D-Wave
The Quantum Computing Company™

Limits on Parallel Speedup for Classical Ising Model Solvers

WHITEPAPER

Summary

Why can't we put together a million cores and make it run a million times faster? Parallel computing systems offer enormous potential for significant runtime speedups over computation by a single CPU core. However, many computational tasks cannot be efficiently parallelized. We explore some practical limits to achieving parallel speedups, with reference to some classical optimization solvers that are competitors to D-Wave quantum computers.

The runtime speedup realized from switching from sequential to parallel computers is captured by the following formula, a refinement of Amdahl's Law [1]:[1]

$$T(P) = (1-\alpha)\cdot T_{seq} + \alpha \cdot T_{par}(1/P) + T_{over}(P).$$

The formula says that the time to run a process on P parallel cores depends on: (a) what proportion of the code $(1-\alpha)$ is inherently sequential; (b) what proportion (α) can be divided among the P cores; and (c) how much parallel overhead accrues. Overhead comes from such things as memory access, synchronization, and network communication, which we refer to generically as *remote access* to data that is not local to a given core. The function $T_{over}(P)$ may be constant or increasing in P.

Relative parallel speedup is defined as the time to run on one core divided by the time to run on P cores, that is, $T(1)/T(P)$. *Absolute parallel speedup* is $T_{CPU}/T(P)$, where T_{CPU} is the time for the fastest known single-CPU implementation, while $T(P)$ may run on a different platform such as a graphics processing unit (GPU).

Figure 1 shows how speedup depends on these time components. The diagonal black line represents the ideal—perfect relative speedup—when $\alpha = 1$ and $T_{over}(P) = 0$. Other curves show less-than-ideal speedup, when $\alpha < 1$ or $T_{over}(P) > 0$, or both. The non-ideal curves have constant upper limits: if $\alpha = .95$, then 5 percent (1/20th) of the code cannot be made faster, and relative speedup never rises above 20, regardless of the number of cores available. If overhead increases with P, parallel speedup can turn into parallel slowdown, as illustrated by the blue curve.

The dashed horizontal line marks time for the fast sequential code: here, T_{CPU} is 25 times faster than $T(1)$ on one GPU core. This gap creates a lower bound on the number of parallel cores needed to realize absolute speedup, as shown by the brown curve, which does not rise above the horizontal line until $P = 35$. Curves that stay below the dashed line correspond to cases where absolute parallel speedup is not possible.

In the real world, the viability of realizing parallel speedup often boils down to the balance of constant factors attached to component times, which often depends on the type of remote access that is needed. Overhead tends to grow as a step function, jumping whenever P crosses a boundary to a bigger system with higher communication overhead. This creates speedup curves that step down when P crosses boundaries, illustrated by the green curve in Figure 1.

Finally, parallel speedup can be limited by typical use. If a given code loop iterates M times, then parallelization yields speedups for $P \leq M$, but nothing further.

[1] The formula is used here as a conceptual model only: real code cannot always be partitioned this way into three neat categories.

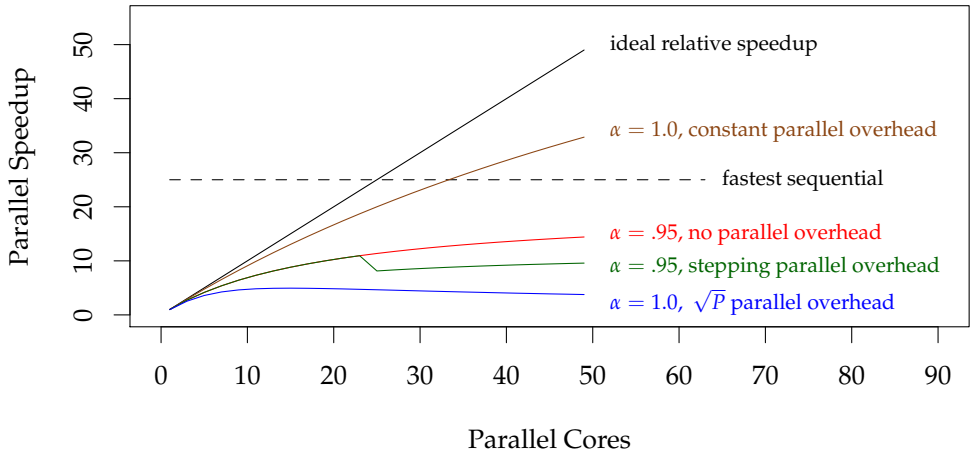

Figure 1: Relative parallel speedup (colored curves), can be less than the ideal shown by the black diagonal line. Absolute parallel speedup compares parallel times to the fastest sequential code (horizontal dotted line).

Parallelizing Ising model solvers. Several state-of-the-art solvers for Ising model problems have been tested in comparison to D-Wave quantum processing units (QPUs). Here are some observations about their parallelizability.

- The solvers of interest (including the QPU) all have a *sampling* loop that is easy to parallelize. With typical solution times on the scale of milliseconds, parallel sampling can yield speedups up to about two orders of magnitude before communication overhead starts to dominate.
- We have implemented [2] some solvers using state-of-the-art GPU platforms with 2048 cores. Taking one (called simulated annealing) as an example, we see relative parallel speedup by factors near 256 (less than the ideal of 2048), and absolute parallel speedup (including parallel sampling) around 1000 (compared to a CPU version with no parallel sampling). Increasing the number of GPU cores does not yield additional speedups, because of the stepping effect described earlier.
- A strong competitor known as the Hamze–de Frietas–Selby (HFS) method cannot run efficiently on a GPU due to high remote access overhead.

More importantly, parallellization does not improve the *quality of solutions* returned by these solvers. That task falls to a specific code loop in each one, which cannot be efficiently parallelized, and which creates large computational bottlenecks. The number of iterations needed for this loop to produce solutions of suitable quality tends to grow exponentially with input size.

Going forward, practical constraints will continue to limit gains from parallel speedup on classical computing platforms. Significant computational bottlenecks will continue to grow. Performance breakthroughs in this area will be driven by access to fundamentally new types of algorithmic resources, such as those provided by quantum computers.

References

[1] M. Horoi and R. J. Enbody, "Using Amdahl's Law as a metric to drive code parallelization: Two case studies," *The International Journal of High Performance Computing Applications*, vol. 15, no. 1, pp. 34–41, 2001.

[2] J. King, S. Yarkoni, J. Raymond, I. Ozfidan, A. D. King, M. M. Nevisi, J. P. Hilton, and C. C. McGeoch, "Quantum annealing amid local ruggedness and global frustration," *D-Wave Technical Report Series*, no. 14-1003A-B, 2016.