HLS LAB #B Cholesky algorithm 110061638 呂政和

I. Introduction of Cholesky algorithm

在線性代數中,cholesky decomposition 是指將一個正定的 Hermiton 矩陣分解成一個下三角矩陣,與其共軛轉置之乘積,利用這種方法能提高代數的運算效率,在使用 monte carlo 上也十分有用。

方程式可以從下圖進行分析。

If we write out the equation

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T = egin{pmatrix} L_{11} & 0 & 0 \ L_{21} & L_{22} & 0 \ L_{31} & L_{32} & L_{33} \end{pmatrix} egin{pmatrix} L_{11} & L_{21} & L_{31} \ 0 & L_{22} & L_{32} \ 0 & 0 & L_{33} \end{pmatrix} = egin{pmatrix} L_{21}^2 & L_{21} \ L_{21} & L_{21}^2 + L_{22}^2 \ L_{31} L_{11} & L_{31} L_{21} + L_{32} L_{22} & L_{31}^2 + L_{32}^2 + L_{33}^2 \end{pmatrix}$$

we obtain the following:

$$\mathbf{L} = egin{pmatrix} \sqrt{A_{11}} & 0 & 0 \ A_{21}/L_{11} & \sqrt{A_{22}-L_{21}^2} & 0 \ A_{31}/L_{11} & (A_{32}-L_{31}L_{21})/L_{22} & \sqrt{A_{33}-L_{31}^2-L_{32}^2} \end{pmatrix}$$

and therefore the following formulas for the entries of L

$$egin{align} L_{j,j} &= (\pm) \sqrt{A_{j,j} - \sum_{k=1}^{j-1} L_{j,k}^2}, \ L_{i,j} &= rac{1}{L_{j,j}} \left(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k}
ight) \quad ext{for } i > j. \end{cases}$$

因此,要計算只需利用其的左、上方元素的值。計算通常是以以下其中一種順序進 行。

- Cholesky Banachiewicz 演算法從矩陣 L 的左上角開始,依行進行計算
- Cholesky-Crout 演算法從矩陣 L 的左上角開始,依列進行計算。

若有需要,整個矩陣可以逐個元素計算得出,無論使用何種順序讀取。

經典 cholesky decomposition 的一個變形是 LDL 分解,即 $A=LDL^*$,其中 L 是一個下三角矩陣,D 是一個對角矩陣。

該分解步驟如下: $A=LDL*=LD^{0.5}(D^{0.5})*L*=LD^{0.5}(LD^{0.5})*$ 以下舉一個實對稱矩陣作為範例:

$$\begin{pmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 \\ 6 & 1 & 0 \\ -8 & 5 & 3 \end{pmatrix} \begin{pmatrix} 2 & 6 & -8 \\ 0 & 1 & 5 \\ 0 & 0 & 3 \end{pmatrix}$$

下面為其 LDL^T 分解:

$$\begin{pmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ -4 & 5 & 1 \end{pmatrix} \begin{pmatrix} 4 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 9 \end{pmatrix} \begin{pmatrix} 1 & 3 & -4 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{pmatrix}$$

II. Run this design in CPU

透過利用 CPU 來計算:不同 size 下所需的運算時間,可以從下圖看到從 128*128 到 512*512 所需的時間會是 128*128 最少,512*512 最大。

```
[g110061638@ic22 cpu_src]$ ./test -M 128 -N 128 -seed 12
INFO: Matrix Row M: 128
INFO: Matrix Col N: 128
INFO: Finish CPU execution
INFO: CPU execution time is:300 us
errA = 0
dataAN = 128
dataAM = 128
INFO: Result correct
```

圖一: 128*128 所需時間為 300us

圖二: 256*256 所需時間為 2509us

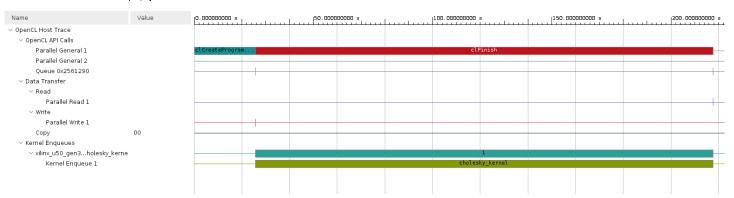
圖三:512*512 需時間為 21390us

這邊特別注意的是,在 cholesky 的 code 中,他的最大值可以設定到 2048*2048 去進行運算,但可以從下圖看到,運算會出現 error,這是因為學校 server 的限制所導致。

```
[g110061638@ic22 cpu_src]$ ./test -M 1024 -N 1024 -seed 12
INFO: Matrix Row M: 1024
INFO: Matrix Col N: 1024
Segmentation fault
```

III. Module 1 BaseLine

在這個 LAB 中,我不是用 vitis GUI 介面來完成,而是利用 terminal 來見環境並跑軟體與硬體的模擬,可以從產生的 run summary 看到執行的結果如下圖:



其中,圖上 OpenCL API Calls 說明所有的 OpenCL API Calls 會記錄在這項

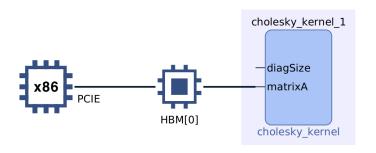
Data Transfer: 這欄會記錄 DMA transfers from the host to the memory。

The data transfer from the host to the device appear under Write as they are written by the host, and the transfers from device to host appear under Read.

Kernel Enqueues: The kernels enqueued by the host program are shown here.

Multiple kernels can be scheduled to be executed at the same time, and they are traced from the point they are scheduled to run until the end of the kernel execution. Multiple entries would be shown in different rows depending on the number of overlapping kernel executions.

System diagram:



Estimated Resources

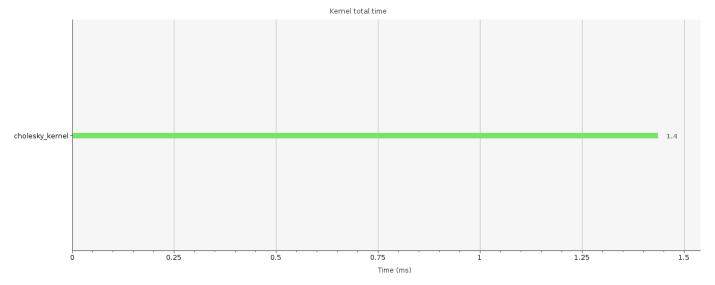
LUT: 324 (0.04 %) BRAM: N/A

URAM: N/A

Register: 169 (N/A)

DSP: N/A

Profile Summary:



Kernel	Enqueues	Total time	Min time	Avg time	Max time
Cholesky_kernel	1	1.436	1.436	1.436	1.436

Performance estimation:

Timing:
* Summary:

i Clock	İ	Target İ	Estimated	Uncertainty
lap_clk	İ	3.33 nsi	2.433 nsl	 0.90 nsl

Latency: * Summary:

+	(cycles)	l Latency	(absolute) i	Inte	rval İ	Pipelinel
	max	min	max	min 1	max I	Type l
; +	i ?	j ?	i ?i	i ?i	? İ	nol

Utilization estimation:

= Utilization Estimates

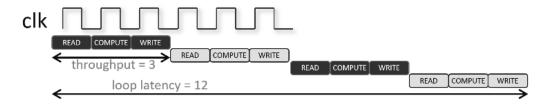
* Summary:					
Name	BRAM_18K	DSP	FF	LUT	URAMİ
IDSP Expression FIFO Instance Memory Multiplexer Register	- - - 4 32 -		- 0 - 3561 0 - 1071	01 2144	-1 01 01
Total	36	201	4632	55661	01
Available SLR	1344	2976	871680	435840	320
Utilization SLR (%)	2	~0	~0	1	0]
Available	2688	59521	1743360	871680	640
Utilization (%)	 1 	~0	~0	~0 	01

RTL Ports	Dir	Bits	Protocol I	Source Object	C Type
s_axi_control_AWVALID	in i	11	s_axi	control	scalar
s_axi_control_AWREADY	louti	11	s_axil	controll	scalari
s_axi_control_AWADDR	l in l	61	s_axil	controll	scalari
s axi control WVALID	l inl	11	s axil	controll	scalari
s_axi_control_WREADY	l outl	11	s_axil	controll	scalari
s_axi_control_WDATA	l inl	321	s_axil	controll	scalari
s_axi_control_WSTRB	l inl	41	s_axil	controll	scalari
s_axi_control_ARVALID	l inl	11	s_axil	controll	scalari
s_axi_control_ARREADY	l outl	11	s_axil	controll	scalari
s_axi_control_ARADDR	in	61	s_axil	control	scalari
s_axi_control_RVALID	l out		s_axil	controll	scalari
s_axi_control_RREADY	in		s_axil	control	scalari
s_axi_control_RDATA	out		s_axil	control	scalari
s_axi_control_RRESP	l out!		s_axil	controll	scalar
s_axi_control_BVALID	l out!		s_axil	controll	scalar
s_axi_control_BREADY	l in l		s_axi!	control	scalar
s_axi_control_BRESP	out		s_axil	controll	scalar
ap_local_block	l out!		ap_ctrl_chain	cholesky_kernell	
ap_clk	l in l		ap_ctrl_chain	cholesky_kernell	return valuel
ap_rst_n	l in l		ap_ctrl_chain	cholesky_kernell	return valuel
interrupt	out		ap_ctrl_chainļ	cholesky_kernel	return valuel
m_axi_gmem_AWVALID	out	11	m_axil	gmeml	pointer
m_axi_gmem_AWREADY	l in l	11	m_axil	gmeml	pointer
m_axi_gmem_AWADDR	l out		m_axil	gmeml	pointer
m_axi_gmem_AWID	l out		m_axil	gmeml	pointer
m_axi_gmem_AWLEN	l outl		m_axil	gmeml	pointerl
m_axi_gmem_AWSIZE	l outl		m_axil	gmeml	pointer
m_axi_gmem_AWBURST	l outl		m_axil	gmeml	pointer
m_axi_gmem_AWLOCK	l outl		m_axil	gmeml	pointerl
m_axi_gmem_AWCACHE	l outl		m_axil	gmeml	pointerl
m_axi_gmem_AWPROT	l outl I outl		m_axil	gmeml	pointerl
m_axi_gmem_AWQOS m_axi_gmem_AWREGION	ı outi I outl		m_axil	gmeml	pointerl
m_axi_gmem_awabuton m_axi_gmam_bmuced	ı outi	11	m_axil	gmeml	pointerl pointerl
m_axi_gmem_AVUSER	ı outi I outl	11	m_axil	gmeml	pointeri
m axi gmem WVALID	ι υμί	11	m axil	gmeml	ротисетт

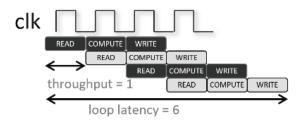
lm_axi_gmem_WREADY	l inl	11	m_axil	gmeml	pointerl
lm_axi_gmem_WDATA	l outl	641	m_axil	gmeml	pointerl
lm_axi_gmem_WSTRB	l outl	81	m_axil	gmeml	pointerl
lm_axi_gmem_WLAST	l outl	11	m_axil	gmeml	pointerl
lm_axi_gmem_WID	l outl	11	m_axil	gmeml	pointerl
lm_axi_gmem_WUSER	l outl	11	m_axil	gmeml	pointerl
lm_axi_gmem_ARVALID	l outl	11	m_axil	gmeml	pointerl
lm_axi_gmem_ARREADY	l inl	11	m_axil	gmeml	pointerl
lm_axi_gmem_ARADDR	l outl	641	m_axil	gmeml	pointerl
lm_axi_gmem_ARID	l outl	11	m_axil	gmeml	pointerl
lm_axi_gmem_ARLEN	l outl	81	m_axil	gmeml	pointerl
lm_axi_gmem_ARSIZE	l outl	31	m_axil	gmeml	pointerl
lm_axi_gmem_ARBURST	l outl	21	m_axil	gmeml	pointerl
lm_axi_gmem_ARLOCK	l outl	21	m_axil	gmeml	pointerl
lm_axi_gmem_ARCACHE	l outl	41	m_axil	gmeml	pointerl
lm_axi_gmem_ARPROT	l outl	31	m_axil	gmeml	pointerl
lm_axi_gmem_ARQOS	l outl	41	m_axil	gmeml	pointerl
lm_axi_gmem_ARREGION	l outl	41	m_axil	gmeml	pointerl
lm_axi_gmem_ARUSER	l outl	11	m_axil	gmeml	pointerl
lm_axi_gmem_RVALID	l inl	11	m_axil	gmeml	pointerl
lm_axi_gmem_RREADY	l outl	11	m_axil	gmeml	pointerl
lm_axi_gmem_RDATA	l inl	64 I	m_axil	gmeml	pointerl
lm_axi_gmem_RLAST	l inl	11	m_axil	gmeml	pointerl
lm_axi_gmem_RID	l inl	11	m_axil	gmeml	pointerl
lm_axi_gmem_RUSER	l inl	11	m_axil	gmeml	pointerl
lm_axi_gmem_RRESP	l inl	21	m_axil	gmeml	pointerl
lm_axi_gmem_BVALID	l inl	11	m_axil	gmeml	pointerl
lm_axi_gmem_BREADY	l outl	11	m_axil	gmeml	pointerl
lm_axi_gmem_BRESP	l inl	21	m_axil	gmeml	pointerl
lm_axi_gmem_BID	l inl	11	m_axil	gmeml	pointerl
lm_axi_gmem_BUSER	l inl	11	m_axil	gmeml	pointerl

IV. Module 2 Pipeline

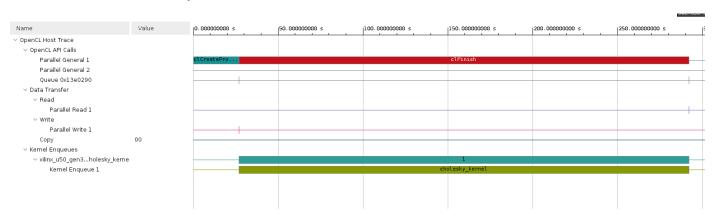
This module is meant to focus on the pipeline pragma and go through the description below. The kernel source code with the loops annotated with the pragma will produce the same results as in module 1, that's because since simple loops and inner loops (for nested loops) are automatically pipelined by the tool. Before pipeline:



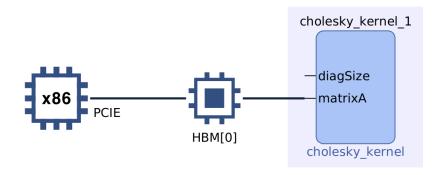
After pipeline:



Run summary:



System diagram:



Estimated Resources

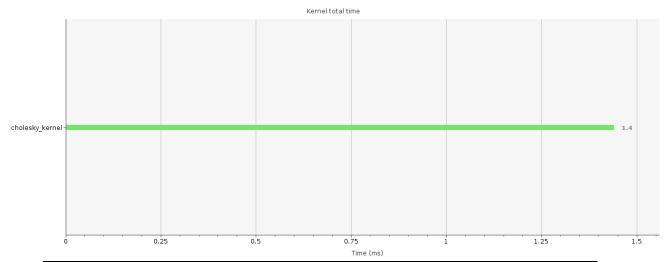
LUT: 324 (0.04 %) BRAM: N/A

URAM: N/A

Register: 201 (N/A)

DSP: N/A

Profile Summary:



Kernel	Enqueues	Total time	Min time	Avg time	Max time
Cholesky_kernel	1	1.436	1.436	1.436	1.436

Performance estimation:

+ Timing: * Summary: +----+ +----+

+ Latency: * Summary:

İ	Latency min	(cycles) max	Latency (min	+ (absolute) max +	Inte min	rval İ max I	Pipelinel Type
Ì	?	?	i ?i	? 	i ?i	? [nol

Utilization estimation:

= Utilization Estimates

*	Summa	ry	
---	-------	----	--

+		DSP	FF		URAMI
IDSP Expression FIFO Instance Memory Multiplexer Register	- - - 4 32 -	- - 20 - - - -	- 0 3561 0 1135	l 01 2144	-1 01 01
Total	36	201	4696	5566	0[
Available SLR	1344	2976	871680	435840	320
Utilization SLR (%)	21	~0	~0	1	0
Available	26881	5952	1743360	871680	640
Utilization (%)		~0	~0 		01 +

+	-	+	+	+ -	+
RTL Ports	l Dir İ	Bitsl	Protocol	Source Object	C Type İ
+	 +	+	+	·+·	+
ls_axi_control_AWVALID	l in l	11	s_axil	controll	scalari
ls_axi_control_AWREADY	l outl	11	s_axil	controll	scalari
ls_axi_control_AWADDR	l in l	61	s_axil	controll	scalari
ls_axi_control_WVALID	l in l	11	s_axil	controll	scalari
ls_axi_control_WREADY	l outl	11	s_axil	controll	scalari
ls_axi_control_WDATA	l in l	321	s_axil	control	scalari
ls_axi_control_WSTRB	inl	41	s_axil	control	scalar
[s_axi_control_ARVALID	l in l	11	s_axil	control	scalari
s_axi_control_ARREADY	out	11	s_axil	control	scalari
ls_axi_control_ARADDR	inl	61	s_axil	control	scalari
[s_axi_control_RVALID	out	11	s_axil	control	scalar
s_axi_control_RREADY	inl	11	s_axil	control	scalari
ls_axi_control_RDATA	out	321	s_axil	control	scalari
ls_axi_control_RRESP	out	21	s_axil	control	scalari
[s_axi_control_BVALID	outl	11	s_axil	control	scalar
s_axi_control_BREADY	inl	11	s_axil	control	scalari
ls_axi_control_BRESP	out	21	s_axil	controll	scalari
lap_local_block	outl	11	ap_ctrl_chain		return valuel
ap_clk	l in l	11	ap_ctrl_chain		return valuel
ap_rst_n	l in l	11	ap_ctrl_chain		return valuel
linterrupt	l outl	11	ap ctrl chainl	cholesky kernell	return valuel

=					
lm_axi_gmemO_AWVALID	l outl	11	m_axil	gmem01	pointerl
lm_axi_gmemO_AWREADY	l inl	11	m_axil	gmem01	pointerl
lm_axi_gmemO_AWADDR	l outl	641	m_axil	gmem01	pointerl
lm_axi_gmemO_AWID	l outl	11	m_axil	gmem01	pointerl
lm_axi_gmemO_AWLEN	out	81	m_axil	gmem01	pointerl
lm_axi_gmemO_AWSIZE	l outl	31	m_axil	gmem01	pointerl
lm_axi_gmemO_AWBURST	l outl	21	m_axil	gmem01	pointerl
lm_axi_gmemO_AWLOCK	out	21	m_axil	gmem01	pointerl
lm_axi_gmemO_AWCACHE	l outl	41	m_axil	gmem01	pointerl
lm_axi_gmemO_AWPROT	l outl	31	m_axil	gmem01	pointerl
lm_axi_gmemO_AWQOS	l outl	41	m_axil	gmem01	pointerl
lm_axi_gmemO_AWREGION	out	41	m_axil	gmem01	pointerl
lm_axi_gmemO_AWUSER	out	11	m_axil	gmem01	pointerl
lm_axi_gmemO_WVALID	l outl	11	m_axil	gmem01	pointerl
lm_axi_gmemO_WREADY	l inl	11	m_axil	gmem01	pointerl
lm_axi_gmemO_WDATA	out	641	m_axil	gmem01	pointerl
lm_axi_gmemO_WSTRB	out	81	m_axil	gmem01	pointerl
lm_axi_gmemO_WLAST	out	11	m_axil	gmem01	pointerl
lm_axi_gmemO_WID	l outl	11	m_axil	gmem01	pointerl
lm_axi_gmemO_WUSER	out	11	m_axil	gmem01	pointerl
lm_axi_gmemO_ARVALID	l outl	11	m_axil	gmem01	pointerl
lm_axi_gmemO_ARREADY	l inl	11	m_axil	gmem01	pointerl
lm_axi_gmemO_ARADDR	out	64 I	m_axil	gmem01	pointerl
lm_axi_gmemO_ARID	out	11	m_axil	gmem01	pointerl
lm_axi_gmemO_ARLEN	l outl	81	m_axil	gmem01	pointerl
lm_axi_gmemO_ARSIZE	l outl	31	m_axil	gmem01	pointerl
lm_axi_gmem0_ARBURST	out	21	m_axil	gmem0	pointerl
lm_axi_gmemO_ARLOCK	l outl	21	m_axil	gmem01	pointerl
lm_axi_gmemO_ARCACHE	out	41	m_axi	gmem0	pointerl
lm_axi_gmemO_ARPROT	out	31	m_axi	gmem01	pointerl
lm_axi_gmem0_ARQOS	out	41	m_axi	gmem01	pointerl
lm_axi_gmemO_ARREGION	out	41	m_axi!	gmemOl	pointer
lm_axi_gmemO_ARUSER	out	11	m_axi!	gmem0	pointer
lm_axi_gmemO_RVALID	l in l	11	m_axi!	gmem0	pointer
lm_axi_gmemO_RREADY	out	11	m_axi!	gmem0	pointer
lm_axi_gmemO_RDATA	l in	641	m_axi!	gmem01	pointer
lm_axi_gmem0_RLAST	l in	11	m_axi!	gmem0	pointer
lm_axi_gmemO_RID	l in l	11	m_axi!	gmemOl	pointer
lm_axi_gmem0_RUSER	l in l	11	m_axi!	gmem0	pointer
lm_axi_gmem0_RRESP	l in	21	m_axi!	gmem0	pointer
lm_axi_gmemO_BVALID	l inl	11	m_axi!	gmemOl	pointer
lm_axi_gmemO_BREADY	out	11	m_axi!	gmemOl	pointer
lm_axi_gmem0_BRESP	l in	21	m_axi!	gmemOl	pointer
lm_axi_gmem0_BID	l in	11	m_axi!	gmem0	pointer
lm_axi_gmem0_BUSER	l inl	11	m_axil	gmem01	pointerl
· ·				1	

V. Module 3 datatype

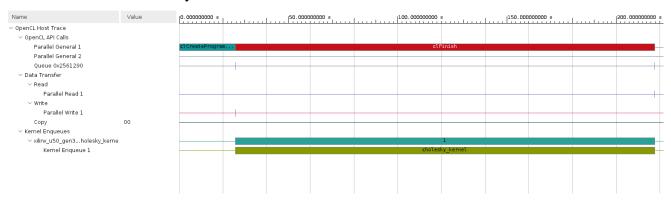
In this module both the kernel and host code are modified to use 32-bit floating point data types(float) instead of the 64-bit floating point(double) to show the performance and Xilinx utilization beneficial impact of downsizing data types.

Kernel Resources Used (regular floating point versus double):

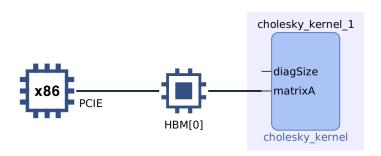
Name	LUT	LUTAsMem	REG	BRAM	DSP
Kernel with double	10190	799	10191	514	18
Kernel with float	5071	746	5124	258	12

As expected, the resource utilization comes down across for both logic and storage

Run summary:



System diagram:



Estimated Resources

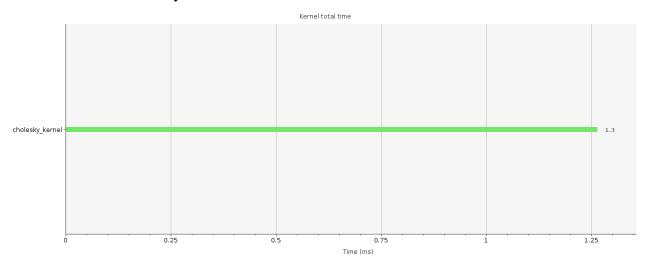
LUT: 324 (0.04 %) BRAM: N/A

URAM: N/A

Register: 169 (N/A)

DSP: N/A

Profile Summary:



Kernel	Enqueues	Total time	Min time	Avg time	Max time
Cholesky_kernel	1	1.264	1.264	1.264	1.264

Performance estimation:

= Performance Estimates

+ Timing: * Summary:

i Clock	İ	Target İ	Estimatedi	+ Uncertainty +
lap_clk	İ	3.33 nsi	2.433 nsl	0.90 nsl

+ Latency: * Summary:

Ì	Latency	(cycles)	++ Latency (min	absolute)	Inte	rval İ	Pipelinel
+-	+		++	+	+	+	+
-		-	! ?!				
+-	+		+	·	+	+	+

Utilization estimation:

 II + i 1	inoti.	w Dat	imates
UT. 1 I	128T.10) III - E.S.T.	imates

* Summary:					
Name	BRAM_18K	DSP	FF	LUT	URAMI
IDSP Expression FIFO Instance Memory Multiplexer Register	- - - 2 16 -		- 0 - 2496 0 - 930	01 1820	-1 01 01
Total	18	14	3426	4502	0
Available SLR	1344	2976	871680	435840	320
Utilization SLR (%)	1	~0	~0	11	01
Available	2688	59521	1743360	871680	640
Utilization (%)		~01	~0	~01	01

RTL Ports	Dir	Bitsl	Protocol	Source Object	С Туре І
+ s_axi_control_AWVALID	in	11	 s_axil	controll	 scalarl
s_axi_control_AWREADY	out		s_axil	controll	scalari
s_axi_control_AWADDR	inl		s_axil	controll	scalari
s_axi_control_WVALID	inl		s_axil	controll	scalari
s_axi_control_WREADY	out		s_axil	controll	scalari scalari
s_axi_control_WDATA	inl			controll	scalari scalari
			s_axil		
s_axi_control_WSTRB	inl		s_axil	controll	scalari
s_axi_control_ARVALID	inl		s_axil	controll	scalari
s_axi_control_ARREADY	out		s_axil	controll	scaļarļ
s_axi_control_ARADDR	in		s_axil	control	scalari
s_axi_control_RVALID	out		s_axil	control	scalari
s_axi_control_RREADY	inl		s_axil	controll	scalari
s_axi_control_RDATA	outl		s_axil	controll	scalarl
s_axi_control_RRESP	outl	21	s_axil	controll	scalarl
s_axi_control_BVALID	outl	11	s_axil	controll	scalarl
s_axi_control_BREADY	inl		s_axil	controll	scalarl
s_axi_control_BRESP	outl		s_axil	control	scalari
ap_local_block	out		ap_ctrl_chain	cholesky_kernell	return valuel
ap_clk	inl		ap_ctrl_chain	cholesky_kernell	return valuel
	inl		ap_ctrl_chain ap_ctrl_chain		return valuel
ap_rst_n					return valuel
interrupt	out		ap_ctrl_chain		
m_axi_gmem0_AWVALID	outl		m_axil	gmem01	pointerl
m_axi_gmemO_AWREADY	inl		m_axil	gmemOl	pointer
m_axi_gmemO_AWADDR	out		m_axil	gmem0	pointer
m_axi_gmemO_AWID !	out		m_axil	gmem0	pointer
m_axi_gmemO_AWLEN	outl		m_axil	gmemOl	pointerl
m_axi_gmemO_AWSIZE	outl		m_axil	gmem01	pointerl
m_axi_gmemO_AWBURST	outl	21	m_axil	gmem01	pointerl
m_axi_gmemO_AWLOCK	outl		m_axil	gmem01	pointerl
m_axi_gmemO_AWCACHE	outl		m_axil	gmem01	pointerl
m_axi_gmemO_AWPROT	out		m_axil	gmem01	pointerl
m_axi_gmemO_AWQOS	out		m_axil	gmemOl	pointerl
m_axi_gmemO_AWREGION	out			gmem01	
			m_axil		pointerl
m_axi_gmemO_AWUSER	out		m_axil	gmem01	pointerl
m_axi_gmemO_WVALID	out		m_axil	gmemOl	pointerl
m_axi_gmemO_WREADY	inl		m_axil	gmemOl	pointerl
m_axi_gmemO_WDATA	out		m_axil	gmem0	pointer
m_axi_gmemO_WSTRB	out		m_axil	gmem0	pointerl
m_axi_gmem0_WLAST	out		m_axil	gmem0	pointerl
m_axi_gmemO_WID	outl		m_axil	gmem01	pointerl
m_axi_gmemO_WUSER	outl	11	m_axil	gmem01	pointerl
m_axi_gmemO_ARVALID	outl		m_axil	gmem01	pointerl
m axi gmemO ARREADY	inl		m axil	gmem01	pointerl
m_axi_gmemO_ARADDR	out		m_axil	gmem01	pointerl
m_axi_gmemO_ARID	out		m_axil	gmemOl	pointerl
m_axi_gmemO_ARLEN	out		m_axil	gmem01	pointerl
m_axi_gmemO_ARSIZE	out			gmemor gmemOl	
) I	m_axil		pointerl
m_axi_gmemO_ARBURST	out		m_axil	gmem01	pointerl
m_axi_gmemO_ARLOCK	out		m_axil	gmemOl	pointerl
m_axi_gmemO_ARCACHE	out		m_axil	gmemOl	pointer
m_axi_gmemO_ARPROT !	out		m_axil	gmemOl	pointer
m_axi_gmemO_ARQOS	out		m_axil	gmem0	pointerl
m_axi_gmemO_ARREGION	outl		m_axil	gmem01	pointerl
m_axi_gmemO_ARUSER	outl	11	m_axil	gmem01	pointerl
m_axi_gmemO_RVALID	in		m_axil	gmemOl	pointer
m_axi_gmemO_RREADY	out		m_axil	gmem01	pointerl
m_axi_gmemO_RDATA	inl		m_axil	gmemOl	pointerl
m_axi_gmemO_RLAST	in		m_axil	gmem01	pointer
m_axi_gmemO_RID	in		m_axil	gmem01	pointer
m_axi_gmemO_RUSER	inl		m_axil	gmem01	pointer
m_axi_gmemO_RGESP I					
m_axi_gmemU_RKEAF	inl	4	m_axil	gmemOl	pointerl
m_axi_gmemO_BVALID	in		m_axil	gmem01	pointer
m_axi_gmemO_BREADY !	out		m_axil	gmemOl	pointer
m_axi_gmemO_BRESP	in		m_axil	gmemOl	pointer
		1.1	1	~~~~\\	
m_axi_gmemO_BID m_axi_gmemO_BUSER	inl	1 1	m_axil	gmemOl gmemOl	pointerl

VI. Module 4 dataflow

The **DATAFLOW** pragma enables task-level pipelining, allowing functions and loops to overlap in their operation, increasing the concurrency of the register transfer level(RTL) implementation, and increasing the overall throughput of the design.

When the **DATAFLOW** pragma is specified, the HLS tool analyzes the data flow between sequential functions or loops and creates channels (based on ping pong RAMs or FIFOs) that allow consumer functions or loops to start operation before the producer functions or loops have completed. This allows functions or loops to operate in parallel, which decreases latency and improves the throughput of the RTL.

```
Specifies DATAFLOW optimization within the loop wr_loop_j.

wr_loop_j: for (int j = 0; j < TILE_PER_ROW; ++j) {
    #pragma HLS DATAFLOW

    wr_buf_loop_m: for (int m = 0; m < HEIGHT; ++m) {
        wr_buf_loop_n: for (int n = 0; n < WIDTH; ++n) {
            #pragma HLS PIPELINE
            // should burst WIDTH in WORD beat
            outFifo >> tile[m][n];
        }
    }
    wr_loop_m: for (int m = 0; m < HEIGHT; ++m) {
        wr_loop_n: for (int n = 0; n < WIDTH; ++n) {
            #pragma HLS PIPELINE
            outx[HEIGHT*TILE_PER_ROW*WIDTH*i+TILE_PER_ROW*WIDTH*m+WIDTH*j+n] = tile[m][n];
        }
    }
}</pre>
```

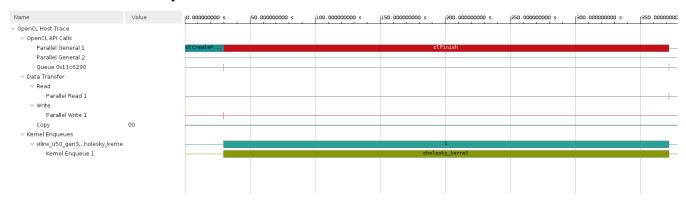
上圖為範例,在 function 中加入#pragma HLS DATAFLOW 讓 HLS 能優化 其迴圈。

在 module4 中,新設了一個變數 NCU, 去表示平行運算的數量, NCU 會被傳到 chol col wrapper 中,以下程式會呼叫 chol col 16 次

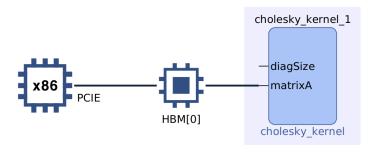
```
template <typename T, int N, int NCU>
void chol_col_wrapper(int n, T dataA[NCU][(N + NCU - 1) / NCU][N], T dataj[NCU][N], T tmp1, int j)
{
    #pragma HLS DATAFLOW

Loop_row:
    for (int num = 0; num < NCU; num++)
    {
    #pragma HLS unroll factor = NCU
        chol_col<T, N, NCU>(n, dataA[num], dataj[num], tmp1, num, j);
    }
}
```

Run summary:



System diagram:



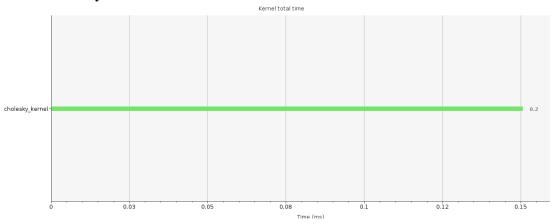
Estimated Resources

LUT: 344 (0.04 %) BRAM: N/A URAM: N/A

Register: 213 (N/A)

DSP: N/A

Profile summary:



Kernel	Enqueues	Total time	Min time	Avg time	Max time
Cholesky_kernel	1	0.151	0.151	0.151	0.151

Performance estimation:

= Performance Estimates

+ Timing: * Summary:

			+ Uncertaintyl
lap_clk i	3.33 ns	2.433 nsi	0.90 nsl

+ Latency: * Summary:

İ	Latency min	(cycles) max	l Latency min	+ (absolute) max +	Inte min	rval i max l	Pipelinel Type l
İ	?	?	j ?	 	i ?i	?i	nol

Utilization estimation:

= Utilization Estimates

* Summary:					
Name	BRAM_18KI	DSP	FF	LUT	URAMI
IDSP Expression FIFO Instance Memory Multiplexer Register	- - - 4 32 -		- 0 - 81560 0 - 631	l 01 I 31891	-1 01 641
Total	361	196	82191	63505	641
Available SLR	1344	2976	871680	435840	320
Utilization SLR (%)	21	6	9	14	201
Available	2688	5952	1743360	871680	640
Utilization (%)		3		7 7 7 7 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1	101

= Interface

* Summary:	, ,				
				Source Object	
s_axi_control_AWVALID	i ni	11	s_axil	control	scalar
s_axi_control_AWREADY			s_axil	controll	
s_axi_control_AWADDR	l inl		s_axil	control	
s_axi_control_WVALID	l inl		s_axil		scalari
s_axi_control_WREADY s_axi_control_WDATA	l outl I inl		s_axil s axil		scaları scaları
s_axi_control_wDTRB	inl		s_axi s_axi	control	scalar scalar
s_axi_control_ARVALID	i inl		s_axil	control	scalar
s_axi_control_ARREADY	l outl		s_axil	control	scalar
s_axi_control_ARADDR	l inl		s_axil	controll	scalar
s_axi_control_RVALID	l outl		s_axil	control	scalar
s_axi_control_RREADY	l inl		s_axil		scalar
s_axi_control_RDATA	l outl		s_axil	controll	scalari
s_axi_control_RRESP s_axi_control_BVALID	l outl I outl		s_axil s_axil	controll controll	scalar scalar
s_axi_control_BREADY	l inl		s_axi s_axi		scalar scalar
s_axi_control_BRESP	l outl		s_axil		scalar
ap_local_block	loutl		ap_ctrl_chain		
ap_clk	l inl	11	ap_ctrl_chain		return value
ap_rst_n	l in l		ap_ctrl_chain		
interrupt	l outl		ap_ctrl_chain	cholesky_kernell	return value
m_axi_gmemO_AWVALID	l outl		m_axil		pointer
m_axi_gmemO_AWREADY	l inl Loutl		m_axil	gmem0l	pointer
m_axi_gmemO_AWADDR m_axi_gmemO_AWID	l outl		m_axil m axil	gmem0 gmem0	pointer pointer
m_axi_gmemO_AWLEN	l outl		m_axi	gmem01	pointer
m_axi_gmemO_AWSIZE	l outl		m_axil	gmem01	pointer
m_axi_gmemO_AWBURST	l outl	21	m_axil	gmemOl	pointer
m_axi_gmemO_AWLOCK	l outl	21	m_axil	gmem01	pointer
m_axi_gmemO_AWCACHE	out		m_axil	gmem0	pointer
m_axi_gmemO_AWPROT	l outl		m_axil	gmem0	pointer
m_axi_gmemO_AWQOS	l outl		m_axil	gmem0l	pointer
m_axi_gmemO_AWREGION m_axi_gmemO_AWUSER	l outl I outl		m_axil m_axil	gmem0 gmem0	pointer pointer
m_axi_gmemO_WVALID	l outl	7.1	m_axi	gmem01	pointer
m_axi_gmemO_WREADY	in l		m_axil	gmem01	pointer
m_axi_gmemO_WDATA	l outl		m_axil	gmemOl	pointer
m_axi_gmemO_WSTRB	l outl	81	m_axil	gmem01	pointer
m_axi_gmemO_WLAST	l outl		m_axil	gmemOl	pointer
m_axi_gmemO_WID	l outl		m_axil	gmem01	pointer
m_axi_gmemO_WUSER m_axi_gmemO_ARVALID	l outl I outl		m_axil	gmemOl	pointer
m_axi_gmemO_ARVADID m_axi_gmemO_ARREADY	l outi		m_axil m_axil	gmem0 gmem0	pointer pointer
					- ·
m_axi_gmemO_ARADDR	l outl		m_axil	gmem01	pointer
m_axi_gmemO_ARID m_axi_gmemO_ARLEN	l outl I outl		m_axil m_axil	gmemOl gmemOl	pointer pointer
m_axi_gmemO_ARSIZE	l outl		m_axil m_axil	gmemor gmemOl	pointer
m_axi_gmemO_ARBURST	l outl		m_axi	gmem01	pointer
m_axi_gmemO_ARLOCK	l outl		m_axil	gmem01	pointer
m_axi_gmemO_ARCACHE	l outl	41	m_axil	gmemOl	pointer
m_axi_gmemO_ARPROT	out	31	m_axil	gmem0	pointer
m_axi_gmemO_ARQOS	l outl		m_axil	gmem0	pointer
m_axi_gmemO_ARREGION	l outl		m_axil	gmemOl	pointer
m_axi_gmemO_ARUSER m_axi_gmemO_RVALID	l outl I inl		m_axil m_axil	gmem0 gmem0	pointer pointer
m_axi_gmemO_RVALID m_axi_gmemO_RREADY	ı ını I outl		m_axil m_axil	gmemor gmemol	pointer pointer
m_axi_gmemO_RDATA	l inl		m_axil	gmemol gmemOl	pointer
m_axi_gmemO_RDHIH m axi gmemO RLAST	in in i		m_axil	gmem01	pointer
m_axi_gmemO_RID	i inl	ĨΪ	m_axil	gmem01	pointer
m_axi_gmemO_RUSER	l in l	11	m_axil	gmem01	pointer
m_axi_gmemO_RRESP	l in l	21	m_axil	gmem0	pointer
m_axi_gmemO_BVALID	l in l	11	m_axil	gmem0	pointer
m_axi_gmemO_BREADY	l outl	11	m_axil	gmem01	pointer
m_axi_gmemO_BRESP	l inl	21 11	m_axil m_avil	gmemOl	pointer
m_axi_gmemO_BID m_axi_gmemO_BUSER	l inl I inl	11	m_axil m_axil	gmem0 gmem0	pointer pointer
				S III C III O I	001015

Result Summary:

Module	CPU	Module1	Moduel2	Module3	Module4
Exe. Time	21461	793950	793732	536784	11698
Speed up(cpu)	1	0.03x	0.03x	0.04x	1.83x
Speed up	N/A	1	1	1.48x	68x

心得:這次的 Lab.實驗需要我們自己按照 github 上面的 Tutorial 來進行操作,但實際操作起來還是碰到非常多的問題,例如:我一開始有遇到在 make file 時,需要輸入"export LC_ALL="C""的這個指令來解決,但在學校的伺服器中並沒有 export 這個指令,請教助教後,將這項指令加在 makefile 裡面就可以解決了,也遇到過 vitis 的版本問題,這時候也是在 makefile 裡面加上-version 2.0.1 就可以解決,最後我在按照 github 執行過程中,並不能像上面一樣使用 vitis GUI 來實驗,因為所給的壓縮檔中並沒有 cholesky_host 檔,有請教助教後,發現 test.cpp 應該就是 host 檔,但我是選擇第二個實驗方法,直接在 terminal 中 make run,就可以解決這個問題,在經過這個 Lab 之後,對於 vitis 的操作有了更深的了解,也清楚知道 baseline、pipeline、datatype、dataflow 之間的差異與 vitis 能帶來的加速方便,相信在之後的課程中會對 vitis 有更多的認識。

Github: https://github.com/hank871116/HLS LAB B 5