Pisco Med Publishing

Simple Analysis of Pipeline Performance and Cycle Optimization in Computer Architecture

Shaoshao Xu

Xiamen University, Sepang 43900, Malaysia.

Abstract: This paper first summarizes the performance analysis of pipeline and some problems related to the performance of pipeline, and then introduces the how to loop optimization in detail. Pipelining technology is a very important technology in the field of computer. Using pipelining technology, multiple executing device in the computer work in parallel. The idea of parallelism comes from the traditional production assembly-line model of factory. As early as in the 1960s, some high-end machines have begun to use pipeline technology. Up to now, the pipeline technology has been very mature. The IBM7030 was the first computer to adopt pipeline technology. Pipelined processor can meet the higher requirements of computer operation and improve the performance of CPU. Pipelined processor has become an important and indispensable part of computer architecture.

Keywords: Pipeline; Throughput Rate; Speed up Ratio; Efficiency; Loop Unrolling, Instruction Scheduling

1. Introduction

The emergence of new components has brought great changes to the development of computers. On the one hand, people are improving the performance of computers by improving old processes and trying to develop new components. On the other hand, attempts have been made to improve the performance of computers by improving their organizational structure. Among the many architecture improvements, the application of pipeline technology has a certain influence.

2. Pipeline performance analysis

The throughput rate, speed up ratio and efficiency are the most important three indexes to evaluate the performance of pipelined processors

2.1 Throughput rate

The throughput rate is the result of the amount of service or output provided by the pipeline per unit of time. TP (Through Put) = n / Tk (1.1)[1]

N is the number of tasks

Tk is the total time to complete n tasks.

In the pipeline with k sub-processes (K-level pipeline). The execution time of each segment is equal, and the tasks are continuously input, and the time taken to complete n tasks is $Tk = (k + n - 1)\Delta t$ (1.2)

Where Δt is the clock period .

According to Equation (5.1), in this case the throughput is equal to

$$TP = \frac{n}{(k+n-1)\Delta t} \quad (1.3)$$

As n approaches infinity $TP_{\max} = \frac{1}{\Delta t}$ (1.4)

If the execution time of each segment is unequal

$$T = \sum_{i=1}^{m} \Delta t_i + (n-1)\Delta t_j \quad (1.5)$$

Where m is the total number of sub-processes

Δ ij is the time required for the slowest period

n is the number of tasks

We can prove by example that pipeline technology improves efficiency and shortens the execution time of computer to process the same tasks. In a computer system, the execution of an instruction can be divided into three processes: instruction fetching, instruction analysis and instruction execution. If the instruction fetching time is 4 Δt , the execution time is 3 Δt , and the analysis time is 2 Δt , The time required to execute 800 instructions sequentially is $(4+2+3)\Delta t \times 800 = 7200\Delta t$ .If the instructions are executed in pipelined system, the time required is

$$(4+2+3)\Delta t + (800-1)\times 4\Delta t = 3205\Delta t$$

2.2 Handle the bottleneck of the pipeline

The speed bottleneck in the pipeline is the function segment with the longest processing time.[2] The bottleneck part of the pipeline is mainly to reduce the processing time of the pipeline. There are two methods to remove the speed bottleneck of the pipeline. The first method is to subdivide the bottleneck function segment.[3] For example, the pipeline is divided into four functional segments, and the time required to flow through each functional segment is Δt，3Δt，Δt，Δt. The bottleneck function segment of the pipeline is 3Δt. The bottleneck segment can be subdivided into three sub-functional segments whose time is equal to Δt. If the bottleneck function segment cannot be further subdivided, the bottleneck sub-function segments connected in parallel. [4] As shown in Figure 5.4.

2.3 Speed up ratio

In the field of parallel computing, the pipelining ratio is how much faster the sequential algorithm is compared to the pipelining (parallel algorithm) for the same task.[5] The formula for calculating the acceleration ratio is as follows:

$$S_p = \frac{T_1}{T_p} \text{------(1.6)}$$

P is the number of CPUs

T_p is, pipeline execution time

T_1 is the sequential execution time

According to formula 5.2, when the execution time of each segment is equal and the input task is continuous

$$S = \frac{k \times n \times \Delta t}{(k+n-1)\Delta t} = \frac{k \times n}{(k+n-1)} \quad (1.7)$$

When n approaches infinity, S = k. This limit indicates that when the number of pipeline segments is limited, the larger the number of tasks, the less obvious the acceleration effect. However, under normal conditions, the acceleration ratio is always greater than 1. The speedup of the above example is $\frac{7200\Delta t}{3205\Delta t} = 2.2465$

2.4 Efficiency calculation in pipeline

The efficiency of the pipeline is the percentage of the time of n tasks and the total running time (time for processing tasks and idle time) which can be understood as the utilization rate of the equipment of pipeline equipment. In the pipeline time-space diagram, the efficiency of the pipeline can be obtained by counting the number of grids. E=the space-time region occupied by n tasks (colored cells)/the total space-time region occupied by k flow segments (all cells).

$$E = \frac{T_0}{k \times T_k} = \frac{k \times n \times \Delta t}{k \times (k+n-1) \times \Delta t} = \frac{k \times n}{k \times (k+n-1)} = \frac{n}{k+n-1} \quad (1.8)$$

The quantitative relationship between throughput, pipeline and efficiency

$$S = \mathrm{k}E \quad (1.9) \quad E = TP\Delta\mathrm{t} \quad (1.10) \quad S = TP\mathrm{k}\Delta t \quad (1.11)$$

Examples 1

Floating-point addition and subtraction usually go through five levels: matching of exponents-Mantissa Operation-Results normalization-rounding-overflow judgment You can calculate the value of 10 floating point numbers by using A 5-segment floating point adder with the same delay time for each segment. The pipeline has enough buffer registers and direct data path. Z = A+B+C+D+E+G+F+H+I+J

Find the throughput rate, efficiency and speed up ratio of the pipeline.

If we set the delay time of each segment as Δt

According to the space-time diagram, the the pipeline has completed nine tasks within 20Δt.

$$\text{Throughput } T_{\mathrm{p}} = \frac{9}{20} = 0.45\frac{1}{\Delta t}$$

$$\text{Speed up ratio } S_p = \frac{9 \times 5}{20} = 2.5$$

$$\text{Efficiency} : E = \frac{45}{20 \times 5} = 45\%$$

Example 2

When calculating addition and multiplication on a dual-function pipeline where addition is divided into three levels and multiplication is divided into five levels, it should be noted that the same section of the pipeline with different functions should not overlap. (A1 + B1) * (A2 + B2) * (A3 + B3) * (A4 + B4)

Calculation order: $x_1 = a_1 + b_1, x_2 = a_2 + b_2, x_3 = a_3 + b_3, x_4 = a_4 + b_4$

$y_1 = x_1 \times x_2, y_2 = x_3 \times x_4$

So z is equal to $y_1 \times y_2$

Notice that when two different functions are realized in the same pipeline, the space-time diagram cannot overlap.

$$Throughput: T_p = \frac{n}{T_k} = \frac{7}{17 \times \Delta t}$$

$$Speedup: S = \frac{(4 \times 3 + 3 \times 5)}{17} = 1.88$$

$$Efficiency: E = \frac{(4 \times 3 + 3 \times 5)}{17 \times 6} = 0.264$$

The comparison of the two examples above shows that in order to improve efficiency, it is necessary to minimize function switching, process as many instructions as possible, and refine each function segment. Pipelining works better for independent instructions.

3. Cycle expansion optimization

3.1 Instruction scheduling

Instruction scheduling is performed by the compiler. [6] As mentioned earlier, the pipeline works better for independent instructions. The purpose of instruction scheduling is to find out such instruction sequences and execute unrelated instructions on the pipeline in order to reduce the execution time of the whole instruction

For convenience of discussion, the delay of floating-point pipeline used is

The instruction that produces the result	Instructions for using results	Delay (Cycles)
Floating point calculations	Another floating-point calculation	3
Floating point calculations	Floating point store (S.D.)	2
Floating point Load (L.D)	Floating point calculations	1
Floating point Load (L.D)	Floating point store (S.D.)	0

Table 3.1

(The result of the floating-point load instruction can be sent to the store instruction through the directional path in time, so the delay is 0)

Example 3

For the following source code, converted to MIPS assembly language, without instruction scheduling and instruction scheduling conditions, analyze the time of a loop.

```
for (i=1000; i>0; i--)
x[ i ] = x[ i ] ＋ S;
```

Start by translating the program into MIPS assembly language code

```
Loop:   L. D        F0, 0(R1)
        ADD.D       F4, F0, F2
        S.  D       F4, 0(R1)
        DADDIU      R1, R1, #-8
        BNE         R1, R2, Loop
```

In the case of no instruction scheduling, according to the instruction execution delay in the floating-point pipeline given in the table, the actual execution of the program is as follows

```
Loop:   L. D        F0, 0(R1)
        ADD.D       F4, F0, F2
        S. D        F4, 0(R1)
        DADDIU      R1, R1, #-8
```

	Instruction outflow clock
Loop :L.D F0, 0(R1)	1
idle	2
ADD.D F4, F0, F2	3
idle	4
idle	5
S.D F4, 0(R1)	6
DADDIU R1, R1, #-8	7
idle	8
BNE R1, R2, Loop	9
idle	10

Table 3.2After the instruction scheduling

	Instruction outflow clock
Loop :L.D F0, 0(R1)	1
DADDIU R1, R1, #-8	2
ADD.D F4, F0, F2	3
idle	4
BNE R1, R2, Loop	5
S.D F4, 0(R1)	6

Table 3.3

3.2 Loop unrolling

Loop unrolling is a kind of program transformation that reduces the number of loop iterations and branching instructions by performing more data operations in each iteration. For example, we can replace the code:

```
for (i = 0; i < len; i++) {
sum += arry[i]
}
```

with the code[7]:

```
for (i = 0; i < len; i += 2) {
    newSum += arry[ i ] + arry[i + 1]
    }
```

3.3 Notes on loop unrolling and instruction scheduling

Pay attention to effectiveness and correctness.

Use different registers.

Delete redundant test instructions and branch instructions.

Correct the end of loop code and the new body of loop code accordingly.

Pay attention to the correlation analysis of memory data and new correlations.

Conclusion

Pipeline technology is one of the two major breakthroughs in the history of processor: it enables the processor to execute multiple instructions at the same time. This special working mode of pipeline technology not only increases the workload per unit time, but also helps to improve the CPU frequency. The classification of pipeline technology is more complicated, its description is mainly through the space-time diagram, through the space-time diagram, we can see the factors that affect the efficiency of the pipeline, and more intuitive to see the characteristics of pipeline technology. The throughput acceleration ratio and efficiency are used to evaluate the pipeline processor. And the scope of study is exaggerated to provide information for pipeline cycle optimization. It is very helpful to understand the pipeline processing technology for the study of computer structure and time parallel technology.

References

[1] Chopra, R. (2008). *Advanced Computer Architecture*. S. Chand Publishing.

Advanced Computer Architecture Rajiv Chopra Chand Publishing, 2008.

[2] H. Tuncer, I., Gülcat, Ü., R. Emerson, D., & Matsuno, K. (2007a). *Parallel Computational Fluid Dynamics 2007*. Springer.

[3] *Understanding Bottlenecks*. (2020, November 13). Investopedia.

[4] Cosnard, M., Ferreira, A., & Peters, J. (1994). *Parallel and Distributed Computing: Theory and Practice.* (First Canada-France Conference, Montreal, Canada, May 19–21, 1994. Proceedings ed.). Springer Science & Business Media.

[5] *Speedup Ratio and Parallel Efficiency: TechWeb: Boston University*. (n.d.).

[6] A. Fisher, J., Faraboschi, P., & Young, C. (1981). *Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools*. Denise E.M Penrose.

[7] Matthes, E. (2020). *CUDA C Programming Authoritative Guide*.