Software Bloat and Wasted Joules: Is Modularity a Hurdle to Green Software?

Suparna Bhattacharya and K. Gopinath, Indian Institute of Science
Karthick Rajamani and Manish Gupta, IBM Research

Adopting an integrated analysis of software bloat and hardware platforms is necessary to realizing modular software that's also green.

System administrators have long observed how software bloat tends to negate increases in computing hardware capacity. While functionally richer and more flexible, newer software packages often incur larger resource overhead in typical execution scenarios. This trend is a consequence of the need to rapidly develop applications with complex business logic and integration requirements while addressing a wide range of operational considerations.

Coincident with the emphasis on application functionality and flexibility, there has been a declining focus on the efficient use of computing resources. During the past two decades, software design paradigms have evolved to prioritize programmer productivity over runtime efficiency, in part due to dramatic improvements in performance enabled through CMOS technology.

However, the gradual reduction in energy efficiency improvements over successive CMOS technology generations, which has limited performance growth, combined with the rising cost of energy, has renewed interest in getting software to better utilize hardware resources.

ENERGY WASTE DUE TO RUNTIME BLOAT

In contrast to programs tuned for a specific use, large software systems are standardized around deeply layered frameworks (or modules) that facilitate rapid development. Each layer is designed to ensure composability of its functions to support a high degree of flexibility for reuse and interoperability. In a typical execution scenario, the system uses only a small subset of functions but still pays the overhead for supporting full functionality. With more layers, the number of potential function combinations grows exponentially, compounding the hidden burden of largely unused combinations.

Software engineering researchers have noted many forms of such runtime bloat—runtime resource consumption disproportionate to the actual function being delivered—including the execution of excess function calls, the generation of excess objects, and the creation of excessively large data structures. For example, Nick Mitchell, Edith Schonberg, and Gary Sevitsky cited hundreds of thousands of method calls and new objects to service a single request and the consumption of a gigabyte of memory per hundred users in an application that needs to scale to millions of users (“Four Trends Leading to Java Runtime Bloat,” IEEE Software, vol. 27, no. 1, 2010, pp. 56-63).

Our review of several case studies supports these findings: a document-exchange gateway that creates six copies or transformations for each input document processed, a telecom application generating a megabyte of temporary objects per transaction, and so on.

Overutilization of resources due to software runtime bloat can result
in higher power consumption and wasted energy.

**ZERO-BLOAT SOFTWARE DESIGN**

To date, server energy-optimization efforts have largely focused on the design of energy-efficient hardware and energy-aware thermal and power management techniques. Energy-proportional design has gained interest as a principled approach to achieve significant energy savings. However, reducing energy waste due to software inefficiencies requires a new perspective.

There’s a parallel between energy waste due to hardware overprovisioning and that arising from the functional overprovisioning of software. Enterprise applications must support extremely demanding levels of variability and interoperability, but they actually exploit only a small fraction of this versatility in a typical deployment situation. Minimizing the runtime energy expended on built-in provisional generality is critical to achieving zero-bloat (green) software.

Of course, some costs in provisioning for generality can’t be completely eliminated. Further, we don’t yet fully understand how to model all such costs to enable systematic approaches for measuring and eliminating associated overhead.

It might be possible to draw a lesson from architecture research, where similar problems have been studied in a more structured setting. For example, researchers with Stanford’s ELM (Efficient Low-Power Microprocessor) project discovered that it’s mostly data and instruction supply overhead in general-purpose programmable embedded processors, not inefficiencies in the core logic, that account for the 50× energy-efficiency gap between these processors and hard-wired media ASICs (W.J. Dally et al., “Efficient Embedded Computing,” *Computer*, July 2008, pp. 27-32). In this project, optimizing instruction and data-supply energy costs improved energy efficiency by 23×, closing the gap with ASICs to within 3×.

Is the overhead expended in getting to the desired data and logic also a key source of bloat in software? Can we address these inefficiencies to help bridge the gap between large framework-based applications and custom-built programs?

**MODELING AND MEASURING BLOAT**

Software doesn’t come with built-in labels that indicate which portions of computation are necessary for a given application and which lead to bloat. Estimating the amount of resources a nonbloated implementation would’ve consumed for a specific execution is also difficult.

Understanding the nature and sources of different types of software bloat is the first step to addressing the issue. The second is to quantify the magnitude of excess resource consumption attributed to each type of bloat. While the latter enables an estimate of how much room for improvement exists, the former provides insights on how to fix the problem.


Current state-of-the-art techniques can’t measure how much overall excess resource consumption and energy waste are attributable to bloat.

MITIGATING AND AVOIDING BLOAT

Tackling the source of bloat usually involves manual source code fixes and assumes some domain knowledge about the application. In a few cases—for example, when the bloat originates in inefficient data structures—advisory tools can minimize manual effort.

Automatic code-optimization techniques can mitigate the symptoms of bloat. For example, static object reuse transformations help reduce the generation of excessive temporary objects due to bloat by amortizing the overhead of repeated object creation (S. Bhattacharya et al., “Reuse, Recycle to De-bloat Software,” Proc. 25th European Conf. Object-Oriented Programming (ECOOP 11), LNCS 6813, Springer, 2011, pp. 408-432). We increased energy efficiency by up to 59 percent with this transformation using the SPECpower_ssj2008 benchmark on an IBM HS21 blade server.

The traditional maxim for creating lean software, as advocated by David Parnas and Niklaus Wirth, among others, is to engineer it right by adopting minimalist design principles that avoid bloat. Such software is built in a series of stepwise refinements carefully crafted to provision each potential use case without sacrificing extensibility or reuse. The Linux kernel illustrates the successful adoption of this principle to efficiently satisfy diverse environments and requirements.

In framework-based environments, however, this approach is impractical. Many redeployable components must be dynamically programmable by business analysts and integrate with dozens of heterogeneous systems and information sources. It’s thus not easy to anticipate usage of a component at design time, nor is it feasible to incrementally change intermediate interfaces later.

Consequently, we propose designing software, programming models, and runtime systems in a way that makes it easier to detect and mitigate bloat. Programmers are often unaware of the overhead that systems might incur during actual deployment, and a low-level runtime optimizer can’t “know” their intentions. Improving cross-layer line of sight into high-level functional intent and interoperability overhead—for example, data-supply inefficiencies such as transformations and copies to facilitate reuse—will help both programmers and runtime systems deliver better energy-optimization solutions.

BLOAT AND ENERGY PROPORTIONALITY

Understanding the causes of software bloat and addressing the challenges to eliminate it are important. However, bloat is unlikely to be completely eradicated where the design focus isn’t just on efficiency but also on flexibility and function. Consequently, it’s equally imperative to understand the exact impact of bloat on system power and energy consumption. Our work using the SPECpower_ssj2008 benchmark (Power-Performance Implications of Software Runtime Bloat: A Case Study with the SPECpower_ssj2008 Benchmark, tech. report RC25150, IBM Research, 2010) revealed several interesting observations in this regard.

The impact of bloat is closely tied to its effect on the utilization of physical resources and whether each such resource is an execution bottleneck. It also strongly depends on the resources’ energy proportionality—that is, their utilization-to-power characteristics.

Reducing bloat that affects a nonbottleneck resource usually decreases system power and energy consumption, the extent of which depends on the change in the resource’s usage and its utilization-to-power characteristics. Reducing bloat that affects a bottleneck resource can increase system throughput, thereby improving energy efficiency and sometimes increasing system power. However, it’s theoretically possible for energy efficiency at the increased throughput to be lower if there’s a disproportionately high cost in power with increased usage of the previously underutilized resources.

In assessing the impact of bloat reduction on a bottleneck resource’s power and energy consumption, it’s important to compare the resource’s efficiency at

- peak achievable performance with and without bloat, and
- equiperformance—that is, at the same performance point—with and without bloat.

The “Power-Performance Impact of Bloat” sidebar illustrates these metrics in more detail. An even more comprehensive analysis by the authors can be found in “The Interplay of Software Bloat, Hardware Energy Proportionality, and System Bottlenecks,” to appear in Proc. 4th Workshop Power-Aware Computing and Systems (HotPower 11), 2011.

In the equiperformance case, bloat can cause a steep increase in power consumption if the underlying hardware has superlinear energy proportionality. For example, we found...
POWER-PERFORMANCE impact of Bloat

Figure A illustrates the power-performance impact of software bloat in the presence of resource bottlenecks and different utilization-to-power characteristics of resources. Figure A1 shows the scenario without bloat, Figure A2 shows the impact of bloat when the bottleneck resource is at its peak utilization, and Figure A3 shows the impact of bloat when the system is at equiperformance.

The diagrams plot the execution time, power, and energy for software that uses three types of resources. R1 is the most power-hungry resource, with a significant utilization-to-power characteristic—for example, a CPU that uses dynamic voltage and frequency scaling. R2 is the bottleneck resource for this workload. R3 is a resource with a bimodal power characteristic—for example, aggressively power-managed memory—using on/off power modes at discrete intervals of coarse granularity. The x-axis represents execution time, with the length of the blue bars indicating the resources’ service-time demands. The z-axis represents power, with the areas over the bars in light and dark brown shading, respectively, indicating energy consumed (power × time) in the scenarios without bloat and incrementally with bloat.

Bloat in bottleneck resource R2 reduces peak achievable performance. Figure A2 shows how this slowdown causes the superlinearly energy-proportional resource R1 to be underutilized (increasing available slack), steeply decreasing its power consumption. The bloat in R3 causes it to be switched on for more time, rounded up to the granularity of its power-management interval. Peak power consumption decreases. The net change in energy consumption can be computed as the difference between the shaded areas for the two scenarios A2 and A1. Note that the energy can increase or even decrease depending on the steepness of power scale down and the granularity of power-switching decisions for R3.

In Figure A3, the bloated resource R2 has been scaled up in performance and power—for example, by changing its operating mode—to have the same execution time as the software without bloat. In this equiperformance scenario, there’s no slowdown due to bloat and thus no change in power consumed by R1. Peak power increases and so does energy consumption; the change can be computed by summing the areas that are shaded dark brown. In the event the power for R2 scales significantly higher with its usage (as with R1), the energy cost of bloat—denoted by the increased shaded area—will be even higher than shown.

that on an IBM Power 750 system with an aggressive load-based energy-saving solution (dynamic voltage and frequency scaling), a reduced-bloat implementation is 1.8× more energy-efficient than a heavily bloated one at equiperformance, though it’s only 1.28× more energy-efficient at peak performance.

Both peak power and equiperformance power have important implications in a datacenter context. In general, peak-power comparisons reflect changes in power-provisioning costs, while equiperformance-power comparisons are good indicators of operational energy cost impacts, particularly in systems with energy-aware management that would run the system at the lowest operating point needed to meet the service-level agreement (SLA). However, equiperformance power can also play a role in power-provisioning considerations. Consider a cluster that is sized for a certain aggregate throughput in which the power delivery limits the individual server’s performance. In this situation, when there’s a change due to bloat reduction, equiperformance power would have a direct impact on power provisioning for the cluster.

Modularity is fundamental to the composability of software packages and to their rapid development and deployment. However, the prevalent approach to achieving it can lead to significant software bloat, which is detrimental to power, performance, and energy efficiency. The real issue isn’t modularity itself but that, because of the difficulty in modularizing functions exactly as
needed, programmers inadvertently introduce superfluous processing and data overhead for reuse.

Understanding the sources and nature of bloat is an important start in addressing the problem. Techniques for measuring, managing, and mitigating bloat face significant challenges but there has also been progress in these areas during the past few years.

Our work demonstrates that the energy impact of bloat isn’t trivial either. It also shows that relations between bloat, bottlenecks, and hardware power characteristics determine the exact impact of bloat on energy efficiency. Consequently, adopting an integrated analysis of software bloat and hardware platforms is necessary to realizing modular software that’s also green.

Suparna Bhattacharya is a PhD student in the Department of Computer Science and Automation at the Indian Institute of Science, Bangalore, as well as a senior technical staff member at the IBM India Systems and Technology Lab. Contact her at suparna@csa.iisc.ernet.in.

K. Gopinath is a professor in the Department of Computer Science and Automation at the Indian Institute of Science. Contact him at gopi@csa.iisc.ernet.in.

Karthick Rajamani is a research staff member and manager of the Power-Aware Systems Department at IBM Research—Austin, Texas. Contact him at karthick@us.ibm.com.

Manish Gupta is director of IBM Research—India and chief technologist of IBM India/South Asia. Contact him at manishgupta@in.ibm.com.

Editor: Kirk W. Cameron, Dept. of Computer Science, Virginia Tech; greenit@computer.org

Selected CS articles and columns are available for free at http://ComputingNow.computer.org.

NEW from IEEE CSPress

THE CONCISE EXECUTIVE GUIDE TO AGILE
by Israel Gat

Get the tools and principles you need to lead an Agile transformation at your organization in this short and practical handbook, delivered digitally right when you need it.

PDF edition • $15 list / $12 members • 21 pp.

Order Online: COMPUTER.ORG/STORE