汇报

黄宗乐 8/1

目录

1 主要思路（上次讨论内容）
2 Formalization
2.1 每次处理1个sample
2.2 每次处理k个sample
2.3 总体计算流程
3 实现思路
3.1 硬件资源数小于等于特征维度时
3.2 硬件硬件资源数大于等于特征维度时
4 实验情况
4.1 实现1
4.2 实现2
4.3 实现3
5 其他问题(语音讨论)

1 主要思路（上次讨论内容）

在网络中每个sample的维度变化如下：2 ---> 128 ---> 1，需要相乘的权重矩阵维度为(2,128)和(128,1)。

其中，根据矩阵运算的原理，累加维度无法并行（除非使用adder tree等结构，但是也不能优化到完全并行），因此第一个矩阵运算的理论最少周期数为2（II=1，128全部并行，共128组DSP资源），第二个矩阵运算的理论最少周期为128(II=1，共1组DSP资源)。然而，STREAM被最长的stage限制，因此第一个矩阵乘法并行到128没有意义。在这种情况下能达到的最好throughput=128 cycles/sample.

可以通过一次送入多个sample来达到提升并行度的效果。如果需要将throughput提升8倍，可以每次循环送入8 samples。对于第一个矩阵，可以通过8次循环完成计算；对于第二个矩阵，可以一次性处理8组数据。虽然第二个矩阵运算仍然至少需要128个时钟，但是着128个时钟完成了8个sample的计算，平均下来的throughput=128/8=16 cycles/sample。

接下来根据throughput=16 cycles/sample计算weight1计算需要的资源：全部并行至少需要2 cycles，则每个sample在特征维度的时钟数为16/2=8，即所需并行资源为128/8=16。因此根据上述计算，可以得到理论上的throughput和资源分配情况：

- throughput = 16 cycles/sample
- 矩阵1：16路并行
- 矩阵2：8路并行

2 Formalization

将上述计算过程进行归纳，可以得到下面两个计算阶段。

2.1 每次处理1个sample

网络有 $i = 1, 2, \dots n$ 层，这些层对应的矩阵维度为 (L_{i-1}, L_i)。则该网络中成为瓶颈的层号为 $BN = argmax_i(L_{i-1})$。能够达到的最大 $throughput = L_{BN-1}$ cycles/sample。对于其他层 $i \neq BN$，为了匹配上述throughput，在L_i维度使用的时钟为$\frac{L_{BN-1}}{L_{i-1}}$，则其需要的资源为$L_i \div \frac{L_{BN-1}}{L_{i-1}} = \frac{L_{i-1}L_i}{L_{BN-1}}$。如果想进一步提升throughput，可以增加同时处理的sample。

2.2 每次处理k个sample

网络有 $i = 1, 2, \dots n$ 层，这些层对应的矩阵维度为 (L_{i-1}, L_i)。每次处理k samples，则$throughput = \frac{L_{BN-1}}{k}$ cycles/sample。则对于$i = BN$，该层需要的的资源$N_{BN} = kL_{BN}$。对于$i \neq BN$，该层需要的资源$N_i = k\frac{L_{i-1}L_i}{L_{BN-1}}$

2.3 总体计算流程

根据上述分析，可以按照如下流程确定理论上分配的硬件资源：

- 确定需要的throughput
- 找到BN，计算相应的k使之能满足throughput
- 得到$N_{BN} = kL_{BN}$
- 得到$N_i = k\frac{L_{i-1}L_i}{L_{BN-1}}$, $i \neq BN$
- 检查各层资源之和是否小于已有硬件资源

3 实现思路

这一部分讨论了：在具体实现上，如何分配这些硬件资源、在什么维度上并行等问题。在第二部分的基础上，我们将网络中的矩阵乘法计算层分成两类：$N_i \leq Li$ 和 $N_i > Li$。下面分别讨论之：

3.1 硬件资源数小于等于特征维度时

第一部分中的第一个矩阵乘法属于这种情况。由于$N_i \leq Li$，因此每个sample本身就能提供足够的并行度。

对于第i层伪代码结构如下：

```
for loop in [0,k):
    for l in [0,L_{i-1}):
        for j in [0,L_i / N_i):
            jj in [0,Ni): parallel.
        end
    end
end
```

3.2 硬件硬件资源数大于等于特征维度时

第一部分中的第二个矩阵乘法属于这种情况。由于$N_i > Li$，因此需要同时处理$m(m > 1)$ samples，其中$m = Ni/Li$。此时输入是一个m sample组成的package。

对于第i层伪代码结构如下：

```
for loop in [0,k/m):
    for l in [0,L_{i-1}):
        j in [0, N_i): parallel
    end
end
```

4 实验情况

本周一共完成了三种实现，后两种实现换了更大的网络（4 ---> 128 ---> 2）。

和上周的结果相比，实现1将延时从2376 cycles降到了974cycles（使用的硬件资源数量不同，新的代码提升了并行度，但DSP使用也只占到了1%）。
实验2、3展示更大网络下的优化，通过优化代码结构，延时从641降到了576。

4.1 实现1

该实现处理的网络尺寸为2-128-1。完成最基本的任务。

- 理论计算的资源分配：$N_1 = 8, N_2 = 4$

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)
top		1276	4.253E3		1277		no	25	~0	78	1
dataflow_parent_loop_proc		1275	4.250E3		1275		no	1	~0	78	1
dataflow_in_loop_VITIS_LOOP_225_5		373	1.243E3		300		dataflow	1	~0	78	1
matmul1_1		299	997.000		299		no	1	~0	40	~0
VITIS_LOOP_38_1		272	907.000	68		4	no				
init		16	53.328	1	1	16	yes				
load_weight_block_loop		47	157.000	17	1	32	yes				
write_out		16	53.328	2	1	16	yes				
act1_1		177	590.000		177		no	0	0	16	~0
outer_loop		144	480.000	36		4	no				
inner_loop		26	86.658	12	1	16	yes				
matmul2_1		217	723.000		217		no	0	0	20	~0
VITIS_LOOP_121_1		4	13.332	1		4	no				
load_weight		140	467.000	14	1	128	yes				
VITIS_LOOP_152_2		4	13.332	1	1	4	yes				
loadIn_142		78	260.000		78		no	0	0	0	0
outer_loop		4	13.332	2	1	4	yes				
act2		52	173.000		52		no	0	0	2	~0
VITIS_LOOP_173_2		48	160.000	12		4	no				
storeDDR_1		70	233.000		70		no	0	0	0	0
VITIS_LOOP_225_5		1274	4.246E3	1274		4	no				

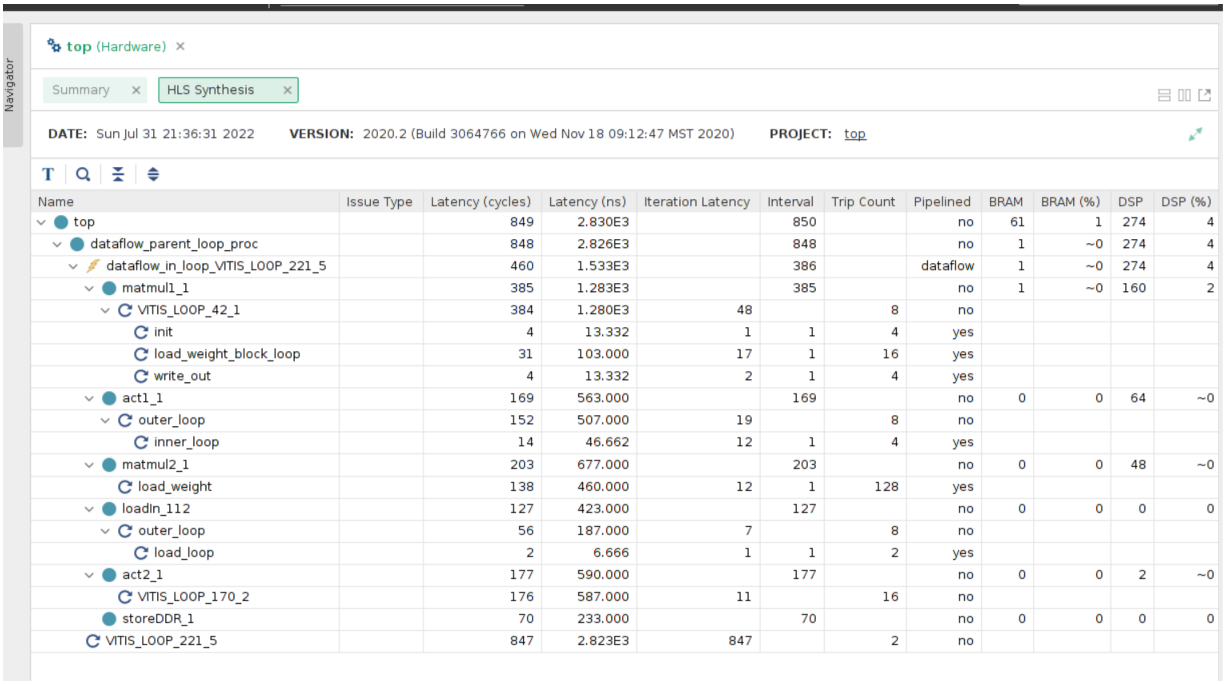
- 由于还有一些额外操作，导致MM1成为了瓶颈，实际上的最优分配如下

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)
top		973	3.243E3		974		no	25	~0	134	1
dataflow_parent_loop_proc		972	3.240E3		972		no	1	~0	134	1
dataflow_in_loop_VITIS_LOOP_225_5		316	1.053E3		218		dataflow	1	~0	134	1
matmul1_1		191	637.000		191		no	1	~0	80	1
VITIS_LOOP_38_1		176	587.000	44		4	no				
init		8	26.664	1	1	8	yes				
load_weight_block_loop		31	103.000	17	1	16	yes				
write_out		8	26.664	2	1	8	yes				
act1_1		113	377.000		113		no	0	0	32	~0
outer_loop		96	320.000	24		4	no				
inner_loop		18	59.994	12	1	8	yes				
matmul2_1		217	723.000		217		no	0	0	20	~0
VITIS_LOOP_121_1		4	13.332	1		4	no				
load_weight		140	467.000	14	1	128	yes				
VITIS_LOOP_152_2		4	13.332	1	1	4	yes				
loadIn_142		78	260.000		78		no	0	0	0	0
outer_loop		4	13.332	2	1	4	yes				
act2		52	173.000		52		no	0	0	2	~0
VITIS_LOOP_173_2		48	160.000	12		4	no				
storeDDR_1		70	233.000		70		no	0	0	0	0
VITIS_LOOP_225_5		971	3.236E3	971		4	no				

4.2 实现2

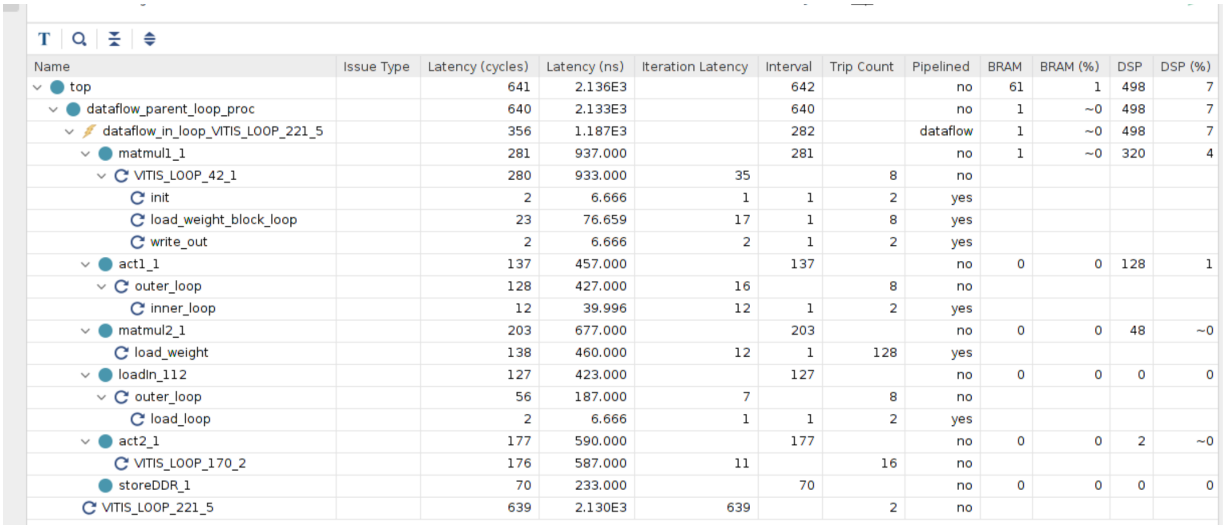
该实现处理的网络尺寸为 4-128-2；最完整的实现（包括MM1in/out的二维数组定义等）。

- 理论计算资源分配：$N_1 = 32, N_2 = 16$



Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)
top		849	2.830E3		850		no	61	1	274	4
dataflow_parent_loop_proc		848	2.826E3		848		no	1	~0	274	4
dataflow_in_loop_VITIS_LOOP_221_5		460	1.533E3		386		dataflow	1	~0	274	4
matmul1_1		385	1.283E3		385		no	1	~0	160	2
VITIS_LOOP_42_1		384	1.280E3	48		8	no				
init		4	13.332	1	1	4	yes				
load_weight_block_loop		31	103.000	17	1	16	yes				
write_out		4	13.332	2	1	4	yes				
act1_1		169	563.000		169		no	0	0	64	~0
outer_loop		152	507.000	19		8	no				
inner_loop		14	46.662	12	1	4	yes				
matmul2_1		203	677.000		203		no	0	0	48	~0
load_weight		138	460.000	12	1	128	yes				
loadIn_112		127	423.000		127		no	0	0	0	0
outer_loop		56	187.000	7		8	no				
load_loop		2	6.666	1	1	2	yes				
act2_1		177	590.000		177		no	0	0	2	~0
VITIS_LOOP_170_2		176	587.000	11		16	no				
storeDDR_1		70	233.000		70		no	0	0	0	0
VITIS_LOOP_221_5		847	2.823E3	847		2	no				

- 实际上的较优分配：把MM1的并行度提高了一倍，但是可以看到，根据Amdahl定理，实际上这种优化的好处在减少。固定开销包括iteration latency超出pipeline重叠的部分；pipe.read和pipe.write等。



Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)
top		641	2.136E3		642		no	61	1	498	7
dataflow_parent_loop_proc		640	2.133E3		640		no	1	~0	498	7
dataflow_in_loop_VITIS_LOOP_221_5		356	1.187E3		282		dataflow	1	~0	498	7
matmul1_1		281	937.000		281		no	1	~0	320	4
VITIS_LOOP_42_1		280	933.000	35		8	no				
init		2	6.666	1	1	2	yes				
load_weight_block_loop		23	76.659	17	1	8	yes				
write_out		2	6.666	2	1	2	yes				
act1_1		137	457.000		137		no	0	0	128	1
outer_loop		128	427.000	16		8	no				
inner_loop		12	39.996	12	1	2	yes				
matmul2_1		203	677.000		203		no	0	0	48	~0
load_weight		138	460.000	12	1	128	yes				
loadIn_112		127	423.000		127		no	0	0	0	0
outer_loop		56	187.000	7		8	no				
load_loop		2	6.666	1	1	2	yes				
act2_1		177	590.000		177		no	0	0	2	~0
VITIS_LOOP_170_2		176	587.000	11		16	no				
storeDDR_1		70	233.000		70		no	0	0	0	0
VITIS_LOOP_221_5		639	2.130E3	639		2	no				

4.3 实现3

网络尺寸为4-128-2：根据理论分析得到的最精简的实现。

- 理论计算资源分配：$N_1 = 32, N_2 = 16$

top (Hardware)

Summary | HLS Synthesis

DATE: Mon Aug 1 00:46:04 2022 **VERSION:** 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020) **PROJECT:** top

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)
top		753	2.510E3		754		no	61	1	304	4
dataflow_parent_loop_proc		752	2.506E3		752		no	1	~0	304	4
dataflow_in_loop_VITIS_LOOP_271_5		412	1.373E3		338		dataflow	1	~0	304	4
blockMul1_1		337	1.123E3		337		no	1	~0	160	2
VITIS_LOOP_56_1		336	1.120E3	42		8	no				
init		4	13.332	1	1	4	yes				
load_weight_block_loop		31	103.000	17	1	16	yes				
act1_1		169	563.000		169		no	0	0	64	~0
outer_loop		152	507.000	19		8	no				
inner_loop		14	46.662	12	1	4	yes				
unrollMul2_1		202	673.000		202		no	0	0	48	~0
load_weight		137	457.000	11	1	128	yes				
act2_1		10	33.330		10		no	0	0	32	~0
loadIn_19		81	270.000		81		no	0	0	0	0
outer_loop		9	29.997	3	1	8	yes				
storeDDR_1		70	233.000		70		no	0	0	0	0
VITIS_LOOP_271_5		751	2.503E3	751		2	no				

- 实际上的较优分配：把MM1并行度提高一倍。问题和实现2中是一样的。

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)
top		576	1.920E3		577		no	61	1	528	7
dataflow_parent_loop_proc		575	1.916E3		575		no	1	~0	528	7
dataflow_in_loop_VITIS_LOOP_198_5		323	1.077E3		250		dataflow	1	~0	528	7
blockMul1_1		249	830.000		249		no	1	~0	320	4
VITIS_LOOP_52_1		248	827.000	31		8	no				
init		2	6.666	1	1	2	yes				
load_weight_block_loop		23	76.659	17	1	8	yes				
act1_1		137	457.000		137		no	0	0	128	1
outer_loop		128	427.000	16		8	no				
inner_loop		12	39.996	12	1	2	yes				
unrollMul2_1		139	463.000		139		no	0	0	48	~0
load_weight		137	457.000	11	1	128	yes				
act2_1		10	33.330		10		no	0	0	32	~0
loadIn_110		81	270.000		81		no	0	0	0	0
outer_loop		9	29.997	3	1	8	yes				
storeDDR_1		70	233.000		70		no	0	0	0	0
VITIS_LOOP_198_5		574	1.913E3	574		2	no				

5 其他问题(语音讨论)

- aggregate相关问题：
  - 资源占用相对更多
  - 实现原理？（省略外层循环、read优化和写入的不优化）
- 实现相关问题：Amdahl定理影响下的throughput上限