

工业控制领域

Cordic算法

在现代工业控制应用场景中存在大量坐标变换和超越函数的计算，Cordic算法凭借其硬件友好的特性，成为工业控制领域实现高效实时运算的基石算法。但是在通用处理器中实现Cordic算法面临着性能瓶颈问题，本文采用隼瞻科技ArchitStudio™工具，为Cordic算法定制RISC-V拓展指令实现硬件加速，最终在少量硬件开销的前提下获得27倍的性能收益，为领域专用的处理器架构的快速探索提供了可能性。

一、背景介绍

在工业自动化的核心场景中，如永磁同步电机（PMSM）的矢量控制（FOC）、工业机器人的实时轨迹规划、并网逆变器的高精度锁相以及多轴伺服驱动系统，都需要在微秒级的时间内完成大量三角函数、反三角函数及坐标变换运算。这些运算的实时性与精度直接决定了设备的能效、响应速度与控制稳定性。

传统通用处理器在处理这类超越函数时往往依赖软件数学库，其延迟与功耗难以满足高性能工业控制的需求。而CORDIC（坐标旋转数字计算机）算法凭借其独特的“移位-相加”迭代结构，无需硬件乘法器即可高精度计算三角函数、双曲函数、对数及开方等，成为工业控制领域实现高效实时运算的基石性算法。其硬件友好的特性，使得它能够以极低的资源开销，在FPGA、ASIC及专用微控制器中实现，为工业设备的小型化、高效化与高可靠性提供了关键支撑。

二、问题与挑战

CORDIC算法的核心流程简洁而优雅。以计算余弦和正弦值的“旋转模式”为例，其预先定义一系列微小角度 $\theta_i = \arctan(2^{-i})$ ，通过旋转迭代的方式逼近目标角度 θ 。每次迭代仅需要进行以下操作：

- 根据当前剩余角度 z_i 的符号，决定旋转方向 $d_i = \text{sign}(Z_i)$
- 执行坐标更新： $X_{i+1} = X_i - d_i * (y_i * 2^{-i})$ $Y_{i+1} = Y_i + d_i * (X_i * 2^{-i})$
- 更新剩余角度： $Z_{i+1} = Z_i - d_i * \theta_i$

CORDIC算法核心代码如下：

```
1 int32_t cordic(int32_t theta_fixed) {
2     int32_t x = FIXED_K, y = 0, z = theta_fixed;
3     for (int i = 0; i < ITERATIONS; ++i) {
4         int32_t d = (z >= 0) ? 1 : -1;
5         int32_t x_new = x - d * (y >> i);
6         int32_t y_new = y + d * (x >> i);
7         int32_t z_new = z - d * cordic_table[i];
8         x = x_new;
9         y = y_new;
10        z = z_new;
11    }
12    return x;
13 }
```

通常经过N次迭代后，即可得到结果。相比泰勒级数展开需要一系列乘法、阶乘和多项式求值等复杂操作，CORDIC算法将计算简化为简单的加法、位移和少量查表的规整操作，用极低的硬件成本换取了可接受的精度和速度，完美匹配了硬件设计的底层逻辑。

尽管算法本身硬件友好，但当在通用处理器架构上以纯软件库形式实现时，却面临显著的效率瓶颈：

- 指令开销严重：每次迭代过程中的条件判断、查表（读取 θ_i ）、可变长位移及加减法，在通用ISA中需要分解为多条指令。完成一次16位精度的Cordic运算通常需要执行200-400条指令，其中绝大部分消耗在循环控制、分支、内存访问等非核心计算上。
- 内存访问延迟：预计算的 θ_i 表需存放于内存或缓存中，频繁的查表操作引入不确定的访问延迟，破坏工业控制所苛求的确定性实时响应。
- 难以流水线化：软件循环中的数据依赖和分支跳转，严重阻碍了处理器深流水线效率的发挥，无法实现高吞吐量的数据流处理。

这些问题导致在高动态性能的伺服驱动或高速机器人应用中，软件CORDIC可能占比20-30%的CPU时间，成为控制系统性能提升的瓶颈。

三、解决方案

RISC-V指令集架构的可扩展性为解决上述瓶颈提供了优雅的途径。通过定义领域专用（DSA）的CORDIC自定义指令，可以将整个算法固化到专用的硬件加速单元中，实现“单指令，多周期计算”的硬件加速范式。

1) 指令集架构设计

通过设计一条简洁的自定义指令来封装CORDIC操作：

- `ecordic vrd, rs1`
- 功能：旋转模式。vrd包括输入坐标x, y以及剩余角度z, rs1为第N次迭代，计算旋转后的向量，完成一次迭代
- 输出：结果向量存入vrd

2) 专用硬件微架构

一条自定义指令背后，对应着一个高度优化的专用数据通路：

- 固化迭代流水线：硬件内部包含固定的硬连线实现的移位器、加法器和角度常数，一条指令即可完成一次迭代的所有操作
- 零开销控制逻辑：方向判断由当前的最高位（符号位）直接控制，无需分支跳转指令。循环计数由硬件状态机隐式管理
- 片上高带宽存储：角度表被固化为硬连线常数，访问延迟为零周期
- 与CPU流水线集成：该加速单元可作为CPU的一个专用功能单元。当指令译码后，操作数被送入此单元，CPU可继续执行没有数据依赖的后续指令

通过拓展自定义指令的方式虽然可以显著地解决算法运行的硬件瓶颈，提升芯片的整体能效，但是工程师却面临着大量复杂和繁重的工作。处理器是一个软硬件协同的复杂系统，为了实现自定义指令，工程师需要聚焦在RTL层级的硬件实现、验证，同时需要为自定义指令提供新的配套软件工具链，包括编译器、仿真器、调试器等，这导致领域专用架构（DSA）处理器落地难度较大。

为此，隼瞻科技提供一套智能自动化的设计工具ArchitStudio™，将工程师从繁重的底层RTL代码、编译器的编写中解放，隼瞻科技定义了一套处理器架构设计语言RISCAL，工程师可通过类C的RISCAL语言设计处理器指令集架构，ArchitStudio™工具可自动生成对应的可综合的RTL代码以及编译器、汇编器和IDE插件等全套工具链。基于生成的IA、CA仿真器、编译器和性能分析器Profiler，工程师可将编译后的应用代码放在仿真器上运行，并获得详细性能分析报告，从而指导工程师重新编写RISCAL模型优化处理器的架构。同时，基于生成的可综合RTL代码，可通过逻辑综合查看面积、时序、功耗等信息，工程师可通过修改RISCAL模型进一步优化处理器的微架构，最终实现处理器架构的快速探索。

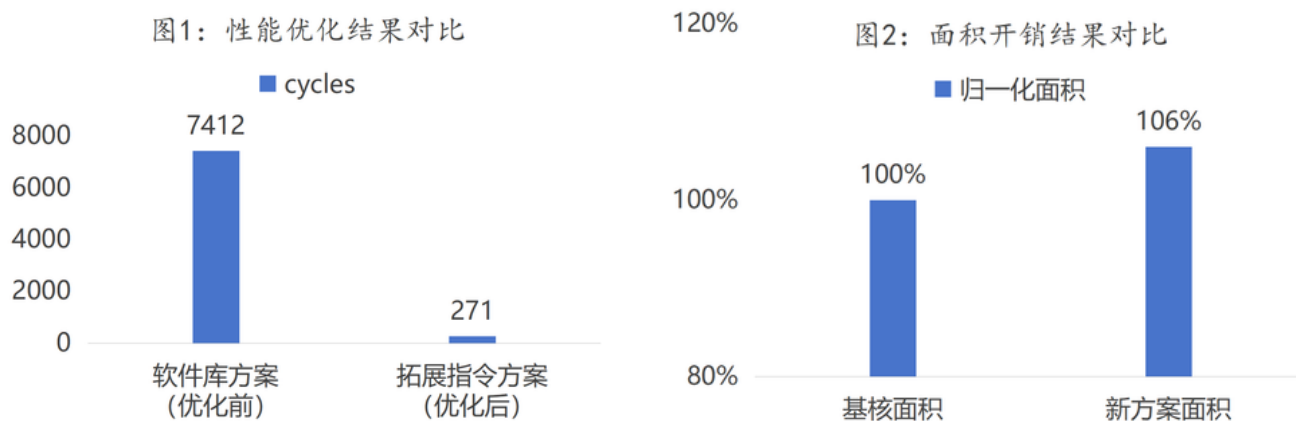
以CORDIC指令为例，RISCAL语言建模如下：

```
1 Instruction ecordic (ev_op vrd, gpr_op rs1){
2   Binary {7'b1000010 ++ rs1 ++ 5'b10101 ++ 4'b1010 ++ vrd ++ 7'b1011011 }
3   Syntax { "ecordic" <+> vrd <+> ", " <+> rs1 }
4   Semantics {
5     SInt32 index = rs1;
6     Vec<SInt32, 4> result = vrd.as(Vec<SInt32,4>);
7     Vec<SInt32, 4> result_new;
8     SInt32 x;
9     SInt32 y;
10    SInt32 z;
11    x = result[0];
12    y = result[1];
13    z = result[2];
14    Vec<SInt32, 20> cordic_table = [51471, 30385, 16054, 8149, 4090, 2045,
15                                   1023, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1, 0, 0, 0];
16    SInt32 table_val;
17    Stage (wice::vex .. wice::vwb in evALU){
18      table_val = cordic_table[index];
19      result_new[0] = (z >= 32'x0s) ? (x - (y >>> index)):(x + (y >>>
20                                     index));
21      result_new[1] = (z >= 32'x0s) ? (y+ (x >>>index)):(y-(x>>>index));
22      result_new[2] = (z >= 32'x0s) ? (z - table_val) : (z + table_val);
23      result_new[3] = 0;
24    }
25    Stage (wice::wwb in evALU){
26      vrd = result_new;
27    }
28 }
```

四、结论

在隼瞻科技M130处理器内核上，计算一组sinx、cosx、tanx函数的不同实现方式的性能差异：

指标	软件库实现	自定义指令实现	收益倍数
Cycles	7421	271	27.3倍性能收益
功耗	高	低	根本性改善
面积开销（归一化）	+0 Gates	+6% Gates	较小面积开销
实时性	差（受缓存、分支影响）	优秀（固定延迟）	根本性改善



在该案例中，通过为RISC-V处理器添加CORDIC自定义指令及对应的极小硬件单元，我们成功地将一个关键但耗能的领域计算负载，从灵活的通用处理器卸载至高效的专用硬件，使得主CPU从繁重的超越三角函数计算中解脱，专注于更上层的控制逻辑、通信和调度任务，提高系统整体性能。这不仅是性能数量的提升，更是计算范式从“通用软件”到“领域专用硬件”的质变。在实际应用中，芯片系统负载着复杂的领域专用算法，工程师可采用ArchitStudio™对算法的热点函数进行性能分析，结合不同热点函数的计算瓶颈和面积开销，灵活的为RISC-V处理器拓展指令，最终实现以最少的面积开销收获最佳的性能提升和功耗优化，形成垂类领域最优的处理器架构方案，ArchitStudio™的智能化敏捷开发能力为上述架构快速探索提供了可能性。

此案例清晰地展示了领域专用架构（DSA）的核心价值：针对特定领域的关键计算模式（如CORDIC之于工业控制），通过软硬件协同设计，在指令集架构层面进行定制，能够实现数量级的性能、能效和实时性提升。在工业控制这个对成本、功耗和可靠性极度敏感，同时又对算力要求日益增长的市场，基于RISC-V的可扩展性进行DSA设计，正成为打造下一代智能、高效工业装备的核心技术路径。