## 15组演讲稿

第一页：

今天由我们小组来汇报相关论文，这篇论文题目是主要讲解的是使用DRAM进行位串行SIMD处理的框架，这篇论文的主要作者是Nastaran （来自西蒙弗雷泽大学的纳斯塔兰）和 Geraldo（来自苏黎世大学的杰拉尔多），论文提出于今年4月份在纽约举办的第26届ACM国际编程语言和操作系统架构支持会议上。

第二页：

在大数据的时代下， 大量的数据应用导致计算的高能量和延迟成本在传统的计算机架构中会使得在有限的内存通道中消耗总量达到60%以上，为了降低这些成本，我们应该将计算移至更接近数据的所在位置，以此来减少内存和处理器之前移动数据。 为了降低这种成本，内存处理（PIM）范式将计算移到更靠近数据驻留的位置。我们大体上有两种解决方案。

一、是在内存附近处理。PIM逻辑被添加到与内存相同的芯片

二、是使用内存处理，其利用存储器单元本身的操作原理，通过使单元之间能够相互作用来执行计算。

第二种方法由于使用内存的处理直接在内存阵列中运行，因此它受益于内存阵列中可用的大内部带宽和并行性，而第一种方法无法充分利用内存。

第三页：

使用内存体系结构处理的一种常见方法是使用位计算。许多广泛使用的数据密集型应用程序（如数据库、神经网络、图形分析）严重依赖于简单（如AND、OR、XOR）位运算。许多以前的设计也探索了能够执行内存位操作的DRAM。然而，一个主要的缺点阻碍了这些方案的广泛应用：它们只支持基本操作（例如布尔运算、加法），不能灵活方便地支持新的和更复杂的操作。

因此作者本文提出了一个框架SIMDRAM。希望高效并灵活的来帮助DRAM进行处理。

第四页：

内存主要由DRAM组成，DARM存储芯片内部由子阵列组成。子阵列中的单元被组织成多行和多列。一个单元由一个存取晶体管和一个存储电容组成，存储电容利用其电压电平对一个数据位进行编码。同一列中所有单元的访问晶体管连接到同一位线。类似地，同一行中所有单元的访问晶体管连接到同一字线。

DRAM中的读写操作包括三个步骤：

（1）激活。选中目标行的字线，该字线将行中的所有单元格连接到各自的位线。通过检测放大器对电压的检测和放大。将读出的数据放在行缓冲区内。

（2） RD/WR。发出读写命令，然后，内存控制器向行缓冲区内的数据发出读或写命令。

（3） 预充电。断开电容器与位线的连接，将位线电压恢复到其原始状态。

第五页：

DRAM行复制（ AAP ）：通过向同一子阵列中的行A和行B发出两个连续的激活命令，可以将行A复制到行B。

第一个激活命令将行A的内容复制到行缓冲区中。第二个激活命令将行B中的单元格连接到位线。由于感测放大器在行B时已经感测并放大了源数据，因此行B的每个单元中的数据被存储在行缓冲器中的数据（即，行A的数据）覆盖。

多数运算（ MAJ ）：通过命令TRA同时激活三行。当激活三行时，连接到每个位线的三个单元同时共享电荷，并导致位线的扰动。如果三个DRAM单元的至少两个电容器分别充电或放电，则感测放大器将位线电压放大至VDD或0。因此，TRA在每个位线上的三个DRAM单元之间产生多数运算（MAJ）。多数运算MAJ仅当其一半以上的输入为1（0）时才输出1（0）。

三输入多数运算可以表示为MAJ（A，B，C）=A·B+A·C+B·C。

第六页：

使用MAJ实现AND和OR运算：通过AP命令序列简单地将一个输入（例如C）设置为1或0来实现布尔AND和OR运算。

通过将C设置为0（即，MAJ（A，B，0）=A AND B）来计算AND运算。OR运算通过将C设置为1（即，MAJ（A，B，1）=A OR B）来计算。

SIMDRAM通过构建逻辑上完整的多数集（MAJ），与使用基本逻辑运算相比，使用MAJ and NOT进行计算需要更少的DRAM命令。

SIMDRAM框架基于以上现有的操作，实现了使用DRAM处理的框架。目标是：

（1）实现复杂操作的高效实现，（2）提供灵活的机制来支持任意用户定义操作的实现。下边将介绍SIMDRAM中的子阵列组织，描述SIMDRAM框架的概述，并解释如何将SIMDRAM集成到系统中。

第七页：

SIMDRAM使用了一种子阵列组织，它包含了执行逻辑基元（即MAJ和NOT）的额外功能。 SIMDRAM只需要对DRAM子阵列（可以同时激活三行的小行解码器）进行最小的修改就可以实现计算。

SIMDRAM将DRAM行分为三组： 数据组（D组）、控制组（C组）和计算组（B组）。

D组包含存储程序或系统数据的常规行。 C组由两个常数行组成，称为C0和C1，分别包含所有0和所有1的值。B组行（称为计算行）被指定执行按位操作。它们都连接到一个特殊的行解码器，该解码器可以使用一个地址同时激活三行（即，执行TRA）

第八页：

SIMDRAM框架包括三个关键步骤，如图所示。 该框架的前两个步骤可以使用户在DRAM中实现任何操作，第三步则根据前两步实现的命令控制DRAM内计算的执行流程。

第九页：

SIMDRAM框架中第一步的目标是构建给定操作的优化MAJ/NOT实现，将给定操作的AND/OR/NOT表示转换为优化的MAJ/NOT表示。

转换过程的第一部分简单地用MAJ原语替换AND/OR原语。每两个输入AND运算或OR运算原语被简单地替换为一个三输入MAJ原语，其中一个输入分别绑定到逻辑0或逻辑1。

转换过程的第二部分采用贪婪算法应用一系列转换，将多个MAJ原语合并为具有相同功能的MAJ原语。

第十页：

第二步将DRAM行分配给操作的输入和输出，并生成所需的DRAM命令序列来执行所需AAP/AP的优化序列。

每个SIMDRAM操作都存储为一个µ程序，该程序由SIMDRAM用于在DRAM中执行SIMDRAM操作的一系列微体系结构操作（µ操作）组成。第二步的目标是采用步骤1中生成的优化MAJ原语，并生成一个µ程序，

第十一页：

第三步执行第二步中的μ程序。程序使用bbop指令发出执行SIMDRAM操作的请求，bbop指令是允许程序与SIMDRAM框架交互的一条指令。CPU当用户程序遇到与SIMDRAM操作相关的bbop指令时，bbop指令通过执行在内存控制器中的μ程序来触发SIMDRAM操作的执行。

SIMDRAM在内存控制器中使用了一个控制单元，该单元按照μ程序的要求，透明地将AAP/AP的操作序列发送给DRAM。当μ程序完成，操作的结果将被保存在DRAM中。

第十二页：

在传统的水平数据布局中，数据元素存储在不同的DRAM行中，每个数据元素的内容从最高有效位到最低有效位（反之亦然）排列在一行中。相反，在垂直数据布局中，DRAM行仅保存多个数据元素的第i位（其中元素的数量由行的位宽度决定）。

当激活垂直数据布局组织中的单个DRAM行时，一次读取来自每个数据元素的单个数据位，从而启用DRAM位内并行计算，增加并行性，大大加快位运算速度。

为了保持与传统系统软件的兼容性，我们将常规数据存储在传统的水平布局中，并提供硬件支持以将水平ID输出数据转置到垂直布局中，用于DRAM内计算。

第十三页：

我们将分别从吞吐量，在实际应用中的性能优势以及相较于使用缓存处理架构的优势三个方面来评估SIMDRAM的性能。

第十四页：

针对16种不同的操作，分别将SIMDRAM与CPU，GPU，Ambit进行比较，根据延迟和元素大小的影响将操作分成了3类，分别对数据大小n的线性，对数，二次方操作。从右图中可以看出，SIMDRAM的吞吐量基准线，优于CPU，GPU和Ambit三者，在16通道的情况下，SIMDRAM的吞吐量达到了CPU的88倍，GPU的5.7倍。即使是1通道SIMDRAM的性能也是相同条件下的Ambit的两倍。而多核CPU和GPU由于其比特序列的二次性。仅在乘法和除法两种操作上优于SIMDRAM，综合看来，SIMDRAM拥有四者中最好的吞吐量表现。

第十五页：

在同样的16组操作中，采用了CACTI标准对能耗进行评估，从图表中看到，SIMDRAM的平均能源效率最高，分别是CPU，GPU，Ambit的257倍，31倍，2.6倍。SIMDRAM的节能主要源于三点，一、避免了从内存加载/存储数据的昂贵往返，二、利用内存设备内丰富的内存带宽，减少执行时间，三、通过优化MAJ操作实现，减少计算给定操作所需的TRA数量。

第十六页：

这里例举了几种不同的实际模型，例如kNN，VGG16等等卷积神经网络，分类算法，图像处理，来对SIMDRAM在实际应用中的性能进行评估，可以看到，即使是单通道的SIMDRAM，它的表现性能也已经是CPU的三倍，而四通道的SIMDRAM已经可以在一些模型上表现出2倍的GPU性能。SIMDRAM利用了自身的高带宽，避免了GPU的内存瓶颈，可以加速许多常用的实际应用程序。

第十七页：

在与使用缓存处理的架构比较中，SIMDRAM的延迟表现均优于使用缓存的处理架构。在使用缓存处理的实际过程中，由于其需要将数据先存储到缓存中，导致了额外的数据移动，从而达不到其理想状态，而SIMDRAM则可以直接在内存中对数据进行处理，避免了不必要的开销，实现了更高的运行效率。