HLS LabC: Vitis Solver Library

LU Decomposition Without Pivoting

Team 3

Team member: 麥智詠、賴志杰

Date: 2022/05/01

https://github.com/CHIHCHIEH-LAI/HLS/tree/main/Vitis_Solver_Library

目錄

Background Introduction

$$A = LU \qquad \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

LU 分解就是高斯消去法的一種表達式，矩陣 L 紀錄消去法化簡 A 的過程，矩陣 U 則儲存化簡結果。LU 分解的外表看似平淡無奇，但它常常被電腦用來解線性方程，逆矩陣與計算行列式，堪稱是最具實用價值的矩陣分解式之一。

以下為 3X3 矩陣 A 的 LU 分解步驟：

$$A = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{bmatrix}$$

先將矩陣 A[0][0]以下的所有元素變成 0，即

$$\begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{bmatrix} \xrightarrow{\text{subtract } 2 \times \text{ row } 1 \text{ from row } 2} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ -3 & 1 & 1 \end{bmatrix}$$

$$\xrightarrow{\text{subtract } -3 \times \text{ row } 1 \text{ from row } 3} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 7 & -2 \end{bmatrix}$$

最終就會得到 U 矩陣，而 L 為剛剛用來相乘的數字

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -\frac{7}{3} & 1 \end{bmatrix}.$$

Code Explaination

Kernel Function

```
template <typename T, int NMAX, int NCU>
void getrf_nopivot(int n, T* A, int lda, int& info)
```

從圖中可見，Kernel function 包含 3 個硬體參數與 3 個函數參數。

typename T: 矩陣的數值型態，如 int、double…等。

int NMAX: 矩陣的長邊，即取 Row Size、Column Size 較大者。

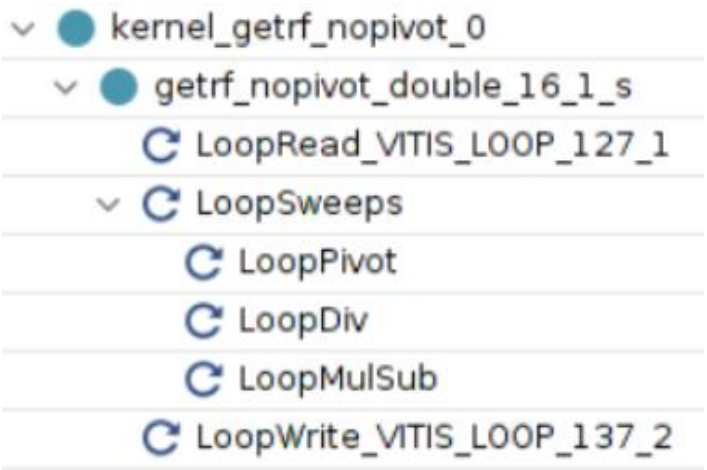
int NCU: 運算單元數量。

Int n: rows/cols 的數量

T* A: 輸入矩陣、輸出矩陣。A 的資料讀入運算後會將 L 與 U 寫回到 A，因此 A 為輸入與輸出。

int lda: 即 Row Size。例如(i,j)的地址為 i*lda + j。

Kernel Function Hierarchy



如右圖所示，整個 Function 結構首先區分成三個部分，分別是 Read、Sweeps 以及 Write。

Read 讀入資料，Sweeps 做 LU 分解運算，Write 寫回資料。

其中 Sweeps 又可再分為 Pivot、Div、MulSub。

Pivot 取出 master row，Div 算出下方每個 row 要減去的倍率，MulSub 將下方的每個 row 依其倍率減去 master row。

以下以 3x3 矩陣為例：$\begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{bmatrix}$

LoopSweeps: for(int s=0 ; s < 3-1 ; s++) →s=0、1

S=0:

Pivot 取出 master row: $[1 \quad 2 \quad -1]$。

Div 算出下方每個 row 要減去的倍率：$\begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{bmatrix}$

MulSub 將下方的每個 row 依其倍率減去 master row: $\begin{bmatrix} 1 & 2 & -1 \\ 2 & -3 & 0 \\ -3 & 7 & -2 \end{bmatrix}$

S=1:

Pivot 取出 master row: $[1 \quad -3 \quad 0]$。

Div 算出下方每個 row 要減去的倍率：$\begin{bmatrix} 1 & 2 & -1 \\ 2 & -3 & 0 \\ -3 & -\frac{7}{3} & -2 \end{bmatrix}$

MulSub 將下方的每個 row 依其倍率減去 master row: $\begin{bmatrix} 1 & 2 & -1 \\ 2 & -3 & 0 \\ -3 & -\frac{7}{3} & -2 \end{bmatrix}$

$\begin{bmatrix} 1 & 2 & -1 \\ 2 & -3 & 0 \\ -3 & -\frac{7}{3} & -2 \end{bmatrix}$即為要準備寫回 A 的運算結果，其中 L = $\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -\frac{7}{3} & 1 \end{bmatrix}$、U = $\begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{bmatrix}$。

Parallelization

```
LoopRead:
    for (int r = 0; r < n; r++) {
        for (int c = 0; c < n; c++) {
            matA[r % NCU][r / NCU][c] = A[lda * r + c];
        };
    };
```

將每一行 Interlaced 分割到不同的 Compute Unit。 如下圖示，CU1 為橘色、CU2 為藍色。

矩陣 A：

(1,1)	(1,2)	(1,3)	(1,4)
(2,1)	(2,2)	(2,3)	(2,4)
(3,1)	(3,2)	(3,3)	(3,4)
(4,1)	(4,2)	(4,3)	(4,4)

CU1：

(1,1)	(1,2)	(1,3)	(1,4)
(3,1)	(3,2)	(3,3)	(3,4)

CU2：

(2,1)	(2,2)	(2,3)	(2,4)
(4,1)	(4,2)	(4,3)	(4,4)

Findings from the Lab Work

一、Library 內建 Test function 無法有效測試 kernel function

原先的 test function 將隨機產生的 dataA 複製兩份到 dataC 與 dataD，將 dataA 丟進 kernel function 裡做 LU 分解後，最後比對 dataD 與 dataC 是否相同。

```
// Initialization of host buffers

const int MAXN = 16;
int inout_size = MAXN * MAXN;
double* dataA = aligned_alloc<double>(inout_size);
double* dataC = aligned_alloc<double>(inout_size);
double* dataD = aligned_alloc<double>(inout_size);

// Generate general matrix dataAN x dataAN
matGen<double>(dataAN, dataAN, seed, dataA);
for (int i = 0; i < inout_size; i++) {
    dataC[i] = dataA[i];
    dataD[i] = dataA[i];
}
```

```
// Calculate err between dataA and dataC
double errA = 0;
for (int i = 0; i < 1; i++) {
    for (int j = 0; j <= i; j++) {
        errA += (dataD[i * dataAM + j] - dataC[i * dataAM + j]) * (dataD[i * dataAM + j] - dataC[i * dataAM + j]);
    }
}
```

從中可以發現 dataD 與 dataC 都是最初 dataA 的副本，因此無論如何比對都會一樣，完全沒有比對到 dataA，因此無法有效檢測 kernel function 的正確性！

二、MulSub 為耗時較長的部分，且 Interval 無法降至 1

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
kernel_getrf_nopivot_0		10261	3.420E4		10262		no	60	2	11	~0	11848	~0	9324	1	0.00
getrf_nopivot_double_16_1_s		10259	3.419E4		10259		no	2	~0	11	~0	10252	~0	7489	~0	0.00
LoopRead_VITIS_LOOP_127_1		258	860.000	4	1	256	yes									
LoopSweeps		9600	3.200E4	640		15	no									
LoopPivot		18	59.994	4	1	1~16	yes									
LoopDiv		49	163.000	35	1	1~16	yes									
LoopMulSub		565	1.883E3	56	2	1~256	yes									
LoopWrite_VITIS_LOOP_137_2		259	863.000	5	1	256	yes									

```
LoopMulSub:
    for (unsigned int i = 0; i < nrows * ncols; i++) {
#pragma HLS pipeline
#pragma HLS dependence variable = A inter false
// clang-format off
#pragma HLS loop_tripcount min = 1 max = NCMAX*NRCU
        // clang-format on

        int r = i / ncols + rs;
        int c = i % ncols + cs + 1;

        A[r][c] = A[r][c] - A[r][cs] * pivot[c];
    };
}
```

```
#pragma HLS array_partition variable = matA complete
#pragma HLS resource variable = matA core = XPM_MEMORY uram
```

由於 matA 在一個 loop 內同時有讀寫兩個動作，因此需要增加 port，而 matA 的 partition 已經開滿了，因此只能更動 memory 型態。

Analysis

以下為針對不同 kernel function 中的 NCU 參數的 HLS synthesis 結果 (type=double, NRC = 16)

NCU = 1

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
kernel_getrf_nopivot_0		10261	3.420E4		10262		no	60	2	11	~0	11848	~0	9324	1	0.00
getrf_nopivot_double_16_1_s		10259	3.419E4		10259		no	2	~0	11	~0	10252	~0	7489	~0	0.00
LoopRead_VITIS_LOOP_127_1		258	860.000	4	1	256	yes									
LoopSweeps		9600	3.200E4	640		15	no									
LoopPivot		18	59.994	4	1	1~16	yes									
LoopDiv		49	163.000	35	1	1~16	yes									
LoopMulSub		565	1.883E3	56	2	1~256	yes									
LoopWrite_VITIS_LOOP_137_2		259	863.000	5	1	256	yes									

NCU = 4

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
kernel_getrf_nopivot_0		6107	2.036E4		6108		no	66	2	33	~0	11799	~0	9037	1	0.00
getrf_nopivot_double_16_4_s		6105	2.035E4		6105		no	8	~0	33	~0	10203	~0	7202	~0	0.00
rowUpdate_double_4_16_s		150	500.000		150		no	0	0	11	~0	1908	~0	1188	~0	
LoopMulSub		147	490.000	30	2	1~60	yes									
LoopRead_VITIS_LOOP_127_1		258	860.000	4	1	256	yes									
LoopSweeps		5445	1.815E4	363		15	no									
LoopPivot		18	59.994	4	1	1~16	yes									
LoopDiv		37	123.000	35	1	1~4	yes									
LoopWrite_VITIS_LOOP_137_2		260	867.000	6	1	256	yes									

NCU = 16

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
kernel_getrf_nopivot_0		2401	8.003E3		2402		no	90	3	176	2	34268	1	24941	2	0.00
getrf_nopivot_core_double_1_16_16_s		1741	5.803E3		1741		no	32	1	176	2	29063	1	19474	2	
rowUpdate_double_1_16_s		57	190.000		57		no	0	0	11	~0	1701	~0	1045	~0	
LoopMulSub		55	183.000	28	2	1~15	yes									
LoopSweeps		1740	5.799E3	116		15	no									
LoopPivot		18	59.994	4	1	1~16	yes									
LoopDiv		34	113.000	35	1	1	yes									
LoopRead_VITIS_LOOP_127_1		258	860.000	4	1	256	yes									
LoopWrite_VITIS_LOOP_137_2		260	867.000	6	1	256	yes									

有一點比較奇怪的是 NCU=1 時，rowUpdate 裡的 LoopMulSub 是在 LoopSweeps 裡，但是 NCU=4,16 時，rowUpdate 卻在 LoopSweeps 之外。

增加 NCU 以增加對 row 運算的平行度，讓整體 latency 從 10261 縮短成 6107 或 2401，但隨之而來的是硬體資源 DSP, FF, LUT 和 BRAM 的增加。

Suggestions for Improvement

一、修正 Test Function，以有效測試 kernel function

針對 findings 的第一點，我們自己寫了一個驗證的 code。dataA 經過 kernel function 後會將 L 和 U 兩個三角矩陣合併儲存在 dataA 裡面，而我們將 L 與 U 取出並相乘得到矩陣 LU，再驗算 L 相乘 U 是否與原本的矩陣相同，這樣就可以達到驗證 kernel function 的目的。

$$A = LU$$

```
// Calculate err between dataA and dataC
double errA = 0;
for(int r=0; r < dataAN; r++) {
    for(int c = 0; c < dataAN; c++) {
        errA += (LU_golden[r][c] - LU[r][c]) * (LU_golden[r][c] - LU[r][c]);
    }
}
errA = std::sqrt(errA) / (dataAM * dataAM);
```

```
double LU_golden[dataAN][dataAN];
double LU[dataAN][dataAN];
double L[dataAN][dataAN];
double U[dataAN][dataAN];
for (int r = 0; r < dataAN; r++) {
    for (int c = 0; c < dataAN; c++) {
        LU_golden[r][c] = dataC[dataAN * r + c];
        LU[r][c] = 0;
        L[r][c] = 0;
        U[r][c] = 0;
    }
}
for(int i=0; i<dataAN; i++) {
    L[i][i] = 1;
    U[i][i] = dataA[dataAN * i + i];
}
for (int r = 1; r < dataAN; r++) {
    for (int c = 0; c < r; c++) {
        L[r][c] = dataA[dataAN * r + c];
    }
}
for (int r = 0; r < dataAN-1; r++) {
    for (int c = r+1; c < dataAN; c++) {
        U[r][c] = dataA[dataAN * r + c];
    }
}
for (int i = 0; i < dataAN; i++) {
    for (int j = 0; j < dataAN; j++) {
        for (int k = 0; k < dataAN; k++) {
            LU[i][j] += L[i][k] * U[k][j];
        }
    }
}
```

二、更改 memory 資源配置以提高速度

針對 finding 第二點，我們將有關 matA 的 pragma 移除，原先以為會變成以 LUT 構成的儲存單元，然而資源清單中可見除了 BRAM 從 60 增加到 62 以外，其他資源消耗皆有所下降。因此推測 matA 的儲存方式變成 BRAM。最後解決 LoopMulSub interval 的問題後，整體速度提高 61%！

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
kernel_getrf_nopivot_0		6373	2.124E4		6374		no	62	2	11	~0	11017	~0	9075	1	0.00
LoopRead_VITIS_LOOP_127_1		257	857.000	3	1	256	yes									
LoopSweeps		5715	1.905E4	381		15	no									
LoopPivot		16	53.328	2	1	1~16	yes									
LoopDiv		48	160.000	34	1	1~16	yes									
LoopMulSub		309	1.030E3	55	1	1~256	yes									
LoopWrite_VITIS_LOOP_137_2		258	860.000	4	1	256	yes									