计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(1):120−126 [doi: 10.15888/j.cnki.csa.006132]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于遗传算法的 Vivado HLS 硬件加速①

陈宝林[1], 黄　晞[1], 张　仕[2], 郭升挺[1], 吴家飞[1], 苏浩明[1]

[1](福建师范大学 医学光电科学与技术教育部重点实验室 福建省光子技术重点实验室, 福州 350007)
[2](福建师范大学 数学与计算机科学学院, 福州 350117)

摘　要: 为适应当前“大数据+深度模型”时代的到来, 利用 FPGA 进行各种算法的硬件加速为其提供了一种可行的解决方案. 本文利用 Vivado HLS 工具, 基于遗传算法设计了一套智能硬件加速架构, 编程实现自动生成 tcl 文件、自动调用 HLS 工具完成仿真和提取报表中的数据进行分析, 并对 Xilinx 公司所给的 FIR 和 DCT 等案例程序进行了测试. 实验中寻找到了较优的解决方案, 效率相比人工不断尝试的方法有了数量级的提升, 满足了当前一般算法在硬件加速的通用性.

关键词: 现场可编程门阵列; 加速; 遗传算法; 高层次综合

引用格式: 陈宝林,黄晞,张仕,郭升挺,吴家飞,苏浩明.基于遗传算法的 Vivado HLS 硬件加速.计算机系统应用,2018,27(1):120–126. http://www.c-s-a.org.cn/1003-3254/6132.html

Vivado High Level Synthesis Hardware Acceleration Based on Genetic Algorithm

CHEN Bao-Lin[1], HUANG Xi[1], ZHANG Shi[2], GUO Sheng-Ting[1], WU Jia-Fei[1], SU Hao-Ming[1]

[1](Fujian Provincial Key Laboratory of Photonics Technology, Key Laboratory of Opto-Electronic Science and Technology for Medicine of Ministry of Education, Fujian Normal University, Fuzhou 350007, China)
[2](College of Mathematics and Computer Science, Fujian Normal University, Fuzhou 350117, China)

Abstract: At present, in order to adapt to the coming of “big data and in-depth model” age, a feasible solution is put forward by using FPGA to realize hardware accelerator of various algorithms. In this study, by using Vivado HLS tools, a set of intelligent hardware acceleration architecture is designed based on the genetic algorithm, which can automatically generate TCL file by programming, and automatically call HLS tool to complete the simulation analysis and extract the data to analyze in the report. What’s more, the case programs like FIR and DCT given by the Xilinx company are tested. A better solution is found in the experiments, and the efficiency is increased by magnitude compared with the manual methods. It has met the universality of the general algorithm in hardware acceleration.

Key words: field programmable gate array; accelerate; genetic algorithm; high level synthesis

过去几年, 机器学习在各个领域和商业应用已经变得非常普遍. 2016 年 3 月, 基于深度学习的人工智能程序与世界围棋冠军、职业九段选手李世石进行人机大战, 同年 12 月, 百度研发的人工智能机器人又与世界记忆大师王峰进行了人脸识别挑战. 至此, 深度学习的热潮再次加速了机器学习和人工智能的发展. 然而, 深度学习模型对精度要求和计算能力也越来越高, 神经网络的大小也发生了爆炸式的膨胀, 例如: 有着 1000 亿神经元连接的百度大脑和 10 亿神经元连接的谷歌猫脸识别[1]. 在如此巨大的数据与规模, 只能靠更好的硬件来加速才能适应其需求. 通常来说, 硬件在执行诸如复杂的算法, 需要将数据进行转移, 以及重复执行各种操作等都比软件操作快得多[2].

到目前为止, 硬件加速主要是使用图形处理单元

① 基金项目: 福建省科技厅工业高校产学合作项目 (2013H6008)
收稿时间: 2017-03-24; 修改时间: 2017-04-13; 采用时间: 2017-04-24; csa 在线出版时间: 2017-12-22

(GPU) 集群作为通用的计算图形处理单元 (GPGPU)[3]. 此外, 开放型并行程序设计标准 (OpenCL) 对 FPGA、DSP 和 GPU 等一些硬件均支持, 并且对开发人员来说是免费、开源的, 因此作为异构硬件编程的工具也深受吸引. 除了 GPU 外, FPGA 由于硬件配置灵活, 具有可编程、高可靠性、高集成度和高速等优点[4], 且在单位能耗下与 GPU 相比有更好的性能, 在功耗、性能和实时性方面显著优于其他处理器. 2015 年, Intel 以 167 亿美元收购 了 Altera、Xilinx 和 IBM 正式联手, 都充分说明了 FPGA 在这个充满竞争的大数据市场的重要性. 当前 FPGA 的主要厂商是 Xilinx 和 Altera, 这两家公司在全球市场中占据着近 90% 的市场份额. 因此无论是在未来深度学习领域里, 还是其他相关领域, FPGA 将更受广大科研人员和业界的关注.

目前 FPGA 的设计主要采用 Verilog HDL 和 VHDL 两种硬件描述语言. 硬件描述语言是属于并行结构, 而且需要有一定的硬件基础才能进行仿真、综合和布局布线. 一般开发人员所熟知的 C/C++语言是属于顺序执行, 硬件描述语言与 C/C++语言还是存在着一定差距, 文献[5]提到, 一百万门的数字逻辑的时序设计需要三百万行 RTL 代码, 这就使得一些软件工程师望而却步.

本文将采用 Xilinx 公司所提供的 Vivado HLS 工具套件, HLS (High Level Synthesis) 为高层次综合, 它能将软件开发人员所编写 C/C++等高级语言代码转到可编程逻辑设计中, 用户无需事先了解相关硬件知识就可实现 RTL 级的硬件功能. 利用 Vivado 套件可以缩短 1/3 的 RTL 仿真时间, 提高超过 100 倍的算法验证速度[6], 这不仅仅受益于软件工程师, 而且大大加速了 IP 创建, 缩短了开发周期, 提高了开发效率. 目前国内外学者大多针对特定算法进行优化加速或只是对 HLS 进行粗糙应用, 并没有具体的指令优化策略和依据, 拓展性不好, 缺乏通用性. 比如: 文献[7]想到了对缓冲区进行管理以及对带宽进行优化, 提出了一种 roofline 模型的设计方法, 通过数据重用减少外部数据获取的延时, 从而找到了最优性能和最低 FPGA 资源消耗. 但此种方法只是针对特定情形, 缺乏通用性, 也没法找到最优全局性能. 文献[8]提出了基于 HLS 开发方式的线性方程组求解数据通路设计, 当中只是对 HLS 工具进行粗糙应用, 并没有具体的指令优化依据和策略, 而且拓展性不好. 文献[9]利用软硬件协同设计方法, 针对深度学习不同拓扑结构下的预测过程和训练过程的通用计算部分进行加速, 但在精确度和性能权衡上并没有说服力, 通用性不好. 文献[10]提出了首个开源程序优化器来自动重写给定程序以便优化延时, 实验结果显示, 生成的程序可享有 12 倍的加速, 同时增加 了 7 倍的精度, 但消耗了 4 倍多的 LUT, 此方法适合 HLS 内部自动优化. 文献[11]提出了两种对于 HLS 自适应 GA 方法: 自适应 GA 算子概率 (AGAOP) 和自适应算子选择 (AOS), AGAOP 和 AOS 展现了比 SGA 更好的鲁棒性, 文中表明自适应方法来解决 HLS 领域具有很大优势.

综合国内外对于 HLS 算法加速的研究, 可以得到: 传统的编程语言 (硬件描述语言 Verilog HDL 或 VHDL) 由于开发周期长, 开发难度大, 需要有一定硬件知识, 很难适应日益复杂的大数据时代, 而基于高层次综合 HLS 的提出完美地解决了此类问题.

1 遗传算法分析

Vivado HLS 工具中提供了 20 来种优化指令, 添加这些优化指令能使计算速度提升, 但同时也会大幅度增加 FPGA 的资源消耗. 主要的优化指令如表 1, 每条指令产生的优化效果可参考官方手册.

表 1 VivadoHLS 优化指令

指令编号	函数优化指令	循环优化指令	数组优化指令	接口优化指令
0	Dataflow	Dataflow	Array_Map	Array_Map
1	Pipeline	Pipeline	Array_Partition	Array_Partition
2	Inline	Unroll	Array_Reshape	Array_Reshape
3	Allocation	Loop_Flatten	Resource	Resource
4	Latency	Loop_Merge	Stream	Stream
5	Interface	Dependence		Interface
6	Function_Instantiate	Loop_TripCount		
7		Latency		

通常来说, 一段复杂程序的函数、循环、数组和接口是非常多的, 经排列组合后将迅速剧增. 例如对 DCT 源程序, 若采取 9 个主要优化点, 进行排列编码后将产生: 8X8X9X9X9X9X6X6=136048896 种组合(其中 8 表示对函数进行优化的 7 种优化指令加上一种默认不添加指令情况, 9 和 6 则分别对应循环和数组的情形), 如果把一条指令下的不同参数也进行编码(本实验采取一定策略的随机生成, 暂不进行编码), 将产生不可估计的组合, 如此巨大的搜索空间单从人工尝试的手段是不可能完成的. 假定产生一种解决方案进行分析将花费 35s 的时间, 在不考虑参数设置的情形下, 1 亿多种组合将花费超过 25 年的时间, 这个还是由机器运行搜索的时间. 倘若采用人工不断尝试的方法来寻找较优的解决方案, 所需的时间至少是这个的几十倍. 这就非常有必要找到一种策略来快速搜索有用的优化点, 附加考虑到时延与资源利用的权衡问题. 因此, 本实验采用遗传算法来快速搜索全局较优解.

遗传算法是源自生物界中的“物竞天择, 适者生存”, 模拟生物体在自然选择和遗传过程中所发生的繁殖、交叉和基因突变现象[12]. 一个种群经过漫长的繁衍, 种群中的基因会逐步向能更好适应环境的方向发展, 优胜劣汰, 后代留下来的基本是适应度较强的优良个体. 本实验利用 GA 从随机产生的初始种群 (这里的种群指初始随机产生染色体条数的总数) 开始搜索, 通过一定的选择, 交叉和变异操作, 逐步迭代产生新的种群.

实验中具体的整个流程框架如图 1, 首先进行优化点的随机选择, 自动生成.tcl 文件后, 调用 Vivado HLS 工具进行仿真分析生成初始种群. 然后, 提取生成报表中的 XML 数据进行分析, 其中图 2 和图 3 为 FIR 程序中未添加任何指令情况下所产生的性能评估与资源评估数据. 接着在进行染色体的编码与适应度的计算. 最后在利用遗传算法进行染色体的选择、交叉、变异等一系列迭代重新生成更优的种群.

1.1 染色体编码

以下代码为 DCT 中读数据的一小段程序, 其中 read_data 为函数名, RD_Loop_R、RD_Loop_C 为循环的语句标号, buf、input 数组名.

```
void read_data(short input[N], short buf[DCT_SIZE][DCT_SIZE])
{
  int r, c;
  RD_Loop_R:
  for (r = 0; r < DCT_SIZE; r++)
  {
  RD_Loop_C:
    for (c = 0; c < DCT_SIZE; c++)
      buf[r][c] = input[r * DCT_SIZE + c];
  }
}
```

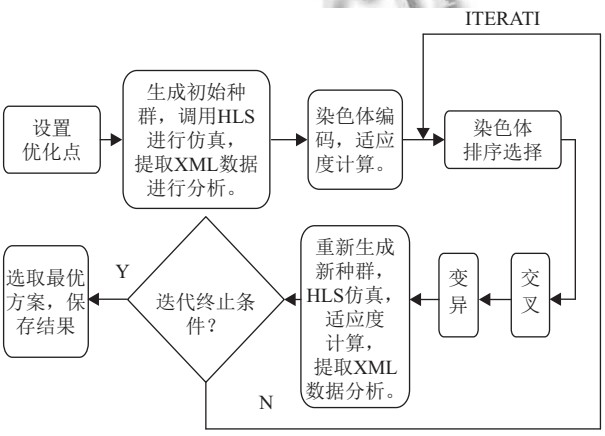

图 1 流程框架图

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.43	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
78	78	79	79	none

图 2 性能评估图

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	39
FIFO	-	-	-	-
Instance	-	4	0	0
Memory	0	-	64	6
Multiplexer	-	-	-	114
Register	-	-	179	-
Total	0	4	243	159
Available	650	600	202800	101400
Utilization (%)	0	~0	~0	~0

图 3 资源利用评估图

实验中主要是针对程序中的函数、循环、数组、接口进行指令的选择性优化. 如表 2 为按照表 3 的染色体编码自动生成的一条染色体, 其中前 6 位的评估指标分别表示 Fitness(适应度)、Latency(时延)、DSP48E、FF(寄存器)、LUT(查找表)、BRAM_18K, 后面部分 (第 7 位到第 16 位) 以 4 个字段为一组, 按照表 2 进行编码, 例如: 0 dct_2d 2 7, 0 表示函数, dct_2d 表示函数名, 2 表示子函数, 7 表示所添加的指令. 其中 Deep1 有两个取值: 1(表示顶层函数)、2(表示子函数); Deep2 表示共有几层循环嵌套, 如 1 表示一层嵌套, 2 表示两层嵌套; Deep3 表示数组维数, 如 1 表示一维数组, 2 表示两维数组; Deep4 表示接口数组维数, 表示方法与 Deep3 相同. Directive1 到 Directive4 分别对应表 1 中的函数、循环、数组和接口的优化指令编号.

表 2 随机生成的一条染色体

染色体位数	1	2	3	4	5	6
评估指标	Fitness	Latency	DSP48E	FF	LUT	BRAM_18K
评估指标值	0.00192186	3321	1	249	362	5
染色体位数	7	8	9	10	11	
四字段编码	0 dct_2d 2 7	0 dct_1d 3 0	1 O_Loop 1 7	1 XRI_Loop 2 7	1 XRI_Loop 2 1	
染色体位数	12	13	14	15	16	
四字段编码	1 XCI_Loop 2 7	1 RD_Loop_C 2 0	1 WR_Loop_C 2 4	2 buf_2d_in 2 0	2 col_inbuf 2 2	

表 3 染色体编码

Chromosome	第一位	第二位	第三位	第四位
Function	0	语句标号1	Deep1	Directive1
Loop	1	语句标号2	Deep2	Directive2
Array	2	语句标号3	Deep3	Directive3
Interface	3	语句标号4	Deep4	Directive4

1.2 适应度计算

本实验的目的是寻找最少时延与面积, 选择适应度值尽可能大的染色体, 因为适应度值越大, 表示的染色体越优质. 以下为所求的适应度公式[11]:

$$Fitness=\frac{1}{Latency}+\frac{1}{DSP48E+FF+LUT+BRAM} \quad (1)$$

说明. Latency 表示时钟周期延迟, 即要得到输出结果得花费的时钟个数. DSP48E、FF、LUT 和 BRAM 表示添加某种优化指令后所产生的资源利用个数, 具体可参考图 3 中的资源利用评估图.

1.3 选择操作

本实验中首先根据提取到的适应度值大小进行染色体排序, 在以一定概率从中选择优良的个体, 剔除劣质染色体, 按照轮盘赌选择法对优良个体进行重插入. 比如表 3 中的三条染色体, 先对 Fitness 进行排序, 然后在择优选取适应度高的 (如第一条), 剔除适应度低的 (如第二条). 最后在将第一条染色体重新插入到第二条中.

1.4 交叉操作

本实验采用的是部分映射杂交, 将父代的样本两两分组, 随机确定两个位置进行两两交叉. 交叉过程中为防止适应度高的染色体产生突变, 对高适应度个体采取不参与交叉和变异操作. 如表 4 中为交叉前的 3 组染色体, 第一组适应度较高, 暂不进行任何操作, 第二组和第三组分为一组进行交叉操作, 依次类推. 然后随机在染色体中确定两个位置, 比如 R1 和 R2, 再对其进行交叉操作 (即进行指令的交换):

0 dct 1 7 <---> 0 dct 1 0

2 row_outbuf 2 4 <---> 2 row_outbuf 2 5

如表 5 为染色体交叉后又重新生成的数据, 此时产生的适应度均高于交叉前的, 具体生成的情况并不固定, 也可能会偏低.

表 4 染色体交叉前

Fitness	Latency	Chromosome			
1	2	R1		R2	
0.00255404	412	0 dct 1 5	……	2 row_outbuf 2 4	……
0.00191727	1256	0 dct 1 7	……	2 row_outbuf 2 4	……
0.00195958	2920	0 dct 1 0	……	2 row_outbuf 2 5	……

表 5 染色体交叉后

Fitness	Latency	Chromosome			
1	2	R1		R2	
0.00255404	412	0 dct 1 5	……	2 row_outbuf 2 4	……
0.00205958	920	0 dct 1 0	……	2 row_outbuf 2 5	……
0.0019921	1050	0 dct 1 7	……	2 row_outbuf 2 4	……

1.5 变异操作

随机确定两个点进行变异操作. 变异是个小概率事件, 因此使用中所设置的参数要较小. 如表 6 所示, 随机确定 R1、R2 两个变异点, 对其进行变异操作:

0 dct 1 0 <---> 0 dct 1 3

2 row_outbuf 2 5 <---> 2 row_outbuf 2 1

表 6 染色体变异前

Fitness	Latency	Chromosome			
1	2	R1		R2	
0.00195958	2920	0 dct 1 0	……	2 row_outbuf 2 5	……

表 7 为变异后重新产生的结果, 可见比变异前的效果更好, 但这种操作具有偶然性的、不确定性, 也可能变得更差.

表 7 染色体变异后

Fitness	Latency	Chromosome			
1	2	R1		R2	
0.0025921	550	0 dct 1 3	……	2 row_outbuf 2 1	……

2 GA 的参数设置

实验中主要针对程序中的函数、循环、数组和接口进行指令选择优化, 同一条指令下的不同参数采用一定策略的随机自动生成. 实验对象选取了 Xilinx 主要的 3 个案例进行分析, 其中 GA 的具体参数设置如表 8.

表 8 遗传算法参数设置

实验对象	种群大小	最大遗传代数	代沟	交叉概率	变异概率	优化点	PC上运行时间(h)
Fir	50	20	0.95	0.8	0.05	4	8.37369
Matrixmul	50	20	0.95	0.8	0.05	5	7.48536
Dct	50	20	0.95	0.8	0.05	11	10.6974

说明. 种群大小、最大遗传迭代、交叉率和变异率对实验结果和运行速度都有一定影响, 但现无理论依据对其选取, 唯有通过不断尝试才能确定其合理的大小. 表中数据是经过数十次尝试 (包括增大迭代次数、种群数等) 而得到的结果, 由于实验对象不是特别复杂, 最优解收敛快, 因此迭代次数没有设置太大.

3 实验结果与性能评估

3.1 实验环境

本实验采用的处理器为: Intel(R) Core(TM) i5-3210M CPU @2.50GHz, 采用的 FPGA 为 Xilinx kintex7, xc7k160tfbg484-1. 利用 Vivado HLS 工具进行各种算法的初始仿真验证. 编译器采用 VS2013 进行主程序的大量仿真实验与遗传算法的测试分析, MATLAB 作为最后实验结果的数据图表分析.

3.2 实验结果与性能评估

如图 4 为 FIR 基于遗传算法寻找最优解的进化过程, 图中各点为每 50 个样本 (共 1050 个) 中选取的最优适应度, 包含初始解总共有 21 个点, 因此图像波动性不大, 为观察方便而绘制成连续图像. 如图, 经 20 次迭代后, 子代的适应度值已明显趋于稳定, 此时的适应度值为 0.33442982, 其中当种群迭代到第 6 代后, 优质染色体发生了突变, 适应度降低, 但由于在逐代进化中保留了父代的优良品质, 因此适应度又迅速提升.

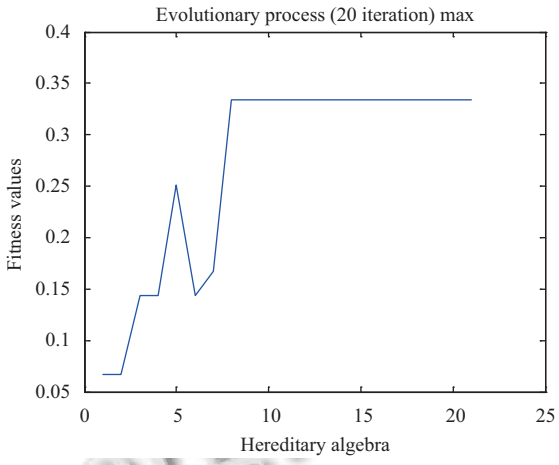

图 4 FIR 最优解的进化过程

图 5 为 FIR 初始产生的样本数据与经过遗传算法最终产生的样本数据的适应度对比图, 由图显然可以看出其后代基本稳定在 0.05 以上, 其中个别经过 GA 后的种群样本适应度偏低以及有大范围波动是由于染色体底层参数 (即一条指令下参数设置不同) 随机变化而产生, 但这并不会影响整个种群中的最优解. 为了探讨参数变化的影响, 实验中特意选取了复杂度较为简单的 Matrixmul (矩阵相乘) 来进行分析, 其中实验条件将一条指令下的参数固定 (即不人为进行随机数生成, 而是采取 HLS 工具中的默认参数). 如图 6、图 7 为 Matrixmul 产生的适应度分析图, 其中图 6, 由于 Matrixmul 复杂度不高, 后代在第 7 代的种群中就发现存在较优的解, 因此其后很快就稳定下来. 由图 7 可显然看出, 在没有参数影响下, 后代适应度全部稳定在

0.251675 中, 当然如果 FIR 也将参数固定也会产生类似结果.

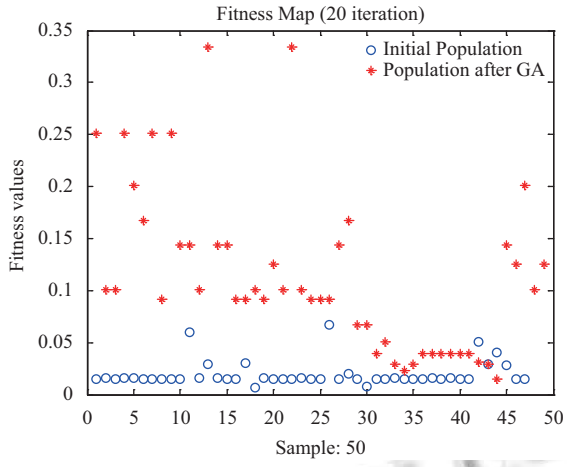

图 5 FIR 初始与 GA 后的样本适应度分布图

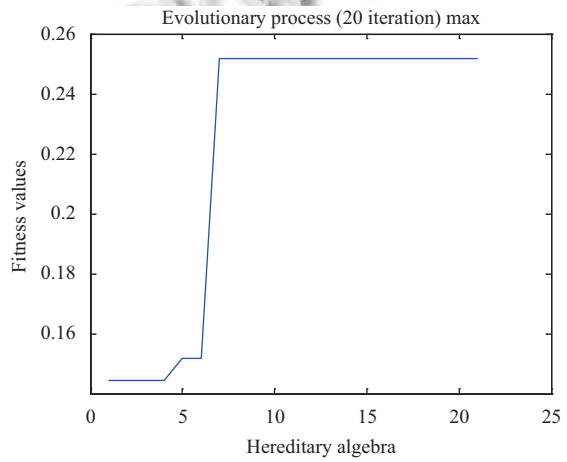

图 6 Matrixmul 最优解的进化过程

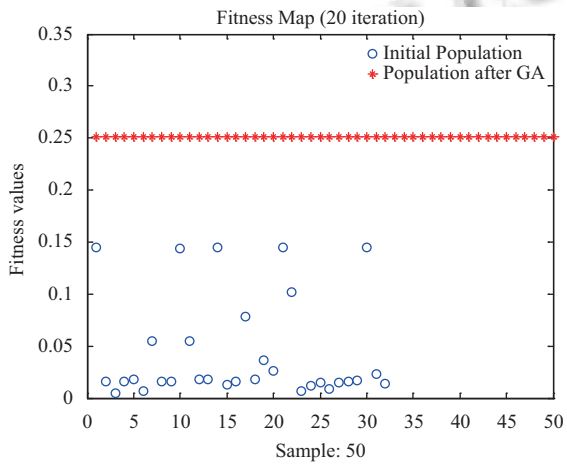

图 7 Matrixmul 初始与 GA 后的样本适应度分布图

表 9、表 10 为 FIR 和 Matrixmul 的性能与资源利用的各自对比, 第一行表示不添加任何优化指令的情况下所产生的结果, 第二行表示 Xilinx 公司例程中所给的最好方案 (实验条件相同下), 第三行表示本文中利用 GA 寻求最优解产生的结果, 对比三种方案易知, 在资源充足下, 权衡时延与面积, 第三种方案是最优的, 而且在资源比 Xilinx 公司提供的方案还少下, 却有较大的适应度, 在此也表明了利用 GA 寻求最优解的有效性.

表 9 Performance and utilization comparison to FIR

Comparison	Fitness	Latency	DSP48E	FF	LUT	BRAM_18K
Original	0.01528357	78	4	243	159	0
Xilinx	0.06745098	15	44	977	254	0
GA	0.33442982	3	44	676	192	0

表 10 Performance and utilization comparison to Matrixmul

Comparison	Fitness	Latency	DSP48E	FF	LUT	BRAM_18K
Original	0.022968	79	1	44	52	0
Xilinx	0.144579	7	27	479	71	0
GA	0.251675	4	27	501	69	0

同理, 利用本软件架构对 DCT 进行寻优, 运行结果与所得数据如图 8、图 9 和表 9, 具体的分析类似前面所述. 由于硬件条件的限制, 实验中减少了解空间的复杂度, 指令下的参数采用默认生成, 数组固定采用 ArrayPartition 的优化指令, 共设置 11 个优化点. 如图 8、图 9, 后代在第 13 次迭代中稳定下来, 寻优期间只有一次波动, 经过 20 迭代后基本在适应度为 0.00743396 中稳定下来. 值得注意的是, 从表 11 可以看出此时的资源消耗也非常大, 但在资源充足情况下, 以空间换时间也未尝不是一种不错的措施.

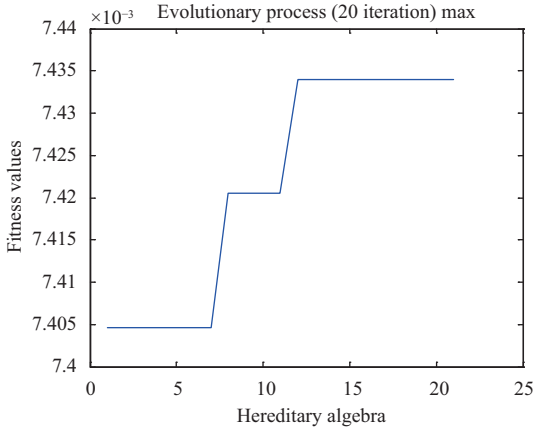

图 8 DCT 最优解的进化过程

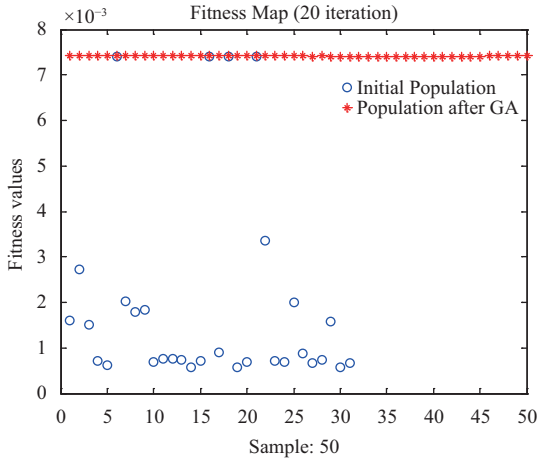

图 9 DCT 初始与 GA 后的样本适应度分布图

表 11 Performance and utilization comparison to DCT

Comparison	Fitness	Latency	DSP48E	FF	LUT	BRAM_18K
Original	0.00182244	3959	1	278	353	5
Xilinx	0.00242689	479	16	2360	566	6
GA	0.00743396	138	48	2875	2400	8

4 结束语

为适应目前的大数据时代, 将各种算法在硬件上进行加速是非常有必要的. 传统的硬件描述语言因其开发周期难, 开发难度大等因素而无法满足当前需求. 通常情况下, 各种算法源代码中的函数、数组、循环和接口是非常之多, 这就造成了优化点的解空间剧增, 因此本文提出了一种基于遗传算法利用 Vivado HLS 工具进行快速寻找最优解. 通过对 Xilinx 公司所提供的 FIR、DCT 和 Matrixmul 三个主要案例进行详细分析, 运用本人所写的一套程序架构对其仿真实验, 寻找到了较优的可行方案, 此程序架构也可适用于其他需要硬件算法加速的程序, 一定程度上满足了通用性. 实验过程中也发现了对于复杂程序, 由于优化点较多将产生组合爆炸, 使得想在庞大的解空间中寻找最优解变得很难, 因此必须采取一定策略来降低这个天文数字, 比如只针对其中的关键程序做优化, 这些都将是今后研究的重点.

参考文献

1 Wang C, Gong L, Yu Q, *et al*. DLAU: A scalable deep learning accelerator unit on FPGA. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2017, 36(3): 513–517.

2 http://blog.chinaunix.net/uid-20361370-id-1962845.html.

3 http://www.codesec.net/view/410065.html.

4 Slimane-Kadi M, Brasen D, Saucier G. A fast-FPGA prototyping system that uses inexpensive high-performance FPIC. Proceedings of 2nd Annual Workshop on FPGAs. Berkeley, CA, USA. 1994. 147–156.

5 O'Loughlin D, Coffey A, Callaly F, *et al*. Xilinx vivado high level synthesis: Case studies. Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies. Limerick, Ireland. 2013. 352–356.

6 思文. Vivado 设计套件将速度提高四倍. 中国电子报, 2013-06-18(007).

7 Zhang C, Li P, Sun G, *et al*. Optimizing FPGA-based accelerator design for deep convolutional neural networks. Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Monterey, CA, USA. 2015. 161–170.

8 王晓璐. 基于 Zynq 的 LS-SVM 算法加速器设计[硕士学位论文]. 哈尔滨: 哈尔滨工业大学, 2015.

9 余奇. 基于 FPGA 的深度学习加速器设计与实现[硕士学位论文]. 合肥: 中国科学技术大学, 2016.

10 Gao X, Wickerson J, Constantinides GA. Automatically optimizing the latency, area, and accuracy of C programs for high-level synthesis. Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Monterey, CA, USA. 2016. 234–243.

11 Mei FCC, Phon-Amnuaisuk S, Alias MY, *et al*. Adaptive GA: An essential ingredient in high-level synthesis. Proceedings of the 2008 IEEE Congress on Evolutionary Computation. Hong Kong, China. 2008. 3837–3844.

12 马永杰, 云文霞. 遗传算法研究进展. 计算机应用研究, 2012, 29(4): 1201–1206, 1210.