Network Acceleration and Performance Improvement

Lou, Fangliang
ZTE
Agenda

- Performance optimization concepts and methods
- Performance optimization crisis
- Software Performance Lean Measurements
- Enhance software parallel scalability
Performance Optimization Concepts

- RFC2544 协议性能测试定义

**Performance concept**

- back to back: burst packet capability caching packet capability
- Packet throughput pps Bps
- Packet loss rate: Bear the load capacity
- latency: The speed at which packets are processed

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

\[
\text{报文处理平均时间} = \frac{1}{\text{报文平均时间间隔}}
\]
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.
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:

- 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

1. Performance optimization concepts and methods
2. Performance optimization crisis
3. Software Performance Lean Measurements
4. Enhance software parallel scalability
Instruction four - layer water structure

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
}
```
### Instruction Flow Indicator

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

<table>
<thead>
<tr>
<th>Port 0</th>
<th>Port 1</th>
<th>Port 2, 3</th>
<th>Port 4</th>
<th>Port 5</th>
<th>Port 6</th>
<th>Port 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU, Shift</td>
<td>ALU, Fast LEA, BM</td>
<td>Load_Addr, Store_addr</td>
<td>Store_data</td>
<td>ALU, Fast LEA, BM</td>
<td>ALU, Shift, JEU</td>
<td>Store_addr, Simple_AGU</td>
</tr>
<tr>
<td>SIMD_Log, SIMD_msc</td>
<td>SIMD_ALU, SIMD_Log</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMD_Shifts</td>
<td>FMA/FP_mul, FP_add</td>
<td></td>
<td>Shuffle</td>
<td>FMA/FP_mul, FP_add</td>
<td>FP_mov, AES</td>
<td></td>
</tr>
<tr>
<td>2nd_Jeu</td>
<td>slow_int,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 指标 | 英文 | 中文
| IPC | instruction per cycle | 单周期执行指令数 |
| eIPC | effective instruction 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 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.

<table>
<thead>
<tr>
<th>指标</th>
<th>英文</th>
<th>中文</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache miss</td>
<td>Cache miss</td>
<td>Cache 没有命中的概率</td>
</tr>
</tbody>
</table>
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.

prefetch0 预取数据到 L1/L2/L3 cache
prefetch1 预取数据到 L2/L3 cache
prefetch2 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。
1) IPC（instruction per cycle）
   每周期指令数
2) eIPC（effective instruction per cycle）
   有效周期指令数
3) IPP（instruction 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，对于编核的程序，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

[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,

<table>
<thead>
<tr>
<th>Bit Position CPUID, AH, EBX</th>
<th>Event Name</th>
<th>UMask</th>
<th>Event Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>UnHalted Core Cycles</td>
<td>00H</td>
<td>3CH</td>
</tr>
<tr>
<td>1</td>
<td>Instruction Retired</td>
<td>00H</td>
<td>COH</td>
</tr>
<tr>
<td>2</td>
<td>UnHalted Reference Cycles</td>
<td>01H</td>
<td>3CH</td>
</tr>
<tr>
<td>3</td>
<td>LLC Reference</td>
<td>4FH</td>
<td>2EH</td>
</tr>
<tr>
<td>4</td>
<td>LLC Misses</td>
<td>41H</td>
<td>2EH</td>
</tr>
</tbody>
</table>
Tools

Perf

VTune
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

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 cannot 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.

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

- 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
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.

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

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

1. Performance optimization concepts and methods
2. Performance optimization crisis
3. Software Performance Lean Measurements
4. 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:
\( T(N) = B + \frac{T - B}{N} \)
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 Amdal law looks like this:

$$T(O, N) = \frac{B}{O} + \frac{(1 - \frac{B}{O})}{N}$$

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:

$$T(2, 5) = \frac{0.4}{2} + \frac{(1 - \frac{0.4}{2})}{5}$$

$$= 0.2 + \frac{(1 - 0.2)}{5}$$

$$= 0.2 + \frac{0.8}{5} = 0.2 + 0.16 = 0.36$$
Two Models

- 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.
Offload Model

Fixed action, Improve performance

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
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.
- DPDK opens up new ways and new ideas for code performance optimization;
Thanks!!