DPDK SUMMIT CHINA 2017

主办方：intel

参与方：腾讯云 ZTE 美团云 Panabit 太一星晨 UnitedStack 云杉网络

协办方：SDNLAB

视频支持方：IT大咖说

Network Acceleration and Performance Improvement

Lou, Fangliang
ZTE

主办方：

参与方：

协办方：

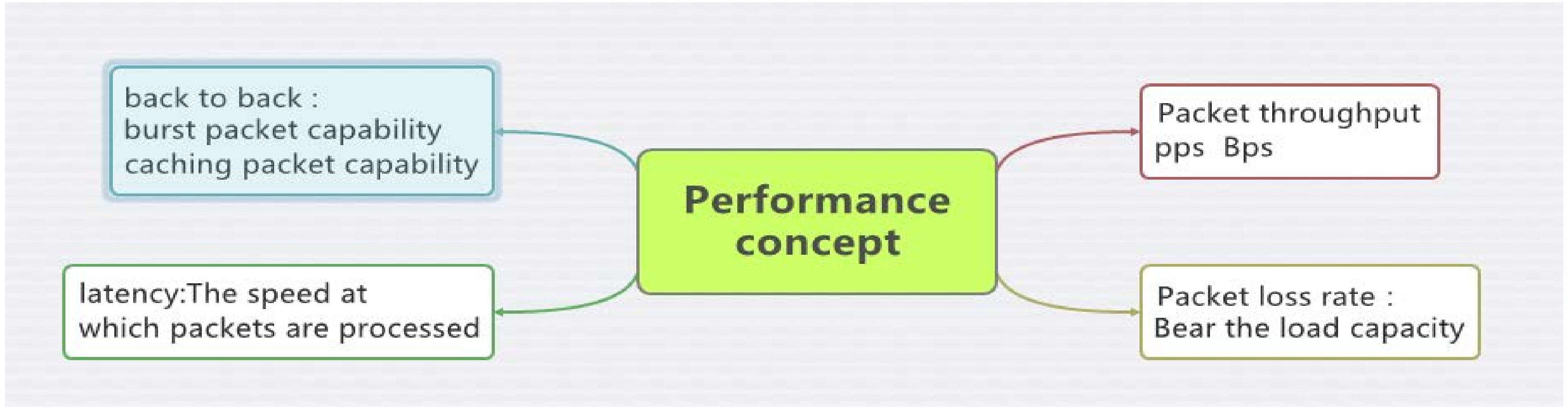
视频支持方：

Agenda

第一	Performance optimization concepts and methods
第二	Performance optimization crisis
第三	Software Performance Lean Measurements
第四	Enhance software parallel scalability

Performance Optimization Concepts

➢RFC2544协议性能测试定义



$$\text{报文转发速率} = \frac{\text{网卡线速}}{\text{pkgsize} + \text{报文先导} + \text{帧间隔} + \text{帧定界符}}$$

$$\text{报文处理平均时间} = \text{报文平均时间间隔} = \frac{1}{\text{报文转发速率}}$$

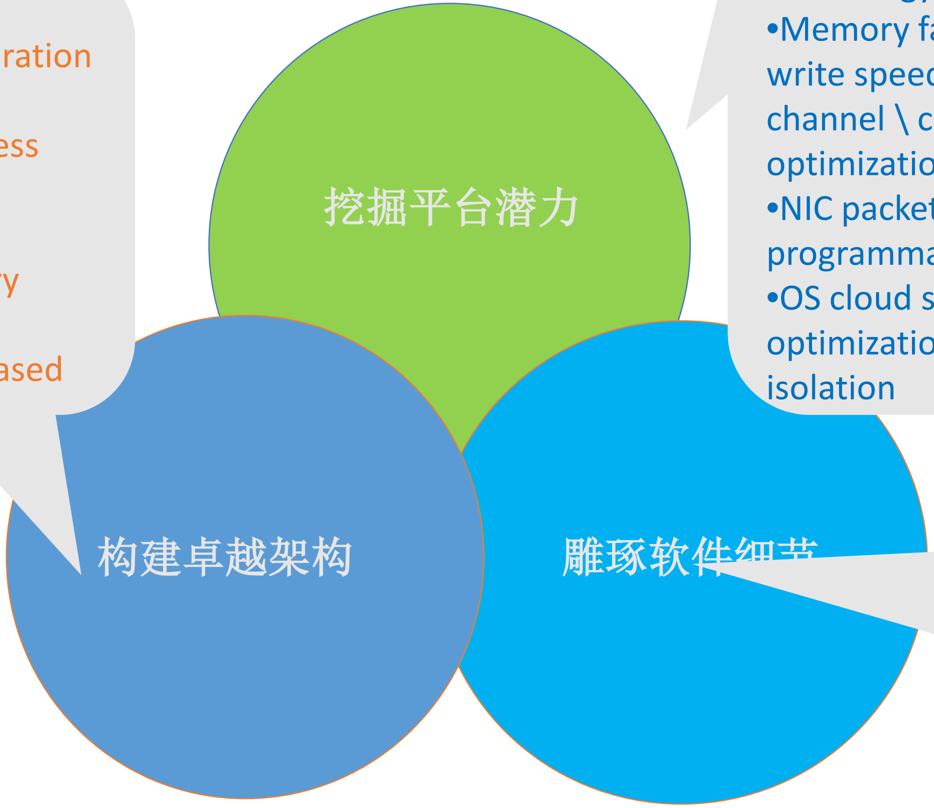
Difficulties of Processing Packet

Software to achieve high-speed packet forwarding is difficult, the smaller the average packet arrival interval is smaller, the higher the CPU processing time requirements, the above table to 2G clock speed CPU, for example, a command cycle is 0.5ns, CPU Access to DDR memory is 140 * 0.5 = 70ns, CPU access L3 cache time is 40 * 0.5 = 20ns. And a 64byte packet arrived at the average time in 16.8ns, this multi-software message processing put forward a very high requirements: in an access time need 70ns, 16.8ns to deal with a packet。

	容量	访问延迟 (cyc)	访问时间 (2G)
指令cache	32K	5/7	2.5ns
L1 cache	32K	5/7	2.5ns
L2 cache	256K	12	6ns
LLC(L3)	20M	40	20ns
DDR	128G	140	70ns

报文长度	网卡线速	最大PPS	报文平均间隔	换算指令周期
64byte	10G	14.88M	67.2ns	134cyc
512byte	10G	2.35M	425.6ns	861cyc
64byte	40G	59M	16.8ns	33cyc
512byte	40G	9.4M	106ns	212cyc

Tradition Optimization



•Control media separation build
•Unlocked, Conflictless Parallel Processing Architecture
•Reasonable memory access range
•Statistical Design Based on Traffic Model

•CPU frequency \ more kernel \ CPU new technology
•Memory faster read and write speed \ multi - channel \ cache optimization
•NIC packet classification \ programmable
•OS cloud system \ kernel optimization \ nuclear isolation

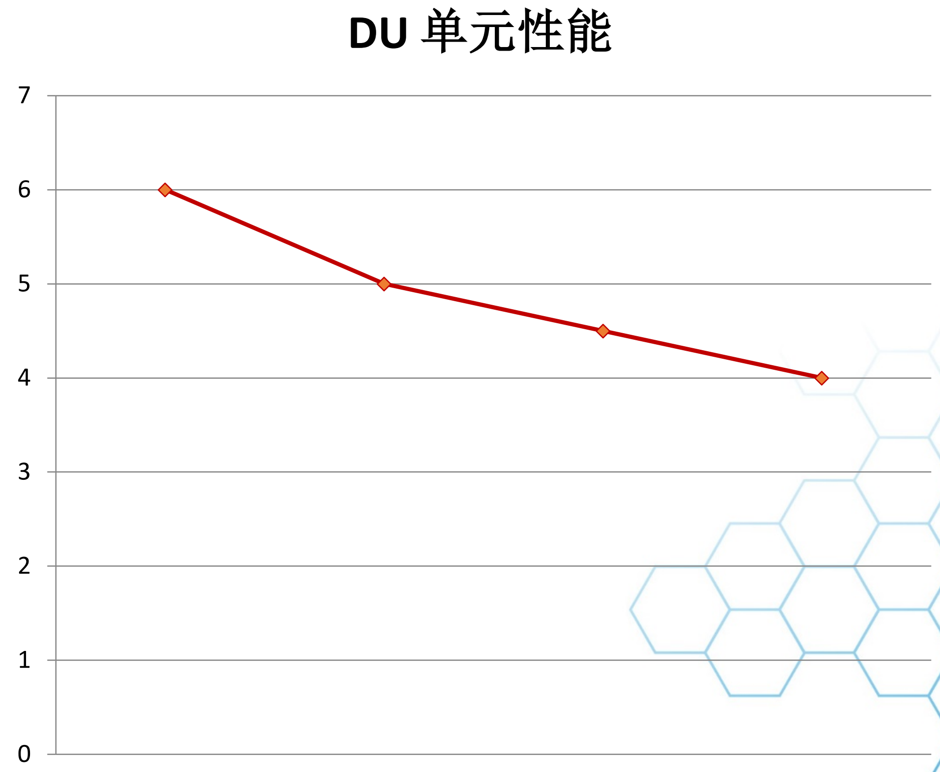
•Compile the potential of mining CPU instructions / specific calculations using a specific instruction set
•Compile Select the appropriate compiler to carefully select the optimization strategy
•Process No Locked Parallel Processing Flattened Process Design Cured Framework Code
•Memory control memory read and write times space for time
•Algorithm main process focus optimization

Agenda

第一	Performance optimization concepts and methods
第二	Performance optimization crisis
第三	Software Performance Lean Measurements
第四	Enhance software parallel scalability

Crisis

- ➢From each version of the performance point of view, due to the addition of some features, DU performance has been declining; each version of the performance tuning work, although there are certain enhancements, very painful repetitive labor;
- ➢The future increase in demand for media, DU software, increased complexity, but also reduce performance; bottleneck in the DU core, how to avoid this?



Software Performance Lean Measurement Method

Software Performance Lean Measurement Method:

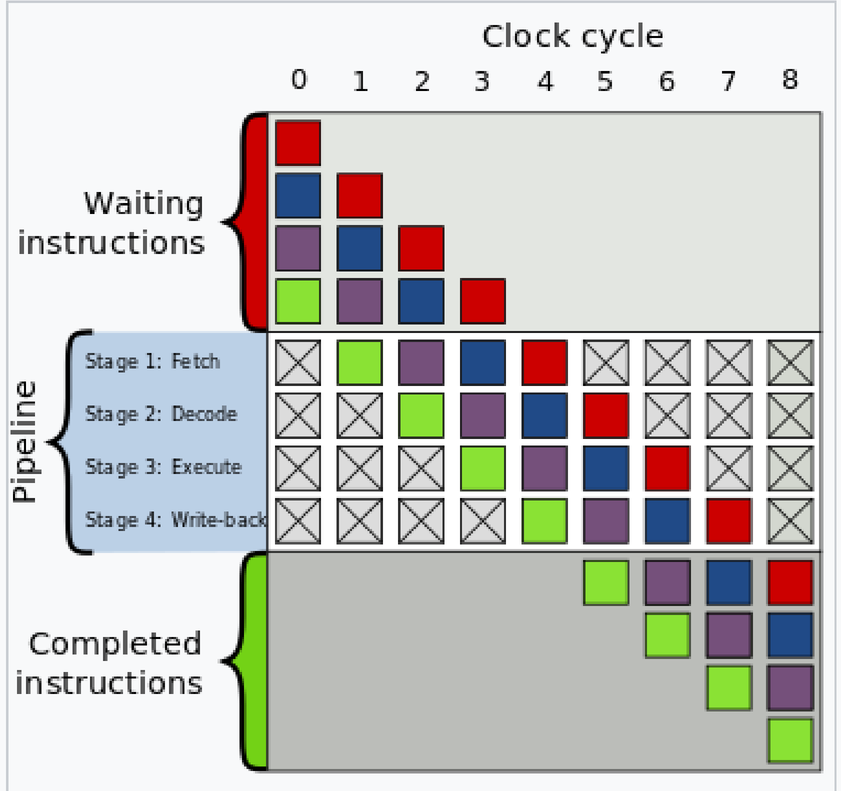
- ➢ **Where is the performance of the ceiling? Does performance optimization have space? Is it worthwhile to spend more effort on further research?**
- ➢ **Often encountered problems, the code modified a line, the performance suddenly dropped a lot of length, how to avoid these problems?**

Today we propose a software performance lean measurement optimization method through the theoretical quantitative calculation, the model can be more accurate calculation of optimization goals.

Agenda

第一	Performance optimization concepts and methods
第二	Performance optimization crisis
第三	Software Performance Lean Measurements
第四	Enhance software parallel scalability

Instruction four - layer water structure



一般的四层流水线架构；不同的颜色格表示不同的指令

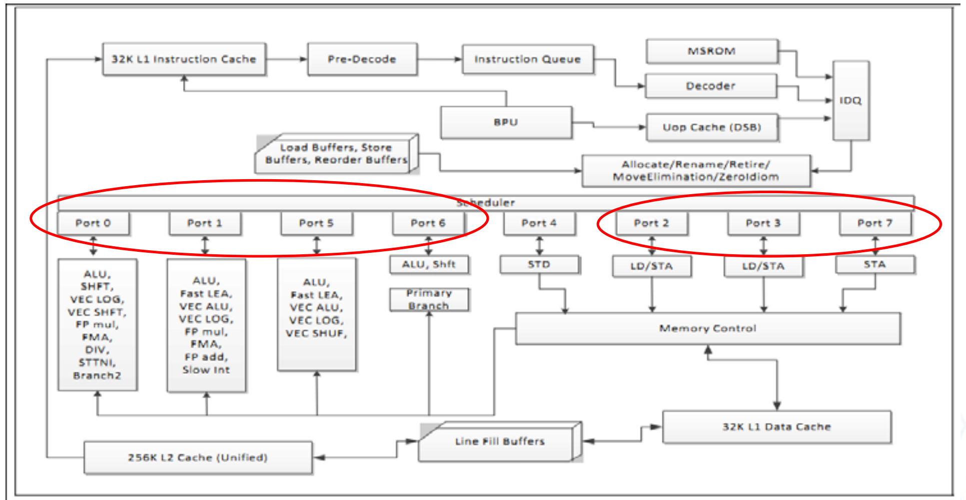

32K L1 Instruction Cache
Pre-Decode
Instruction Queue
MSROM
Decoder
IDQ
BPU
Uop Cache (DSB)
Load Buffers, Store Buffers, Reorder Buffers
Allocate/Rename/Retire/ MoveElimination/ZeroIdiom
Scheduler
Port 0
Port 1
Port 5
Port 6
Port 4
Port 2
Port 3
Port 7
ALU, SHFT, VEC LOG, VEC SHFT, FP mul, FMA, DIV, STTNI, Branch2
ALU, Fast LEA, VEC ALU, VEC LOG, FP mul, FMA, FP add, Slow Int
ALU, Fast LEA, VEC ALU, VEC LOG, VEC SHUF,
ALU, Shft
Primary Branch
STD
LD/STA
LD/STA
STA
Memory Control
32K L1 Data Cache
Line Fill Buffers
256K L2 Cache (Unified)

Figure 2-2. CPU Core Pipeline Functionality of the Haswell Microarchitecture

Assuming an instruction requires a clock cycle, the following procedure takes a few iterations over a loop?

```
while (1)
  {
          mov  $0x1, %eax
          mov  $0x2, %ebx
          mov  $0x3, %ecx
  }
```

1	XOR a, b
2	XOR b, a
3	XOR a, b

Instruction Flow Indicator

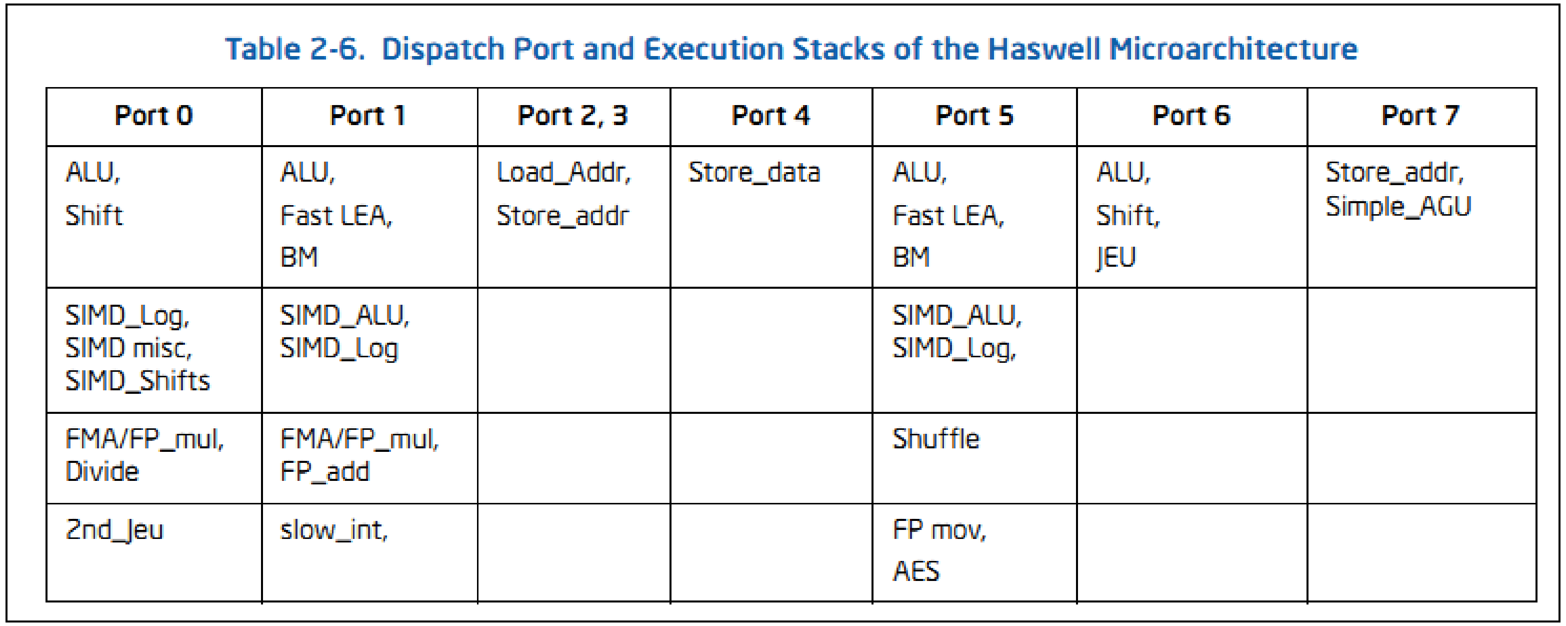


Table 2-6. Dispatch Port and Execution Stacks of the Haswell Microarchitecture

Port 0	Port 1	Port 2, 3	Port 4	Port 5	Port 6	Port 7
ALU, Shift	ALU, Fast LEA, BM	Load_Addr, Store_addr	Store_data	ALU, Fast LEA, BM	ALU, Shift, JEU	Store_addr, Simple_AGU
SIMD_Log, SIMD misc, SIMD_Shifts	SIMD_ALU, SIMD_Log			SIMD_ALU, SIMD_Log,		
FMA/FP_mul, Divide	FMA/FP_mul, FP_add			Shuffle		
2nd_Jeu	slow_int,			FP mov, AES		

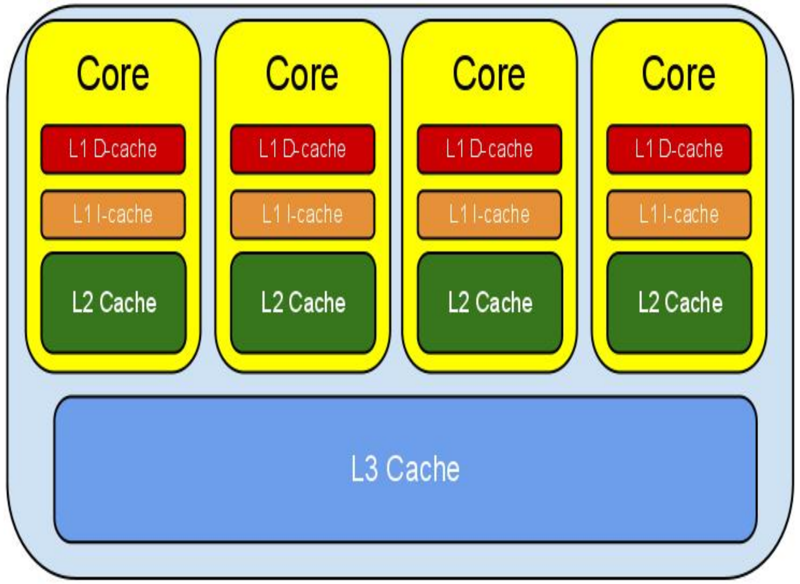
指标	英文	中文
IPC	instuction per cycle	单周期执行指令数
eIPC	effective instuction per cycle	单周期执行的有效指令数

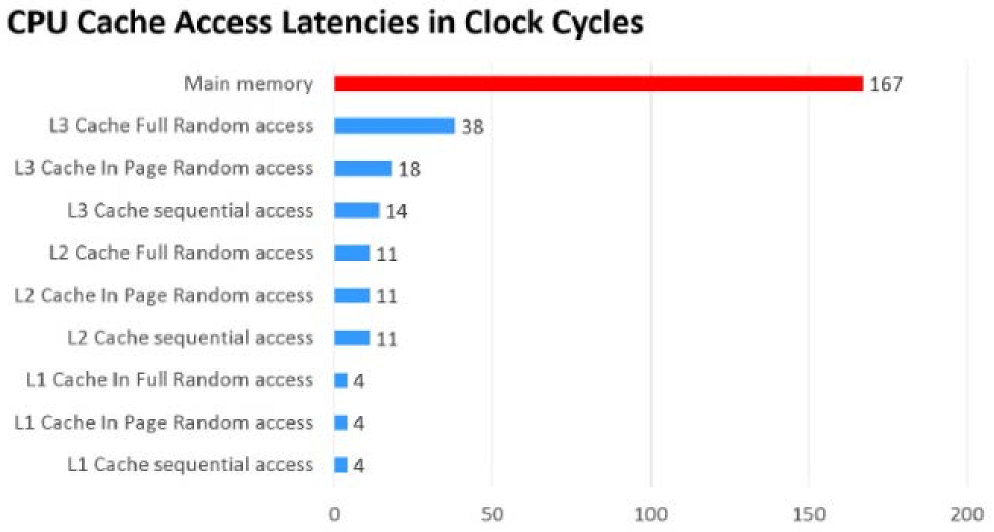
Out of Order Execution -> Order to Commit

Out of order execution:
Use the following steps to disrupt the order:
(1). get the instruction ;
(2). The instruction is sent to a sequence of instructions (execution of a buffer or a reserved station);
(3). The instruction waits for a direct data operation object in the sequence to be fetched, and then the instruction sequence is allowed to leave the buffer before entering the old instruction and before;
(4). The instruction is assigned to a suitable functional unit and executed by him
(5). The result is placed in a sequence;
(6) The result of this instruction is written to the register only if all the instructions before the instruction have written their results to the register. This process is called a graduation or retirement period

➢**Out of order to use other "executable" instructions to fill the gaps in the time, and then reorder the results at the end of the operation, in order to achieve the results of instruction execution in order to submit the code.**

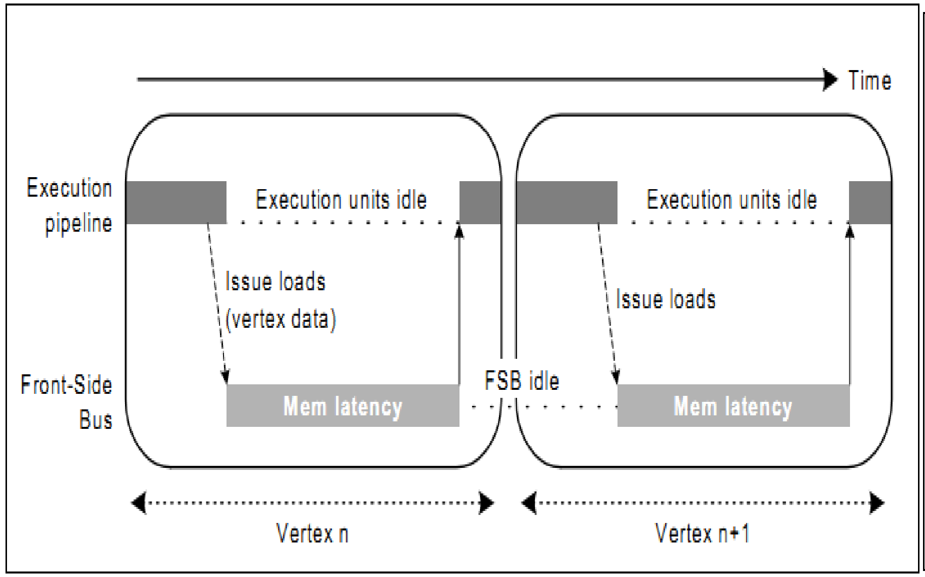
Cache

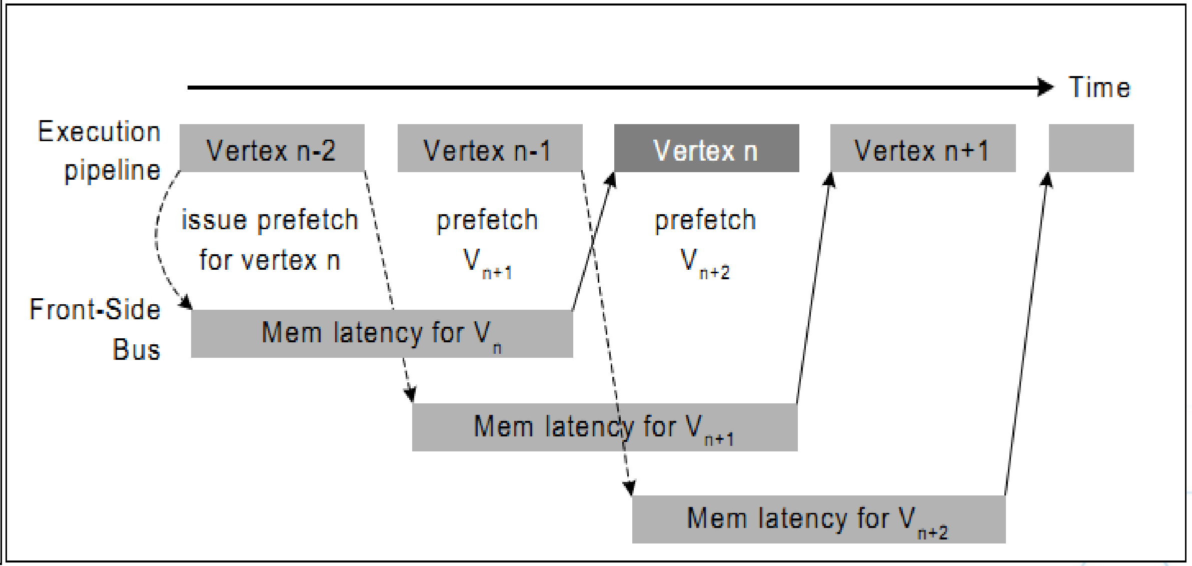




Cache is the cache (Cache Memory), in order to solve the read physical memory delay, because the modern CPU processing speed and memory access speed.

Cache Miss



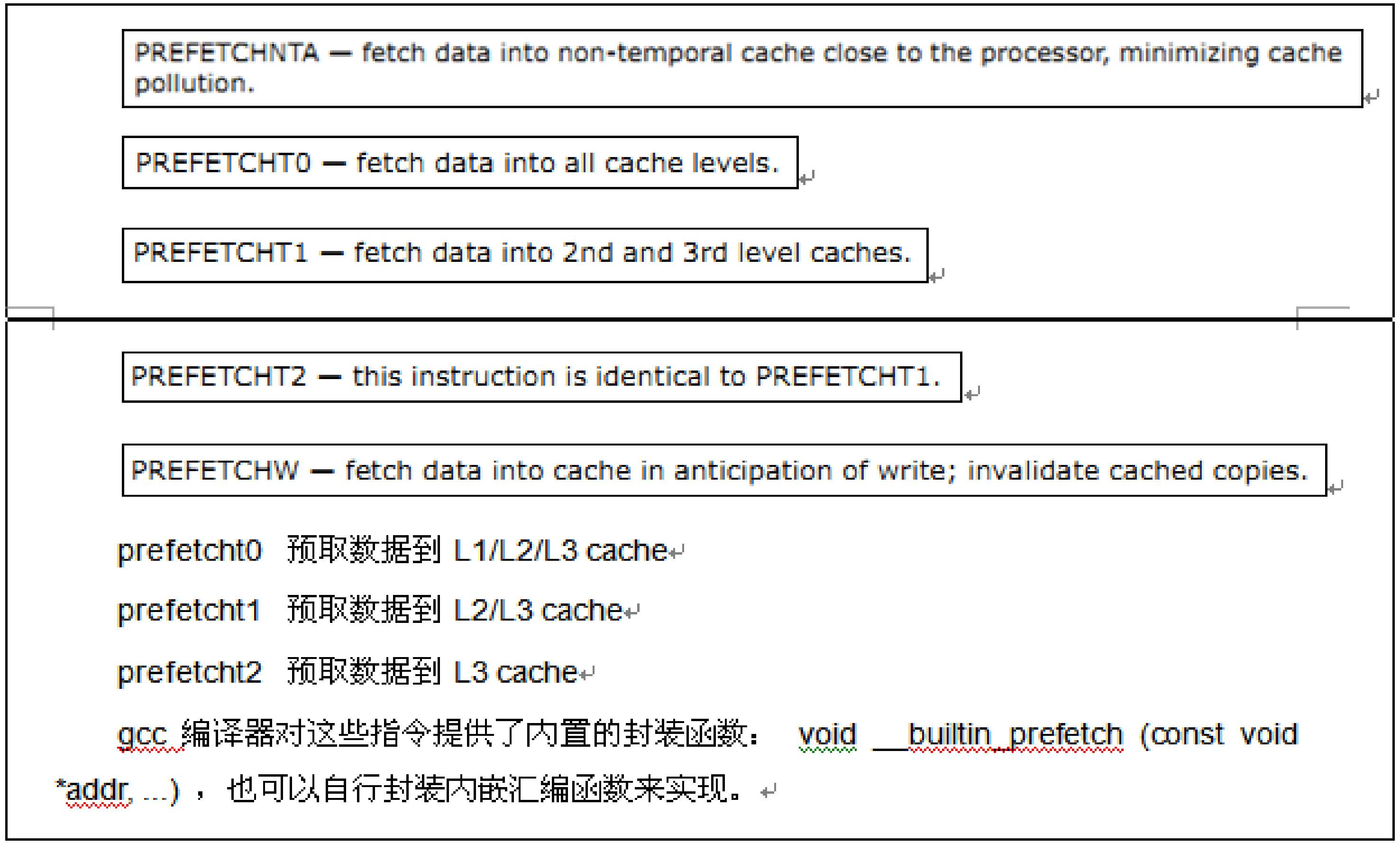


As shown above, when the prefetch is not performed, the execution unit is forced to wait due to the delay of reading the memory from the front side bus (FSB).

After prefetching in advance, the data required by the execution unit has been read in advance to the cache and read the data directly in the cache. So that the execution unit eliminates the latency of memory access.

指标	英文	中文
Cache miss	Cache miss	Cache 没有命中的概率

Cache Prefetch



PREFETCHNTA — fetch data into non-temporal cache close to the processor, minimizing cache pollution.

PREFETCHT0 — fetch data into all cache levels.

PREFETCHT1 — fetch data into 2nd and 3rd level caches.

PREFETCHT2 — this instruction is identical to PREFETCHT1.

PREFETCHW — fetch data into cache in anticipation of write; invalidate cached copies.

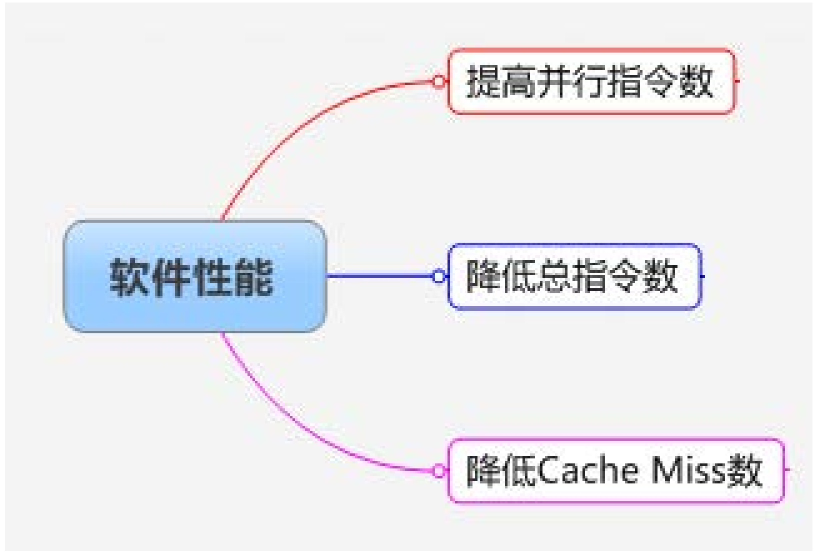
prefetcht0 预取数据到 L1/L2/L3 cache

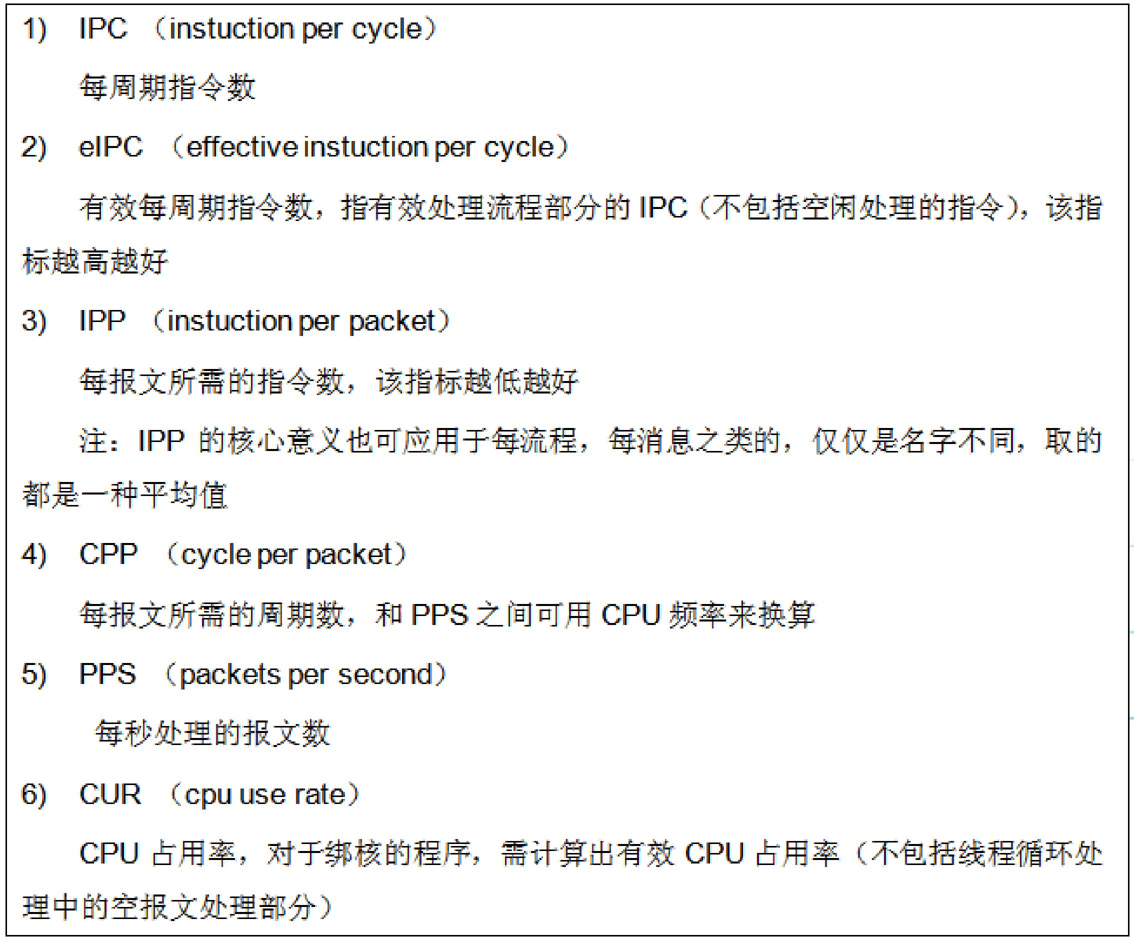
prefetcht1 预取数据到 L2/L3 cache

prefetcht2 预取数据到 L3 cache

gcc 编译器对这些指令提供了内置的封装函数： void __builtin_prefetch (const void *addr, ...)，也可以自行封装内嵌汇编函数来实现。

Metrics




1) IPC （instuction per cycle）

每周期指令数

2) eIPC （effective instuction per cycle）

有效每周期指令数，指有效处理流程部分的 IPC（不包括空闲处理的指令），该指标越高越好

3) IPP （instuction per packet）

每报文所需的指令数，该指标越低越好

注：IPP 的核心意义也可应用于每流程，每消息之类的，仅仅是名字不同，取的都是一种平均值

4) CPP （cycle per packet）

每报文所需的周期数，和 PPS 之间可用 CPU 频率来换算

5) PPS （packets per second）

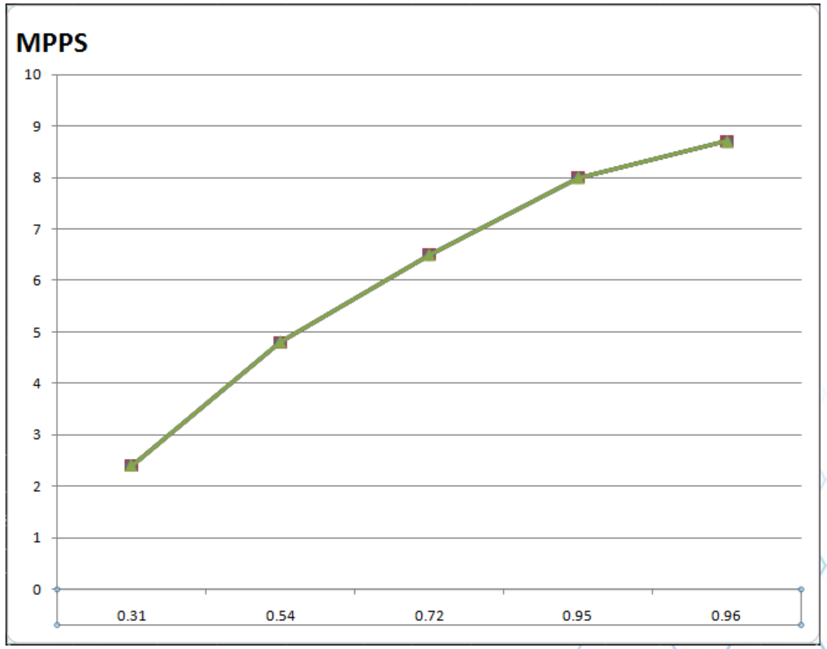
每秒处理的报文数

6) CUR （cpu use rate）

CPU 占用率，对于绑核的程序，需计算出有效 CPU 占用率（不包括线程循环处理中的空报文处理部分）

Metrics

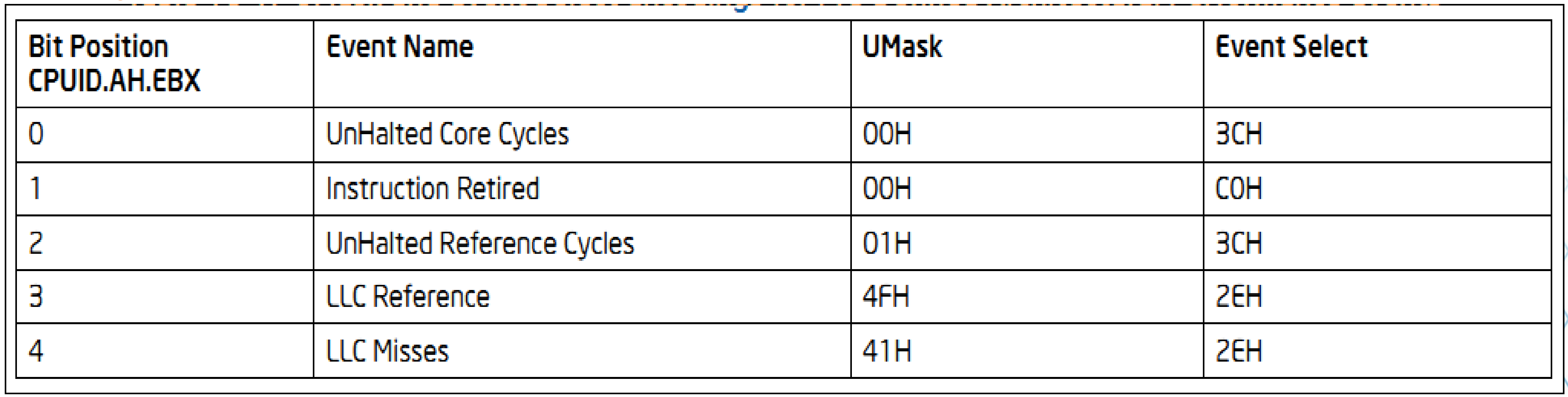
$$
\begin{aligned}
\mathrm{PPS} &= \frac{\mathrm{Freq}(\text{CPU 频率}) \times \mathrm{R}(\text{CPU 占用率})}{\mathrm{CPP}} \\
&= \frac{\mathrm{Freq}(\text{CPU 频率}) \times \mathrm{R}(\text{CPU 占用率})}{\frac{\mathrm{IPP}}{\mathrm{eIPC}}} \\
&= \frac{\mathrm{Freq}(\text{CPU 频率}) \times \mathrm{R}(\text{CPU 占用率}) \times \mathrm{eIPC}}{\mathrm{IPP}}
\end{aligned}
$$



[1] eIPC indicators higher, the better the performance;
[2] IPP indicators lower, the better performance;
[3] When the CPU occupancy rate is low, the performance and occupancy rate are close to the linear relationship because the eIPC and IPP can be considered unchanged; when the CPU occupancy rate is high, the performance and occupancy rate are non-linear, because eIPC and IPP (Due to the greater flow of concurrent processing will increase, will affect the eIPC and IPP indicators);

Data Collection Method

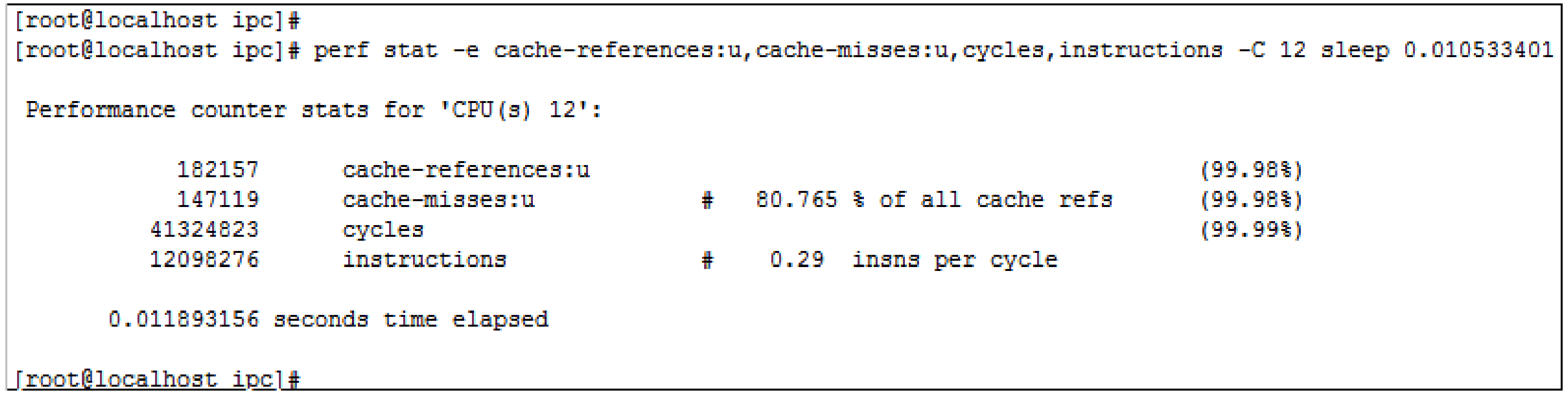
➢Intel CPU built-in hardware Performance Monitoring Unit (PMU)
Provides a lot of hardware event counters (such as Haswell provides 334 hardware event counters) You can monitor the CPU instruction execution in detail, such as the number of Cache-miss, the number of branch prediction failures,



Bit Position CPUID.AH.EBX	Event Name	UMask	Event Select
0	UnHalted Core Cycles	00H	3CH
1	Instruction Retired	00H	C0H
2	UnHalted Reference Cycles	01H	3CH
3	LLC Reference	4FH	2EH
4	LLC Misses	41H	2EH

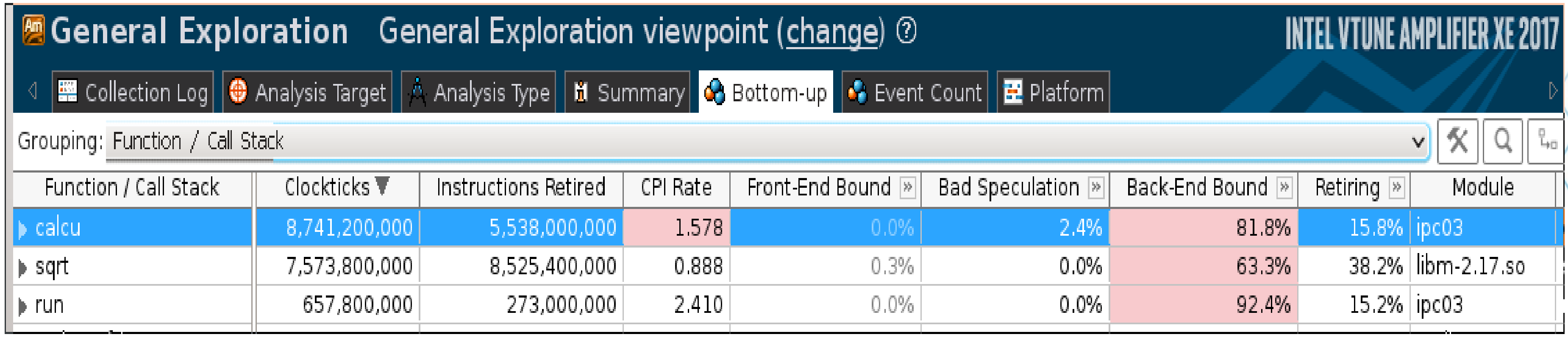
Tools

Perf



```
[root@localhost ipc]#
[root@localhost ipc]# perf stat -e cache-references:u,cache-misses:u,cycles,instructions -C 12 sleep 0.010533401

 Performance counter stats for 'CPU(s) 12':

           182157      cache-references:u                                          (99.98%)
           147119      cache-misses:u            #   80.765 % of all cache refs      (99.98%)
         41324823      cycles                                                      (99.99%)
         12098276      instructions              #    0.29  insns per cycle

       0.011893156 seconds time elapsed

[root@localhost ipc]#
```

VTune

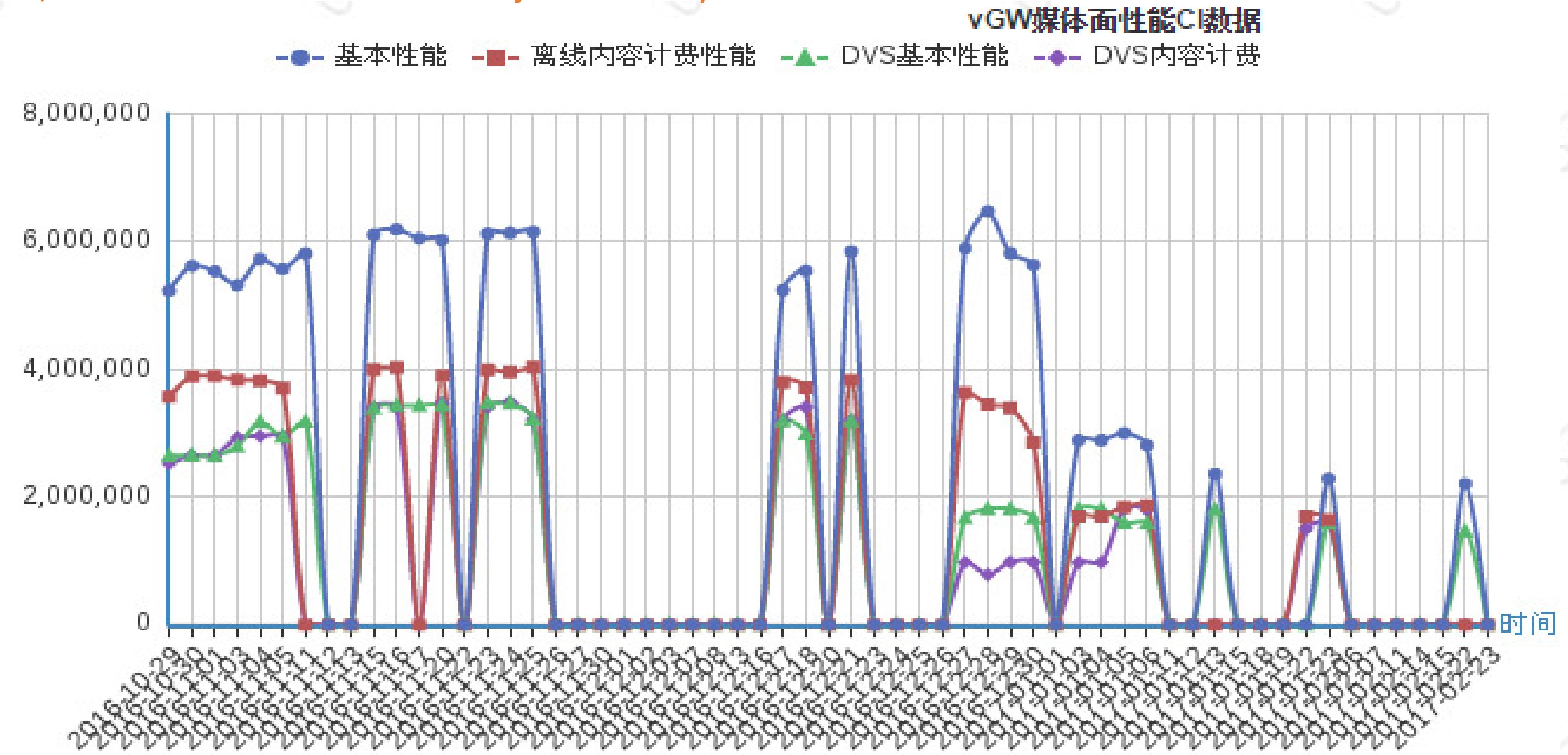


Grouping: Function / Call Stack

Function / Call Stack	Clockticks	Instructions Retired	CPI Rate	Front-End Bound	Bad Speculation	Back-End Bound	Retiring	Module
calcu	8,741,200,000	5,538,000,000	1.578	0.0%	2.4%	81.8%	15.8%	ipc03
sqrt	7,573,800,000	8,525,400,000	0.888	0.3%	0.0%	63.3%	38.2%	libm-2.17.so
run	657,800,000	273,000,000	2.410	0.0%	0.0%	92.4%	15.2%	ipc03

CASE 1: Quick Feedback, Timely Correction

CI to monitor the performance of media performance indicators to monitor, quickly find defects, which version, which led to a decline in the joint. Timely correction



CASE 2： Guidance Coding

```
template <typename Ta, typename Tb, typename Tc, typename Talpha, typename Tbeta>
void seq_matrix_mul_int_colMajor(
    const int M, const int N, const int K, const Talpha alpha, const Ta *B,
    const Tb *A, Tc *C, const Tbeta beta) {
    int i, j, k;
    // use sequential
    for (i = 0; i < M; i++) {
        for (j = 0; j < N; j++) {
            Tc temp = 0;
            for (k = 0; k < K; k++) {
                temp += B[k * M + i] * A[j * K + k];
            }
            C[j * M + i] = alpha * temp + beta * C[j * M + i];
        }
    }
}
```

This code can only be functional verification, the efficiency is very poor, mainly due to two reasons:

➢ the innermost layer of the temp for the calculation of the cycle, the dependencies between the various iterations can not be parallel processing;

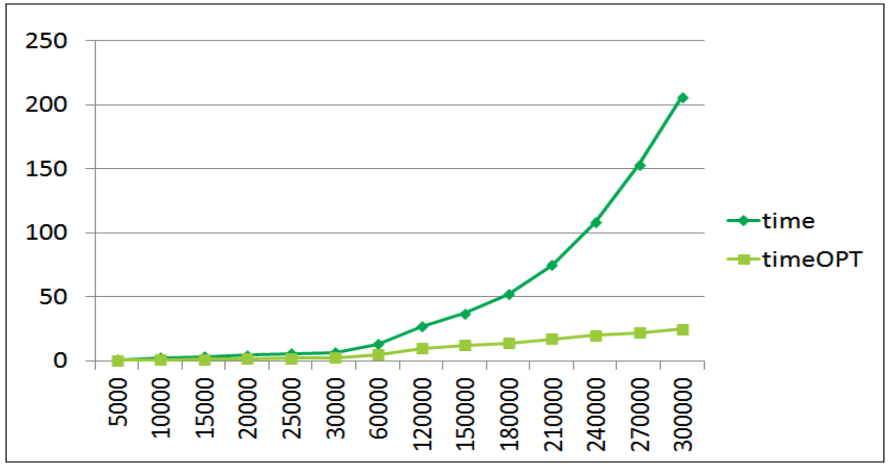
➢ Calculate temp, the B matrix by B [k * M + i] reference is to press B line to jump to take the data, if the B number of columns is large, will lead to access to memory when the span is large, can not use hardware pre- Take, will lead to a lot of cache-miss.

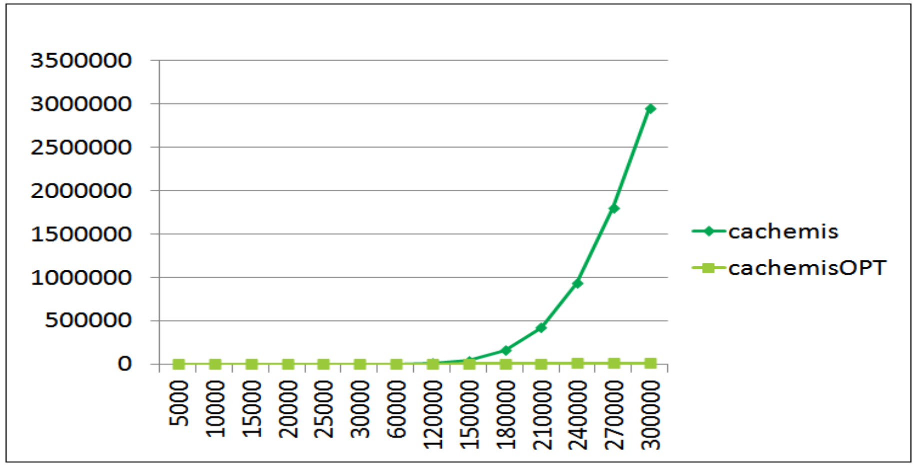
```
for (j = 0; j < N; j++) {
    for (k = 0; k < K; k++)
    {
        for (i = 0; i < M; i++)
            ptmp[i]+=A[j*K + k] * B[k*M + i];
    }
    for (i = 0; i < M; i++)
    {
        C[j*M + i] = alpha * ptmp[i]
                   + beta * C[j*M + i] ;
        ptmp[i] = 0;
    }
}
```

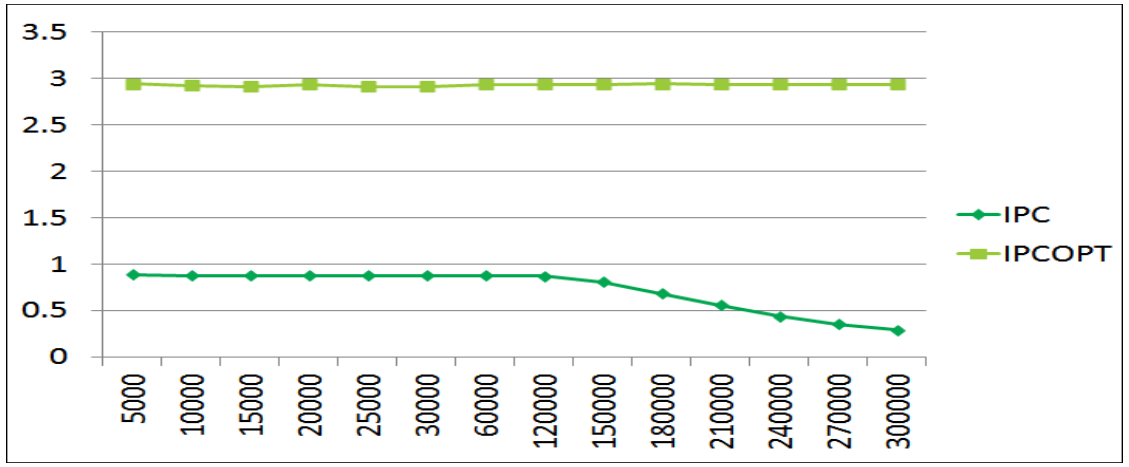
➢ Calculate the for loop of ptmp [i], the dependency of each iteration can be processed in parallel;

➢ Calculate ptmp [i], the B matrix is accessed by column, you can use hardware prefetching, cache-miss greatly reduced.

CASE 2： Guidance Coding







Case3 ： Guide the Software Architecture Design.

We believe that the CPU instruction execution level, the effect of performance optimization is ultimately reflected in two aspects:

(1) eIPC (effective instruction per cycle) indicators, that is, the code should make full use of CPU pipelines to improve the number of instructions within a clock cycle can be implemented, such as the use of VPP architecture, loop expansion and reduce the code between the statements, Hardware and software prefetching.

Features: The total number of instructions executed at this node is constant, but the concurrency increases significantly;

(2) IPP (instruction per packet) indicators of the reduction, that is to say as much as possible to reduce the number of instructions required to deal with the use of efficient instructions, algorithms or architecture decomposition, such as the use of SIMD instructions, ahead of time rather than the cycle of each iteration Calculation, the drive burst transceiver (one call to send and receive multiple messages), node split, traffic unloading, etc.,

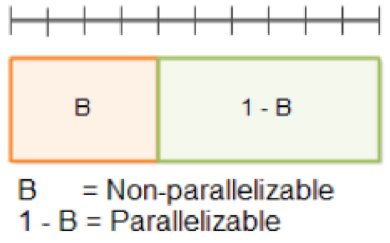
Features: The number of instructions executed by this node has been significantly reduced;

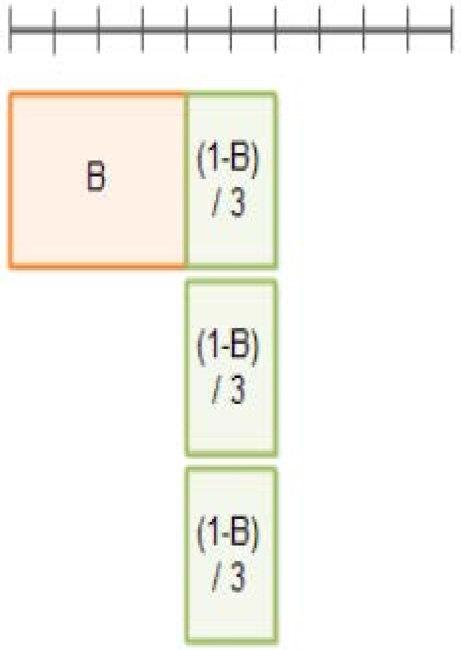
Agenda

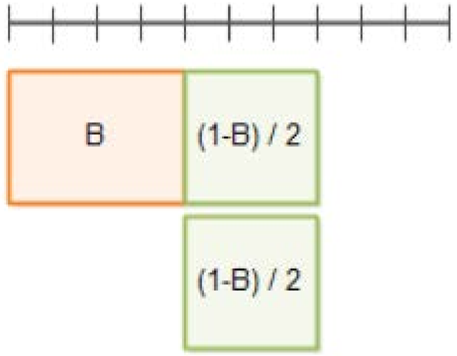
第一	Performance optimization concepts and methods
第二	Performance optimization crisis
第三	Software Performance Lean Measurements
第四	Enhance software parallel scalability

Amdahl Law

A program (or an algorithm) can be divided into the following two parts by whether it can be parallelized:
Can be parallelized
Can not be parallelized
Defined as follows:
T = total time of serial execution
B = can not be parallel to the total time
T-B = total time of the parallel part
When a parallel portion of a program is executed using N threads or CPUs, the total time taken is:
T (N) = B + (T - B) / N







Amdahl Law

As can be seen from Amdahl's law, the parallelizable part of the program can run faster by using more hardware (more threads or CPUs). For non-parallelizable parts, only by optimizing the code to achieve the purpose of speed. Therefore, you can optimize the non-parallel part of the program to improve your running speed and parallel ability. You can do nothing on the algorithm to do some changes, if possible, you can also move some of the parallel to the release of the part.

Optimize the serial component

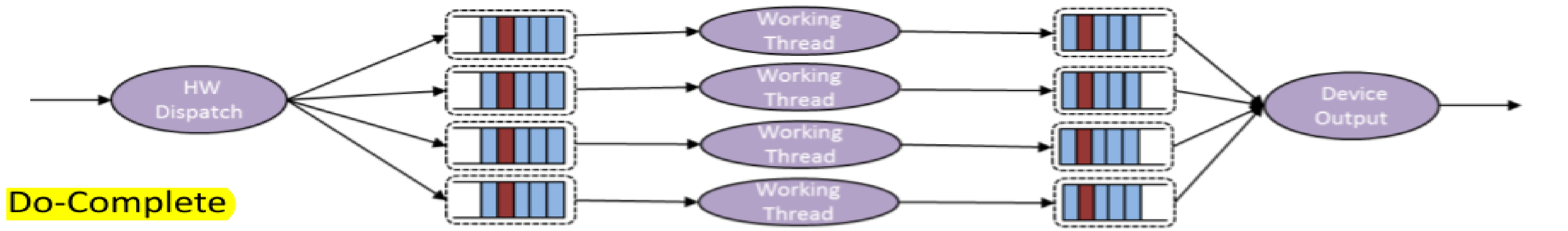
If you optimize the serialization of a program, you can also use Amdal's law to calculate the program execution time after optimization. If the parallel part can be optimized by a factor O, the Amdar law looks like this:

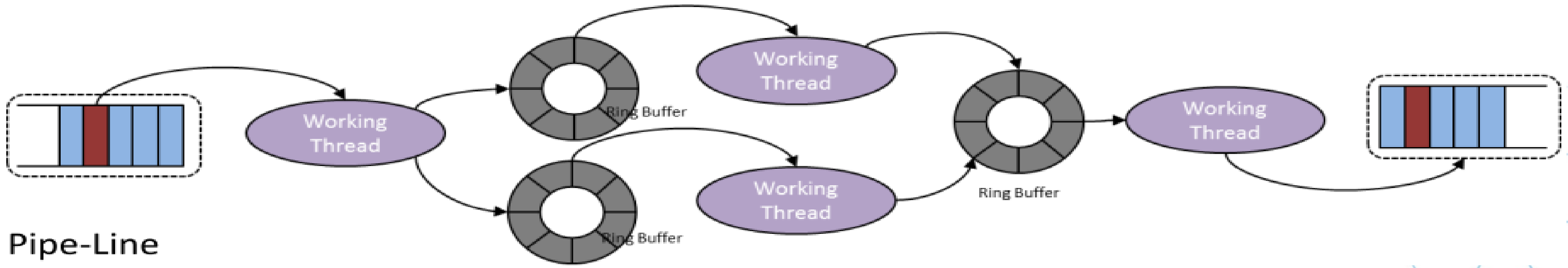
T (O, N) = B / O + (1 - B / O) / N

In the non-parallel part of the program occupies the B / O time, so the parallel part of the account of the 1 - B / O time. If B is 0.1, O is 2, N is 5, the calculation looks like such:

T (2,5) = 0.4 / 2 + (1 - 0.4 / 2) / 5
= 0.2 + (1 - 0.4 / 2) / 5
= 0.2 + (1 - 0.2) / 5 = 0.2 + 0.8 / 5 = 0.2 + 0.16 = 0.36

Two Models



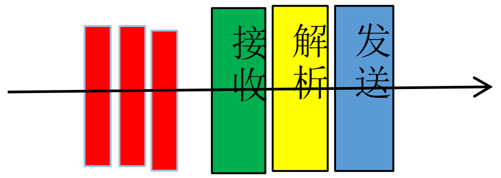


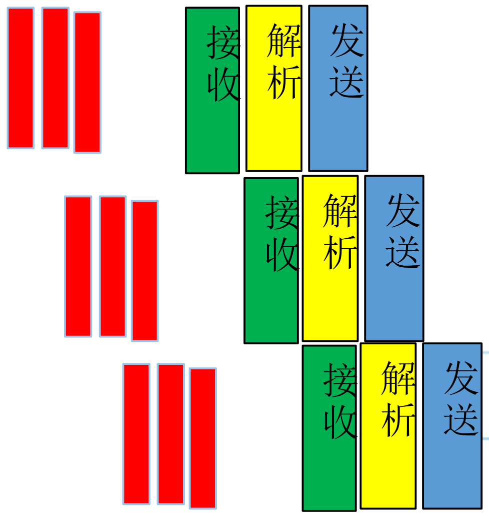
Pipe-Line

- •Parallel processing
- •Multi-threaded parallel processing
- •No lock
- •Unique messages are distributed to the same thread to achieve no lock. Eliminate lock waiting time

Serial Parallelization

➢The distribution thread is split into driver thread and parse thread, running on both HTs to improve performance 。

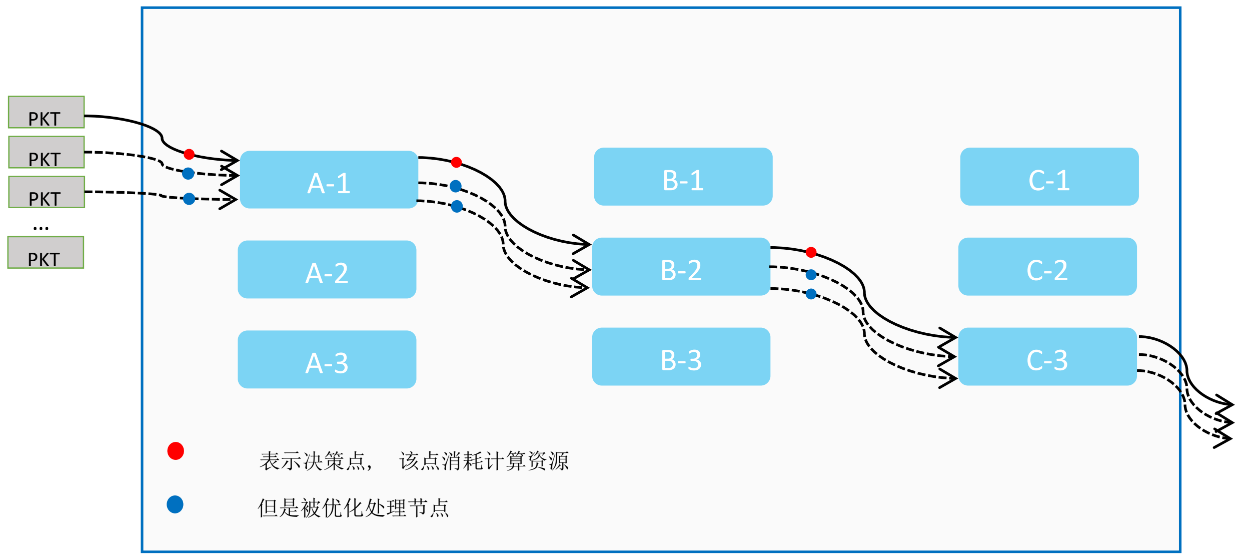




DPDK SUMMIT CHINA 2017

Offload Model

Fixed action， Improve performance



● 表示决策点， 该点消耗计算资源

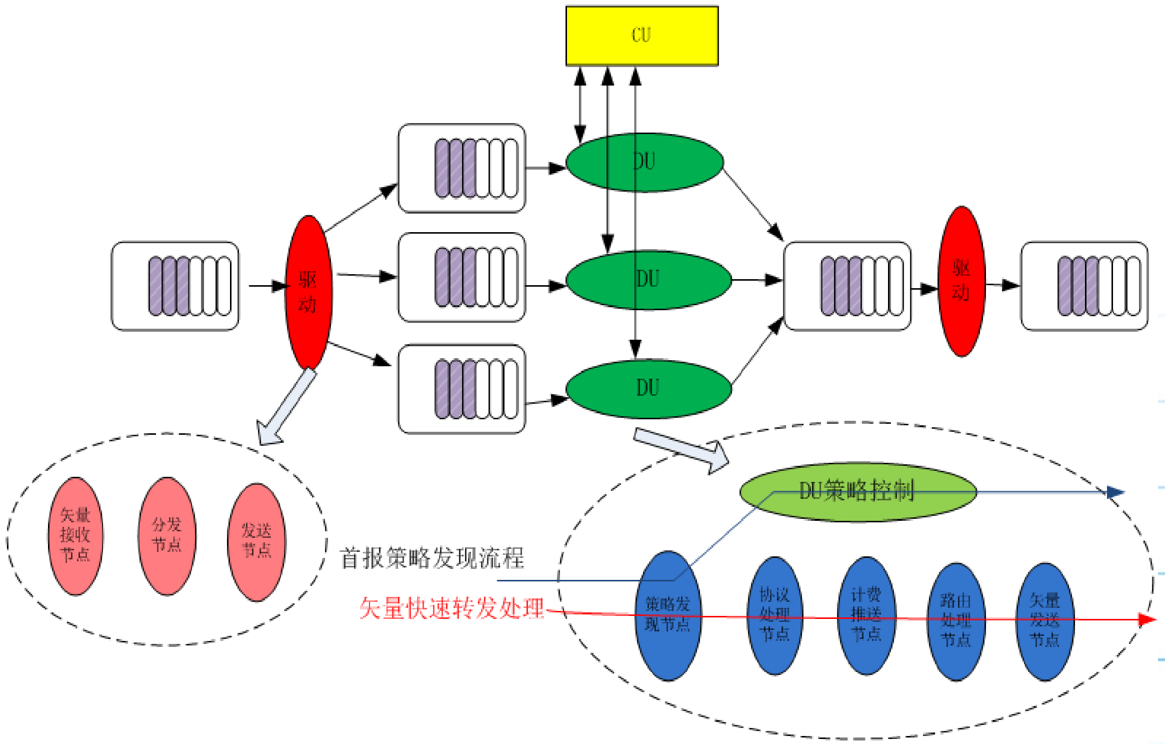
● 但是被优化处理节点

首包决策，固化执行路径

The first package for policy access，Follow the package fixed action

Software Parallel Scalability Optimization

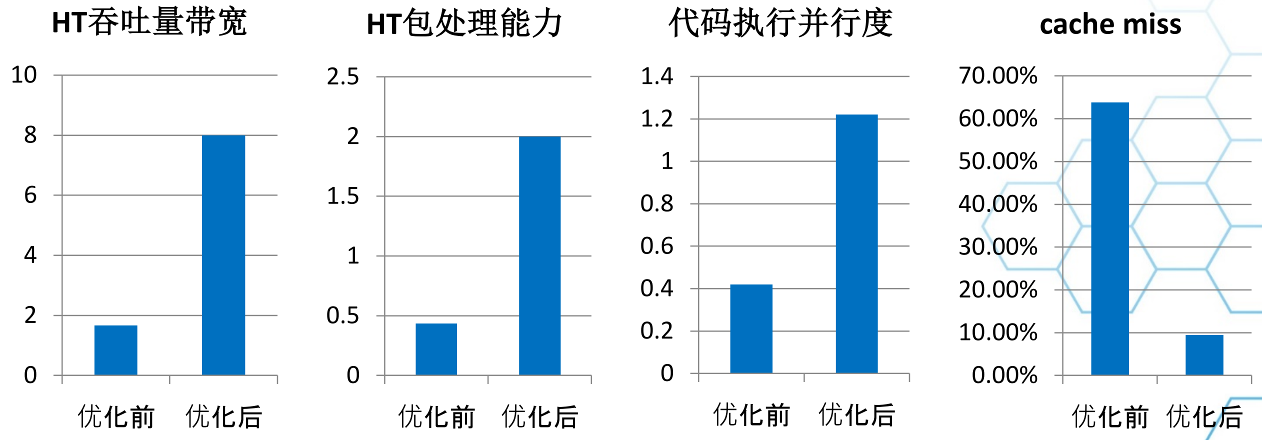
✓For a stream, not every message requires the whole process to determine the processing, only the first report to get the forwarding strategy can be, and the rest of the traffic all unloaded to a fixed vector node processing node;
✓For vector nodes to handle nodes, require extremely high performance, the code needs to be optimized to the simplest, and the code is optimized by white boxing metrics.
✓The details of the measurement will be detailed later



Effect

✓CPU occupancy rate dropped by 70%
✓Code parallelism increased by 3 times,
✓Cache miss drops by 54%.
✓Single thread processing capacity close to 2Mpps

	HT吞吐量带宽（Gbps）(512BYTE)	HT包处理能力(Mpps)	代码执行并行度（IPC）	cache miss
优化前	1.664G	0.436	0.4197	63.80%
优化后	8G	2.M	1.22	9.49%



Content Review

- **A software performance of the lean measurement optimization method: to guide the code to write the level of the definition of indicators, Optimize the code based on these metrics. Raise the code to perform parallelism, reduce cache miss.**
- **A Theorem: Amdahl's Law, Determine the Direction of Optimization, Eliminate System Bottlenecks, and Pursuit of Performance Parallel Expansion.**
- **DPDK opens up new ways and new ideas for code performance optimization;**

Thanks!!

欢迎关注**DPDK开源社区**