



(<https://www.pugetsystems.com/>)

[Create Account \(/register.php\)](/register.php) | [Log In \(/login.php\)](/login.php)

[Home \(https://www.pugetsystems.com/\)](https://www.pugetsystems.com/) / [View All Articles \(/all_articles.php\)](/all_articles.php)

/ [Estimating CPU Performance using Amdahls Law](#)

Estimating CPU Performance using Amdahls Law

Written on May 4, 2015 by Matt Bach (<https://www.pugetsystems.com/bios.php?name=mattbach>)

Share: [f](#) 7 [t](#) [g+](#) 4 [in](#) [+](#) 15

Table of Contents:

1. Introduction
2. What is Amdahls Law?
3. Amdahls Law Limitations
4. Step 1: Test your program with various number of CPU cores
5. Step 2: Determining the parallelization fraction
6. Step 3: Estimate CPU performance using the parallelization fraction
7. Easy Mode - Using a Google Doc spreadsheet
8. Conclusion

Introduction

If you are in the market for a new computer (or thinking of upgrading your current system), choosing the right CPU can be a daunting - yet incredibly important - task. Just in our current product line here at Puget Systems, we are selling about 60 different Intel CPU models each with their own unique specifications. While the actual list of CPUs you will need to choose between will be a bit smaller than that based on your other system requirements (ECC RAM, Mobile, PCI-E lanes, etc.) it is still incredibly difficult to determine which CPU will give you the best possible performance while staying within your budget.

While Tom's Hardware, Anandtech, and a multitude of other hardware review sites do a great job reviewing and comparing different CPUs, unless they specifically test the application(s) you personally use their results may not accurately reflect the performance that you would see. After all, as good as those sites are if they were to test every possible application they simply would not be able to complete their testing by the time the CPU becomes obsolete!

When you are choosing a CPU, there are two main specifications you need to pay attention to that define the relative performance of CPUs:

1. The **frequency** is essentially how many operations a single CPU core can complete in a second (how fast it is).
2. The **number of cores** is how many physical cores there are within a CPU (how many operations it can run simultaneously).



(<https://www.pugetsystems.com/images/id=36487&width=800&height=800>)

This doesn't take into account any differences in architecture (AMD versus Intel, Haswell versus Ivy Bridge, etc.) but when comparing two CPUs from the same family they are the two main specifications that determine the relative performance capability of a CPU. If your software only uses a single core, the frequency is a decent indicator of how well a CPU will perform. However, if your software is able to utilize multiple CPU cores it becomes very difficult to estimate the performance of different CPU models since almost no program is going to be 100% efficient at using those cores. The trick is to determine exactly how efficient your program is at using multiple CPU cores (it's parallelization efficiency) and use that number to estimate the performance of different CPU models.

To calculate the parallelization efficiency, you need to use a mathematical equation called Amdahl's Law. We were first introduced to this equation about a year and a half ago when we hired a Dr. Donald Kinghorn (<https://www.pugetsystems.com/bios.php?name=donkinghorn>) to help us get established in the scientific computing market. He has been invaluable as a resource in that segment (just check out our HPC blog section (https://www.pugetsystems.com/all_hpc.php) for a sample of what we have learned from him so far), but the knowledge he has brought to Puget Systems has been useful in many ways we never anticipated - including the practical application of Amdahl's Law.

What is Amdahls Law?

At the most basic level, Amdahl's Law is a way of showing that unless a program (or part of a program) is 100% efficient at using multiple CPU cores, you will receive less and less of a benefit by adding more cores. At a certain point - which can be mathematically calculated once you know the parallelization efficiency - you will receive better performance by using fewer cores that run at a higher frequency than using more cores that run at a lower frequency.

Amdahl's Law:

$$S(n) = \frac{T(1)}{T(n)} = \frac{T(1)}{T(1)(B + \frac{1}{n}(1 - B))}$$

- $S(n)$ is the theoretical speedup
- $T(n)$ is the time an algorithm takes to finish when running n threads
- B is the fraction of the algorithm that is strictly serial (so $1-B$ is how much of the program can be run in parallel)

Unless you deal with complex equations regularly, this may be a bit daunting of an equation. However, since we are primarily concerned with the maximum speedup that can be achieved by increasing the number of CPU cores, this equation can be simplified a bit into the following:

Parallelization Formula:

$$S(n) = \frac{1}{(1 - P) + \frac{P}{n}}$$

- $S(n)$ is the theoretical speedup
- P is the fraction of the algorithm that can be made parallel
- n is the number of CPU threads

What this is basically saying is that the amount of speedup a program will see by using n cores is based on how much of the program is serial (can only be run on a single CPU core) and how much of it is parallel (can be split up among multiple CPU cores).

In order to use this equation, you first need to determine the parallelization efficiency of your program. With that number, you can then use Amdahl's Law and a CPU's frequency to fairly accurately estimate the performance of almost any CPU that uses a similar architecture to the CPU you used for testing. While you are certainly invited to follow this guide in its entirety, if you are more concerned about actually estimating a CPU's performance than all the math behind it feel free to skip ahead to the Easy Mode - Using a Google Doc spreadsheet section.

Amdahls Law Limitations

While the method we described above is great for determining how much of a program can be run in parallel, it (and Amdahl's Law in general) has some limitations:

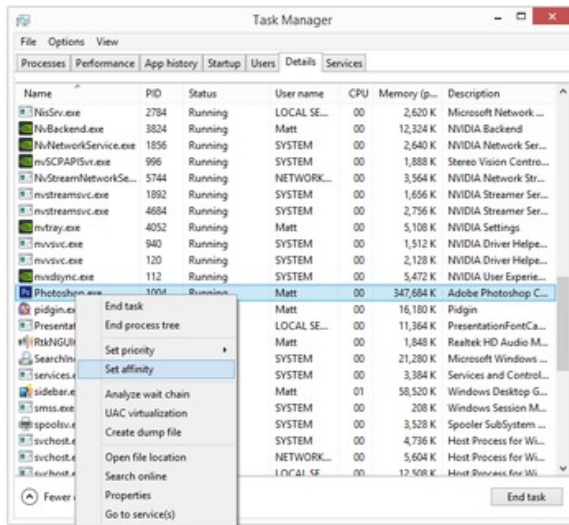
- **Not every action done in a program will have the same amount of parallelization.** If you look at the results of our recent article: Adobe Photoshop CC CPU Multi-threading Performance (<https://www.pugetsystems.com/labs/articles/Adobe-Photoshop-CC-CPU-Multi-threading-Performance-625/>) you will notice that how many CPU cores Photoshop can use varies greatly depending on what you are actually doing. You can mitigate this limitation somewhat by testing various tasks and calculating the parallelization efficiency for each task individually (which is what we did), but depending on the program it may not be feasible to test every single possible action.
- **Amdahl's Law only applies if the CPU is the bottleneck.** If what you are doing is not being limited by the CPU, you will find that after a certain number of cores you stop seeing any performance gain. If your video card, RAM, or hard drive performance is preventing the program from running any faster, adding more CPU cores will never help even if the program is 100% parallel. Also, keep in mind that if you end up purchasing a faster CPU than the one you tested with, it is entirely possible that the new CPU will be fast enough that something else in the system (RAM, HD, GPU, etc.) may then become the bottleneck and limit the performance of your new, faster CPU.
- **Many programs are hard-coded to use a certain number of cores.** Even if it may be possible for a program to try to use more cores, many programs have a hard-set number of CPU cores that can be utilized. In fact, a large majority of software available today still only uses a single CPU core! This is done for a variety of reasons ranging from the nature of what the program is doing making it non-conducive to using multiple CPU cores to it simply being easier to program for a fixed number of cores.
- **Estimating the performance of a CPU will only be accurate for CPUs based similar architecture.** If the CPU you used to determine the parallel efficiency of a program is vastly different than the CPU you are considering purchasing, you may not be able to accurately estimate the performance of a CPU. Even outside of AMD vs Intel CPUs, if the CPU you used to test is more than a generation or two old the actual performance of a newer CPU may be vastly different (and usually faster) than what Amdahl's Law will estimate. You can still accurately calculate the parallel efficiency and use that to compare the relative performance of two or more CPUs that use the same architecture but you won't be able to determine more than a general idea of the actual performance.

Step 1: Test your program with various number of CPU cores

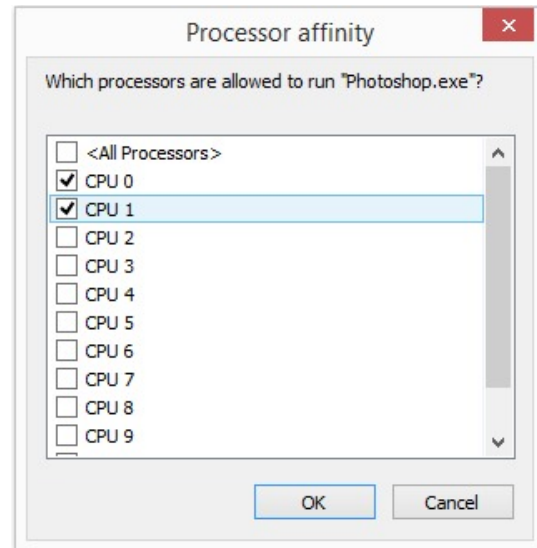
Unfortunately, determining the parallelization efficiency of a program is not something you can find just by looking in a ReadMe.txt file. The easiest way we have found to do this is to simply run your program and time how long it takes to complete a task with the number of CPU cores it can use limited artificially. Luckily, you don't need to change out your CPU a bunch of times to do this. Instead, you can simply set the program's affinity through Task Manager in Windows (or by using the "taskset" command in Linux). This is not as good as completely disabling the CPU cores through the BIOS - which is possible on some motherboards - but we have found it to be much more accurate than you would expect.

If you want your results to be as accurate as possible, we recommend disabling "Turbo Boost" in the BIOS before performing your tests. Intel (and AMD) CPUs turbo to different frequencies depending on the number of cores that are being used which can throw off your Amdahl's Law curve significantly. Disabling Turbo Boost removes this variable.

To set the affinity, simply launch the program you want to test, open Task Manager, right-click on the program listing under Details, select "Set Affinity", and choose the threads that you want to allow the program to use. Note that if your CPU supports Hyperthreading there will actually be twice as many threads listed than your CPU actually has cores. You can either disable Hyperthreading in the BIOS before doing your testing, or simply select two threads for every CPU core you want to test. Hyperthreading threads are always listed immediately after the physical core in Windows, so you would select two threads for every CPU core you want the program to use. In other words, selecting threads 1&2 will allow the program to just use a single CPU core, selecting threads 1-4 will allow the program to use two CPU cores, etc., etc.



(https://www.pugetsystems.com/images/pic_disp.php?id=36486&width=800&height=800)



(https://www.pugetsystems.com/images/pic_disp.php?id=36484&width=800&height=800)

Note that setting the affinity only lasts until the program is closed. The next time you run the program, you have to re-set the affinity again. However, if you want to quickly test a single action using various numbers of CPU cores, you don't have to close the program before changing the affinity - just click on "Set Affinity" and change it on the fly. However, you will get more accurate results by closing the program between runs as that will clean out the RAM that is already allocated to the program.

With the ability to set how many CPU cores a program can use, all you need to do is perform a repeatable action using a variety of CPU cores. For example, you may time how long it takes to complete a render in AutoCAD or export images in Lightroom using a variety of CPU cores. The larger the variety of number of cores you test the better, but you need to at least test with a single CPU core and all possible CPU cores. If possible, we recommend testing with as many combinations as possible (so if you have an eight-core CPU, test with 1,2,3,4,5,6,7, and 8 cores).

Step 2: Determining the parallelization fraction

At this point, you should have a list that shows how long it took your program to complete an action using various numbers of CPU cores. Just to have an example, lets say your results look like those in the "Action Time (seconds)" column in the chart to the right:

The easiest way we have found to use these results to determine the parallelization efficiency of a program is to first determine how much faster the program completed the task with N cores versus how long it took with a single core. To find this out, you simply need to divide how long the action took with a single core by how long it took with N cores. In our example, for two cores the speedup is 645.4/328.3 which equals 1.97 . Fill this in for each row and we can use these numbers to determine the parallelization fraction of the program.

There is a complex mathematical way to use the actual speedup numbers to directly find the parallelization fraction using non-linear least squares curve fitting, but

# of Cores	Action Time (seconds)	Actual Speedup	Amdahl's Law Speedup (97% efficient)
1	645.4	1	1
2	328.3	1.97	1.95
3	230	2.8	2.8
4	172	3.75	3.67
5	140.3	4.6	4.5
6	117.5	5.5	5.2
7	108	6	5.9
8	97.8	6.6	6.6

the easiest way we have found is to simply guess at the fraction, see how close the results are, then tweak it until the actual speedup is close to the speedup calculated using Amdahl's Law. Using a program like Excel or Google Doc's Sheets makes this much easier, but you can do it with just a calculator and a pad of paper if you want to do it manually and have hours to kill.

To find the parallelization fraction, you need to use the parallelization equation we listed earlier and plug in different values for P:

$$S(n) = \frac{1}{(1 - P) + \frac{P}{n}}$$

A good place to start might be to try P=.8 (or 80% parallel efficient) and perform this calculation for each # of cores. For example, for 4 cores the equation would be

$$S(n) = \frac{1}{(1 - .8) + \frac{.8}{4}}$$

which equals 2.5. Compare this to our actual speedup in our example (which was 3.75) and you will see that our example program is actually more than 80% efficient so we need to increase the parallelization fraction to something higher. In our case, the actual fraction was .97 (97%) which is pretty decent. You will notice that the results don't line up perfectly every single time since there is a certain margin of error that always exist when you run benchmarks - you simply have to average it out and get it as close as you can. Having this in a spreadsheet where you can graph both data series makes it much easier (see the Easy Mode - Using a Google Doc spreadsheet section for a link to a Google Doc with all the calculations already performed and a graph setup).

Step 3: Estimate CPU performance using the parallelization fraction

Once you have the parallelization fraction, you can use it to estimate the performance of any other CPU that uses the same or similar architecture as the CPU. If you are interested in a CPU that uses an entirely different architecture, you can still use this method to determine the relative difference in performance between a number of different CPU models from the same family, but it will likely not be an accurate representation of the actual performance you would see with that CPU.

To estimate a CPU's performance, you need to know the operating frequency and how many cores both the CPU you used to benchmark with and the CPU you are interested in has. With those specs in hand, you first need to calculate how many effective cores both CPUs have which is done by using the equation:

$$EffectiveCores = \frac{1}{(1 - P) + \frac{P}{CPU_{Cores}}}$$

Basically, this is using the same parallelization equation we used earlier only using the actual number of cores the CPU has. This gives us the effective number of CPU cores the CPU has when running your program if the program was actually 100% efficient. From this, we can multiple the number of effective cores with each CPU's operating frequency to get what is essentially how many operations per second the CPU is able to complete (or GFLOPs):

$$GFLOPs = CPU_{Frequency} * EffectiveCores$$

Finally, we can estimate how long it would take the CPU you are interested in to complete the same action you benchmarked by dividing the GFLOPs of the two CPUs and multiplying it by the time it took your test CPU to complete the action with all of it's cores enabled:

$$Performance = \frac{CPU1GFLOPs}{CPU2GFLOPs} * CPU1BenchTime$$

With this, you should end up with an estimation of how long it would take a CPU to complete the action you benchmarked.

Easy Mode - Using a Google Doc spreadsheet

If you want to estimate the performance of a CPU using Amdahl's Law and don't love math, you will probably have a headache by the time you complete this guide. Lucky for you, we took the time to put together a Google Doc that has all the equations already done and ready: **Estimating CPU Performance**

(<https://docs.google.com/spreadsheets/d/1z1KWmp4dMgm7jl4EQ8w1nZAss4GZamZij706PszwM4g/edit?usp=sharing>) . You will need to make a copy of the Doc (go to File->Make a Copy), but once you have done that you will be able to use it as much as you like.

To use this doc, do the following:

- Complete Step 1: Test the program with various number of CPU cores (<https://www.pugetsystems.com/labs/articles/How-to-Estimate-CPU-Performance-619/#Step1TesttheprogramwithvariousnumberofCPUcores>) . Unfortunately, you simply have to do this step yourself.
- Once you have tested your application with various numbers of CPU cores active, input your results into the orange cells in the Google Doc (replacing the example results)
- Adjust the parallel efficiency fraction (the yellow cell) until the two lines on the graph are similar. If you cannot get the two lines to line up, it may be that your program is not CPU limited (see the Amdahl's Law Limitations (<https://www.pugetsystems.com/labs/articles/How-to-Estimate-CPU-Performance-619/#AmdahlsLawLimitations>) section)
- Change the light blue cells to reflect the cores and frequency of the CPU you used for testing (row 28) and the CPU(s) you are interested in estimating the performance of (row 29-30)
- You should see an estimation of how long it should take each CPU to perform the action you benchmarked in the green cells

This is much easier than trying to keep track of all the different equations, although we understand that there are some people who strangely love doing math.

Conclusion

Whether you followed the step-by-step instructions or simply used the Google Doc we linked, you should now have the resources and information needed to estimate the performance of a CPU for your exact program and application. While this is not the easiest process in the world, it can be invaluable when trying to decide what CPU to use in your new computer.

Say you are purchasing a new system but are torn between two CPU models that are similar in cost, but very different in terms of frequency and core count. As an example, lets use a Xeon E5-2667 V3 and a Xeon E5-2690 V3. Using the data from the example in Step 2 (<https://www.pugetsystems.com/labs/articles/Estimating-CPU-Performance-using-Amdahls-Law-619/#Step2Determiningtheparallelizationfraction>) and assuming that our test CPU was a Xeon E5-2660 V3 2.6GHz Ten Core we can estimate the performance of these two CPUs to be:

CPU Model	~MSRP	Estimated Action Time
Intel Xeon E5-2660 V3 2.6GHz Ten Core (Test CPU)	\$1450	85.3 seconds
Intel Xeon E5-2667 V3 3.2GHz Eight Core	\$2057	82.5 seconds
Intel Xeon E5-2690 V3 2.6GHz Twelve Core	\$2090	74.4 seconds

In this example, a E5-2667 V3 should take about 82.5 seconds to complete the action we benchmarked while a E5-2690 V3 should only take about 74.4 seconds. Since the two CPUs are only \$33 apart in price, this makes it almost a no-brainer that the E5-2690 V3 is the best choice in this instance.

Remember that this only applies to CPUs that are of a similar architecture to the one you used for testing and only for the action that you benchmarked. Anything different (even within the same program) may have drastically different results. However, if you keep finding yourself waiting on a render to finish, an export to complete, or any other single task you can limit your testing to just those tasks. Each may have a different parallelization efficiency, but if you determine the efficiency for each task and give them a certain weight (likely based on how often you are waiting on each to finish) you can make a much more educated decision on which CPU is right for you.

In fact, this is exactly what we use to determine what CPU we should offer in our growing list of Recommended Systems (<https://www.pugetsystems.com/featured.php>). While we have only completed testing for AutoCAD (<https://www.pugetsystems.com/recommended/Recommended-Systems-for-AutoDesk-AutoCAD-134>), Photoshop (<https://www.pugetsystems.com/recommended/Recommended-Systems-for-Adobe-Photoshop-139>), and Imaris (<https://www.pugetsystems.com/recommended/Recommended-Systems-for-Bitplane-s-Imaris-133>) so far, we expect to grow this list to include a large number of different software that many of our customers use.

If you followed this guide, we'd love to hear what you tested, what problems (if any) you ran into, and what parallelization fraction you found to be the closest match. Just let us know in the comments below!

Tags: Amdahl's Law, multithreading, multi core


Related Articles

- Ivy Bridge CPU TIM Paste Replacement (/labs/articles/Ivy-Bridge-CPU-TIM-Paste-Replacement-160/)
- How it works: Windows 7 Libraries (/labs/articles/How-it-works-Windows-7-Libraries-88/)
- Technology Primer - Sandy Bridge (/labs/articles/Technology-Primer---Sandy-Bridge-85/)

Tagged

How-to Guide (/all_articles.php?tag=How-to Guide) Processors (/all_articles.php?tag=Processors)

Subscribe

 (http://feeds2.feedburner.com/pugetsystems_all) [Subscribe](https://www.pugetsystems.com/subscribe.php)

Or receive articles by email:

[Subscribe](#)

8 Comments

Puget Systems

 [Login](#) ▾

 [Recommend](#)

 [Share](#)

[Sort by Oldest](#) ▾



Join the discussion...



LORD ODIN · 9 months ago

"When you are choosing a CPU, there are two main specifications you need to pay attention to that define the performance of a CPU:

The frequency is how many operations a single CPU core can complete in a second (how fast it