Lab1

Lab1.1 Xilinx高层次综合工具实验

针对矩阵乘算法进行HLS加速，具体提交实验报告。实验报告具体包括：

- C/C++代码，含pragma或directive TCL设置。
- 展示生成的Verilog或VHDL（关键截图）
- 不同加速方式得到的效果，包括latency值和不同资源数量对比。
- 必要的分析。
  如果时间充足，可尝试更复杂的C/C++应用，通过不同pragma或者C代码变换，得到不同的加速效果。
  另外，请将你修改后的C++代码，同时也压缩一并提交。

init

代码实现了矩阵乘法，时间复杂度O(n^3)

- 外层循环 (`for (i = 0; i < N; i++)`): 遍历矩阵 A 的行。
- 中间循环 (`for (j = 0; j < N; j++)`): 遍历矩阵 B 的列。
- 内层循环 (`for (k = 0; k < N; k++)`): 计算矩阵 A 的第 i 行与矩阵 B 的第 j 列的点积。

```C++
void matmult(short int A[N][N],short int B[N][N],int C[N][N]){
    int i, j, k;
    int sum = 0;

    for (i = 0; i < N; i++){
        for(j = 0; j < N; j++){
            for(k = 0; k < N; k++){
                sum += A[i][k] * B[k][j];
            }
            C[i][j] = sum;
            sum = 0;
        }
    }
}
```

Verilog & VHDL

- Verilog

```
[/home/eda304/large/zibo/lab1_2/prj/solution_init/syn/verilog]$ll
total 64
-rw-rw-r-- 1 eda304 eda304  1698 Sep 27 03:40 matmult_flow_control_loop_pipe.v
-rw-rw-r-- 1 eda304 eda304  1791 Sep 27 03:40 matmult_mac_muladd_16s_16s_32s_32_4_1.v
-rw-rw-r-- 1 eda304 eda304  1130 Sep 27 03:40 matmult_mul_mul_16s_16s_32_4_1.v
-rw-rw-r-- 1 eda304 eda304 51960 Sep 27 03:40 matmult.v
```

- VHDL

```
[/home/eda304/large/zibo/lab1_2/prj/solution_init/syn/vhdl]$ll
total 96
-rw-rw-r-- 1 eda304 eda304  1960 Sep 27 03:40 matmult_flow_control_loop_pipe.vhd
-rw-rw-r-- 1 eda304 eda304  2960 Sep 27 03:40 matmult_mac_muladd_16s_16s_32s_32_4_1.vhd
-rw-rw-r-- 1 eda304 eda304  2328 Sep 27 03:40 matmult_mul_mul_16s_16s_32_4_1.vhd
-rw-rw-r-- 1 eda304 eda304 85503 Sep 27 03:40 matmult.vhd
```

效果

latency值

matmult_csynth.rpt

large > zibo > lab1_2 > prj > solution_init > syn > report > matmult_csynth.rpt

```
14
15  ================================================================
16  == Performance Estimates
17  ================================================================
18  + Timing:
19      * Summary:
20      +--------+----------+----------+------------+
21      |  Clock |  Target  | Estimated| Uncertainty|
22      +--------+----------+----------+------------+
23      |ap_clk  |  13.00 ns|  8.262 ns|     3.51 ns|
24      +--------+----------+----------+------------+
25
26  + Latency:
27      * Summary:
28      +---------+---------+-----------+-----------+------+------+---------+
29      |  Latency (cycles) |   Latency (absolute)  |   Interval  | Pipeline|
30      |   min   |   max   |    min    |    max    |  min |  max |   Type  |
31      +---------+---------+-----------+-----------+------+------+---------+
32      |     2057|     2057|  26.741 us|  26.741 us|  2058|  2058|       no|
33      +---------+---------+-----------+-----------+------+------+---------+
```

不同资源数量

matmult_csynth.rpt

large > zibo > lab1_2 > prj > solution_init > syn > report > matmult_csynth.rpt

```
49  ================================================================
50  == Utilization Estimates
51  ================================================================
52  * Summary:
53  +-----------------+---------+------+--------+--------+-----+
54  |       Name      | BRAM_18K|  DSP |   FF   |   LUT  | URAM|
55  +-----------------+---------+------+--------+--------+-----+
56  |DSP              |        -|    16|       -|       -|    -|
57  |Expression       |        -|     -|       0|     678|    -|
58  |FIFO             |        -|     -|       -|       -|    -|
59  |Instance         |        -|     -|       -|       -|    -|
60  |Memory           |        -|     -|       -|       -|    -|
61  |Multiplexer      |        -|     -|       -|     266|    -|
62  |Register         |        -|     -|     607|       -|    -|
63  +-----------------+---------+------+--------+--------+-----+
64  |Total            |        0|    16|     607|     944|    0|
65  +-----------------+---------+------+--------+--------+-----+
66  |Available        |     1590|  1260|  728400|  364200|    0|
67  +-----------------+---------+------+--------+--------+-----+
68  |Utilization (%)  |        0|     1|      ~0|      ~0|    0|
69  +-----------------+---------+------+--------+--------+-----+
```

accelerate

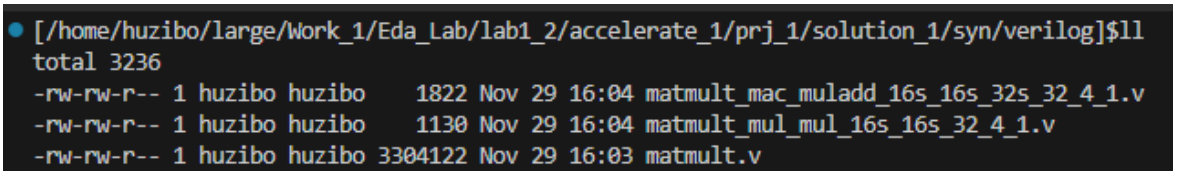
- 提高并行性：通过对数组的分区和循环的优化，增加硬件的并行访问能力。
- 减少延迟：通过流水线化循环操作，减少执行时间。
- 平衡资源与性能：根据硬件资源约束，选择适当的优化策略。

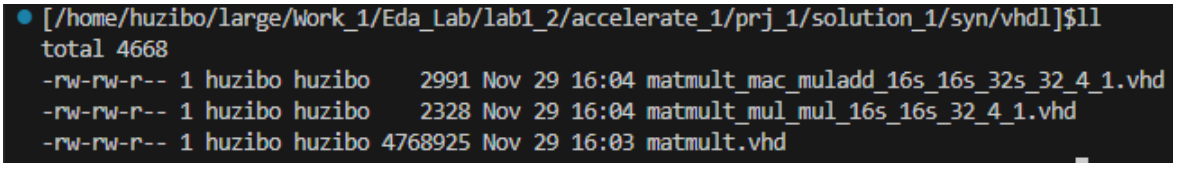
加速1

流水线循环操作，提高吞吐量；循环展开提高并行度

```
void matmult(short int A[N][N],short int B[N][N],int C[N][N]){
    int i, j, k;
    int sum = 0;
    #pragma HLS PIPELINE
    for (i = 0; i < N; i++){
        for(j = 0; j < N; j++){
            for(k = 0; k < N; k++){
                #pragma HLS UNROLL factor=4
                sum += A[i][k] * B[k][j];
            }
            C[i][j] = sum;
            sum = 0;
        }
    }
}
```

Verilog & VHDL

- Verilog
  - 

- VHDL
  - 

效果

- latency
  - 与基准比下降了1个数量级

```
================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+----------+----------+------------+
    |  Clock |  Target  | Estimated| Uncertainty|
    +--------+----------+----------+------------+
    |ap_clk  | 13.00 ns|  8.262 ns|     3.51 ns|
    +--------+----------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+----------+----------+-----+-----+---------+
    |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
    |   min   |   max   |    min   |    max   | min | max |   Type  |
    +---------+---------+----------+----------+-----+-----+---------+
    |      226|      226| 2.938 us| 2.938 us|  128|  128|      yes|
    +---------+---------+----------+----------+-----+-----+---------+
```

- resource
  -
```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+------+--------+--------+-----+
|       Name      | BRAM_18K|  DSP |   FF   |   LUT  | URAM|
+-----------------+---------+------+--------+--------+-----+
|DSP              |        -|  4096|       -|       -|    -|
|Expression       |        -|     -|       0|   62722|    -|
|FIFO             |        -|     -|       -|       -|    -|
|Instance         |        -|     -|       -|       -|    -|
|Memory           |        -|     -|       -|       -|    -|
|Multiplexer      |        -|     -|       -|    6120|    -|
|Register         |        -|     -|  105442|       -|    -|
+-----------------+---------+------+--------+--------+-----+
|Total            |        0|  4096|  105442|   68842|    0|
+-----------------+---------+------+--------+--------+-----+
|Available        |     1590|  1260|  728400|  364200|    0|
+-----------------+---------+------+--------+--------+-----+
|Utilization (%)  |        0|   325|      14|      18|    0|
+-----------------+---------+------+--------+--------+-----+
```

必要分析

- `#pragma HLS PIPELINE` 允许多个循环在硬件中并行执行
  - 减少总延迟，提高吞吐量，同时会造成硬件资源增加
- `#pragma HLS UNROLL factor=4` 将最内层k循环，每4次迭代展开为并行执行的硬件逻辑
  - 通过并行计算减少总延迟，同时增加硬件的并行度
  - 会产生
  - 每个时钟周期能完成4次迭代计算

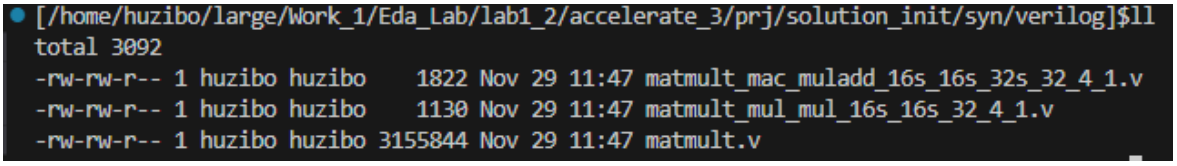
加速1中，如果数组 `A` 和 `B` 没有分区，则 `A[i][k]` 和 `B[k][j]` 的访问可能会发生冲突（因为默认情况下数组映射到单一存储块，存储端口有限），从而限制性能提升。

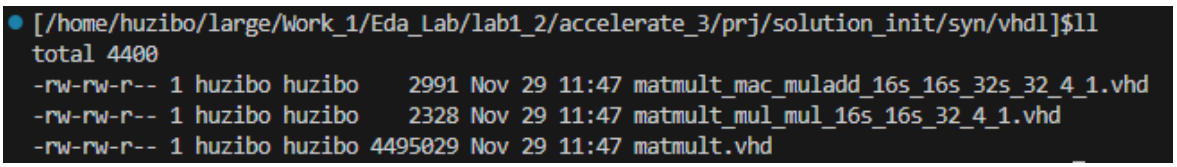
加速2

加速1中，如果数组 `A` 和 `B` 没有分区，则 `A[i][k]` 和 `B[k][j]` 的访问可能会发生冲突（因为默认情况下数组映射到单一存储块，存储端口有限），从而限制性能提升。

```
void matmult(short int A[N][N],short int B[N][N],int C[N][N]){
    int i, j, k;
    int sum = 0;
    #pragma HLS ARRAY_PARTITION variable=A complete
    #pragma HLS ARRAY_PARTITION variable=B complete
    #pragma HLS ARRAY_PARTITION variable=C complete
    #pragma HLS PIPELINE
    for (i = 0; i < N; i++){
        for(j = 0; j < N; j++){
            for(k = 0; k < N; k++){
                sum += A[i][k] * B[k][j];
            }
            C[i][j] = sum;
            sum = 0;
        }
    }
}
```

Verilog & VHDL

- Verilog
  - 

- VHDL
  - 

效果

- latency

accelerate_3 > prj > solution_init > syn > report > matmult_csynth.rpt

```
14
15  ================================================================
16  == Performance Estimates
17  ================================================================
18  + Timing:
19      * Summary:
20      +--------+----------+----------+------------+
21      |  Clock |  Target  | Estimated| Uncertainty|
22      +--------+----------+----------+------------+
23      |ap_clk  | 13.00 ns|  8.658 ns|     3.51 ns|
24      +--------+----------+----------+------------+
25
26  + Latency:
27      * Summary:
28      +---------+---------+----------+----------+-----+-----+---------+
29      |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
30      |   min   |   max   |    min   |    max   | min | max |   Type  |
31      +---------+---------+----------+----------+-----+-----+---------+
32      |       18|       18| 0.234 us| 0.234 us|    8|    8|      yes|
33      +---------+---------+----------+----------+-----+-----+---------+
```

- resource

accelerate_3 > prj > solution_init > syn > report > matmult_csynth.rpt

```
43
44  ================================================================
45  == Utilization Estimates
46  ================================================================
47  * Summary:
48  +-----------------+---------+------+--------+--------+-----+
49  |       Name      | BRAM_18K|  DSP |   FF   |   LUT  | URAM|
50  +-----------------+---------+------+--------+--------+-----+
51  |DSP              |        -|  4096|       -|       -|    -|
52  |Expression       |        -|     -|       0|   62722|    -|
53  |FIFO             |        -|     -|       -|       -|    -|
54  |Instance         |        -|     -|       -|       -|    -|
55  |Memory           |        -|     -|       -|       -|    -|
56  |Multiplexer      |        -|     -|       -|    4791|    -|
57  |Register         |        -|     -|   80203|       -|    -|
58  +-----------------+---------+------+--------+--------+-----+
59  |Total            |        0|  4096|   80203|   67513|    0|
60  +-----------------+---------+------+--------+--------+-----+
61  |Available        |     1590|  1260|  728400|  364200|    0|
62  +-----------------+---------+------+--------+--------+-----+
63  |Utilization (%)  |        0|   325|      11|      18|    0|
64  +-----------------+---------+------+--------+--------+-----+
```

必要分析

- `#pragma HLS ARRAY_PARTITION`，将数组A、B、C完全展开，这样硬件可以同时访问数组多个元素，提高了并行性。
- `#pragma HLS PIPELINE`，将整个循环流水化显著提高硬件吞吐量，减少硬件执行延迟
- 改为流水线，且提供了经可能高的并行访存，latency大幅降低，但资源使用大幅增高

附件：HLS pragmas

HLS pragma

HLS 工具提供各种*编译指示*，用于最优化设计、降低时延、提升吞吐量性能，以及减少生成的 RTL 代码的面积和器件资源利用率。这些编译指示可直接添加到内核源代码中。

类型	属性
内核最优化	- pragma HLS aggregate - pragma HLS alias - pragma HLS disaggregate - pragma HLS expression_balance - pragma HLS latency - pragma HLS performance - pragma HLS protocol - pragma HLS reset - pragma HLS top - pragma HLS stable
函数内联	- pragma HLS inline
接口综合	- pragma HLS interface - pragma HLS stream
任务级流水线	- pragma HLS dataflow - pragma HLS stream
流水线	- pragma HLS pipeline - pragma HLS occurrence
循环展开	- pragma HLS unroll - pragma HLS dependence
循环最优化	- pragma HLS loop_flatten - pragma HLS loop_merge - pragma HLS loop_tripcount
阵列最优化	- pragma HLS array_partition - pragma HLS array_reshape
结构封装	- pragma HLS aggregate - pragma HLS dataflow
资源使用情况	- pragma HLS allocation - pragma HLS bind_op - pragma HLS bind_storage - pragma HLS function_instantiate