What GPU Computing Means for High-End Systems

RICHARD VUDUC AND KENT CZECHOWSKI
Georgia Institute of Technology

Between 2018 and 2020, the first exascale supercomputers will come online.[1] If realized, these systems will perform a staggering 10^{18} or more floating-point operations per second (1 exaflop/second, or 1 EF/s), which is two to three orders of magnitude more than what's available today. Such systems promise to advance computer-based simulation of diverse phenomena in climate, energy, materials, combustion, geoscience, and biomedicine, among numerous other areas.[2]

What will the architecture of such a machine look like? GPU computing dominates the conversation.[3] The reason is simple: GPUs today deliver more peak performance and bandwidth relative to conventional CPU designs within a comparable power footprint. As such, a GPU-based supercomputer should be smaller and more energy-efficient than its non-GPU counterpart, and therefore cheaper to buy with a lower energy bill over its lifetime. Three of today's five fastest machines use GPUs (http://top500.org), as do half of the top 10 green supercomputers as measured by performance per Watt (http://green500.org). Most high-performance computing (HPC) analysts would bet that GPU-like designs are perhaps the most viable path toward exascale computing.

But is this a sure bet? Here, we suggest that exactly the opposite might be true: relative to an ideal GPU-like system at exascale, a system with slower processors—but, critically, better-balanced ones—might yield an overall system with higher performance while consuming less energy. The analysis behind this claim is not actually about the specifics of CPUs versus GPUs.[4-7] Rather, it draws on a fundamental principle of algorithmic performance engineering—namely, the principle of system balance.[8-10]

Balance and intensity

The classical principle of balance says simply that we should strive to design systems in which compute time equals I/O time.[9] Compute time might be the time to perform flops, and I/O time can refer to memory or network traffic time. When balanced, we can hide I/O time, making the computation compute bound rather than I/O bound.

Balance gives us an analytical framework for relating architectures and algorithms. Consider the hypothetical multiprocessor with p cores shown in Figure 1a. It has Z words of local storage (that is, registers, cache, and scratchpad memory). Each core can perform up to C_0 operations per unit time, and the peak off-processor bandwidth is β words per unit time. Given a computation, we will apply the balance principle to derive precise analytical relationships among these parameters.

To do so, we also must characterize a computation by its intensity, which is the number of operations (for example, flops) it performs divided by the number of words it must transfer between the fast and external memories. Intensity has units of operations per word and is an intrinsic property of the computation. For example, dense-matrix multiply usually has a high intensity because it has a lot of data reuse per operation; sorting, by contrast, has streaming behavior and a relatively lower intensity. The precise value of intensity depends on the size of the fast memory Z. More fast memory usually means fewer required transfers, so intensity will tend to grow with Z. Thus, we denote intensity by $I(Z)$.

If we apply the balance principle to a given computation and this processor system, we will obtain the constraint[8-10]

$$\frac{p \cdot C_0}{\beta} = I(Z) \tag{1}$$

where $p \cdot C_0/\beta$ is the processor's balance and $I(Z)$ is the intensity. When Equation 1 is true, the processor delivers operations and data at the rate required by the computation. Furthermore, it gives an explicit relationship among the architectural parameters.

A nice way to visualize the impact of balance and intensity on performance is through a roofline diagram.[11] Figure 1b is an example. Each line represents a typical processor from a particular platform class (GPU, CPU, or mobile); the y-axis shows the maximum achievable

0272-1732/11/$26.00 © 2011 IEEE

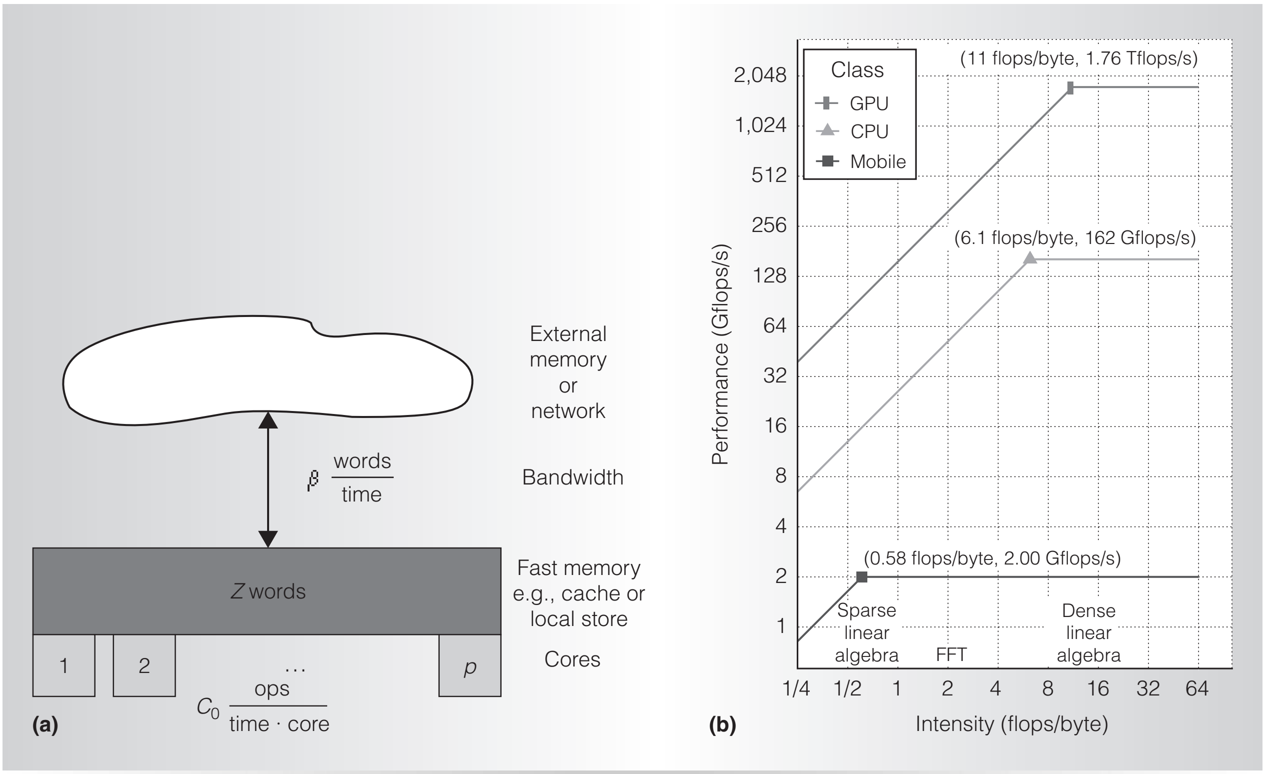

Figure 1. Illustration of balance principles: a hypothetical multi-/many-core processor (a). A roofline diagram comparing ideal performance for typical processors in various platform classes (b). Performance is in single-precision Gflops/s.

performance given some intensity along the *x*-axis. The ridge point is the balance point of Equation 1. All intensities to the right of the ridge point are compute bound, whereas all to the left are I/O bound.

Figure 1b shows that GPUs have much higher peak performance than CPUs, but they also demand more intensity from the application to reach peak. The speedup of GPU over CPU depends on the intensity; a well-tuned application with high intensity might be more than 10× faster on a GPU, but 5× faster if the intensity is small. This difference in speedups is due to the general inability to increase bandwidth at the same rate as computational peak.

Still, if an application only requires a system with one processor, then the roofline shows that a GPU-like design is likely faster regardless of whether it's compute bound or I/O bound. But as we will see next, the story can change when we start putting processors together.

Why balance matters

GPUs are a natural building block for an exascale system, given their high compute density (peak and bandwidth) and energy efficiency. In particular, a GPU-based system will need fewer processors to reach a desired level of system peak compared to a CPU-based system. For a compute-bound application, GPUs seem like a clear price-performance win.

And what if the computation is I/O bound? GPUs still deliver more absolute memory bandwidth, and so, again, using them seems like a win. But in a distributed memory system, there will be two sources of communication cost: memory communication between the processor and local main memory, and network communication between processors. This profoundly affects how we might design the overall system.

To see this effect, we analyzed potential designs for distributed 3D fast Fourier transforms at exascale. A 3D FFT is an interesting case for two reasons, beyond its broad uses in computational science and engineering. First, it's algorithmically similar to other widely used data-intensive computations, like sorting. Secondly, it's I/O bound rather than compute bound. Conclusions about FFTs should apply to a broader class of I/O-bound problems.

The full analysis isn't difficult but it is tedious,[12] so we just sketch it here. An $n \times n \times n$ FFT operates on n^3 data items, which we will assume can be evenly distributed over P processors. (P is the number of processors, where each processor is multicore or many-core with p cores.) Since the FFT is heavily I/O bound, the time to perform flops is negligible and can be ignored for even

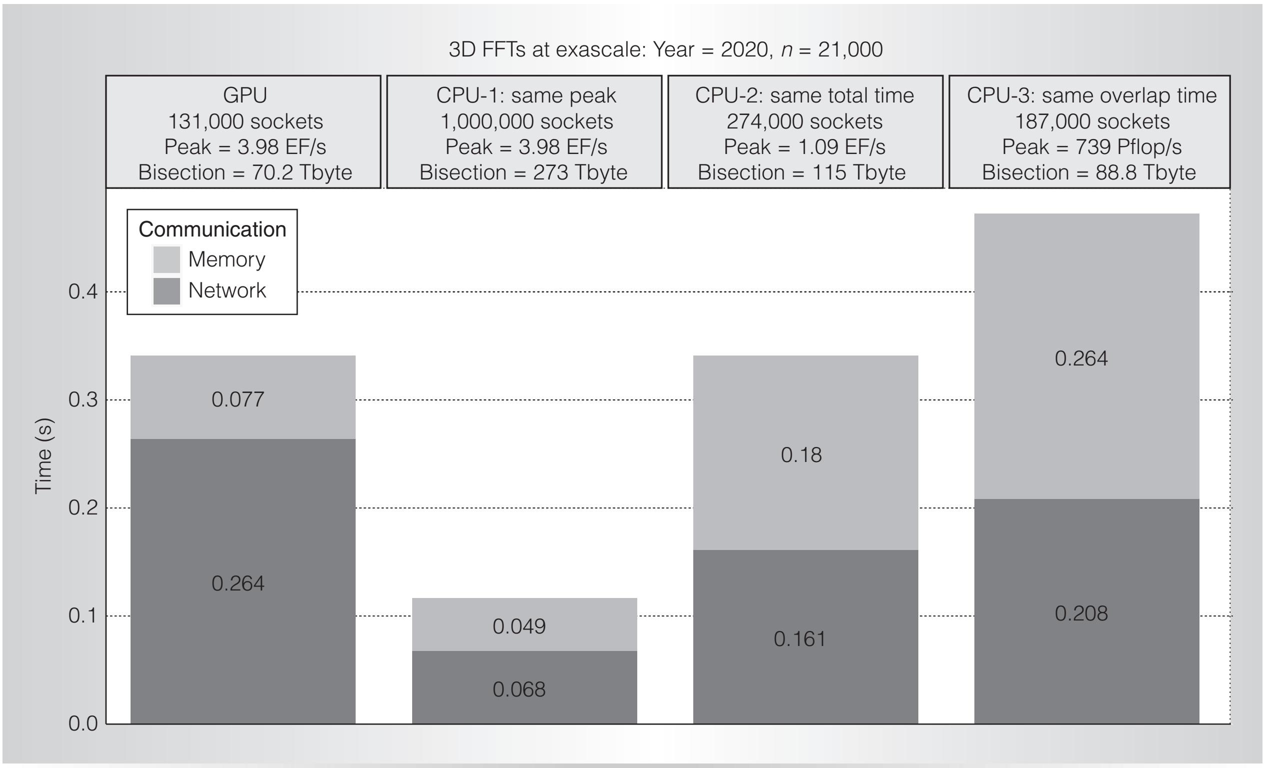

Figure 2. Projected execution time for a distributed 3D ($n \times n \times n$) fast Fourier transform. Time is broken down into network versus memory communication for a hypothetical GPU-like exascale system (left) compared to three hypothetical CPU-like systems. For all three CPU-like systems, network communication time is lower than the GPU-like system, at the cost of more physical processors.

moderately sized problems. At exascale, we estimate that a large 3D FFT will spend 1,000× more time on communication than on flops.

The first communication cost is memory transfer between the processor's fast memory and external memory. Assuming a 3D stacked future,[13] we can regard this cost as ''on-chip.'' The minimum time for this on-chip data movement is, assuming the architecture of Figure 1a,

$$T_{\text{mem}} \approx O\left(\frac{n^3}{P} \cdot \frac{\log_Z n}{\beta_{\text{mem}}}\right)$$

where β_{mem} is the memory bandwidth. As it happens, this factor is a provable lower bound, meaning no algorithm can move less data.

The second communication cost is network transfer between processors. This off-chip cost will depend on the network topology and other factors. For the sake of discussion, let's assume a 3D torus. The time spent on network transfers is at least

$$T_{\text{net}} \approx O\left(\frac{n^3}{P^{2/3} \cdot \beta_{\text{link}}}\right)$$

where β_{link} is the network link bandwidth. The denominator reflects the network's bisection bandwidth and will vary with the topology.

Here is the connection to balance. If machine peak is fixed, then $P \propto 1/(p \cdot C_0)$ and T_{mem} becomes

$$T_{\text{mem}} \approx O\left(\frac{p \cdot C_0}{\beta_{\text{mem}}} \cdot n^3 \log_Z n\right)$$

The factor of $p \cdot C_0/\beta_{\text{mem}}$ is our processor's balance factor from Equation 1. As such, increasing processor imbalance increases this factor, which then increases T_{mem}. Network communication time also depends inversely on P, so it too will increase. However, it does so more slowly, in proportion to $(p \cdot C_0)^{2/3}/\beta_{\text{link}}$ rather than $p \cdot C_0/\beta_{\text{mem}}$. In other words, network time is more robust than memory time to increases in peak.

An exascale projection

The lesson is that increasing imbalance by increasing the processor peak $p \cdot C_0$ too quickly relative to memory bandwidth β_{mem} can increase both memory and network communication time. However, this analysis has undisclosed constants. What happens if we refine these constants and then plug in projected values for the machine parameters in, say, 10 years?

Figure 2 shows the results of just such an analysis on exascale systems built from hypothetical processors in

Table 1. Processor architecture projections, from starting values on the US National Science Foundation's Keeneland GPU-based initial delivery system in 2010. Cores refers to the processor manufacturer's own usage rather than, say, floating-point functional units. Peak shown is double-precision, unlike Figure 1b, which uses single-precision peak. (See http://keeneland.gatech.edu.)

Parameter		2010 values (Keeneland)	Doubling time (in years)	10-year increase factor	Value
Cores	p_{cpu}	6	1.87	40.7×	134 cores
	p_{gpu}	448	—	—	18,000 cores
Peak	$p_{cpu} \cdot c_{cpu}$	67 Gflops/s	1.70	59.0×	4 Tflops/s
	$p_{gpu} \cdot c_{gpu}$	515 Gflops/s	—	—	30 Tflops/s
Memory bandwidth	β_{cpu}	25.6 Gbytes/s	3.00	9.7	248 Gbytes/s
	β_{gpu}	144 Gbytes/s	—	—	1.4 Tbytes/s
Fast memory	Z_{cpu}	12 Mbytes	2.00	32.0	384 Mbytes
	Z_{gpu}	2.7 Mbytes	—	—	86.4 Mbytes
Network bandwidth	β_{link}	10 Gbytes/s	2.25	21.8	218 Gbytes/s

the year 2020.[12] We consider a GPU-based exascale system, compared to three CPU-based systems. The processor and network link projections are shown in Table 1, which extrapolates current trends from various sources.[12] The extrapolations assume that current CPU and GPU processor designs will scale architectural parameters with the same doubling rates—a strong assumption. We use these projected values to construct four systems at exascale.

First, based on the Top 500's historical peak system-performance data, we assume that in 2020, we will be able to build a 4 EF/s (double-precision) system out of GPU-like processors. We assume each processor has a direct network connection, optimistically avoiding things like PCI Express channels, which should be possible with stacking or other direct integration techniques. On the basis of Table 1's projections, this system will need to have about $P_{gpu} = 131{,}000$ processors. The values of T_{mem} and T_{net} for this system appear in the leftmost bar in Figure 2. Network time dominates memory time by 3.4×.

The second bar in Figure 2 shows T_{mem} and T_{net} for CPU-1, a 4 EF/s system built instead from projected CPUs. This system spends less time than the GPU system in both forms of communication, with its network communication time nearly one-quarter the time. The reason is better processor balance, which translates into lower memory and network time. However, this advantage isn't free: it requires over 7.6× as many processors. The price of such a system could be prohibitive, if dollar cost is proportional to the number of processors.

Is there an alternative? One idea is to give up on matching peak performance while still delivering the same actual performance. Doing so is possible for an FFT because communication time dominates the total time, and the flops are almost free. In the third bar in Figure 2, we consider CPU-2, a CPU-based design in which the total communication time for the FFT, $T_{mem} + T_{net}$, exactly matches that of the 4 EF/s GPU system. This change requires fewer processors than CPU-1 and has a lower overall peak of 1 EF/s, making it perhaps cheaper than CPU-1.

More interestingly, observe that relative to the GPU system, CPU-2 trades higher on-chip memory communication time for lower off-chip network communication. Thus, if the dominant energy cost is off-chip communication compared to on-chip communication,[1] this design could save energy.

The CPU-2 system might be pessimistic in its execution time, because it assumes that we can't overlap T_{mem} and T_{net}. The rightmost bar in Figure 2 considers CPU-3, where we assume that the total time to perform the FFT is not the sum $T_{mem} + T_{net}$ but rather $\max\{T_{mem}, T_{net}\} = 0.264$ seconds when the two communication phases overlap. The CPU-3 system loses in total time but still wins in reducing off-chip network communication time. Again, this situation is an overall win if off-chip communication consumes much more energy than on-chip communication. Also, this system uses 187,000 CPUs, putting it within 1.4× of the total number of processors in the GPU system.

Looking forward

Assuming there is enough parallelism, GPUs deliver when a single processor is needed, regardless of application intensity, as Figure 1b suggests. But for distributed and I/O-bound applications, our analysis serves as a cautionary tale about the consequences of increasing imbalance too aggressively. Since current and future workloads might have a complex mix of compute- and I/O-bound components, perhaps the final chapter is yet to be written.

Our intent is not to pit CPU against GPU at exascale, even though we have framed it this way to be topical and provocative. In fact, such comparisons for exascale are in our view a red herring. Rather, the analysis abstracts away these details and says, simply, that balanced processors might yield better overall systems. The current trend in aggressive GPU and CPU designs is, as Figure 1b and Table 1 suggest, toward greater imbalance.

Let us close on a suggestive note. Bell's Law reminds us about the economics of computing:[14] the high-volume, low-cost, general-purpose processors used in ''low-end'' computing platforms eventually displace the low-volume, high-cost processors used in higher-end systems. For example, in the 1990s, commodity personal computers gave rise to clusters, which then fundamentally altered high-end supercomputer design.[15] Interestingly, Figure 1b might be a performance corollary to Bell's Law: low-end mobile processors are actually more balanced than the others, which our analysis says should be better for supercomputer performance. This observation suggests the form that tomorrow's supercomputers will take, a notion that may be part of an emerging view.[16,17]

References

1. P. Kogge et al., *Exascale Computing Study: Technology Challenges in Achieving Exascale Systems,* tech. report, DARPA, 2008.
2. H. Simon et al., *Modeling and Simulation at the Exascale for Energy and the Environment,* tech. report, US Department of Energy, Office of Science, 2007.
3. J. Nickolls and W.J. Dally, ''The GPU Computing Era,'' *IEEE Micro,* vol. 30, no. 2, 2010, pp. 56-69.
4. R. Bordawekar, U. Bondhugula, and R. Rao, ''Believe It or Not! Multicore CPUs Can Match GPU Performance for a FLOP-Intensive Application!'' *Proc. 19th Int'l Conf. Parallel Architectures and Compilation Techniques,* ACM Press, 2010, p. 537-538.
5. E.S. Chung et al., ''Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPUs?'' *Proc. 43rd IEEE/ACM Int'l Symp. Microarchitecture,* IEEE CS Press, 2010, pp. 225-236.
6. V.W. Lee et al., ''Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU,'' *ACM SIGARCH Computer Architecture News,* vol. 38, no. 3, 2010, pp. 451-460.
7. R. Vuduc et al., ''On the Limits of GPU Acceleration,'' *Proc. USENIX Workshop Hot Topics in Parallelism,* USENIX Assoc., 2010, http://www.usenix.org/events/hotpar10/tech.
8. K. Czechowski et al., ''Balance Principles for Algorithm-Architecture Codesign,'' *Proc. USENIX Workshop Hot Topics in Parallelism,* Usenix Assoc, 2011, http://www.usenix.org/events/hotpar11.
9. H.T. Kung, ''Memory Requirements for Balanced Computer Architectures,'' *Proc. ACM Int'l Symp. Computer Architecture,* ACM Press, 1986, pp. 49-54.
10. J. McCalpin, ''Memory Bandwidth and Machine Balance in High Performance Computers,'' *IEEE Technical Committee Computer Architecture,* newsletter, Dec. 1995.
11. S. Williams, A. Waterman, and D. Patterson, ''Roofline: An Insightful Visual Performance Model for Multicore Architectures,'' *Comm. ACM,* vol. 52, no. 4, 2009, pp. 65-76.
12. K. Czechowski et al., *Prospects for Scalable 3D FFTs on Heterogeneous Exascale Systems,* tech. report GT-CSE-11-02, Georgia Institute of Technology, 2011.
13. G.H. Loh and Y. Xie, ''3D Stacked Microprocessor: Are We There Yet?'' *IEEE Micro,* vol. 30, no. 3, 2010, pp. 60-64.
14. G. Bell, ''Bell's Law for the Birth and Death of Computer Classes,'' *Comm. ACM,* vol. 51, no. 1, 2008, pp. 86-94.
15. T. Anderson, D. Culler, and D. Patterson, ''A Case for NOW (Networks of Workstations),'' *IEEE Micro,* vol. 15, no. 1, 1995, pp. 54-64.
16. J. Markoff, ''The iPad in Your Hand: As Fast as a Supercomputer of Yore,'' blog, *New York Times,* 9 May 2011; http://bits.blogs.nytimes.com/2011/05/09/the-ipad-in-your-hand-as-fast-as-a-supercomputer-of-yore.
17. V.J. Reddi et al., ''Web Search Using Mobile Cores: Quantifying and Mitigating the Price of Efficiency,'' *ACM SIGARCH Computer Architecture News,* vol. 38, no. 3, 2010, pp. 215-314.

Richard Vuduc is an assistant professor in the School of Computational Science and Engineering at the Georgia Institute of Technology. His research lab, The HPC Garage (http://hpcgarage.org), studies high-performance computing, with an emphasis on parallel algorithms and performance analysis and tuning. Vuduc has a PhD in computer science from the University of California, Berkeley. He is a member of IEEE, the ACM, and SIAM.

Kent Czechowski is a PhD student in the School of Computational Science and Engineering at the Georgia Institute of Technology. His research interests include algorithm-architecture codesign, performance modeling for GPU/many-core architectures, and parallel and distributed algorithms. Czechowski has an MS in computer science from the Georgia Institute of Technology.