

高性能复分析辅助作图器设计报告

致理-信计 11 唐睿柏 新雅 21/计 21 张凯文

2024 年 6 月 2 日

1 项目设计背景

在分析学中，往往需要研究方程的解所具有的性质。对于某一具体的方程，现有的数学工具并非都能将方程的解表示为初等形式，更多时候我们可以使用数值方法计算方程解的分布。

例如，图 1 展示了多项式方程 $z^5 + iz^2 - 1 = 0$ 解的分布情况，对于一个复平面坐标 z ，方程左侧的值越接近 0，在图中标注的颜色越深。

现有的数值工具，如 numpy, matlab, wolframalpha 等提供了丰富的接口和高效的实现，但在一些大规模数值计算中，软件实现的性能还有广阔的开发空间，如果部分数值功能采用硬件实现，那么整个软硬件协同实现在性能提升上将具有极大的潜力。

2 问题与目标

2.1 原始问题

对于定义在区域 $D \subset \mathbb{C}$ 上的解析函数 $f(z)$ ，作出方程 $f(z) = 0$ 解的分布图

2.2 问题归约

引理 1： $f(z)$ 是 D 上的解析函数，则 $\forall z_0 \in D, \exists \delta > 0, \forall z \in B(z_0, \delta)$

$$f(z) = \sum_{n=0}^N a_n (z - z_0)^n + o((z - z_0)^N)$$

引理 2： $f(z)$ 是紧集 K 上的解析函数，若 $f(z)$ 有无穷个零点，则

$$f(z) \equiv 0$$

引理证明见 [3]。一般来说，为作出方程 $f(z) = 0$ 解的分布图，可以将定义域划分为若干更小的区域，在每个区域选取一个中心点 z_i ，若有 $|f(z_i)| < \epsilon$ 则该区域中可能存在零点。

根据上述引理，我们可以将所有解析函数归约为局部的多项式进行处理，即选取展开式中的主要部分，忽略掉展开式中小量的部分。其合理性在于 $\forall \epsilon > 0$ ，若区域 $B(z_0, r)$ 中心点 $|f(z_0)| < \epsilon/2$ ，则 $\exists \delta > 0, s.t. \forall z \in B(z_0, \delta)$

$$\left| \sum_{n=0}^N a_n (z - z_0)^n \right| < \epsilon (1 + \delta^N)$$

证明由 $f(z)$ 在 z_0 处的连续性、绝对值的三角不等式以及小量的定义即得。

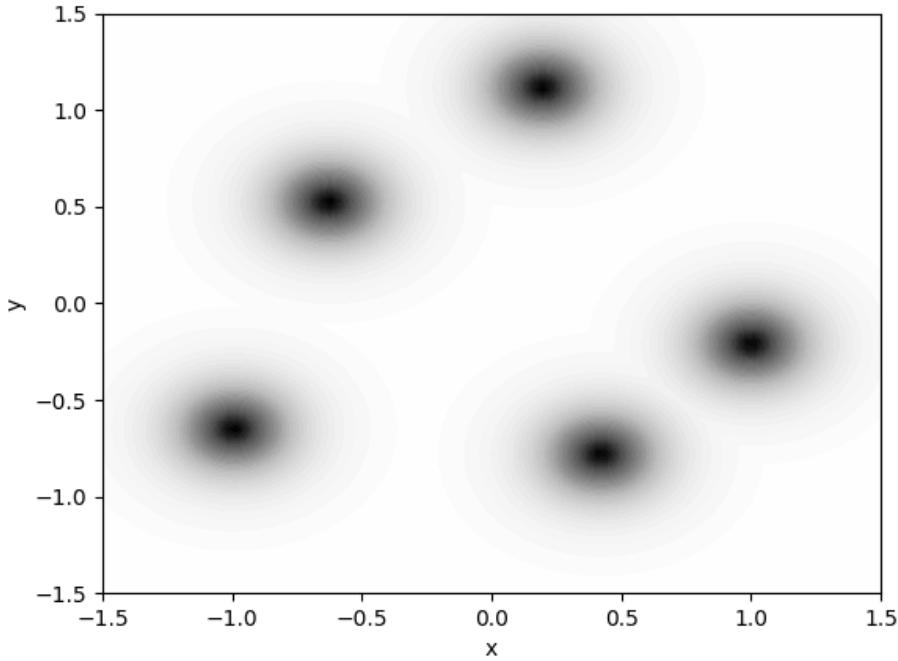


Figure 1: 方程 $z^5 + iz^2 - 1 = 0$ 解的分布图

另一方面, 如果 $f(z)$ 在 $\overline{B(z_0, \delta)}$ 中有无穷个零点, 由引理 2, 其恒等于 0, 否则只有有限个零点, 与多项式的性质相似。

综上, 我们的设计只需要考虑多项式的情形:

对于复多项式 $P(z) = a_n z^n + a_{n-1} z^{n-1} + \dots + a_0$, 从数值上计算出它的 n 个复根

2.3 高性能目标

我们希望能尽量快速地处理大规模的请求, 即对于一批 N 个多项式 (通常 N 在 $10^6 \sim 10^9$ 级别), 我们的设计需要能快速计算出这些多项式的根。使用 C/C++ 语言作为 benchmark 性能参考。

3 项目设计的理论基础

3.1 带原点位移的 QR 分解迭代法 [4]

对于多项式 $P(z) = z^n + a_{n-1} z^{n-1} + \dots + a_0$, 其伴随矩阵的特征值与多项式的根一一对应

$$\begin{pmatrix} -a_{n-1} & -a_{n-2} & \dots & -a_1 & -a_0 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}$$

求解矩阵的特征值所使用的伪代码如 Algorithm 1 所示, 算法输出的 n 个特征值也即原多项式的 n 个根。算法时间复杂度与矩阵大小 n , 迭代次数 ITERATION_TIMES 以及 QR 分解复杂度有关。一

般而言，迭代次数越多，计算结果越精准，算法采用原点位移方法也是为了加速迭代的收敛速度，但为了兼顾性能与结果，迭代次数需要根据实际情况取一个恰当的值。

Algorithm 1 带原点位移的 QR 分解迭代法

Input: $A \in M_n(\mathbb{C})$ (输入矩阵，下标从 0 开始，矩阵元素 $a_{i,j}$)

Output: $eig[0...n - 1]$ (输出特征值)

```

1:  $m \leftarrow n$ 
2: while  $m \geq 2$  do
3:   for  $i = 0, 1, \dots, \text{ITERATION\_TIMES}$  do
4:      $s \leftarrow a_{m-1,m-1}$ 
5:      $A \leftarrow A - sI_m$ 
6:      $(Q, R) \leftarrow \text{QR\_DECOMP}(A)$ 
7:      $A \leftarrow RQ + sI_m$ 
8:   end for
9:    $eig[m - 1] \leftarrow a_{m-1,m-1}$ 
10:   $m \leftarrow m - 1$ 
11:   $A \leftarrow A[0...m - 1][0...m - 1]$ 
12: end while
13:  $eig[0] \leftarrow a_{0,0}$ 
14: return  $eig[0...n - 1]$ 

```

3.2 Givens Rotation 计算 QR 分解 [4]

Givens Rotation 是一种旋转变换，具有如下形式

$$\begin{pmatrix} c & s \\ -\bar{s} & \bar{c} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sqrt{|a|^2 + |b|^2} \\ 0 \end{pmatrix}$$

其中

$$c = \frac{\bar{a}}{\sqrt{|a|^2 + |b|^2}}, \quad s = \frac{\bar{b}}{\sqrt{|a|^2 + |b|^2}}$$

矩阵的 QR 分解是指分解 $A = QR$ ，其中 Q 是酉矩阵， R 是上三角矩阵

在我们的问题中，矩阵 A 非常特殊。初始矩阵形如上 Hessenberg 矩阵（如下图），而无论经过 Algorithm 1（内层循环）多少次迭代，矩阵 A 总是保持 Hessenberg 矩阵的形式不变

$$\begin{pmatrix} \times & \times & \dots & \times & \times \\ \color{red}{\times} & \times & \dots & \times & \times \\ & \color{red}{\times} & \dots & \times & \times \\ & & \ddots & \times & \times \\ & & & \color{red}{\times} & \times \end{pmatrix}$$

利用 Givens Rotation，我们可以将矩阵对角线下的元素按顺序置 0（如下图），蓝色表示 Givens Rotation 操作后数值可能发生改变的位置。

$$\begin{pmatrix} c_1 & s_1 \\ -\bar{s}_1 & \bar{c}_1 \\ & 1 \\ & \ddots \\ & & 1 \end{pmatrix}
 \begin{pmatrix} \times & \times & \dots & \times & \times \\ \textcolor{red}{\times} & \times & \dots & \times & \times \\ \textcolor{red}{\times} & \dots & \times & \times \\ \ddots & & \times & \times \\ & & & \textcolor{red}{\times} & \times \end{pmatrix}
 = \begin{pmatrix} \textcolor{blue}{\times} & \textcolor{blue}{\times} & \dots & \textcolor{blue}{\times} & \times \\ 0 & \textcolor{blue}{\times} & \dots & \textcolor{blue}{\times} & \times \\ \textcolor{red}{\times} & \dots & \times & \times \\ \ddots & & \times & \times \\ & & & \textcolor{red}{\times} & \times \end{pmatrix}$$

$$\begin{pmatrix} 1 & & & & \\ & c_2 & s_2 & & \\ & -\bar{s}_2 & \bar{c}_2 & & \\ & \ddots & & & \\ & & & & 1 \end{pmatrix}
 \begin{pmatrix} \times & \times & \dots & \times & \times \\ \times & \dots & \times & \times \\ \textcolor{red}{\times} & \dots & \times & \times \\ \ddots & & \times & \times \\ & & & \textcolor{red}{\times} & \times \end{pmatrix}
 = \begin{pmatrix} \times & \times & \dots & \times & \times \\ \textcolor{blue}{\times} & \dots & \textcolor{blue}{\times} & \times & \times \\ 0 & \dots & \textcolor{blue}{\times} & \times & \times \\ \ddots & & \times & \times \\ & & & \textcolor{red}{\times} & \times \end{pmatrix}$$

假设每一步进行 Givens Rotation 变换左乘的酉矩阵为 Q_i , 则有

$$(Q_{n-1} \dots Q_2 Q_1) A = R$$

将 Q_i 移到等式右侧即得 QR 分解。

但事实上, 每一次迭代都有下面一步

$$A \leftarrow RQ = (Q_{n-1} \dots Q_2 Q_1) A (Q_1^H Q_2^H \dots Q_{n-1}^H)$$

因此实际实现中我们不需要显式地进行 QR 分解

4 算法的硬件描述

使用多周期实现, 会导致性能不佳; 流水线程度过高, 会超出 FPGA 片内资源限制。因此需要在时间和空间上进行取舍。

Name	714551	1077392	438	214	268	2645	58	8	4	1	1
N_mod_top										0	0
genblk1[0].bram_i (bram_of_1024_complex)	0	0	0	0	2	0	0	0	0	0	0
genblk1[1].bram_i (bram_of_1024_complex)	0	0	0	0	2	0	0	0	0	0	0
genblk1[2].bram_i (bram_of_1024_complex)	0	0	0	0	2	0	0	0	0	0	0
genblk1[3].bram_i (bram_of_1024_complex)	0	0	0	0	2	0	0	0	0	0	0
genblk1[4].bram_i (bram_of_1024_complex)	0	0	0	0	2	0	0	0	0	0	0
genblk1[5].bram_i (bram_of_1024_complex)	0	0	0	0	2	0	0	0	0	0	0
graph_memory (bram_of_1080p_graph)	468	14	160	80	256	0	0	0	0	0	0
m_cache2graph (cache2graph)	281	204	0	0	0	0	0	0	0	0	0
m_gen_poly (generate_poly)	207916	324555	5	2	0	743	0	0	0	0	0
m_iterations (iteration)	497591	739860	245	120	0	1878	0	0	0	0	0
m_pixels2bram (pixels2bram)	7	10	0	0	0	0	0	0	0	0	0
m_roots2pixels (roots2pixels)	8007	12420	24	12	0	24	0	0	0	0	0
m_travel_forward (travel_forward)	7	21	0	0	0	0	0	0	0	0	0
u_dpy_scan (dpy_scan)	28	35	4	0	0	0	0	0	0	0	0
u_ip_pll (ip_pll)	0	0	0	0	0	0	0	0	2	1	0
u_ip_rgb2dvi (ip_rgb2dvi)	158	142	0	0	0	0	0	8	0	0	1
u_led_scan (led_scan)	23	34	0	0	0	0	0	0	0	0	0
u_video1080p60hz (video)	22	32	0	0	0	0	0	0	0	0	0

Figure 2: 超出 FPGA 资源限制, 无法实现

4.1 128 级流水线实现

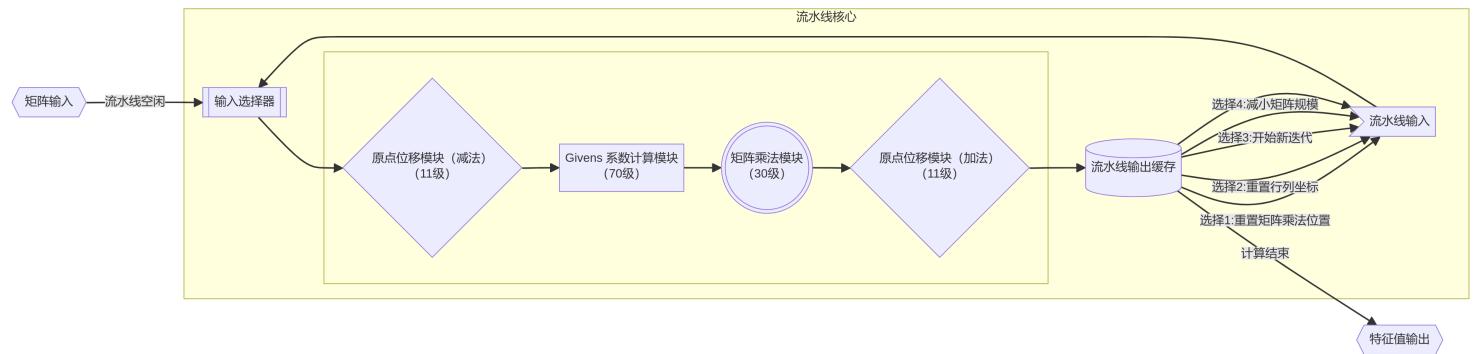


Figure 3: 流水线核心框架图

我们采用循环流水线的方式来节省 FPGA 资源，也即每个输入的矩阵会从头到尾经过该流水线多次，每次经过，都会触发一个或多个内部模块对它进行操作，至于不操作的模块，相当于起到了暂存数据的作用，我们也就不再需要额外的内存池来保存它们了。

在流水线输入端，一旦流水线存在空闲，则输入选择器会从外部获取一份输入，否则将会接受流水线的输出作为输入。在流水线输出端，如果处理结果满足结束条件，则会输出到外部，否则进行状态更新，重新作为流水线输入。

流水线内部由四大模块组成，分别是原点位移（减法）模块、Givens 系数计算模块、矩阵乘法模块、原点位移（加法）模块。原点位移模块只有当输入矩阵处于开始或结束 QR 迭代的状态才会对矩阵对角线上的元素加减偏移值。Givens 系数计算模块能对处于矩阵 $(i, j), (i - 1, j)$ 位置的元素计算一对 (c, s) 值。矩阵乘法模块是经过特殊化处理的，只需要进行形如 $ax + by$ 的运算，因为我们只需要关心 Q_i 中包含 c, s 的二阶方阵部分，每一次矩阵乘法只会影响两行或者两列。

自然，我们不可能指望经过一次流水线就能完成我们算法中的所有操作，例如 Givens 系数计算模块一次只能计算一对 (c, s) 值，在计算下一对之前，必须先对矩阵应用 Givens Rotation（矩阵乘法）。

在我们的设计中，矩阵乘法模块既可以整行整列并行（只需要流过一次就能完成一次矩阵乘法），也可以只计算几个元素（相应的需要重复经过流水线多次），后者主要是为了分配一部分资源给 VGA 进行展示，前者主要用于讨论我们设计的最高性能。

下面通过模拟说明如何使用该流水线完成期望的算法操作，在一次 QR 迭代中，矩阵 A 第一次经过流水线会被原点位移（减法）模块、Givens 系数计算模块、矩阵乘法模块三个模块处理

$$A \leftarrow A - sI_n$$

$$A \leftarrow Q_1 A$$

第 $i = 2, 3, \dots, n - 1$ 次经过流水线会被 Givens 系数计算模块、矩阵乘法模块处理

$$A \leftarrow Q_i A$$

第 $n - 1$ 次经过流水线后， A 变换为上三角矩阵，也就是我们所求的 R 矩阵，为了节省资源，我们直接对矩阵 A 进行操作，并不将 R 和 Q 矩阵保存下来，而只保存每次求得的 (c_i, s_i) 系数，接下来需要完成

$$A \leftarrow RQ_1^H Q_2^H \dots Q_{n-1}^H + sI_n$$

第 $n - 1 + i$ ($i = 1, 2, \dots, n - 1$) 次经过流水线，直接使用以前计算出的 (c_i, s_i) ，进行矩阵乘法

$$A \leftarrow A Q_i^H$$

第 $2n - 2$ 次经过流水线时，矩阵乘法结束后顺便通过原点位移（加法）模块完成了对角线加法的操作

$$A \leftarrow A + sI_n$$

之后，可以选择开始一次新的迭代、减少矩阵规模或者选择输出到外部

4.2 模块内部设计

本设计为追求高性能目标，并未采用标准的 AXI4-stream 协议进行流水线实现，仅使用 valid 作为输入是否合法的表示。由于浮点数的各类运算具有确切的计算周期数，因此可以使用这些常量来规划流水线缓存的级数。在实际实现中，这些常量会在编译期通过最底层模块所用周期数相加计算得到，并非是硬编码的 Magic Number，会随着底层模块的实现相应改变。

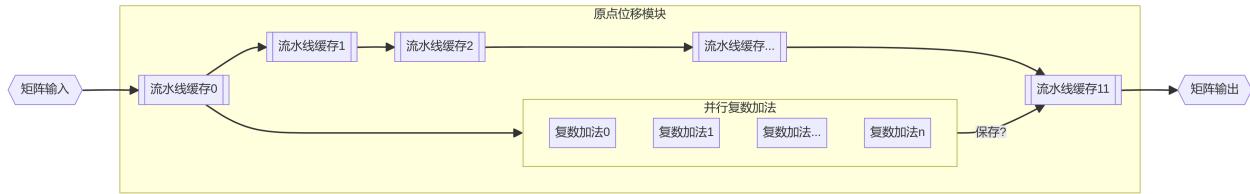


Figure 4: 原点位移模块

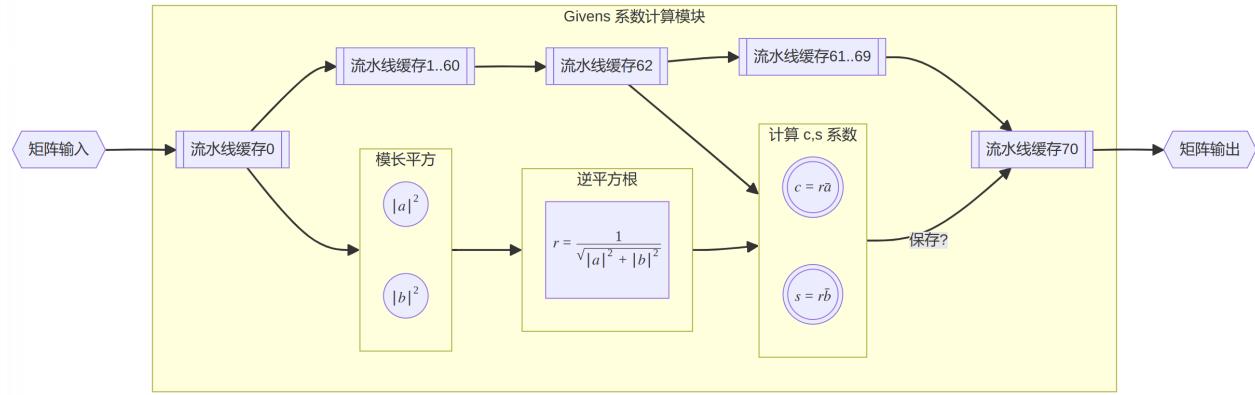


Figure 5: Givens 系数计算模块

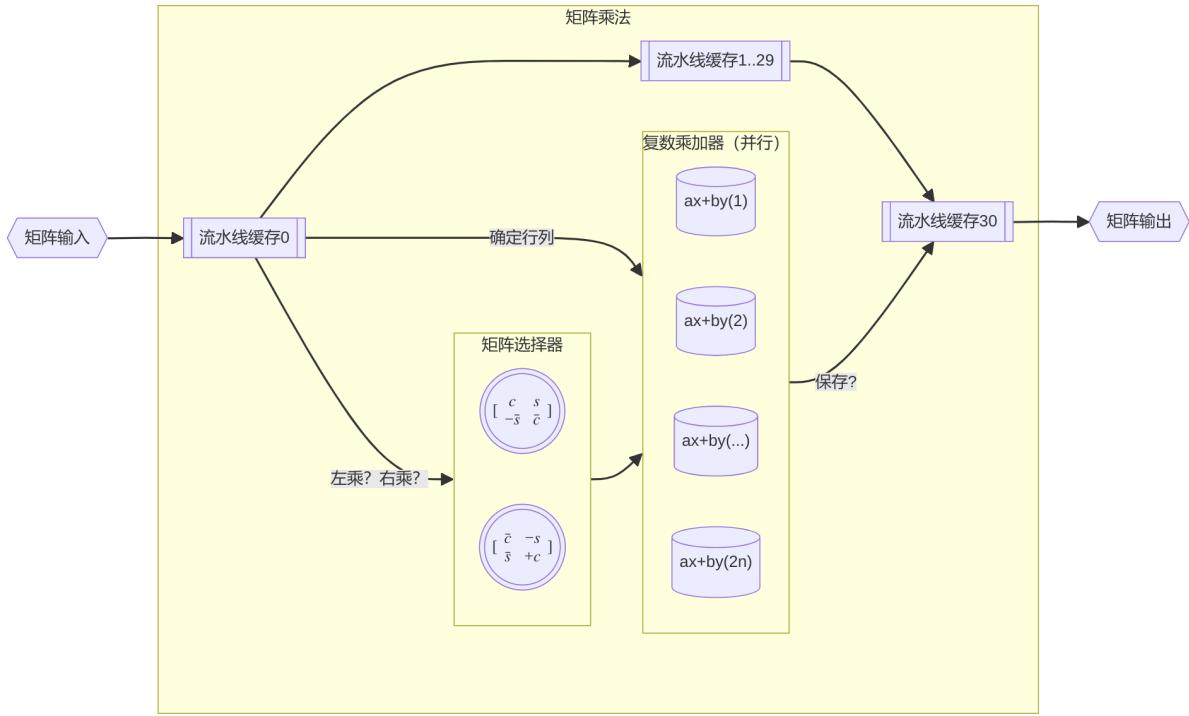


Figure 6: 矩阵乘法模块

为了记录每个矩阵的状态，图中的矩阵输入和矩阵输出不只是矩阵本身，还包括许多控制变量和数据寄存器，用以支撑流水线的运行。例如，如图 Figure 5 所示，计算完 c, s 系数后，会保存至该矩阵的数据寄存器中，便于矩阵乘法时的索引和复用。同理，每个矩阵还会保存当前操作的行列位置，矩阵乘法方向（左乘、右乘），原点位移偏移值，迭代次数等变量。

4.3 状态转移逻辑

在矩阵被四大模块处理完成后，需要根据其所处的状态进行下一步处理操作的选择。

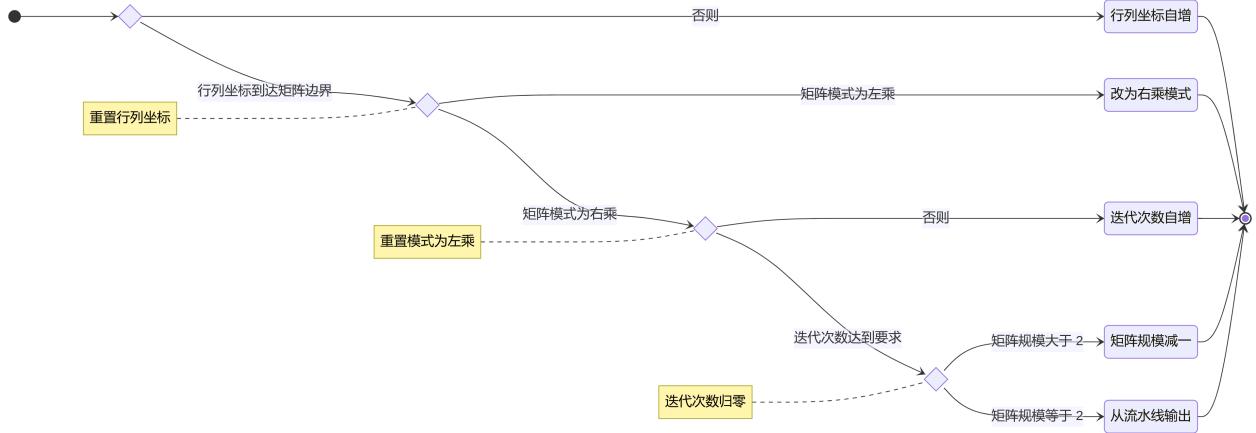


Figure 7: 状态转移逻辑

如 Figure 7 所示，我们把行列坐标的自增作为矩阵状态变化的最小粒度。事实上，状态变化的最小

粒度可以随时细化，只需要在该状态转移逻辑外层嵌套一个分支即可。例如，如果我想节省矩阵乘法模块的资源，削减并行复数乘加器的个数，只需要记录一个该资源的迭代器，以这个迭代器的自增作为最小粒度驱动行列坐标自增即可。

4.4 流水线效率分析

若流水线共 N 级，经过一级需要一个时钟周期，则每次至多可以从外部连续获取一批 N 个输入，之后需要等待流水线空闲才能获取。假设一个矩阵需要流过 K 次该流水线才能结束计算，则从一批第一个输入到一批最后一个输出间隔的总周期数为

$$C_{batch} = (K + 1)N$$

处理每个输入的均摊周期数为

$$C = \frac{1}{N} C_{batch} = K + 1$$

当然，如果下一批输入能紧接着上一批的输出，最终均摊周期数就有 $C = K$ ，这个在我们的设计中也进行了实现。接下来分析 K 的构成。

假设输入矩阵原始规模为 n ，当前规模为 m ，在每个规模迭代 T 次，当 $m = 2$ 且迭代完成后，输出到外部。根据上文的模拟，每次迭代需要流过 $2m - 2$ 次流水线，于是

$$K = T \sum_{m=2}^n 2m - 2 = n(n - 1)T$$

在我们的设计中 $n \leq 6$ ，经过实践，取 $T = 10$ 已经有六位有效数字正确，可以满足一般的精度需求，计算得 $C = 300$ 。

这意味着，计算一个 6 次复多项式的所有复根，我们的设计只需要 300 个周期（均摊）。而且由于该流水线设计时序良好，可以运行在 100MHz 的时钟下。

4.5 性能比较

我们使用 C/C++ 语言编写了一份 Algorithm 1 的程序，其中 QR 分解使用 Givens Rotation，矩阵乘法逻辑与硬件的优化相同，都只对必要的两行和两列进行形如 $ax + by$ 的运算，不保存 Q , R 矩阵，直接对 A 矩阵操作。

取 $n = 6$, $T = 10$, 开启 -Ofast 优化，更换不同输入重复测量 $M = 400^2 = 160000$ 次，在一块 11th Gen Intel i7-11800H (16) @ 4.600GHz 的 CPU 上。测得程序所用 CPU 时间为 1.309s。

在型号为 XC7A200T-2FBG484I 的 FPGA 上，使用 100MHz 时钟，以及相同的 n , T , M 参数，实现我们的设计，运行算法所用周期数约为 $C \times M = n(n - 1)TM = 4.8 \times 10^7$ ，时间约为 0.481s，有 2.7 倍的加速比。

另一方面，一块 11th Gen Intel i7-11800H (16) @ 4.600GHz CPU 的功率大约在 35W 左右 [1]，我们的芯片所用功率不超过 4W，考虑到功率差距较大，我们的设计的确在性能上有显著提升。

5 总体设计

我们希望在展示根的分布图的同时表现我们的高性能设计。我们决定构造大量多项式输入，采用上述流水线计算，并将它们根的分布图合并显示在屏幕上。

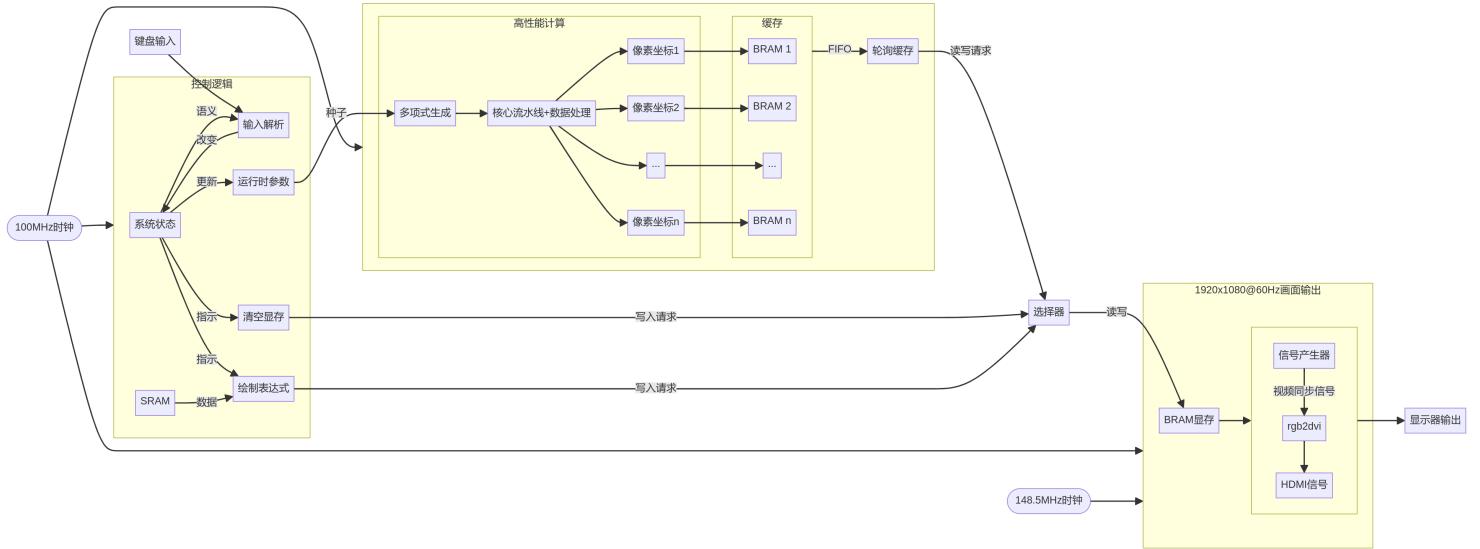


Figure 8: 总体设计图

我们最终还是决定使用 1080p 60Hz 的画面输出，尽管这可能并不是被推荐的行为。由于 ip 核 rgb2dvi 的最短脉冲长度限制，视频输出时钟上限为 136MHz[2]，超过该上限的时钟可能导致图像输出无法正常工作。一方面，分辨率的确对我们的展示非常重要，我们也尝试过降低帧数输出，但显示器并不支持（可以显示，但会有画面边界显示的问题）非 60 帧以外的 1080p 画面；另一方面，它确实能正常工作，我们就不必将分辨率降低为 1440x900 了。

关于显存，每个像素我们使用 4bit 位宽的灰度值，那么一共需要 $1920 \times 1080 \times 4b / 13140\text{Kb} = 61.64\%$ 的片内 RAM 空间，当一个像素坐标映射到显存对应位置时，我们将其灰度值加 1，最高 15，输出时等距映射到 0 ~ 255 的值。

由于我们的流水线性能较高，在每一个计算批次结束时，对于 n 次多项式，每个周期会产生 n 个复根作为结果输出，进一步被转化为 n 个像素坐标，为了一个一个处理这些像素坐标，我们使用 n 个 BRAM 将这些坐标暂存起来，等到流水线一个计算批次输出结束后，距离下一批次输出还有大量周期，在这个间隙，我们可以轮询这些 BRAM，依次读取新增的像素坐标，应用到显存上。

多项式的生成需要一定的规律性，如果完全随机，那么输出也是随机的散点，没有什么意义。因此我们设计了两个参数 t_1 和 t_2 ，以及对 t 采样的三种模式。第一种模式是在 $[-r, r]$ 的区间内等距采样；第二种模式是在 $[-\pi, \pi]$ 内等距取样，然后再 $\theta \mapsto e^{i\theta}$ ，这个可以打表实现；第三种模式是我们自行设计路径采样点，通过打表实现。这样可以确保输出的图像具有一定规律、对称性和美观度。

至于为何选择 100MHz 时钟作为主逻辑时钟，为什么不使用更高的频率，事实上，在我们的设计中，逻辑延迟很小，但我们的设计面积很大，打开 Implement Design 会发现我们的设计几乎写满了整个 FPGA，路径延迟就已经有 8ns，再进行优化确实需要花费更多时间。

5.1 工程文件

Table 1: 工程文件一览

目录	文件	含义	作者
src	cache2graph.sv	将缓存的数据映射到显存里	唐睿柏
	circle_chart.sv	对单位圆周采样的数据表	唐睿柏
	coef_in_controller.sv	输入参数控制器	唐睿柏, 张凯文
	complex.sv	复数基本运算	唐睿柏
	complex.h	整个工程的参数和信号定义	唐睿柏, 张凯文
	dpy_scan.sv	数码管显示	助教
	draw_top_bar.sv	绘制屏幕顶部的表达式	唐睿柏
	generate_poly.sv	产生大量多项式输入	唐睿柏
	givens_rotations.sv	计算 givens 变换的系数	唐睿柏
	keyboard_parser.sv	解析键盘信号输出控制信号	唐睿柏, 张凯文
	led_scan.sv	LED 灯显示	助教
	mod_top.sv	顶层文件	唐睿柏, 张凯文, 助教
	mul_givens_mat.sv	矩阵乘法	唐睿柏
	pixels2bram.sv	缓存像素坐标到 bram 里	唐睿柏
	poly2mat.sv	输出多项式的伴随矩阵	唐睿柏
	poly_value.sv	计算多项式点值	唐睿柏, 张凯文
	ps2_keyboard.sv	键盘输入	助教
	qr_decomp.sv	QR 分解迭代法流水线	唐睿柏
	real2circle.sv	计算三角函数	唐睿柏
	reset_all.sv	清空显存	唐睿柏
	roots2pixels.sv	将覆根映射到像素坐标	唐睿柏
	sampling_coefs.sv	对多项式参数采样	唐睿柏
	single_shift.sv	原点位移法	唐睿柏
	sram_controller.sv	SRAM 控制器	唐睿柏, 计组课程队友
	system_status.sv	系统状态	唐睿柏, 张凯文
	travel_forward.sv	读取显存输出到 VGA	唐睿柏, 张凯文
	video.sv	控制 VGA 信号	助教
	wb_arbiter.sv	显存读写仲裁器	唐睿柏, 计组课程队友
src/sim	*_tb.sv	仿真测试	唐睿柏, 张凯文
src/c_dir	*	C/C++ 语言测试代码	唐睿柏
src/py_dir	*.py	Python 辅助工具	唐睿柏
	bar.bin	用于绘制表达式的 SRAM 初始化文件	使用 Python 生成
	cos_sin_chart_hex.coe	三角函数数据表	使用 Python 生成

6 展示效果与说明

6.1 展示效果

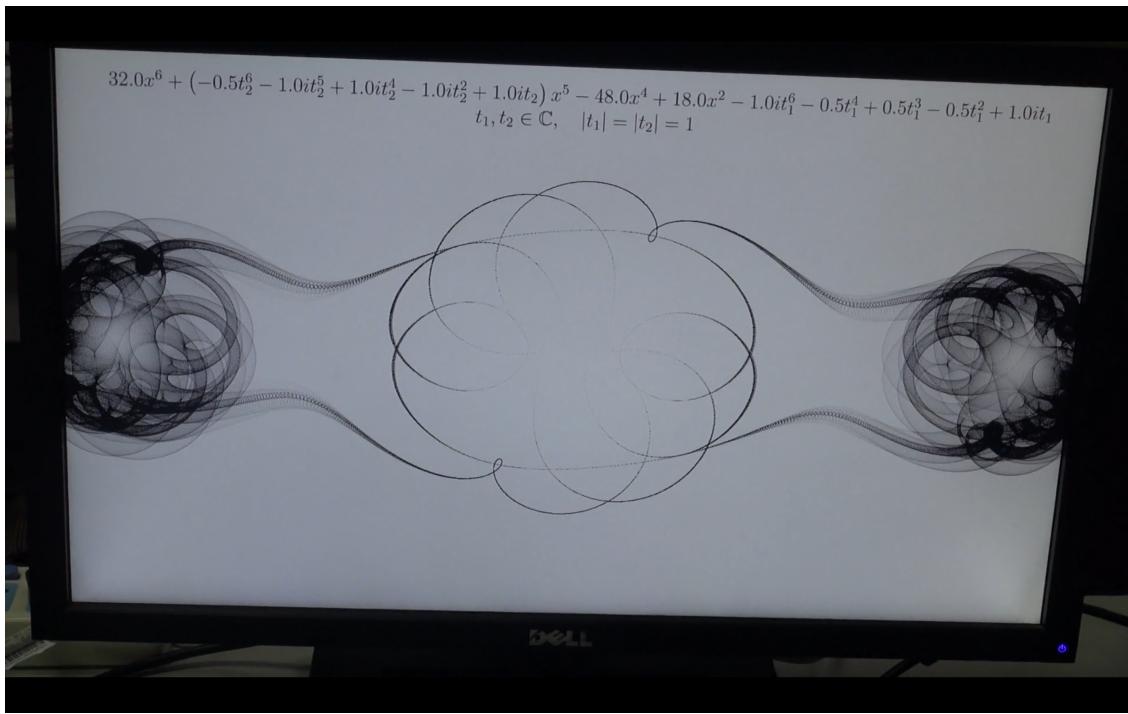


Figure 9: 复现来源: https://x.com/S_Conradi/status/1727769885885808695

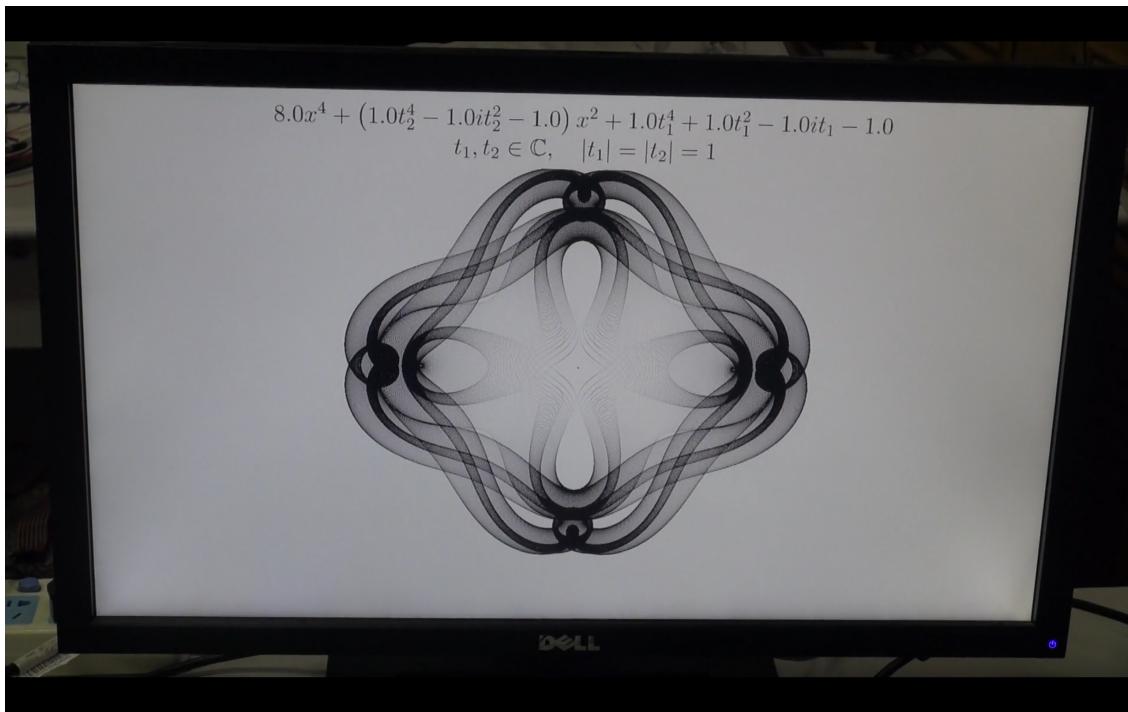


Figure 10: 复现来源: https://x.com/S_Conradi/

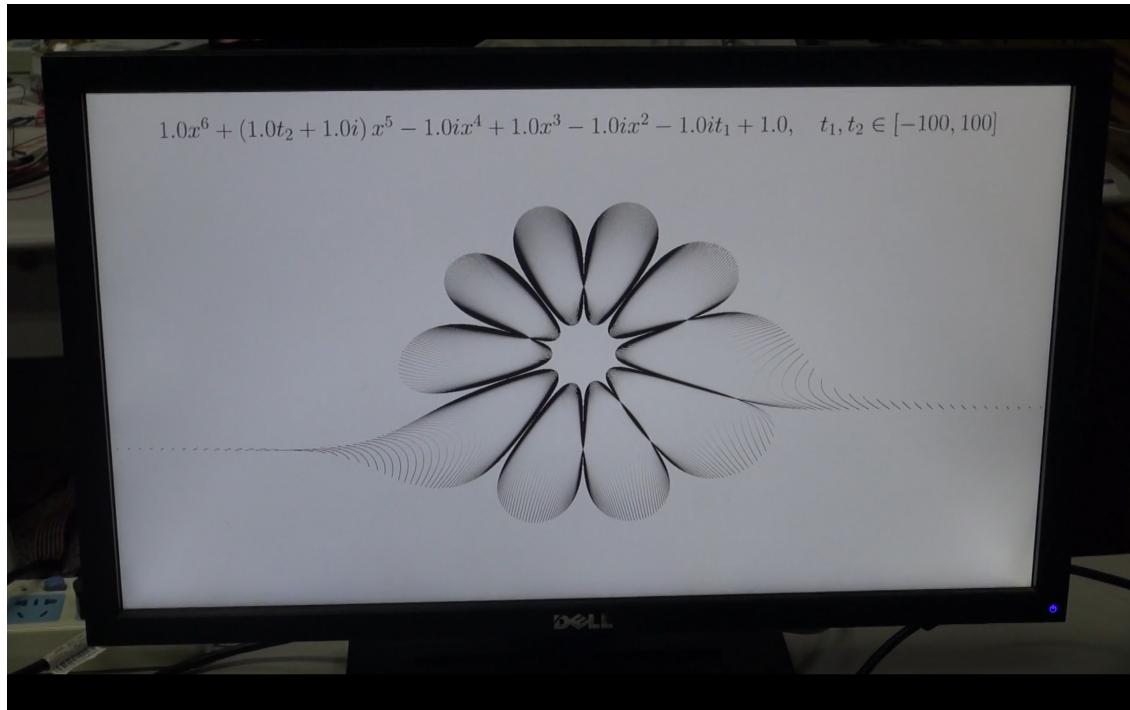


Figure 11: 复现来源: https://x.com/S_Conradi/status/1727064942178619490

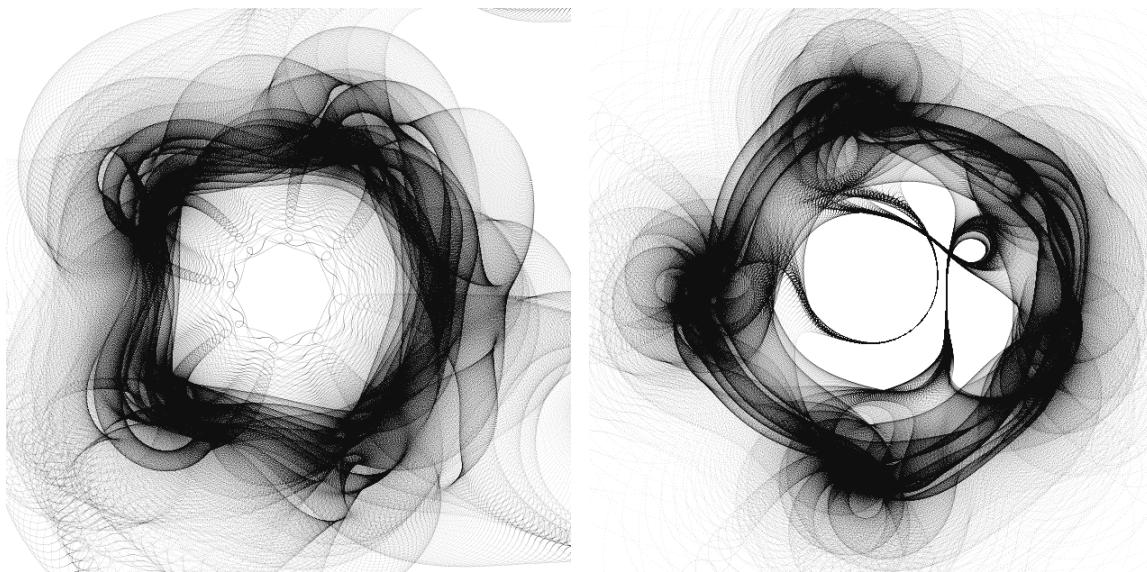


Figure 12: 来源: 自制, 实验版计算后写入 SRAM, 电脑读取数据绘制

6.2 实验演示说明

首先需要向 SRAM 写入 py_dir/bar.bin, 这样才能正常加载屏幕上方 1920x144 区域中显示的表达式图片。烧入 bitstream 将直接进入选择界面, 使用键盘 jk 按键分别控制选择光标的上下移动, 按回车选中表达式进行绘图。按下实验版上的 rst 键复位。

6.3 波形仿真说明

可以在工程根目录下或者 wave_config 文件夹下查看到若干.wcfg 文件，在 vivado 中添加并设置对应的顶层 testbench 文件即可进行仿真，由于信号较多，为节省报告篇幅，此处不做详细说明。

6.4 实验中遇到的问题与解决办法

实验中遇到的第一个问题，就是一开始的设计流水线程度过高，比本篇报告所讲述的设计还要快上 10 倍，大约 20 个周期就能出结果，导致仿真、综合以及实现都非常慢，仿真一秒只能推进 4 个周期，综合实现更是需要接近两个小时，最后报错发现资源使用远远超过 FPGA 的限制，进行了很多次性能与资源的平衡，最终变成了现在的设计。

第二个问题与 rgb2dvi 这个 ip 核有关 [2]，我们希望展示 1080p 60Hz 的图像而使用 148.5MHz 的像素时钟，但时序报告显示 TPWS 为负数，原因在于 rgb2dvi 会产生一个像素时钟 5 倍频率的序列化时钟，这个时钟的 min pulse width 无法满足要求，查询文档可以知道，在这个限制下，像素时钟最高只能为 136MHz，至于去年的 intel 板子最高能支持 1080p 60Hz 是因为使用了外置的芯片代替了 rgb2dvi 的工作。

第三个问题表现在 VGA 信号不稳定，一开始我们认为是我们的显存一边被写一边读引起的，要么改为双缓存，要么改在消隐区写入。但无论怎么改，这个问题都存在，即使是显存只被读取，不被写入，VGA 信号依然不稳定，我们甚至怀疑双口 BRAM 不能同时读，之后我们又进行了多次对比实验，还将 VGA 输出改为初始模板提供的输出，发现只要我们启动核心计算模块，VGA 显示就会黑屏，核心计算模块结束，VGA 才会恢复正常显示，时序报告也一切正常。种种迹象表明，我们的高性能计算模块会从物理上干扰 VGA 信号。助教更倾向于认为是电压的问题，但受限于没有工具，也没办法测量与证实。最终，为了展示，我们稍微降低了计算模块的性能，再次用时间换了空间，使得 VGA 能正常显示，至此困扰了我们最久的问题也得以解决。

References

- [1] 第十一代智能英特尔®酷睿™ i7 处理器产品规范. <https://ark.intel.com/content/www/cn/zh/ark/products/213803/intel-core-i7-11800h-processor-24m-cache-up-to-4-60-ghz.html>.
- [2] Elod Gyorgy. Rgb-to-dvi (source) 1.4 ip core user guide. <https://github.com/Digilent/vivado-library/blob/master/ip/rgb2dvi/docs/rgb2dvi.pdf>.
- [3] 史济怀, 刘太顺. 复变函数. 中国科学技术大学出版社, 1998.
- [4] 喻文健. 数值分析与算法 (第 3 版). 清华大学出版社, 2020.