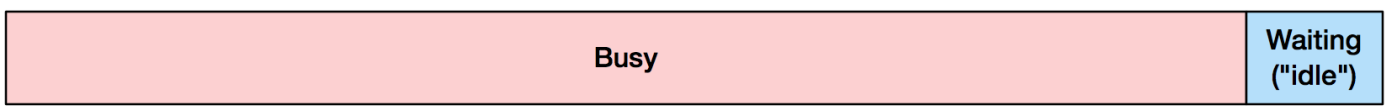
# CPU Utilization is Wrong

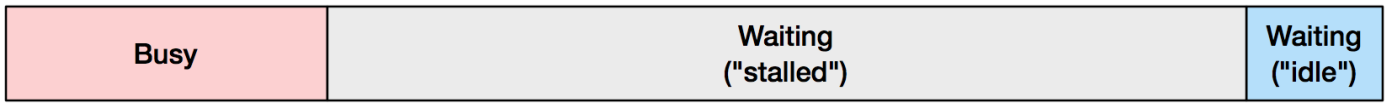
09 May 2017

The metric we all use for CPU utilization is deeply misleading, and getting worse every year. What is CPU utilization? How busy your processors are? No, that's not what it measures. Yes, I'm talking about the "%CPU" metric used *everywhere*, by *everyone*. In every performance monitoring product. In top(1).

What you may think 90% CPU utilization means:




What it might really mean:




Stalled means the processor was not making forward progress with instructions, and usually happens because it is waiting on memory I/O. The ratio I drew above (between busy and stalled) is what I typically see in production. Chances are, you're mostly stalled, but don't know it.

What does this mean for you? Understanding how much your CPUs are stalled can direct performance tuning efforts between reducing code or reducing memory I/O. Anyone looking at CPU performance, especially on clouds that auto scale based on CPU, would benefit from knowing the stalled component of their %CPU.

## What really is CPU Utilization?

The metric we call CPU utilization is really "non-idle time": the time the CPU was not running the idle thread. Your operating system kernel (whatever it is) usually tracks this during context switch. If a non-idle thread begins running, then stops 100 milliseconds later, the kernel considers that CPU utilized that entire time.

This metric is as old as time sharing systems. The Apollo Lunar Module guidance computer (a pioneering time sharing system) called its idle thread the "DUMMY JOB", and engineers tracked cycles running it vs real tasks as a important computer utilization metric. (I wrote about this before.)

So what's wrong with this?

Nowadays, CPUs have become much faster than main memory, and waiting on memory dominates what is still called "CPU utilization". When you see high %CPU in top(1), you might think of the processor as being the bottleneck – the CPU package under the heat sink and fan – when it's really those banks of DRAM.

This has been getting worse. For a long time processor manufacturers were scaling their clockspeed quicker than DRAM was scaling its access latency (the "CPU DRAM gap"). That levelled out around 2005 with 3 GHz processors, and since then processors have scaled using more cores and hyperthreads, plus multi-socket configurations, all putting more demand on the memory subsystem. Processor manufacturers have tried to reduce this memory bottleneck with larger and smarter CPU caches, and faster memory busses and interconnects. But we're still usually stalled.

## How to tell what the CPUs are really doing

By using Performance Monitoring Counters (PMCs): hardware counters that can be read using Linux perf, and other tools. For example, measuring the entire system for 10 seconds:

```
# perf stat -a -- sleep 10

 Performance counter stats for 'system wide':

     641398.723351      task-clock (msec)         #   64.116 CPUs utilized            (100.00%)
          379,651      context-switches          #    0.592 K/sec                    (100.00%)
           51,546      cpu-migrations            #    0.080 K/sec                    (100.00%)
       13,423,039      page-faults               #    0.021 M/sec
 1,433,972,173,374      cycles                    #    2.236 GHz                      (75.02%)
   <not supported>      stalled-cycles-frontend
   <not supported>      stalled-cycles-backend
 1,118,336,816,068      instructions              #    0.78  insns per cycle          (75.01%)
   249,644,142,804      branches                  #  389.218 M/sec                    (75.01%)
     7,791,449,769      branch-misses             #    3.12% of all branches          (75.01%)

      10.003794539 seconds time elapsed
```

The key metric here is **instructions per cycle** (`insns per cycle`: IPC), which shows on average how many instructions we were completed for each CPU clock cycle. The higher, the better (a simplification). The above example of 0.78 sounds not bad (78% busy?) until you realize that this processor's top speed is an IPC of 4.0. This is also known as *4-wide*, referring to the instruction fetch/decode path. Which means, the CPU can retire (complete) four instructions with every clock cycle. So an IPC of 0.78 on a 4-wide system, means the CPUs are running at 19.5% their top speed. Newer Intel processors may move to 5-wide.

There are hundreds more PMCs you can use to dig further: measuring stalled cycles directly by different types.

### In the cloud

If you are in a virtual environment, you might not have access to PMCs, depending on whether the hypervisor supports them for guests. I recently posted about The PMCs of EC2: Measuring IPC, showing how PMCs are now available for dedicated host types on the AWS EC2 Xen-based cloud.

## Interpretation and actionable items

If your **IPC is < 1.0**, you are likely memory stalled, and software tuning strategies include reducing memory I/O, and improving CPU caching and memory locality, especially on NUMA systems. Hardware tuning includes using processors with larger CPU caches, and faster memory, busses, and interconnects.

If your **IPC is > 1.0**, you are likely instruction bound. Look for ways to reduce code execution: eliminate unnecessary work, cache operations, etc. CPU flame graphs are a great tool for this investigation. For hardware tuning, try a faster clock rate, and more cores/hyperthreads.

For my above rules, I split on an IPC of 1.0. Where did I get that from? I made it up, based on my prior work with PMCs. Here's how you can get a value that's custom for your system and runtime: write two dummy workloads, one that is CPU bound, and one memory bound. Measure their IPC, then calculate their mid point.

## What performance monitoring products should tell you

Every performance tool should show IPC along with %CPU. Or break down %CPU into instruction-retired cycles vs stalled cycles, eg, %INS and %STL.

As for top(1), there is tiptop(1) for Linux, which shows IPC by process:

```
tiptop -                    [root]
Tasks:  96 total,   3 displayed                                screen  0: default

  PID [ %CPU] %SYS     P   Mcycle   Minstr   IPC   %MISS   %BMIS   %BUS COMMAND
 3897   35.3  28.5     4   274.06   178.23  0.65    0.06    0.00    0.0 java
 1319+   5.5   2.6     6    87.32   125.55  1.44    0.34    0.26    0.0 nm-applet
  900    0.9   0.0     6    25.91    55.55  2.14    0.12    0.21    0.0 dbus-daemo
```

## Other reasons CPU utilization is misleading

It's not just memory stall cycles that makes CPU utilization misleading. Other factors include:

- Temperature trips stalling the processor.
- Turboboost varying the clockrate.
- The kernel varying the clock rate with speed step.
- The problem with averages: 80% utilized over 1 minute, hiding bursts of 100%.
- Spin locks: the CPU is utilized, and has high IPC, but the app is not making logical forward progress.

## Update: is CPU utilization actually wrong?

There have been hundreds of comments on this post, here (below) and elsewhere (1, 2). Thanks to everyone for taking the time and the interest in this topic. To summarize my responses: I'm not talking about iowait at all (that's disk I/O), and there are actionable items if you know you are memory bound (see above).

But is CPU utilization actually wrong, or just deeply misleading? I think many people interpret high %CPU to mean that the processing unit is the bottleneck, which is wrong (as I said earlier). At that point you don't yet know, and it is often something external. Is the metric technically correct? If the CPU stall cycles can't be used by anything else, aren't they are therefore "utilized waiting" (which sounds like an oxymoron)? In some cases, yes, you could say that %CPU as an OS-level metric is technically correct, but deeply misleading. With

hyperthreads, however, those stalled cycles can now be used by another thread, so %CPU may count cycles as utilized that are in fact available. That's wrong. In this post I wanted to focus on the interpretation problem and suggested solutions, but yes, there are technical problems with this metric as well.

You might just say that utilization as a metric was already broken, as Adrian Cockcroft discussed previously.

## Conclusion

CPU utilization has become a deeply misleading metric: it includes cycles waiting on main memory, which can dominate modern workloads. Perhaps %CPU should be renamed to %CYC, short for cycles. You can figure out what %CPU really means by using additional metrics, including instructions per cycle (IPC). An IPC < 1.0 likely means memory bound, and an IPC > 1.0 likely means instruction bound. I covered IPC in my previous post, including an introduction to the Performance Monitoring Counters (PMCs) needed to measure it.

Performance monitoring products that show %CPU – which is all of them – should also show PMC metrics to explain what that means, and not mislead the end user. For example, they can show %CPU with IPC, and/or instruction-retired cycles vs stalled cycles. Armed with these metrics, developers and operators can choose how to better tune their applications and systems.

---

**brendangregg** Mod ➚ commenter • 2 years ago

Right, memory aside, %CPU doesn't factor the utilization of each processing unit. This is similar to problems with utilization measurements on a storage array that's shown as a single volume: 100% might mean one disk is busy all the time, but others are idle.

I think the CPU/memory problem is different, and worse, as memory is an external and different component. So %CPU is bound by some other resource on the system that isn't CPU.

2 ^ v • Share ›

**commenter** ➚ brendangregg • 2 years ago

Indeed, you are right. As a normal user you would not anticipate, that a high CPU usage is caused by another system component.

1 ^ v • Share ›

**Vojtech Kurka** ➚ commenter • a year ago

It's very similar to i/o wait time

^ v • Share ›

**Tose Nikolov** • 2 years ago

I would argue that CPU utilization is pretty much spot on as a metric. It is the time your process is using that core and no other process is able to use it. Which is what most people want to know from monitoring software. "What is using my cpu?"

The metrics you mentioned are solid for profiling tools aimed at speeding up a program, but not for ones that are meant to monitor a system. The IPC is meaningless for 99% of the people, it shows you a thing you can do nothing about.

Also: "If your IPC is > 1.0, you are likely instruction bound. Look for ways to reduce code execution..." No to this. You want that IPC as high as it goes. There is no such thing as instruction bound, That is code you need ran, and the higher this number the better it is. The way you know if you are instruction bound is to check the time. If your program runs in 50s and you want it done in 30s(or 30ms), that is when you are "instruction bound".

6 ^ v • Share ›

**brendangregg** Mod ➚ Tose Nikolov • 2 years ago

Other processes can use those "utilized" CPU cycles via hyperthreading. So the CPU "utilized" metric really is wrong. Without hyperthreading, you could argue that the CPU is "utilized waiting" (which sounds like an oxymoron), which might make the metric technically not wrong, but it is deeply misleading.

CPU is such an important metric, and there are things everyone can do (developers and operators) if they know their system is memory bound. One of our systems became memory bound after it was moved from 1 node NUMA to 2 node NUMA, and the operator has the choice of how many nodes to deploy on. I could go on with other tunables, but sure, I get that most people do not know about IPC and have not tuned based on it yet. That's why I wrote the post.

Arguing that people who monitor a system don't need to know IPC -- or any other metric to indicate where they are truly bound -- I think is a harmful argument for our industry. We should be convincing performance monitoring companies to provide the metrics their users need to understand %CPU -- not pretending that it's unnecessary. That's the kind of argument I'd expect to hear from a performance monitoring company, as an excuse for not doing work

expect to hear from a performance monitoring company, as an excuse for not doing work.

The word "bound" is used to describe the limiting factor. If the limiting factor is really just instruction execution, then you are instruction bound. The biggest wins, in that case, is by eliminating unnecessary work (often, at my company, by using a CPU flame graph). That's why I said to reduce code execution. Sorry if that wasn't clear.

6 ^ v • Share ›

**MarcusK** ➚ brendangregg • 2 years ago

agreed. But there is also a relationship with data locality as Roman showed graphically, so I say performance tools should include IPC *and* data locality.

^ v • Share ›

**smaudet** ➚ brendangregg • 2 years ago

I think her argument was for consumers, for whom none of the options mentioned are really viable. Nobody is going to go out and buy a new cpu just because they have an issue, or recompile code, etc.

Agreed that its a fairly useless metric for building systems that run in specialized scenarios, however.

For the more savy consumer, maybe willing to compile their own code, there is htop, this program lets you divide out the various metrics, although HT is still confusing to look at even then, because increased utilization still doesn't scale properly.

^ v • Share ›

**Tarjei T. Jensen** ➚ smaudet • a year ago

Buying faster memory might be a viable option for consumers.

Having said that, buying more memory is usually the right answer.

^ v • Share ›

**MarcusK** • 2 years ago

An process with IPC < 1.0 is obviously memory bound, but IPC > 1.0 is not so obvious. I think that we can say that a process with IPC > 2.0 is definitely instruction bound and the program benefits from a faster CPU, while an IPC between 1.0 and 2.0 means that the program is well-behaving on the hardware.

2 ^ v • Share ›

**brendangregg** Mod ➚ MarcusK • 2 years ago

Makes sense to me, and these will need future refinement. Whenever I get a new processor type I run some known workloads to see where the IPC is.

3 ^ v • Share ›

**Travis Downs** ➚ MarcusK • a year ago

It is common to see processes with IPC < 1 which are not memory bound, but just achieve low IPC due to branchy code, dependency chains and cache latency. I would guess that memory latency might still be the #1 reason for very low IPC, but certainly the other reasons combined add up to more than memory latency.

In any case, it's kind of pointless to discuss the reasons for low IPC across some kind of hypothetical corpus of applications (I'll just pick a corpus that suits my argument and you yours, I guess) - what matters is your application, and there you can simply measure it as the many other great posts here explain. Making decisions based on a heuristic like "IPC < 1

means memory bound", even if right 40% or 80% of the time, would be nuts since once you can measure IPC you can probably measure with the same tool all of the other counters which will tell you _exactly_ where your retirement slots are going.

Share ›

**mwf** • 2 years ago

Thanks. And here's hoping your second graphic helps a few folks realize that if they are "CPU bound" and they move to a higher clock speed without raising the rate and/or reducing contention of the memory bus they will often have paid money just make the pink smaller and the "waiting (stalled)" bigger.

2 Share ›

**brendangregg** Mod ➚ mwf • 2 years ago

exactly!

1 Share ›

**bigbadlosangeles** • 2 years ago

Thanks for posting this, I've been trying to explain to folks for sometime that CPU percent utilized isn't what they expect it is. perf seems like the best bet for higher precision, but at a more macro level I expect this is where load average (with its own nuances) helps out. At the end of the day we're worried about the work that isn't getting done, that is, the number of processes that are stuck in the run queue?

2 Share ›

**William Tu** • 2 years ago

It would be even better if the stalled-cycles-frontend and stalled-cycles-backend are shown. The front-end is responsible for the fetch and decode phases, while the back-end executes the instructions. So if I see high frontend stalled, usually I will check my branch prediction, tuning the likely() and unlikely() in my condition, and it usually helps. If it is mostly backend stalled, then it's mostly due to memory IO bound.

1 Share ›

**Roman** • 2 years ago

Funny enough, this reminds me of stuff I did for my PhD:





see more

1 • Share ›

**brendangregg** Mod ➚ Roman • 2 years ago

I love the breakdowns. This is what I want to see in performance monitoring products. :)

1 • Share ›

**Papastavros Vaggelis** • 2 years ago

Also check the performance application programming interface

http://icl.cs.utk.edu/papi/

1 • Share ›

**Rick Gorton** • 2 years ago

Perf (Linux) is a very coarse high level, coarse grained, heavyweight mechanism. AMD's LWP Light Weight Profiling is a much better method. Excepting Linux Politics, that is. Yes, Ingo is an ass.

2 • Share ›

**Eddie** ➚ Rick Gorton • 2 years ago

AMD ditched LWP starting in 2015, and most instructions for Bulldozer family of chips.

"Zen will support all the extensions of previous AMD CPU cores except four specific ones introduced with bulldozer and these include TBM, FMA4, XOP, and LWP. These AMD exclusive extensions will no longer be supported due to their underutilization to save both area and power that can go into building more useful structures in the core."

4 • Share ›

**Travis Downs** ➚ Rick Gorton • a year ago

To be fair to Intel, they introduced "Intel Processor Trace" which covers a lot of the same ground as AMD's (since discontinued) LWP, and then some. It's not zero overhead, but it is low overhead and recent version of perf do support it. Essentially you can record the entire execution history of an application in a compressed format, and then view it with flame graphs or do whatever other analysis you want.

• Share ›

**brendangregg** Mod ➚ Travis Downs • a year ago

Which is great, but it's not currently available in virtualized machine clouds, where many companies and workloads are moving. EC2 just enabled the architectural set of PMCs. The best we can do today is:

1. Measure IPC.
2. Work with involved parties on enabling more PMCs (and BTS, LBR, etc) and processor trace.

I've worked on both. If you're on the cloud and want to help with 2, please do.

1 • Share ›

**Travis Downs** ➚ brendangregg • a year ago

Yes, I hope we see it there at some point. The basic pattern seems to be (1) Intel adds support for something cool, then (2) various miscellaneous tools or kernel modules add support then (3) perf adds support and then (maybe) (4) cloud providers add support.

In the past (3) often took a long time but perf seems to be in a kind of golden era now where support for new features is coming quite quickly and contrary to

my order above (2 before 3) perf has recent almost been the first tool to support some of the new stuff. Between various Intel guys and Andi Kleen, perf seems to be moving pretty quickly these days, but the "mostly do it in the kernel" model often makes it tough for end-users to adopt it.

(4) is very new and very cool as you point out! I don't know how it will pan out in practice but of course more demand makes everything more likely. I don't have any direct leverage over any of the primary IaaS guys though. Just calling out people who are adding support here on this blog probably helps a lot.

I also feel like the "do it in the kernel" approach perf (more precisely, perf_events) takes may actually help cloud adoption since a lot of the work is already in the kernel and that's anyway the right place to do it for exposure in paravirtualized environments. In principle anyway :)

Share ›

**Travis Downs** • a year ago

Catchy title, but to be fair, I think CPU utilization is about as good as a single metric could be (ignoring the issue with iowait, which isn't being discussed here) and is plainly not _wrong_ in both a theoretical sense (it conforms to its definition) and in a practical sense (it provides a simple, easy-to-use definition that is widely applicable for diagnosing system performance).

The core premise here seems to be that most people draw a strong distinction between waiting on memory and waiting on any other CPU resource, and would expect "CPU utilization" not to include cycles waiting on memory.

This seems to me a problematic assumption. My experience is that people have a variety of levels of understanding of low-level performance concerns. Those with a relatively basic understanding are not likely to even have a mental model of "DRAM wait" versus "other core waits/bottlenecks" and so the metric is accurate for them. Those with a deeper understanding probably already know exactly that CPU % is a "active period" measurement calculated by the scheduler, and so inherently includes things like waits on main memory.

Second, you might look at how actionable a "CPU utilization" metric is for typical groups who monitor these things - which mostly boils down to various flavors of operations teams. For these people, the CPU utilization metric has a specific and precise meaning which is useful (again, ignoring

see more

Share ›

**brendangregg** Mod ➚ Travis Downs • a year ago

First: If someone doesn't have a mental model of DRAM wait, the metric isn't accurate at all. Given high "CPU utilization" in the cloud, they are switching from their memory-optimized instances (better memory bandwidth, CPU caches) to CPU-optimized instances (faster clock cycle), then wondering why performance is worse. They've been misled. Instead of trying to rationalizing these misleading metrics, I'd rather fix them.

Second: As for how actionable -- they are by everyone, system administrators and developers. Much of the industry doesn't do this type of tuning at the moment, but that doesn't mean they can't, and they don't because monitoring products don't suggest they should. That needs to change.

Third: My division in the diagrams was not into DRAM latency, but rather stalled cycles. I use DRAM latency as an example, but one of the metrics I'm proposing is IPC, which covers all stall cycle types.

So, not just a "catchy title" after all.

Share ›

**Travis Downs** ➚ brendangregg • a year ago

I misunderstood: I had thought that you were specifically calling out DRAM latency as the stall cycles of interest. If you are just talking about presenting pure IPC, regardless of the stall, then that complaint disappears. I think I can be at least partly forgiven for the misunderstanding since even on a re-read it really seems like DRAM is being used not as an example of one type of stall, but as "the" stall that results in waiting, e.g., :

> Nowadays, CPUs have become much faster than main memory, and waiting on memory dominates what is still called "CPU utilization". When you see high %CPU in top(1), you might think of the processor as being the bottleneck – the CPU package under the heat sink and fan – when it's really those banks of DRAM.

I still totally disagree with the assessment that CPU % is wrong, and that somehow IPC should "replace" it. They are totally different metrics useful in different scenarios. CPU % gives insight into exactly how loaded your machine is, and is why tools like top, which focus on system-wide resource usage display it. It helps you determine "is my machine maxed out". CPU usage also scales naturally to multiple cores and multi-threaded process (how would you do the same for IPC - there are a few ways but none are confusion free).

**see more**

Share ›

**brendangregg** Mod ➚ Travis Downs • a year ago

"Performance monitoring products that show %CPU – which is all of them – should also show PMC metrics to explain what that means, and not mislead the end user. For example, they can show %CPU with IPC, and/or instruction-retired cycles vs stalled cycles."

That's in the conclusion. I wasn't suggesting that IPC "replace" %CPU at all -- it can't -- I suggested provided it with %CPU for context (which is what tiptop does). You're disagreeing with things I didn't say. I'm also not wedded to IPC -- I'm providing it as an example. It's reasonably well known, works reasonably well, is already provided by many tools (perf stat), and is an improvement (with %CPU) over what people use today (just %CPU). I think you could argue that stalled cycles per instruction (SCPI) is better (with %CPU), and that the top down metrics are better. Great!

Share ›

**Travis Downs** ➚ brendangregg • a year ago

Well at least in your title you argue that CPU % is wrong and then make the case that IPC is a better CPU % since it accounts for stall cycles.

If your claim is in fact that CPU % is right after all and that it can be also be enhanced by PMU-based metrics such as IPC, then we are in violent agreement!

Well in mostly violent agreement anyway: I think IPC can be much more misleading and tough to use, unless it is accompanied by enough additional metrics to do a basic "top down" like breakdown of the unused slots. Examples of how IPC can be confusing:

(1) Low IPC caused by branch mispredicts implies very different remedies versus say low IPC caused by memory latency. Repeat this point for most of the other various causes of low IPC.

(2) There is a correlation between "efficient code" and "high IPC" but it is far from exact! Case in point: after vectorization IPC often drops despite the new code getting much more work done per unit time: because vectorized code

see more

Share ›

**brendangregg** Mod ➚ Travis Downs • a year ago

I welcome criticism, but you are attacking something I didn't say, and I already corrected you on this. I would keep responding, but it's going to get very repetitive, and I already covered it in the original article.

Share ›

**Travis Downs** ➚ brendangregg • a year ago

Fair enough. Your claims seemed clear enough to me, but perhaps I am reading a different article than you or maybe I am parsing it differently.

Share ›

**Garnett D.** • a year ago

Makes sense but it seems difficult to implement.

Share ›

**doublemarket** • 2 years ago

Hello, thank you for the great post. Since this post became popular in Japan as well, I translated this into Japanese.

https://yakst.com/ja/posts/...

If it's a problem for you, please let me know. Thank you again!

Share ›

**Armin Catovic** • 2 years ago

G'day Brendan! I agree that performance monitoring products should include PMCs. I would go beyond what you stated and also mention loads and stores, branch prediction counters, and DDR bandwidth as very handy metrics to go along with CPU utilisation, i.e. the option to have various PMC counters monitored over time should be there for all to use, at the cost of some overhead. Saying that though, the headline of your article is a bit misleading. I wouldn't say CPU utilisation is wrong. On Linux-based systems, the workrate of a system is exposed via Linux scheduler. As far as applications/processes are concerned, if it's not scheduled, it doesn't exist, and CPU% column in "top" directly extrapolates what the scheduler in kernel exposes via stime/utime/rtime. Monitoring this is an elegant way of seeing what threads are being scheduled, how frequently, and what demand they put on the scheduler. It's definitely worth coupling that with PMCs though! From my experience working with embedded PPC and ARM systems, I actually found it very useful to couple utilisation together with instructions, loads, stores, and BTB counters, per cycle (or unit of time). This tells you if your code is very scattered, or whether you're making many trips down memory lane. But CPU% has always been a fantastic guide. I agree on more complex cloud systems CPU utilisation is insufficient (alone) but should not be called wrong! Since you're a cloud performance champion, I would love to see you push AWS and others into exposing more performance metrics, and not just limiting to select few and only for very specific clusters! Thanks and keep up the great work!

few and only for very specific clusters. Thanks and keep up the great work.

Share ›

**brendangregg** Mod ➚ Armin Catovic • 2 years ago

Yes, I'm for more PMCs, and have been working on wish list extended sets beyond the architectural ones. At a bare minimum, I'd like tools that show and monitor %CPU to also show IPC, just as a starting point.

I think a lot of people have interpreted high %CPU to mean the processing unit is the bottleneck, which is wrong (hence the title). At that point you don't yet know. Once you understand how it all works, you might call it just deeply misleading. (There's also hyperthreads, which really can break %CPU, but I didn't talk about that in this post.)

Your points about %CPU being useful: sure, and the way it's used for some tasks like load balancing it works fine (and isn't wrong). Imagine if it was called %CYC, short for cycles, showing utilized system cycles? (Again, hyperthreads breaks it, but ignoring that for a moment.) If the system is running at 90% CYC, that's not telling you the wrong thing. Anyway, just one idea.

1 Share ›

**Justin Flowers** • 2 years ago

Haha, this reminds me of a very similar blog post I wrote about memory management. Seems both are quite deceptive!

http://jflow.io/?p=152

Share ›

**smaudet** ➚ Justin Flowers • 2 years ago

Again, while all true, those in the know, or who need to, know. Those playing Overwatch don't. The added complexity is irrelevant to them. And, its your job to make sure those ugly details don't see the light of day. :)

Share ›

**Justin Flowers** ➚ smaudet • 2 years ago

I think it's dangerous for sometime to have confidence in an idea that's wrong, though. I understand that the average person might not need the detailed comprehension that's discussed in these posts. Maybe it's important for the average person to just understand that the number they're seeing for "memory used" is very arbitrary and could be hiding quite a bit.

1 Share ›

**smaudet** ➚ Justin Flowers • 2 years ago

I'm not a huge fan of the memory used metric in e.g. windows - something simpler is good for most users, the issue isn't with the metric but the presentation.

To play devils advocate, if you really wanted to, reduce usage, down to t-shirt sizes. Thats what most users are interested in, anyways.

Share ›

**Justin Flowers** ➚ smaudet • 2 years ago

Hmm, its an interesting idea. Maybe if it wasn't so specific (like "exactly 250,234 kb used") and more like "eeehh, you have.... MEDIUM usage" users might understand that the number is less scientifically rigid and more produced

from a combination of black magic and a coin toss.

Share ›

Avatar This comment was deleted.

**Ivan Kurnosov** Guest • 2 years ago

Ratio between busy and idle does not make much sense: you can have it very high or very low, but neither tells whether it's good or bad.

Share ›

**Brandon Ryan** • 2 years ago

Check the spelling of "Turbobost"

Share ›

**brendangregg** Mod Brandon Ryan • 2 years ago

thanks!

Share ›

**David Weinehall** • 2 years ago

Surely having the IPC information shown isn't all that useful without also showing the IPC width?

Share ›

**brendangregg** Mod David Weinehall • 2 years ago

Well, we already know it's 4 for all our systems, but that's going to change soon with Skylake, so yes, it's something we'll need to know once some systems are 4 and some 5!

Share ›

**Travis Downs** brendangregg • a year ago

Actually Skylake doesn't materially change the underlying width of the CPU pipeline. They did bump the "legacy decoder" delivery rate from 4 to 5 and the uop cache delivery rate from 4 to 6, but the core bottleneck of 4 in the renamer still remains. So you can't go wider than 4, even on Skylake - a restriction that has been around since Sandy Bridge.

Note that all the numbers above are all "fused domain uops" which *mostly* correspond to instructions, but not always. If you really want to talk about instructions, the max IPC of Intel chips all the way back to Sandy Bridge has been ~6, which you get on highly artificial code that has two fused-and-not-taken branches per 4 instructions. That's the same in Skylake.

Personally I think if you have only Intel hardware looking at fused uops is probably the cleanest mapping for actually reasoning about the performance of code, but instructions isn't terrible either. If you want to mix AMD and Intel then instructions gets a big boost since you can't directly compare Intel's two flavors of uops with AMDs.

FWIW, Ryzen is 5 wide by the "fused uops" metric (actually mops which are pretty close), which is a bit of a coup for them - but oddly the 5 mops can only come from at most 4 instructions (ignoring branch fusion here). So you have this big debate now about whether Ryzen is wider or not than [Sandy Bridge ... Skylake]. I'd say as a practical matter it is, by a touch since any cycles 2 mop instructions mean it is 5-wide(ish). It still usually gets lower IPC (Ivy Bridge approximately) though due to other

bottlenecks.

Share ›

**Ryan Lovett** ➚ brendangregg • 2 years ago

Is there a tool that can query and report a system's CPU IPC width?

edit: i see how using noploop can be used for this.

Share ›

Load more comments

Subscribe Add Disqus to your siteAdd DisqusAdd Disqus' Privacy PolicyPrivacy PolicyPrivacy

*You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).*



About this blog