arXiv:2302.01675v2 [cs.AR] 2 Nov 2023

在批量位运算存内处理中实现关系数据库分析处理

Ben Perach Ronny Ronen Shahar Kvatinsky维特比电气与计算机工程学
院以色列理工学院，以色列海法 benperach@technion.ac.il
ronny.ronen@ef.technion.ac.il shahar@ee.technion.ac.il

摘要——**批量按位存内处理（PIM）是一种新兴计算范式，它利用存储器阵列作为计算单元，已被证明有益于数据库应用。本文展示了如何在批量位运算 PIM 中高效执行分组和连接这两种以往研究尚不支持的数据库操作，从而实现关系数据库分析处理。我们扩展了 gem5模拟器，并在星型模式基准测试上评估了硬件改动。结果表明，与以往研究相比，我们的改动平均可将执行时间缩短 1.83×、能耗降低 4.31×，并将系统寿命延长 3.21×。此外，与现代先进的内存数据库 MonetDB 相比，我们实现了 4.65× 的加速比。**

索引词—**存内处理、数据库、联机分析处理、忆阻器**

一、引言

存内处理（PIM）是一种新兴的计算范式，它通过在数据所在的位置进行计算，降低数据移动带来的延迟和能耗。本文聚焦于一种特定的存内处理技术，称为 批量位运算 *PIM* [1–6], ，在这种技术中，存储器阵列也充当位向量处理单元，对其中存储的数据进行运算。以往关于批量位运算 PIM 的研究表明，数据库应用，尤其是关系数据库联机分析处理（联机分析处理）[7], 可通过批量位运算 PIM 得到显著加速 [1, 2, 4–6]。联机分析处理数据库查询会处理大量记录，通常需要汇总从多个指定记录子组中获得的信息。然而，以往研究加速的数据库操作仅限于筛选和聚合操作，未支持执行完整联机分析处理查询所需的其他数据库操作，例如分组和连接。连接操作将多个数据库关系（表）合并，从而可以查询其组合信息。分组操作将记录划分为若干子组，然后分别汇总每个子组。

本文介绍了支持关系数据库联机分析处理（OLAP）中分组和大多数连接操作的批量位运算 PIM 技术。据我们所知，这是首项利用批量位运算 PIM 支持此类操作的工作。支持连接需要大量数据移动。由于批量位运算 PIM 对数据移动的支持有限，因此通过在存内处理存储器中存储预连接关系来支持连接。预连接作为反规范化 [8] 或物化视图 [9], , 是加速查询执行的已知方法。然而，预连接会增加维护复杂度和存储开销。我们认为，预连接关系适用于批量位运算 PIM 上的联机分析处理，并展示了批量位运算 PIM 如何缓解预连接的缺点。为支持分组，我们将一种云内处理分组技术 [10]应用于批量位运算 PIM，将工作分配给主机处理器和 PIM。为高效地将这种分组技术适配到批量位运算 PIM，我们在存内处理存储器中添加了一种电路，以加速 PIM 聚合并延长存储单元寿命。

我们在 gem5 [11] 全系统模拟（包括操作系统）中实现了所提出的方法，并测量其每次的性能。我们使用星型模式基准测试（SSB） [12], 这一关系数据库分析处理基准测试来评估性能。据我们所知，尚无其他工作在批量按位处理存内计算系统上设计并评估过完整的数据库基准测试。我们将查询执行时间与 MonetDB [13],这一现代内存数据库系统进行比较；相较于 SSB 基准测试的标准版本和预连接版本，几何平均加速比分别达到 7.46× 和 4.65× 。总之，本文的贡献如下：

- 我们展示了批量位运算 PIM 如何缓解预连接关系的缺点。
- 我们针对批量位运算 PIM 改进了一种分组算法。
- 我们在存储器阵列的外围电路中加入聚合电路，以加速聚合操作；相较于先前的工作，平均加速比达到 1.83×，能效提升了 4.31×，系统寿命延长了 3.21×。
- 我们使用 gem5 对基于电阻式随机存取存储器的存内计算系统进行全系统模拟，并采用 SSB 基准测试评估了所提出的方案。结果表明，与现代内存数据库相比，在分别采用和不采用预连接关系的情况下，运行时间分别缩短了 4.65× 和 7.46× 。

本研究得到欧洲研究委员会的支持，资助来源为欧盟地平线2020研究与创新计划（资助协议编号757259）以及欧盟地平线欧洲研究与创新计划（资助协议编号101069336）。

II. 背景

A. 关系数据库与分析处理

关系数据库是一种数据模型，它将数据组织为关系（表） [7]。数据库中的每个关系都包含多条记录，可将其视为关系中的行。每个关系都有若干属性，可将其视为关系中的列；每条记录在每个属性上都有一个值。关系中的一个属性或多个属性的组合被指定为其键，用于唯一标识其记录。

数据库查询是关于数据库记录的问题，即，返回记录，或者是对满足其属性特定条件的记录执行的函数。分析处理应用于业务决策支持流程等场景，具有专用的数据库结构，并以特定类型的查询为特征 [7, 12]。此类查询通常采用选择-来源-条件-分组的形式，即查询在一个或多个关系（来源）中搜索满足特定条件（条件）的记录，并对这些记录的某个属性执行聚合（例如，求和、平均值、最大值）（选择）。通常需要根据某些属性对选中的记录进行分类，并针对每个子组执行聚合，这称为分组。

当查询涉及多个关系时，就需要对这些关系执行连接操作。连接操作根据各关系属性上的条件，将不同关系中的记录关联起来，从而能够同时检查多个关系中的属性。在分析处理过程中，连接操作可能占据 90% 以上的执行时间 [14], , 但仍保留关系分区，以维持执行的灵活性、避免数据重复并简化数据库维护 [8]。

B. 批量按位存内处理

批量位运算 PIM 利用存储器阵列及其外围电路（例如，译码器、感测放大器和电压驱动器）进行处理

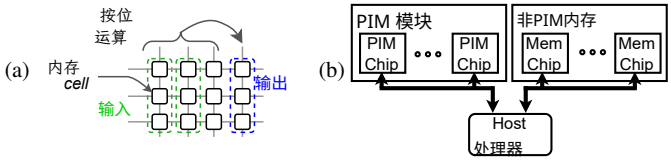


图 1：（a） 3×4 存储器交叉开关阵列执行按位列逻辑运算（例如，或非）。运算的输入和输出分别是左侧两列存储单元的值（绿色）和右侧列存储单元的值（蓝色）。按位行运算也可以用类似方式执行。（b）一个作为内存秩连接到主机的批量按位PIM模块。

。处理在存储器阵列内部进行，直接从存储单元读取输入数据，并将输出数据直接写入存储单元，从而消除数据移出存储器阵列的过程。这些存储器交叉开关支持的基本运算是逻辑运算（例如，或非 [1,3]）。由于存储器阵列具有规则结构，因此可以在阵列中的大量存储单元上并行执行这些逻辑运算。此外，许多存储器阵列可以并行工作，从而实现大规模逻辑运算，即批量按位运算。图 1a展示了使用存储器交叉开关阵列进行批量按位运算的示例。更复杂的运算（例如加法和乘法）可以通过一系列基本逻辑运算构建。动态随机存取存储器 [2, 4] 和新兴的非易失性存储器技术 [1, 3, 5, 6] 已被提出用于实现此类批量位运算 PIM。

批量按位 PIM 存储器可用作主机处理器的主存 [1, 2, 4]。除标准非PIM（例如，动态随机存取存储器）存储器秩外，该PIM 模块还配备支持PIM的存储芯片，并作为内存秩运行（图1b）。在这种情况下，主机可以使用标准加载和存储指令从PIM存储器读取数据或向其写入数据。要执行PIM计算，主机会向PIM模块发送一条名为*PIM* 请求的内存命令。PIM 请求连同地址和数据一起发送，与存储指令类似，用于指定计算、操作数和结果位置。通过限制PIM 请求只能使用和修改单个内存页内的数据来支持虚拟内存 [1]。PIM 请求的地址指定了该页。发出PIM 请求时，用户级程序会使用虚拟地址发送请求。虚拟地址通过标准转换方法转换为物理地址，并转发到相应的内存位置。

为了使页能够在内存内独立运行，每个存储芯片上的每一页都配备了一个专用控制器，称为 *PIM* 控制器。当 PIM 请求到达存储芯片时，目标页的 PIM 控制器会管理所需的基本逻辑运算序列

到该页的所有交叉开关阵列。为保持高并行度，会使用大页（例如，2MB 或 1GB），并在属于该页的所有交叉开关阵列上同时执行相同操作。

将数据库映射到批量位运算 PIM 存储器时，每个关系都有其存储关系记录的页。每条记录对应一个交叉开关行，所有记录的属性分别对齐存储在交叉阵列列上 [1,2,4,5]。若要根据某个条件筛选记录，则通过对该关系所有记录中的相同属性执行按列PIM操作来实现该条件。筛选结果为每条记录对应一个比特，用于指示该记录是否满足筛选条件。因此，读取筛选结果时，只需读取每条记录对应的一个比特，而不必读取经过筛选的属性。聚合通过并发聚合该关系所有交叉开关阵列中的相同属性来完成。随后读取每个交叉开关阵列中的聚合值，并在主机端将这些值合并。因此，进行聚合时，每个交叉开关阵列只需读取一个值，而不必读取整个相关属性



图 2：用于存储销售信息的星型模式示例。该数据库包含一个存储销售数据的事实关系（每笔销售对应一条记录），以及两个存储客户和供应商信息的维度关系。每条记录。由于读取操作减少，数据移动的大幅减少是批量按位 PIM 的主要优势；与不使用批量按位 PIM 相比，读取量最多可减少 99% [1]。

III. 支持预连接关系

连接操作需要匹配两个或更多关系中的记录，通常采用多对多或一对多的方式。由于无法假设来自不同关系的匹配记录位于同一个 PIM 交叉开关上，因此必须移动数据才能执行连接操作。此外，记录的匹配项取决于记录数据，因此需要进行数据相关移动。虽然批量位运算 PIM [3], 支持内存内部的数据移动，但尚未证明其支持虚拟内存；而且，这种移动并非数据相关。因此，它无法直接支持连接操作。另一方面，在交叉开关内部本地执行的操作，即 针对单个关系执行的操作，将受益于批量位运算 PIM [1]的高并行性。因此，我们提出在 PIM 模块中保留预连接关系，从而能够针对单个关系执行完整查询。

预连接关系可能源于反规范化 [8]或物化视图 [9], ，这两者都是已知的加速查询执行的方法，但会牺牲其他方面。这些方面包括：(1) 灵活性有限——预连接关系有时才有帮助，因为可能存在许多不同的连接操作，而选定的预连接关系可能并非查询所需。(2) 额外存储——连接结果可能大于输入关系的总和。(3) 维护复杂——由于连接操作经常会将同一数据复制到多个位置，因此更新等操作会变得更加复杂。

关于灵活性有限这一方面，我们注意到，对于星型模式——业务支持流程中常见的一种联机分析处理数据库结构， [7,12], 只有少数几种常用的连接选项。星型模式包含一个大型中央关系和多个较小的关系，分别称为事实和维度关系。图2 展示了一个星型模式示例。事实关系中的一条记录表示一个事件及其定量细节（例如，一次购买及其价格）。事实关系的某些属性是维度关系的键（即外键），用于将事件与其附加信息关联起来（例如客户信息）。查询通常要求维度关系与事实关系之间执行连接操作，并对维度键应用相等条件 [7], 即，基于维度键进行等值连接。因此，星型模式中的大多数联机分析处理查询都能从保留预连接关系中受益。维度关系将通过基于维度键的等值连接预先与事实关系连接起来，同时不会损害灵活性。例如，这种连接方式能够满足星型模式基准测试（SSB）中的所有查询 [12]。

关于额外存储方面，由于连接操作是针对维度的键进行的，且键具有唯一性，因此连接会将一条事实关系记录与一条维度关系记录相连。因此，事实关系中的记录不会重复，连接输出关系的记录数与事实关系相同。不过，事实关系中的每条记录都会添加维度信息，例如，一位客户的信息会附加到他的所有购买记录中，

算法 1： 用于在内存内值与
立即数值之间进行多路复用器选择的 PIM 控制器算法，使用内存内选择。

输入： $v_{n-1}...v_0$ - 要更新的内存内属性位
$c_{n-1}...c_0$ - 作为立即数值的更新值位
s - 内存内选择位
结果： 对于所有 i： $v_i \leftarrow c_i$ 如果 $s = 1$，否则 v_i 保持不变。

对每个 $i \in [0, .., n-1]$ **执行**
  如果 $c_i = 1$，**则**
    $InMemory(v_i \leftarrow v_i\ OR\ s)$
  else
    $InMemory(v_i \leftarrow v_i\ AND\ NOT(s))$

从而增大记录的大小。对于批量位运算 PIM，关系存储在专用页中，通常每条记录都会导致交叉开关行未被充分利用 [1]。如果在此处存储每条关系记录的更多信息，就能利用这部分未使用的行内存。因此，如果事实关系采用批量位运算 PIM，存储事实关系和维度关系的预连接关系就能利用未使用的内存，因为其记录数与事实关系相同。这样便无需额外的内存。

一般而言，预连接关系生成的记录可能比单个交叉开关行更大。在这种情况下，可以对预连接关系进行垂直分区 [1], , 将关系的属性存储在多个对齐的页上。然而，这种垂直分区会增加内存开销并降低性能，因为中间结果必须在各分区之间传输。因此，该分区应将常用属性放在同一个交叉开关中，以避免常见情况下传输中间结果。不过，对于 SSB 基准测试，预连接关系记录不会超过行大小，因此不需要进行此类分区。第V-A节仍会评估这种情况。

预连接关系通常需要复杂的维护 [8,9]。具体而言，由于数据重复，更新操作会变慢。借助批量位运算 PIM，可以通过筛选关系记录中待替换属性值来执行更新操作。随后，利用筛选结果覆盖仅目标记录的属性，也就是说，筛选结果用作选择位，供 PIM 实现的多路复用器（MUX）使用。受 [1], 启发的 PIM 多路复用器算法如算法1所述。此更新操作仅需存内处理操作，无需读取操作，几乎完全消除了数据移动。

IV. 支持分组

为支持分组操作，我们采用了一种专为云内处理设计的算法 [10]。云内处理和批量位运算 PIM [1] 支持相同的数据库基本操作：筛选和聚合。不过，批量位运算 PIM 的特性（例如处理延迟、数据检索延迟）与云内处理不同，因此其行为和参数也有所不同。

GROUP-BY 技术： 为查询筛选出所需记录后，可以通过两种方案聚合 GROUP-BY 中的每个子组。第一种方案使用存内处理操作分别聚合每个子组。每个子组还会进一步进行筛选，然后再进行聚合，这两个步骤均使用存内处理操作。我们将此方案命名为 *pim- gb*。pim-gb 对整个关系进行操作，因此其延迟与每个子组中的记录数量无关，但取决于关系中的记录数量和子组数量。尽管 PIM 聚合的延迟较低，但待聚合的子组可能很多，导致 pim-gb 延迟较高。第二种方案使用主机读取筛选结果和通过筛选的记录。首先读取每条记录，然后根据其属性值将其分配到某个子组，并由主机完成聚合。这

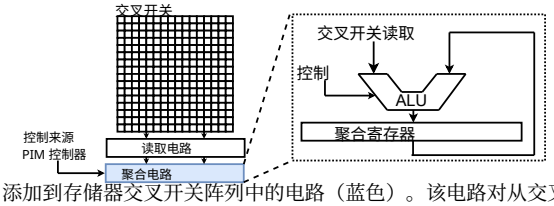


图 3：添加到存储器交叉开关阵列中的电路（蓝色）。该电路对从交叉开关阵列读取的数据进行聚合。算术逻辑单元支持求和、最小值和最大值操作。

种方案可同时处理多个子组，且除了查询指定的筛选操作外，无需执行存内处理操作。我们将此方案命名为 *host-gb*。host-gb 的延迟主要取决于所需的内存读取次数。这些读取次数取决于关系中的记录数量、所有子组中的记录总数，以及每条记录中待读取属性的大小。

在本工作中，GROUP-BY 技术利用了这样一个事实：pim-gb 和 host-gb 依赖于不同的参数。pim-gb 取决于子组的数量，与子组中的记录数量无关；而 host-gb 则恰好相反。此外，GROUP-BY 技术还利用了这样一个事实：数据库数据并非均匀分布 [15]，且 GROUP-BY 子组的大小并不相同。因此，对子组进行划分，让少数几个较大的子组由 pim-gb 聚合，而将数量众多的较小子组（其中可能有空子组）交由 host-gb 处理。pim-gb 和 host-gb 之间如何划分子组，取决于数据分布和具体的查询需求。为确定如何进行这种划分，主机会对查询选中的记录抽取一小部分样本，并利用该样本估算每个子组的大小。主机根据经验模型（本节后文将对此进行说明），决定哪些子组由存内处理进行聚合，哪些子组由主机进行聚合。这项技术最初是为云内处理提出的 [10]，我们在此将其改用于批量位运算 PIM。

加速 PIM 聚合： 先前的研究表明，批量按位 PIM 聚合操作在执行时间、功耗和存储单元耐久性方面（对于新兴非易失性存储器技术而言）代价高昂 [1]。这是因为执行聚合需要大量基本操作。因此，为了实现高效的 GROUP-BY 执行，我们在每个存储器交叉开关外围增加一个算术电路，以支持所需的聚合操作（见 [1]），如图3所示。这种方法与纯批量按位 PIM 架构有所不同，同时又与其相辅相成，旨在缓解批量按位 PIM 聚合的弱点。

算术电路接收从存储器交叉开关读取的数据。PIM 请求指定的聚合值按顺序逐个读取，并在算术电路中进行聚合。该电路采用标准 CMOS 逻辑设计，仅执行聚合所需的逻辑操作 [1]：求和、最小值和最大值，并由 PIM 控制器控制。由于交叉开关读取的位长固定为 [16]（在我们的评估中为 16 位），若要支持更长字长的聚合，就需要算术逻辑单元支持对其操作数进行移位和掩码处理。随后，使用经过修改的写入控制逻辑 [6]，将最终聚合结果从算术电路写入交叉开关。写入结果的位置由主机发送的聚合 PIM 请求指定。之后，主机可以通过标准内存读取来访问最终聚合结果。

经验建模： 如上所述，GROUP-BY 技术需要决定将哪些子组分配给 pim-gb，哪些分配给 host-gb。要进行定量决策，需要为每种方案建立延迟模型。为获得这样的模型，我们进行了

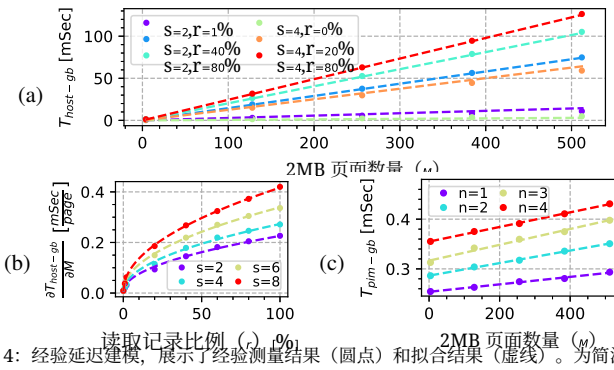


图 4：经验延迟建模，展示了经验测量结果（圆点）和拟合结果（虚线）。为简洁起见，仅展示了部分测量结果。(a) 对于不同的每条记录读取次数（s）和读取记录比例（r），$T_{host\text{-}gb}$ 与页面数量（M）的关系。(b) 对于不同的 s，$\frac{\partial T_{host\text{-}gb}}{\partial M}$ 与 r 的关系。(c) 单个子组的 $T_{pim\text{-}gb}$ 与页面数量的关系，其中聚合属性读取次数不同（n）。

延迟测量（在第V-A节所述系统上，使用合成数据库），并将每种方案的测量结果拟合为数学表达式（如下文所述）。这一拟合过程可由数据库管理软件 [17]自动完成。在下列所有表达式中，M 表示存储该关系所需的 2MB 页面数。

host-gb 延迟 $T_{host\text{-}gb}$ 包括使用存内处理筛选相关子组、读取筛选结果（位向量）以及读取选中的记录。筛选 PIM 操作延迟主要由筛选结果读取次数决定 [1]，，其延迟仅取决于关系大小。读取选中的记录时，延迟取决于选中的记录数量以及每条记录所需的读取次数。我们用 r 表示选中的记录数占关系中记录总数的比例，用 s 表示每条记录所需的读取次数（由子组标识符和聚合属性组成）。因此，host-gb 取决于三个参数：关系大小（M）、待读取记录的比例（r）以及读取次数（s）。图 4a 显示，对于特定的 r 和 s 值，$T_{host\text{-}gb}$与M呈线性关系。给定 s 时，斜率 $\frac{\partial T_{host\text{-}gb}}{\partial M}$ 随 r 变化的情况如图 4b 所示。给定 s时，斜率呈现 $a\sqrt{r}+b$形式的关系，其中 a 和 b 为常数。由于交叉开关阵列的读取长度固定（在我们的评估中为 16 位），s 只能取少数几个离散值。因此，我们将 a 和 b 表示为关于 s取值的查找表，得到以下表达式：

$$T_{host\text{-}gb}(M,s,r) = M \cdot \left(a(s) \cdot \sqrt{r} + b(s)\right). \quad (1)$$

单个子组的 pim-gb 延迟（$T_{pim\text{-}gb}$）取决于关系大小（M）以及从单个交叉开关中读取聚合属性所需的读取次数（标记为 n）；它与聚合记录的数量无关。我们测量了不同关系大小和操作数大小下的聚合延迟。结果如图 4c所示。聚合延迟随关系大小呈线性变化，其系数取决于 n。与上文的 s 一样，n 可以取若干离散值。

因此，系数 $\frac{\partial T_{pim\text{-}gb}}{\partial M}$ 和 $T_{pim\text{-}gb,0}$（自由系数）表示为 n 值的查找表。单个子组存内计算聚合的拟合延迟为：

$$T_{pim\text{-}gb}(M,n) = M \cdot \frac{\partial T_{pim\text{-}gb}}{\partial M}(n) + T_{pim\text{-}gb,0}(n). \quad (2)$$

为执行该操作，我们需要确定分配给 pim-gb id 的子组数量。我们将该数量标记为 k。根据定义，这些是 k 最大的子组。host-gb 中剩余记录数与记录总数的比率取决于 k 和数据分布，并标记为 $r(k)$。函数 $r(\cdot)$ 通过前面提到的记录采样进行估计。因此，总分组延迟 T_{gb} 为：

单个 RRAM 存内计算模块			
总容量	32GB	大页大小	2MB
存储器秩	1	近存计算芯片	8
交叉阵列行	1024	交叉阵列列	512
交叉开关读取	16 位	批量按位逻辑周期	30 纳秒 [5]
交叉开关读写能耗	0.84\6.9 皮焦耳/位 [5]	批量按位逻辑能耗	81.6 fJ/bit [20]
单个聚合电路功耗	25.4 uW	单个存内处理控制器功耗	126 微瓦 [1]
评估系统			
处理器核心	6 个核心，X86 3.6GHz	主存	32GB动态随机存取存储器 DDR4-2400
一级缓存	私有，16KB，64B块，4路	L2缓存	共享，2MB，64B块，16路
一致性协议	MESI	RRAM 存内处理模块	1

表 I：架构和系统配置

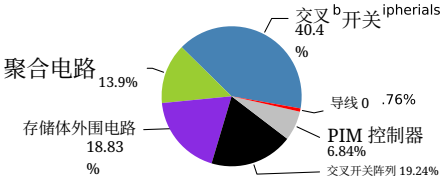


图 5：PIM芯片面积细分

$$T_{gb}(M,n,s,k_{MAX},k,r(\cdot)) = k \cdot T_{pim\text{-}gb}(M,n) + (1-\delta_{k,k_{MAX}}) \cdot T_{host\text{-}gb}(M,s,r(k)), \quad (3)$$

其中 k_{MAX}是子组总数，而 $\delta_{i,j}$是克罗内克尔 delta。该 $(1-\delta_{k,k_{MAX}})$项表示，如果使用 pim-gb 聚合所有子组，则不会执行 host-gb。利用（3），从查询和数据库定义中获取 M、n、s 和 k_{MAX}，并估计 $r(\cdot)$，我们可以找到 PIM 聚合子组的最优数量 k，以实现最低的 T_{gb}。

V. 评估

A. 方法

模拟：为评估我们提出的方法，我们开发了一个 gem5 模拟 [11]，基于 [1,18] 提供的系统，地址为 [19]，。该系统采用 RRAM 按位批量处理存内计算，以全系统模式运行（运行 Linux 内核），并包含 gem5 的 Ruby 缓存系统。我们采用了 [18]中的一致性解决方案和范围一致性模型。系统参数列于表I。

基准测试：我们运行了 SSB 基准测试 [12]，比例因子为十（$SF=10$），以评估所提方法的性能。SSB 是一种联机分析处理基准测试，包含 13 个查询，分为四个查询组。我们采用了 [1] 中的编译流程。使用离线内部编译器，将 SSB SQL 查询编译为 C++ 代码，然后使用 gcc 编译。查询执行将关系记录分为四个相等的组（以页粒度划分），并将每组分配给一个线程。对于分组操作，第IV 节所述的记录采样和估算只执行一次，并由四个线程共享。采样在单个 2MB 页上进行，即32K 条记录。

SSB 基准测试中的关系存储为单个预连接关系，这是事实关系与维度关系根据维度键进行等值连接的结果。我们按照 [15] 使用非均匀数据填充该关系。必要时，我们会调整查询参数，使查询的选择率（筛选后的记录数占总记录数的比例）与原始均匀数据 [12] 中的相近。预连接关系包含原始关系中的所有属性，但不包括

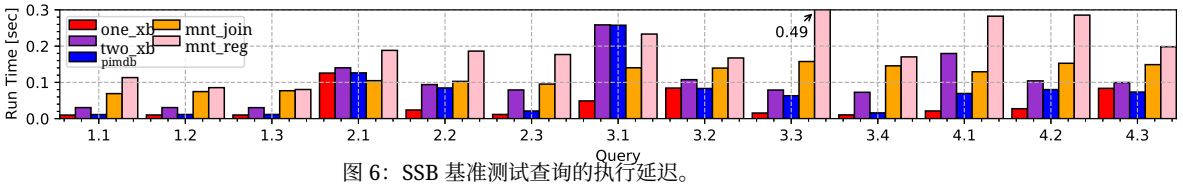


图 6：SSB 基准测试查询的执行延迟。

Q	选择率	总计子组	子组样本中	PIM 聚合子群		
				one-xb	two-xb	PIMDB
1.1	2.3e-2	1	-	1	1	1
1.2	6.6e-4	1	-	1	1	1
1.3	8.4e-5	1	-	1	1	1
2.1	1.2e-2	280	121	4	0	0
2.2	1.6e-3	56	33	56	0	0
2.3	2e-4	7	4	7	0	7
3.1	3.4e-2	150	150	150	0	0
3.2	1.3e-3	600	27	27	0	0
3.3	4.7e-5	24	2	24	0	0
3.4	6.6e-7	4	0	4	0	4
4.1	2e-2	35	35	35	0	35
4.2	2.3e-3	50	29	50	0	0
4.3	9.1e-5	800	3	3	0	0

表 II：查询摘要：**选择率**（筛选后的记录数占总记录数的比例）、根据查询和数据库详情得出的潜在子组总数（**子组总数**）、抽样中为分组估计找到的子组数（**样本中的子组**），以及**PIM 聚合子组**的数量。Q1.1–3 无需分组，并使用 PIM 执行单次聚合。

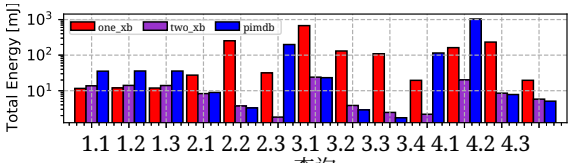


图 7：SSB 基准测试的 PIM 存储器能耗。

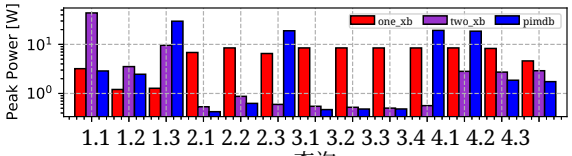


图 8：SSB 基准测试中单个 PIM 芯片的峰值功率。芯片面积构成，其中聚合电路占据13.9% 的芯片面积。聚合电路的面积开销相对较高，因为每个交叉开关中都添加了该电路。为了估算查询执行期间 PIM 模块的功率和能耗，我们从 [1] 中获取 PIM 模块各部分的功率和能耗参数（汇总见表I），并通过 gem5 模拟将它们相加。

客户 和 供应商 维度关系。这些属性是较长的文本，SSB 查询不会使用它们。由于不包含这些属性，我们可以让预连接关系中的单条记录适配单个交叉开关行。因此，第 III 节中所述的垂直分区并非必需。

我们以两个版本运行了该方案。第一个版本名为*one-xb*，将预连接关系中的一条记录存放在单个交叉开关行中。第二个版本名为 *two-xb*，实现第III节所述的垂直分区，将记录拆分到两个交叉开关阵列中，并评估记录过大、无法适配单个交叉开关的情况。事实关系的所有属性都放在一个交叉开关阵列中；维度关系的属性则放在另一个交叉开关阵列中。由于需要通过主机在分区之间额外传输结果，垂直分区会改变 PIM 聚合的延迟 [1]。对于 SSB 中的所有 GROUP-BY 操作，子组标识符属性来自维度关系，而聚合属性来自事实关系，因此 two-xb 属于最坏情况的分区方式。要在存内处理（PIM）中执行聚合，每个 子组的筛选结果都要在 PIM 聚合之前在页之间传输，这种最坏情况的分区方式会造成显著开销。如果事先知道常见的子组标识符，就可以将最常见的标识符与这些属性放在同一个交叉开关阵列中

来自事实关系，从而减少所需的数据移动。从这个角度来看，one-xb 评估了此类分区的最佳情况。我们重复了第 IV节中针对 two-xb 版本所述的 pim-gb 和 host-gb 经验建模。

面积与功耗： 为了评估第IV节所述新增聚合电路的面积与功耗，我们使用 Verilog 设计了该电路并进行综合，然后使用 Synopsys Design Compiler 和 Cadence Innovus，基于台积电 28 nm 互补金属氧化物半导体工艺确定其面积与功耗。对于 PIM 模块的芯片面积，我们修改了 NVSim [21]，根据 [1] 加入 PIM 控制器，并为每个交叉开关加入我们的聚合电路。PIM 模块由八个芯片组成，每个芯片占据 $346mm^2$。图 5 展示了 PIM 模块

对比： 我们将本方案与 MonetDB 进行比较。MonetDB 是一款现代化的联机分析处理 [13],内存数据库管理系统，运行在一台配备两颗英特尔至强处理器（每颗具有 16 个 2.1 GHz 核心）、总计 256 GB DDR4-2400 内存并运行 Ubuntu 的服务器上。我们运行了两个版本的 MonetDB。第一个版本名为*mnt-reg*，采用原始的 SSB 关系模式。第二个版本名为 *mnt-join*，采用了与 one-xb 中相同的预连接关系。MonetDB 报告的延迟仅包含执行时间，不包括 SQL 解析和优化所需的时间。

我们还将自己的聚合电路方案与作为本系统基础的 PIMDB [1], 进行了比较。我们扩展了 PIMDB，使其像 one-xb 一样使用预连接关系和分组。PIMDB 仅在其 PIM 聚合方面有所不同：它完全通过批量按位逻辑来执行聚合，而我们的方案使用第IV节中的聚合电路。我们按照第IV节所述，对 PIMDB 的 PIM 聚合进行经验建模。PIMDB 与 one-xb 的其他方面均相同。

B. 结果

执行延迟： 图6 展示了 SSB 查询的执行延迟。查询执行摘要列于表II。*One-xb* 实现了最佳执行延迟，其几何平均加速比分别为 7.46× 和 4.65×，相较于 mnt-reg 和 mnt-join。如果将预连接关系垂直分区到两个交叉开关阵列中（*two-xb*），与 one-xb 相比，几何平均延迟增加了 3.39×，但仍比 mnt-join 1.37× 更快。与 one-xb 相比，PIMDB 的延迟增加了 1.83×，这表明我们的聚合电路带来了延迟改善。聚合延迟的改善使 GROUP-BY 技术能够分配更多子组进行 PIM 聚合，如表II所示。

对于选择率最高的 GROUP-BY 查询，尤其是 Q2.1、Q3.1 和 Q4.1，存内处理方案相比 MonetDB 的加速比较低，甚至出现性能下降。由于从存内处理存储器读取数据时，会跨越一个页中的多个交叉开关阵列 [1]，而一个记录位于页中的单个交叉开关内，因此读取单条记录会将内存中的大量记录带到主机，在我们的系统中为 32 条记录。这种读取放大抵消了由

存内处理筛选所实现的读取量减少。对于存内处理而言，读取量减少很少甚至没有，因此 MonetDB 的读取缩减技术（例如筛选、索引）以及 MonteDB 所采用的更强系统在此类情况下更具优势。请注意，当子组数量足够少时，即使选择率较高，也可以执行纯存内处理分组，并实现加速；Q3.1 和 Q4.1 中的 one-xb 就是如此。

能量与功率： PIM 模块消耗的总能量以及 PIM 芯片的峰值功率如图所示。7 和8。对于所有查询，PIM 模块的能耗都低于 1J，且每个 PIM 芯片的峰值功率都低于 44W。当 PIMDB 使用 PIM 聚合（Q1.1–1.3、Q2.3、Q3.4 和 Q4.1）时，其几何平均能耗4.31× 高于 one-xb，且其峰值功率2.92× 也高于 one-xb。不过，在其他查询中情况恰好相反，因为 one-xb 使用 PIM 聚合，而 PIMDB 不使用。由于批量位运算 PIM 会执行宽操作，并发跨越许多交叉开关行，因此与 PIM 模块中的读取操作相比，它能在更短时间内消耗更多能量。因此，PIMDB 通过不执行 PIM 聚合，以能耗换取延迟。对于*two-xb*，在 Q1.1–1.3 中，更多交叉开关阵列上的 PIM 操作和读取操作会提高峰值功率。在其他查询中，纯 host-gb 则能保持较低的能耗和峰值功率。

耐久性： 图9 展示了连续运行每个查询时（100% 占空比），单个存储单元在十年内所需的耐久性。图中的数值假设采用了磨损均衡技术，并且操作均匀分布在每一行的存储单元上 [1]。对于每个查询，存储单元的使用量为单个交叉开关行经历的最大操作次数除以该行中的存储单元数量。阻变存储器的报告耐久性 [22] 显示为 10^{12} 每单元写入次数，足以满足所有方案十年的需求。比较 one-xb 和 PIMDB，one-xb 中使用聚合电路并不总能提高耐久性。这是因为 one-xb 可能会执行 PIM 聚合，而 PIMDB 执行的是纯主机端聚合。在 Q2.3 和 Q4.1 上，两者都执行 PIM 聚合，PIMDB 的延迟更长，因此需要更多时间来完成操作。one-xb 可以通过停顿达到相同效果。在 Q1.1–1.3 和 Q3.4 上，one-xb 和 PIMDB 执行的 PIM 聚合都很少，因此延迟相近；one-xb 的寿命在几何平均值上 3.21× 更优。对于*two-xb*，每个存储单元的写入次数甚至更少

因为存内处理操作分布在两页上。

VI. 相关工作

先前关于批量位运算 PIM 的研究 [1,2,4–6] 提出了将其用于数据库应用，特别是联机分析处理应用。然而，这些研究并未展示如何执行完整的数据库基准测试，因为它们关注的是体系结构和硬件，而非算法。具体而言，这些研究既未说明如何执行分组操作，也未说明如何处理连接操作。

PushdownDB [10] 展示了如何利用现有的云端数据库原语执行分组操作。这些原语与批量位运算 PIM 的基本数据库操作相似：筛选和聚合。因此，可以在这两个领域之间借鉴相关技术。

VII. 结论

本文首次展示了批量位运算 PIM 如何执行完整的数据库基准测试，重点关注联机分析处理数据库查询。我们说明了如何通过添加聚合电路并建模批量位运算 PIM 操作的延迟，将分组操作适配到批量位运算 PIM。我们还指出，在批量位运算 PIM 中使用预连接关系对于联机分析处理应用是高效的。批量位运算 PIM 可以大幅加速执行

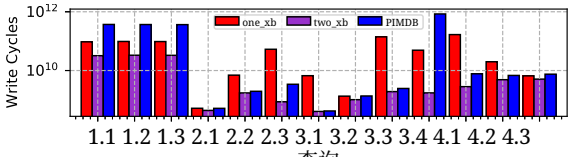


图 9：SSB 基准测试查询所需的存储单元耐久性，假设每个查询连续执行十年。对于单个关系，因此存储常用的预连接关系可以加速常见情况。

我们使用分组和预连接关系，在 SSB 基准测试上评估了批量位运算 PIM 的性能，并将其与现代内存数据库 MonetDB 进行了比较。与 MonetDB 的标准版和预连接版 SSB 相比，批量位运算 PIM 分别实现了 7.46× 和 4.65× 的几何平均加速比。

参考文献

[1] B. Perach 等，“PIMDB：通过数据库分析理解存内批量位运算”，2022年。 [在线]。网址：https://arxiv.org/abs/2203 10486 。[2] V. Seshadri 等，“AMBIT：利用商用 DRAM 技术实现批量位运算的存内加速器”，发表于 MICRO-50，2017年。[3] M. S. Q. Truong 等，“RACER：使用电阻式存储器进行位流水线处理”，发表于 MICRO-54，2021年。[4] N. Hajinazar 等，“SIMDRAM：使用 DRAM 实现位串行 SIMD 处理的框架”，发表于 ASPLOS-26，2021年。[5] N. Talati 等，“CONCEPT：用于在 RRAM 中高效执行存储和存内处理操作的面向列的存储控制器”，IEEE Micro，第39卷，第1期，第33–43页，2019年。[6] S. Li 等，“Pinatubo：面向新兴非易失性存储器的批量位运算存内处理体系结构”，发表于 DAC-53，2016年。[7] R. Kimball 和 M. Ross，《数据仓库工具箱：维度建模权威指南》，第3版。Wiley Publishing，2013年。[8] S. K. Shin 和 G. L. Sanders，“从数据仓库检索数据的反规范化策略”，决策支持系统，第42卷，第1期，第267–282页，2006年10月。[9] R. Chirkova 和 J. Yang，“物化视图”，Foundations and Trends® in Databases，第4卷，第4期，第295–405页，2012年。[10] X. Yu 等，“PushdownDB：利用 S3 计算加速数据库管理系统”，发表于 ICDE-36，2020年。[11] N. Binkert 等，“gem5模拟器”，SIGARCH 计算机体系结构新闻，第39卷，第2期，第1–7页，2011年8月。[12] P. O’Neil 等，《星型模式基准测试与增强型事实表索引》。柏林，海德堡：Springer-Verlag，2009年，第237–252页。[13] S. Idreos 等，“MonetDB：列式数据库体系结构的二十年研究”，IEEE 数据工程简报，第35卷，第1期，第40–45页，2012年。[14] M. Dreseler 等，“量化 TPC-H 瓶颈及其优化”，VLDB基金会会刊，第13卷，第8期，第1206–1220页，2020年4月。[15] T. Rabl 等，“星型模式基准测试的变体：测试数据倾斜对查询性能的影响”，发表于 ICPE-4，2013年。[16] C. Xu 等，“克服交叉阵列电阻式存储器架构的挑战”，发表于 HPCA-21，2015年。[17] B. Dageville 等，“Oracle 10g 中的自动 SQL 调优”，发表于 VLDB-30。VLDB基金会，2004年。[18] B. Perach 等，“批量位运算存内处理的一致性”，发表于 HPCA-29，2023年。[19] B. Perach，“gem5 批量位运算 PIM 一致性。” [在线]。网址：https://github.com/benperach/gem5 bulkbitwise PIM consistency [20] N. Talati 等，“使用忆阻器辅助逻辑（MAGIC）在忆阻存储器中进行逻辑设计”，IEEE 纳米技术汇刊，第15卷，第4期，第635–650页，2016年。[21] X. Dong 等，“NVSim：面向新兴非易失性存储器的电路级性能、能耗和面积模型”，IEEE 集成电路与系统计算机辅助设计汇刊，第31卷，第7期，第994–1007页，2012年。[22] F. Zahoor 等，“阻变随机存取存储器（RRAM）：材料、开关机制、性能、多级存储单元（MLC）存储、建模与应用综述”，Nanoscale Research Letters，第15卷，2020年4月。