Processing-in-Memory for Energy-efficient Neural Network Training : A Heterogeneous Approach

Jiawen Liu .etc , Micro 2018

范志华

2019-04-18

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

Background

- 由于处理器和内存之间频繁的数据搬运，神经网络的训练过程有很大的能量和时间开销。此外，训练过程包含大量复杂的计算和存储器访问操作，难以提高并行度。
- 现有的研究工作使用低精度数据或修剪NN模型，然而这些技术并没有从根本上解决NN模型训练中的数据搬运带来的能耗和时间开销问题。
- 神经网络训练可以显着地受益于异构PIM（process in memory）：其中包括固定功能逻辑和可编程内核 。以实现最佳能源效率和并行性与可编程性之间的平衡。

Motivation

TABLE I: Operation profiling results for three neural network models. "CI"= computation intensive; "MI"=memory intensive.

VGG-19					
Top 5 CI Ops	Execution Time(%)	#Invocation	Top 5 MI Ops	#Main Memory Access(%)	#Invocation
1. Conv2DBackpropFilter	40.15	16	1. Conv2DBackpropFilter	42.52	16
2. Conv2DBackpropInput	32.68	15	2. BiasAddGrad	35.68	16
3. BiasAddGrad	11.92	16	3. Conv2DBackpropInput	21.06	15
4. Conv2D	10.34	16	4. MaxPoolGrad	0.22	16
5. MaxPoolGrad	1.49	16	5. Relu	0.14	19
Other 13 ops	3.37	232	Other 13 ops	0.38	229
AlexNet					
Top 5 CI Ops	Execution Time(%)	#Invocation	Top 5 MI Ops	#Main Memory Access(%)	#Invocation
1. Conv2DBackpropFilter	33.64	5	1. BiasAddGrad	44.64	3
2. Conv2DBackpropInput	33.46	4	2. Conv2DBackpropInput	36.61	4
3. MatMul	13.54	6	3. Conv2DBackpropFilter	14.79	5
4. Conv2D	10.48	5	4. Relu	1.20	8
5. BiasAddGrad	4.62	3	5. Conv2D	0.46	5
Other 13 ops	4.26	121	Other 13 ops	2.30	119
DCGAN					
Top 5 CI Ops	Execution Time(%)	#Invocation	Top 5 MI Ops	#Main Memory Access(%)	#Invocation
1. Conv2DBackpropFilter	19.98	4	1. Conv2DBackpropFilter	37.21	4
2. Conv2DBackpropInput	17.18	4	2. Conv2DBackpropInput	28.09	4
3. MatMul	14.28	12	3. Slice	17.18	14
4. Conv2D	10.53	4	4. Conv2D	5.45	4
5. Mul	9.89	84	5. Mul	2.22	84
Other 47 ops	28.14	821	Other 47 ops	9.85	819

- 只有几个操作主导了训练执行时间
- 最耗时的操作也是内存最密集的操作
- 耗时且占用大量内存的操作需要异构计算类型

Motivation

- 只有几个操作主导了训练执行时间
- 最耗时的操作也是内存最密集的操作

采用PIM架构来加速NN训练，以减少主处理器和主存储器之间的数据搬运。

- 耗时且占用大量内存的操作需要异构计算类型

→ 采用异构PIM架构，结合固定功能逻辑和可编程内核

Design

- 总体设计

本文总体设计为软硬件协同的设计，主要包括三个部分：

1、采用异构PIM架构，它将固定功能逻辑和可编程内核（ARM）集成在3D叠层主存储器中。

2、扩展了OpenCL 编程模型，将主机CPU和异构PIM映射到OpenCL的平台模型，以实现高效的运行时调度。

3、提出了一个运行时系统，它可以最大化PIM硬件利用率和NN操作级并行性。

Design

- 总体设计

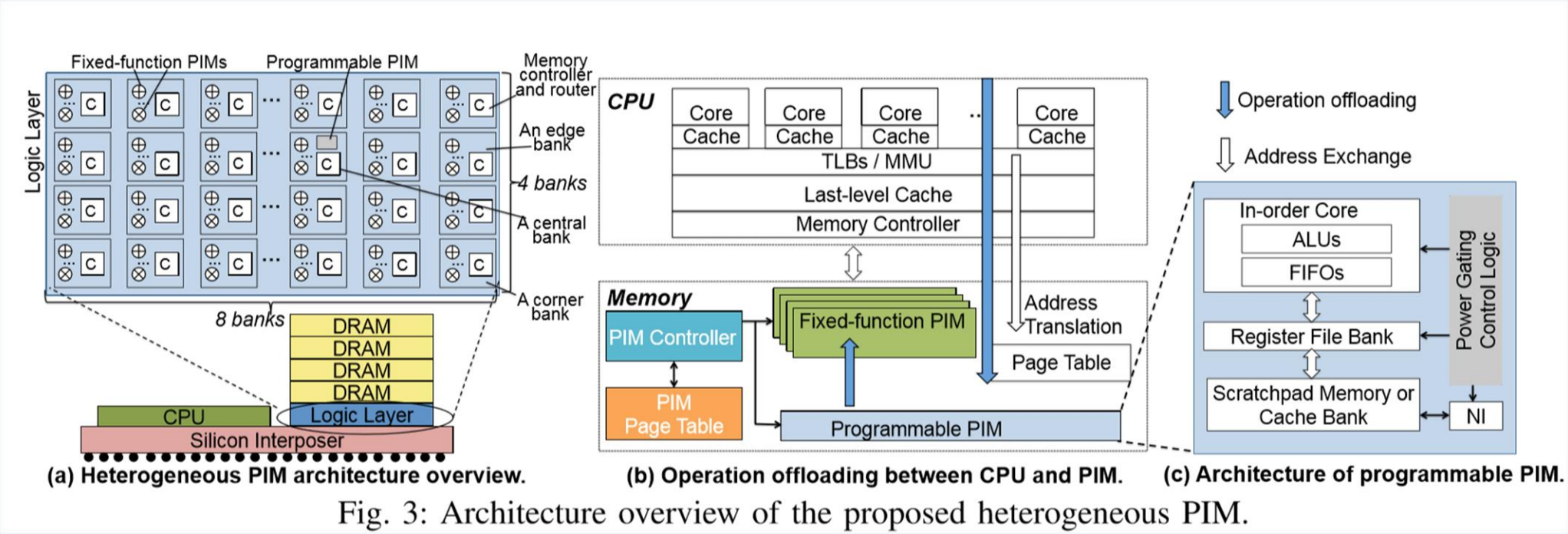
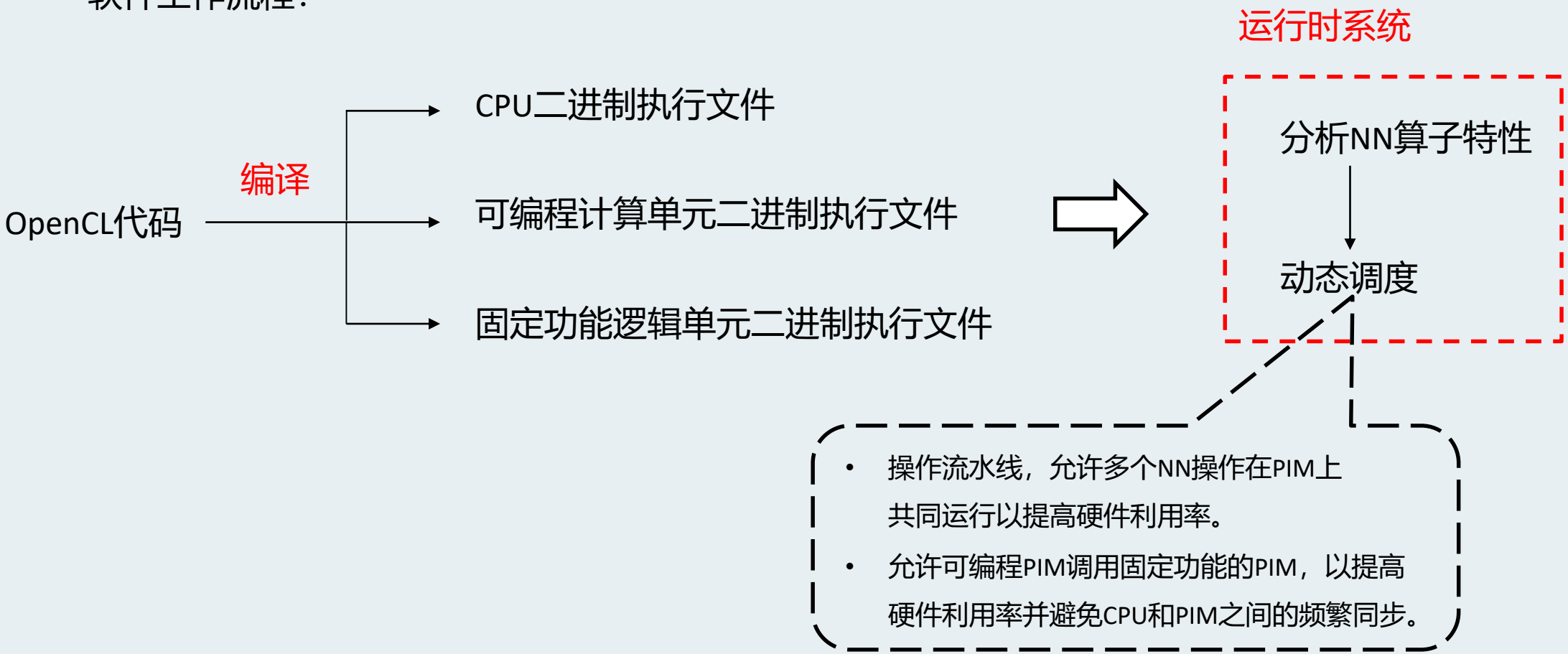


Fig. 3: Architecture overview of the proposed heterogeneous PIM.

Design

- 软件工作流程：



Design

- 异构PIM架构
  - ✓ 可编程的PIM，ARM处理器
  - ✓ 固定功能PIM，由分布在内存bank中的加法器和乘法器构成
- 异构PIM架构编程模型

  基于扩展的OpenCL模型
  - ➢ 允许不同计算单元之间互相调用，以支持复杂的NN操作
  - ➢ 通过流水线以有限的并行性提高小型NN操作的硬件利用率
  - ➢ 扩展内存模型以支持主机处理器和加速器之间共享的单个全局内存
  - ➢ 在不同的PIM和CPU（主机处理器）之间添加显式同步，以跨NN操作强制执行命令

在内核调用固定功能的PIM结束之前，整个固定功能的PIM更新到内存位置是不可见的

Design

Execution Time	Memory Access %	Example Operations
Long	Low	*Conv2D* in VGG-19
Long	High	*Conv2DBackpropFilter* in VGG-19
Short	High	*Slice* in DCGAN
Short	Low	*Reshape* in AlexNet

- 运行时系统

目的：最大限度地利用PIM和CPU来优化系统吞吐量

1、分析（运行一次迭代过程，找出执行时间长且频繁访存的算子）

运行时分别统计不同算子的执行时间和主存储器访问次数，两个表按照降序排列，每个算子具有两个索引。 对于每个算子，将两个表中的索引相加得到全局索引，运行时将全局列表中的靠前的操作来移植到PIM。

2、调度

✓ 尽可能多得将操作移植到固定功能逻辑单元

✓ 尽可能多得将操作移植到PIM

✓ 调度过程需要满足数据依赖性

3、流水线

当某个操作不能完全利用固定功能的PIM时，会调度下一步中的操作，利用空闲的固定功能PIM执行其部分计算，只要这两个操作没有依赖性。

Implementation

- API functions for PIM
  - ✓ 将算子加载到特定的PIM单元
  - ✓ 跟踪PIM单元的状态，包括是否空闲
  - ✓ 询问特定的算子是否计算完成
  - ✓ 询问指定算子的计算单元地址以及输入/输出数据地址

Name	Description
int pim_fix(int* pim_ids, void* args, void* ret, size_t num_pim)	Asks specific fixed-function PIMs to work with input arguments `args` and return results `ret` and a work ID.
int pim_prog(int pim_id, pim_program kernel, void* args, int* args_offset, void* ret, size_t ret_size)	Asks a programmable PIM to work on a kernel (an operation) and return a work ID.
int pim_status(int pim_id)	Checks whether a specific PIM is busy.
int work_query(int work_id)	Checks whether a specific operation is completed.
void work_info(int work_id, int* pim_ids, int* data_loc)	Queries the computation location (pim_ids) and input/output data location (i.e, which DRAM banks) for a specific operation.

Implementation

- 运行时系统实现

1、CPU运行时系统

通过添加大约2000行代码来扩展TensorFlow的运行时系统，CPU上的运行时根据低级API提供的硬件利用率信息调度CPU和PIM上的操作。

- ✓ 使用OpenCL内在函数进行设备初始化和表征
- ✓ 为PIM设备创建设备上下文和实例
- ✓ 为Tensor流程的其他组件提供新的OpenCL设备抽象
- ✓ 一种与可编程的运行时通信的机制

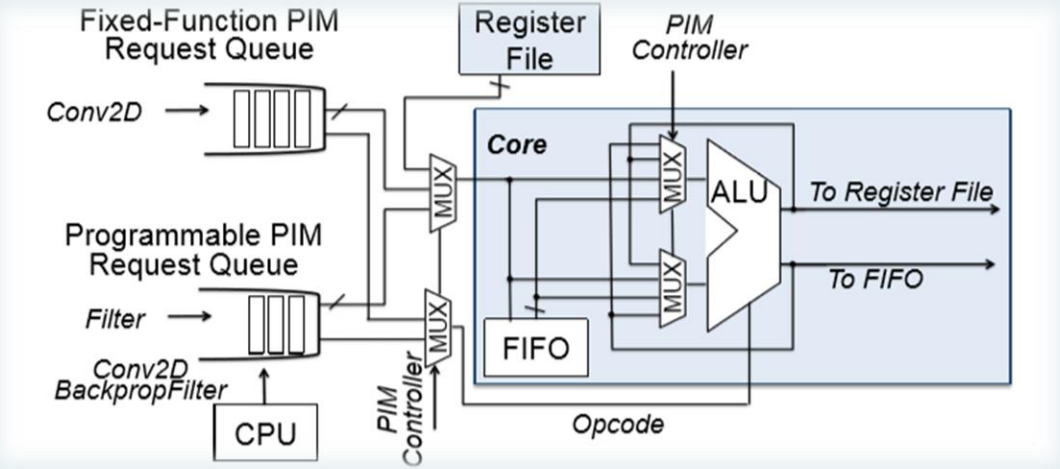
2、PIM运行时系统

可编程PIM上的运行时支持递归PIM内核和操作流水线

Implementation

- 硬件实现

  - ✓ 一个ARM Cortex-A9 可编程计算单元，频率为2GHZ
  - ✓ 固定功能逻辑单元为32位浮点加法器和乘法器（成对出现），一共排布444个，分布在内容bank的边缘和角落



Experimental setup

- 仿真
  - ✓ 固定功能逻辑单元：Synopsys Design Compiler
  - ✓ 可编程逻辑单元：Prime-Time
  - ✓ 3D 存储：HMC 2.0 参数和配置，312.5MHz
  - ✓ 指令跟踪：Pin
  - ✓ 执行时间计算：基于python设计的框架

Experimental setup

- 功耗和面积建模
  - ✓ 10nm工艺：CPU、PIM
  - ✓ 25nm工艺：DRAM
  - ✓ CPU功耗评估：VTune
  - ✓ GPU功耗评估：nvidia-sim
  - ✓ 可编程PIM功耗面积评估：McPAT
  - ✓ 固定功能PIM功耗面积评估：Synopsys Design Compiler、PrimeTime

Experimental setup

- 测试程序

参数 \ 模型	VGG-19	DCGAN	ResNet-50	Inception-v3	LSTM	AlexNet
数据集	ImageNet	MNIST	ImageNet	ImageNet	PTB	ImageNet
Batch size	32	64	128	32	20	32

- 硬件配置

CPU	Intel Xeon E5-2630 V3@2.4GHz
Main memory	16GB DDR4
Operating system	Ubuntu 16.04.2
GPU	NVIDIA GeForce GTX 1080 Ti (Pascal)
GPU cores	28 SMs, 128 CUDA cores per SM, 1.5GHz
L1 cache	24KB per SM
L2 cache	4096KB
Memory interface	8 memory controllers, 352-bit bus width
GPU main memory	11GB GDDR5X

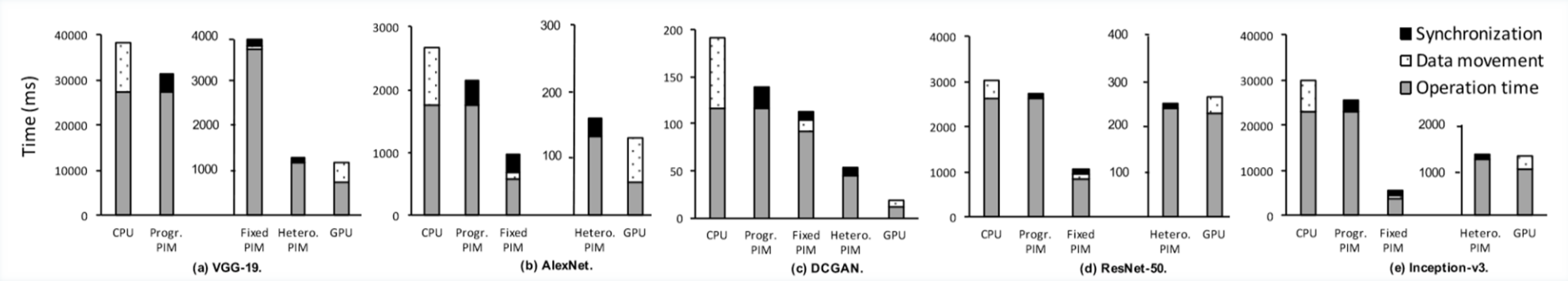
Evaluation

本文对比了5种不同配置的性能：

- **CPU**：所有的运算都在CPU上进行
- **GPU**：所有的运算都在GPU上进行
- **Progr PIM**：仅可编程PIM，可根据工作负载的需要在尽可能多的基于ARM的可编程内核上执行所有操作（无需运行时调度）
- **Fixed PIM**：仅固定功能PIM，可根据工作负载的需要在尽可能多的固定功能逻辑单元上执行所有操作（无需运行时调度）
- **Hetero PIM**：本文的设计

Evaluation

- 执行时间分析
  - ✓ 执行时间对比

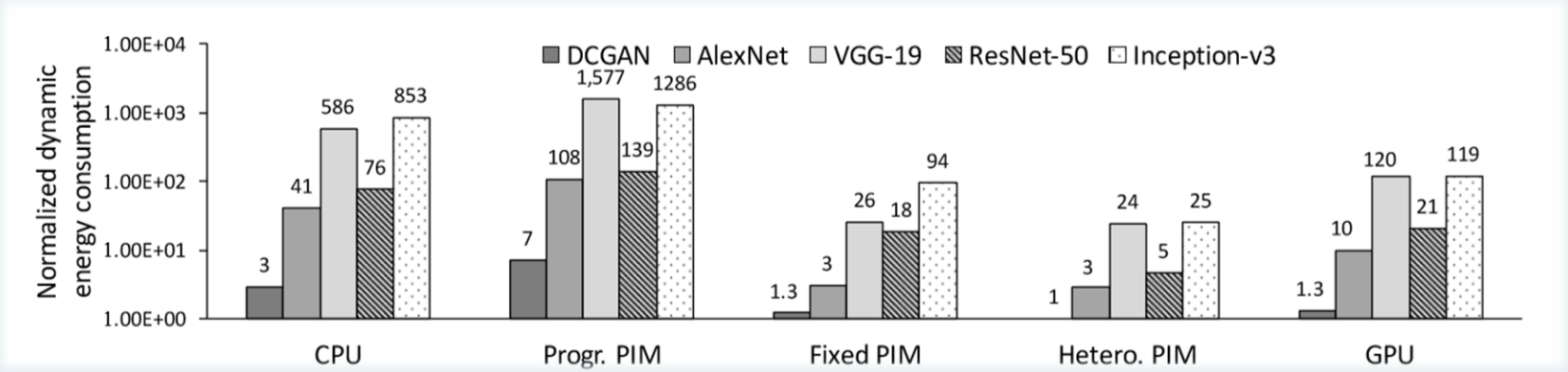


  - ✓ 执行时间分析

与使用ResNet的GPU相比，异构PIM可带来更好的性能。 通过其他训练模型，Hetero PIM可以使性能接近（低于10%）GPU。 GPU具有良好的性能，因为它具有巨大的线程级并行性。 此文的设计比其他所有配置都具有更好的性能。

Evaluation

- 能耗分析
  - ✓ 能耗对比

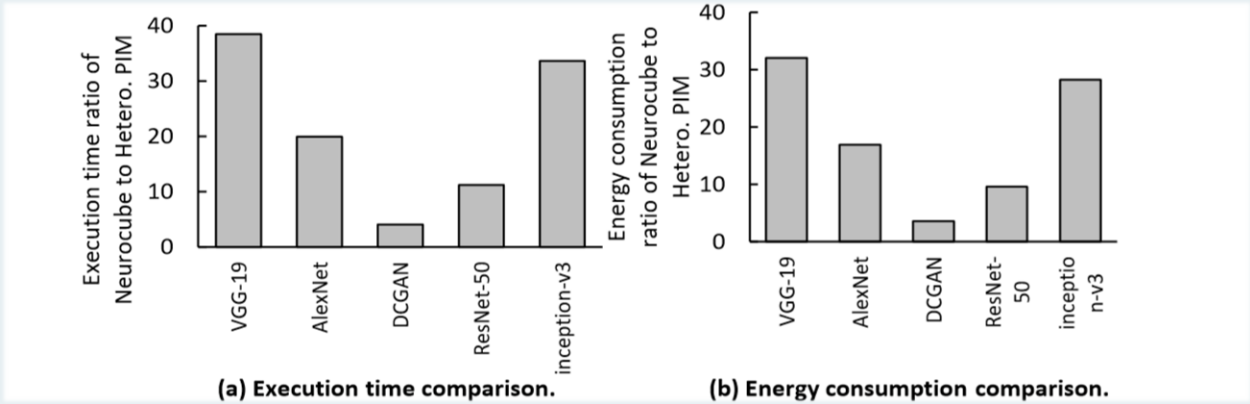


  - ✓ 能耗分析

总的来说，Hetero PIM可以在所有配置中实现最低的动态能耗。

Evaluation

- 与前人的基于PIM的神经网络加速器(Neurocube)对比
  - ✓ 对比结果



(a) Execution time comparison. (b) Energy consumption comparison.

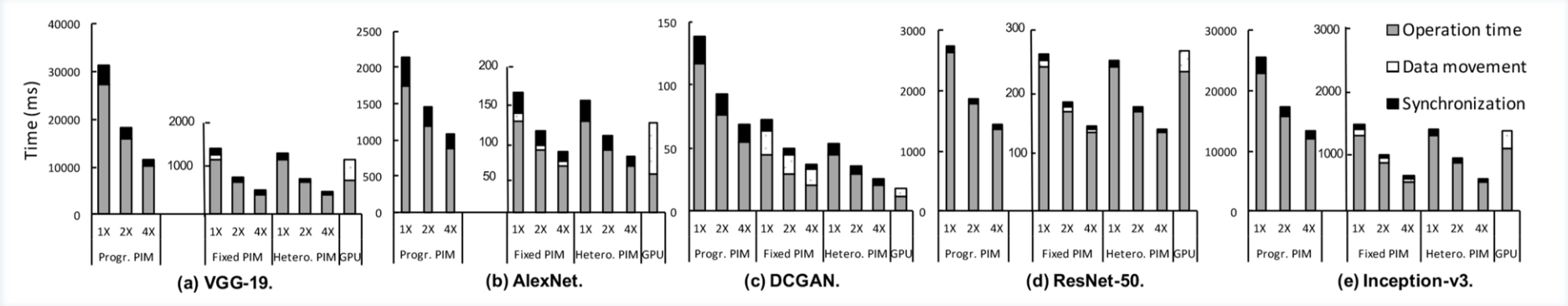
  - ✓ 对比分析

本文的工作在性能和能源效率方面优于Neurocube，原因如下：（1）Neurocube只采用可编程PIM，而本文的设计采用高能效，高度并行的固定功能PIM来加速细粒度操作；（2）本文的设计采用运行时调度，有效地优化了硬件利用率

Evaluation

- 敏感性研究
  - ✓ 频率

对固定功能的PIM和可编程PIM采用三种不同的频率：它们的原始频率（1×），它们的频率加倍（2×）和它们频率的四倍（4×）



观察到，频率越高，异构PIM的性能会优于GPU；

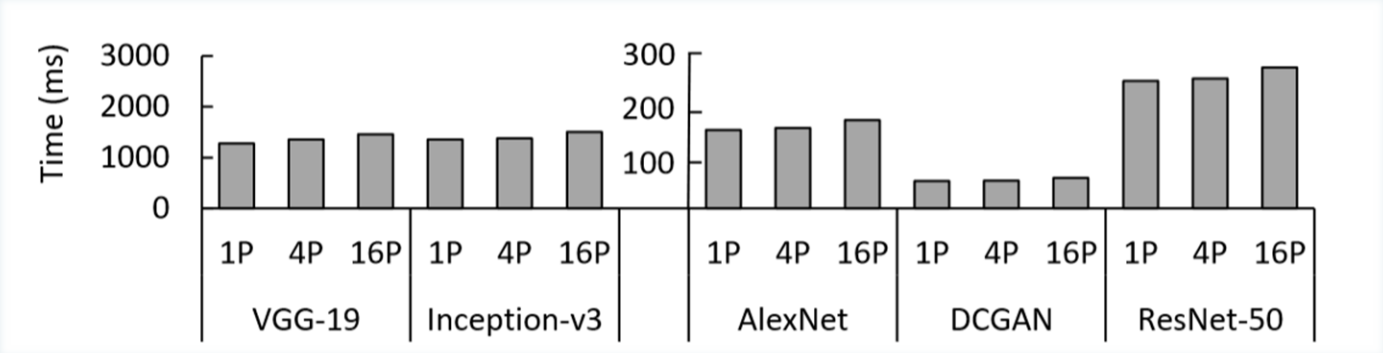
当使用更高频率时，同步和数据搬运的开销会减少。

Evaluation

- 敏感性研究

  ✓ Progr PIM 数量

为Hetero PIM采用三种不同的配置，同时保持存储器堆栈中逻辑芯片的面积不变。 将Progr PIM（ARM内核）的数量从1扩展到2到16，而逻辑芯片区域的其余部分用于实现固定PIM。 这三种配置分别标记为1P，4P和16P。



图中显示三种配置之间的性能差异相对较小，原因有两方面：（1）一台Progr PIM足以让NN模型进行调度和流水线管理；（2）在存储器堆栈的逻辑层中给定恒定区域的情况下，使用更多的Progr PIM会丢失更多的固定PIM单元。

Conclusion

- 本文提出了一种异构PIM方法的软硬件协同设计，结合了可编程PIM和固定功能PIM的功能，用于NN训练。
- 本文的设计通过各种NN训练优化工作实现了性能和能源效率的显着提高。
- 对OpenCL编程模型进行扩展，以适应PIM异构性并提高机器学习框架的程序可维护性
- 提出了一个基于NN操作的动态配置，在异构PIM上动态映射和调度NN操作的运行时系统。

Related work

- 机器学习中的内存计算

  Azarkhish和Shuichi等人采用RISC-V内核和芯片堆叠式DRAM中的流式协处理器来加速卷积网络或SGD

  Neurocube 通过在3D芯片堆叠DRAM的逻辑层中集成可编程处理元件来加速CNN推理和训练

- 内存计算用于一般应用程序

  Fujiki等人提出了一种基于ReRAM的内存处理器架构和数据并行编程框架

  Ahn等人介绍了用于并行图处理的PIM。

  Akin等人提出了一套机制，可以使用3D堆叠DRAM在内存中实现高效的数据重组

- 用于机器学习的其他加速器优化

Authors

Jiawen Liu： ： PhD student

Hengyu Zhao： PhD student

Matheus Almeida Ogleari： PhD student

Dong Li： Assistant professor in the Department of Electrical Engineering and Computer Science, University of California, Merced
Parallel Architecture, System, and Algorithm lab

Jishen Zhao： Assistant Professor in the Computer Science and Engineering Department at University of California, San Diego
architecting emerging memory technologies
computer architecture and system software
domain-specific acceleration
edge computing systems

After reading

阅读这篇论文之后，觉得本文的一个缺点在于：在神经网络算子执行时间和访存次数分析（目的是找出执行时间既长又频繁访存的操作）时，只是把两个表的索引简单相加作为全局表的索引，通过这个全局索引进行调度。我觉得这里存在误差。也就是说在算子分析时可以进行优化。

THANKS

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