1.1 Computing's Energy Problem (and what we can do about it)

Mark Horowitz

Departments of Electrical Engineering and Computer Science,
Stanford University, Stanford, CA

1. Introduction

Technology scaling has decreased the cost of computing to the point where it can be included in almost anything. As a result, we now live in a world surrounded by computing devices. They power our searches on Google, connect to our friends on Facebook, answer our questions to Siri, and serve us our entertainment on Youtube; they are in our homes everywhere, in all our appliances (I recently had to reboot my refrigerator), cars, workplaces, and even in the cards we send to each other. We have become so accustomed to computing becoming faster, cheaper, and lower power, we simply assume it will continue. Already, smartphone capabilities are being embedded in eye glasses [1] and smart watches [2].

While scaling computing performance has never been easy, a number of factors have made scaling increasingly difficult this past decade, and have caused power to become the principal constraint on performance. Section 2 quickly reviews how computing became power limited, even before Dennard constant-field scaling [3] broke down, and explains the difficulties of using a technology change to fix our problems. The rest of the paper explores different approaches to addressing this computing-energy-consumption challenge, and shows that it will take more than parallelism to get the results we need. The new key to scaling computing performance is to create applications and hardware which are better matched to the task and each other. Accomplishing this will require tools that allow application experts to create these new efficient systems. While creating these tools is challenging, they will enable a renaissance in application-optimized computing!

2. Processor Scaling

Performance data for commercial microprocessors is now accessible on the web [4], and Figure 1.1.1 uses this data to show how gate speed and processor performance have scaled over time. The gate speed is an approximation of the FO4 (the delay of an inverter in that technology driving a load that is 4× its input capacitance) and the processor performance is normalized to that of a 386, using SPEC scores as a criterion. More details about how these numbers are derived are given in [5]. Note that the gate speed has improved by 100× since CMOS processors were introduced in the mid 80s, while application-level uniprocessor performance has increased by over 3000. During this same period, the number of transistors has followed a nice Moore's law exponential growth, and the rate of feature size scaling has also been remarkably consistent (see Figure 1.1.2). But, what has not scaled according to plan, is the power density of the processors (see Figure 1.1.3). The underlying cause of the observed exponential power growth can be traced to two factors: the fact that we did not scale power supply voltages at the constant field rate, which Dennard himself reported [6], and the fact that in our quest for performance, we scaled clock frequencies faster than dictated by constant-field scaling (see Figure 1.1.4). In the 1990s, voltage scaled down, but slower than technology (at about the square root of feature-size scaling), and frequency scaled up, but faster (at about the *square* of feature-size scaling). Because power is CV^2F, and C scales with technology, power should have become an issue much sooner than it did. However, during this same time frame, designers added many effective power saving techniques which extended this period of rapid performance gains.

But, this rapid scaling of clock frequency stopped in the early 2000s, as a result of two factors: First, processors hit the power wall for air cooling (using a low-cost heatsink, and air flow at a noise level acceptable for an office) which is around 100W. Worse still was the fact that voltage scaling also slowed down, since it was no longer possible to scale the threshold voltage due to rising leakage currents. To keep power in check, processor frequencies were reduced, and multiple processors were added to each die. However, since processor "performance" was changed to measure throughput, it continued to scale, as more cores were added each generation.

Like most chips today, processors used to run at a fixed supply voltage, and this voltage depended on the fabrication technology that was used. But, as processors became power constrained and leakage current grew, it became apparent that one could dramatically reduce the power dissipation, and improve the performance yield of a processor if each processor chip could specify the supply voltage that was required for it to operate at the desired performance. This would allow a chip fabricated with high-leakage, lower-average-V_{th} transistors, to run at a lower supply voltage, reducing both the dynamic and leakage power, for overall power optimization. Correspondingly, processors with higher V_{th} transistors, and lower leakage could run at a higher supply voltage while still operating within the total power budget, enabling these transistors to operate at the desired speed. While this has been good for processor specification, it has made it much more difficult to track how the average supply voltages have been scaling over the past decade. Thus, the numbers in the voltage plot in Figure 1.1.4 are the peak allowable supply voltages, and do not represent the average voltages used. From limited data, the actual operating supply voltages seem to remain in the 0.9 to 1.1 volt range for peak performance. But, the recent move to 3-D channel structures with reduced leakage currents, has enabled about a 100 to 200mV decrease in operating voltage.

3. Technology to the Rescue?

Given the current limitations of CMOS scaling, it is natural to look toward other technologies to enable computing performance to continue to scale. After all, computing started with mechanical devices, moved to electro-mechanical (relays), then to electronic tubes, transistors, bipolar ICs, nMOS, and finally to the CMOS technology we are using today. In fact, our CMOS technology has changed dramatically over the past 30 years: moving from single level Al connection to over 10 levels of Cu, away from SiO_2 gate oxide and back to metal gates, adding Ge for strain, and now to 3-D topologies. While CMOS will continue to evolve, and everyone should hope that the research in off-roadmap technologies is successful, there is a real possibility that, computing at least, will stay with CMOS-like technologies. The problem is not simply the potential abandonment of the manufacturing capability that the enormous investment has created: CMOS VLSI also has shaped our design abstractions that have allowed us to build functional artifacts of enormous complexity.

To move off the CMOS roadmap (for example, to quantum computing), would require a very different hardware platform. (Note, that we have already accepted that technology advances such as finFET are an extension of CMOS rather than a separate manufacturing process,) Such a new platform would disrupt the entire design abstraction hierarchy. Thus, the insertion cost for such a disruptive technology would include the creation of both a state-of-the-art manufacturing facility, and the tools and training in all the design abstraction layers that require change, which is not going to "come cheap". And, that is the problem. The fundamental challenge for any new startup (new idea) is insertion – how to minimize the investment needed to compete with the status quo. A radical idea must demonstrate its utility with only a modest investment since there is significant chance it will fail. The larger the investment, the lower the risk the investors are willing to tolerate, which is why large industries change incrementally. Thus, changing technology to fix the power problem is a perfect catch-22.

In some ways, the key problem is the capabilities of modern CMOS technologies: With CMOS, we can create chips with millions of transistors at nearly no cost, have all the transistors work, and run at GHz frequencies. While I am sure there are better technologies out there, I am not sure how we can afford the investment to find and develop them to the point where they are competitive. While there are many interesting new technologies, for any to develop will require a niche market different from computing, where they can be successful and earn the resources they need to grow. It is important to remember that most successful radical ideas create **new** markets. Only when these markets become large, do they challenge the players of the status quo, often indirectly, by changing the underlying rules of that market. So while it is possible that a new technology will eventually compete with CMOS for computing, it is not likely it will be from a frontal assault, and it will take time for this new technology to grow large enough to compete. Given this dynamic, CMOS-like technology will continue to dominate computing for the foreseeable future, and we will need to figure out ways to make our computing systems more energy efficient by means other than technology scaling.

978-1-4799-0920-9/14/$31.00 ©2014 IEEE

4. The Limits of Parallelism

When uniprocessors ran into the power wall, to continue to scale performance, manufacturers began to include more processor cores on each die. It had been known for decades [7] that if the application was parallel, a parallel machine would be more power-efficient. The reason can be easily seen from the data in Figure 1.1.5: Here, we have taken data on early processors, and plotted the energy/operation vs. the peak performance that the processor could achieve, and approximately normalize out the technology's effect on the energy and the performance. The curve clearly shows that achieving more operations/second means that each operation consumes more energy, which means that power increases super-linearly with performance, since power is energy/operation (ops/sec). This is quite contrary to what one needs if the system is power limited: In the power limited world in which we now live, increasing performance means we need to decrease the energy/operation to keep the total power constant. Thus, twice the performance requires each operation consume half the energy.

The move to parallel processing avoided the performance-energy correlation. It allowed each core to be more energy efficient by having a lower peak performance, and added multiple cores on the die to increase the overall performance. In addition, the move to multicore allowed processors to use the energy/delay scaling of changing supply voltages to their advantage. Remember that we need to reduce the energy/operation to enable us to keep the power in check when running at peak performance with all cores operating. To achieve this goal, the processor is run at a supply voltage that is below the peak safe operating voltage for this technology, at a level that is tuned to the individual die. However, now sequential applications run slower than before, since each uses only a single slower core. To improve the performance of sequential application, the supply voltage is increased, and that one core gets overclocked, while the other cores are parked, (that is put in a low-power state). While running at a higher voltage decreases its energy efficiency, since the other core(s) are idle, the overall chip remains within its power window.

Such turbo modes make the accounting of supply voltages even more difficult, since now there are at least three voltages to track, one each for turbo mode, normal mode, and low power/battery mode. For mobile processors, the power-supply voltage is even more complex. Since these systems are concerned about total energy usage (a.k.a. battery life), they need to do computation as efficiently as possible, that is at as low a voltage as possible, while still meeting performance targets. As a result, these systems support a large number of frequencies and supply voltages at which they can run, and have a number of control loops which set the supply voltages and frequencies. This technique is referred to as DVFS (dynamic voltage and frequency scaling). In systems which dissipate static power, the optimal frequency selection becomes a little more complex, since slower operation increases the energy/operation when static power is present,

Parallelism alone is not going to allow computing performance to scale; this is apparent from the graph in Figure 1.1.5 which shows diminishing returns at both extremes. As we try to lower energy, we soon need to forgo large performance factors for small energy savings. Said differently, once we have moved off the “bleeding edge”, the energy gain for going even slower is modest. The same is true for supply voltage. This means that as we scale technology, reducing device switching capacitances, we still get an energy saving since everything is smaller. Thus, a 2× linear shrink will reduce energy by 2×, which should allow the number of cores to increase by a similar amount, if we run each core at the same frequency as they run today. If the intrinsic gate speed improves, small additional performance gain might be possible (by lowering supply to save a little more energy). Note that this scenario is growing the number of cores at half the rate possible from device scaling. This is much less performance scaling than we are used to, since a 2× shrink no longer enables 4× the number of cores on the same die area.

Yet even this slow performance scaling might be somewhat optimistic since it ignores the energy cost of the memory system that is attached to the processor. As shown below in Section 5, this memory-system energy can dwarf the energy of an efficient processor.

5. Don't Forget the Memory Energy

The data in Figure 1.1.5 is not completely accurate, since it assumes that the processor performance would scale linearly with the clock rate as technology scaled. This is not true unless the time the processor spends waiting on memory also scales with the clock rate. Since DRAM access times have scaled slowly, to scale memory stall time we need to decrease the miss rate of the processor's caches by adding a large last-level cache. Today this cache is on the order of 8MB, and needs to be added to the processor's energy budget. The leakage of the large last level cache is very dependent on technology and the circuit tricks used to reduce the leakage when the cache is idle (since most of the cache is idle most of the time). Thus, it is hard to predict how cache leakage will scale. We estimate the leakage to be around 100mW/MB, which matches data we had for 45 to 32nm parts. Correspondingly, we have added this leakage power to the energy of each processor presented in Figure 1.1.6. When this energy is added, the energy efficiency advantage of the older processors goes away, and makes the processor energy change with performance even flatter. A comparison of Figures 1.1.5 and 1.1.6 leads to a startling conclusion: the leakage power of a modern last-level cache is larger than the power of a simple core running full out. This means that a slower energy efficient processor is really not efficient when the entire memory system is considered, unless future technology can greatly reduce this leakage power. Without this development, it is better to use multiple processors sharing the cache such that its leakage cost can be amortized.

The importance of memory energy is shown in Figure 1.1.7 which gives the power breakdown of a recent 40nm, 8-core superscalar processor with an 8MB last-level cache. Over 50% of the processor die energy is dissipated in the caches and register files in this machine. While the cache hierarchy consumes a significant amount of energy, it reduces the overall system energy by eliminating energy intensive memory accesses. Given that the energy/operation is now critical, it is worth revisiting many of the cache design strategies with a view to minimizing the average memory access energy (AMAE) [8]. In this optimization, it is important to consider adding a small level-0 cache to reduce the energy cost of loads, and to consider leakage to determine the optimal cache size, since bigger is no longer always better!

AMAE minimization must also take into account the DRAM energy of the system. DRAM power is generally left out of most processor power analysis, since it is off-die, but must be included in any computing-system analysis. The energy cost of a DRAM access (1 to 2nJ) is a couple of orders-of-magnitude higher than the cost of an internal cache access or functional operation (10pJ). Part of this high cost comes from the very energy-inefficient I/O that DRAM systems use [8], which not only takes over 20pJ/bit, but also require static power to keep the I/O active. This static power makes the effective cost/bit even higher, especially for efficient computation, which minimizes the number of memory accesses. One hopes that this interface power problem can be resolved soon, since very efficient I/O has been demonstrated by many different companies [9], but changing standards can be difficult. Yet even when the I/O is improved, the energy cost of a DRAM access will still be large (10pJ/bit, 0.6nJ/8B) compared to a core processor operation, since requests and data still must travel a large distance on the processor and memory chips to reach this efficient I/O. Thus, much work still needs to be done in this area to find ways to minimize this energy, or at least minimize it for specific kinds of access patterns.

6. Gaining Efficiency through Specialization

Given the energy costs of the memory system, and the constraints on both parallelism and technology scaling, it might seem like there is not much room for energy improvement. Yet there are numerous examples of specialized hardware which is 2 to 3 orders-of-magnitude more efficient than a processor-based solution as shown in Figure 1.1.8 [10]. The key to understanding how much energy saving is available is to look at the energy costs for the fundamental operations of the application. It is critical to consider both the computation and the communication (memory accesses) that are required. The data for various operations in a 45nm technology are shown in Figure 1.1.9, which also gives the energy breakdown of a simple in-order processor. These numbers make two things very clear: First, the programmable nature of a processor has high energy overhead, 70pJ/instruction, vs. a few pJ for an operation. Fetching the instruction, and clocking the state registers that keep the data organized, add a significant cost. Second, if high energy efficiency is required, the application must have very good data locality, since a cache fetch is 20pJ. Fetches from the first level cache have an energy cost that is a significant fraction of that of an instruction, so if data needs to be fetched often, the energy improvement will be modest.

While the energy cost of programmability is high, the dollar cost of creating custom solutions is also high. This combination of forces has instigated a change from bare processor dies to SoCs which integrate a number of application accelerators and processors for energy efficiency. Most processor dies today contain the CPU, GPU, and image processing accelerator, and a number of audio, video, and wireless codecs. This growth in application accelerators has led to research into whether there is enough commonality between a set of applications to create a user programmable hardware engine for accelerating that entire application class. We have seen this sort of evolution before in graphics, where we went from units with limited programmability to a highly-programmable engine for a class of data-parallel floating-point (FP) applications. In fact, since FP arithmetic takes around 1/10 the energy of a simple instruction, forming a SIMD engine which performs the same operation on about 10 data lanes, easily makes the machine's instruction energy dominated by the FP operation, minimizing the cost of the programmability. This is the approach that GPUs have taken, so it is likely that this class of machine might be able to handle all data-parallel FP computation efficiently. Current machines cannot yet do this, since they were architected to maximize performance, and they do not yet leverage locality as much as they could. However, both hardware and software are migrating in this direction, since they need to exploit locality as well, to reduce their energy usage.

Getting the highest energy efficiency requires a very specific combination: very low energy operations, and extreme locality. 1000 MOPS/mW = 1pJ/operation. Such efficiency levels are possible only if the application works on short integer data (8 to16bits), and tens of data operations are completed for each local memory fetch, and roughly a thousand operations are completed for each DRAM fetch. While this degree of reuse seems unlikely, it actually is quite common, and is very similar to what is needed to implement convolution. A problem with extremely-efficient computation is that each computation does not consume much energy. Thus, to amortize the cost of each memory access, it is critical that the outputs of one operation are forwarded directly (or through a local buffer) to another intense computation. While this convolution-like data flow is a very restricted form of computation, most of the accelerators being proposed today fall into this form, including the image, video, and modem processors being integrated on SoCs.

As a result, my research group has been investigating different approaches to creating accelerators for convolution-like applications. We have created an abstract machine model for this class of stencil computations, and are working on a hardware generator and a programmable engine that can support it. Since generating software and drivers is always difficult for custom accelerators, we have created a domain specific language (DSL) for stencil computations that will enable application developers to take advantage of the special hardware in the final product.

7. Tools for Enabling Design

A convolution engine solves only one class of applications, and that is the problem with specialization. Any special solution does not work for all applications, so there are always more problems that need to be addressed. Generating solutions requires a deep understanding of the application, and the corresponding energy costs of computation. Rarely will an unmodified application or algorithm run, let alone reach the desired efficiency on an accelerator. It takes people thinking and working on their problem to find a truly efficient hardware implementation. Thus, if specialization is going to be one of the key strategies for continuing to scale computing performance, our principal task will be to enable a larger group of application experts to participate in creating the efficient hardware/software systems that they require. While the breadth of the application areas makes it unlikely that we can automatically generate highly-efficient hardware given just the algorithm, codifying the principles of efficient hardware implementations seems feasible. We are starting to see the precursors of these tools now, from the current generation of high-level synthesis systems, to hardware generators such as Chisel [11] and Genesis 2 [12] to more domain-specific systems like SPIRAL [13]. Nearly thirty years after the adoption of logic synthesis and place-and-route tools that opened the IC industry to creating application-specific integrated circuits (ASICs), we have come full circle, to the point where we need a new set of tools to enable application experts to access our technology.

As we work on creating these tools, we should remember that not all problems require the "bleeding edge" of performance or energy efficiency. For many applications, current processors/microcontrollers are more than adequate to meet the required system performance. Unfortunately, the people who know that these parts exist and how to use them are likely a distinct group from the people who have applications that they want to implement. Like the high-performance space, we again need tools that allow application experts access to technology, but now at the level of building board-level systems from existing parts. Imagine what would happen if creating a new hardware widget was as easy as writing an iPhone or Android application. This environment would drive a new wave of innovative uses of computing.

8. Conclusion

In summary, our challenge is clear: The drive for performance and the end of voltage scaling have made power, and not the number of transistors, the principal factor limiting further improvements in computing performance. Continuing to scale compute performance will require the creation and effective use of new specialized compute engines, and will require the participation of application experts to be successful. If we play our cards right, and develop the tools that allow our customers to become part of the design process, we will create a new wave of innovative and efficient computing devices.

Acknowledgements:
This work was partially funded by DARPA under the SEEC project, the STARnet CFAR center, and Stanford's Center for Pervasive Parallelism. The author would like to acknowledge the help from Dejan Markovic and Steven Richardson and all his current and former students which made this research and paper possible, especially Ofer Shacham, John Brunhaver, Andrew Danowitz, Sameh Galal, Steven Bell, Krishna Malladi, Wajahat Qadeer, and Rehan Hameed.

References:

[1] http://www.google.com/glass/start/
[2] https://getpebble.com
[3] R. H. Dennard, F. H. Gaensslen, H. N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFETs with very small physical dimensions", IEEE J. Solid-State Circuits, vol. SC-9, pp.256 -268, 1974
[4] http://cpudb.stanford.edu/
[5] Andrew Danowitz, Kyle Kelley, James Mao, John P. Stevenson, and Mark Horowitz, "CPU DB: Recording Microprocessor History", Communications of the ACM 55, Issue 4, pp. 55-63, April 2012.
[6] Robert H. Dennard, Jin Cai, Arvind Kumar, "A Perspective on Today's Scaling Challenges and Possible Future Directions", Solid-State Electronics, Volume 51, Issue 4, pp. 518-525, April 2007.
[7] A. Chandrakasan, Low Power Digital CMOS Design, Ph.D. Thesis, UC Berkeley, 1994.
[8] Krishna T. Malladi, Benjamin C. Lee, Frank A. Nothaft, Christos Kozyrakis, Karthika Periyathambi, and Mark Horowitz, "Towards Energy-Proportional Datacenter Memory with Mobile DRAM", ISCA '12, Proc. of International [9] Symposium of Computer Architecture, pp. 37-48, June 2012.
[9] J. W. Poulton, W. J. Dally, Xi Chen, J. G. Eyles, T. H. Greer, S. G. Tell, C. T. Gray, "A 0.54pJ/b 20Gb/s Ground-Referenced Single-Ended Short-Haul Serial Link in 28nm CMOS for Advanced Packaging Applications," *ISSCC Dig. Tech. Papers*, pp. 404,405, Feb. 2013.
[10] Dejan Markovic, EE292E Lecture Notes, Lecture 5, Stanford University, 2013
[11] https://chisel.eecs.berkeley.edu/
[12] http://genesis2.stanford.edu/
[13]http://www.spiral.net/

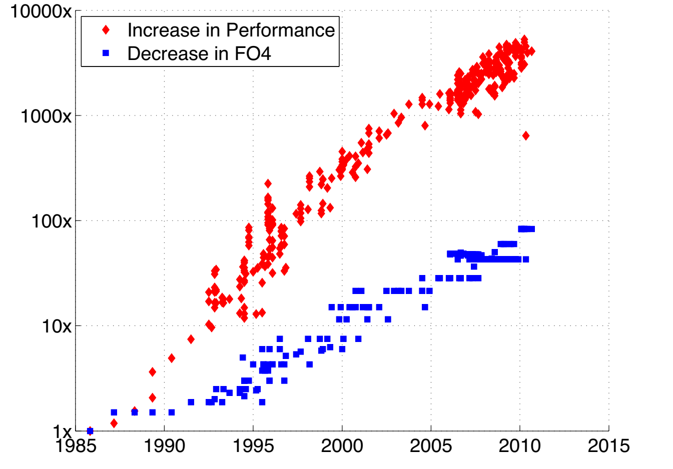

Figure 1.1.1: Improvement in microprocessor and gate performance vs. year.

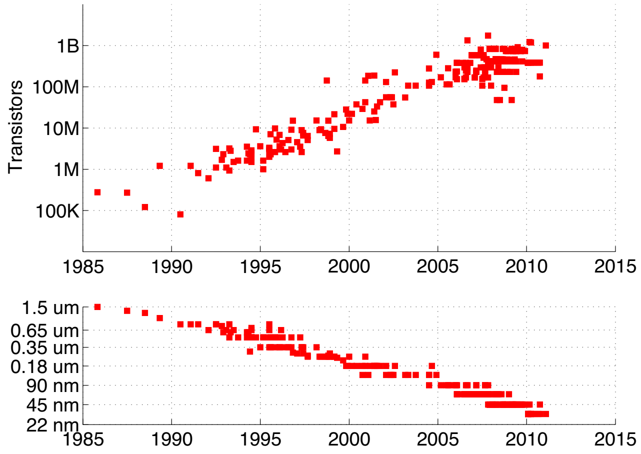

Figure 1.1.2: Number of transistors and feature size vs. year.

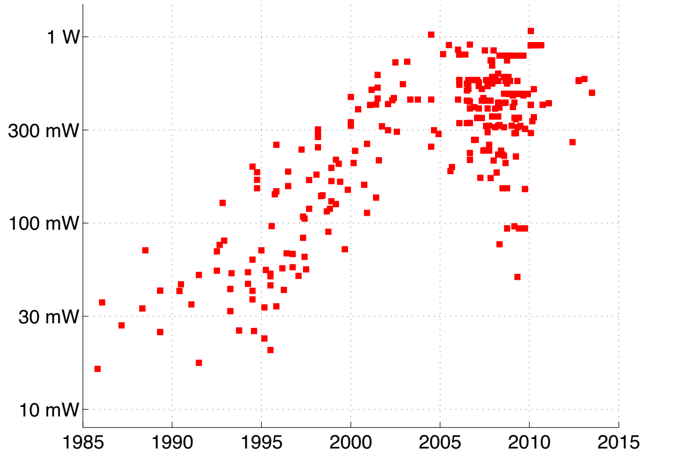

Figure 1.1.3: Power density in mW/mm² vs. year.

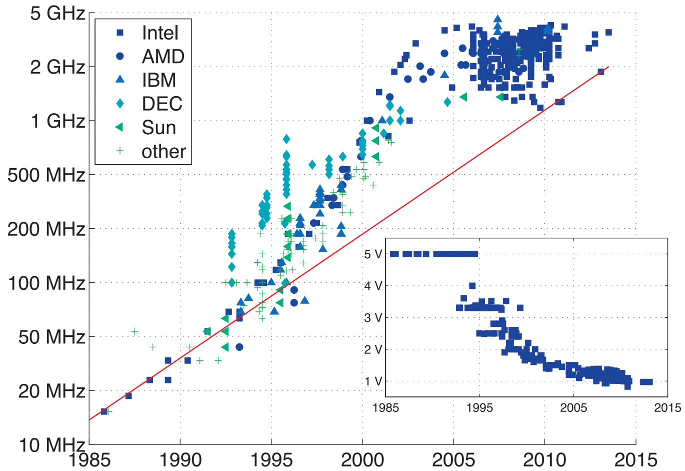

Figure 1.1.4: Clock frequency vs. year. The red line indicates frequency increase due to gate speed. The insert plot is Vdd vs. year.

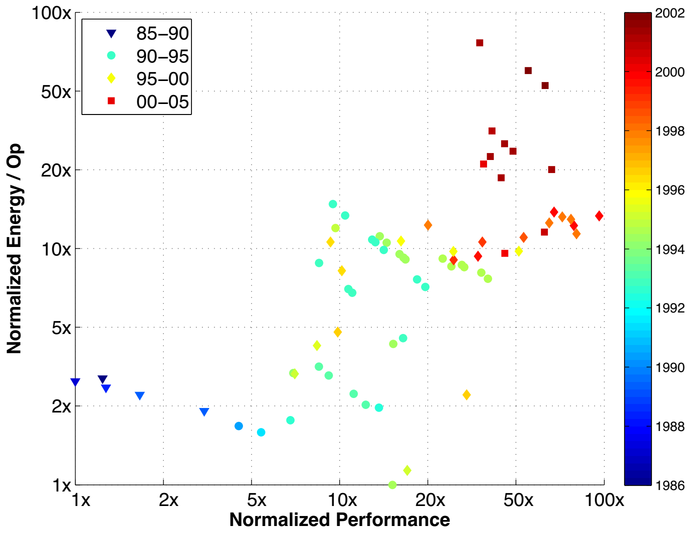

Figure 1.1.5: Instruction energy vs. peak performance (normalized).

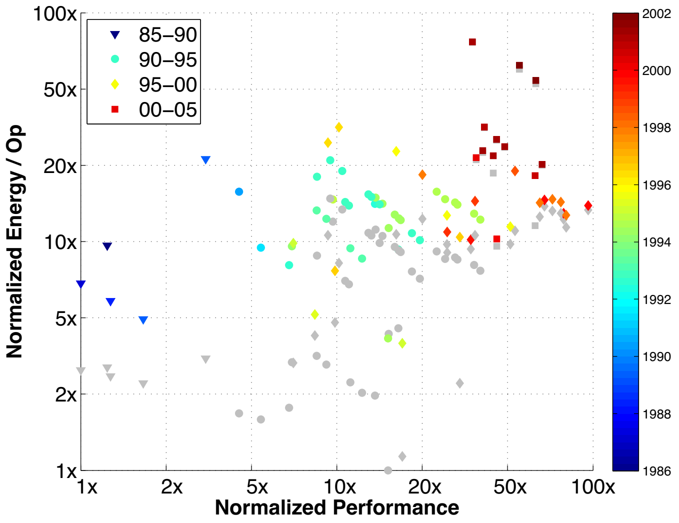

Figure 1.1.6: Instruction energy vs performance, with LLcache leakage added, with original points shown in grey for comparison.

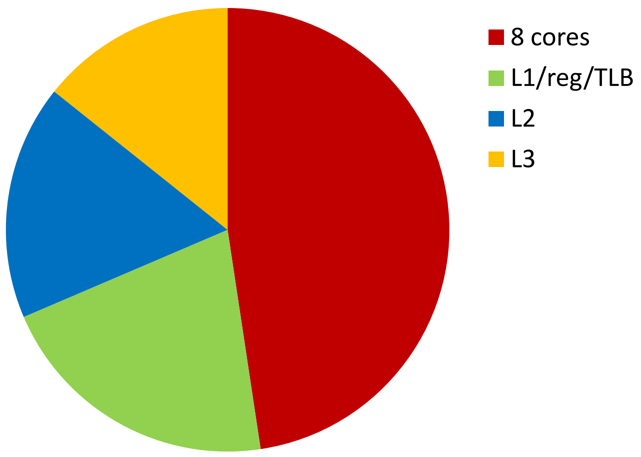

Figure 1.1.7: Power breakdown of an 8 core server chip.

Chip	Year	Paper	Description
1	2009	3.8	Dunnington
2	2010	5.7	MSG-Passing
3	2010	5.5	Wire-speed
4	2011	4.4	Godson-3B
5	2013	3.5	Godson-3B1500
6	2011	15.1	Sandy Bridge
7	2012	3.1	Ivy Bridge
8	2011	15.4	Zacate
9	2013	9.4	ARM-v7A

Chip	Year	Paper	Description
10	2012	10.6	3D Proc.
11	2013	9.3	H.264
12	2012	28.8	Razor SIMD
13	2011	7.1	3DTV
14	2011	7.3	Multimedia
15	2011	19.1	ECG/EEG
16	2010	18.4	Obj. Recog.
17	2012	12.4	Obj. Recog.
18	2013	9.8	Obj. Recog.
19	2011	7.4	Neural Network
20	2013	28.2	Visual. Recog.

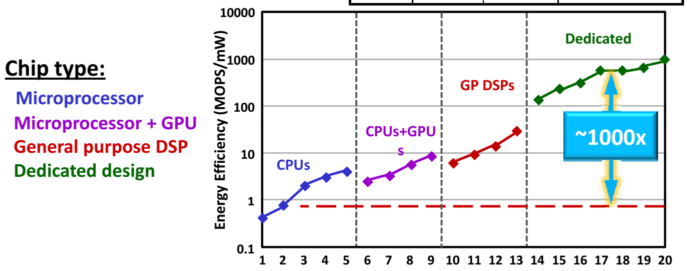

Figure 1.1.8: Energy efficiency of specialized processing, from [10].

Integer	
Add	
8 bit	0.03pJ
32 bit	0.1pJ
Mult	
8 bit	0.2pJ
32 bit	3.1pJ

FP	
FAdd	
16 bit	0.4pJ
32 bit	0.9pJ
FMult	
16 bit	1.1pJ
32 bit	3.7pJ

Memory	
Cache	(64bit)
8KB	10pJ
32KB	20pJ
1MB	100pJ
DRAM	1.3-2.6nJ

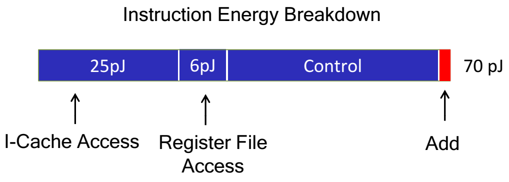

Figure 1.1.9: Rough energy costs for various operations in 45nm 0.9V.