VIEWPOINTS

September 2022

Point/Counterpoint On the Model of Computation

William Dally and Uzi Vishkin

William Dally

DOI:10.1145/3548783

Point: We Must Extend Our Model of Computation to Account for Cost and Location

IMAGE BY ANDRIJ BORYS ASSOCIATES, USING SHUTTERSTOCK

FOR DECADES WE have used the RAM (random-access memory)[2] and PRAM (parallel RAM) models[5] along with asymptotic analysis to measure the complexity of algorithms. The RAM and PRAM models treat all operations, from an integer add to a global memory load as unit cost. This approximation was appropriate during the early days of computing when the costs of arithmetic and communication were somewhat comparable. Over time, however advancing semiconductor technology has caused the cost of arithmetic and logic to shrink by orders of magnitude while the cost of communication has reduced at a much slower rate. As a result, today fetching two 32-bit words from main memory expends 1.3nJ of energy while performing a 32-bit add operation (which requires two 32-bit words as input) takes only 20fJ of energy 64,000x less. A model of computation like RAM or PRAM that treats these two operations as equivalent does a poor job of estimating the cost of a computation, and hence does a poor job of comparing alternative algorithms.

Communication Is the Dominant Cost in Computing. Whether we consider cost to be energy or time, communication dominates modern computation. A 32-bit add operation takes only 20fJ and 150ps. Moving the two 32-bit words to feed this operation 1mm takes 1.9pJ and 400ps. Moving the 64 bits 40mm from corner to corner on a 400mm^2 chip takes 77pJ and 16ns. Going off chip takes 320pJ with a delay of 6ns per meter.

While memory accesses are costly operations, almost all of the cost of accessing memory is communication cost. The time and energy needed to read or write a single bit cell is negligible. The vast majority of the time and energy of a memory access is due to communication: moving the address to the bit cell and moving the data from the bit cell to the consumer. Because communication dominates, accessing a small, local memory is far less expensive than a global memory access. On-chip SRAM memories, for example, are built from small 8KB (64Kb) subarrays. Accessing 64b from a sub-array costs 0.64pJ and takes 300ps. Accessing a 256MB memory (approximately 100mm2) constructed from these subarrays costs 58pJ and 12.3ns—of which 57.4pJ and 12ns are due to communi-Arithcation —15mm each way, 30mm total. Because communication completely dominates the cost of

> With the end of Moore's Law, domain-specific architectures are emerging as a leading candidate to continue scaling computing performance.

Consider the problem of summing a table that fits in the on-chip memory of a 4×4 array of 256-core processor chips that are each 16mm on a side. Each core is located at the center of a 2MB SRAM array and can access the local array with 1mm (round trip) of communication cost (1.9pJ and 400ps for a 64b access). A random access incurs 21.3mm of on-chip communication cost for the 1/16 of accesses to the local chip and an additional 640pJ +21.3mm of cost for the 15/16 of accesses that go off chip for an average random access energy of 680pJ. With a model of location, we can have each core sum the 256K words in its local memory and then forward the result up a tree to compute the final sum. We can perform the same computation by placing the threads and data randomly. The cost of the randomly placed computation is 354x higher than the local computation. The PRAM model considers the local and random computations to be of equal cost despite the orders of magnitude difference in cost.

Store or Recompute. The large difference in energy between arithmetic and communication often drives the decision whether to store or recompute an intermediate value. Training a multilayer perceptron (MLP) requires the activation values for each layer during the back-propagation step. If there is insufficient on-chip memory, the activations can be stored to off-chip memory at a cost of 640pJ per 16b activation (write + read), an O(*n*) operation—where *n* is the number of activations in this layer. Alternatively the activations can be recomputed on the fly by performing a matrix-vector multiplication, an O(n^2) computation, at a cost of 160fJ/MAC. The PRAM model would suggest storing the intermediate result—preferring the O(*n*) store over the O(n^2) M × V. However, for vectors smaller than *n* = 4,000, recomputation is less expensive. For a typical MLP with *n* = 256, recomputation is 16x less expensive.

Constant Factors Matter. Under the PRAM model, the asymptotic complexity of the summation example here is O(*n*) for both the local and the random cases. The 230×

Constant factors do matter, however, particularly when the asymptotic complexity is the same. The local computation is 230× cheaper. Just as you would not settle for paying 230× too much for your groceries, you should not pay 230× too much for a computation because you are using an incomplete model of computation. Large constant factors can even overcome a difference in asymptotic complexity (this is why Strassen's O($n^{2.8}$) matrix multiply algorithm [6] is rarely used). This is the case in our store O(n) vs. recompute O(n^2) example here, where for typical values the higher asymptotic complexity is less expensive.

Caches Are Not Enough. Modern computers use cache memories to improve locality. Caches work great in cases where there is temporal locality (that is, reuse). However, for computations such as the summation example here they do not help. Every word is visited exactly once, there is no reuse. Many HPC codes and neural-network models with a batch size of one have the same issue.

Domain-Specific Architectures Highlight the Issue. With the end of Moore's Law, domain-specific architectures (DSAs) are emerging as a leading candidate to continue scaling computing performance. [3,4] By removing most of the overhead associ-ated with general-purpose CPUs, DSAs make the large gap between arithmetic and communication more apparent. The overhead of a CPU turns the 20fJ of a 32b add operation into an 80pJ add instruction. In comparison the 1.2nJ memory access is only 15x more expensive rather than 64,000×. With a DSA this overhead is eliminated revealing the full size of the gap between arithmetic and communication.

Computational Models are for Performance Programming. In some situations, such as user-interface code, programmers concentrate on functionality without concern for cost. In these situations a naïve code is *fast enough* and there is no need for a model of computation (RAM, PRAM, or other) to estimate cost and compare alternatives. In other cases, however, such as training neural network models, performing large scientific simulations, and running optimizations, cost is of paramount importance. Huge amounts of energy and expensive computation time are wasted by an inefficient algorithm. It is in these cases of *performance programming* where we need a model of computation to estimate cost and compare alternatives, and that model of computation needs to reflect the communicationdominated nature of today's computing hardware.

The Planet Demands We Be More Efficient. Datacenter computers alone used 1% of

2030.[1] With such a large fraction of the world's energy going to computing, it is incumbent on us to design efficient computations to minimize the carbon footprint of computation. The first step in being more efficient is having an accurate model to analyze algorithms, so we can pick the most efficient algorithm for a given problem. We can no longer afford to ignore the constant factor difference between arithmetic and communication or the increase in energy with distance.

PECM: A Simple, Accurate Model of Computation. Two simple changes to the PRAM model can fix its two main problems. To account for the large difference in cost between arithmetic operations (like add) and memory access, we assign them different costs. Arithmetic operations remain unit cost, but memory operations have a higher cost, proportional to distance. Each processing element and each memory is assigned a location *l* ∈ *L* and a distance function D: *L* × L ⇒ R is defined to determine the cost of sending a message or making a memory access between two locations.

We can use different distance functions to model different computational targets. Two-dimensional Manhattan distance models on-chip communication. Locations are *x,y* coordinates and distance is the Manhattan distance between two coordinates. Off-chip communication can be modeled by adding additional distance when either coordinate spans a chip boundary (as in the example here).

By allowing us to reason about the true costs of computations, this parallel explicit communication model (PECM) will allow us to design more efficient computations and in doing so reduce the carbon footprint of computing.

References

1. Anders, A. and Edler, T. On global electricity usage of communication technology trends to 2030. *Challenges 6*, 1 (2015), 117–157.

2. Cook, S.A. and Reckhow, R.A. Time-bounded random access machines. *Journal of Computer Systems Science* 7, 4 (Apr. 1973), 354–375.

3. Dally, W.J., Yatish, T., and Han, S. Domain-specific hardware accelerators. *Commun. ACM 63*, 7 (July 2020), 48–57.

4. Hennessy, J.L. and Patterson, D.A. A new golden age for computer architecture. *Commun. ACM 62*, 2 (Feb. 2019), 48–60.

5. Karpand, R.M. et al. A survey of parallel algorithms for shared- memory machines. In *Handbook of Theoretical Computer Science*. North-Holland, 1988.

6. Strassen, V. Gaussian elimination is not optimal. *Numer. Math. 13*, 4 (Aug. 1969), 354–356.

William Dally (dally@stanford.edu) is an adjunct professor of computer science at Stanford University and chief scientist and senior vice president of research at NVIDIA, Incline Village, NV, USA.