Towards a Model of Energy Complexity for Algorithms

Ravi Jain, David Molnar, Zulfikar Ramzan

***Abstract*— Energy is a fundamental resource limitation in mobile and wireless devices. A great deal of research in mobile and wireless networking over the past decade has examined ways of reducing energy usage, including specific techniques such as energy-aware protocols for routing and communication. However, to our knowledge, no systematic way has been developed for reasoning generally about the energy consumption of algorithms. Techniques to understand and reason about the time and space complexity of algorithms, in particular asymptotic analysis and the big-Oh notation, have helped place computer programming as well as system design on a firm theoretical and practical footing. Clearly a method for analyzing energy complexity at the same abstract algorithmic level would be invaluable. However, it is not clear that a uniform abstract model of energy complexity can be developed that is both theoretically tractable and has practical predictive ability. Minimizing energy consumption requires making tradeoffs between many resources, including computation, communication, and memory accesses; taking any single resource as a proxy for energy cost neglects these tradeoffs and may lead to a poor model.**

In this paper we survey techniques for modeling and minimizing energy consumption at various system levels, so as to place algorithmic energy complexity in perspective. We then discuss the attributes that a model of energy complexity should have, and describe our initial approach towards developing such a model. We end by discussing future technical directions.

I. Introduction

Energy consumption is a fundamental problem for today's mobile wireless systems. Clearly, energy is a precious resource in any system with entities that depend upon a limited battery source for operation. For example, the energy consumption of a network protocol is due to all entities that are involved in the protocol, including not only the communicating end devices, but intermediate switches, routers, and proxies. In most cases, fixed and wired entities such as routers have a reliable, effectively unlimited, power supply. A mobile device may also enjoy effectively unlimited power, for example when it is used in a car and derives power from the car battery. However, from a purely functional point of view it is typically the mobile wireless devices in the system that face the most critical energy limitation (although minimizing energy consumption at entities with unlimited power may be important for other reasons e.g. environmental concerns [14]). We thus focus on minimizing energy consumption in energy-limited mobile wireless devices.

R. Jain and Z. Ramzan are with DoCoMo Communications Laboratories, USA, Inc. Email: {jain, ramzan}@docomolabs-usa.com.

D. Molnar is with the Department of Electrical Engineering and Computer Science, University of California at Berkeley. Email: dmolnar@cs.berkeley.edu. This work was conducted while he was visiting DoCoMo Communications Laboratories, USA.

Energy consumption in mobile wireless devices is likely to be an issue of growing importance in the future. For several decades, the CPU speeds, storage densities, and the bandwidth capabilities of mobile devices have been increasing rapidly. For example, research prototypes of cellular radio interface subsystems have been shown to be capable of providing data rates of up to 100 Mbps [31]. On the other hand, battery technology has provided capacity increases of only a few percent per year. This *energy bottleneck* indicates that energy will continue to be a fundamental limitation for many years to come in mobile wireless devices, and in fact, is likely to become the most important limitation.

To reduce the energy bottleneck effectively, it must be addressed at all levels of a mobile wireless device. There has been a great deal of research over the past decade to find ways of doing so. We will discuss this research briefly in the next section, but here we point out that it ranges from work at the gate and logic level, to compilers, operating systems, protocols and all the way up to applications. For example, there is a growing body of research on power-aware mobile networking protocols; see Jones et al. [15] for a survey.

In some sense, the research done on reducing energy consumption of a device is analogous to the the research done for reducing computation time in a device, which also ranges from the gate level to compilers, networking protocols and applications. Over the past several decades many techniques have been developed for modeling and analyzing computation time at various system levels and at varying degrees of granularity. These techniques include experimental methodologies, benchmarks, testbeds, and various performance modeling and analysis theories, such as queuing networks, Petri nets, stochastic analysis, as well as related tools.

Most importantly, at the bottom, machine computation time can fundamentally be understood and analyzed in terms of theoretical models of computation, such as the Turing machine and the RAM model of computation [19]. In particular, the scalability of algorithms can be understood in terms of an appropriately chosen model, such as the RAM model, which has been experimentally validated, and it is possible to reason about the computational time complexity of an algorithm in an asymptotic sense. Similarly the space complexity of an algorithm can also be analyzed.

In fact, asymptotic analysis of algorithms can be applied at all levels of a computer, as operations of various degrees of sophistication exist at all levels and their number can be estimated in an asymptotic sense. Thus algorithm analysis techniques and the big-Oh notation are a fundamental part of the toolkit of researchers, system designers, and practitioners

alike. Like all tools, they have limitations and must be used with care, but there is no doubt that they rest on a sound, validated theoretical complexity model and that they have contributed greatly to development and systematic understanding in many areas of computing.

Our central observation is that no such fundamental model seems to exist for reasoning about the energy complexity of algorithms. Such a model could be invaluable in guiding the development of energy-reducing techniques for mobile wireless devices, and this paper aims to take a first step towards such a model.

A device consumes energy in various operations, including computation, communication and memory usage. Thus energy consumption involves tradeoffs that need to be considered carefully. For example, communicating a single bit on a mobile wireless device can be as much as 1000 times more expensive than executing a single instruction. Therefore, one might neglect computation cost and focus on minimizing communication in order to minimize energy. Recently, however, Barr-Asanović analyzed the issue of compressing data before sending it. They found that the amount of computation may outweigh the savings in communication [3]. Through extensive experimentation, they showed that both communication and computation must be considered to reduce total energy cost. Thus developing a model of energy complexity must take several functions into account, and the tradeoffs among them; using CPU time or communication cost as a sole indicator or proxy for energy cost can be highly misleading.

When trying to understand and model energy complexity it should be noted that the effectiveness of an algorithm or energy reduction technique depends upon the energy context in which it is applied. This is especially important for mobile computing, as wireless devices range from relatively power-rich laptops to severely resource-constrained sensor nodes. For example, the multicast time maximization protocol of Floréen et al. [7] requires each node to solve a linear program at each step, in order to reduce energy consumption by minimizing computation. For a laptop or even a cell phone, this may be reasonable. For a sensor node, it may not be, as the cost of solving a linear program may be too high compared to the cost of communication. While sensor nodes have come to be recognized as special cases due to their extreme limitations, there are more and less powerful sensor nodes, and other classes of devices also have gradations. Thus the energy usage of an algorithm, like its computation time, must be evaluated, and its effectiveness judged, in the intended application context.

A complication in the development of an energy complexity model is that unlike CPU time and storage resources, the energy available from a fixed unit source such as a battery seems to have some ”elasticity” that depends upon the usage pattern; for example typical dry-cell alkaline batteries last longer if they are used in short bursts rather than drained continuously [5]. In our initial approach we do not deal with this issue directly, and instead assume for simplicity that it suffices to estimate the energy consumed by various device functions such as computation, communication and the like. However, as we discuss below, we indirectly consider this issue in terms of the timing at which various functions are carried out.

Our approach is to develop an energy complexity model with parameters that precisely express the functions invoked by the algorithm, such as computation, communication and memory usage, as well as the tradeoffs between them. We take as our starting point the most basic of all models of computation, namely the Turing machine. Our Augmented Turing Machine (ATM) model explicitly includes a notion of “switching cost” – the energy cost to switch between different units of the processor. While switching cost has recently been recognized by the compiler community as an important component of energy cost, algorithms do not currently take it into account. The ATM model allows us to treat switching cost more formally, thereby easing the task of producing widely applicable energy-efficient algorithms.

Practice and theory have different but related demands for an energy model. A practitioner needs an energy model that has predictive power: given a piece of code and some information about the input, the model should return an answer for energy consumption close to the consumption of real hardware. A theorist, on the other hand, requires a model that allows for ease and generality of analysis, and provides an answer that is not dependent upon the specific hardware. The Turing machine is clearly a model suitable for theorists rather than practitioners. However, it provides a convenient, sound and well-accepted starting point that can be a basis for more practical models. For instance, just as the Turing machine model has been replaced by other models, such as the RAM model, that are more convenient for carrying out algorithm analysis, we believe our approach can also be developed in a similar manner. However, it is not yet fully clear that a uniform abstract model of energy complexity can be developed that is both theoretically tractable and has practical predictive ability.

This paper makes several contributions. We argue that an abstract model for algorithmic energy complexity is desirable and possible for mobile wireless devices. We give an initial approach to developing such a model starting at the theoretical end via the Augmented Turing Machine. We then show that our model provides a “well-behaved” complexity measure in the sense of Blum [27], and derive basic theorems relating to the languages that the ATM accepts. We also show that time-space tradeoff results for algorithms like the Fast Fourier Transform, in the basic Turing Machine model, imply energy-space tradeoffs in the ATM model.

The rest of this paper is organized as follows. Section II-A briefly surveys the large range of technical work on reducing energy consumption at various levels of mobile wireless devices, including logic, processors, operating systems and compilers. We also survey, in Section II-B, two previous approaches to developing models that can provide inspiration for our work, namely I/O complexity models and a time-energy cost model. In Section III we define the ATM and derive some of its basic theoretical properties. A rich field of future work awaits, which we sketch in Section IV.

II. Related Work

We now discuss low-level work on improving energy consumption as background for our model. Work on improving

the energy consumption on embedded devices has been done at four levels: the logic design level, the processor level, the operating system level and the compiler level. We discuss each of these lines of research in turn. A portion of the exposition presented below is a summary of the survey in chapter 2 of Havinga's thesis [10]. In addition, a considerable amount of work has been done in reducing energy at various levels of the networking protocol stack. We will not discuss this work here due to space limitations, but refer the reader to surveys and recent papers and references therein [15], [4], [3], [30].

Thus, and as will be clear from the discussion below, work on reducing energy consumption of mobile wireless devices has been proceeding systematically at several levels. However, a notable exception is in the fundamental area of algorithms. We believe that a model for reasoning about the energy complexity of algorithms will fill an important gap.

Finally this section discusses an algorithmic complexity model based on I/O rather than space or time as well as a combined time-energy model. Such alternate models can guide us in putting forth our own model.

A. Reducing Energy Consumption

1) Improving Energy Complexity at the Logic Level: One of the lowest levels at which one can improve energy consumption is in the underlying logic design. For example, we can use cell libraries which comprise transistors of varying sizes. For this reason, cell libraries have lower capacitances than gate arrays and therefore consume less power when switching. Moreover, one can try to optimize the cells themselves to use less power; for example, by preventing long rise and fall times.

Reversible logic is another very low-level mechanism for saving energy. The central premise of using such logic in the design of systems is to effectively recycle the energy used, thereby emitting very little heat. Whereas traditional logic overwrites information resulting from a logical operation (e.g., the "AND" operator loses information about which of the input bits were set to 0 if the AND evaluates to 0) reversible logic is designed to not throw any such information away. This can actually impact energy consumption since whenever a bit of data is overwritten, energy is discharged at the location where that bit was stored because of the change in voltage (e.g., from positive to negative). More formally, $\ln 2 \times kT$ joules of energy are dissipated whenever a system erases a single bit of information; here k is Boltzmann's constant and T is the temperature. At room temperature (approximately 300 Kelvin), the dissipation is about 2.9×10^{-21} joules. For further information on reversible computing, see the manuscript based on Frank's MIT Ph.D. thesis [8].

Frequent "switching" during execution also leads to high energy complexity – and such switching can often be analyzed at the algorithmic level (though, to the best of our knowledge, switching as a complexity-theoretic measure has not been studied previously). The following measures identify switching as a key component of energy consumption:

- **Clock Gating.** Power consumption in CMOS is proportional to the clock frequency. One can therefore reduce power by dynamically turning off any clocks to logic that is temporarily unused. A clock gating scheme employs a control signal which, when true, toggles the clock, and when false holds the clock steady. Clock gating can be employed locally (e.g., clocks to specific registers) or more globally (e.g., to turn off clocks to larger modules in the processor). In either case, trying to exercise logic thoroughly before switching to other logic can lead to lower energy consumption.
- **Changes to State Machines.** Similar to clock gating, one can turn off portions of a state machine that are inactive, which may be especially effective since only one "sub-machine" can be active at any given time. Another energy-optimization to state machines is to create an encoding of the state space that strives to minimize the expected number of bits toggled during a state change; i.e., to minimize the Hamming distance between the encoding of the state that was transitioned from and the state transitioned to. Of course, such an optimization requires understanding the behavior of state transitions at a statistical level. We remark that one may use clock-gating to lower state machine power consumption by disabling an actual transition when a self-loop is detected, as was studied in the work of Koegst et al. [16]
- **General Logic Encoding.** In general, any state change in a signal (e.g., a clock, a data pin, or an address line) consumes energy. One possible way to reduce energy consumption, therefore, is to choose an encoding that results in fewer transitions. Consider, for example, a way to encode a program counter. The standard mechanism (using a base-2 ripple-carry adder to increment) results in n bit flips, if the least significant n bits are 1 and the $(n+1)^{st}$ bit is 0. On average, this yields a change of two bits per increment. One might try alternate encoding schemes (such as one based on a Gray code) which might result in a single bit change. Of course, an incrementer in such a case might require more logic to implement. An alternate scheme for encoding information on a bus with N lines is the *bus-invert code* [23] which works as follows. If two consecutive values to be transmitted on the bus have Hamming distance greater than $N/2$, one can instead transmit the bitwise not of the second value. An extra bus line is needed to specify whether the bits were flipped, but at least we guarantee that the Hamming distance between successive values is at most $N/2$.

Having discussed low-level mechanisms at the logic level, we move to a higher level of abstraction and discuss how processor design and layout can effect power consumption.

2) Improving Energy Complexity at the Processor Level: Several techniques improve processor-level energy complexity. We consider two mechanisms: organization of memory and layout of communication channels.

ORGANIZATION OF MEMORY. There are three major types of memory: main memory, cache, and secondary storage – each consumes a significant amount of energy. We discuss how one might try to minimize the impact of each type in turn:

- **Main Memory.** Main memory, which might be in the form of DRAM or SDRAM, is typically in one of three

states: active, standby, or off. As one might expect, energy consumption is greatest in the active state – potentially several hundred times larger than when in the standby state [11]. Therefore, trying to cluster memory requests so that they can all be handled at once would leave main memory in the standby state longer, thereby reducing energy consumption. Along the same lines, breaking a large chunk of memory up into smaller "sub-memories" can reduce energy consumption since only one of the sub-memories need be active at any given time. This approach is followed in the Direct Rambus DRAM system [20]. One can further design memory chips to be more energy efficient – not surprisingly, this can be achieved by making the chips faster, thereby increasing the amount of time spent in standby mode [22].

- **Cache.** The concept of caching is one of the most well-known ideas in computer science. One can reduce energy consumption by exploiting spatial and temporal locality of memory accesses so that data that is more likely to be accessed is placed in smaller, faster and more energy-efficient storage. Moreover, with a good cache-hit ratio, the traffic in other parts of the processor (e.g., the bus) is decreased, which further lowers energy consumption. We remark, however, that to create energy-efficient caches may itself be a challenge. For example, if a complicated architecture is used to obtain a better hit ratio, then this may have a negative impact on the energy complexity.
- **Secondary Storage.** On a typical processor today, secondary storage involves a magnetic disk. While the motor of the disk is running, energy is being consumed. Although one can turn this motor off, if it needs to be turned on again, the process may itself consume considerable time and energy. As an alternate method for secondary storage, especially for more weight sensitive mobile devices, we may use flash memory. Like a hard disk, flash is non-volatile and can store data without consuming energy. Flash also consumes far less memory than a traditional magnetic disk. Estimates range at a 60% to 90% reduction in secondary storage power consumption compared to a traditional magnetic disk [6].

In addition to various memory and storage considerations, energy consumption may be improved in the intra-processor communications channels such as buses.

INTRA-PROCESSOR COMMUNICATION. It has been shown that between 10 to 40% of power is dissipated in communications channels such as buses [1]. One can use similar principles from caching. In particular, if buses are designed to exploit locality – for example, by breaking up a system into smaller subsystems where the majority of communication is done within a given subsystem as opposed to between two subsystems – then most data can be passed on shorter and more energy efficient "local" buses.

3) Improving Energy Complexity at the Operating System Level: One common way to reduce energy consumption at the operating system level is via scheduling. Traditionally, scheduling is based on factors such as priority and latency. However, careful scheduling can also lead to energy reduction. For example, if the system enters an idle state in which there is no important computation being performed, the operating system can power off significant portions of the system. In general, there is much research on this and similar approaches for improving energy consumption at the operating system level. See [32], [33], [34] and the references therein.

4) Improving Energy Complexity at the Compiler Level: Several techniques reduce compiler-level energy complexity. These include instruction choice and instruction ordering. The latter is geared towards the obvious maneuver of trying to determine which instructions tend to consume more energy, and trying to avoid such instructions when compiling. With respect to instruction ordering, the goal is to minimize the number of switches between functional units.

Of course, traditional compiler optimizations for code size and speed will also yield energy reductions. For example, if specific instruction operands are placed in the same memory bank, an optimizing compiler might retrieve them with a single double-operand move operation. This eliminates a memory access and reduces.

B. Models for I/O and Energy

1) Algorithmic Complexity Models for I/O: In addition to the traditional complexity-theoretic measures of time and space, researchers have worked on adding I/O to the list. See the survey by Shriver and Nodine [21]. This line of work was motivated by algorithms operating on a data set too large to fit into main memory. Such algorithms are required to make (parallel) I/O calls to retrieve data from a secondary storage device (e.g., disk or tape), operate on this data, and then write it back to the secondary storage device. Because calls to off-chip I/O devices require far more time than on-chip operations, limiting the number of such calls leads to performance improvements in practice. Furthermore, not taking I/O calls into account may also lead to algorithms with worse *asymptotic* behavior. For example, Alpern et al. [2] showed that cache and page misses can result in a $O(k^5)$ running time for the (naive) $O(k^3)$ implementation of multiplication for $k \times k$ matrices.

To decrease the number of parallel I/O calls, one typically has to design algorithms that can effectively break up the original problem into subproblems involving a working data set that can fit into the available on-chip memory. Moreover, since typical I/O devices have restrictions on how data can be read in parallel, data corresponding to a sub-problem should be placed on disk to decrease the number of calls needed to get this data into main memory.

2) Time-Energy Cost Models: Martin [18] also sought to develop a complexity theory of energy. He argues that an overall energy-time cost measurement for an algorithm should be $E \times t^2$ (where E is the energy used and t is the time used) as opposed to $E \times t$ (which others have used). The reasoning is as follows. Time is directly related to voltage in that if one reduced the voltage in a processor by a certain factor, than the amount of time needed for execution would essentially go up by that same factor. Energy, on the other hand, is directly proportional to the square of the voltage. So, suppose you had two algorithms A_1 and A_2 where the time required by A_1 is

twice that required by A_2, but the energy consumed by A_1 is half that consumed by A_2. According to the measure $E \times t$, these algorithms have equivalent cost. But, if you halve the voltage on the machine running A_2, then both A_1 and A_2 would run in the same amount of time. However, the energy consumed by A_2 would go down by a factor of 4 (because of the quadratic relationship between energy and voltage).

Martin left open the question of how one could treat energy as a complexity-theoretic resource at the algorithmic level. This is our focus.

III. Augmented Turing Machine Energy Model

The Turing Machine model was introduced by Alan Turing in 1936 [26]. The goal was to capture the kinds of mathematical functions that are computable. Since then, the Turing Machine has also found its place in computational complexity theory – a field which is interested not only in what is computable, but also in how *hard* it is to compute.

At a high level, a Turing Machine has a finite control, an input / output tape divided into individual cells (each of which can contain a symbol), and a tape head that can overwrite the contents of a cell and move at most one cell over according to instructions received from the finite control. A single move of a Turing Machine depends on the current symbol the tape head is scanning and the finite control's current state. In a single move this symbol is overwritten (possibly by the same symbol – leaving the tape contents identical), and the tape head is moved either to left or right. Figure 1 provides a high-level illustration of a Turing machine.

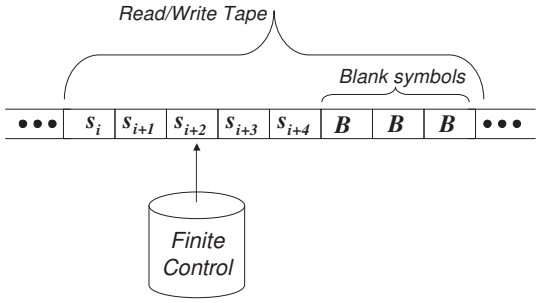

Fig. 1. A high-level depiction of a single-tape Turing Machine

We consider a theoretical energy consumption model based on an *Augmented* Turing Machine (ATM). We the Turing Machine's finite control to also include the amount of energy consumed during a transition between states. The total energy consumed on an input is then the sum of the energy consumed by each transition. More formally, an energy-augmented Turing Machine M' is a 9-tuple:

$$(Q, \Sigma, \Gamma, \delta, \xi, q_0, B, F, A),$$

where all components except ξ are the same as for a basic Turing Machine, and are defined as follows:

- Q is a finite set of states.
- Γ is a finite alphabet of allowable tape symbols.
- $B \in \Gamma$ denotes the blank symbol.
- $\Sigma \subseteq \Gamma - B$ is the alphabet of input symbols. In a typical situation, the inputs consists of binary strings, in which case Σ would equal $\{0, 1\}$.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function. Its inputs are the current machine state and the current symbol being read. It specifies the next state, the symbol which overwrites the contents of the tape cell being scanned, and the direction that the tape head moves.
- $q_0 \in Q$ is the machine's start state; i.e., the state the finite control is in prior to reading any input symbols.
- $F \subseteq Q$ is the set of final states – once the machine is in such a state, the computation is terminated.
- $A \subseteq F$ is the set of accept states. We say that a Turing Machine accepts its input if the finite control enters one of the states in A.
- $\xi : (Q \times \Gamma) \times (Q \times \Gamma) \to \mathbb{N}$ is a computable function specifying the number of energy units consumed as a function of the machine's current state and the symbol under the tape head, and the next state and tape symbol. Here $\mathbb{N}$ is the set of positive integers.

If, on input x, the machine M' requires t time steps and the state-symbol pairs are:

$$\langle (q_0, s_0), (q_1, s_1), \ldots, (q_t, s_t) \rangle,$$

then we denote the total number of energy units consumed by

$$E_{tot}(M', x) \triangleq \sum_{i=1}^{t} \xi((q_{i-1}, s_{i-1}), (q_i, s_i)).$$

If, for any input of length n, the machine M' consumes at most $E(n)$ energy, we say that M' is an $E(n)$-energy bounded augmented Turing Machine. We can also define an analogous complexity class, for any fixed measurement function ξ:

$$\mathsf{DENER}(E(n), \xi) = \{L \mid L = L(M') \text{ for an } E(n) \text{ energy-bounded Turing Machine } M'\},$$

where $L(M')$ is the language accepted by machine M'. Further we define $\mathsf{DENER}(E(n)) \triangleq \bigcup_{\xi} \mathsf{DENER}(E(n), \xi)$.

Discussion. Before proceeding, two remarks are in place about ξ. First, observe that its range is the positive integers $\mathbb{N}$ (as opposed to the real numbers $\mathbb{R}$). In this case, the number of possible choices of ξ is countably infinite, which facilitates theorems whose proofs might require enumerating all possible machines (e.g., for a diagonalization argument). Second, we do not allow ξ to take on the value 0 for any input. Permitting the value to be 0 is not intuitively appealing since we expect some energy to be consumed during a single operation. Interestingly, allowing 0 values also results in theoretically unpleasant consequences. In particular, consider an energy mapping ξ for a Turing Machine M that is 0 everywhere except for transitions into a final or halting state (where it takes on the value 1). Then, determining whether M has non-zero energy consumption is tantamount to determining if M halts, which is an undecidable problem.

The ATM model also captures several traditional sources of energy consumption. For example, if ξ is identically 1 on all inputs, then the energy complexity amounts to the time complexity. Similarly, if ξ is higher for certain output symbols (such as any non-blank symbol) then, the energy complexity

captures the I/O complexity (which can capture communication complexity as well as the number of "memory" accesses). Finally, if we can imagine partitioning the states in the finite control into various "functional units," then we can have ξ be higher for transitions between states in different units as opposed to transitions between states in the same unit. This captures the energy consumption of switching between functional units that Steinke et al. examined [24].

ENERGY COMPLEXITY THEOREMS. Our energy complexity metric satisfies the requirements of a Blum complexity measure [27], which immediately implies a theoretical "speed-up" theorem. This means that our measure is theoretically well behaved. We can also derive an energy hierarchy theorem similar to those obtainable for time and space. We describe these results below.

A function $A(M, x, n)$ is a *basic complexity measure* for energy if it takes as input a description of an ATM M, and checks whether M consumes n units of energy when run on input x; i.e., $A(M, x, n) = 1$ if and only if M on input x consumes n units of energy. Blum's seminal paper [27] set out two axioms for such complexity measures. Axiom 1 states that if M halts, then there exists an n such that $A(M, x, n) = 1$. Axiom 2 states that A is *total recursive*; i.e., there is a deterministic Turing Machine implementing the function A that halts on all inputs. The next two theorems respectively show that the ATM model provides us with a Blum complexity measure and that, as a result, there is a "speed-up" theorem.

Theorem 1: There is a Blum complexity measure for energy consumption in an Augmented Turing Machine.

Proof: We implement A as a Turing Machine that simulates M on input x, and keeps track of E_{tot} (it is easy to keep track of E_{tot} since it only involves simple sums). Axiom 1 is satisfied because if M halts, then we can set n to be E_{tot}. Axiom 2 is satisfied because E_{tot} is monotonically increasing in the time steps taken by M (each step uses at least one energy unit). Therefore, within $n + 1$ steps M has either halted or used more than n energy units. Since A need not continue simulating M past this point, it can halt. ■

Theorem 2: Let $r(n)$ be an unbounded computable function. There exists a language L such that for all machines M that decide L using energy $t(n)$, there exists an M' that decides L using energy $r(t(n))$.

For details on the speed-up theorem, see Blum [27]. Note that if $r(n) \leq n$, then the energy consumed by M' is less than that consumed by M. We can further prove the following theorems, using standard proof techniques. For further information, see Papadimitriou [19]. The first theorem is a "gap" theorem which shows that for any energy bound, there is a language which cannot be decided within it.

Theorem 3: For any energy bound $E(n)$ there is a recursive language L not decidable in $\mathsf{DENER}(E(n))$.

For the next theorem, we need the notion of an energy-constructible function. We call a function $E(n)$ energy constructible if there is an augmented Turing Machine M' and an n-bit input x for which M' actually consumes exactly $E(n)$ energy. The next theorem says that with an increase in energy by a logarithmic multiplicative factor, we can decide new languages. From a practice-oriented perspective, this "hierarchy" theorem tells us that adding a little extra battery power allows us to solve new problems.

Theorem 4: Let $E(n)$ be an energy-constructible function and suppose that we restrict ourselves to mappings ξ that take on a maximum value μ. Then, there is a language L in $\mathsf{DENER}(\mu E(n) \log E(n))$, but not in $\mathsf{DENER}(E(n))$.

Proof: (Sketch) Observe that $\mathsf{DENER}(E(n)) \subseteq \mathsf{DTIME}(\mu E(n))$. This follows since at most μ units of energy can be consumed at each time step. Now, by the standard time hierarchy theorem (see Hartmanis-Stearns [9] or Hopcroft-Ullman [13], there is a language L in $\mathsf{DTIME}(\mu E(n) \log E(n)) - \mathsf{DTIME}(\mu E(n))$. Finally, $\mathsf{DTIME}(\mu E(n) \log E(n)) \subseteq \mathsf{DENER}(\mu E(n) \log E(n))$ since at least one unit of energy is used at each time step. Therefore, $L \in \mathsf{DENER}(\mu E(n) \log E(n))$, but not in $\mathsf{DTIME}(\mu E(n))$. ■

Note also that if $\mu = 1$, energy complexity is exactly time complexity. In this case, the theorem yields a gap between $\mathsf{DENER}(E(n))$ and $\mathsf{DENER}(E(n) \log E(n)$, which is known for time complexity.

ENERGY-SPACE TRADEOFFS FOR FFT. We now discuss how the ATM model can capture space-energy tradeoffs, using the Fast Fourier Transform as an example. Aside from energy, RAM space is also an important resource for mobile devices. While the ATM model can capture I/O (including memory access) in the energy cost, the total amount of space used is a separate resource. Many wireless devices are embedded or low-power, and so space is at a premium. At the same time, we would like to perform multimedia computation involving the FFT and other algorithms. It turns out we can state some precise tradeoffs in the ATM model between energy and space, leveraging previously known results. Again, these theorems are well known, but have not been previously applied directly to energy cost. For example, Savage-Swamy [29] showed:

Theorem 5: Every circuit computing FFT that uses at most S space requires $\Omega(n^2/S)$ time.

Proof: See Tompa [28]. ■

Tompa extended this bound to similar time-space tradeoffs for a wide class of functions, including convolution and polynomial multiplication. In the ATM model, this gives an energy-space tradeoff, as energy cost is at least time cost. Therefore we see that every ATM machine for FFT that uses at most S space must use $\Omega(n^2/S)$ energy.

BENEFITS AND DRAWBACKS. The ATM model is based firmly on a theoretical foundation. We found that the notions introduced seem to be well behaved from a complexity theoretic perspective. Moreover, we can leverage existing results in computational complexity theory. On the other hand, a theoretical model such as this one gives us little predictive power and is not easy to use for a practitioner.

The ideal model should allow us to leverage the existing body of theoretical work, but still have practical predictive properties. By "practical prediction" we mean that the model should be simple, with few parameters which can be easily estimated from hardware measurements. We could then state results about algorithms in this model, and then use the estimated parameters to translate these into a reasonable

prediction of the algorithm's energy use. We believe this is a promising direction.

IV. Conclusions and Future work

We believe the large body of work on energy optimization which has existed at the logic/processor level, the operating system level, and the compiler level should be considered at the algorithmic level.

Our ultimate goal is to have one energy model for both practical and theoretical uses. As a first step, we developed the augmented Turing Machine model, which extends well-known definitions. The augmented Turing Machine model appears to be theoretically well behaved. In particular, the resulting energy complexity measure satisfies the Blum axioms, and yields gap, speedup, and hierarchy theorems. Furthermore, the augmented Turing Machine model captures notions that are traditionally used to estimate energy consumption such as time, communication, and I/O.

Theoretical models allow us to leverage existing results from computational complexity theory. On the other hand, a theoretical framework, such as the one we suggested, does not have very much practical predictive power. That is, it does not give us an accurate indication of how much energy an algorithm could consume on an actual microprocessor. The ideal model should be a marriage between theory and practice. It should provide enough predictive power that an algorithm designer or implementer can accurately estimate energy consumption. At the same time, the model should be simple (i.e., have few parameters), and theoretically sound (i.e., it should allow one to prove new theorems or leverage existing ones).

We are currently developing an energy complexity model that we believe will be theoretically tractable but still provide predictive power. We are also conducting experiments to validate the model for a mobile wireless device environment.

V. Acknowledgement

We thank Lawrence Brakmo for a number of very helpful discussions on energy consumption.

References

[1] A. Abnous and J. Rabaey. "Ultra-low-power domain specific multimedia processors," *Proceedings of VLSI Signal Processing IX*, pp 459–468, November 1996.

[2] B. Alpern, L. Carter, E. Feig, and T. Selker. "The Uniform Memory Hierarchy Model of Computation." *Algorithmica*, 12(2/3):72–109. August and September 1994.

[3] K. Barr and K. Asanovic, "Energy Aware Lossless Data Compression", First International Conference on Mobile Systems, Applications, and Services (MobiSys-2003) , San Francisco, CA, May 2003

[4] C. F. Chiasserini, P. Nuggehalli, V. Srinivasan and R. R. Rao, "Energy-Efficient Communication Protocols," (Invited Paper), *Proc. Design Automation Conf. (DAC)*, June 2002.

[5] C. F. Chiasserini and R. R. Rao, "Pulsed Power Discharge in Communication Devices," *Proc. Mobicom*, 1999.

[6] F. Douglis, F. Kaashoek, B. Marsh, R. Caceres, K. Li, and J. Tauber. "Storage Alternatives for Mobile Computers," *Proceedings of the 1st Symposium on Operating Systems Design and Implementation*, pages 25–37, November 1994.

[7] P. Floréen and P. Kaski and J. Kohonen and P. Orponen. "Multicast time maximization in energy constrained wireless networks," Proceedings of the 2003 joint workshop on Foundations of mobile computing. Pages 50-58, San Diego, CA, USA. ACM Press.

[8] M. P. Frank. "Reversibility for Efficient Computing," *Manuscript based on MIT Ph.D. Thesis.* December 1999.

[9] J. Hartmanis and R. E. Stearns. "On the computational complexity of algorithms," *Transactions of the American Mathematical Society*, volume 117, pp 285–306.

[10] P. J. M. Havinga. "Mobile Multimedia Systems" Ph.D. thesis, University of Twente, February 2000, ISBN 90-365-1406-1.

[11] P.J.M. Havinga and G.J.M. Smit. "Design Techniques for Lower Power Systems," *Journal of Systems Architecture*, Volume 46, Number 1, 2000.

[12] F. C. Hennie. "One-tape off-line Turing Machine Computations," *Information and Control*, Volume 8, Number 6, pp 553–578.

[13] J. E. Hopcroft and J. D. Ullman. "Introduction to Automata Theory, Languages, and Compuation," *Addison-Wesley Publishing Company*, 1979.

[14] R. Jain and J. Wullert II, "Challenge: Environmental Design for Pervasive Computing Systems," *Proc. Mobicom*, 2000.

[15] C. E. Jones, K. M. Sivalingam, P. Agrawal and J. C. Chen, "A Survey of Energy Efficent Network Protocols for Wireless Networks," *Wireless Networks*, vol. 7, no. 4, 343-358, July 2001.

[16] M. Koegst, G. Franke, S. Ruelke, and K. Feske. "Lower power design of FSMs by state assignment and disabling self-loops," *Proceedings of Euromicro 1997*. Pages 323–330, September 1997.

[17] P. Lettieri and M.B. Srivastava. "Advances in Wireless Terminals," *IEEE Personal Communications*, pp6-19, February 1999.

[18] A. J. Martin. "Towards an Energy Complexity of Computation," *Information Processing Letters*, 77 (2001) 181–187.

[19] C. Papadimitriou. "Computational Complexity," *Addison-Wesley Publishing Company*, (Reprinted with corrections, 1995.)

[20] Rambus, Inc. `http://www.rambus.com.`

[21] E. Shriver and M. Nodine. "An introduction to parallel I/O models and algorithms," *in R. Jain, J. Werth and J. C. Browne, Input/Output in Parallel and Distributed Computer Systems*, Kluwer, 1996.

[22] T. Simunic, L. Benini, and G. De Micheli. "Cycle-Accurate Simulation of Energy Consumption in Embedded Systems," *Proceedings of the Design Automation Conference, DAC 1999.*

[23] M.R. Stan and W.P. Burleson. "Bus-Invert Coding for Lower-Power I/O." *IEEE Transactions on VLSI Systems*, Volume 3, Number 1, pp 49–58, 1995.

[24] S. Steinke, M. Knauer, L. Wehmeyer, and P. Marwedel. "An Accurate and Fine Grain Instruction-Level Energy Model Supporting Software Optimizations," *Proceedings of PATMOS 2001.*

[25] V. Tiwari, S. Malik, and A. Wolfe. "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume 2, Number 4, December 1994.

[26] A. M. Turing. "On computable numbers, with an application to the *Entscheidungsproblem*," *Proc. London Math. Society*, 2, 42, pp. 230–265, 1936. Also, 43, pp. 544-546, 1937.

[27] M. Blum. "A Machine-Independent Theory of the Complexity of Recursive Functions." *J. ACM* 14(2), pp 322-336, 1967.

[28] M. Tompa. "Time-space tradeoffs for computing functions, using connectivity properties of their circuits." *STOC 1978*.

[29] J.E. Savage and S. Swamy, "Space-Time Tradeoffs on the FFT Algorithm",Brown university, Providence, R.I., August 1977.

[30] J. Subramanian, M. Bs and S. R. Murthy, "On Using Battery State for Medium Access Control in Ad hoc Wireless Networks," *Proc. Mobicom*, 2004.

[31] M. Yoshikawa, "New Directions in Mobile Communications Services," (Invited Talk), *Proc. Mobicom*, 2004.

[32] J. Flinn and M. Satyanarayanan. "Energy-aware adaptation for mobile applications." *Proc. 17th ACM Symposium on Operating Systems Principles December*, 1999.

[33] H. Zeng, C. Ellis, A. Lebeck, and A. Vahdat. "ECOSystem: Managing Energy as a First-Class Operating System Resource," *Proc. Architectural Support for Porgramming Languages and Operating Systems (ASPLOS)*, 2002.

[34] J. Lorch and A. J. Smith. "Software strategies for portable computer energy management," *IEEE Personal Communications Magazine*, 5(3):60-73, June 1998.