Performance modeling on DaVinci AI core 精读报告

1 背景

由于达芬奇架构的特点以前的性能建模工作在达芬奇架构上无法使用，并且对于达芬奇架构的一些特性还未知。所以需要设计一系列 benchmark 来揭示达芬奇架构的特性，并设计一个性能模型来预测程序的执行时间。该论文设计 benchmark 展示达芬奇架构的特性，并设计了一个预测程序执行时间的模型 Verrocchio。

2 micro-benchmark

首先设计 benchmark 来确定争用源和运行时的行为，然后用不同争用比率的 benchmark 来量化争用时的带宽共享。

2.1 对常规硬件单元的 benchmark

测试 core 内部的硬件单元的性能，单核 benchmark 测试后进行双核测试。

结论：

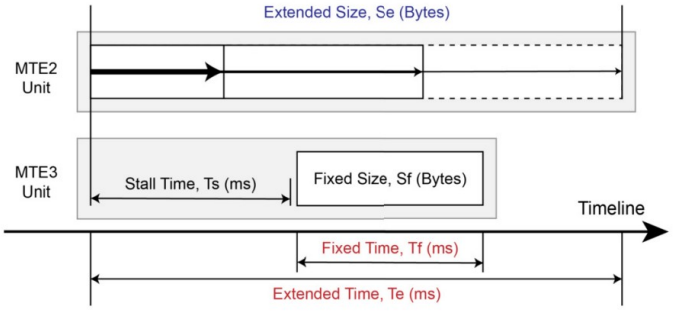
- core 内的硬件单元只需要 core 的资源，不会因为总线的争用在双核执行时导致性能下降，所以双核的性能正好是单核性能的两倍。
- 单核测试和双核测试时 kernel 启动时间相差不大，由于达芬奇 core 没有内核间的通信机制所以假设，310 上两个达芬奇 core 同步启动

2.2 总线争用源与运行时行为 benchmark

2.2.1 benchmark 设计

设计 benchmark 来观察双核执行时总线的争用源和运行时行为，这些争用是由 MTE2 和 MTE3 引起的。

MTE2 先处理 Se Bytes 的数据，然后空闲直到结束。MTE3 先 stall Ts(ms)，然后处理 Sf 数据。在达芬奇核心上在前一个单元和后一个单元的指令之间插入 dummy Scalar instruction。达芬奇 core 上标量 PSQ 必须等待标量指令结束才能继续执行下一条指令。所以 dummy 指令会阻塞标量 PSQ，并使标量 PSQ 空闲。空闲一段时间后

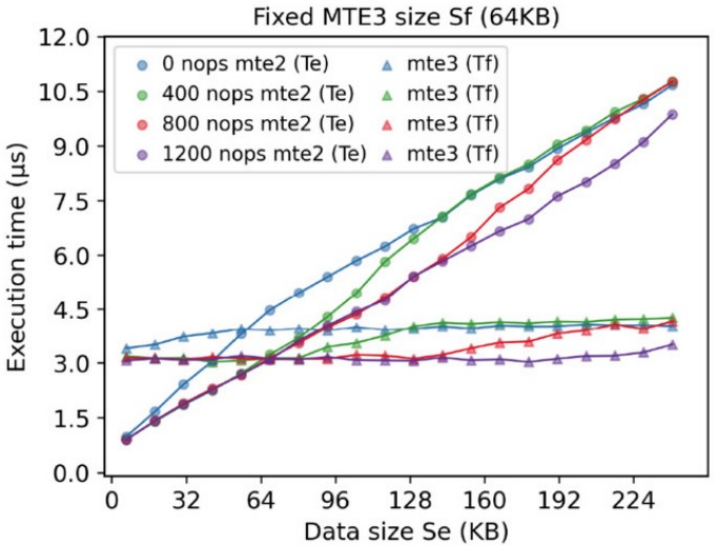


MTE3 开始执行指令与 MTE2 发生冲突，然后测量两个单元的时间 Te 和 Tf，并随着 Se 的增加而变化。不同 Te 的值将两个执行单元分为三种不同阶段：争用前，争用时，争用后（应该是指结束争用时的状态）。

- Te < Ts，争用前，MTE2 与 MTE3 未发生争用。
- Te > Ts 且 Te < Ts + Tf，争用时。
- Te > Ts + Tf，争用后。

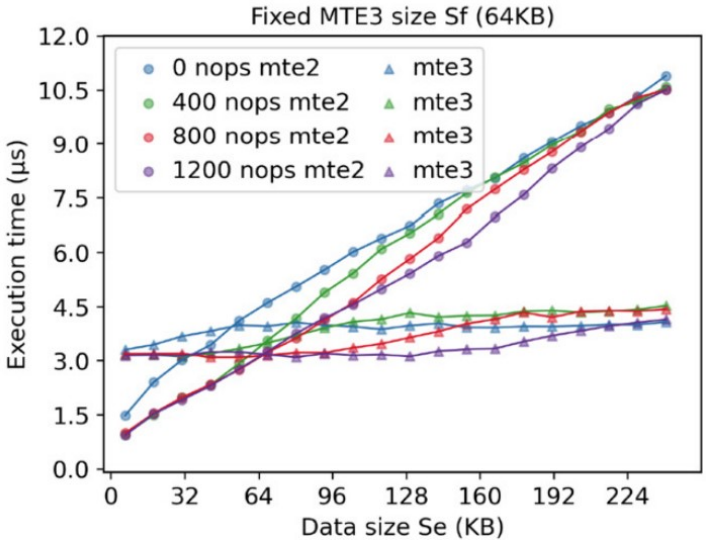
2.2.2 争用源和运行时行为

下图 MTE 单元运行时带宽发生变化即发生争用。MTE2 从外部存储读数据而 MTE3 向外部存储写数据，发生对总线的争用。

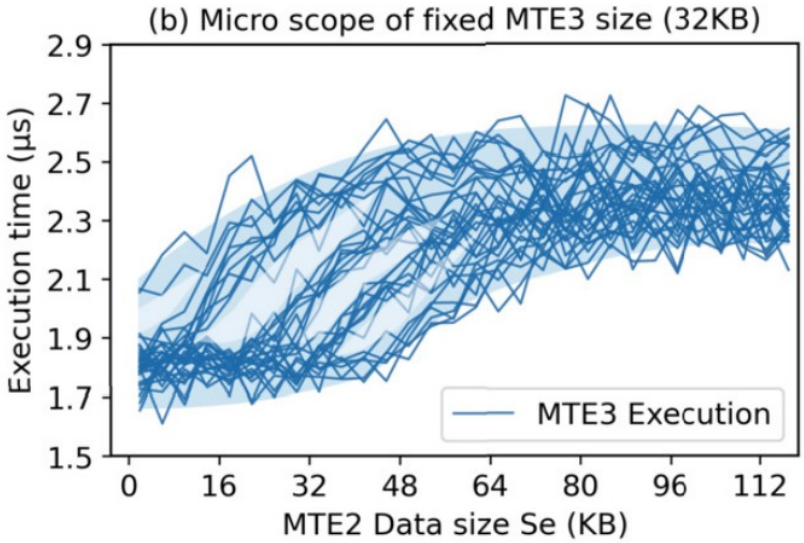


以蓝色线（0 nops）为例，程序一开始就发生争用。Se 小于 64KB 时 MTE2 属于第二类，即在争用状态结束执。S2 大于 64KB 的情况下 MTE2 属于第三类，争用结束后 MTE2 继续执行（无争用），此时的带宽与未发生争用时相同。

绿色线（400 nops）前一段没有争用，中间发生争用斜率增加，后一段争用结束回归正常。红色线（800 nops）前一段无争用，后一段发生争用。紫色线（1200 nops）MTE3 的 nops 指令太多，无争用。



上图是测试两个 core 之间争用的 benchmark 结果。比较单核和双核执行的结果，执行时间、变化趋势和斜率相似，这表明它们激活了相同的竞争源。因此，断言 Ascend 310 处理器总线带宽的竞争源是图 6(b) 和 (c) 的公共部分，即核外互连总线



上图是以较小的步长增加 Ts 即 nop 指令的数量。均匀增加 Ts 的时间，争用开始的时间（第二段起始位置）应该沿着横轴均匀分布，但是图中会聚集在某些点处。执行模式显示，MTE 单元在执行后并不立即启动数据传输，并且在执行后立即形成争用，这表明数据是以块或段的形式传输的。所以推测在块传输之前，互连总线必须管理并决定如何根据每次执行的单元数量来共享带宽。

2.3 争用发生时的带宽共享 benchmark

实验方法是在前面的 benchmark 基础上修改，增加处理的数据量并使用最小二乘法计算争用时的带宽，并且分别在一个 core 和两个核心上测试。一共测试了 6 种可能发生争用的情况。最终得到结论，每个操作平等地共享互联总线带宽，互连总线是半双工总线，写操作和读操作可以同时执行，但相互影响。

3 性能模型

模型将每条指令看作一个事件，所有事件 $E_{u,n}$ 分为三类：

- 普通的计算或内存操作
- set_flag
- wait_flag

整个程序的终止时间可以看作最后一个事件结束的时间。对于 set_flag 和 wait_flag 操作会按照不同的方式来更新其终止时间。

为每个事件 $E_{u,n}$ 分配一个三元组 (u_a, u_b, r) 来分类，其中 u_a, u_b 与 set_flag 和 wait_flag 有关，r 是其对应的信号量寄存器。

3.1 计算或内存操作

对于这类事件直接令 $E_{u,n} = (0, 0, 0)$，即这类事件与二进制信号量无关。首先计算该事件的执行时间来量化该事件占用核 u 的时间，公式如下：

$$P_{u,n} = \frac{D_{s_{u,n}}}{Th_u} + Init$$

根据 FIFO 的规则，该事件开始的时间为 u 上前一个时间结束的时间。所以得到该事件结束的时间为：

$$T_{u,n} = T_{u,n-1} + P_{u,n}$$

3.2 set_flag 操作

对于 set_flag 操作 $E_{u,n}$，设置 $E_{u,n} > 0$。例如 set_MTE1_M 事件转化为 $E_{MTE1,3} = (MTE1, M, 1)$，即 MTE1 结束工作，Cube Unit 终止 wait_flag 操作并对寄存器 1 进行下一步操作。因为 set_flag 操作不会暂停或完成任何工作负载，所以终止事件

$$T_{u,n} = T_{u,n-1}$$

在性能建模之前，需要创建并维护一个数组 $S_{E_{u,n}}$，store the Set Flag operations with the subscript of the 3-tuple (u_a, u_b, r)。$S_{E_{u,n}}$ 的定义如下：

$$S_{E_{u_1,n_1}} = S_{E_{u_2,n_2}} = S_{(u_a,u_b,r)} = \langle (u_1,n_1),(u_2,n_2),\ldots \rangle$$
$$\forall E_{u,n} > (0,0,0), n_i < n_j, \forall i < j$$

$S_{E_{u,n}}$ 按照操作的序号 n 排列。

3.3 wait_flag 操作

对于 wait_flag 操作设置 $E_{u,n} < 0$，图中的 wait_MTE1_M 被转换成 $E_{M,1} = (-MTE1, -M, -1)$。

同样维护一个数组 $W_{E_{u,n}}$ 来存储二元组 (u,n)：

$$W_{E_{u_1,n_1}} = W_{E_{u_2,n_2}} = W_{(-u_a,-u_b,-r)} = \langle (u_1,n_1),(u_2,n_2),\ldots \rangle$$
$$\forall E_{u,n} < (0,0,0), n_i < n_j, \forall i < j$$

一个 set_flag 事件 $E_{u,n}$ 一定存在一个对应的 wait_flag 操作 $E_{u',n'}$，且 $E_{u,n} = E_{u',n'}$。数组 S 和 W 用来确定 set 和 wait 操作对。

3.4 带宽争用更新

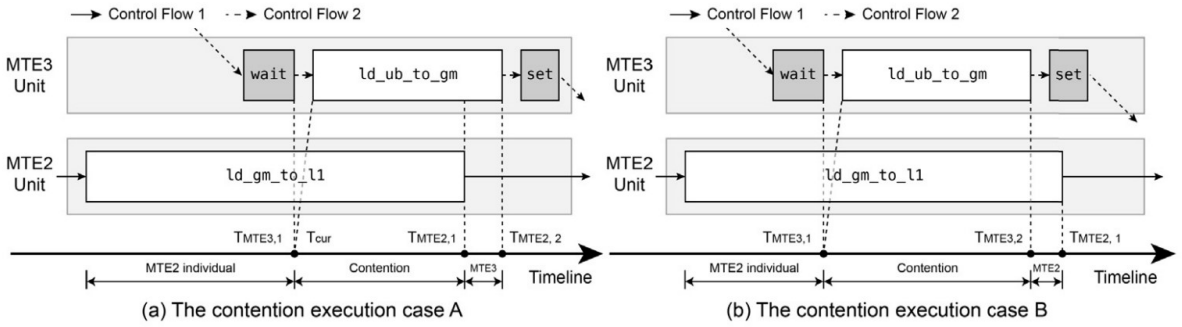


Fig. 10. Examples of Verrocchio contention execution.

模型 V 会在一个事件结束时检查是否存在争用出现和结束，然后更新相关事件的结束时间。例如图中 $T_{MTE3,1}$ 时刻，MTE3 开始执行 ld_ub_to_gm 指令，争用发生。

当 MTE2 单元执行完 ld_gm_to_l1 指令后，争用结束，MTE3 单元的带宽改变导致该事件结束事件改变。

计算步骤如下：

$$Ds'_{u,n} = Ds_{u,n} \times \frac{T_{u,n} - T_{cur}}{P_{u,n}}$$

$Ds'_{u,n}$ 是争用结束后 $E_{u,n}$ 还需要处理的数据，T_{cur} 是当前时间也是触发新带宽争用状态的事件的结束时间。例如图 a 中 $T_{cur} = T_{MTE3,1}$，wait 事件结束争用开始。所以 ld_gm_to_l1 新的结束时间如下：

$$T'_{u,n} = T_{cur} + Ds'_{u,n}/Th'_u$$

3.5 多核执行性能建模

根据 benchmark 的结果，达芬奇硬件单元分为两类：一类依赖独立的核心内部的资源、一类依赖互连总线并导致争用。第一类在多核执行时带宽保持不变，总带宽成比例增加。

$$Th_{single} = C_{single} \qquad Th_{total} = N \times Th_{single}$$

N 为达芬奇 core 的数量。第二类按照 benchmark 的结论，所有 core 平均共享总带宽：

$$Th_{single} = \frac{Th_{total}}{N} \qquad Th_{total} = C_{total}$$

下面需要设计预测普通计算或内存操作事件的执行事件。由于达芬奇架构采用 SIMD 模型，所以假设程序会被平均分布到各个核心，如下图所示。

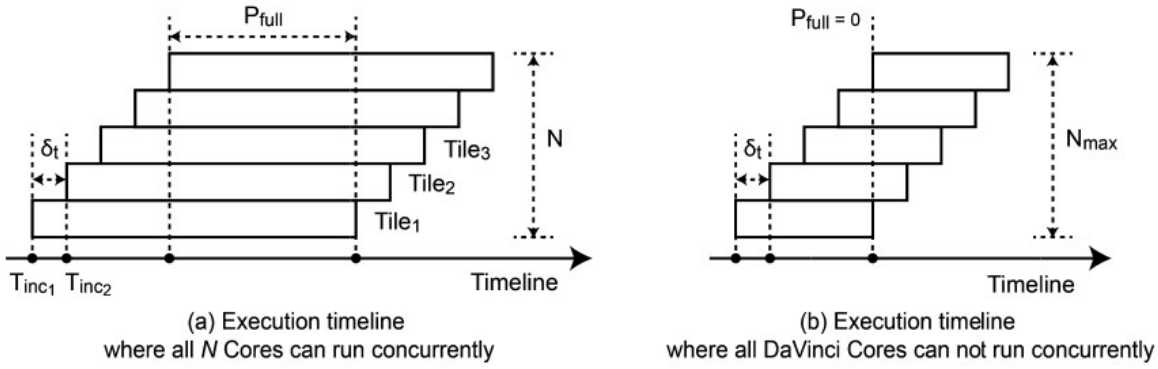


Fig. 11. Execution timeline of $E_{u,n}$, which is tiled and submitted to multiple DaVinci Cores.

为了模拟在不同 core 中事件开始时间的差异引入 δ_t，表示在所有 core 中两个最近

事件时间之间的平均间隔，且可以在 benchmark 中测量。

图 a 中总的执行时间为：

$$P_{u,n} = 2(N-1)\delta_t + P_{full}$$

图 b 中总的执行时间为：

$$P_{u,n} = 2(N_{max}-1)\delta_t$$

为了计算时间总时间还需要算出 P_{full} 和 N_{max}。对于 a 中的情况可以列出如下等式：

$$2Th_{single}\delta_t \times \sum_{i=1}^{N-1} i + N \times Th_{single}P_{full} = N \times Ds_{single}$$
$$\Longrightarrow P_{full} = \frac{Ds_{single} - (N-1)Th_{single}\delta_t}{Th_{single}}$$

令 $P_{full} = 0$ 可以得到 b 的公式：

$$P_{full} = \frac{Ds_{single} - (N_{max}-1)Th_{single}\delta_t}{Th_{single}}$$
$$= 0 \Longrightarrow N_{max} = \frac{Ds_{single}}{Th_{single}\delta_t} + 1$$

对于第二类单元，同样的思路可以得到如下公式：

$$2Th_{total}\delta_t \times (N-1) + Th_{total}P_{full} = N \times Ds_{single}$$
$$P_{full} = \frac{NDs_{single} - 2(N-1)Th_{total}\delta_t}{Th_{total}}$$
$$N_{max} = \frac{2Th_{total}\delta_t}{2Th_{total}\delta_t - Ds_{single}}$$

在🄵腾 310 上只有两个 core，N=2、$\delta_t = 61ns$ 是 kernel 启动时间的 2.5% 可以忽略。所以在🄵腾 310 上可以化简如下。

- 第一类：

$$P_{u,n} = P_{full} = \frac{Ds_{single}}{Th_{single}}$$

- 第二类:

$$P_{u,n} = N \times \frac{Ds_{signle}}{Th_{total}}$$

4 Evaluation

首先用基本的 kernel 来评估模型的准确性，然后对比在模型的辅助下调优后矩阵乘法的性能与 CANN 的性能。

4.1 准确性评估

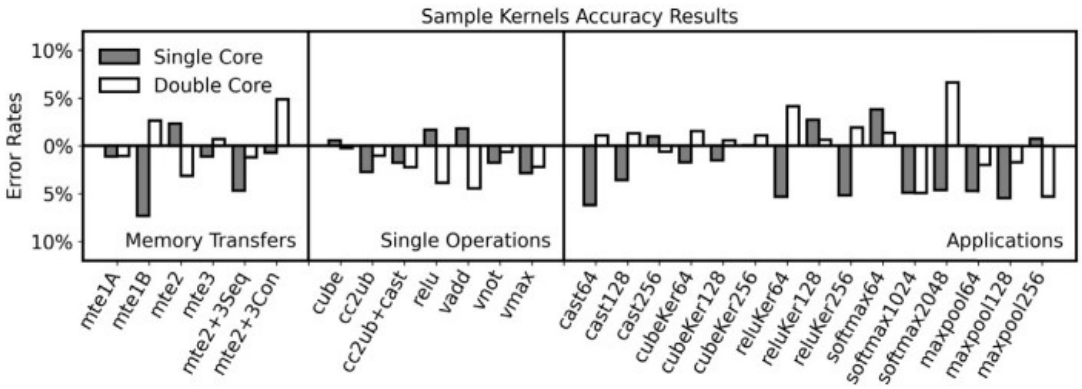


Fig. 12. The Verrocchio error rate results for sample kernels.

4.2 矩阵乘法优化

4.2.1 优化矩阵乘法和 tiling 参数选择

可以发现在一些形状如 $W \times 2W \times 2W$ 相比 CANN 提升不是很明显，但对于一些不规则的形状如 $2W \times W \times 3W$ 提升很多。同时对于小形状的调优效果不是很好有时出现低于 CANN 性能的情况，但对于大形状的矩阵乘法效果普遍较好。

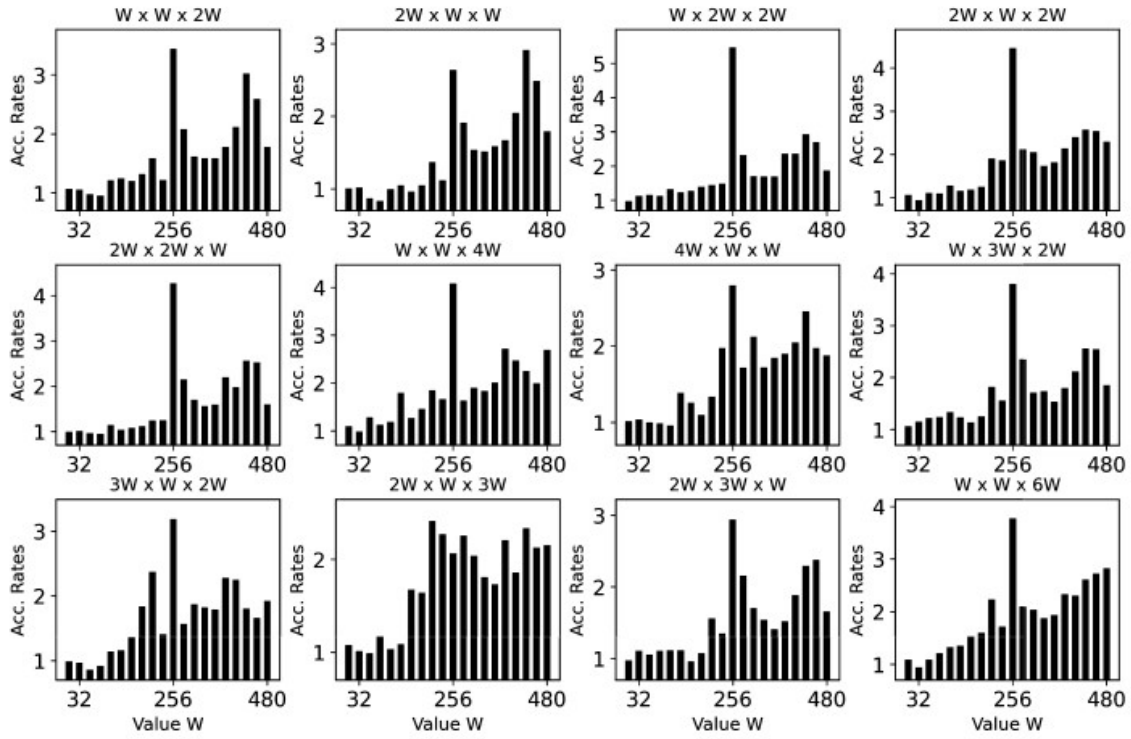


Fig. 15. Accelerations of the improved matrix multiplication compared with the CANN operators.