮。目前，GPU 驱动了第一阶段浪潮，第二个阶段将是 ASIC 加速器，而第三阶段可能就是神经形态计算。图灵奖得主 John L. Hennessy 和 David A. Patterson 在 2018 年的图灵报告 [3] 里也提到，未来 10 年是重新定义计算机体系结构的 10 年。目前体系结构因发展路径的多样性，将带来体系基础结构创新的重大契机，逐步会有越来越多的非冯处理器崛起，包括神经形态处理器，基于量子力学规律的处理器等 [4]。

二、现状分析：专用芯片 vs “通用”框架（语言）

张悠慧介绍，目前已经有很多种类的类脑芯片开发，比如，斯坦佛大学 Neurogrid 团队和滑铁卢大学 NEF 团队合作的 Braindrop；英特尔的 Loihi；以及 TrueNorth、SpiNNaker、BrainScales 等。

这些类脑芯片大多以软硬件协同设计为主，各芯片提供了自己特有的软硬件接口以及工具链。好处是端到端的设计对于适配的应用而言效率比较高； 不足之处则是纵向设计，把上层应用和下层系统软硬件等绑定在一起，导致缺乏应用的可移植性，降低开发的效率，开发人员不得不面对底层硬件暴露的约束。

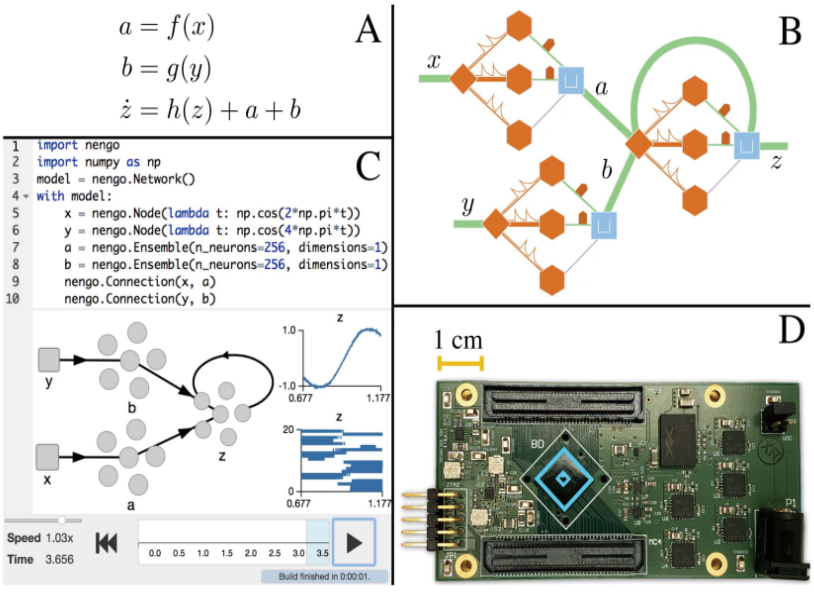
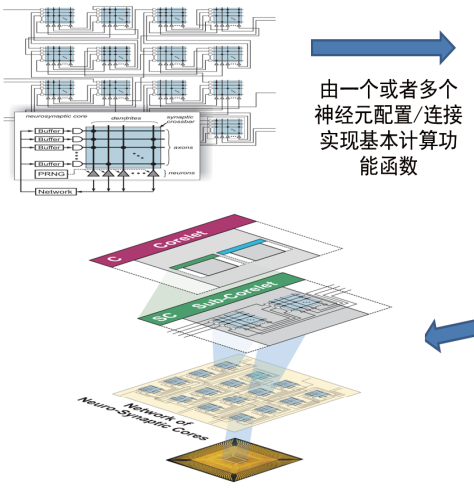


图 4：Braindrop[6]

与 Loihi 等系统相比，Braindrop 做得比较灵活。Loihi 等系统对外提供的编程原语是基于神经元、神经元突触、神经网络这个层次。而 Braindrop 则是利用滑铁卢大学的 NEF 框架，在功能级上提出的编程原语。用户可以将认知计算函数描述成为一个非线性的微分方程系统，并且与底层硬件是无关的、去耦合的，然后通过 NEF 将每个方程转换成一组物理上的神经元，并映射到模数混合的芯片上。但它的问题在于当进行设计的时候，软硬件的所有层次都要协同设计，底层硬件设计也要严格遵循理论框架，它的上、下层耦合度很高，只是耦合到了 NEF 框架里去了。



Type	Name (# of neurons)
Arithmetic	absolute value (3), addition (1), division (2), fixed-gain div./mult. (1), log (1), min (2), max (2), modulation (1), multiplication (1), rate clip (2), rate match (3), sigmoid (1), square root (3), square (2), subtraction (1)
Control	bistable (1), tristable (1), delay (1), one shot (1), pass gate (1)
Data Generation	spontaneous (1), ramp up/down (3), triangle wave (5), random distributions (1)
Logic	AND (1), NAND (1), NOR (1), NOT (1), OR (1), XNOR (3), XOR (3)
Memory	binary (1), rate store (1)
“Neural”	integrate-and-fire (1), bursting (2), coincidence (1), McCulloch-Pitts (1), Boltzmann (1), motion history (1), on/off pair (2) , onset/offset pair (2)
Signal Processing	decorrelation (1), rate attractor (1), convolution (1), low-pass filter (1), high-pass filter (1), band-pass filter (1), spatiotemporal filter (1), Bloom filter (1)
Probabilistic	noisy-OR: prob. union (1), noisy-AND: prob. intersection (1)
Time-to-Spike	L1 distance (2), max (N), min (N), median (N+1), coincidence (1), anti-coincidence (1), matched filter (1)

通过Corelet编程，将基本计算功能函数构建成复杂函数乃至目标算法

而Corelet程序的编译/运行过程则反过来，将程序分解为最为基本的个神经元配置与链接，从而构成相应的神经元网络

图 5：TrueNorth[7]

TrueNorth 是基于认知计算编程语言 Corelet 进行应用开发的。硬件基础单元是 TrueNorth 中的神经形态基元模型，包括神经元计算模型、突触模型及其扩展，通过一个或多个神经元配置互联实现基本功能的函数。通过 Corelet 编程，软件工具链采用分而治之策略，将一个复杂算法分解为简单算法，反复分解直至变换为一个神经突触核所能完成的计算，这时候形成神经形态网络层面的基础单元以及单元的连接。实际上，这种方法也同样把硬件底层的基础单元暴露给了上层开发者。

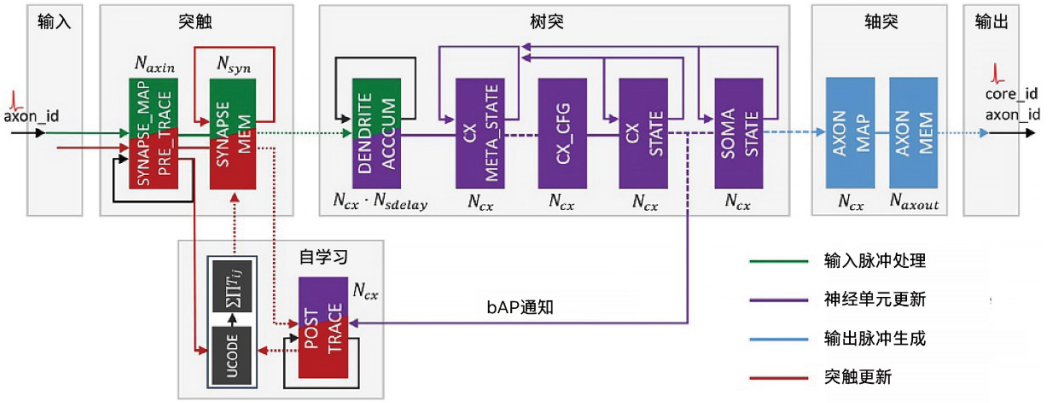


图 6：Loihi 神经元计算硬件流水线 [8]

Loihi 对外提供一系列编程原语，用于脉冲神经网络拓扑描述的神经元间室模型、突出模型、突出踪迹、描述 SNN 动力学的神经元模型以及突出学习规则。如图 6，实际上该硬件流水线基本是对照 SNN 的仿真模型实现的，计算神经学中仿真神经元计算怎么做，Loihi 便把它用硬件实现。相当于 Loihi 把神经网络模拟里的软件流水线硬件固化，形成基本功能单元再暴露给上层的开发人员。首尔大学基于 Loihi 结构进行了拓展 [9]，支持 SNN 的各类学习算法，这些算法主要是基于 STDP 或者 STDP 衍生的众多算法，首尔大学分析了多种算法并归纳出更多原语进行扩展支持。但不管怎样，还是把底层的原语直接暴露给上层。

现在很多研究越来越往通用框架演进，目的则是提供一种高层次的编程抽象，实现与具体芯片无关的统一开发框架，使底层硬件规则约束能够对应用开发透明。好处在于较高层次的抽象能够屏蔽硬件细节，提高可移植性和开发效率。但目前的问题在于，芯片底层缺乏一致的硬件接口，仍存在一对一的移植工作。现在很多采用开放社区与标准化方案鼓励大家做相关工作，比如，美国国家实验室 2019 年做的 Fugu 项目就是其中典型案例 [10]。它试图提供一个通用开发平台，研究人员不需要大量背景知识就能开发应用程序，开发框架通过开源社区的方式去做，将具体应用、脉冲神经网络算法设计、以及神经形态硬件平台的具体细节分离开来，形成三个层次。框架最高层的目标人群是科学计算或者应用开发人员，他们对自己领域很熟但是不熟悉神经网络计算方法； 中间层用户比较熟悉神经网络计算，用神经网络实现功能、提供函数，但是不熟悉硬件平台； 第三层用户是硬件平台专家开发人员，能够优化和裁剪针对特定硬件的神经网络算法。从公开文献看目前该项目还没有支持某一款特定的芯片，仍然是用模拟器进行替代。

三、当前进展：通用计算机启示下的类脑计算研究

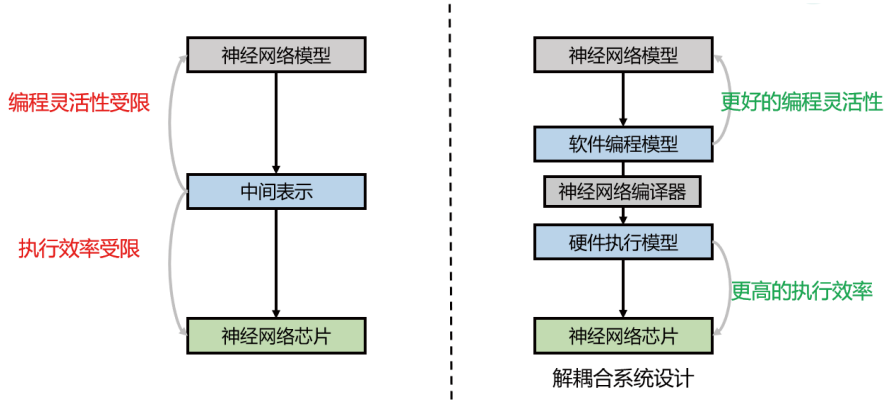


图 7：解耦合的系统设计

张悠慧认为，基于通用处理器的计算机系统层次结构采用的是非常典型的软硬件去耦合的设计方法，这将给予我们方法论上的启示。

总体思路，如图 7 所示，也是采用软硬件去耦合的系统设计，但在应用和芯片之间引入了软件编程模型、硬件执行模型两层抽象。软件编程模型提供给软件开发人员使用，不需要关心底层硬件的规格与约束，把这些交给硬件执行模型处理，在两层之间通过“编译器”进行编译转换。

以神经网络计算为例，上层的编程模型是数据流或者计算图的编程模型，下层的硬件执行模型也可以看作是数据流，但是，它的原语和规模都是受限的。关键的问题是要把两者进行等价转换。

张悠慧介绍，他们团队在 2016 年便提出了解耦合系统原型 [11]，主要通过神经网络的冗余来容忍硬件层面的约束，实现了在类脑硬件具有精度、连接度、函数类型等约束的情况下上层模型到下层模型的自动转换；利用神经网络的冗余性和容错性，通过去除一定的冗余性之后降低精度来匹配硬件约束。该工作支持“天机一代”芯片约束下的“编译”。

2018 年，张悠慧团队进一步做了神经网络编译器 [12]。这包括两个方面，一是对硬件执行模型的形式化定义，引入点积和非线性函数两个硬件基本算子，二是实现严格硬件约束下的神经网络模型转换，即把神经网络转换为约束条件下等价的网络。因为软件神经元和硬件神经元能力不一样，转换成硬件神经元以后规模可能需要膨胀，以规模换取能力实现精度损失可控。

在工作 [12] 的基础上，张悠慧团队做了架构设计与映射工具 [13]。研究 [12] 说明方案是可行的，还需要进一步研究该方案的效率。基于所需的基本原语，他们提出基于忆阻器的可重构神经网络芯片架构（FPSA），证明了方案的高效性与可实现性。整体上是遵从 RISC 设计思想，简化硬件设计利用忆阻器高效实现两个原语，上层各种各样软件的灵活性通过编译器实现。同时，引入 FPGA 可重构路由，解决传统 BUS 和 NOC 无法匹配忆阻器高速计算的通信瓶颈问题。

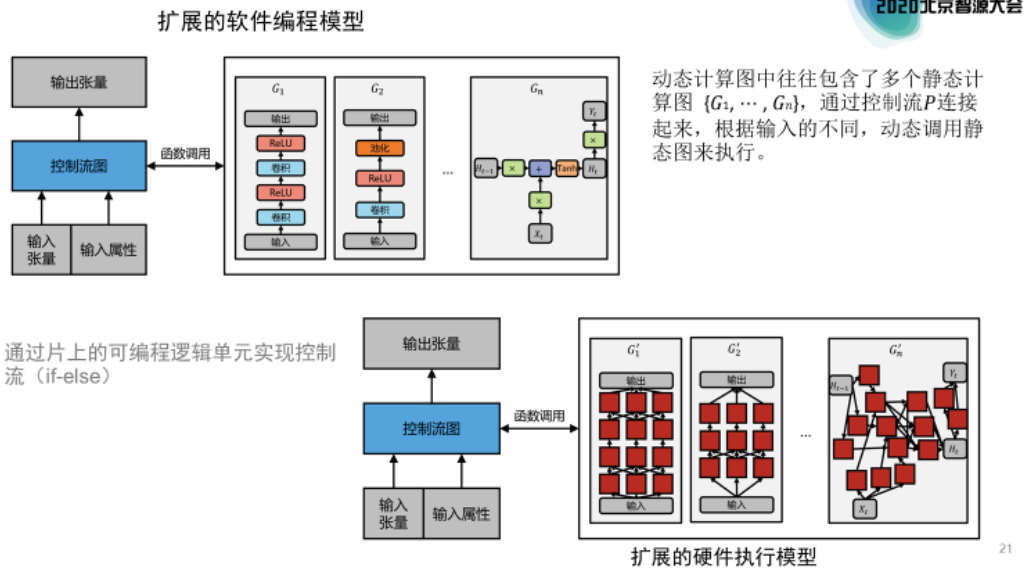


图 8：动态图软硬件模型

张悠慧团队还把相关工作拓展到动态计算图 [14]，即在软件编程模型和硬件执行模型当中使用控制流来支持动态图，使其功能更加完整。动态计算图中包括多个静态计算图，在软件编程模型里根据输入的不同动态调用静态图来执行。底层硬件很大程度上借鉴了 FPGA 的设计，利用可编程的逻辑单元来实现控制流，即硬件上控制器选择不同的分支来完成不同的执行路径。

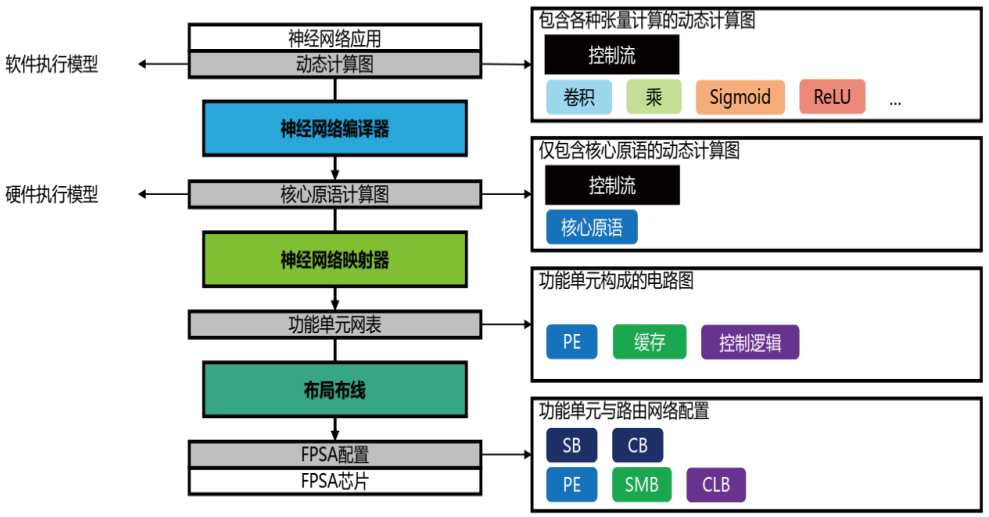


图 9：FPSA 系统栈

FPSA 系统栈如图 9。最上层是包含各种张量的动态计算图，即各种各样有控制的、不受限制的神经网络操作，通过神经网络编译器编译成等价的仅包括控制流本身和硬件支持的几个核心原语的动态计算图。核心原语目前主要有两个，一个是点积，一个是非线性函数。然后再通过神经网络映射器，把硬件执行模型进一步转化为包

含核心原语计算、缓存和控制逻辑的网表，即计算逻辑、计算调度、数据移动和暂存等，最后直接复用 FPGA 的功能实现相关的布局布线。

在前端，张悠慧团队做了基于 GPU 的高性能脉冲神经网络开发与仿真平台[15]，它支持被广泛应用的 SNN 的开发语言——PyNN。针对 SNN 行为的稀疏性、事件驱动等特征，他们采用了细粒度神经网络表示与执行模式，以及带时间戳的稀疏矩阵存储等方法进行特殊优化。使得其性能比现有最新的基于 GPU 的 SNN 模拟器要快很多。

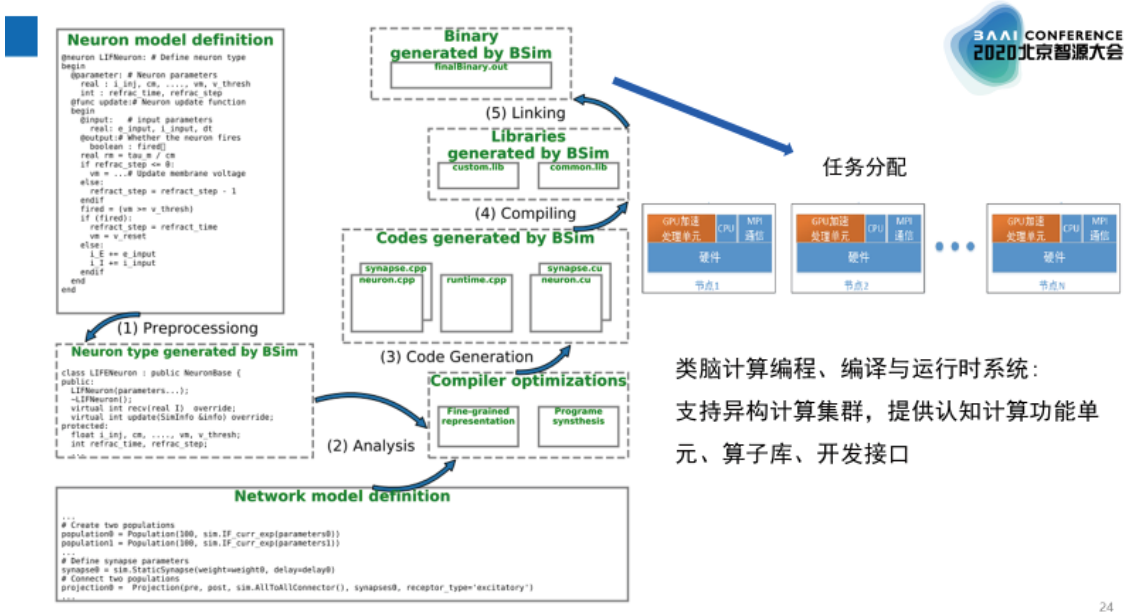


图 10：BSIM 工作流

图 10 是 BSIM 工作流，最上层给开发人员使用，并且屏蔽了硬件的细节，开发人员可以定义模型进行预处理，将网络各种层次连接起来，最后通过编译生成 GPU 上的执行程序。同时，它还支持不同的底层硬件神经元和突出模型，方便进行功能验证。作为类脑计算编程、编译与运行时系统，目前该平台支持 GPU 集群异构计算并提供认知计算功能单元、算子库、开发接口。

四、下一步研究：通用类脑计算

张悠慧认为，当前在类脑计算领域，不管是各家基于芯片构建的系统，还是意图通用的、去耦合的软硬件系统栈都面临的问题是：各平台都有自己不同的开发接口、编程接口，各用各的。

所以从长远来看，我们需要一个解决方案能够应对多个平台。因此希望未来硬件接口能够稳定下来，开发出相同（或者兼容）的接口。

通用计算机系统在这方面做得比较好，软硬件去耦合的计算机层次结构是经典计算机迅猛发展的体系结构基础，这是通用计算机几十年来快速发展背后体系结构方面的逻辑。

如果类脑计算要超越或替代冯·诺依曼模型，我们不能总基于归纳法进行软硬件体系结构的设计，因为这种方法未来无法满足新的应用，并且存在软硬件绑定的问题。

五、总结

传统计算机系统的软硬件去耦合的层次化设计思想是 IT 技术得以飞速发展的体系结构基础；目前类脑计算的系统和芯片，虽然具体类型有所不同但都比较侧重于端到端的软硬件协同设计方法，缺乏能将算法、芯片和器件等不同领域技术和需求有机结合起来的软硬件系统栈设计；因此，我们需要进一步从传统的通用计算系统的成功思路中汲取经验，既要形似，也要神似！

Q&A

陈云霁：作为计算机研究者来说，通用性和完备性非常重要，这正是像图灵机这样具有通用性完备性的模型，使得我们今天研究的是计算机，而不仅仅是一个计算器。第一个问题，中间层和 TVM 的差别是什么？

张悠慧：从在系统中的位置和作用来说，它们是类似的。不同在于，第一，TVM 主要支持 DNN，我们这边更倾向于做 SNN。第二，我们目前正在做的工作是想提出一套硬件原语，从理论上确保这个硬件原语从神经系统计算层面来说是足够的。我们目前做了这么一个简单的原型系统，能够证明带来三个好处：1）如果硬件能够实现这套原语的话，理论上确保软件模型和硬件模型的等价；2）至少理论上可以使得底层的类脑硬件能够支持通用的计算；3）我们这种设计将来引入新的系统评测维度，一般系统讲应用性能和硬件开销之间的 Tradeoff，我们引入了近似粒度，能够容忍怎样的 Error。这是一个三角关系。

陈云霁：我想请教一个问题，我自己感觉在深度学习发展里面有这样一个问题，一开始在学术界应用相对少一些它会显得比较干净规整，但是一旦进入工业界之后，很多“肮脏的东西”就会往上跑。比如我们看现在比较大规模的深度神经网络，比如自然语言处理里面的 BERT，或者图像检测里面的 Faster R-CNN，仅仅说是向量内机（点积）和非线性激活函数就感觉 hold 不住了。目前 SNN 在工业上的实际部署相对来说比较少一些，您能否预测 SNN 未来进入大规模工业应用会不会有这样的问题出现？

张悠慧：我觉得肯定会有这样的问题出现，去耦合和 Codesign 永远是相辅相成的。理想情况下，基石是去耦合的，通过提供一系列原语来实现。同时，针对特定应用而言，还要通过 Codesign 在去耦合基础上去提升性能和效率，我相信将来系统肯定是这么一个混合式系统，包括传统计算机也是这样的，完备性是个基石，但是 Codesign 是提升效能的关键，所以我们可能会针对不同应用需求去扩展做 Codesign。

参考文献

[1] C. Mead.Neuromorphic electronic systems. Proceedings of the IEEE, 1990, 78(10):1629-1636. Schuman C D, Potok T E, Patton R M, et al. A Survey of Neuromorphic Computing and Neural Networks in Hardware[J]. 2017.

[2] John L. Hennessy, David A. Patterson. A New Golden Age for Computer Architecture. Communications of the ACM, February 2019.

[3] Kathleen E. Hamilton, et al. Accelerating Scientific Computing in the Post-Moore's Era. ACM Transactions on Parallel Computing. 2020.

[4] Jack D. Kendall, Suhas Kumar. The building blocks of a brain-inspired computer. Applied Physics

Reviews 7, 011305 (2020).
[5] "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model. Proceedings of the IEEE, 2019"
[6] A. Amir et al., Cognitive computing programming paradigm: A Corelet Language for composing networks of neurosynaptic cores, IJCNN 2013.
[7] Lin C K, Wild A, Chinya G, et al. Programming Spiking Neural Networks on Intel Loihi[J]. Computer, 2018.
[8] E. Baek et al., FlexLearn: Fast and Highly Efficient Brain Simulations Using Flexible On-Chip Learning, MICRO 2019.
[9] James B. Aimone, William Severa, Craig M. Vineyard. Composing neural algorithms with Fugu. International Conference on Neuromorphic Systems. 2019.
[10] Yu Ji, et al. "NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints".
[11] Yu Ji, et al. "Bridge the Gap between Neural Networks and Neuromorphic Hardware with a Neural Network Compiler".
[12] Yu Ji, et al. "FPSA: A Full System Stack Solution for Reconfigurable ReRAM-based NN Accelerator Architecture".
[13] Y. Ji, Z. Liu and Y. Zhang, "A Reduced Architecture for ReRAM-based Neural Network Accelerator and Its Software Stack".
[14] P. Qu, Y. Zhang, et al, "High Performance Simulation of Spiking Neural Network on GPGPUs".

美国南加州大学钱学海：去中心化分布式训练系统

整理：智源社区 吴继芳

钱学海，美国加州大学电子工程系助理教授，于 2013 年获得伊利诺伊大学香槟分校博士学位，并在加州大学伯克利分校从事过博士后研究工作。曾荣获首届北美计算机华人学者协会新星奖和 NSF CAREER AWARD。在 ISCA、ASPLOS、MICRO、HPCA 四个主要体系结构会议发表论文 37 篇，并进入 ASPLOS 和 HPCA 的名人堂 (Hall of Fame)。钱学海研究团队的网站是：http://alchem.usc.edu。

钱学海的演讲主题是《去中心化分布式训练系统》。在本次演讲中，钱学海系统地介绍了他和他团队近期关于去中心化的同步和异步算法的分布式训练系统设计的相关研究工作。钱学海介绍，当前流行的机器学习系统大都采用中心化和同步的并行训练，造成了通讯的瓶颈并容易受异构运行环境的影响，针对这个问题，他们提出了有效的去中心化分布式训练的系统设计方法。对同步执行，钱学海团队利用执行状态差别和通信图的关系，设计了系统资源控制的同步协议；对异步执行，他们则提出了一种新的通信原语和避免系统等待的有效方法，使得机器学习系统能有效地消除通信瓶颈并达到和当前集中式训练相当或更好的性能。下面是智源编辑为大家整理的讲座内容。

一、研究动机

1.1 背景

图灵奖获得者 John Hennessy and David Patterson 在文章“A New Golden Age for Computer Architecture”[1] 中指出：随着摩尔定律和 Dennard 定律的终结，我们已经进入计算机体系结构的黄金时代，即无法通过简单的提高系统复杂性和主频来提高计算机系统性能。我们需要深入理解应用特性，通过设计面向特定领域的体系结构 (Domain-specific architecture (DSA)) 和编程语言 (Domain-specific language (DSL)) 来实现在不大幅度提高系统复杂性和成本的情况下提高应用的性能。目前，在技术方面，随着许多新技术 (比如：NVM, ReRAM, RDMA 等) 的出现，为计算机体系结构设计带来新的机遇；在应用方面，人工智能和大数据都给系统计算和存储都带来新的挑战。当前的系统都被划分为很多层次 (比如：按应用、特定领域语言及执行框架、计算机体系结构和实现技术等划分)，在设计软硬件系统的时候有一个重要方法是 Vertical Integration [1]，即要注意不同层次之间的融合以及交互，从而发现新的机遇。

钱学海本次主要分享的是在软件方面面向特定领域的分布式训练系统。随着深度学习在众多领域的成功应用，快速和高效的模型训练越来越重要。为了获得高准确率，模型大小和训练数据都日益增加，分布式并行训练成为不可或缺的技术。对于面向深度学习或者人工智能的系统和体系结构，需要考虑下面两个方面：

- 训练 (Training)。在训练过程中需要用到大规模数据集和复杂模型计算，对系统的计算和存储要求比较高，因此通常需要采用分布式和并行性技术，达到在短时间内训练出需要的模型。
- 推理 (Inference)。在推理过程中，运用训练得到的模型处理实际应用中的数据，获得有用的信息。通常推理过程的计算比训练过程的简单一些，但是推理一般部署在低功耗或者资源受限的系统上，能源效率是一个重要挑战。

接下来，钱学海向大家解析了关于训练过程的去中心化的分布式训练系统设计。

1.2 挑战

钱学海认为，分布式训练系统的核心问题是不同计算设备的通信以及由于计算和通讯性能的差异带来的异构性，具体可以看做面临三个方面的挑战，分别是： 通信（Communication）、异构性（Heterogeneity）和多种因素交互造成的复杂性（Complex Trade-offs ）。

1.2.1 通信（Communication）

通信与系统结构相关，早期的分布式训练系统 Parameter Server（PS）[2]（图 1 所示）是一个中心化的结构，每一个 Worker 会把自己计算得到的梯度上传到中心节点，中心节点对所有 Worker 的梯度做平均化处理后，再传回各个 Worker。作为中心化的结构，PS 中存在中心节点通信瓶颈问题。第二代分布式训练系统设计方法是 Ring All-Reduce [3]（图 2 所示），采用了节点同步环节采用了并行流水线的工作方式，使得节点间的通信变得更加高效。但是这种方法是同步的，当一个平台上不同的 Worker 具有不同的运行性能时，整个系统的性能受限于运行效率最低的 Worker。

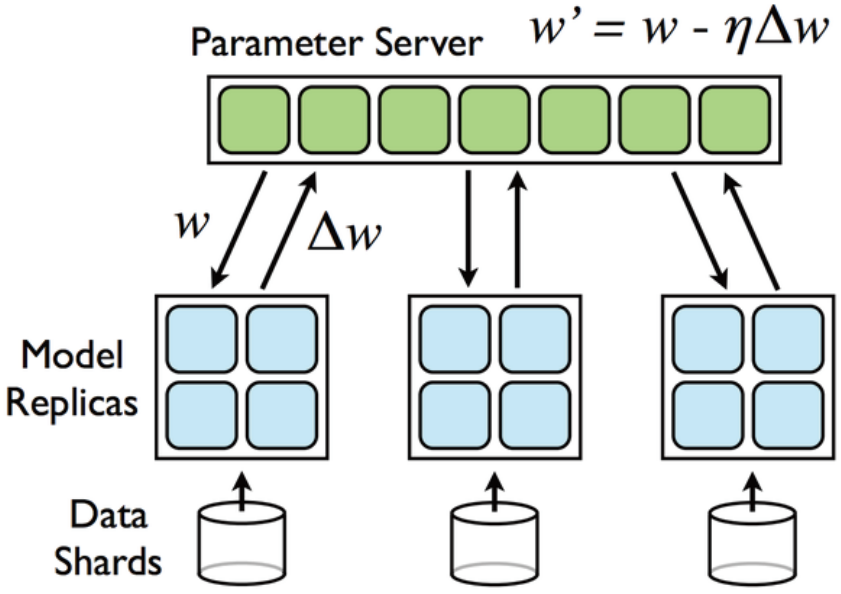


图 1：Parameter Server （PS）系统结构

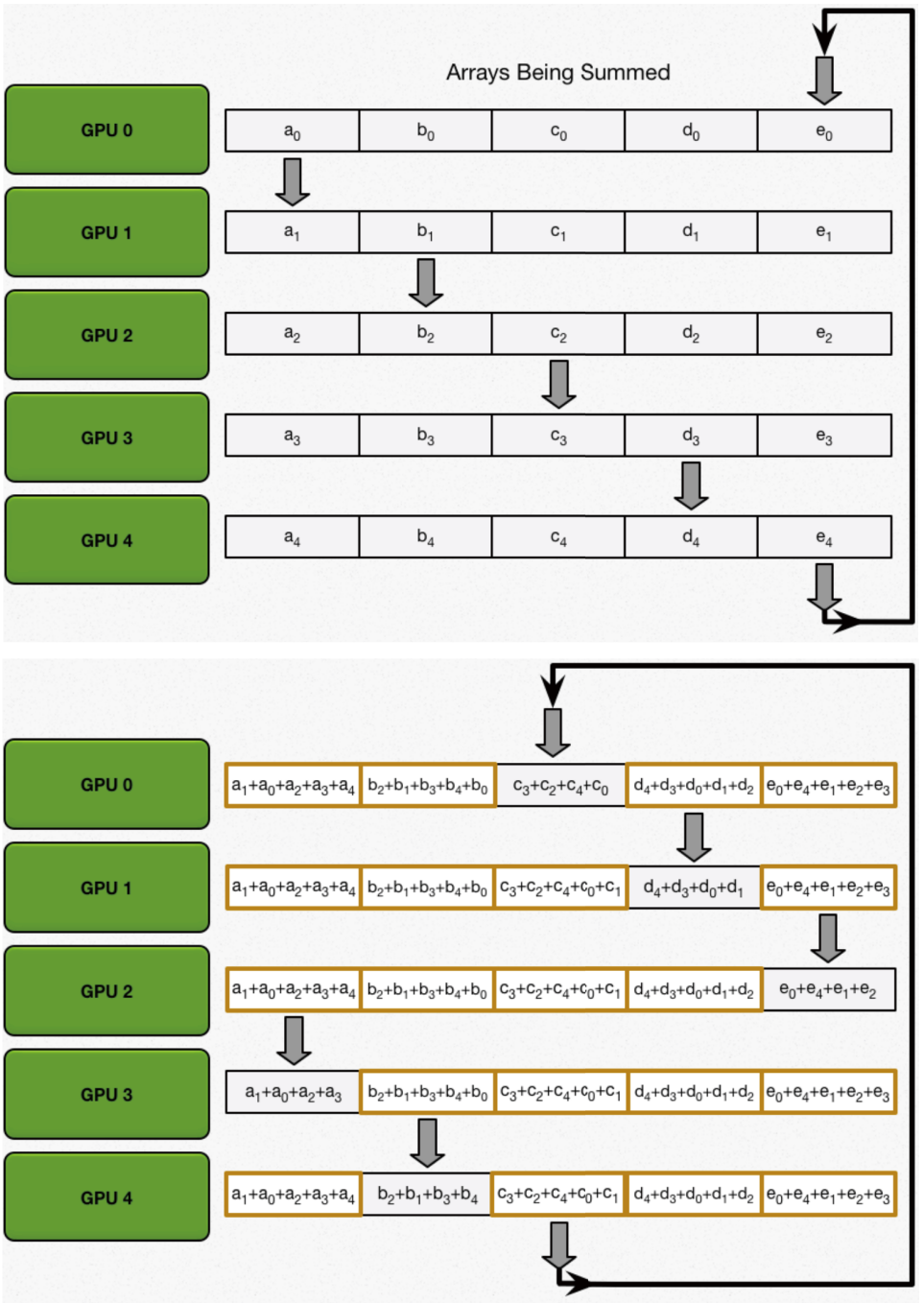


图 2：Ring All-Reduce 系统结构

1.2.2 异构性 (Heterogeneity)

异构性一般主要考虑两种：确定异构性 (Deterministic heterogeneity) 和 动态异构性 (Dynamic heterogeneity)。确定异构性是指不同的设备通常有不同的计算能力 (比如：采用不同的 GPU 或 TPU 系列)，并且设备或节点具有不同的网络通信性能 (比如：同一个节点中的不同 GPU 的通信速度比不同节点间的通信速度快)。动态异构性是由资源共享 (Resource sharing) 造成的，即同一个设备对不同的应用提供的计算性能也是不一样的。对于分布式训练系统来说，在一定的情况下会产生 Slow Worker，这些 Slow Worker 被称为 Straggler。Straggler 问

题也被称之为异构性（Heterogenity）问题，指的是当一个平台上不同的 Worker 具有不同的运行性能时，整个系统的性能受限于运行效率最低的 Worker。

1.2.3 复杂性（Complex Trade-offs）

由于分布式训练系统多个因素以及它们和算法存在复杂的交互，需要通过全局的方法考虑系统的设计。比如：

- 并行性和通信之间的关系（Parallelism and Communication）。目前有两种并行方法：Data parallelism（图 3 所示）和 Model parallelism（图 4 所示）。采用不同的并行方式，会造成不同的通信行为和性能。因此，如何选择最优的方式，是一个重要的问题。
- 系统执行和模型训练效率之间的关系（Hardware vs. Statistic efficiency）。能够提高系统吞吐率的设计一定能够使模型训练快速收敛吗？

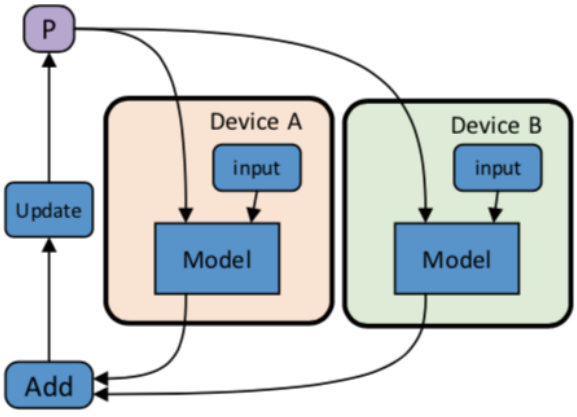


图 3：Data Parallelism

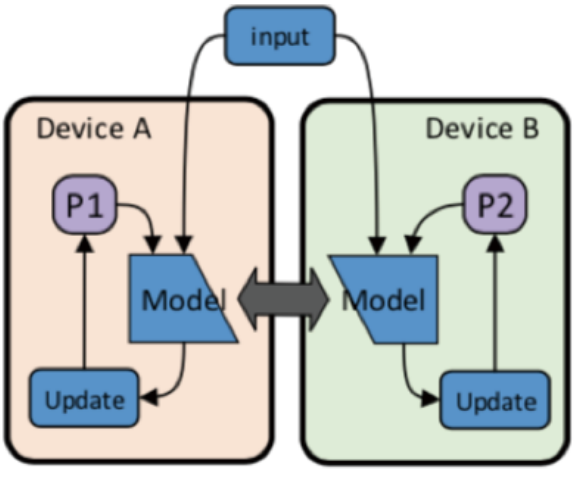


图 4：Model Parallelism

近几年来，钱学海针对这些挑战做的研究工作，主要有两个方面：一方面是在通信（communication）和异构性（Heterogeneity）问题方面，研究了去中心化的训练（Decentralized Training）；另一方面是并行性和通信之间的

关系（Parallelism and Communication），主要考虑系统具有多个加速器或者 GPU 的情况下，如何决定并行模型来取得最优的通信量和性能。本次钱学海的分享主要是第一方面的相关研究工作，关于同步和异步算法的去中心化分布式训练系统设计。

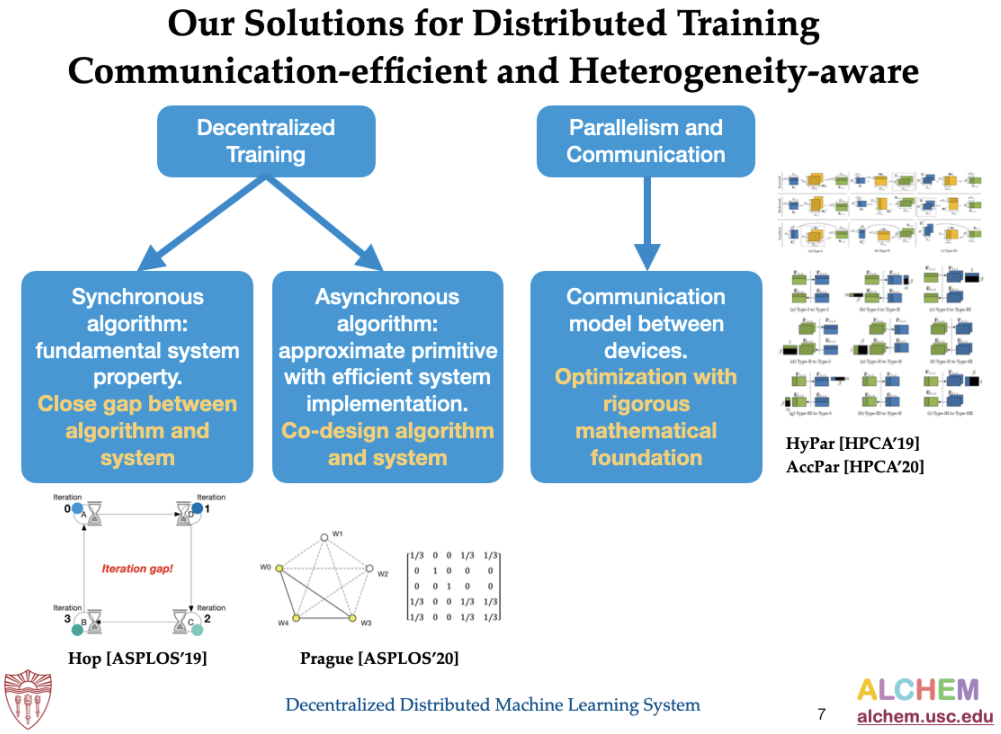


图 5：钱学海团队和合作者关于并行训练的系统和体系结构的研究

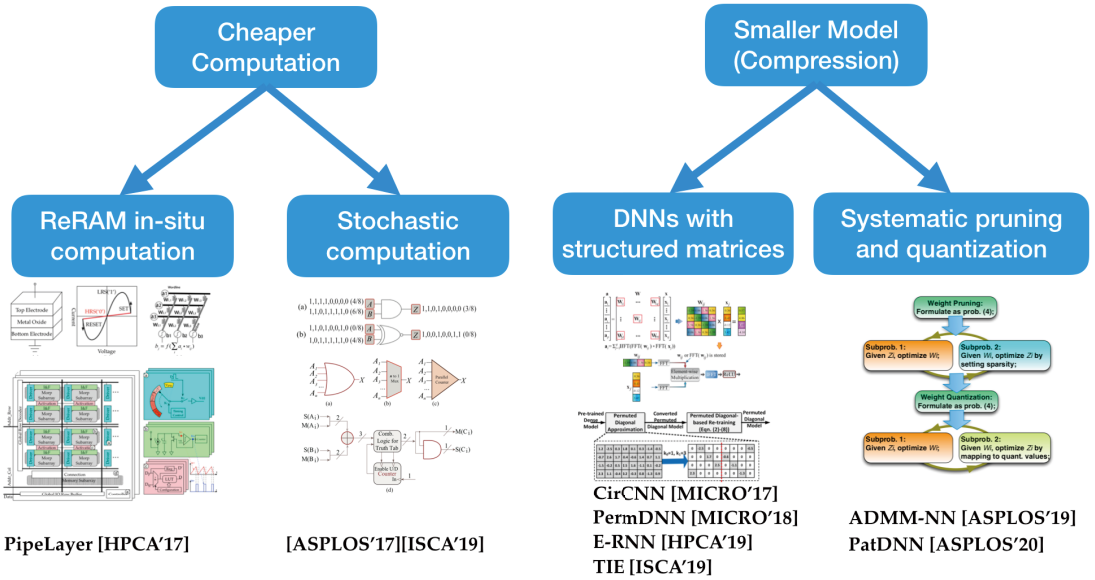


图 6：钱学海团队和合作者关于深度学习加速器体系结构的研究

二、去中心化分布式训练系统

什么是去中心化的分布式训练？如图 7 所示，去中心化的分布式训练算法的假设是： 首先存在一些 Worker，这些 Worker 中没有一个中心节点作为 Parameter Server，它们之间是对等的。同时它们之间的通信拓扑结构是可以用户自定义的 (即自定义通信图)。如果不考虑算法，可以把 PS 和 Ring All-Reduce 看做是这个方法的特殊情况，PS 可以看做是一个类似星型的通信图，Ring All-Reduce 可以看做是一个 All-to-All 的通讯图。在去中心化的方法下，可以定义任何模式的通信图。虽然相关去中心化算法或者理论的论文中对通信图的拓扑结构有一定要求，但是对于给定的 Worker 可以有很多种选择，这提供了一个更灵活的通讯方式。我们也可以根据系统的配置信息来设计通讯图，这为算法与系统协同设计提供了机会。那么，去中心化得训练方法能不能达到与 PS 和 Ring All-Reduce 相类似的效率呢？这个问题在文献 [4] 中已经回答了，答案是肯定的。

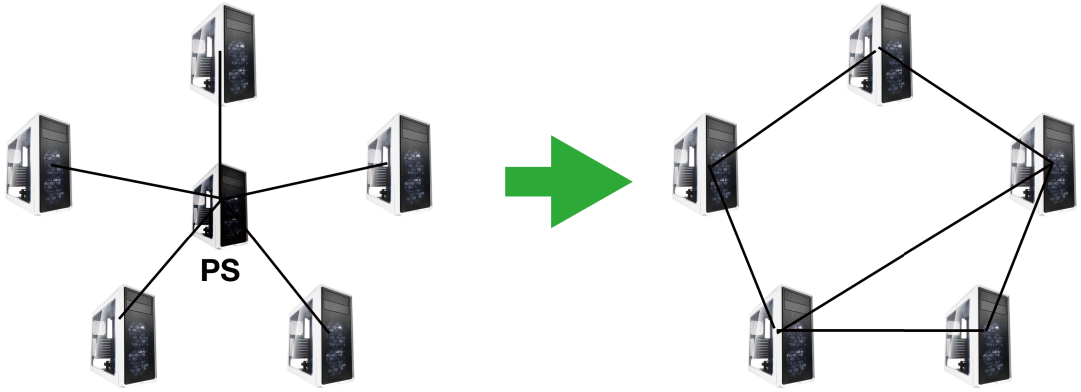


图 7：中心化训练 vs. 去中心化训练

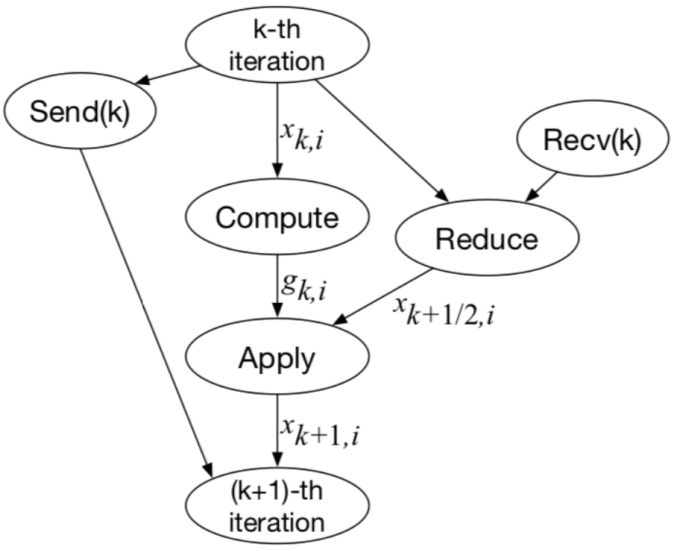


图 8：去中心化训练过程中每个节点 Worker 的操作（计算图）

去中心化训练的执行是 Iterative 的处理过程，每个 Worker 保持自己的模型参数和计算梯度。在每一个 Iteration 中，每个 Worker 按照通信图进行节点间通信，每个节点 Worker 的操作如图 8 所示：

- 根据 Min-batch 数据和当前的模型进行梯度计算，并更新模型；
- 把参数发送给根据通讯图决定的“邻居”Worker；
- 接收到其他 Worker 的参数，并对参数进行 Reduce 操作；
- 更新模型参数，进入下一个 Iteration。

2.1 同步算法的系统设计

在去中心化的分布式训练中的同步是指每个 Worker 在做 Reduce 操作时，所有的参数应该出自同一个 Iteration。这是在构建去中心化分布式训练系统中独特的一个问题，在 PS 和 Ring All-Reduce 中都不存在这样的问题，因为它们有一个非常自然的中心节点或者有一个同步的操作去划分 Iteration。

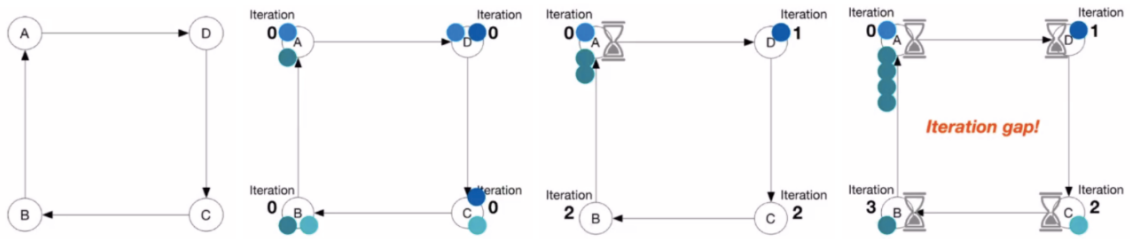


图 9：Property of decentralized training

在同步的去中心化的算法中有一个重要的性质。如图 9 所示，有四个 Worker：A、B、C、D，由环形结构的通信图相连，开始它们都处在 Iteration(0)，执行时每个 Worker 会先进行梯度计算，然后根据通信图把参数的更新发送到邻居节点，每个节点收到参数的更新后，会进行 Reduce 操作，并进入下一个 Iteration。现在假设 Worker A 由于某种原因（比如：由于资源竞争，得不到足够的计算资源）停滞不前（Slowdown），没有给 Worker D 发送参数更新。由于是分布式的协议，D 并不知道这个情况，D 仍然在进行梯度计算，并且可以将参数更新发送给 Worker C，然后 C 可以给 B 发送，B 可以给 A 发送。这样 B 和 C 可以进入下一个 Iteration，D 由于没有收到 A 的参数更新，无法进入下一个 Iteration。依次的，这个停滞不前会逐渐传递给所有的 Worker。在这里面，有一个重要的概念：Iteration gap。可以看到 A 已经收到 4 个参数的 Update，这 4 个更新都是从 B 来的，区别是它们分别来自不同的 Iteration。根据同步的条件，在 A 被唤醒后进行 Reduce 操作时需要正确的选择更新的版本。所以，A 需要有足够的 Buffer 来保存这些 Update，那么这个 Buffer 的 size 是跟 Iteration gap 相关的。在论文 [5] 中也已经证明了 Iteration gap 与通信图的拓扑结构有关。比如图 9 中 A 到 B 的最大 Iteration gap 是由 A 到 B 的路径决定的。由图中可以看出 A 到 B 的路径长度是 3，所以 A 到 B 的 Iteration gap 是 3；A 到 C 的路径长度是 2，所 A 到 C 的 Iteration gap 是 2。Iteration gap 的大小，决定了需要的 Buffer size 的大小。

2.1.1 系统挑战

关于同步算法的去中心化分布式训练系统设计面临的挑战是：

- 如果用户给定一个通信图，Buffer size 是通信图拓扑结构的一个函数。系统应该能够自己决定给应用分配多少资源，而不是根据需求不断的分配。
- 针对于系统的异构性问题，Backup Worker 是解决异构问题的一种技术，但是会造成无穷大的 Iteration gap（图 10 所示）。

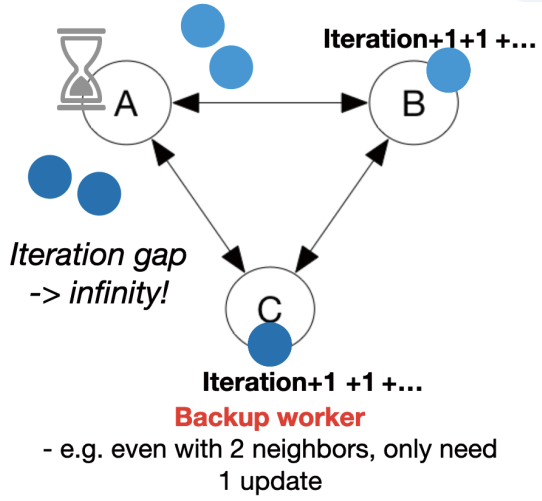


图 10：Tolerate heterogeneity with backup Worker

钱学海团队设计系统的思路是：首先系统要能够支持所有的通信图，即使通信图很大，也能够通过分配一定 Size 的 Buffer 来支持。其次希望 buffer size 要能够控制同步的松紧程度（Bounding Iteration Gap）。比如：只分配一个小的 Buffer，协议也能够执行，只是对通信和同步要求更严格。针对上面提到的问题，他们提出一个解决方案：基于队列的同步（Queue-based Synchronization）。

2.1.2 基于队列的同步（Queue-based Synchronization）

基于队列的同步 [5] 是指每个 Worker 有两个队列，一个是 Update Queue（UQ），用来存储邻居发送来的信息，一个是 Token Queue（TQ），用来控制同步。如图 11 所示，假设在部署协议时，设定 Iteration Gap 的最大容忍是 2（即 Iteration Gap<=2），则 TQ 的 Token 数最多为 2。协议开始执行以后，每一个 Worker 都可以向其他 Worker 发送参数更新。Worker 要想进入下一个 Iteration，必须要从邻居的 TQ 中获取 Token 放入自己的 TQ 当中。图 9 中如果 C 要进入下一个 Iteration，必须先从 TQ_A->C 和 TQ_B->C 中获取 token，分别放入 TQ_C->A 和 TQ_C->B 中。其他节点也是做同样的操作。

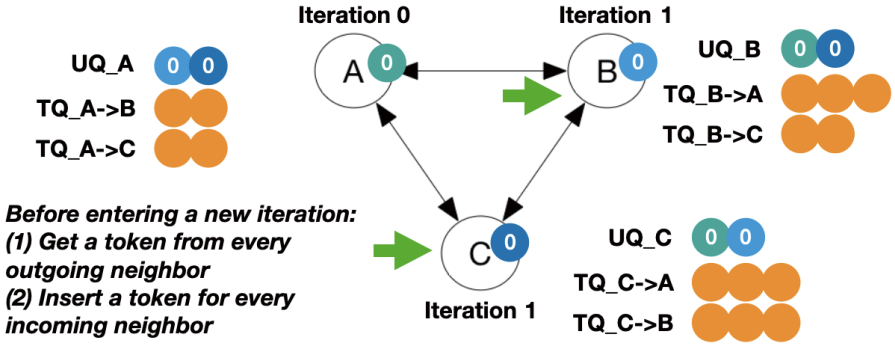


图 11：Queue-based Synchronization without backup Worker

假设系统设计中采用 Backup Worker 技术，那么是怎么实现 Bounding Iteration Gap 呢？如图 12 所示，设定 Backup Workder 的数量为 1（Number of backup = 1），如果节点 A 出现 Slowdown，B 和 C 仍然可以继续进

行。当 B 和 C 进入 Iteration(2) 之后，由于 A 仍然在 Iteration(0) 且 A 的 TQ 中已经没有 token 了，所以 B 和 C 也会停止。这样就确定了 Iteration gap 的边界。这个边界值是由 TQ 的长度决定的。

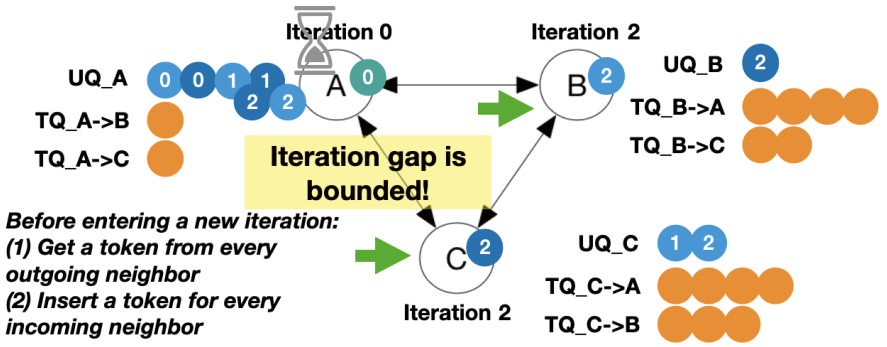


图 12：Queue-based Synchronization with backup Worker

2.2 实验

前面提到可以根据系统的设置来设计通讯图，钱学海团队通过设计实验比较了三种通讯图 (图 13 所示)，得到了一个比较有趣的实验结果。这三个通信图连接的是在三个 Node 中的 Worker，不同 Node 上的 Worker 用不同颜色表示，分别为：灰色、红色和蓝色。图中可以看出第一个通信图 (Setting 1) 中的 Worker 间的通信很充足，从原理上来讲，它的收敛应该很快。Setting 2 和 Setting 3 的区别是不同 Node 之间只有一个 Worker 互相通讯。原理上来讲，它们的收敛应该是慢的。但是实验之后发现，第一个通信图的收敛速度比第二、三个通信图要慢。原因是一般节点的带宽是有限的，所以不断得增加通讯图中 Worker 间的连接，并不一定能够达到很好的效果。反之，Setting 2 和 Setting 3 是根据硬件平台的结构进行设计，节点内的 Worker 可以多通信，节点间的则少通信。这样反而能达到比较好的效果。

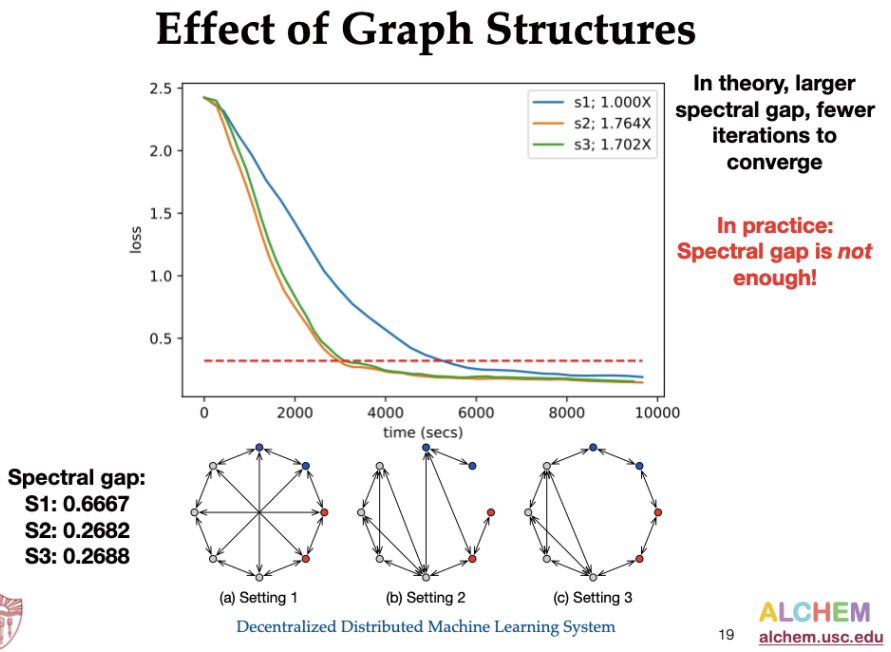


图 13：三种不同通信图效率比较

2.3 异步算法的系统设计

同步算法的系统设计虽然可以通过一些机制（比如：TQ）来容忍一定的异质性问题，但是根据这些协议，如果有一个 Worker 特别的慢，那么最终一定会影响到其它 Worker。从本质上来说，前面提到的方法是运用了系统的技术来容忍一定的异质性，其实还可以从算法的角度来解决这个问题。发表在 ICML 2018 上的 AD-PSGD [6] 是去中心化异步训练的一个代表算法。它的核心思想是基于随机的通信，即每个 Worker 执行完一个 Iteration 以后需要通信，这个通信是随机的从邻居 Worker 中选择一个，而不是按照固定的通信图进行通信。同时，要求通信的 Worker 对之间进行原子性的模型参数平均化操作。

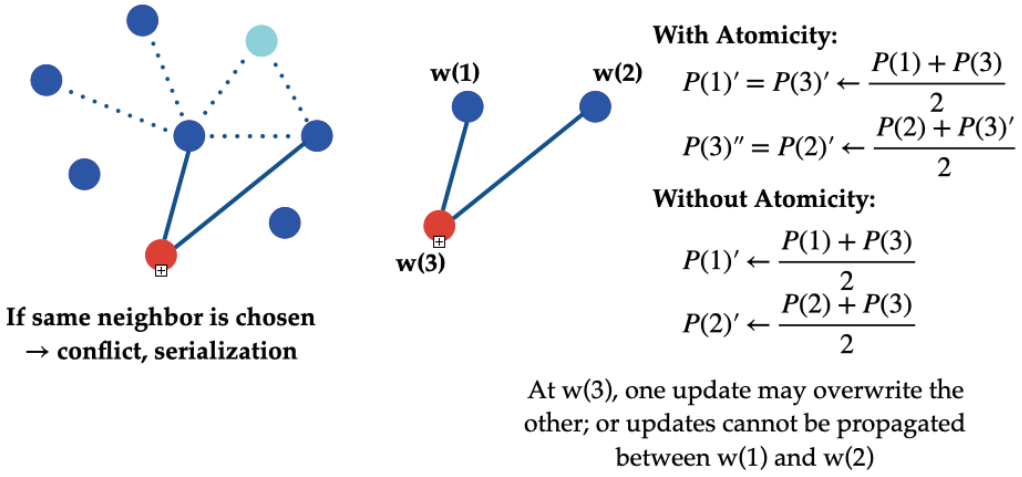


图 14：原子性操作的必要性

为什么需要原子性操作呢？简单来说就是可以确保参数在 Worker 之间更好的传播。如图 14 所示，当两个 Worker w(1) 和 w(2) 同时选择了 w(3) 进行通信时，如果进行原则性操作是指 w(1) 和 w(3) 之间的同步与 w(2) 和 w(3) 之间同步必须串行执行。从图中简单的原子性计算公式中可以看出，两个串行操作执行完以后，w(2) 的模型更新中也包含 w(1) 的模型更新。如果不进行原子性操作，w(1) 的模型更新就不能传递到 w(2) 的模型更新中。钱老师和他的团队也做了相关实验证明，如果没有原子性操作，模型的收敛速度会大大降低。

2.3.1 系统挑战

在异步算法的去中心化分布式训练系统设计中面临的挑战是：在分布式的情况下，如何高效的实现原子性操作。如果直接按照 AD-PSGD 算法来实现原子性操作，在分布式系统中性能会受到很大影响，具体表现在两个方面：原子性操作会导致大量同步成本。如图 15(a) 中的实验结果所示，在整个训练过程中，AD-PSGD 在不同的数据集上训练的同步开销都比 Ring All-Reduce 要大。

在异构环境下，AD-PSGD 的收敛速度比 Ring All-Reduce 要快；在同构环境下，Ring All-Reduce 比 AD-PSGD 要快很多 (如图 15(b) 所示)。那么，能不能够设计一个协议，在同构和异构环境下都能达到很好的效果呢？

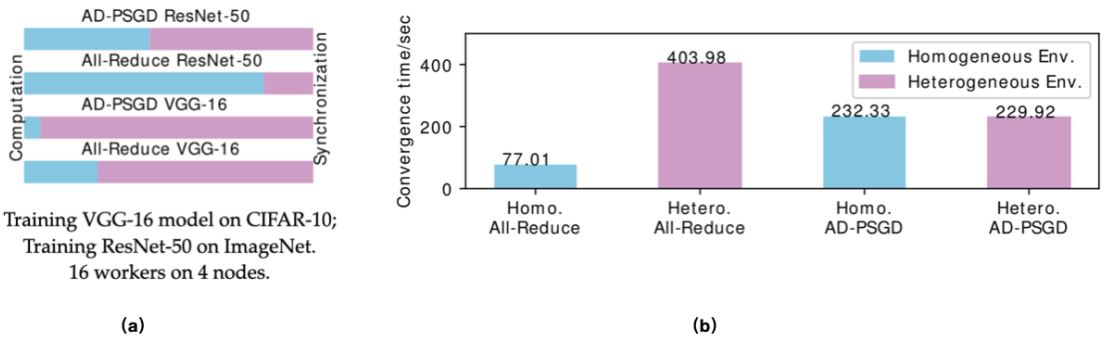


图 15：AD–PSGD vs Ring All–Reduce

针对以上问题，钱学海团队主要从两个方面考虑：(1) 如何让同步操作本身更快；(2) 如何减少冲突，降低由于串行操作带来的延迟。鉴于此，钱学海团队提出了 Partial All–Reduce [7]。

2.3.2 Partial All–Reduce

接下来，钱学海介绍了 Partial All–Reduce 的工作原理。

假设有 n 个 Worker，每个 Worker 有 N 个参数，则全部 Worker 的参数可以表示成 N x n 的矩阵 X。同步的操作可以表示 n x n 的同步矩阵 W，则原子同步操作可以表示成两个矩阵相乘。如图 16 所示，有两个冲突的同步操作，Worker0 和 Worker4 同时选择了 Worker3 做参数平均化。因为参数平均化操作的原子性，同一时间 Worker0 和 Worker4 中只能有一个和 Worker3 进行参数平均化，两次同步可能以任意的顺序发生。正是这类因冲突而串行化的过程使得 AD–PSGD 具有较大的通信开销。针对这个问题，他们提出一个近似同步，即基于组的 Worker 同步机制，也就是 Partial All–Reduce（简称为 P–Reduce）机制。如图 16 中所示，不单独依次考虑节点对的同步，而是把两个同步当成一个操作，采用参数平均化矩阵。这样操作不会影响收敛速度，同时还有一个重要的好处是可以用现有的 All–Reduce 的相关库高效实现方案。

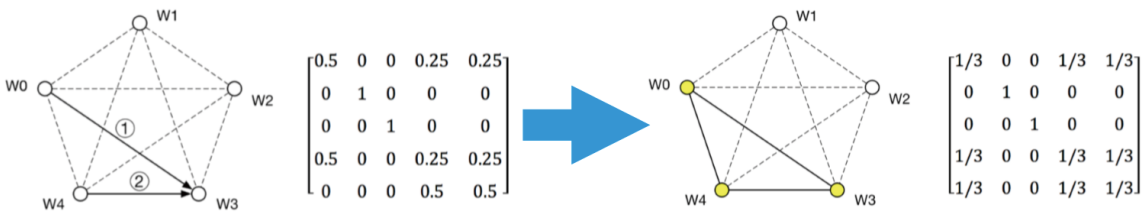


图 16：AD–PSGD 中的同步操作与 Partial All–Reduce 中的同步操作

在 P–Reduce 中，一组 Workers 作为同步的基本单元。Prague 系统的设计思路如下：

- 每个 Worker 在本地训练模型，进行梯度计算及更新模型参数，
- 从 Group Generator（GG）中获得自己的分组信息，
- 同一组中的 Worker 统一进行 P–Reduce 操作。

Group Generator (GG) 是一个独立中心化部件，专门用于为 Worker 集群生成组，当 Worker 想要同步的时候，只需要联系 GG，GG 会返回代表当前 Worker 的组信息 (即针对每个 Worker 接下来将进行的操作)，同时将生成的组信息放入组内相关成员各自对应 的 Group Buffer (GB) 中。每个 Worker 发送请求给 GG 时，如果 GG 总是仅生成一个组，则可能会产生很多冲突 (图 17(a) 所示)。针对这个问题，他们提出 Global Division (GD) 方法，让 GG 在第一次收到请求分组的时候，进行全局划分，提前为 Worker 生成多个没有冲突的分组 (图 17(b) 所示)。

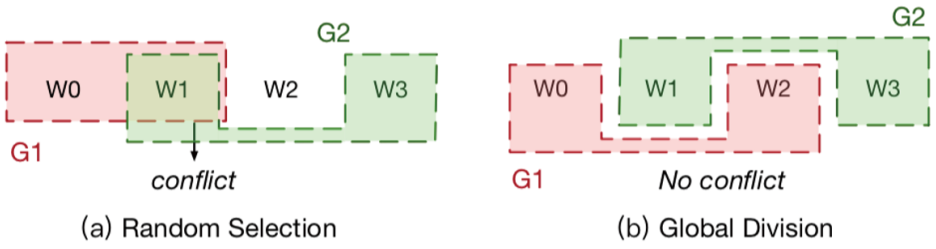


图 17：Random Selection vs. Global Division

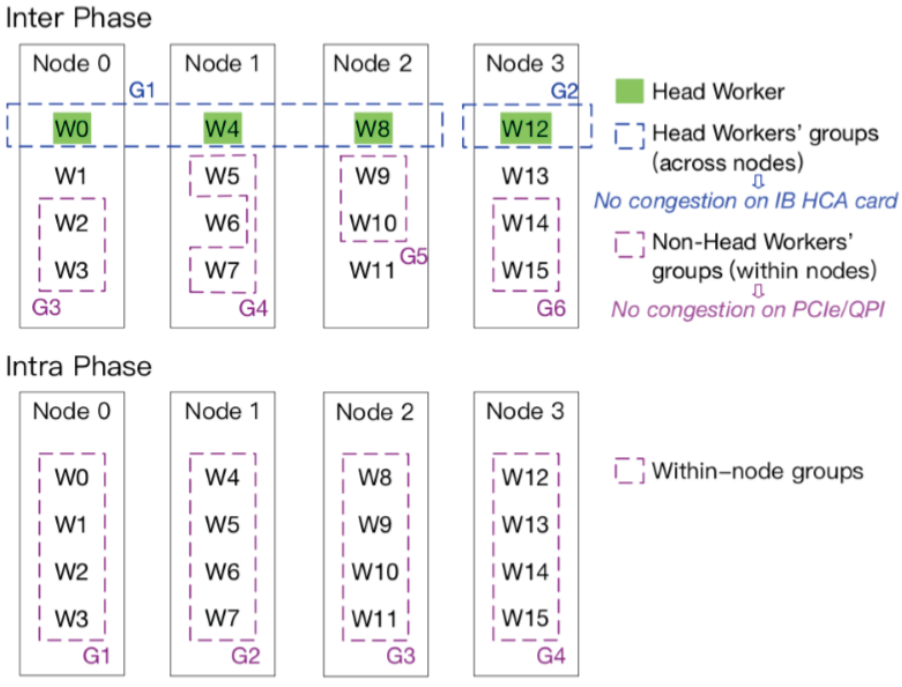


图 18：Inter-Intra Synchronization

一个节点上可以运行多个 Worker，且不同节点之间的带宽是有限的。为了能够平衡不同节点之间的带宽，Prague 提出 Inter-Intra Synchronization (图 18 所示)。在 Inter Phase，每个节点上选择一个 Worker 作为 Head Worker，这些 head Worker 之间采用全局划分 (GD) 进行跨节点的同步操作。由于不同节点间只有一个 Worker 进行同步，所以这样可以避免网络拥塞。在 Intra Phase，节点内部的 Worker 之间进行同步操作，不存在节点间的通讯，可以加快执行速度。

2.3.3 实验

钱学海团队设计了实验，对比验证 Prague 分别在同构和异构环境下的执行效果。Prague 基于 TensorFlow 实

现，实验在 Maverick2 集群上运行。采用的模型和数据集分别是:

- VGG-16 on CIFAR-10
- ResNet-50 on ImageNet

实验结果如图 19、20 所示，在同构和异构环境下，Prague 都能获得最好的训练性能。

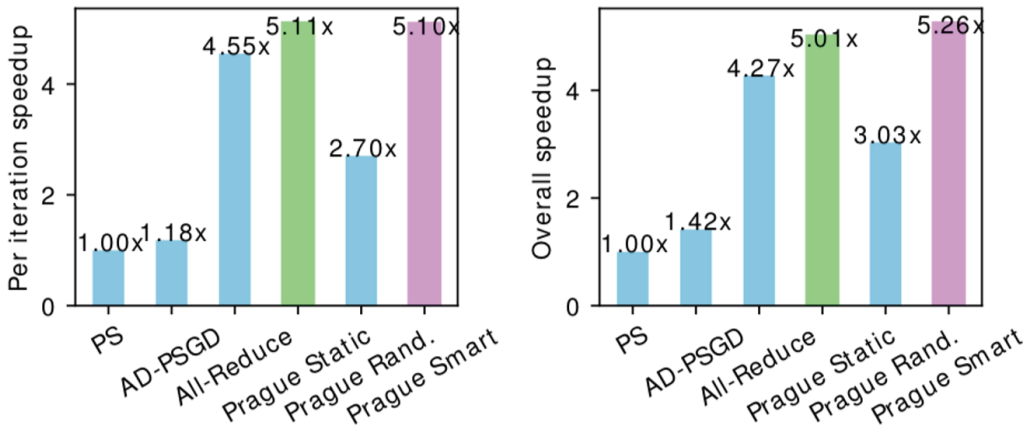


图 19：Speedup in Homogeneous Environment

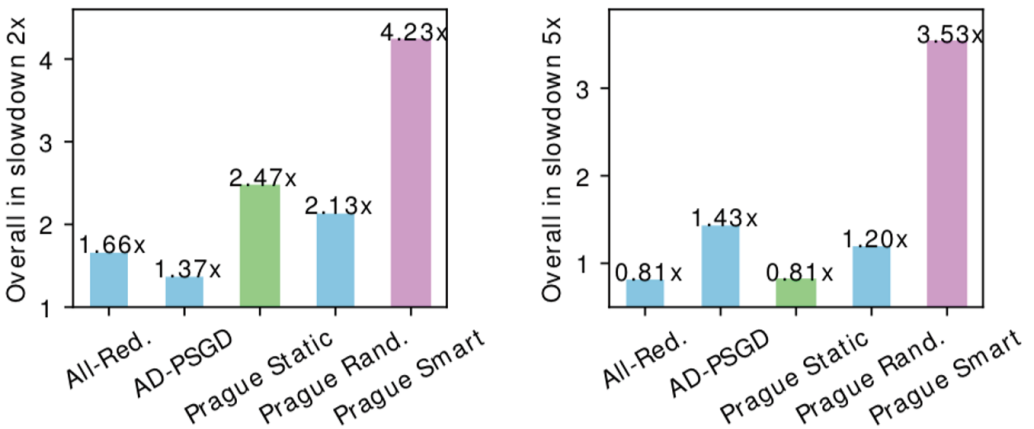


图 20：Speedup in Heterogeneous Environment

三、结语

本次演讲中，钱学海主要分享了他团队最近的两个研究工作：Hop [5] 和 Prague [7]。这两个工作主要是针对去中心化的同步和异步算法的分布式训练系统设计思路。分布式训练系统的核心问题是不同计算设备的通信以及由于计算和通讯性能的差异带来的异构性。针对这个问题，他们提出了有效的去中心化分布式训练的系统设计方法。对同步算法，他们利用执行状态差别和通信图的关系，设计了基于队列的同步协议。对异步算法，他们设计了由 Partial All-Reduce，Group Buffer 和 Group Division 等一些列问题解决和优化方案组成的分布式

训练方法，基于它们实现的系统有效的消除了通信瓶颈并达到和当前中心化训练相当或更好的性能。演讲最后，钱学海再次强调：算法和系统紧密相连，算法和系统的协同设计非常重要。

最后，下面这张图呈现了近期用于图计算的 DSA 和 DSL 相关的一些论文之间的关系。

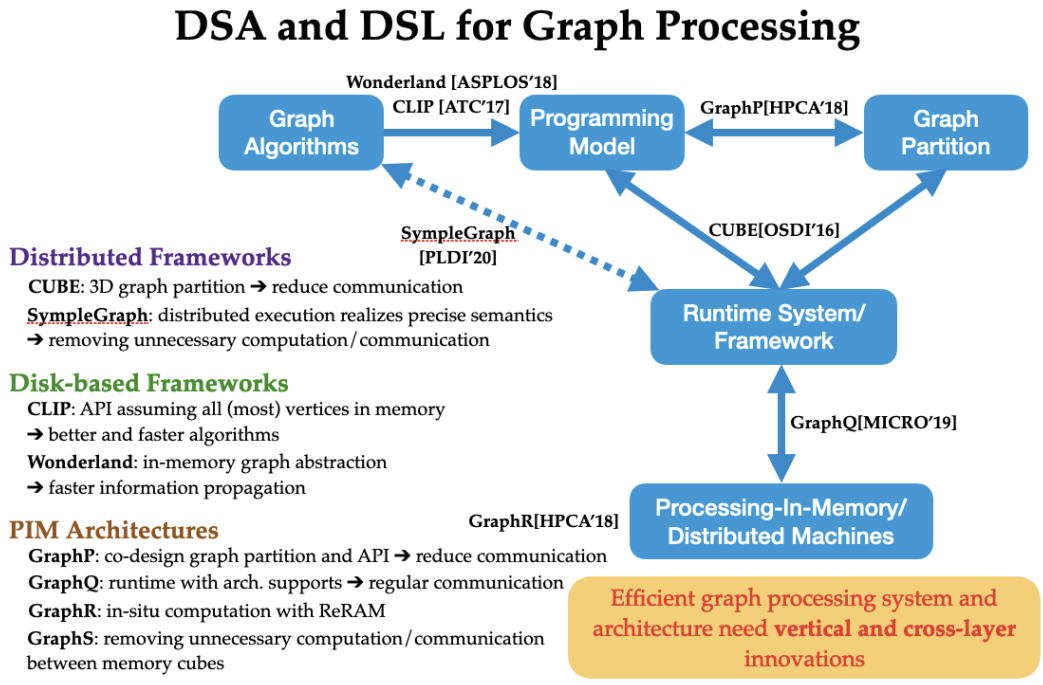


图 21：钱学海团队和合作者关于图计算的 DSA 和 DSL 近期相关论文之间的关系

Q&A

Q：去中心化的分布式训练非常有意思，未来有没有可能像网格一样，每个人的手机都能参与一部分训练，做特别多的机器、特别大范围连接的训练?

A：这个是有可能的。这个场景跟谷歌提出的联邦学习比较类似，他们的思想是说可以在众多的移动设备上随时进行训练。后来我们也研究一下他们的论文，发现很多思路是相通的，所以我们认为有可能把这些移动设备应用到这个场景上。但是，我想说这两个场景在实际考虑的问题上可能有所不同，在这里面我们主要考虑的是性能。在那种情况下，我觉着主要考虑的是隐私和安全。因为如果每个用户或者设备都能够更新模型，会存在一些恶意的用户，从而影响训练结果。

参考文献

[1] Hennessy J L, Patterson D A. A new golden age for computer architecture[J]. Communications of the ACM, 2019, 62(2): 48–60.

[2] Li M, Andersen D G, Park J W, et al. Scaling distributed machine learning with the parameter server[C]//11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14). 2014: 583–598.

[3] Sergeev A, Del Balso M. Horovod: fast and easy distributed deep learning in TensorFlow[J]. arXiv preprint arXiv:1802.05799, 2018.

[4] Lian X, Zhang C, Zhang H, et al. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent[C]//Advances in Neural Information Processing Systems. 2017: 5330–5340.

[5] Luo Q, Lin J, Zhuo Y, et al. Hop: Heterogeneity–aware decentralized training[C]//Proceedings of the Twenty–Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. 2019: 893–907.

[6] Lian X, Zhang W, Zhang C, et al. Asynchronous decentralized parallel stochastic gradient descent[C]// International Conference on Machine Learning. 2018: 3043–3052.

[7] Luo Q, He J, Zhuo Y, et al. Prague: High–Performance Heterogeneity–Aware Asynchronous Decentralized Training[C]//Proceedings of the Twenty–Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 2020: 401–416.

清华大学教授陈文光：人工智能超算的基准测试初探

整理：智源社区 王光华

无规矩不成方圆。放在超级计算机的研发领域，没有一个大家普遍接受的算力评测指标，便难以推动超算迅猛发展。

而现在伴随着人工智能的发展，大量专门针对人工智能而研发的超算系统不断涌现。原来的评测指标，由于种种原因，已经不再适合来评测 AI 超算——“坏的指标导致坏的结果”。

于是如何设计新的 AI 超算基准测试，俨然已经成为超算研究的当务之急。谁能抓住时机，制定出适用于 AI 系统的“好指标”，并将之推广成国际标准，谁便能主导 AI 超算评测的核心工作。

2020 年 6 月 21 日下午，在第二届智源大会“智能体系架构和芯片”专题论坛上，清华大学陈文光教授做了题为“人工智能超算的基准测试初探”的专题报告，介绍了由鹏城实验室和清华大学共同开展的 AI 超算算力基准测试程序的研究进展。

陈文光，国内系统研究的领军人物之一，中国计算机学会副秘书长，曾任 ACM 中国理事会主席、ACM 中国操作系统分会 ChinaSys 主席、ACM 通讯中文版主编等。

陈文光的演讲分为四部分，首先分析了为什么需要人工智能算力基准测试程序，指出好的 Benchmark 对于领域发展具有引领作用；接着梳理了人工智能基准测试的关键要素及程序特点，并在对比了现有 AI 算力 Benchmark 的基础上，陈文光详细介绍了基于 AutoML 的测试方案并以及在广州超算中心、鹏城实验室进行测试的实验情况；最后，他展望了这个领域，在未来的几个工作重点方向。

一、为什么需要人工智能算力基准测试程序?

我们先来看传统的超算。

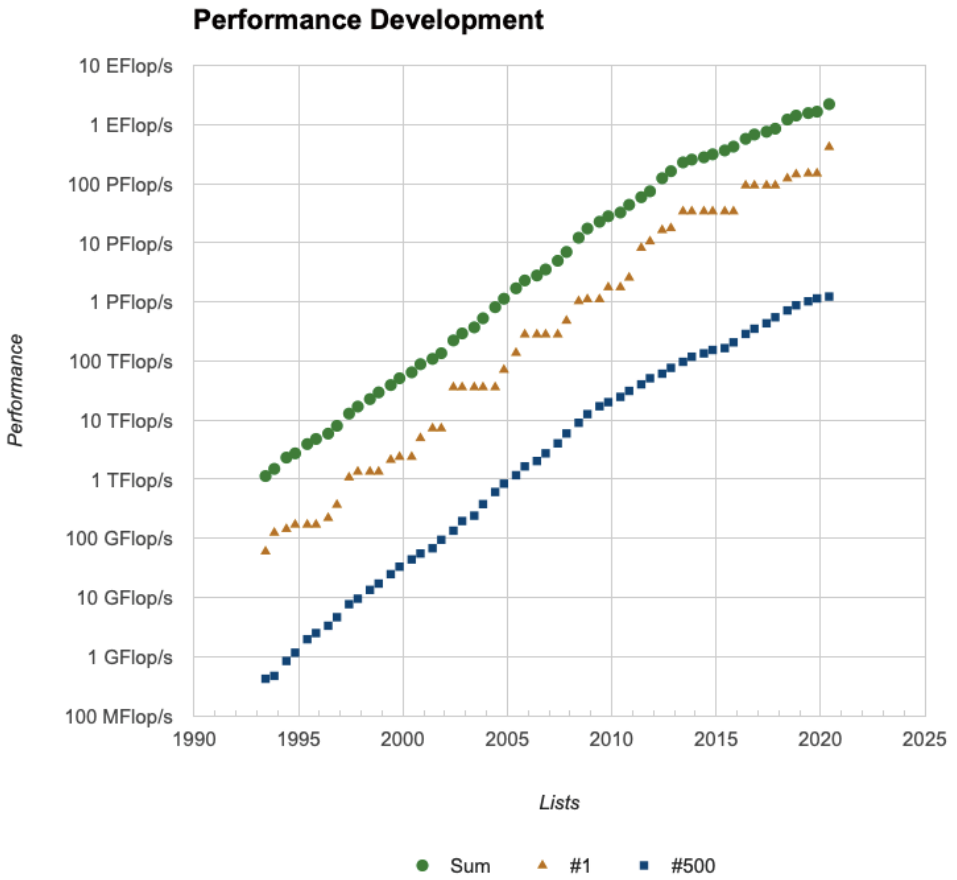


图 1：高性能 TOP500 计算机系统（https://www.top500.org/statistics/perf-devel/）

上图是每年通过基准测试评选出来的高性能 TOP500 计算机系统的性能表现。三条线中，最下面的是 TOP500 中最慢的性能，橙色线是第一名机器的性能，绿色线是前 500 名性能之和。

从这幅图可以看出，任何一台机器都对应一个值。这个值是高性能计算领域常用的标准，即 Linpack 测试值，通过用高斯消元法求解 N 元一次稠密线性代数方程组的测试，评价高性能计算机的（浮点）性能指标。

现在有很多人工智能应用和系统在做人工智能计算，例如用寒武纪处理器、GPU 等构成的系统。

对于多样的智能计算系统，能不能像传统的高性能计算机一样，用一个简单的指标来回答哪一套系统的人工智能算力更强？我们能不能用一个数字来表示这个指标，给任意一个机器一个确定的数值，并在通过分析这些数值来预测未来一段时间内的发展趋势？

好的指标可以引领一个领域的健康发展，比如 Linpack 很大程度上就主导了对高性能计算机指标要求，从而鼓励更有针对性的设计。

但反过来说，一个不好的指标会阻碍领域的发展。传统的高性能计算主要使用双精度浮点运算，而人工智能计算里面训练主要是单精度浮点数以及 16 位浮点数，推理甚至可能只用 8 位，还可以通过网络裁剪和量化做到更

低的位宽。如果我们用双精度浮点数的性能来评价一个人工智能机器的性能，显然是不合适的。不好的指标有可能引领这个领域往错误的方向发展。所以传统的 Linpack 指标本身并不能直接用来评测一个人工智能计算系统的算力。

二、人工智能算力基准测试的主要因素

设计人工智能算力的基准测试，需要考虑以下几个因素:

首先，怎样实现具有人工智能意义的全机规模测试? 现阶段，全机规模指的是几千块甚至上万块的加速卡，未来的机器有可能要准备 10 万块甚至更多的加速卡。目前，还没有单个人工智能的训练任务能够达到计算机的全机规模。即使勉强把一些人工智能应用跑到这个规模，得到的结果可能是收敛速度并没有变得更快，或者准确率上并没有改进，反而变得更差了。所以强行用全机做人工智能并没有意义，并不会有训练时间和准确率的改进，这也不是一个好的测试。

其次，如何做到规模可变? 如果要测规模变化巨大的人工智能系统，从几块、几十到几千上万块加速卡，都能对它们进行客观测试并得出分数，这个测试就必须规模可变。Linpack 的规模是方程组的阶数 N，通过改变 N 便可以改变测试规模。如果要测试规模范围巨大的人工智能集群计算机，测试程序必须是规模可变的。

第三，如何考虑准确率? 这个点很重要！对于人工智能来说，准确率受限的因素很多，包括训练方法、网络结构以及各种参数。如果很快就能算到给定的很低的准确率，那么测试就没有意义； 另外有一种思路是把准确度也计到分数里面去。

基于以上考虑，以及现有超算算力 Benchmark 成功案例的分析，陈文光认为，人工智能算力基准测试程序应该包含以下四个特性:

1. 一个分数。从 Linpack 的成功，我们可以得出一个结论，即一个分数非常重要。而且分数应该近似线性，当机器规模增大一倍的时候，这个分数最好提高一倍； 当然不可能正好增加一倍，达到百分之七八十，都算比较好的线性。不能机器增加，分数反而下降。

2. 可变问题规模。这是设计人工智能基准测试程序最重要的一个考虑。可变问题规模有两点： 第一，它可以适应单卡内存规模的变化，一块加速卡本身内存大小存在很多变化，比如从 8G 到 32G 等，怎么能够适应这个内存规模的变化； 第二，怎么适应从 1 块卡到 1 万块卡的变化?

3. 计算要有人工智能意义。以 HPL-AI 来说，仅使用人工智能的运算精度是不够的，人们更希望计算本身是具有人工智能意义的。现在最流行的 AI 运算是神经网络运算，可以认为具有一定的人工智能意义。

4. 多机通信可以少，但不能没有。比如，两台机器相隔一万公里，各算各的，最后得出一个分数，这样显然可以达到线性加速比。但这个事情是不对的，因为这没有对系统的通信（包括 IO、存储等各方面）的协同能力做出挑战，可能导致系统很多任务并不能完成。测试程序应该在紧耦合的机器上测评才能得到一个比较好的性能，而不是完全各干各的。

三、人工智能基准测试程序

目前，在人工智能性能基准测试这个领域已经有很多探索，比如 MLPerf，小米的 mobile AI bench，百度的 Deepbench，中国人工智能产业发展联盟的 AIIA DNN Benchmark 等。其中，MLPerf 是国际上大家比较认可的机器学习的标准。此外，混合精度的 HPL–AI 是在全双精度的 Linpack 基础上把它改成混合精度的算法，即先用半精度 / 单精度计算得到一个近似解，再用双精度计算得到一个更精确的解。但其实它除了用到一些半精度 / 单精度的运算外，实际上和 AI 并没有直接的关系，HPL–AI 更多是用于混合精度算法的研究。

图 2：现有 AI 性能 Benchmark 程序

已有的人工智能基准测试程序的存在两个主要问题。首先，除 HPL–AI 外都是固定规模的，没有解决可变规模的问题； 其次，它们都采用多个应用，这个思路有点像 NAS 等测八个应用。它们都想完整的反映系统的特点，但是最后公众能够理解和记住的还是 HPL（High Performance Linpack）。一个系统一个分数，这是公众最容易接受和理解的东西。

那么如何设计符合 AI 的算力 Benchmark 呢?

一种方案是利用 AutoML 来做基准测试，通过算法自动搜索合适的神经网络模型结构和超参数，找到特定任务效果比较好的优化的解。这个方案的原始思路是清华大学张悠慧老师提出来的。它是一个类似于搜索的问题，因此需要的计算资源很高，而且做每一次搜索，得到一个神经网络，要对这个模型进行训练，然后才能知道这个搜索出来的模型好不好。

AutoML 包括两方面，一个是网络结构搜索，一个是超参数搜索。网络结构搜索主要是改变当前的网络结构；超参数搜索更多是在每一层的连接参数上做一些变化。这两个是正交的，也就是说搜出一个网络结构以后，再对它进行超参数搜索。

AutoML 具备足够的并行度，通过多次测试或统计平均改善搜索的随机性问题。实验得出，整体上找到解的优劣程度和搜索消耗的计算量基本呈正比，也就是说投入的计算量越多，搜索的空间越大，统计上可以获得的解也越好。

AutoML 做人工智能基准测试程序有这样的特点:

1) 任务自身是深度学习任务设计中的关键环节，且任务能够直接体现超算系统在 深度学习方面的综合性能 (包括训练、推断) ；
2) 找到的解的优劣程度随着搜索所消耗的计算量而逐渐提升;
3) 可以换算成计算效率 (%)

- **AutoML框架：**
  - **Master节点将模型历史及其达到的精度发送至Slave节点**
  - **Slave节点根据模型历史及其精度，搜索出一个新的模型，并进行训练**
  - **Slave节点将根据某种策略（如连续10个Epoch的测试集精度没有提升）停止训练，并将此模型及其达到的精度发送至Master节点**
  - **Master节点接收并更新模型历史及精度**

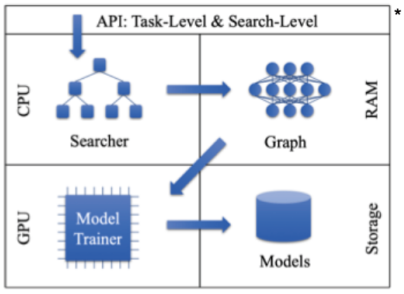


图 3：结构搜索工作流程

图 3 是结构搜索的工作流程。在 AutoML 框架中，Master 节点将模型历史及达到的精度发送至 Slave 节点，slave 节点根据模型历史及其精度搜索出一个新模型并进行训练，Slave 节点将根据某种策略停止训练，并将此模型及其达到的精度发送至 Master 节点，Master 节点接收并更新模型历史及精度（参见文献 1）。

1. 在广东超算中心的初步测试

首先，在广东超算中心对 16 块卡、32 块卡做了测试。从图 4 结果可以看出 32 块卡训练精度达到 78.9%，而 16 块只达到 76.87%。卡数越多 flops 越高，从 16 块到 32 块，卡数增加 1 倍，但是 flops 是 3 倍多，这是传统超性能计算里面很少见到线性加速比。原因在于网络结构搜索时开始的起点是比较小的，搜索到越来越大的网络时候，随着网络越来越大运算效率逐渐增高，就会得到这样一个超线性的加速。从图 5 可以看出 32 个 GPU 时当遇到后面比较大的网络的时候，它的平均 flops 比前面的高一些。

GPU总量	16	32
训练时间	24小时	24小时
训练模型数	213	401
最优模型精度	76.87	78.90
FLOPS	38.83 TFLOPS	128.50 TFLOPS
GPU效率	15.5%	25.6%

图 4：广东超算中心 16 块卡、32 块卡测试结果

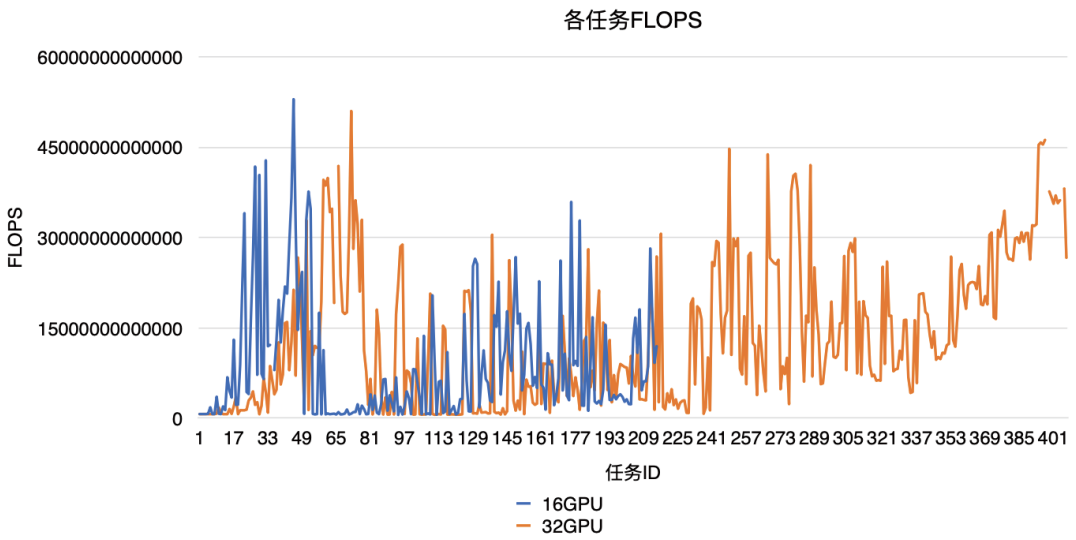


图 5：广东超算中心各任务的 FLOPs 性能

2. 在鹏城实验室的初步测试

在鹏城实验室做了最多到 128 卡的测试。

GPU总量	32	64	128
训练时间	24小时	24小时	24小时
训练模型数	1014	1696	3059
最优模型精度	81.56%	81.69%	82.78%
FLOPS	200.9 TFLOPS	373.3 TFLOPS	633.9 TFLOPS
GPU效率	40.0%	37.2%	31.5%

图 6：鹏城实验室 32、64、128 卡测试结果

图 6 表明，最优精度 32 块卡到 81.56%，128 块卡时到了 82.78%。Flops 是增加的，效率是下降的，这和 HPC 的经验比较相似。随着规模增大，同步、负载等开销不均衡，会带来一些问题让性能下降。另外测试的精度为 82% 还有很大提高的空间，而人类在这个上面已经可以达到 99% 了，两者之间的差距是非常大的。当然，这和采用的搜索框架、搜索起点有关。

任务的运行的 Flops 的情况、参数量如图 7、图 8 所示。

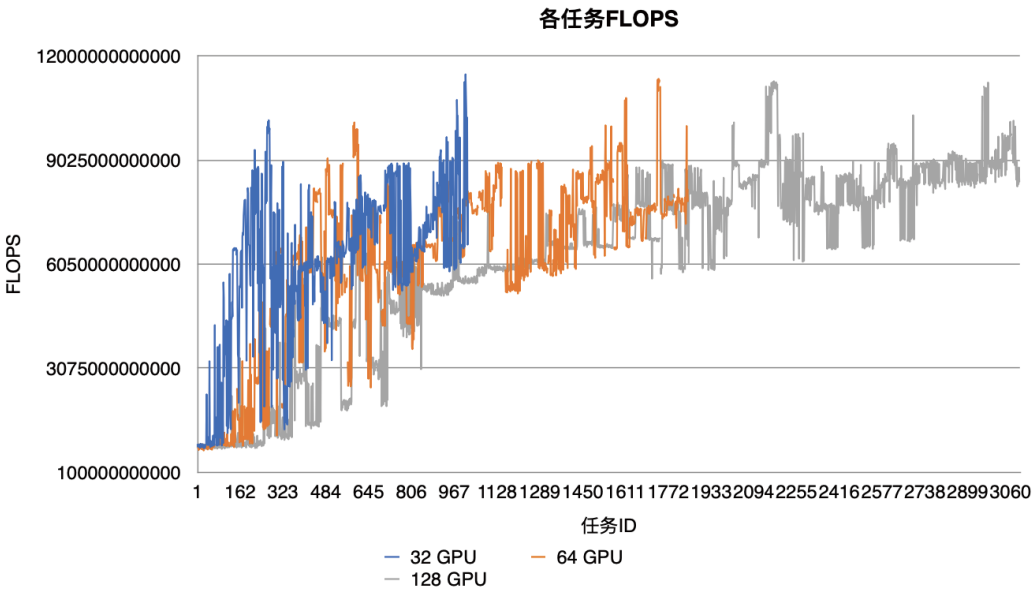


图 7：鹏城实验室各任务的 FLOPs 性能

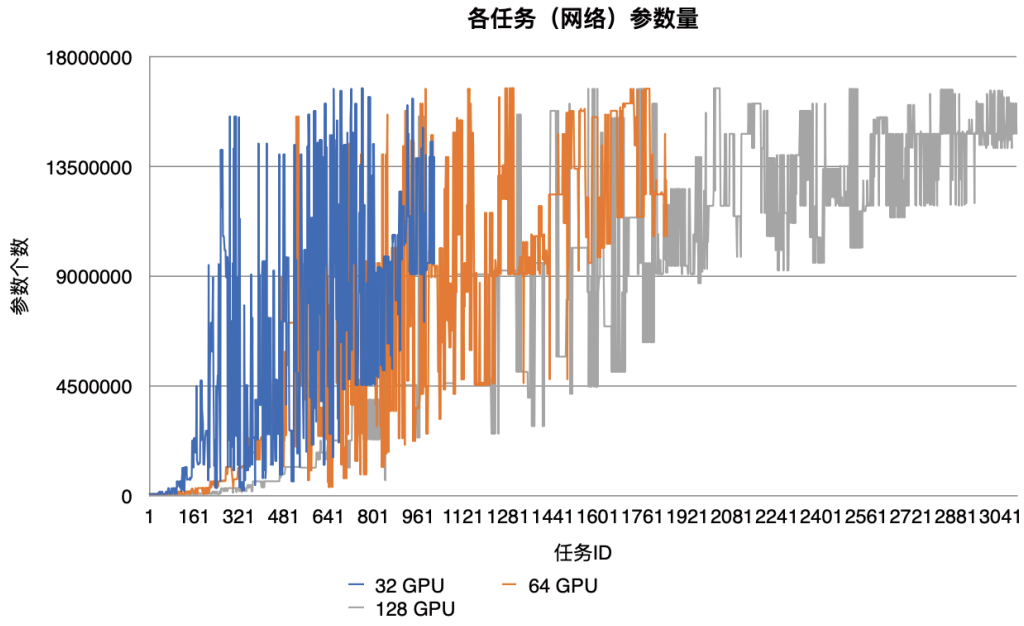


图 8：鹏城实验室各任务的参数量

用 Network Morphism 算法后续生成神经网络时，需要将之前所有的历史都看一遍。所以它的生成开销跟历史个数 N 有关系，当 N 不断的增加，到后期开销相当大。这也是后续需要解决的问题。

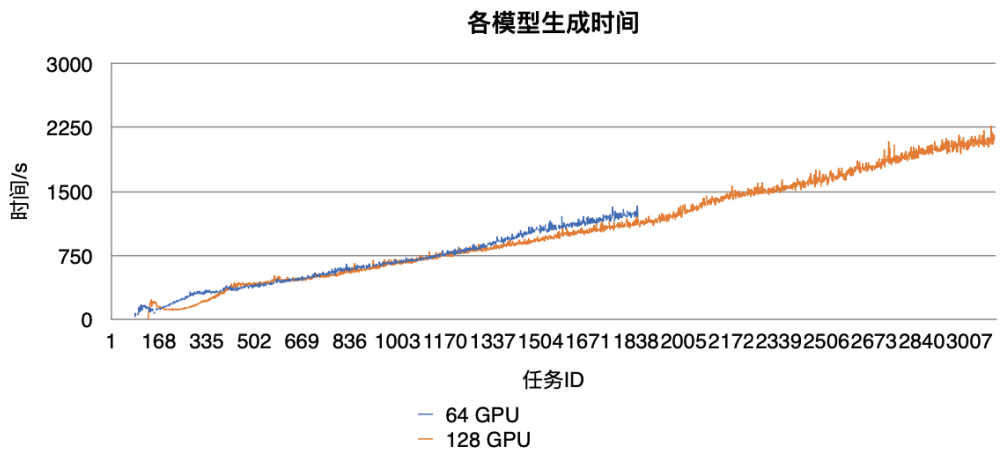


图 9：Network Morphism 算法各模型生成时间

怎么解决这个问题？如果生成网络时间太长，我们可以增加一个并行度，即在当前网络结构上进行超参数搜索。超参数搜索的好处是搜索时网络结构相对比较稳定，也就是说计算性能差别不大。另外，超参数搜索的生成时间也比较稳定，生成新的超参数的时间和已经评测的超参数个数是无关的，仅和搜索空间相关。超参数搜索的坏处是并行度并不高，所以需要把超参数搜索和结构搜索结合起来，才能进一步提高精度。

GPU总量	32
训练时间	33小时
训练模型数	200
FLOPS	193.2 TFLOPS
GPU效率	38.4%

图 10：超参数搜索的测试结果

图 10 是超参数搜索的测试结果，可以看到它的每个任务在网络结构上做超参数搜索的性能相对很稳定，生成时间也很稳定，这是很好的特点。

四、总结与未来工作

人工智能超算需要在计算精度上是符合人工智能特点的，计算内容具有人工智能意义，而且规模是可变的，能够适应很大范围的机器规模。

展望未来，陈文光指出后续要做的事情包括：

1. 进一步提高模型生成与搜索的效率。
   1）**并行化模型生成算法。**现在是在单线程上做的，但是 CPU 几十个核，如果可以并行做会得到更好效果。

2）**优化模型生成算法。**利用其他的模型生成的算法，比如遗传算法，不用保留全部的历史，只需要保留当前一代即可。

3）**选择更好的优化起点。**当前最后得到的 accuarcy 和人类相比是很低的，因为从 60% 多这个非常差的起点开始搜索，如果我们从相对比较好的起点上去开始搜索，就能够获得更好的最终效果。

4）**使用更大规模的数据集。**现在使用的 CIFAR-100 数据规模比较小，希望用更大规模的数据集，这样对整个系统的 IO、网络、通信也会有更好的测试。

2. 结合参数搜索。

3. 在多种和更大平台上开展测试。

4. 做流程管理。借鉴 TPC-C 等都有相对完整的运行规则、辅助工具、报告和审查机制，希望以后能够真正的做一个相对公平和客观的测试结果发布，能够对 AI 超算的发展起到一定的支撑作用。

参考文献

1. Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-Keras: An Efficient Neural Architecture Search System. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'19). Association for Computing Machinery, New York, NY, USA, 1946—1956.

北京大学杨玉超：非线性忆阻神经网络

整理：智源社区 王光华

杨玉超，北京大学微纳电子学系研究员、国家杰青获得者，长期从事类脑计算、智能硬件、忆阻器等相关领域的研究工作，迄今共发表 Nature 子刊等期刊和会议论文 90 余篇，撰写中英文专著 5 章，主持重点研发计划、霍英东教育基金会青年教师基金等项目，曾获科学探索奖、求是杰出青年学者奖、Wiley 青年研究者奖、《麻省理工科技评论》中国区 35 岁以下科技创新 35 人等奖项。现担任 Nano Select 等多个期刊副主编、编委等、中青科协信息与电子专业委员会副秘书长 / 理事、中青科协提案专门委员会理事、中国电子学会青工委委员。

类脑智能、神经形态器件与系统研究对于实现复杂、通用人工智能，提高计算系统的智能与能效水平具有重要的学术意义与应用价值，是后摩尔时代突破传统冯诺依曼架构瓶颈、推动半导体技术持续发展的关键技术。

针对这一主题，在第二届智源大会“智能体系架构和芯片”专题论坛上，北京大学研究员杨玉超做了题为“非线性忆阻神经网络”的专题报告。

杨玉超的演讲，首先介绍了类脑计算与忆阻器非线性动力学及具有非线性动力学的人工神经元，阐述了神经元与突触均基于忆阻器的神经网络与硬件系统，该系统能够充分发挥神经形态器件的优势，优化神经网络的智能处理水平与能效，并在多种智能任务中演示验证；此外，基于神经形态器件混沌、时域动力学等多重非线性物理性质，它可以实现基于神经形态器件的新原理神经网络，支撑复杂优化问题的高效求解。

一、类脑计算与忆阻器非线性动力学

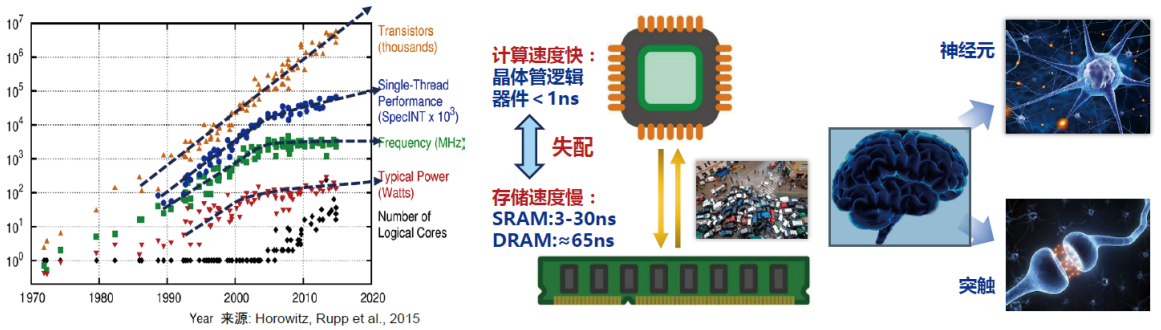


图 1：后摩尔时代面临的挑战

随着后摩尔时代的到来，通过不断缩小器件尺寸的方法提高算力的传统路线越来越难以持续，难以满足人工智能和后摩尔时代越来越高的算力需求，同时冯诺依曼架构面临存储墙、功耗墙等非常关键的问题。要想进一步提高算力，就需要新的驱动力，比如忆阻器等比较典型的新器件和借鉴人脑存算一体等原理的类脑计算架构等的新的计算架构。目前基于忆阻器的类脑计算架构取得了一定的进展。

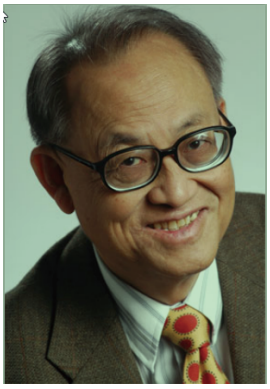


图 2：忆阻器发明人、非线性电路奠基人蔡少棠先生

忆阻器作为电阻、电感、电容之外人类发现的第四种无源电路元件，具有丰富的动力学特征，它跟静态电阻是有区别的。然而目前类脑架构中通常是将忆阻器作为非易失性存储器。从非线性动力学角度重新来看忆阻器并充分利用这些动力学特性能够给计算或者信息处理带来全新的可能性。

二、具有非线性动力学的人工神经元

忆阻器中的非线性动力学引入到计算系统当中以后给计算带来哪些额外的新的功能。

2.1 神经元的“ 非线性动力学”

传统的观点认为生物神经元是泄露累积发放单元，即满足 LIF 模型（Leaky–integrate–fire model）。累积发放单元意味着它可以对时间和空间接受的信息进行整合，当达到特定阈值时发放动作电位，是比较简单的操作。但是近来的神经科学研究发现，单个神经元的计算能力可能超出传统观念上认为的能力，很多情况下它可以执行非常复杂的逻辑计算甚至是算术运算。所以如何在人工神经元当中也实现一些复杂的非线性操作?

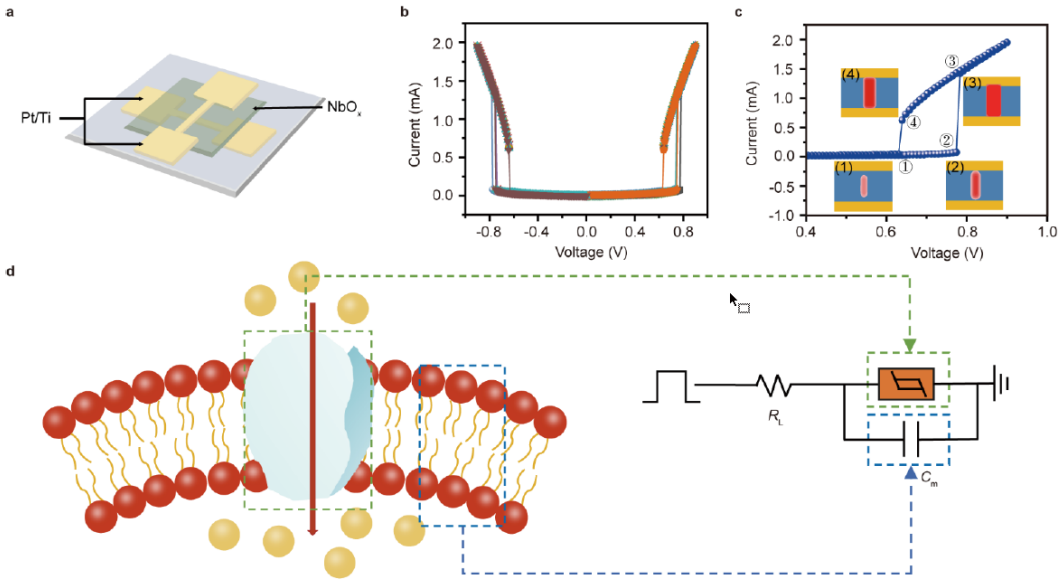


图 3：基于 NbOx 忆阻器的脉冲神经元

杨玉超团队利用氧化铌忆阻器构建脉冲神经元。对氧化铌忆阻器施加正偏压或者负偏压的电压，当超过特定阈值可以触发高阻态到低阻态的转变，可见它是不依赖于偏压极性的。但是，这个转变是易失性的，只要撤掉外界激励以后，器件可以自发从低阻态转变回到高阻态。为了构建脉冲神经元，需要构建一个经典的振荡器这样的神经元电路结构，即把氧化铌忆阻器与一个电容并联再和一个负载电阻串连。并联的电容实现电荷累积的功能，用来模拟细胞膜的功能，负载电阻用来调控整体氧化铌忆阻器输入信号的强弱，相当于调节连接突触整体强度，氧化铌忆阻器用来模拟离子通道的动力学过程。

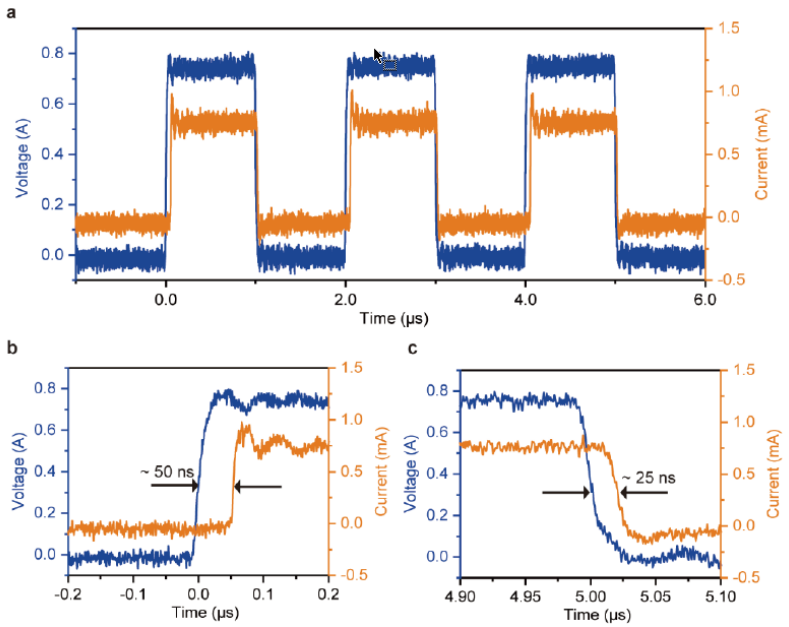


图 4：状态切换瞬态响应 [6]

在该脉冲神经元结构中氧化铌忆阻器是非常高速的器件，从高阻态到低阻态转变过程要快于 50 纳秒，而从低阻态转变成高阻态过程快于 20 纳秒而且是自发的，只要撤掉外界的电压，器件就自动回到高阻态。另外，这种器件具有非常不错 cycle–to–cycle 和 device–to–device 一致性。这种时间和空间上的一致性，对于工程化是比较重要的。

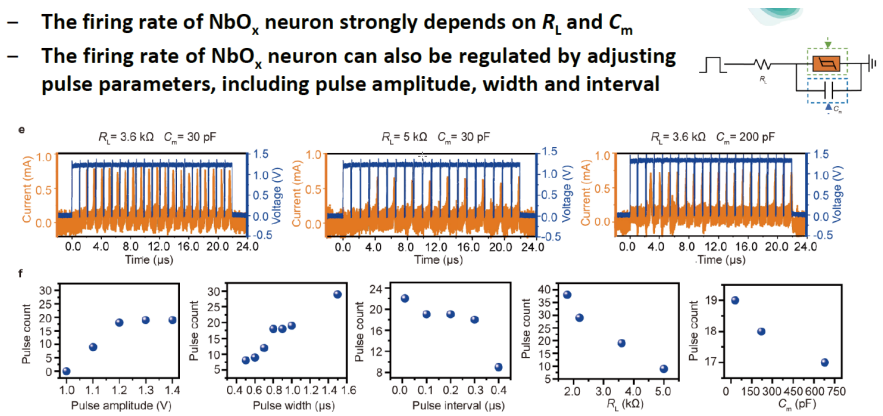


图 5：脉冲神经元的响应 [6]

通过电路的参数以及电学激励的参数可以灵活有效地调控该脉冲神经元的振荡或者脉冲发放行为。该电路当中两个最重要的电路参数，即负载电阻和并联电容。实验证明，负载电阻控制着整体输入信号强度，负载电阻越大相当于加到氧化铌忆阻器上的信号越弱，会降低整体发放频率。同样，并联电容控制电荷积分时间，电容越大电荷积分达到氧化铌阈值的时间就会延长，发放频率也会下降。除了电路参数，施加的电学激励的参数也可以有效控制脉冲发放频率，包括脉冲的宽度、幅值、不同脉冲之间的时间间隔。增加幅值或者宽度相当于增强整体输入信号会增强发放频率。如果增加不同脉冲之间的时间间隔，脉冲发放频率会下降。虽然，脉冲串中脉冲个数、幅值、宽度完全一样，只是增大不同脉冲之间的间隔，发放频率就会下降。说明不同脉冲之间有非常重要的泄漏动力学，就是电容上的电荷在施加脉冲间隔这个时间范围内缓慢通过氧化铌泄漏。随着时间的泄漏是一个非常重要的物理性质，它使得脉冲神经元不但具有空间上整合能力，还可以具有时间上信息整合的能力。这点可以在行为层面上比较清楚地观察到。

2.2 神经元的时空信息整合能力

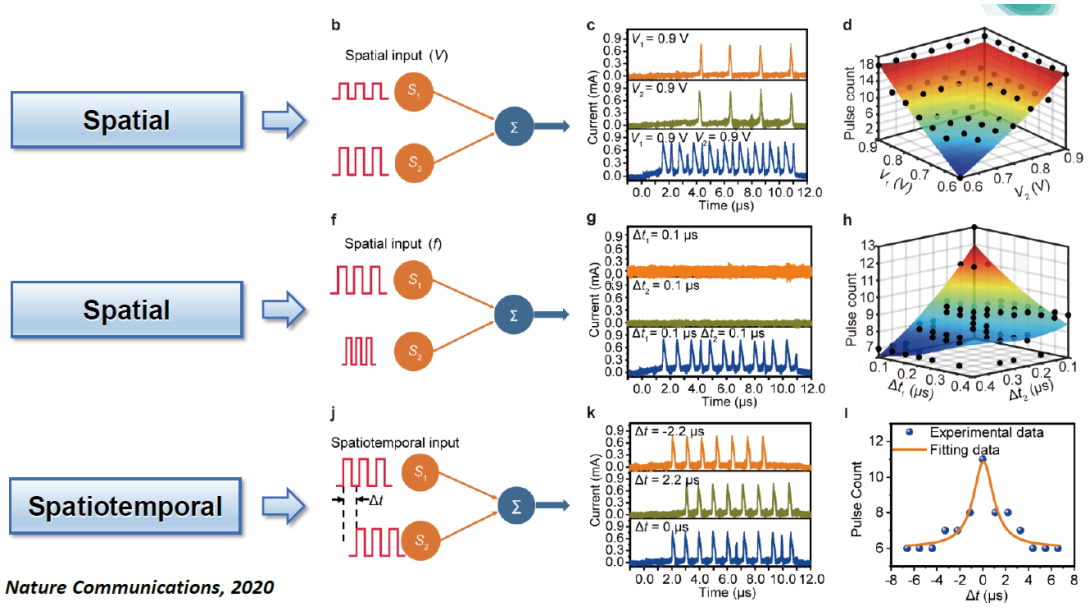


图 6：氧化铌脉冲神经元的时空动力学 [6]

对于脉冲神经元来讲，空间信息整合能力是指从不同树突上接收到的幅值调制的信号或者频率调制的信号会共同在神经元的胞体上进行累加，从而在整体的胞体上引起更强的脉冲发放频率。除了空间整合，事实上不同脉冲时间关系也可以影响胞体的发放效果，获得时空整合的动力学特性。如图 6，当两个脉冲串之间的相位或者时间差不同的时候，可以得到不同的发放特性。在特定情况下设置相位差为零时，整体发放频率是最高的，但是相位差大于零或者小于零发放频率都会下降。总之，不仅能对空间上信息进行整合，而且在时间维度上同时对信息进行了整合，这对于计算来讲是非常重要的。

2.3 神经元的非线性逻辑

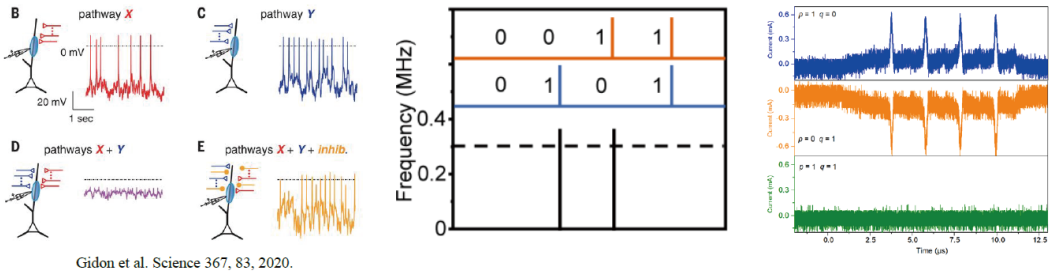


图 7：氧化铌神经元的动态逻辑

从逻辑计算角度来讲，空间信息整合对逻辑计算来讲是一个简单的“与”逻辑，不同树突上接收的信息同时都比较强的时候在胞体上才能实现一个比较强的刺激，即一一得一。当然，如果调控阈值比较低时是逻辑上的“或”，即零零得零。但是不管是“与”逻辑还是“或”逻辑都是线性可分的逻辑。线性不可分的逻辑操作是“异或”逻辑，异或逻辑问题在历史上对神经网络技术的发展起了非常重要的影响，正是因为线性神经网络无法解决异或逻辑的问题，当时马文明斯基写了非常著名的论文认为神经网络技术用处不大，所以使得神经网络技术经历长时间的低谷期。但是，2020 年 Science 上发表了一篇神经科学研究论文发现单个生物神经元胞体上可以实现传统线性不可分的异或逻辑，这个影响力非常大。同时，利用氧化铌脉冲神经元也可以在单个神经元上实现线性不可分的异或逻辑。氧化铌忆阻器有个非常重要的特点就是不依赖于电压极性，不管树突输入的是正电压还是负电压，只要超过特定强度就可以引起脉冲发放。如果在两个树突上同时施加一个正电压和负电压的强刺激，它们在电容上所引起的电荷积累的效果抵消，即可在单个神经元胞体上实现线性不可分的异或逻辑。这证明了利用人工神经元也可以在单个计算单元上实现非常复杂的非线性操作，这在网络层面是非常需要的。

2.4 神经元的增益调制

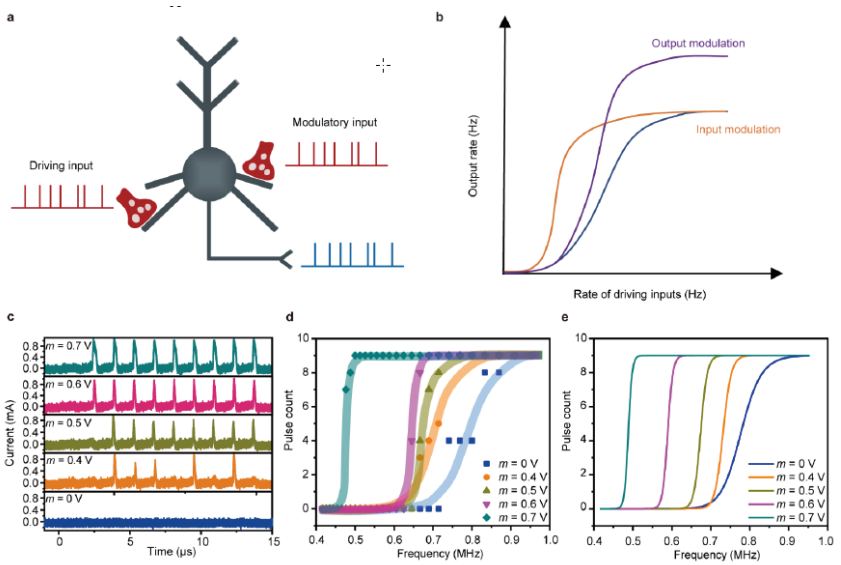


图 8：氧化铌忆阻器的增益调制 [6]

除了时空整合动力学和非线性操作之外，人工神经元还可以获得的非线性是增益调制的功能。生物神经元也有增益调制的功能，对于胞体的多个输入有些比较强的刺激是 driving input，有些则是 modulatory input。那么在已有的 driving input 基础上再增加 modulatory input 的情况下对生物神经元是什么影响？生物学研究发现这种调制是增益的调制，可以是加法型的，也可以是乘法型的。加法型是在原有的基础上增加新的调制输入，会使整个神经元响应曲线平移，使得可以用更低的 driving input 就可以引起神经元的发放，证明了 modulatory input 的效果降低了原来所需要的输入信号的强度。生物神经元的增益在响应曲线上的具体表现就是斜率，但是在很多情况下，响应曲线并不是简单的平移，而是曲线的斜率发生变化。当增加新的调制输入的情况下，在神经元的胞体上会获得非线性的响应，表现为斜率增加、增益变大，这些复杂的动力学过程在人工神经元上也可以得到实现。实验的结果表明随着新增加的 modulatory input 增强，斜率可以变得越来越大，相当于实现乘法型的增益调制功能，在网络层面上可以看到它为计算带来很多新的功能。

三、全忆阻神经网络

非易失性忆阻器可以很容易构建传统的人工突触，易失性的氧化铌忆阻器具有丰富动力学特性可以用来构建脉冲神经元。杨玉超团队结合具有不同动力学特性的忆阻器，分别实现神经元和神经突触，构建神经元和神经突触全都基于忆阻器的全忆阻器神经网络。

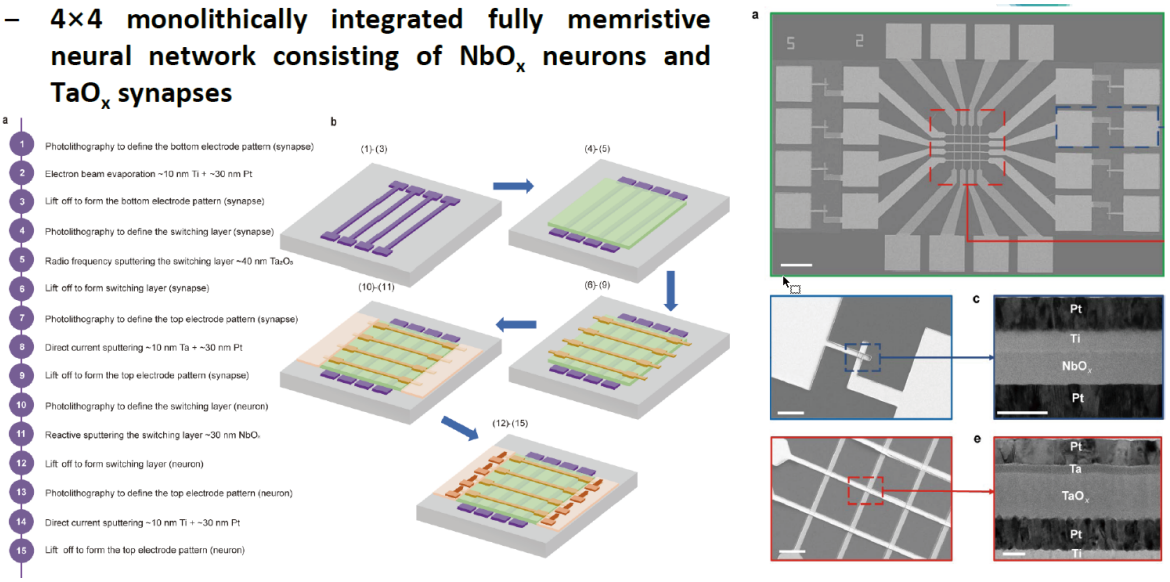


图 9：全忆阻器脉冲神经网络 [6]

如图 9 是在硬件上实现了 4 × 4 的一体化集成的全忆阻器神经网络。其中，红色框内的核心是一个 4 × 4 非易失性的氧化钽人工突触，它采用模拟的状态存储不同的权值，起到存储突触连接的作用。而在每一个输出端上，是由易失性的氧化铌忆阻器所组成的人工神经元。这样以非常高的集成度把神经元和神经突触完全以忆阻器的形式集成在了一起。

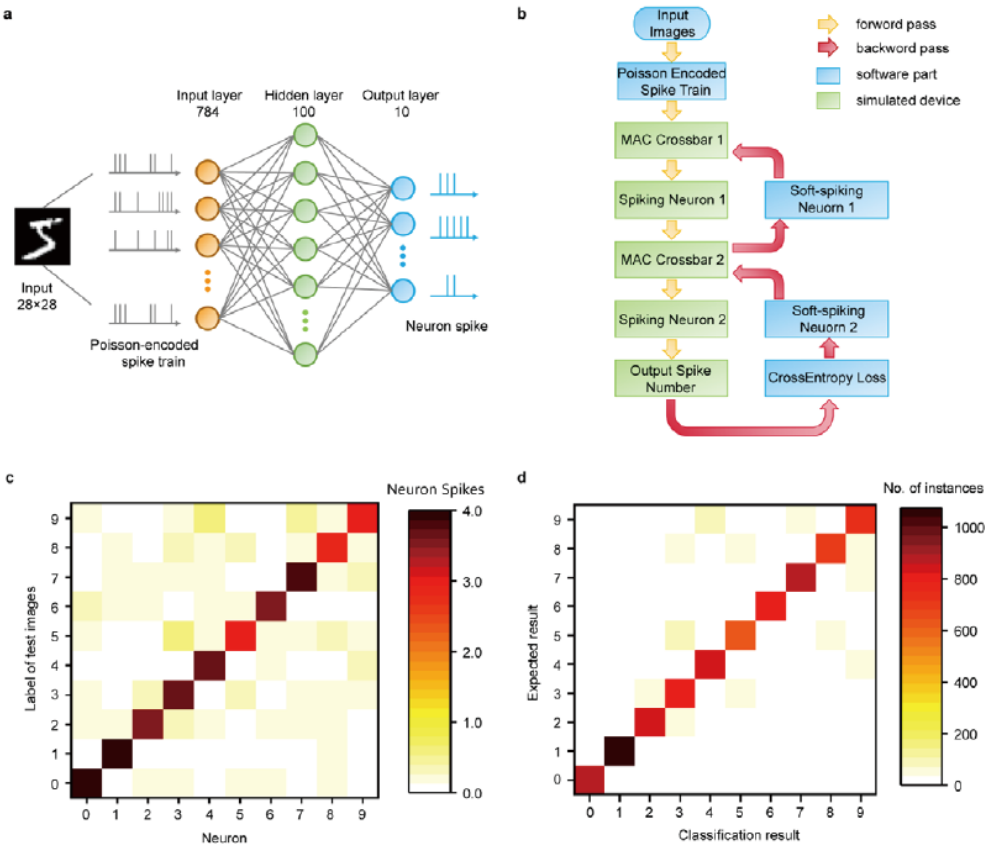


图 10：三层脉冲神经网络的仿真 [6]

基于该全忆阻神经网络，可以很方便进行在线训练，同时，还可以构造更大的 SNN 的网络实现更复杂的模式，比如手写体识别等等。但是，虽然这里使用脉冲神经元，所有信息都是脉冲化的，但是它的训练算法还是类似 BP 算法这样传统的人工神经网络训练算法。基于人工神经元独特的非线性动力学特性，能够进一步实现有别于传统神经网络或者传统计算的新的计算功能。

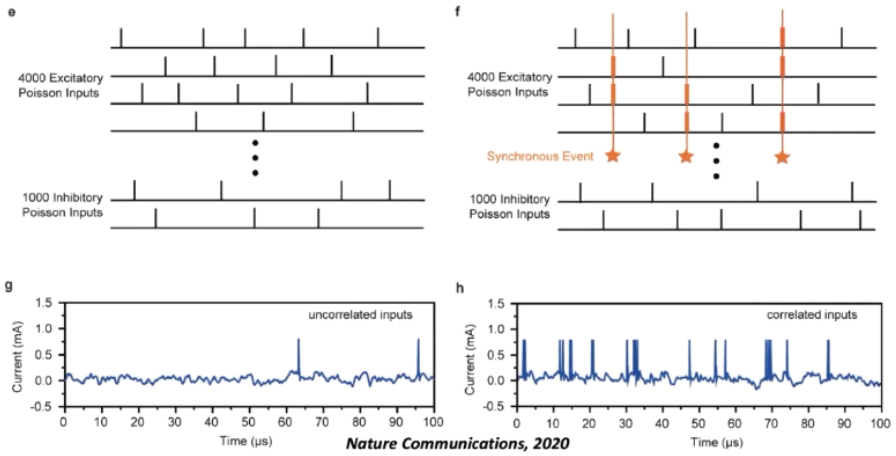
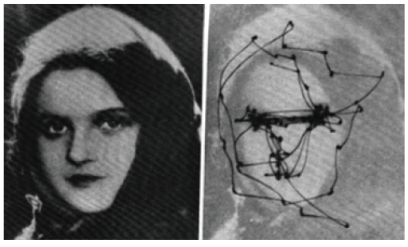
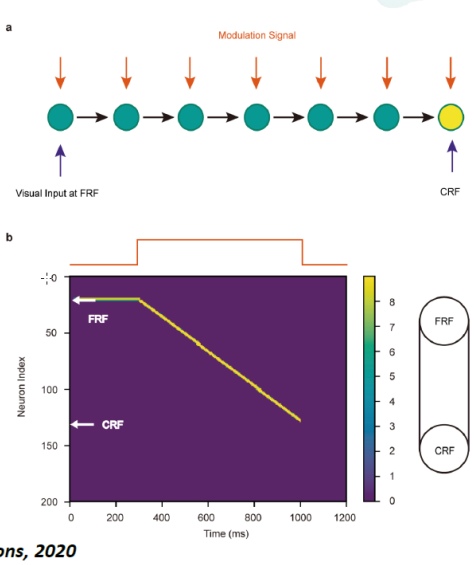


图 11：基于时空动力学特性的同时性检测 [6]

氧化铌人工神经元中重要的动力学特性是对于时空信息整合，依赖于不同树突信息相互的相位和时间先后关系，神经元具有不同的响应。利用该物理性质，可以非常方便地用全忆阻神经网络实现脉冲同时性检测。把不同的脉冲序列输入到不同的氧化钽突触，根据它们之间的时间和相位关系，这些氧化钽突触所连接的脉冲神经元就会给出相应的响应。比如，这些脉冲序列完全同步时，在氧化铌脉冲神经元上得到非常强的发放，在它们完全异步时给出非常弱的发放，或者它们比较随机时得到比较弱的发放。从根本上这是因为氧化铌脉冲神经元具有时空信息整合的特性。当然，把这种同时性检测扩大到更大规模，包含 4000 组兴奋性的脉冲输入、1000 组抑制性的脉冲输入的更大系统。这样 5000 组脉冲序列当中如果包括仅仅百分之零点几的同时性脉冲序列操作，同样可以在氧化铌的脉冲神经元上给出明确的响应。该方法能够在物理上高效实现复杂同时性检测，利用的是时空整合动力学特性。



- **Memristive networks consisting of the spiking neurons with multiplicative gain modulation can achieve receptive field remapping**
- **The RF remapping in hardware could significantly enhance the stability of artificial visual systems in real world**



Nature Communications, 2020

图 12：用于感受野重映射的增益调制

氧化铌忆阻器另一个重要的动力学特性是实现乘法的增益调制。借鉴人的视觉系统，乘法的增益调制在增强人工视觉稳定性方面有很多应用。人在观察世界的时候，比如我们在看一幅图像时图像是稳定的，但是事实上我们的眼睛是在进行非常频繁的眼动，在不断地扫描我们视野范围内不同的点。但奇怪的是，我们尽管眼睛在频繁的扫描，看到的视野没有晃动，也没有感到眩晕，得到的却是稳定的感受野，这是因为生物神经元具有增益调制的功能。其实，生物神经元在眼动之前，眼动趋势已经作为增益用来调控生物神经元的感受野，使其感受野发生了拉长，这种拉长使未来的感受野重新 remapping 到现在的感受野上，我们才感受到整个视野是稳定的。利用类似的机理可以基于氧化铌脉冲神经元构建一维的神经网络，一维神经网络里面每一个神经元都具有乘法增益调制的非线性。这样的网络有调制信号到达的时候它的感受野发生类似的拉长，实现 receptive field remapping 从而增强感受野的稳定性。针对人工视觉系统在应用中不可避免面临机械振动或者外界环境不稳定，如果在硬件上能够支持重映射，就可以在硬件层面上增强视觉系统的稳定性。

四、暂态混沌神经网络

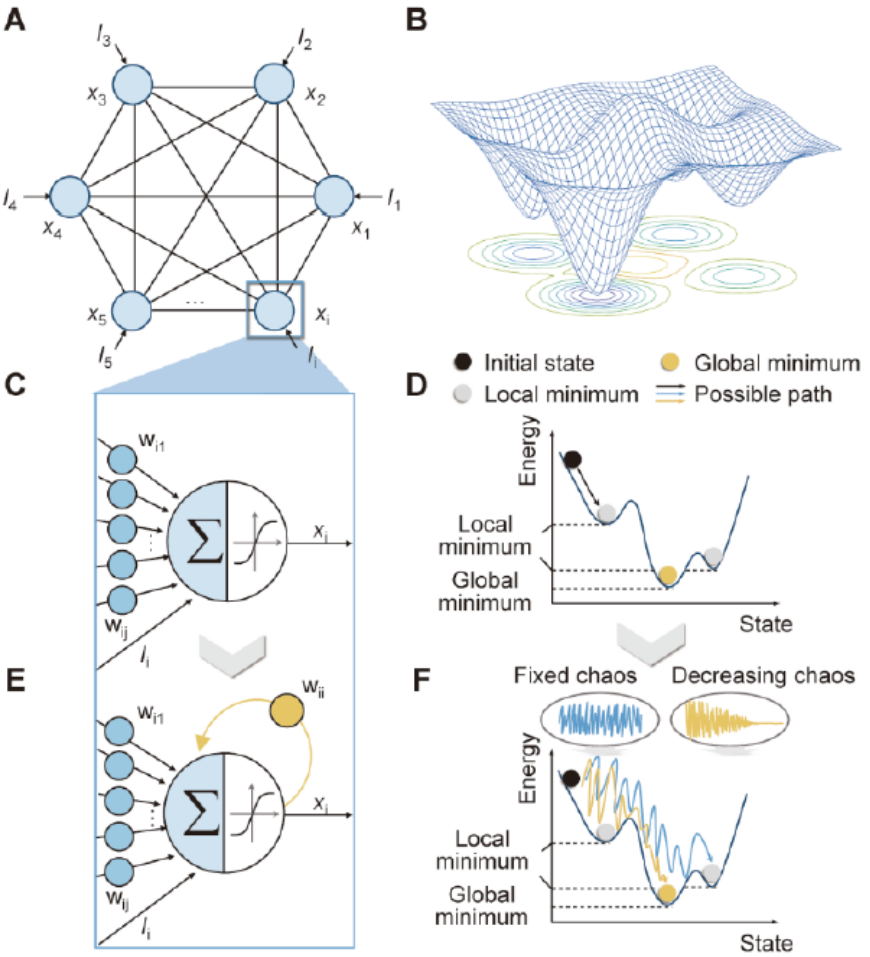


图 13：暂态混沌与 Hopfield 网络 [7]

混沌是非线性的一个特例或者形式，在复杂优化求解当中有很重要的作用，比较有代表性的模拟退火算法就是一种传统优化问题的求解算法。Hopfield 网络中每个神经元和其他神经元形成对称连接，随着网络演化能量逐渐减小，神经元达到稳定之后，它会最终收敛在极小值或者离散吸引子，从而可以实现联想记忆等诸多应用。比如，即使缺失信息，它从任意初始值开始最终都会收敛在能量上的极小值点，实现很多记忆方面的功能。但是如果用于优化问题求解可能会收敛在局部最小值，而没有达到全局最优值。为了解决这个问题，可以在传统的神经网络基础上引入混沌，由于混沌具备局部遍历性可以帮助跳出局部最小值，从而有机会达到全局最优值。但是，如果混沌一直存在有可能使网络难以收敛，影响网络稳定性。最理想是随着时间逐渐最终消失的暂态混沌，在初始阶段利用局部遍历性对解空间进行寻优，在最后阶段能够从混沌状态退出达到收敛状态。

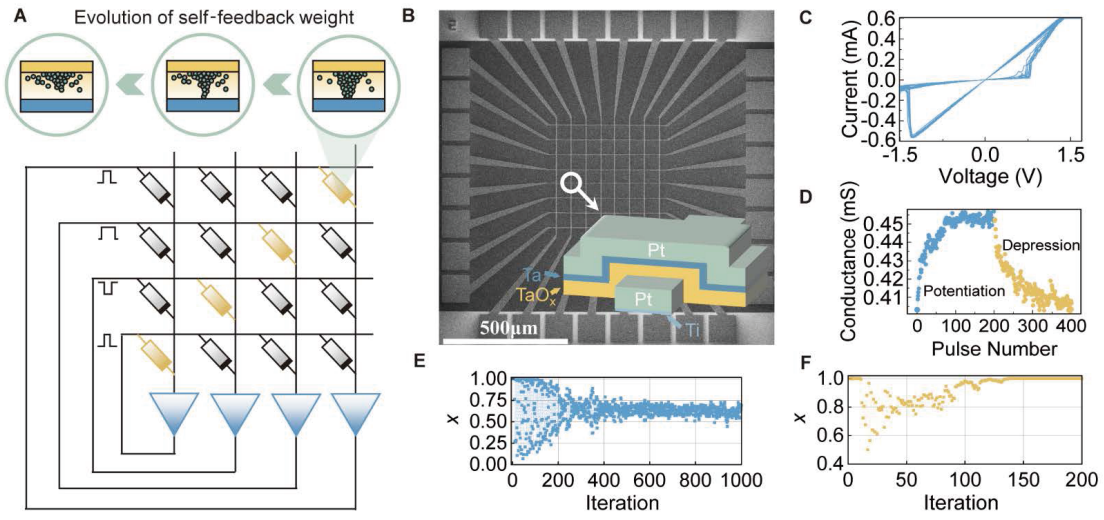


图 14：暂态混沌 Hopfield 网络的忆阻器阵列实现 [7]

杨玉超团队将暂态混沌神经网络在单个忆阻器交叉阵列上面进行硬件实现。传统的 Hopfield 网络因为神经元和神经元自身是没有连接的，对角线上所有的权值都是零，每个器件都是高阻态。但是为了引入混沌，需要给对角线上的忆阻器赋予一个权值，并且对角线上权值随时间逐步减少，从而实现暂态混沌。即对器件的数据进行 reset，从高电导状态逐渐过渡到低电导状态，最终网络达到稳定收敛状态。实验证明，神经元采用逐步 reset 调控网络动力学方法之后，随着时间的推移神经元从混沌状态退出达到收敛的状态。除此之外，为了能够寻找到最优解，神经元还要能够追踪更低能量。为了追踪最低能量对 Hopfield 网络施加外界偏压时 Hopfield 神经元应尽可能输出最高值，即神经元输出变强。实验表明，当施加偏压时神经元仍然会从混沌当中退出，达到一个收敛值，但是这个收敛值变得更大。这样的神经元既具有了暂态混沌，同时能够追踪最小能量值，就可以用神经网络求解优化问题。

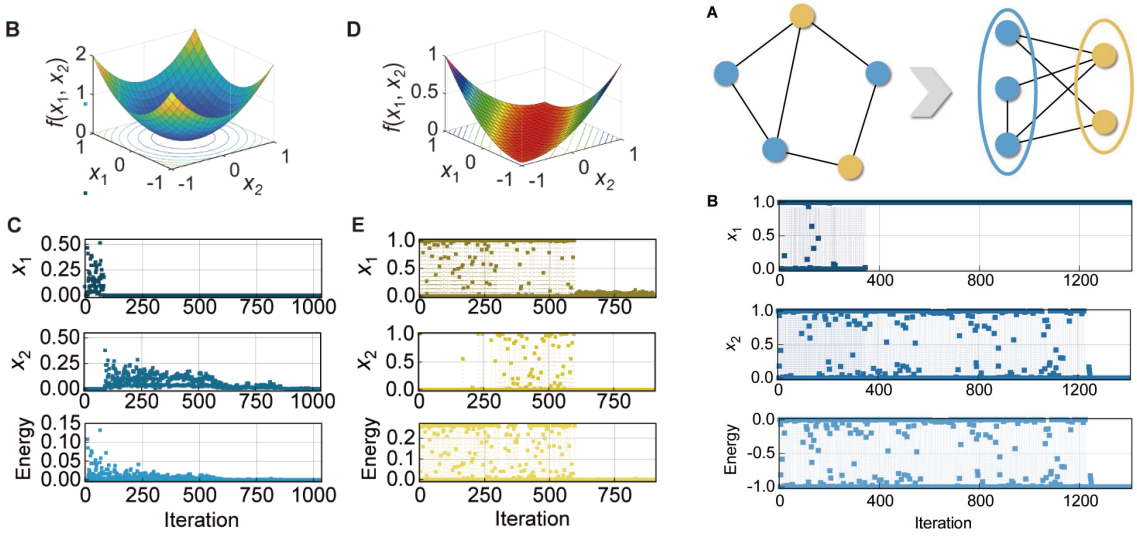


图 15：函数优化问题与组合优化问题 [7]

第一类优化问题是函数优化问题，函数优化问题简而言之是寻找函数最小值的问题。一个比较经典的函数是二维球函数，它的最小值是在 X1 和 X2 都等于 0 的位置。通过实验，随着暂态混沌神经网络逐渐演化，从混沌状态退出到收敛状态，能量逐渐减小，最终 X1 和 X2 都逐渐收敛到 0 的位置，找到最小值。第二类优化问题是组合优化问题，比如最大分割问题、旅行商问题等，在电路设计等很多场景有重要应用。如何找一个分割方法使它切出的边最多？旅行商第一站到哪个城市，第二站到哪个城市？类似地也可以把问题映射到硬件上，然后利用网络的演化最终找到问题的最优解。

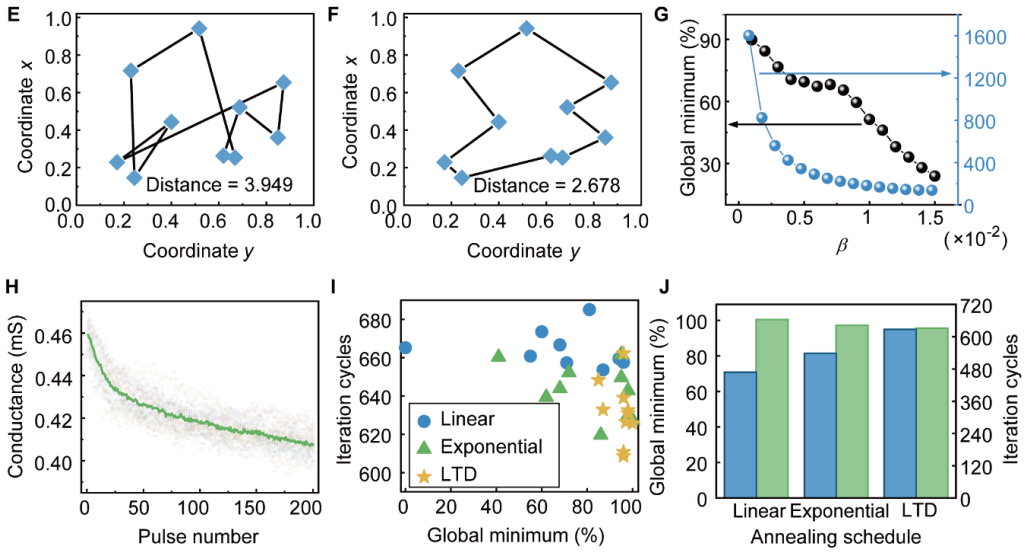


图 16：基于忆阻器本征非线性的高效退火 [7]

所以不管求解函数优化还是组合优化，其核心就是把问题映射成为 Hopfield 能量。比如，函数优化就是把求解函数的最小值变成求解 Hopfield 能量的最小值，这种情况下实现硬件的求解方法。但是抛开具体问题形式，背后的共性问题是采用什么样的动力学过程，来让网络从混沌退出达到收敛状态，也就是采用哪种具体的退火策略。原则来讲任何数学形式都可以实现退火，比如线性退火、指数退火等等。但是，忆阻器自身提供的非线性 reset 过程提供了一种非常理想的退火策略。如图 16，左下图是施加完全相同的脉冲信号，使器件从高电导回到低电导状态所经历的非线性 Reset 过程。在优化问题求解背景下这种非线性退火跟线性退火、指数退火相比得到最优解概率最高，付出的代价、所需要的时间或操作步骤最少。优化问题求解效率最重要的两个方面就是获得最优解的概率与时间上的效率。所以，器件自身的非线性过程为利用暂态混沌神经网络求解优化问题是提供了非常理想的退火策略。

五、小结

本文首先阐述具有丰富动力学特性的人工神经元可以作为类脑计算的元器件，而后基于忆阻器动力学特性构建全忆阻神经网络并且进行了新的计算方面的应用，最后利用暂态混沌现象构建基于忆阻器的优化问题求解器。

参考文献

[1] Chua, Leon. (2011). Local activity is the origin of complexity. International Journal of Bifurcation and Chaos. 15. 10.1142/S0218127405014337.

[2] Fuller, Elliot & Keene, Scott & Melianas, Armantas & Wang, Zhongrui & Agarwal, Sapan & Li, Yiyang & Tuchman, Yaakov & James, Conrad & Marinella, Matthew & Yang, Jianhua Joshua & Salleo, Alberto & Talin, A.. (2019). Parallel programming of an ionic floating–gate memory array for scalable neuromorphic computing. Science. 364. eaaw5581. 10.1126/science.aaw5581.

[3] Wang Z, Joshi S, Savel'ev SE, et al. Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing. Nat Mater. 2017;16(1):101–108. doi:10.1038/nmat4756.

[4] Yi, Wei et al. "Biological plausibility and stochasticity in scalable VO2 active memristor neurons." Nature communications vol. 9,1 4661. 7 Nov. 2018, doi:10.1038/s41467–018–07052–w.

[5] Kumar, S., Rajkumar, V., Kyle, R. et al. Multiple myeloma. Nat Rev Dis Primers 3, 17046 (2017). https://doi.org/10.1038/nrdp.2017.46.

[6] Duan, Q., Jing, Z., Zou, X. et al. Spiking neurons with spatiotemporal dynamics and gain modulation for monolithically integrated memristive neural networks. Nat Commun 11, 3399 (2020). https://doi.org/10.1038/s41467–020–17215–3.

[7] Yang, K., Duan, Q., Wang, Y. et al. Transiently chaotic simulated annealing based on intrinsic nonlinearity of memristors for efficient solution of optimization problems. Sci Adv 6, eaba9901 (2020).

北京大学教授罗国杰：真存内计算的自动综合方法

整理：智源社区 黄善清

虽然人们都说摩尔定律已经失效，然而现实中集成电路的复杂度却还在不断提高，这也带来了一个核心问题——数据移动，于是人们提出了“存内计算”想法，希望能一劳永逸地解决这个问题。

“但我们回看过往的工作，就会发现每个忆阻器单元虽然都用于进行算和存，但存数据部分的 AreaUtilization 都很低，都在 10% 以下。换句话说，这些工作不能算是存内计算，它们只是利用了忆阻器的计算功能，但作为阵列的话并非用于存数据。”在第二届北京智源大会上，北京大学罗国杰教授在演讲中如此说道。

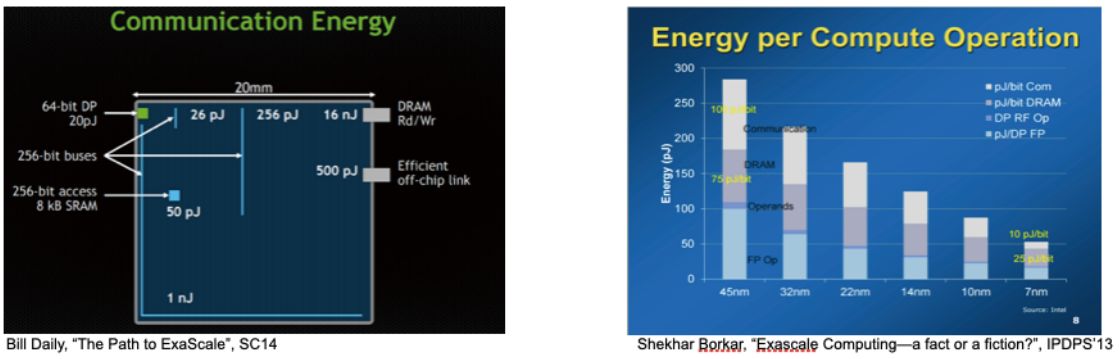
在《真存内计算的自动综合方法》的报告中，针对阵列利用率低的问题，罗国杰也提出了自己的解决方案： 设计更稠密的数据、设计更高效的 EDA 算法、针对 RRAM 计算的工艺优化，其最终目的是找到一个设计自动化的最佳路径，为后世沉淀一套类似“pytorch”的 EDA 通用框架工具。

罗国杰强调，虽然业内如今都说摩尔定律已经失效，但依然不能忽略集成电路的复杂度还在不停提高——无论是 2.5D 还是 EMIB 的集成。以 N100 为例，它所使用的 HBM2 带宽内存，不但是一个高集成度的内存，同时还与芯片以 2.5D 或类似 2.5D 技术集成在了一起。总的来说，我们看见芯片几何缩放等受到影响，但它的集成度其实还在不断提高。

下面，请大家阅读罗国杰报告的详细介绍。

一、数据移动损耗问题

罗国杰指出，随着集成度的提高，数据移动就成了一个问题。



Bill Daily, “The Path to ExaScale”, SC14

Shekhar Borkar, “Exascale Computing—a fact or a fiction?”, IPDPS'13

图 1： 数据移动损耗情况

他随后展示了两张图，图中可以看到数据在不同地方损耗的能量 / 时间分别属于什么级别，左上角绿色部分是 64 位双精度浮点所需要的能量，而数据移动距离则体现在中间的细蓝色指标，访问则为蓝框——如果在芯片上移动，对应消耗的能量将在这体现。

罗国杰总结道，光是计算就花了 20PJ，但模数据本身其实就已经属于这个量级，要是再考虑片外挪到片上，差的可能就不仅仅一个量级了。

二、现有解决方案

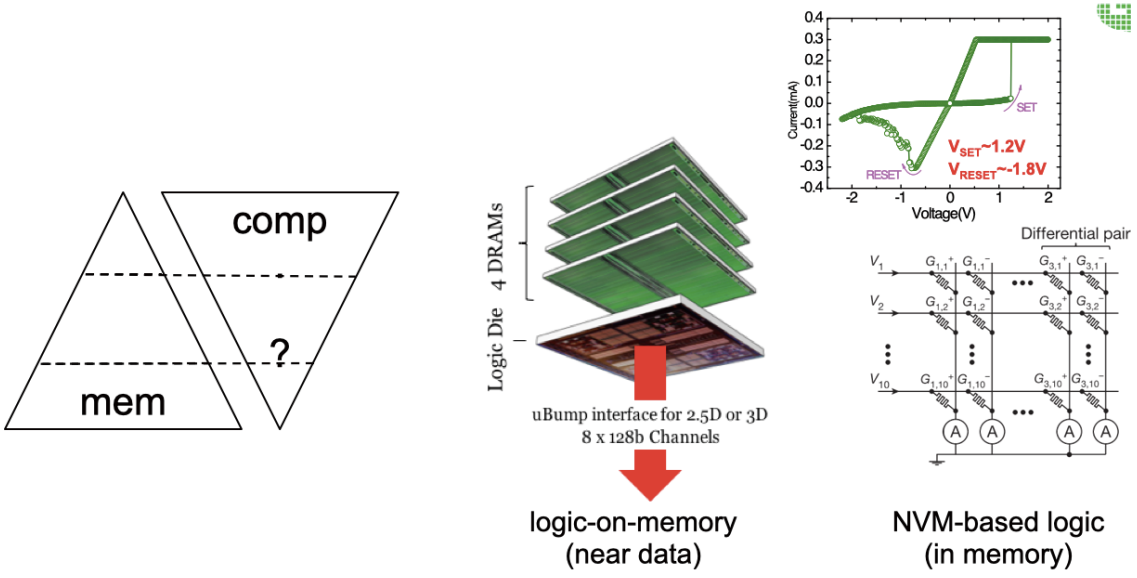


图 2：Hierarchy 解决方案

罗国杰随后分享了现有的两种解决方案，一种是 Hierarchy，一种是 RRAM。

罗国杰认为对传统做体系结构研究的老师来说，现有的解决方法就是 Hierarchy，而新趋势是不仅会看 Memory 的 Hierarchy，还会综合考虑计算在 Hierarchy 中扮演的角色——在 FreeD 的内存上给一个逻辑处理，FreeD 由几层 Memory 及一层 Logic 组成，而我们可以选择在 Logic 层面动些手脚。

至于 RRAM 或更广泛的 MEM 其实都具有共同的特性，就是不仅可以用于存储，还可以在上头添加不同的电阻电压以实现一定的计算。

罗国杰也强调，之前提到的方案都不算新想法，最早提到类似方案的文章发表于 1970 年，虽然当时的工艺较落后，但想法已经存在——假设有一堆存储芯片，只要在里面加处理单元的 Memory，就可以实现某种程度的存内计算。

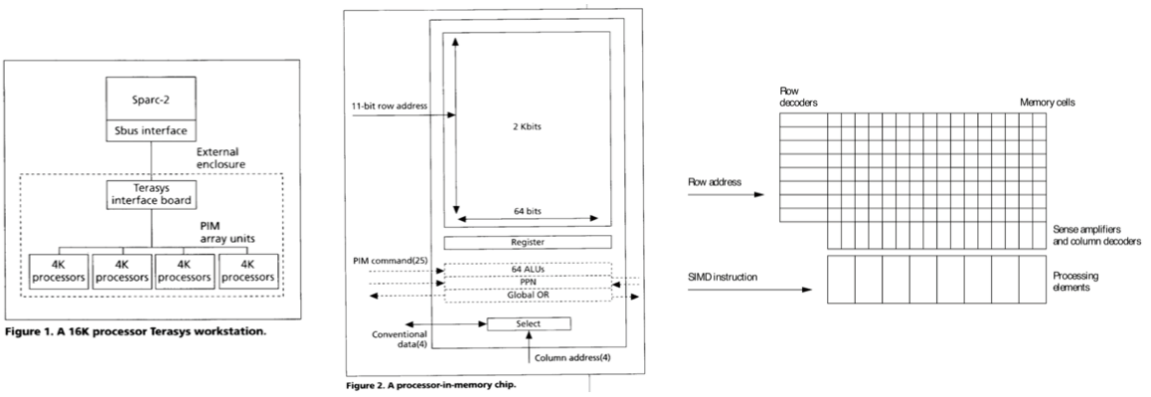


图 3：最早提到存内计算的文章发表于 1970 年

- 1970 - Stanford Research Institute - Logic in Memory (LiM)
- 1990 - Toronto - Computational RAM (C-RAM)
- 1992 - Berkeley - Intelligent RAM (IRAM)
- 1995 - Notre Dame - Processing in Memory (PIM)
- 1995 - Supercomputer Research Center - Processing in Memory (PIM)
- 1996 - KMU - Processing in Storage
- 1998 - UC Davis - Reconfigurable Architecture DRAM (RADram) & Active Pages
- 1999 - UIUC - FlexRAM
- 2000 - Stanford - Smart Memories

图 4：2000 年以前存内计算思想的演变过程

有趣的地方在于，这篇发表于 70 年的论文，其核心思想一直到 90 年代才重新被伯克利大学、UIUC 等大学捡起，而 70 年代到 90 年代这段时间正好是摩尔定律最兴旺的时期。总的来说，存内计算是一个自然而然的延伸概念。

三、数字计算的设计自动化问题

- MAGIC NOR implementation $Z = \mathrm{NOR}(X,\ Y)$
  - $V_G > 2 \cdot V_{RESET}$
- Parallel execution over rows and columns
  - WL parallelism: $R_{im} = \mathrm{NOR}(R_{i1}, R_{i2})(i \in [1,m])$
  - BL parallelism: $R_{mi} = \mathrm{NOR}(R_{1i}, R_{2i})(i \in [1,m])$

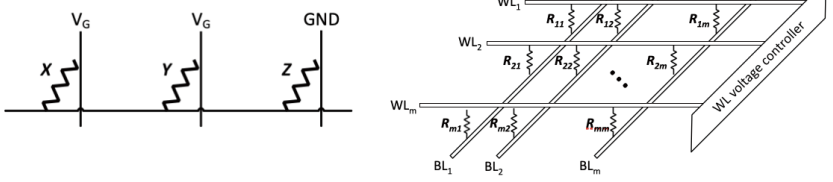


图 5：RRAM Array

罗国杰接着举了一个存内计算的例子 —— 用 RRAM 做逻辑。如果是用阵列来存数据，一般大家会自然而然地用数字来进行计算，然后算完存在该阵列里，但通过数字计算，将会面临设计自动化的问题。

他进一步解释道，忆阻器要是在几个点加上不同电压，最后 Z 的组织就会受 X 和 Y 两个忆阻器的影响，进而实现一定的逻辑运算。与此同时，要是有个阵列可以对每个 WL 同时加上电压，或对每个 BL 加上同样的电压，同样也能实现一定的并行度，也就是大家所熟悉的“布尔模式”。

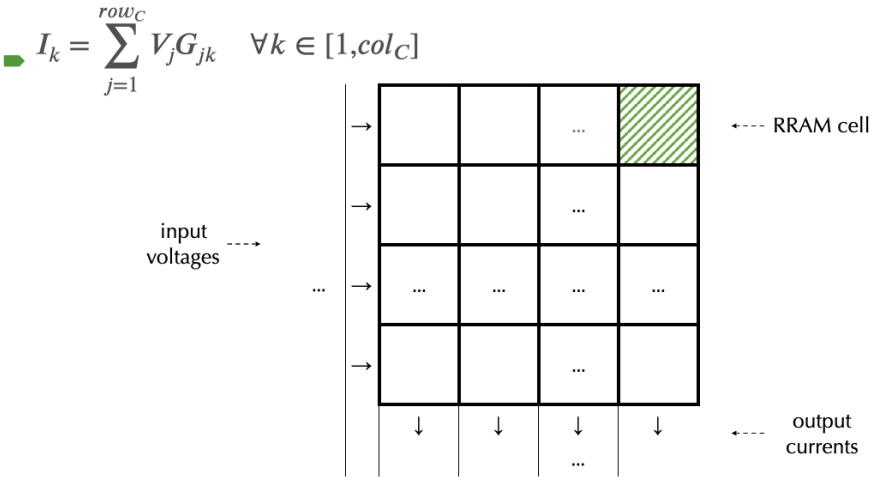


图 6：Analog 模式

他还提到了 Analog 模式，除了最开始被大家用来做内积，他表示其实还有很多可以进行计算的东西，比如加一个非线性，就可以做类脑相关的计算。

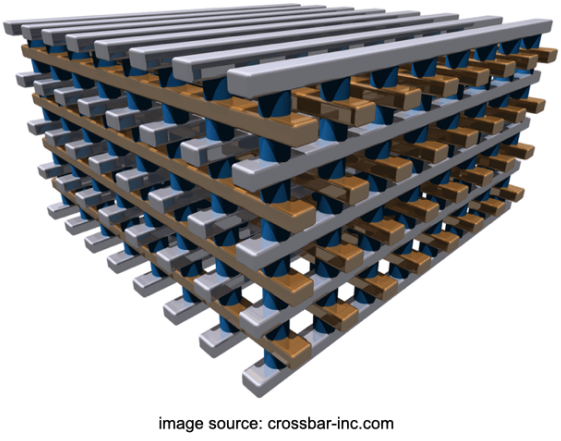


图 7：Memory 模式

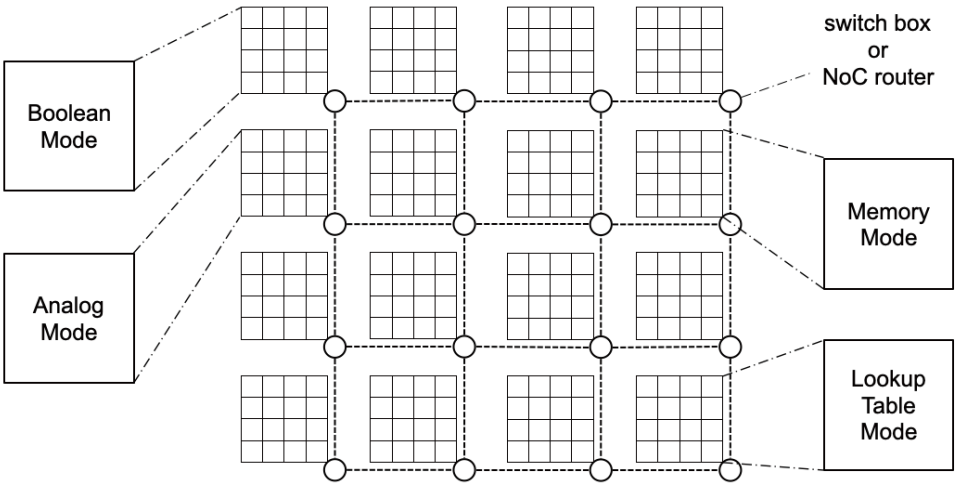
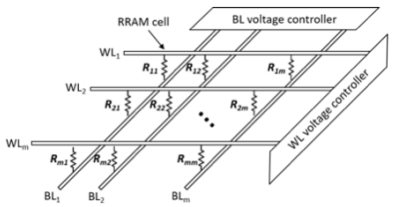


图 8：RRAM 可进行配置的结构

除了本身可以作为存储，罗国杰团队更感兴趣的是它的结构——每块都可以被配置，块与块之间都采取类似 Switchbox 或者 Nocrouter 结构。

四、设计自动化方向的解决方案

- Functionally complete
- SIMD-like computation



Work	Stateful logic operations
[1]	IMP
[2]	NOR, NOT
[3]	NAND, NIMP
[4]	NOR, NAND, Min, OR
[5]	NOR, NOT, NAND, NIMP, XOR

[1] Borghetti, J., Snider, G. S., Kuekes, P. J., Yang, J. J., Stewart, D. R., & Williams, R. S. (2010). "Memristive" switches enable "stateful" logic operations via material implication. *Nature*.
[2] Kvatinsky, S., Member, S., Belousov, D., Liman, S., Satat, G., Member, S., … Weiser, U. C. (2014). MAGIC — Memristor-Aided Logic. *TCAS-II*.
[3] Huang, P., Kang, J., Zhao, Y., Chen, S., Han, R., Zhou, Z., … Liu, X. (2016). Reconfigurable Nonvolatile Logic Operations in Resistance Switching Crossbar Array for Large-Scale Circuits. *Advanced Materials*.
[4] Avati, V., Eggert, K., & Taylor, C. (2018). FELIX: Fast and Energy-Efficient Logic in Memory. In *ICCAD*.
[5] Xu, L., Bao, L., Zhang, T., Yang, K., Cai, Y., Yang, Y., & Huang, R. (2018). Nonvolatile memristor as a new platform for non-von Neumann computing. In *ICSICT*.

图 9：过往的逻辑运算工作

罗国杰表示，关于阵列里的具体实现，如果只是处理数字部分，已经有相当多工作做了任务未及的运算，我们可以很容易就抽出一个功能完备的逻辑运算。此外，阵列有一定并行度，类似 SIMD，要是我们把阵列存在数据里，就可以对各个行或者各个列同时做运算。

Benchmark	Work	Area utilization (A) [1]	Computation area		
			External inputs (E)	Intermediate variables (V)	Unused cells (U)
LGSynth'91	SIMPLE [1]	4.16%	4.15% [2]	37.85%	53.84%
ISCAS'85	Logic [2]	1.45%	1.44%	22.54%	74.57%
	Scalable [3]	0.99%	0.98%	15.35%	82.68%
	Look-ahead [4]	0.94%	0.94%	14.65%	83.47%
	Staircase [5]	0.54%	0.54%	8.42%	90.5%
	SAID [6]	2.95%	2.94%	46.02%	48.09%
Convolution	IMAGING [7]	8.82%	3.91%	86.88%	0.39%

图 10：过往工作的 AreaUtilization 偏低

虽然都被称作“存内计算”，但是回看过往的工作，就会发现每个忆阻器单元虽然都用于进行算和存，但存数据部分的 AreaUtilization 都很低，几乎都在 10% 以下。罗国杰认为这些工作都不能算是存内计算，而只是利用了忆阻器的计算功能，但作为阵列看待的话，并非用来存数据。

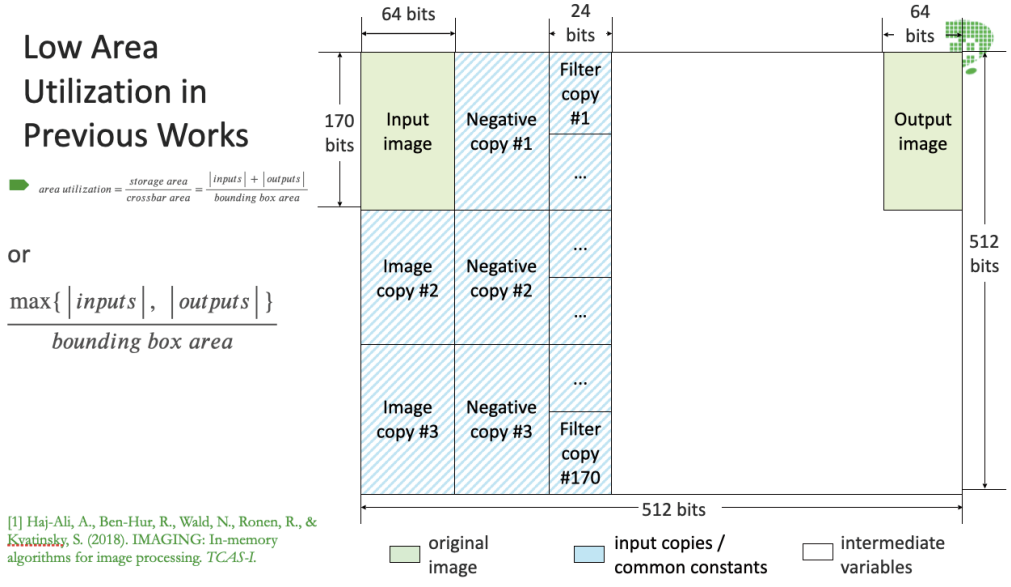


图 11：输入图像与输出图像中间耗损很大一部分面积

罗国杰举了一个做 Image 卷积的例子，由于输入图像与输出图像中间耗损很大一部分面积，一除就会发现这个面积的利用率不是特别高。

罗国杰总结道，这些工作只能说是利用了忆阻器做计算，而不能说是做了存内计算。

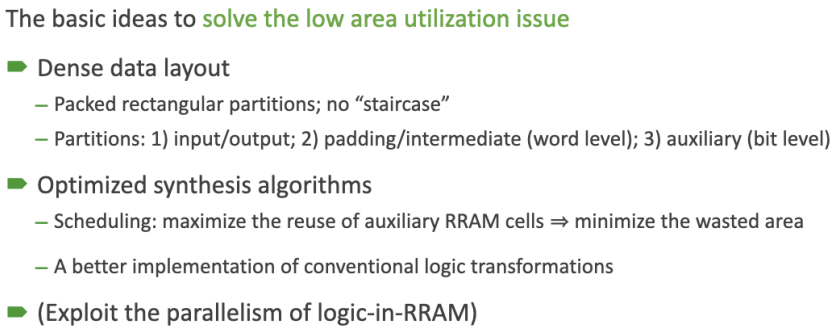


图 12：低利用率的 3 点解决方案

那要怎么解决低利用率问题？罗国杰总结了 3 点：

首先是设计更稠密的数据。之前有些工作为了算得更快，会将忆阻器作阶梯状使用，但这样会导致 Array 的降低，因为利用率很低。只需把数据分为以下几类：一类是输入输出；一类是 Word Level 的中间结果；一类是 Bit Level 的中间结果，然后将每个结果都限制在一个紧凑的区域里。

另一个思路跟 EDA 相关——设计一个高效的综合算法，以确保计算面积的最大化利用，这与编译、EDL 的 Sketch 有一定的关联。

另外，还可以对传统逻辑进行优化，尤其针对 RRAM 计算做工艺相关的优化。

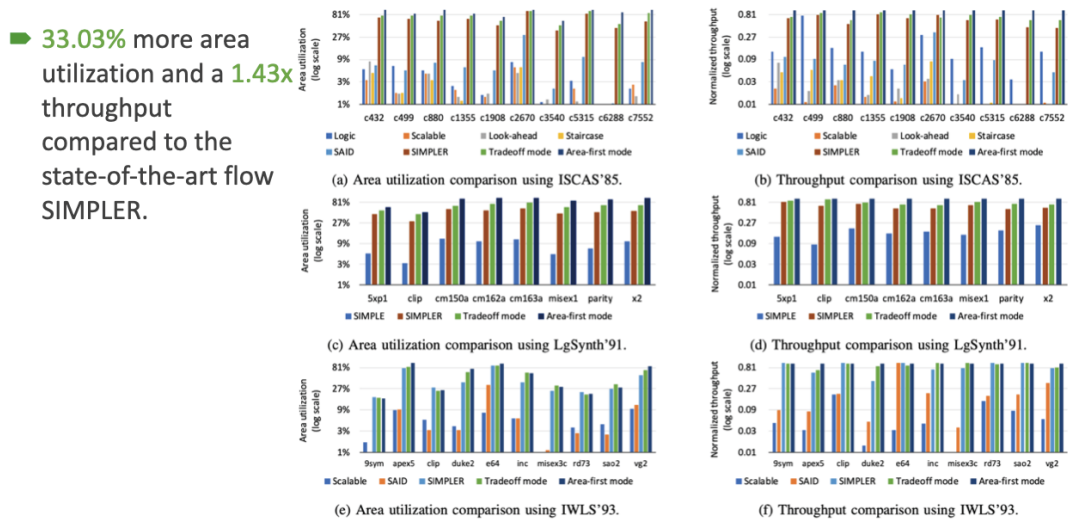


图 13：合成流的评估

最右边两列呈现的是我们的结果，右边数起第三列是当前最好的结果。而更早期的，使用的 utilization 都不到 10%。

罗国杰的技术方案与当下最好的工作相比，可以省下 33% 的 area utilization，而且输出还有所提高，他强调，这个工作在进行时 SIMPLER 还没发表，虽然解决的是同一类问题，但在结果上他们要好出不少。

- Save 40% cells
- Save 77% erasing operations

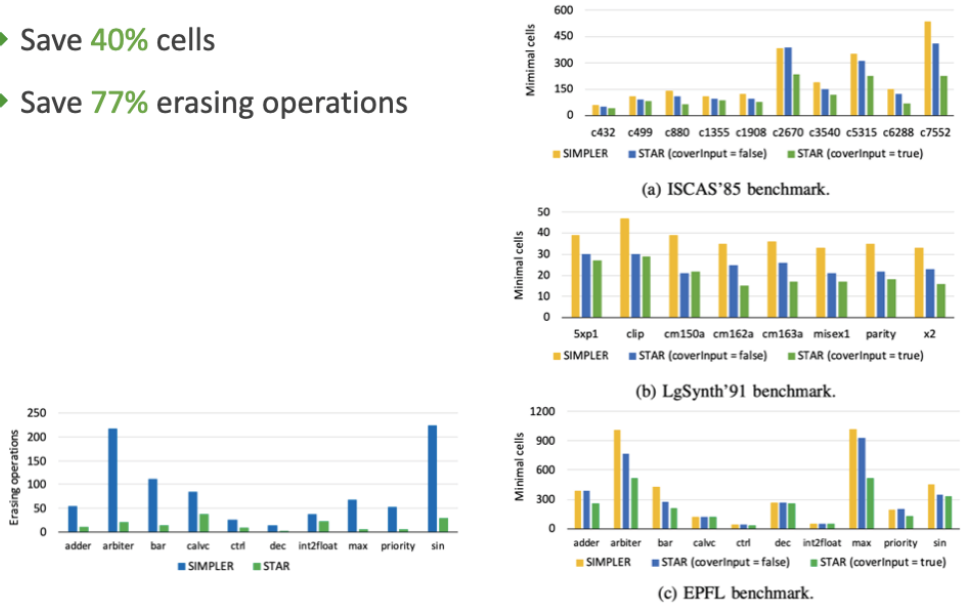
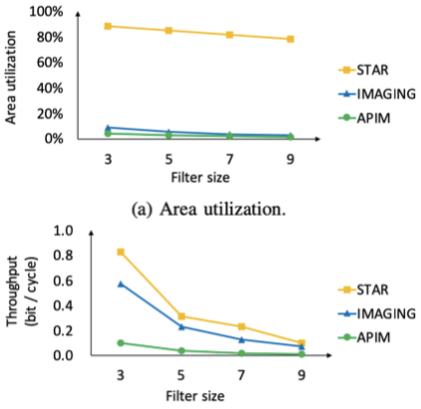


图 14：Scheduling 算法工作

罗国杰还提到一个 Scheduling 细节，使用了类似计算器分配的算法——总体问题有点类似，但具体细节差得比较远，但问题的难度是一样的，都是 NP 难问题，使用 Scheduling 将省下 40% 的成本。

- Achieve 78% more area utilization and a 1.48x throughput



Work		IMAGING	STAR
Image size	row_{max}	170	510
	col_{max}	8	56
Area utilization		8.10%	88.16%
Data transfer / pixel (bit)	duplication	24	0
	filter	4.24	0
	padding	1.06	0.31
Execution cycles / pixel		11.03	9.63
Throughput (bit / cycle)		0.57	0.83

图 15：Image Convolution 工作

与之前一个做图像卷积 imageConvolution 的工作相比，同样都是使用忆阻器，但它们的阵列存数据的比例只有 8%，其余 92% 都用于做计算或压根就没派上用场； 而罗国杰团队的方法能确保阵列的 88% 用来存数据，至于剩下的 12% 则用来做中间结果。

罗国杰教授表示，除了单个阵列，阵列 array 也有很多问题需要被解决。他表示过往工作都只看数字计算的模

式，而当今主流更强调看用模拟或非线性特性做计算和存储。

当然，自动综合只能支持 Toy Problem，如果面对的大型 Case，还是只能依靠手工设计。这里头有很多需要考虑的东西，比如 APPlicationn-level，是要放在阵列还是 CPU 里计算，比如定性的放在 RRAM，并行度更高的可以分布在 RRAM 上。

罗国杰教授表示，大家可以根据不同工艺的实现做能量上的平衡。

除了 APPcationlevel，还存在一个 kernel-level，李国杰教授的团队在尝试用 DSL 去描述不同形式 RRAM 的计算模式，他认为当中有几个可以通过工具进行的事情：逻辑和算数的优化、阵列里的映射、阵列之间的数据移动、多个阵列的映射等等。

Kernel 可以通过不同的模式进行操作，如果是数字，可以采用布尔模式；如果是类脑，就要与其他老师合作加入类脑部分的内容。由于各种计算都有模板，只要确定计算模式，就能通过流程来完成调度布局和通信。

五、openBELT 计划

刚刚讲的都是针对 RRAM 的一个阵列所做的映射，要是阵列组成网络，就会存在更多需要考虑的事情。

罗国杰团队提了一个 openBELT 计划，希望在做完这些实践工作后，最终能够给大家提供基础组件，让大家搭建属于自己的 EDA 工具。

他们做了一个利用强化学习决定逻辑优化序列的工具，我们都知道一个逻辑网络其实可以做很多优化工作，比如把深度变得均衡，还有实现从左到右的逻辑等价等。而所谓的逻辑综合，既是不停用逻辑优化去处理逻辑网络，最后得出更优的逻辑网表。

如果用过逻辑优化专家，就会发现 ABC 里有专家的建议，比如 Resyn1 由 8 条命令组成，Resyn2 也由 8 条命令组成。通过强化学习，可以将这两个命令按顺序执行，进而生成比这两条命令质量更好的阵列。与不停运行命令的模式相比，这个方式只需耗费 50% 的命令数目，此外面积和深度也会减少。

虽然是罗国杰是 EDA 背景出身，但他过去却从来未做过逻辑综合相关的事情，但这个工具也能达到跟逻辑综合命令专家建议一样的效果。

简单来说，罗国杰希望做成一个 EDA 界的“Pytorch”，他现场通过一个 Demo 展示了如何利用这套工具做逻辑综合、布局布线。他希望该工具未来能帮助大家一目了然看到逻辑综合背后的做法，然后大家可以根据自己的需求生成合适的架构或工具。

他们会将 EDA 分为几个部分，最上面的接口留给芯片设计者、架构设计者或者工艺研究者，而工具里会储备一些经典的 EA 算法和求解器，只要通过简单的接口，就能搭出属于自己的 EDA 流程。

罗国杰最终强调，他们不仅仅是把这个流程做出来，还希望通过这个流程去驱动一些基础的 Operator。

中科院包云岗：开源 RISC-V 处理器核的敏捷设计实践

整理：智源社区 黄善清

中科院计算所研究员、智源学者包云岗本次的演讲主题是《开源 RISC-V 处理器核 COOSCA 的敏捷设计实践》。

包云岗，中科院计算所研究员，所长助理，先进计算机系统研究中心主任，中国科学院大学岗位教授，博士生导师，中国开放指令生态（RISC-V）联盟秘书长，于 2019 年入选智源学者。

在演讲中，包云岗首先追溯了开源硬件的三股驱动力，归纳了构建开源芯片的四个核心要素，并着重介绍了其中的两个要素：开放指令集（以 RISC-V 为代表）和软硬件协同（以包云岗团队研发的低成本 RISC-V 全系统原型验证平台等为案例）。

下面，是包云岗演讲的精彩要点介绍。

一、开源硬件的三个驱动力

开源软件已经给互联网行业带来了巨大的改变，当前，开源硬件也逐渐面临着强烈的需求。包云岗指出，推动开源硬件发展，有三股驱动力。

1.1 第一个股驱动力，AIoT 的需求碎片化问题。

智能物联网（AIoT）时代到来，处理器芯片规模将达到千亿颗以上。这些芯片应用到不同领域、不同场景，将会有很多碎片化的需求，于是便需要做定制化处理器。相比于以前的模式，一个 Intel/ARM 处理器应对所有应用问题，定制化处理器对设计方法、速度、成本的要求都提升不少。

作为对比，在 20 年前，互联网需求也是碎片化，但开源软件降低了 APP 开发门槛，Linux、Apache 等开源组件已经能够帮助我们解决 90% 的代码复用，用户做一个 APP，很多时候写的代码在解决方案里甚至不到 10%，因此他可以专注于针对需求做定制，快速设计和开发应用。门槛的降低，也吸引和培养了大批互联网人才，才使中国互联网产业具备国际竞争力。因此，我们可以学习和借鉴开源软件的经验。

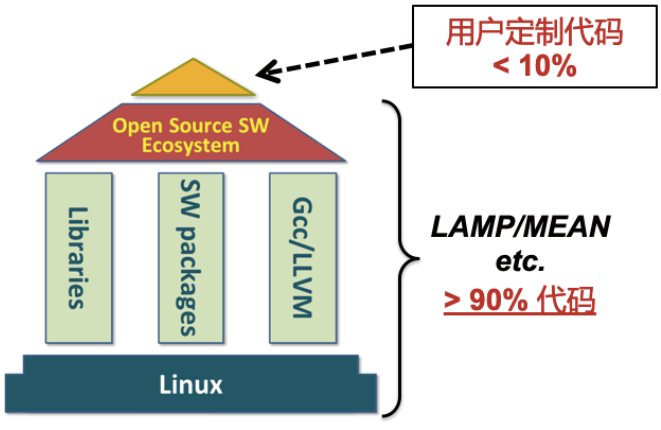
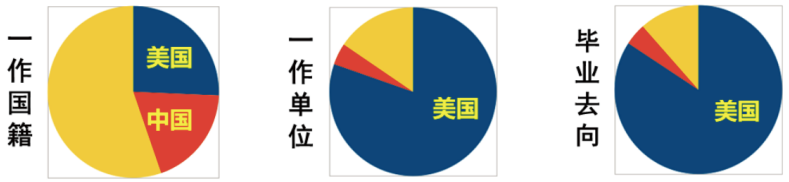


图 1：开源组件可以帮助我们解决 90% 的代码复用

1.2　第二个驱动力，来自于芯片设计人才的欠缺。

包云岗统计了体系结构顶会 ISCA 过去十年的论文第一作者，其中国籍为中国的第一作者大概占全部的 1/5，与美国差距不是特别大；但如果看研究单位和毕业去向，就会发现几乎全部（85%）在美国工作，只有 4% 在中国。



体系结构顶会（ISCA）十年论文统计情况

图 2：体系结构顶会（ISCA）十年论文统计情况

这种局面事实上与芯片设计门槛过高有关。芯片开发过程分为设计 – 制造 – 封装，即使单把“设计”这一环节就要求懂设计需求分析、体系结构设计、逻辑设计、逻辑验证等，还需要有后端经验。但，在我国高校中几乎没有这样的机会让学生去训练和实践这些事情，所以也就培养不出芯片人才，这是我们面临的困境。

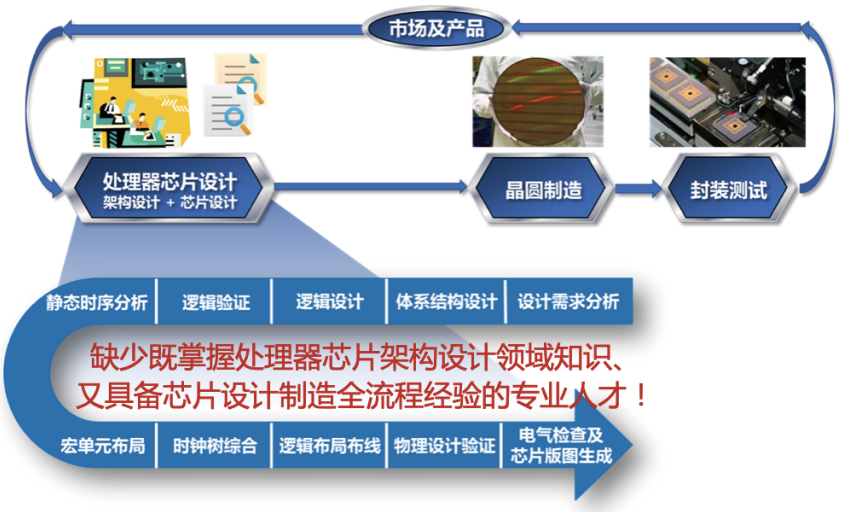


图 3：我国的芯片人才培养困境

1.3 第三个驱动力，降低芯片设计门槛，推动产业变革。

美国在七十年代末、八十年代初，也面临芯片设计门槛过高、人才稀缺的危机，1982 年全美做半导体研究的教授和学生加起来不到 100 人。

1981 年美国政府启动一个叫 MOSIS 的项目，该项目提出 MPW 模式从而让芯片设计成本降低了几十倍，大幅降低了当时芯片设计的门槛，从而使学校也可以参与到芯片设计中。据了解该项目已经为美国带来了 6 万多款芯片，并培养了数万名学生。MOSIS 项目对人才的培养也催生了新的商业模式，比如先后出现了台积电、英伟达、高通这些专注于设计和生成芯片的公司。

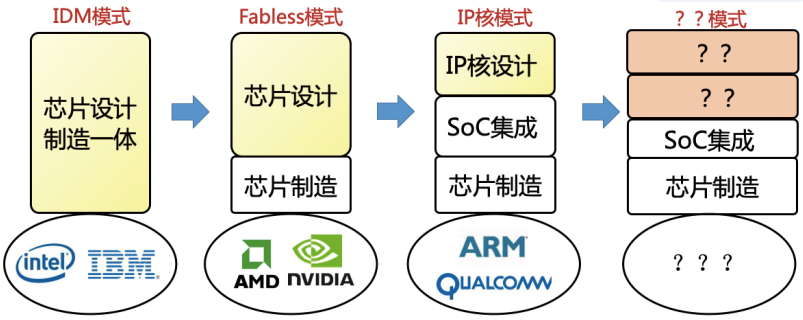


图 4：美国 MOSIS 项目

事实上，我们回顾处理器设计方法的变革历程，会发现芯片设计是一个不断解耦的过程。每一次变革都伴随着设计敏捷度的大幅提升，并孕育出新的世界性领军企业。

到了今天，芯片设计门槛再次变高，这表现在两个方面： 时间长、成本高。例如英伟达的 Xavier SoC 设计用了 8000 人年； 终端芯片 14nm 工艺为例，需要上亿元研发经费。这使得只有少数企业才能承受中高端芯片的研发成本，大学根本没有能力开展芯片研究。这制约了芯片领域的创新，也导致芯片人才培养能力严重不足。

结合以上三个驱动力，开源硬件事实上已经势在必行。但是当下，芯片领域做开源并非容易，因为存在开源死循环： 因为开发投入很大，所以开发之后不愿开源，导致社群没有开源可用，于是大家只能花高价去买 IP，买回来以后为了减少出现打水漂的情况，会花很多时间去验证，这又增大投入，导致更不愿意开源。

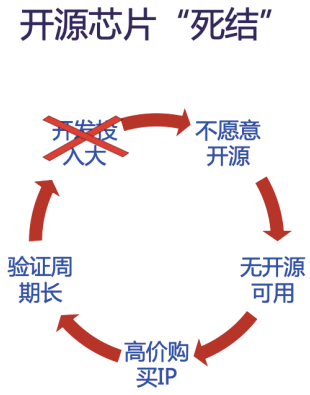


图 5：开源芯片“死结”

包云岗指出，打破这种循环，唯有从降低开发成本入手，他把降低开发芯片门槛分为几个步骤：

- 让学生不再害怕做芯片；
- 让本科生可以带着自己设计的芯片毕业；
- 让 3–5 人的团队可以创办芯片创业公司；
- 让做芯片像写 APP 那么简单；
- 让天下没有难做的芯片；
- ……

包云岗提到有四个要素可以帮助去构建开源芯片的生态，从而降低开发投入环节的人力、EDA、IP 成本：

第一，**开放的指令集。**RISC-V 带给我们这个机会，有了指令集可以做开源 IP 甚至开源 SOC；
第二，**开源的 EDA 工具链。**这能够帮助我们不像现在这样被卡脖子；
第三，**仿真和验证平台。**在芯片设计领域有很大的成本开销，就是仿真和验证，这也要更好思路去降低；
第四，**软硬协同。**芯片做出来要有软件支持，所以系统、编译器要有很强大的支持，构建软硬件生态结合起来，实现软硬件协同生态。

设想，当你想做一个芯片，90% 的模块功能都是现成的，可以拿来复用； 你只需要专注于自己定制的特性即可。这将对芯片设计行业产生颠覆性影响。

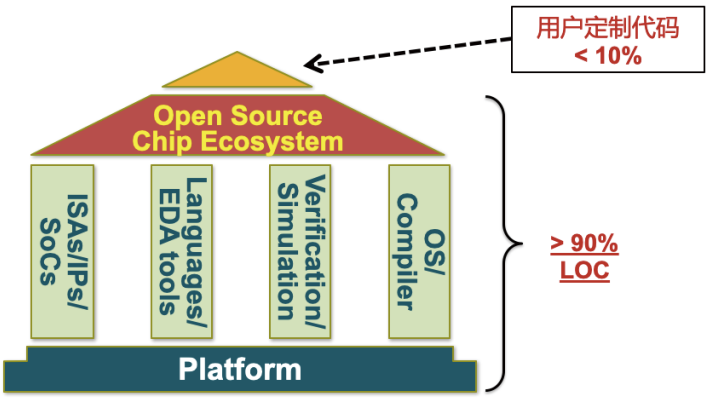


图 6：构建开源芯片生态的 4 个要素

接下来，包云岗针对第一和第四两个要素做了详细介绍。

二、开源指令集

2010 年，UC Berkeley 的研究团队想要设计一款 CPU，他们在为该项目选架构的时候，对比了当时已有的 ARM、MIPS、SPARC 和 X86，发现这些指令集不仅越来越复杂，还有很多 IP 法律问题，再加上 X86 授权难以获取，ARM 授权价格昂贵，于是他们决定设计一套全新的指令集。于是成立了一个四人小组，用 3 个月的时间完成了 RISC-V 指令集的开发。

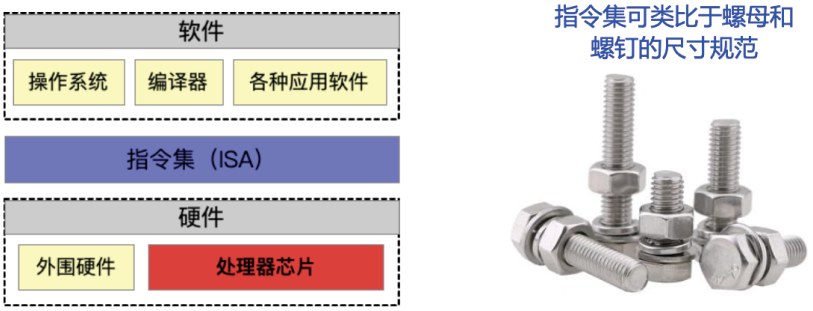


图 7：RISC-V 指令集

RISC–V 本身是一种规范。所谓的规范，就像螺母和螺钉的尺寸，定义宽度为 5 个毫米便是规范，人们根据这个规范去做螺丝帽和螺丝钉。螺丝帽和螺丝钉相当于软件和硬件，指令集在里面起到让软件和硬件沟通时的规范作用。但要把一个指令集变成芯片产品还需要分成几个步骤：1）根据指令集设计微架构；2）根据微架构来做产品。

首先是指令集，其表现形式是手册。比如英特尔 X86 的手册单纯指令集有 2200 多页，ARM 的指令有 2700 多页，RISC–V 目前的指令手册是 200 多页。这些是指令规范，有了这个手册，你就可以知道每个指令的定义是什么。但仅仅这样是不够的，我们还只是在指令集层次上。

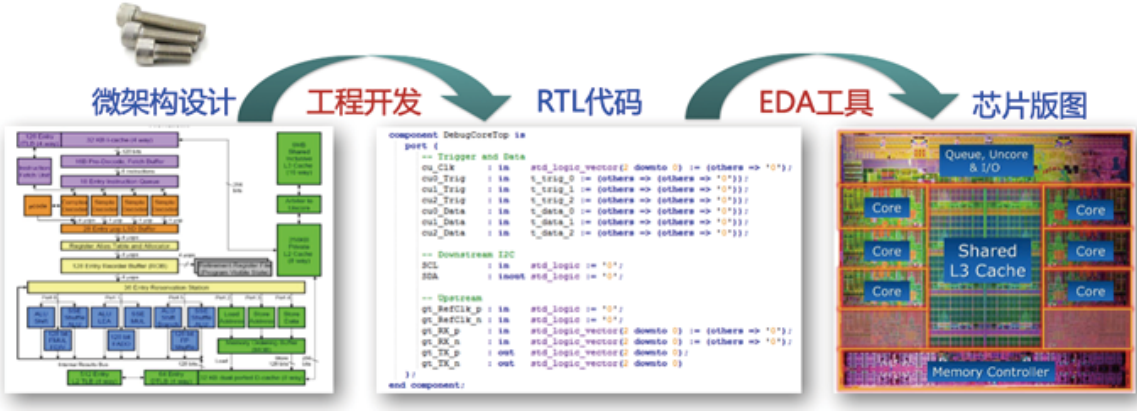


图 8：指令集变为具体产品的流程

图 9：指令集变为具体产品的流程

真正要变成的是微架构。微架构设计是把指令集手册里面定义的功能实例化，我们要开发源代码，真正变成 RTL 代码才是微架构的实现。然后我们再把 RTL 通过 EDA 工具链变成芯片版图，之后才能把芯片造出来。源代码从指令集到微架构，目前是芯片设计中所费时间最多的一块。有了芯片版图之后，便可以交给台积电或中芯国际等企业流片，然后由长电等企业完成封装，从而获得一个完整的芯片。

技术与市场	x86或ARM架构	RISC-V
架构篇幅	数千页	少于三百页
指令数	指令数繁多，且不同分支不兼容	基本指令集47条
模块化	不支持	支持
可扩展性	不支持	支持
硬件实现	复杂度高	硬件设计与编译实现非常简单
商业运作	x86封闭，ARM架构授权昂贵	开源、免费
生态环境	成熟	快速起步中
应用市场	服务器与桌面PC (x86)，移动和嵌入式 (ARM)	从物联网切入，可覆盖从微控制器到超级计算机的全计算领域
应用特点	服务器、PC、移动和嵌入式应用领域居垄断地位	高性能（面积、功耗和性能） 普适（可自由扩展和剪裁）可控（满足差异化和定制化）
应用风险	不可控、缺乏应用弹性、较高成本	生态不足、碎片化、专利问题

图 10：X86/ARM X RISC-V 对比小结

RISC-V 相较于 X86/ARM 有一些特点:

1. **指令更精巧。**X86 指令集在 2015 年已经有 3600 多条，而 RISC-V 采用模块化方式，基础指令 47 条，而且已经冻结，永远不会再变了。
2. **模块化。**实现一个 X86/ARM 处理器，需要实现上千条指令，复杂度极高； 而 RISC-V 指令集采用模块化设计，必要的只有 47 条，其余指令可选扩展，也可以根据需求自由组合，灵活适配，且支持自定义指令。
3. **开放免费。**如下表所示，Intel 处理器无论是指令集还是微架构都是封闭的，它们的服务是产品；ARM 指令集是可授权使用的，但只能做封闭设计； 基于开放免费的 RISC-V，则可以完全免费开放微架构的设计。这也正式开放指令集带来的真正价值所在。

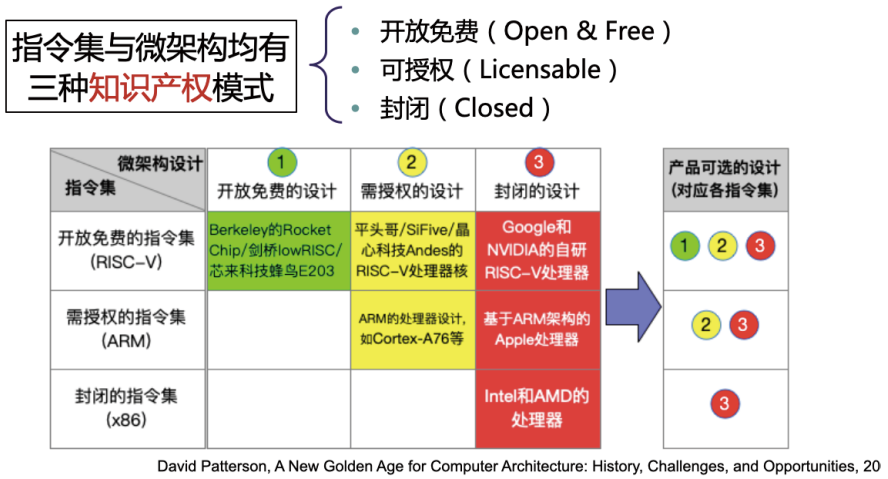


图 11：RISC-V 的特性

当然，尽管 RISC-V 开源、免费，但也有一些问题，比如生态不足、碎片化、专利问题等都是挑战。

三、软硬协同设计平台

RISC–V 的实践，平台很重要的。当前，大家都在做一些点的技术，但需要有一个平台把这些点整合起来。

包云岗认为这可以从几方面考虑:

第一，RISC–V 开源开放带来指令集可用;
第二，近几年高效的开发语言 Chisel 给我们带来便利;
第三，在处理器开发模式需要有新创新和理念变化，比如可以尝试用搭积木方式搭建芯片，即“乐高积木式开发”;
第四，怎样让芯片开发更加方便、成本更低。

这些都可以融合起来放到一个平台上，用一个平台把它们支撑起来，即支持芯片敏捷设计的软硬件协同设计平台。

	基于FPGA芯片的传统板卡模式	基于FPGA的SoC模式 (如Zynq*板卡)	FPGA云模式 (如亚马逊AWS F1公有云)
外设支持	• RISC-V处理器可通过控制器软IP或实现低速控制器访问外设	• 外设由FPGA内置硬核控制	• 外设一般由软件或RTL模拟
发行版Linux	• 需扩展NVMe SSD才能加载发行版Linux	• 需扩展NVMe SSD才能加载发行版Linux	• 通过虚拟块设备加载发行版Linux
用户干预	• 加电自启，无法额外配置	• 加电后需要用户参与配置	• 加电后需要用户参与配置
典型代表	• SiFive Freedom • ETH PULP Ariane • 平头哥无剑平台	• ETH PULP HERO • 剑桥lowRISC • UC Berkeley Rocket-Chip	• UC Berkeley FireSim

图 12：面向 RISC–V 的 FPGA 原型验证平台

全世界范围来看，现在越来越多采用 FPGA 开发，把这些资源需求融合起来，总体来说这里有三种模式，一种是直接在 FPGA 板卡上做验证，但这对系统级的仿真验证有一些开销。比如它只支持 Linux 等等，还有访存频率倒挂等等都还是挑战。另一个趋势是基于云的模式做芯片开发，这也是亚马逊提供 F1 平台，但它也带来一些挑战，这个架构本身就是用几个 FPGA 插在 host 主机上的，因此管理上会带来一些开销和通信。

包云岗介绍，他们团队提出了一种新的架构，相比于以上三种，使得密度更高、可扩展性更好、更方便。

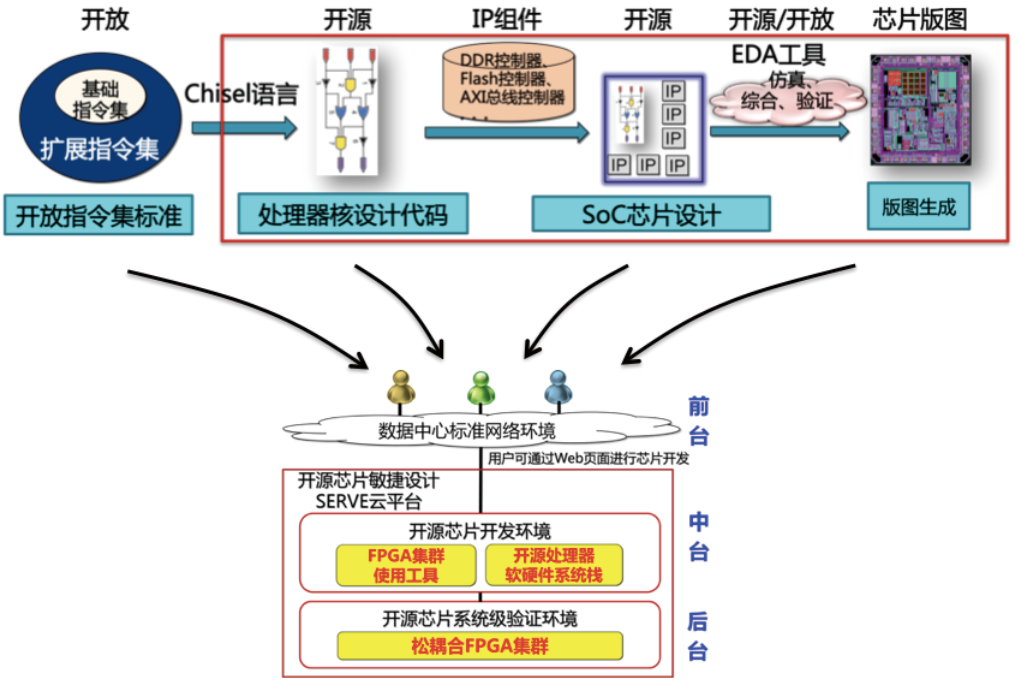


图 13：云平台设想

这个平台一方面能够把开源芯片设计的流程可以整合起来，集成指令集和开源处理器的过程，EDA 等过程都可以在这个平台上提供。同样，这也是一个云平台，包括有前台、后台、中台几个层次。

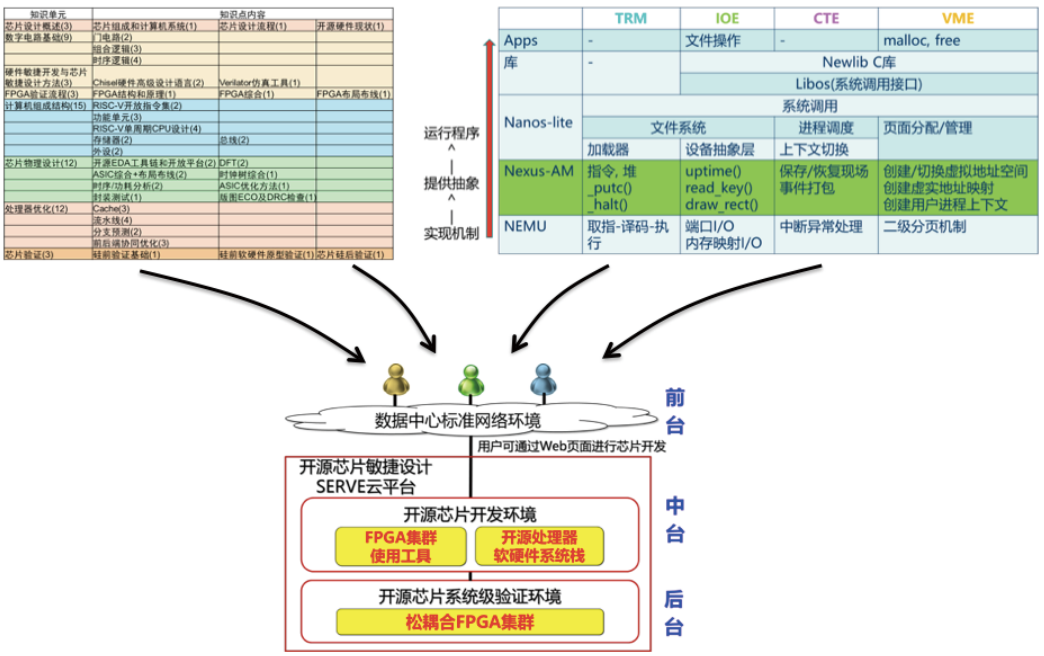


图 14：应用于国科大教学的云平台

基于这个平台，包云岗团队将它与教学耦合起来，在国科大教学中率先使用，因为这可以以更低的试错成本，不断迭代优化。

包云岗在报告中提到，该平台的目的包括： 一，尽可能地逼近真实系统； 二，支持跨层次开发，包括做 CPU 设计、OS 开发、APP 开发； 三、每一层都是可替换、可修改的； 四、使用便利，支持远程访问； 五、支持多用户共享，降低成本……

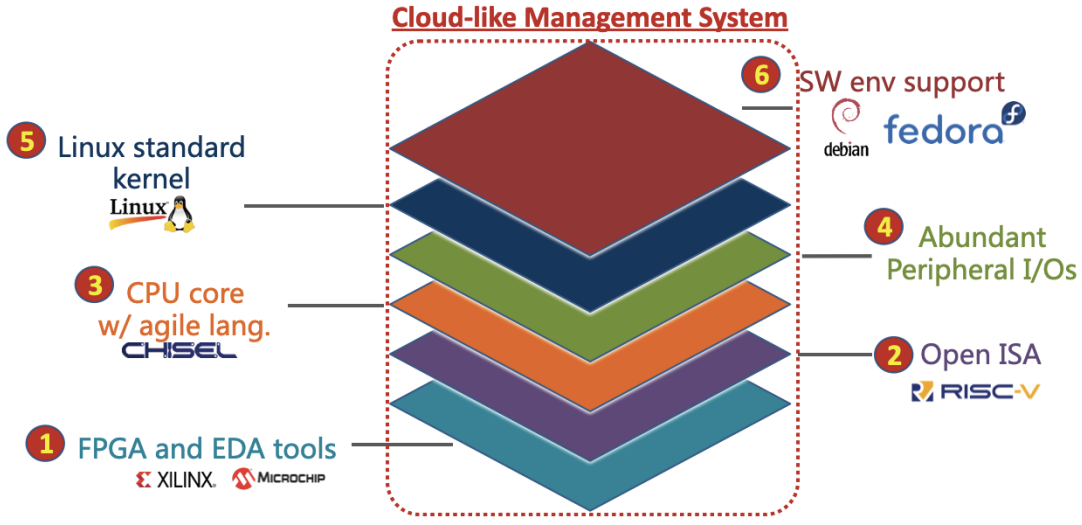


图 15：云平台架构

架构分为多个层次：1）底层是 FPGA 的阵列；2）上面可以支持 RISC–V 这种开放的指令集；3）然后我们可以支持在上面像敏捷开发语言做微架构设计；4）系统工作起来还要有控制器和串行总线等等外围的 I/O；5）支持操作系统，例如 Linux 操作系统；6）光支持操作系统也不够，还要有开发环境，支持 debian 等开发环境。这样形成全系统层次，便可以支持敏捷的、系统级的开发。除此之外，包云岗等人在设计了云平台的管理系统，这样能够做更好资源的虚拟化和多用户同时访问。

基于以上的思考，包云岗等人提出了面向 RISC–V 生态的系统级原型验证服务平台——思沃（SERVE），该平台可以支持系统级的开发。

图 16：思沃平台

包云岗介绍，思沃平台的发展主要分为三个阶段：

第一版可追溯至 2016 年，当时平台成功验证了云平台关键技术；
第二版则根据国科大的教学需要，在 2017 年进行了专向定制；
第三版不但可以支持复杂开源处理器核验证，同时将 FPGA 资源提升了 11 倍。第三版产品最快将会在今年面世。

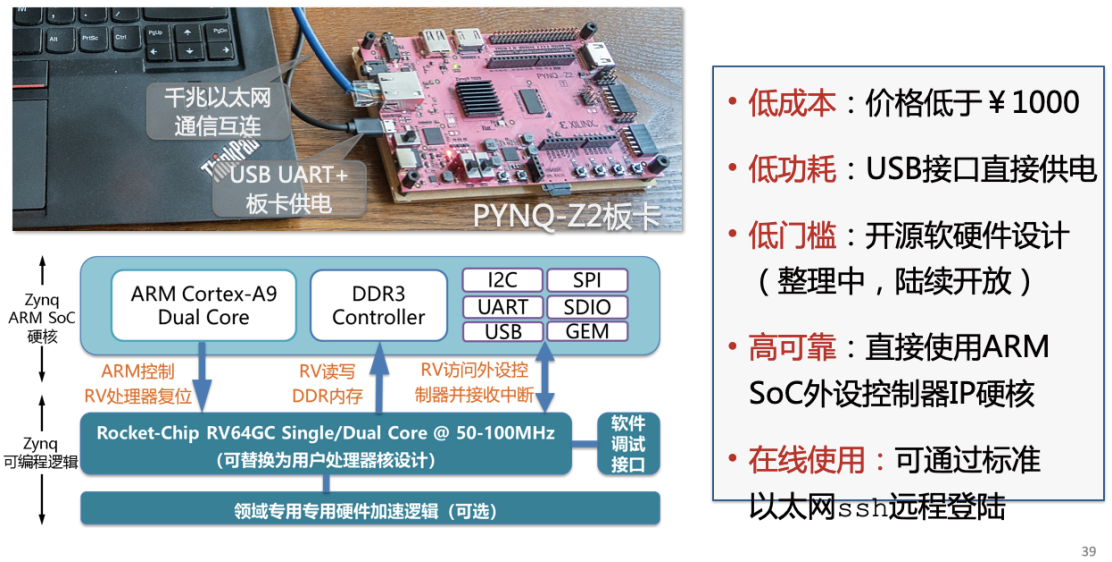


图 17：低成本 RISC-V 全系统原型验证平台

第一代思沃产品的板子由于是自行供应的，不但成本低，还将包括开发工具在内的系统进行了集成。此外，这一代全栈开发环境对开发者非常友好，同时支持 Linux、kernel 等文件系统。

图 18：SERVE.r – 低成本 RISC-V 全系统原型验证平台

图 19：SERVE.c – 云平台版本

在第一代思沃产品的基础上，包云岗与团队搭建了云平台。由于是为国科大专向定制，因此每个节点都带有国科大 logo，且可以每天 24 小时不间断提供服务。

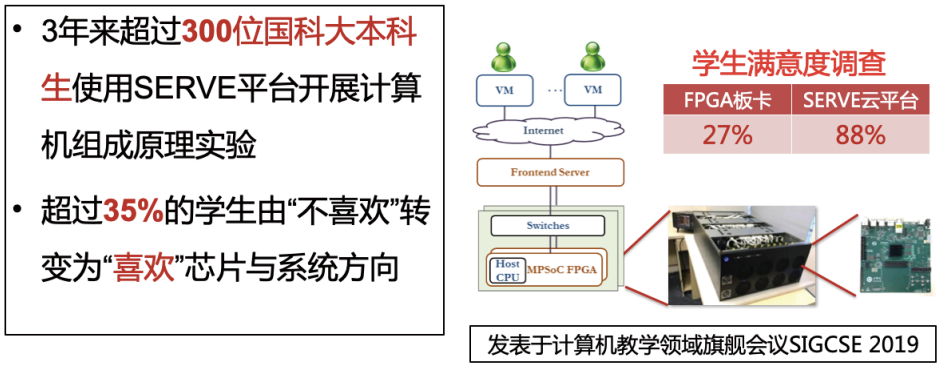


图 20：国科大组成原理教学实践，使用 SERVE 平台为国科大本科组成原理实验服务

包云岗表示，过去几年已有超过 300 名国科大本科生在思沃平台进行实验开发，学生使用平台后的满意度高达 88%，另外还有 35% 的学生从一开始对硬件芯片开发无感，在使用平台以后变得喜欢起来。这个结果给了包云岗团队很大鼓舞，他们相信只要坚持把平台做下去，未来将能让更多学生不再害怕这个方向的工作。

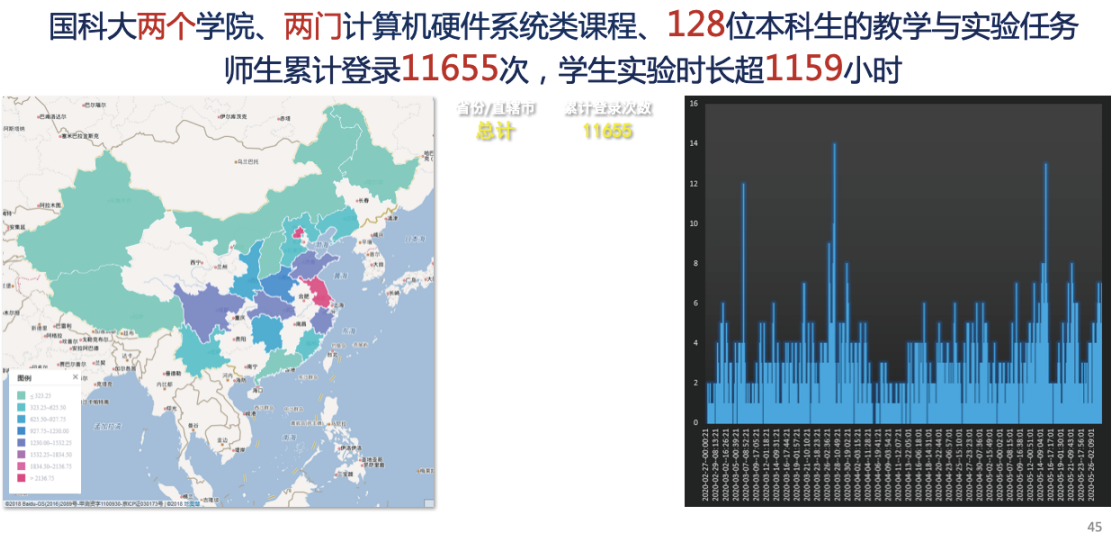


图 21：疫情期间保障实验工作有序推进

另外值得一提的是，疫情期间思沃平台成功确保学生硬件实验的有序推进。以 2020 年 5 月 28 号的数据为例，上万次登陆、上千小时的实验时长，全国各地的 128 位国科大学生通过平台完成了课程实验。

- “龙芯杯”全国大学生计算机系统能力培养大赛屡获佳绩
  - 2018年特等奖、2019年一等奖

- 北京市优秀本科毕业论文 (全校2位)
  - 学生：徐易难
  - 导师：包云岗

图 22：培养成果

此外，这个平台也让学生的能力得到明显的提高，除了在比赛中屡获佳绩，包云岗指导的一名学生所做的访存调控机制，甚至被应用在华为手机里，并拿到了北京市优秀本科毕业论文。

- 2019年8月底，首批5位国科大2016级本科生，以流片为目标，以一款教学版软件模拟器为基础，实现完整的RISC-V核+SoC，并于12月17日完成第一次投片

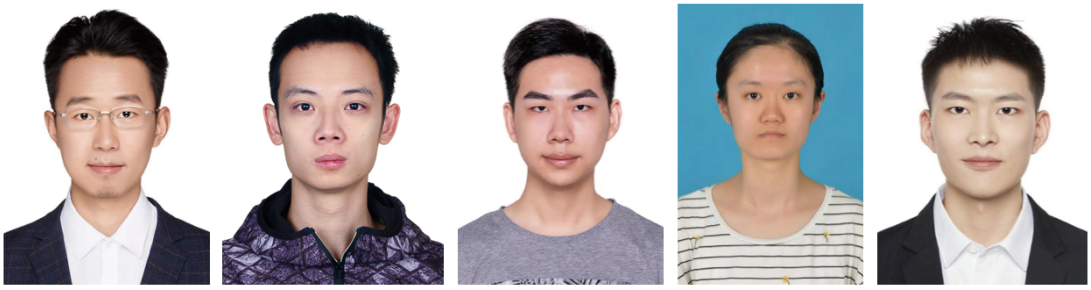


图 23：学生带着自己的处理芯片毕业

包云岗甚至将开源芯片、敏捷设计和本科实践结合起来，让学生最终可以带着自己设计的处理器芯片毕业。该计划从去年 8 月开始启动，共有 5 位本科生参与，他们需要设计一个 RISC-V 芯片并流片。包云岗表示，学生最终的答辩演示效果很理想。

图 24：将学生的工作进行开源 – 源代码 + 软件模拟器 +FPGA 仿真环境

此外，他们还会将学生所做的工作进行开源，让业界进行检验，正在整理的源代码包括开发过程中的模拟器、FPGA 环境下正常工作的核等等。

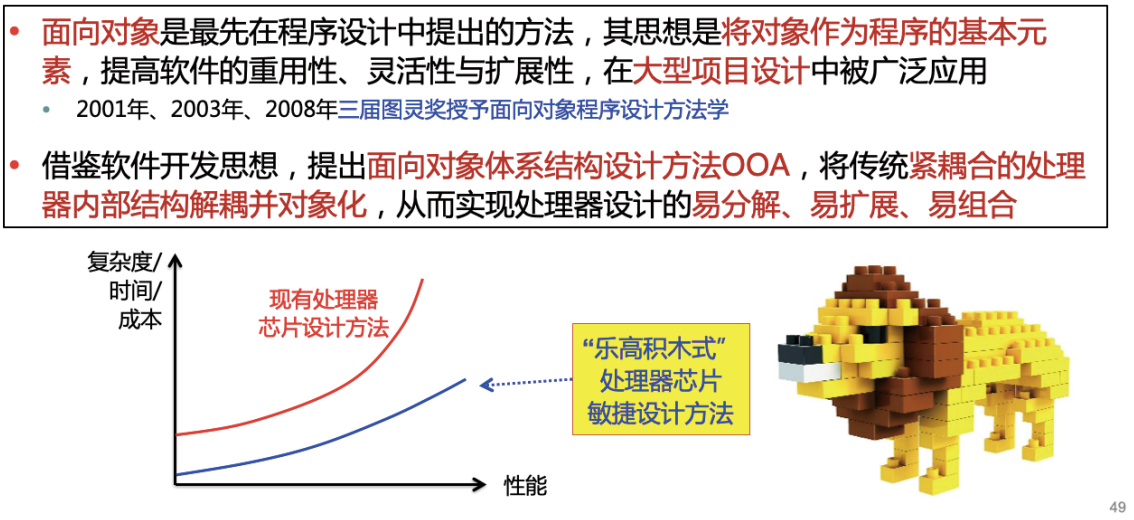


图 25：乐高式敏捷设计方法

从 8 月份启动到 12 月份流片，在这么短的时间内要把一个处理器实现出来，并确保 Linux 等操作系统得以应用，背后的设计理念会跟原来的设计方法有很大的不同。包云岗称之为“乐高式敏捷设计方式”，主要借鉴了软件工程面向对象的思想，先将一个个处理器模块变成对象，最后再将它们综合起来。

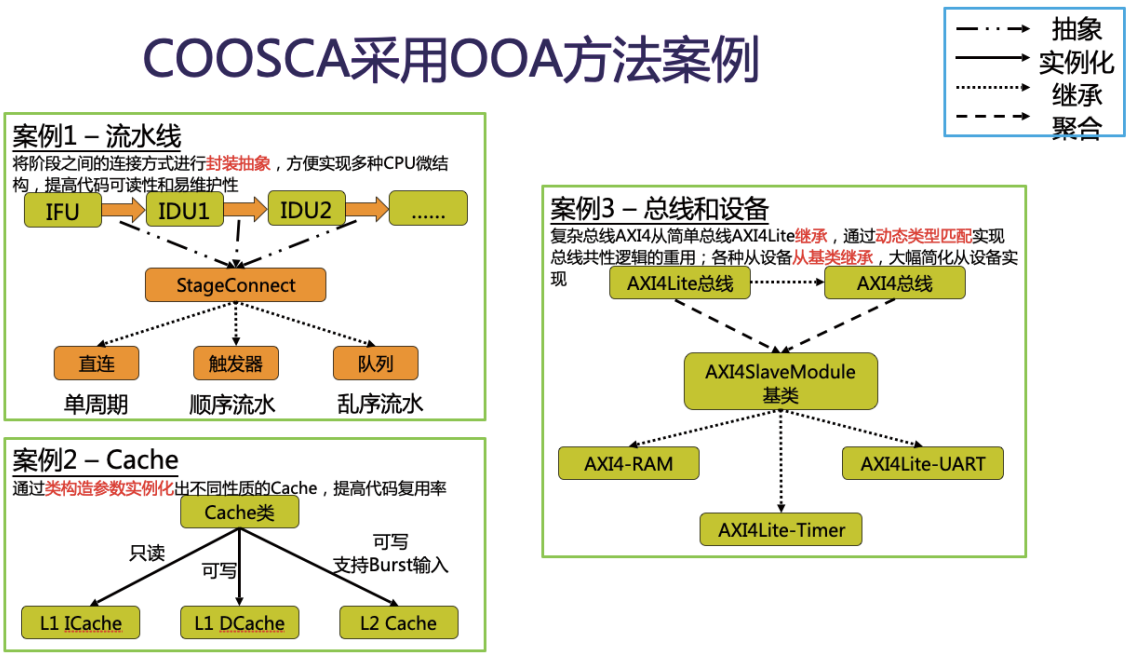


图 26：有效提高代码的复用程度

举个例子，流水线对象化，有了集类就会派生出不同的流水集，就可以根据这些类来进行扩展，也相当于对某些功能实例化。这样可以大大提高代码的复用程度，有利于加速芯片的开发。虽然是第一次尝试，但包云岗表示已经看到很好的效果。

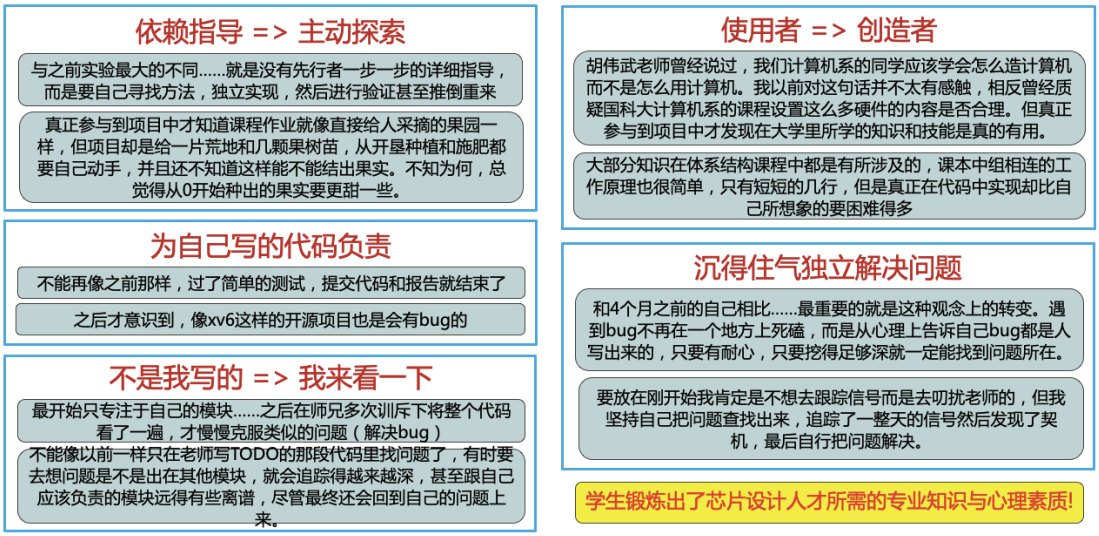


图 27：学生感悟

包云岗强调，整个探索过程最大的收获在于学生的成长，不光锻炼了专业知识，还在心理素质上也有明显的提高，比如教会学生独立探索；使学生对学习的课程有更深的理解；以及一些心态和观念上的变化等。

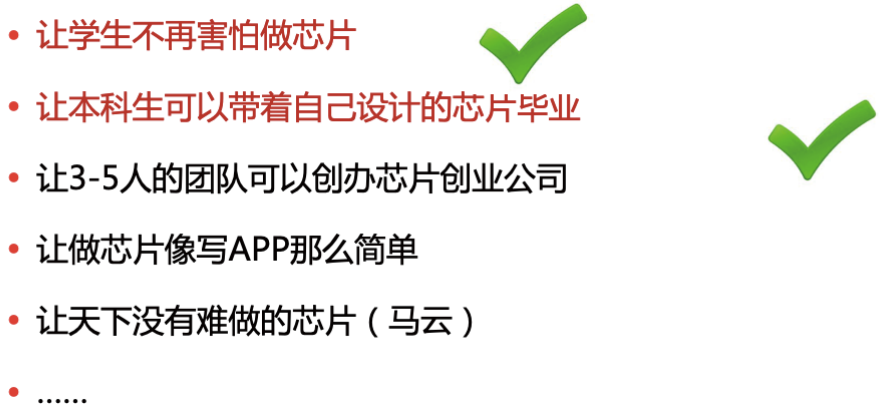


图 28：已实现的目标

总的来说，该实践项目已经可以做到以下这两点： 第一，让学生不再害怕做芯片，第二，让本科生可以带着自己设计的芯片毕业，特别是复杂度很高的处理器芯片。

四、结论

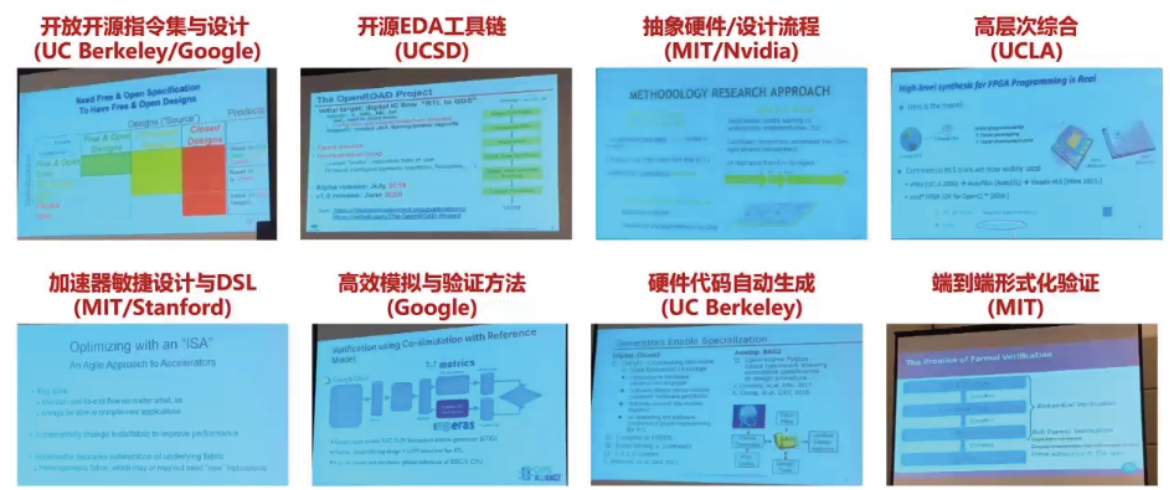


图 29：远景研讨会话题 — 开源 + 敏捷设计

包云岗表示很多人存在一个认知误区，就是认为开源芯片不过是把源代码开放出来，实际上没那么简单。

去年的愿景计划报告就涉及到当中的许多问题，不光是指令集，还有工具链、抽象设计流程，甚至端到端的形式化验证，都是未来可以拓展的研究工作。另外，随着越来越多硬件源代码被开放出来，在硬件上找 BUG 会带来很大价值——意味着有更多新问题、新方法会被发掘出来。

图 30：Micro 期刊关于开源芯片与敏捷开发主题的特刊

包云岗将会主理 Micro 期刊今年 8 月关于开源芯片与敏捷开发主题的特刊，他表示当中收录的 13 篇论文当中只有 1 篇来自国内，他也趁此机会呼吁同行更多关注这个新的行业发展趋势。

百度欧阳剑：百度昆仑芯片——适用于各种工作负载的 AI 处理器

整理：智源社区 黄善清

百度智能芯片团队总经理欧阳剑的报告主题是《百度昆仑芯片——适用于各种工作负载的 AI 处理器》，在报告中，他从工业界和用户的角度，分享了 AI 芯片面临的主要挑战，以及百度的相关解决方案等。

一、AI 计算背景简介

欧阳剑在报告中将 AI 应用划分为三大类:

第一类为**语音类应用**，如语音识别、语音合成等;
第二类为**图像类应用**，如图像分类、检测、分割等;
第三类为**自然语言理解类应用**，这类应用虽不像语音和图像类那么直观，但它存在于人们生活常见的推荐系统、对话系统当中，应用上非常广泛。

图 1：AI 计算场景

此外，AI 使用场景如今也非常多样化，如企业的云数据中心、各地兴建的超算中心、新基建的智慧工业、一直很火热的自动驾驶领域等等。

二、百度昆仑产品简介

欧阳剑在报告中抛出了一个核心问题：用户究竟需要一个什么样的 AI 芯片？他认为 AI 芯片必须比 GPU 具备更多的优势，才能真正打动用户，这些优势包括了：性能好，价格低廉；通用。

目前市面上的 AI 芯片主要分为两类：一类为通用 AI 芯片，能支持不用的应用场景，且具有良好的可编程性及应用性；另一类为专用 AI 芯片，它专攻某一类需求，在服务器端、数据中心端用得较多。

换句话说，用户一般对通用 AI 芯片的期待值较高，通用 AI 芯片因此面临设计复杂度高、强依赖复杂软件栈的问题。

核心指标	GPU	百度昆仑	专用AI芯片
通用性	很高	高，能支持图像，语音，NLP等多种模型	一般只针对图像模型优化，其他模型做不好或者不支持
可编程性	很高	高，运行用户使用C/C++编程	粗粒度命令式控制，可编程性一般
实际性能	一般	高	一般
性价比	一般	高	一般

	内存带宽(GB/s)	Int8算力(Tops)	Int16/fp16算力(Tops)	Fp32/int32算力(Tops)	是否支持训练	开发工具栈
昆仑1	512	256	64	16	是	C/C++
T4 GPU	320	130	65	8	是	CUDA C/C++

图 2：对比结果

欧阳剑用一张表介绍了百度昆仑芯片的具体参数，并将之与 GPU 进行对比，得出以下结论:

1. 通用性。今天很大一部分 AI 芯片通用性比不过 GPU ，而百度昆仑可以较好地支持语音、图像、NLP 等不同模型的应用;
2. 可编程性。GPU 的可编程性非常高，百度昆仑能做到支持 C 和 C++ 语言进行编程;
3. 实际性能;
4. 性价比。

在具体参数上，百度昆仑芯的带宽做得比 T4GPU 要高； 在 INT8 和 FP32 的算力上也是佼佼者； 开发工具栈上用的是 C 和 C++，并做了语法扩展。

三、百度昆仑芯片技术介绍

做通用 AI 芯片的想法很美好，但在实现上却非常困难，欧阳剑在报告中向大家介绍了百度是如何应对这些挑战的。

- **超过8年的FPGA AI加速器积累，累计上线超过1万个**
- **XPU架构及软件栈在实际业务中超过8年的持续迭代**
- **X- diverse**

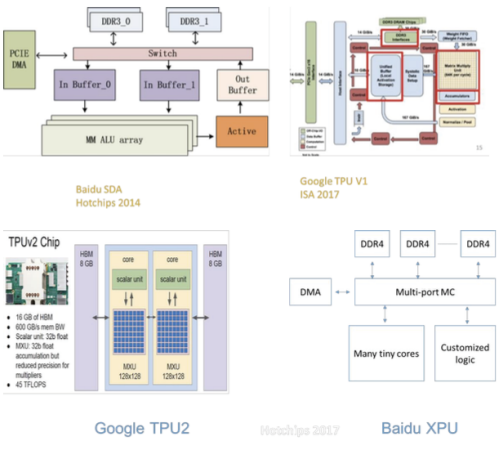


图 3：XPU 架构

简而言之，做 AI 芯片需要对芯片架构、软件栈及需求理解上有所积累。

早在 2010 年，百度便开始利用 FPGA 做 AI 加速计算。当时没选择做芯片，是考虑到这是一件大事情，所以百度选择先从 FPGA 做起，涉猎诸如定制化、差异化计算的工作，从而累积势能。在 2011–2017 年之间，百度内部就上线了超过 1 万多个 FPGA 的加速器应用。

在这过程中，团队对硬件架构对用户需求有了较深积累，并提出 XPU 架构—— X 代指“多样性”，表示应对多样化的 AI 计算需求。欧阳剑补充道，XPU 架构的提出源于团队过在 FPGA 的经历中发现用户的需求经常变动，而模型与软件也需要经常经历迭代，一旦架构不通用，在后续的推进上会变得愈发艰难。

有了这些积累后，百度在 2018 年正式立项昆仑芯片项目，2019 年百度昆仑流片，到 2019 年底芯片正式回来，前后经历了一年半至两年时间。

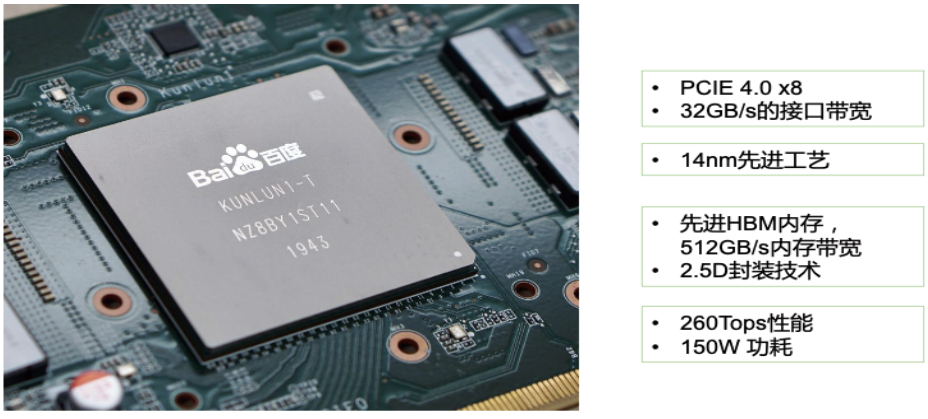
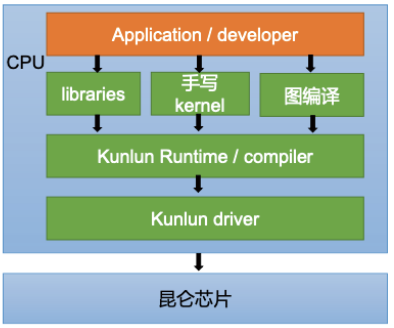


图 4：百度昆仑的特性

值得一提的是，百度昆仑是业界较早采用 PCIE4.0 接口 的 AI 芯片，因此能达到较高的接口带宽。除了采用先进的 14 纳米工艺，百度昆仑还采用了 HBM 内存，使内存带宽达到 512GB/s，且封装采用的是 2.5D 技术。最终测试结果显示，百度昆仑理论性能为 260Tops，功耗则是 150W。

图 5：板卡实现

这是做成板卡后的模样，大小和尺寸与 GPU 版卡一样。除此还有另一版 P100 板卡。



- AI芯片的胜负，关键在于软件栈
- 百度昆仑提供类似CUDA的软件栈
  - 底层驱动，runtime
  - C/C++的编译器
  - 编程模型
  - 类似tensorRT的XTCL
  - 类似cublas的高性能库

图 6：编程模型及软件栈

欧阳剑认为，做好 AI 芯片的软件挑战也非常大，除了编程模型等具体技术，还涉及到一个生态门槛的问题。

而百度希望打造类似 CUDA 的软件生态——以上图为例，用户有需求可以直接调用 Libraries，百度昆仑会提前包装好库，且支持手写 Kernel。

同时百度昆仑也涉猎了当下 AI 领域较火的图编译，欧阳剑解释道，这是因为 AI 计算的算法、模型开发者占比较大，比起库或 Kernel，他们更希望通过输入图描述，直接转成底下可执行的文件，再放到芯片里执行。

这里有一个具体的编程例子。

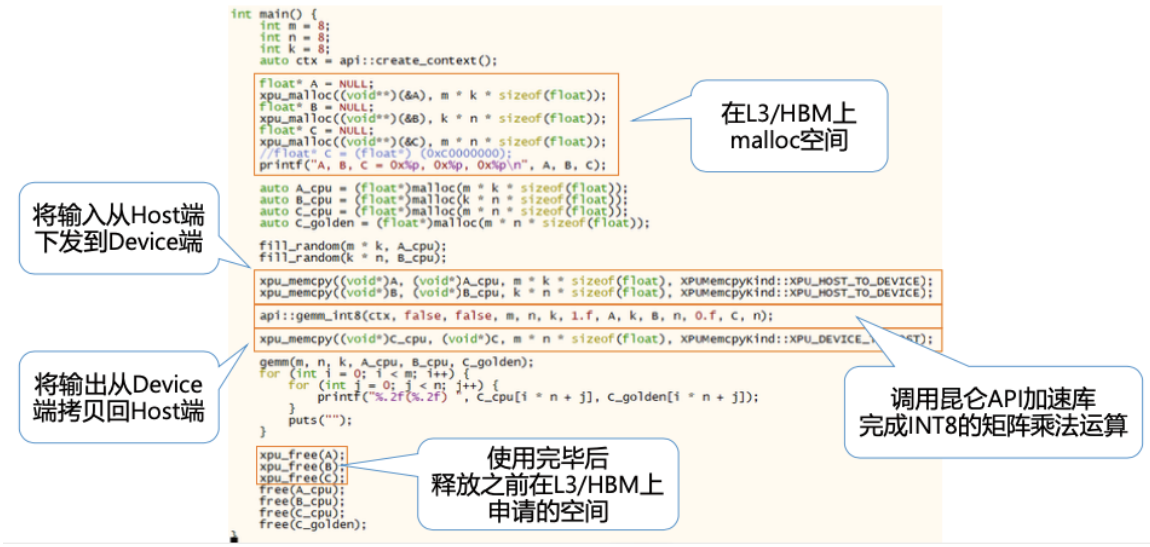


图 7：编程实例

其中涵数跟 CUDA 较像，都是一开始在芯片 L3/HBM 上 Malloc 空间，再把数据拷贝过去，调完计算后又把数据拷回来。换句话说，上手门槛较低。

四、百度昆仑芯片应用案例

为了证明芯片已经达到设计目标，欧阳剑在报告中举了几个测试例子。

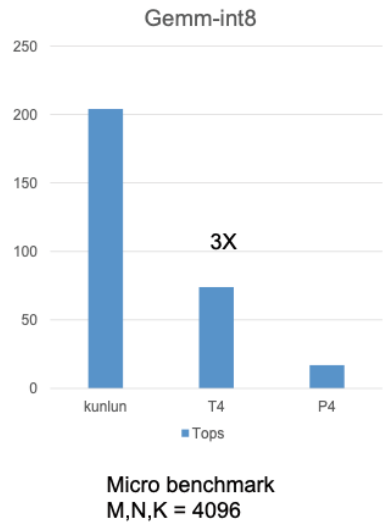


图 8：基础 benchmark 的对比结果

首先是标准的矩阵乘矩阵的计算，欧阳剑团队选择了 int8 数据精度，假设是 4K × 4K 两个矩阵做乘法，百度昆仑能跑 200 多 Ops，而 T4 大概在 40oPS 左右，在 micro Benchmark 中，昆仑跑出的结果比 T4 高了 3 倍多。

测试模型：

Bert_Base_Uncased. 12 layer, heads_num = 12, hidden_size = 768.
Sequence length = 128
GPU：TensorRT-FP16
Kunlun：int16

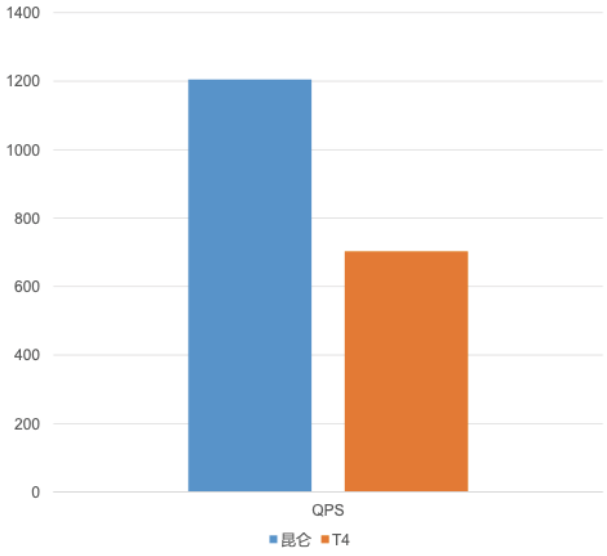


图 9：模型 QPS 测试对比结果

接着是模型测试。在相应配置下，昆仑的 QPS 可以做到 1200 多，而 T4 大概是 700 多，意味着在 Bert 模型下，百度昆仑的性比 T4 高上 1.7 至 1.8 倍。

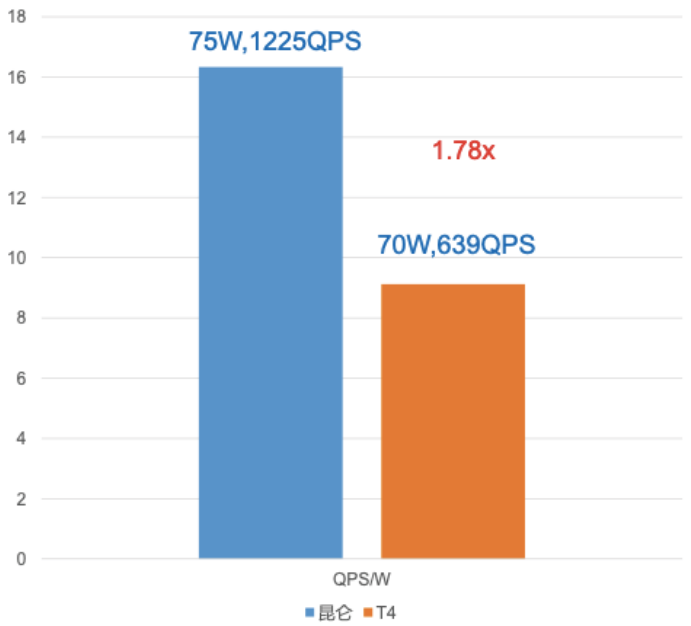


图 10：模型能耗效率对比结果

现在来看一下大家最关注的能耗效率问题，百度昆仑在 1250 QPS 的功耗是 75W，而 T4 在 639QPS 的功耗是 70W——按每瓦特 QPS 进行计算，百度昆仑差不多是 16 点多，而 T4 是 8 点多，意味着能耗效率上昆仑比 T4 有了 1.78 倍的提升。

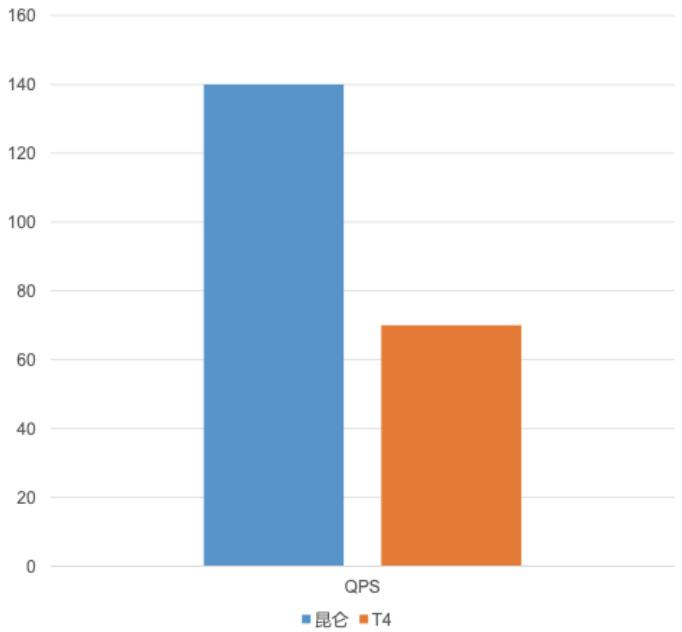


图 11：YOLOV3 模型对比结果

与 YOLOV3 模型进行对比，百度昆仑在性能上有近 2 倍的提升；能耗效率方面，百度昆仑大概是 T4 的 1.44 倍。

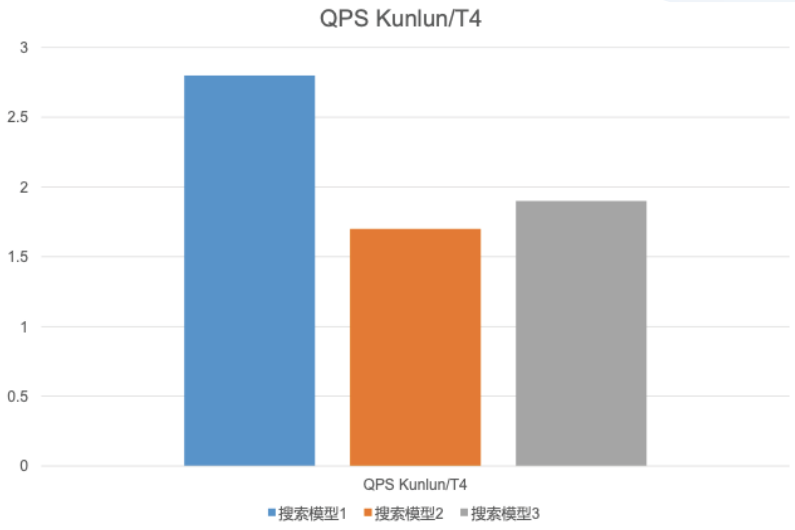


图 12：实际线上系统性能数据结果

欧阳剑介绍，除了开源的标准模型，百度昆仑去年底流片回来后也在百度内部做了大规模推广。图中可以看到百度内部使用的 3 个模型，相对 T4 而言，QPS 大概在 1.6 倍、1.78 倍到 2.78 倍之间，效果非常不错。

图 13：百度伙伴的工业智能应用场景

除了百度数据中心，我们还能看到百度昆仑在工业智能场景的使用。

欧阳剑介绍，百度的某个合作伙伴将 AI 芯片应用在了一个质检工业机器上，有效提升了工业检测的效率——过往的生产流水线需要很多工人做低效的肉眼检测，用上光学仪器与高效的 AI 计算后，检测效果大大提供，且不受人为主观因素的影响。

欧阳剑最后表示，除了推理场景，百度昆仑还可以用于训练任务，目前 Bert 的训练已经能够跑通整个过程。虽然在性能上比不过最新的 V100 或 N100——当初立项时并未定义那么高的功耗与性能标准，但昆仑芯片的优势在于性价比极高，且功耗和成本相对较低。要是基于性价比来考虑大规模训练系统的话，还是具有一定优势的。