浙江大學

本科生课程作业

并行矩阵乘法设计

课程名称： 并行计算与多核编程

姓名： 陈卓

学院： 计算机科学与技术学院

专业： 计算机科学与技术

学号： 3170101214

指导老师： 楼学庆

2020 年 4 月 17 日

1 设计原理

本实验分别采用 Cannon 算法和 DNS 算法实现 4×4 矩阵乘法。

1.1 Cannon 算法

对于 $N \times N$ 矩阵乘法 $A \times B$，本实验 Cannon 算法采用**二维网孔结构**。步骤如下：

i) A_{ij} 向左循环移动 i 步；B_{ij} 向上循环移动 j 步[1]；各结点清零；

ii) 各结点做乘法运算并累加；

iii) A_{ij} 向左循环移动 1 步；B_{ij} 向上循环移动 1 步；

iv) 重复 $N-1$ 次。

1.2 DNS 算法

2 设计思路与实现

2.1 Cannon 算法

根据 1.1 中的描述，本实验将 4×4 矩阵分块为 16 个结点，每个结点处理两个数的乘法运算。

2.1.1 cannon_unit 模块

首先设计出表达每个结点的 `cannon_unit` 模块。

```
module cannon_unit(
 input wire clk,
 input wire rst,
 input wire en,
 input wire a_init[7:0],
 input wire b_init[7:0],
 input wire a_in[7:0],
 input wire b_in[7:0],
 output wire a_out[7:0],
 output wire b_out[7:0],
 output reg s[7:0]
);
```

其中 `rst`, `a_init`, `b_init` 用于步骤 i) 的初始化和清零；`a_in`, `b_in`, `a_out`, `b_out` 用于在二维网孔结构中连接其它结点；`s` 用于输出；`en` 用于计算完成后阻断后续计算。

在 `cannon_unit` 模块中有三个寄存器 `a`, `b`, `s`，分别用于保存 A, B 矩阵的元素和结果。在每个时钟周期内，寄存器 `a`, `b` 接收来自输入 `a_in`, `b_in` 的值，同时将二者乘积累加在寄存器 `s` 中；若 `rst` 置位，则 `s` 清零，`a`, `b` 接收 `a_init`, `b_init` 的值。

[1] $i, j \in [0, N)$

2.1.2 cannon 模块

`cannon` 模块是一个状态机。根据 Cannon 算法设计 4 个状态，分别是 `IDLE`, `LOAD`, `CALC`, `END`。状态转移如图 2.1 所示。

```
module cannon(
  input wire clk,
  input wire rst,
  input wire A[3:0][3:0][7:0],
  input wire B[3:0][3:0][7:0],
  output wire S[3:0][3:0][7:0],
  output reg finished
);
```

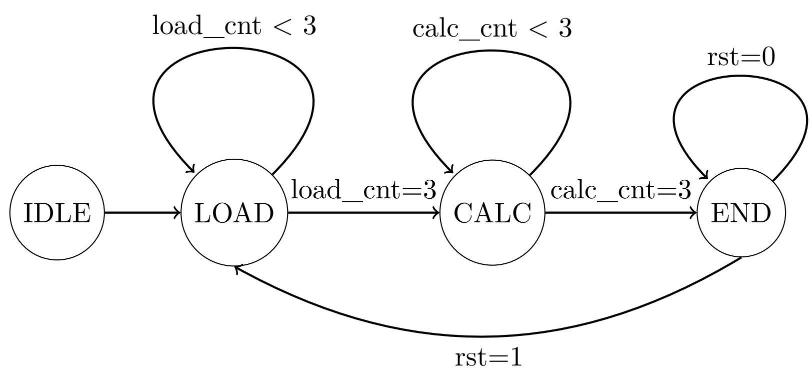

图 2.1: `cannon` 模块状态图

每个状态的动作如表 2.1 所示。

表 2.1: `cannon` 模块状态表

状态	描述
`IDLE`	模块初始化
`LOAD`	加载 A, B 矩阵，并初始化各结点
`CALC`	循环位移计算
`END`	计算结束

详细代码见 `cannon.v`。

3 评价

3.1 Cannon 算法

设 k^2 个结点，记单次传输时间 t_r，单次计算时间 t_w，由于步骤 ii) 和 iii) 同时进行，且通常 $t_r << t_w$，故总时间

$$T = (k-1)t_r + kt_w$$

此外，若单个结点中分块矩阵的乘法采用串行算法，则

$$\begin{aligned} T &= (k-1)O(1) + kO((\frac{n}{k})^3) \\ &= O(k) + O(\frac{n^3}{k^2}) \\ &= O(\sqrt{p}) + O(\frac{n^3}{p}) \\ &= O(\frac{n^3}{p}) \end{aligned}$$

在本实验中，每个结点仅负责一阶矩阵的运算，故 $T = O(\sqrt{p})$。

Cannon 算法的不足之处在于，数据初始化需要 $k-1$ 次的循环位移，在本实验中，由于规模不大，可以通过直接在代码中交换输入实现加速，但在大规模矩阵运算中仍然有这一方面的劣势。

3.2 DNS 算法

4 $N \times N$ 矩阵乘法设计

4.1 Cannon 算法

Cannon 算法是容易推广的——尽管本实验每个结点仅处理两个数的运算，但在实际使用中可以令每个结点处理两个小矩阵的运算，即 Cannon 算法描述中的**分块**。然而如此一来，如何对大矩阵进行分块，分块方式不同对运算复杂度是否有影响，就成为应该思考的问题。此外，本实验使用 verilog 设计 4×4 矩阵乘法，而对于高阶矩阵乘法，我们为它不可能设计新的硬件，应该以小模块为基础，利用高层的并行算法（如 MPI）实现高阶矩阵乘法。

假设已有用 Cannon 算法处理 S 阶矩阵乘法的结点 P。由于任何 $S'(< S)$ 阶矩阵，我们可以通过补零的方式用 P 计算。因此对于 $N(> S)$ 阶矩阵，我们可以将其补成 N' 阶矩阵，其中

$$N' = \lceil \frac{N}{S} \rceil S = kS$$

此时通过使用 k^2 个 P 结点，就可以组成更高层的 Cannon 算法（称为**法一**）。下面与硬件直接实现的 N 阶矩阵乘法（称为**法**二）比较。

记循环移动的次数、乘法累加的周期数分别为 r, w。设法一不同结点之间的带宽是 S 阶矩阵的大小，则需要 k 个周期初始化移动；接下来的 $k-1$ 次循环移动计算中，每次都需要 P 结点内 S 个周期

初始化，并有 $S-1$ 次结点内的循环移动计算。则有

$$\begin{aligned}
r_1 &= k-1+k(2S-1) \\
&= 2N'-1 \\
w_1 &= kS \\
&= N' \\
r_2 &= 2N-1 \\
w_2 &= N
\end{aligned}$$

从以上可以看出，使用法一所带来的性能损失仅为 $O(S)$，因此我们可以使用法一扩展 Cannon 算法。

4.2 DNS 算法