[B91] Bailey, D. H. (1991). Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers Computers.

This article describes how scientific papers in the area parallel computing often present their results in the best light or to make it look faster in comparison to other systems. The article isn't strictly scientific but rather humorous and shows twelve distinct arguments.

1. **Quote only 32bit performance results:** Because 64bit computing is slower, one should use only 32-bit computing or not mention the word width at all and compare your 32bit with another 64bit system.
2. **Present performance figures from the kernel, represent these figures as the performance of the entire application:** Don't give the performance of the whole system, but rather of the critical section that where the code takes the most time.
3. **Quietly employ assembly code and other low-level language constructs:** Trivial.
4. **Scale up the problem size with the number of processors, but omit any mention of this fact:** Do plot the performance for problems whose sizes scale up with the number of processors, just dont mention this fact in the paper.
5. **Quote performance projected to a full system:** Project your results that ran on a small cluster linearly to a big cluster. This will make your results look impressive.
6. **Compare your results against scalar, unoptimized code on Crays:** Trivial.
7. **When direct run time comparisons are required, compare with an old code on an obsolte system:** Use your optimized code with a optimized system and compare your results to an old system with outdated code. This will make your system look faster in comparison.
8. **MFLOPS: base the opcount on the parallel, not on the sequential implementation:** A good parallel system achieves a lot of MFLOPS. Put empty loops in there and obtain your results from a parallel system which often has much more MFLOPS in the sum because it contains redundant operations.
9. **Quote performance in terms of processor utilization, parallel speedups or MFLOPS per dollar:** Use other measures as processor utilization (do unneccecary calculation) or MFLOPS per dollar (new systems cost a lot of money $\rightarrow$ comparison is in favor of your system).
10. **Mutilate the algorithm used in the parallel implementation to match the architecture:** The changes you made to the algorithms to have it running on parallel computers should be adapted to the architecture.
11. **Measure parallel run times on a dedicated system, but measure conventional run times in a busy enviroment:** In order to have your system look very performant, you can compare the runtime on a parallel system without usage to a runtime on a busy sequential system. This will make your implementation look much faster.
12. **If all else fails, show pretty pictures and animated videos, and don't talk about performance:** If your results are not good to look at, show something that is.

[K12] Kohavi, R., Deng, A., Frasca, B., Longbotham, R., Walker, T., & Xu, Y. (2012, August). Trustworthy online controlled experiments: Five puzzling outcomes explained. In Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 786-794). ACM.

A controlled experiment is an experiment that uses controls, usually separating the subjects into one or more control groups and experimental groups. When analyzing results of controlled experiments, sometimes you have unexpected results. This paper uses five experiments to describe this phenomena.

In the first experiment, the Microsoft Bing team had a bug in an experiment which resulted in poor search results. Therefore, users had to click on more links (and thus ads), increasing the per user queries and revenue in the short-term. This however decreases the average customer lifetime value which is the most important measurement for a search engine.

The second experiment is an adition of code, slowing down the user-interface. The puzzling outcome was that the users didn't care about the slow user-interface and instead, clicked more. The explaination is that different browsers handle clicks and the triggered request differently, sometimes not cancelling image request sent to a server. This resulted in higher click rates from users.

Experiment number three is about the trend of an inintial effect. It is expected that at first, the users might have to get used to a new feature. Then, there is an break-even point and at some point, the users might like it more. This is a typical learning effect. However, due to statistical effects there will be no linear progession but rather a logarithmic one. Thus, the trend may never cross the $> 0\%$ line.

Experiment number four concluded that running an experiment longer does not provide additional statistical power. On first look this might seem wrong because the trivial assumption is that the more users an experiment has, the more valid it must be. However, for metrics like clickthrough, the confidence interval even shrinks on longer running experiments and for metrics like sessions per user the interval does not even change.

In the fifth experiment the authors analyzed carryover effects from a user going from one experiment bucket to another. He carried the experiences from prior experiments over to the subsequent ones. This is due to the nature of putting users in buckets at random. It concludes that bad experiences carry over much longer and migitation by re-randomization after each experiment or by localized re-randomization, localized meaning inside of an already randomized bucket.

Controlled experiments are the best way to gather meaningful data. However, really understanding every detail is something is far from trivial — especially when at first glance the results are assumed to be wrong.