Towards an Energy Complexity of Computation

Alain J. Martin [1]

Department of Computer Science
California Institute of Technology
Pasadena CA 91125, USA

Abstract

Energy consumption is becoming a critical complexity parameter along with time (delay) in the design and optimization of algorithms at both the hardware and software levels. This paper proposes that a new complexity measure including energy E and time t in the form of the expression $E \times t^2$ be used as the measure of the efficiency of a computation. We prove that the metric is optimal. As an example, a new result concerning the optimal length of a pipeline is derived.

Key words: VLSI, Integrated circuits, CMOS, Energy consumption, Power consumption, Energy-delay product, Pipelining, Energy complexity

1 Introduction

In spite of the energy efficiency of CMOS, today's VLSI technology of choice, energy consumption is becoming a critical factor in the design of computing and communication systems. The energy requirements of high-end computing systems are starting to reach the limits of what public utility services can offer at a point in time. At the other end of the spectrum, the emergence of so-called information appliances—computing devices in consumer appliances—puts severe limitations on the energy that can be consumed by computations running on such devices.

Because there is an upper bound on the running time of most computations, high energy consumption is usually accompanied by high power consumption.

[1] E-mail: `alain@cs.caltech.edu`. Supported by DARPA under contract AFOSR F29601-00-K-0184

High power consumption is also becoming critical because of the added cost of cooling devices, the inconvenience of weight, heat, and short battery-life, and because of the destructive effects of high current density.

The measures to be taken at the circuit level in order to decrease the energy consumption of a *given* algorithm are well-known: avoid superfluous activity, reduce capacitance, reduce voltage. But the question remains of finding the most energy-efficient algorithm for a given problem. Can we develop an energy-complexity theory of computations similar to the existing time-complexity theory? What is the relationship between time efficiency and energy efficiency?

If the algorithm is truly sequential, the answer to the second question is simple: the most energy-efficient algorithm is also the most time-efficient since, in both cases, we want to minimize the number of transitions (elementary steps). But few algorithms are truly sequential when they are analyzed at the VLSI level, as we do here. (For example, a word of data is a collection of bits that are stored and moved in parallel.)

In a parallel algorithm, the relationship between computation time and computation energy is more complex. Let us oversimplify the problem a little. We describe a specific computation, i.e. a particular execution of an algorithm, as a partial order of elementary steps represented in an acyclic directed graph. The time complexity t is proportional to the longest path from the initial step to the final step, the so-called *critical path*. The energy complexity E of the computation is proportional to the total number of nodes in the graph. A sequential computation is the particular case where all nodes are ordered in a single chain from the initial node to the final node. So, one would think that the designer's task is to rearrange the nodes of the graph so as to minimize the length of the critical path under invariance of E. But rearranging the nodes of the graph so as to introduce concurrency (and shorten the critical path) usually increases the energy because additional nodes are required for distributing and collecting information. Hence, there is a trade-off between energy complexity and time complexity: modifying the concurrency of an algorithm may increase E and decrease t or vice-versa.

It so happens that, for a given algorithm, there is also a trade-off between E and t at the electrical level through voltage adjustment, and to a lesser extent through transistor sizing.

The different trade-offs are confusing enough that, even today, experts in the area of low-power design use an efficiency metric, the energy-delay product, that is just plain wrong! The main purpose of this paper is to clarify the issue by introducing a metric that makes it possible to compare two algorithms for both energy and delay without the obfuscation of physical parameter (voltage) adjustment. At the same time, the metric formalizes the trade-off between

energy and delay within the same algorithm.

2 Energy and Delay in VLSI Computations

We consider only irreversible computations, i.e., computations in which assigning a value to a variable destroys the previous value of the variable; and we base our physical model on an implementation of elementary operators (or gates) called *restoring logic*. In a restoring logic implementation, the energy required to switch the value of the output of a logic gate is supplied independently of the energy required to switch the value of its inputs, and is therefore supplied by an independent power supply [1]. Although restoring logic is not optimal in terms of energy consumption (some of the input energy could be reused as output energy), it has the advantage of providing modularity to the model by isolating the energy consumption of each stage of logic. Restoring logic also increases the electrical robustness of circuits.

The current VLSI technology of choice for digital computation, CMOS, is ideal for implementing restoring logic. We shall therefore derive a general formula for energy and delay based on CMOS implementation. We believe that the formulation of energy and time and the method we are going to present can be easily adapted to other technologies.

We have proved that any algorithm can be expressed at the physical level of implementation as a collection of *production rules*[3]. A production rule is a restricted form of Dijkstra's guarded commands. It is of the form: $B \rightarrow t$, where B is a Boolean expression (the guard of the production rule), and t is a single assignment of the value true or false to a Boolean variable. Such an assignment is called a *transition*. A logic gate is the physical implementation of the pair of production rules that set and reset a given variable, for instance the pair:

$Bu \rightarrow z\uparrow$
$Bd \rightarrow z\downarrow$

Let $E_{z\uparrow}$ and $E_{z\downarrow}$ be the energy needed to fire the first and second guarded commands respectively. In a CMOS implementation, the energy needed to fire a production rule non vacuously is the energy needed to charge or discharge the capacitor Cz associated with the physical representation of z. (The energy spent in short-circuit current and leakage current is assumed to be negligible.)

$$E_{z\uparrow} = \frac{Cz \times V^2}{2}, \tag{1}$$

where V is the power-supply voltage, and $E_{z\uparrow} = E_{z\downarrow}$.

The delay $t_{z\uparrow}$ required to fire the first production rule non vacuously is the ratio of the final electrical charge Qz on Cz over the current i needed to charge Cz: $t_{z\uparrow} = \frac{Qz}{i}$, with $Qz = Cz \times V$. This current is the current flowing in the transistor network connecting the constant power supply to z when and only when Bu holds. And similarly for the delay $t_{z\downarrow}$.

It is difficult—if not impossible—to express the current i accurately as a closed formula for the different operating modes of the CMOS transistor. Fortunately, the systems we have designed exhibit a behavior that is much more uniform than the complex models for the CMOS transistor allowed us to believe. This behavior is clearly dominated by the *saturation* regime of the transistor in which the current i is of the form $i = Ku \times V^2$, where Ku is a constant that depends on the structure of the guard Bu. With this value of i, the expression for the delay becomes:

$$t_{z\uparrow} = \frac{Cz}{Ku \times V} \tag{2}$$

As programmers, how do we use those results? Equation 1 is remarkably simple: The energy spent to fire production rule $Bu \rightarrow z\uparrow$ depends only on the size of the output capacitance Cz, i.e., in programming terms, it depends only on the fanout of the operator. In particular, the energy is independent of the guard Bu.

Equation 2 is more complicated: The delay depends on both the output capacitance Cz, and the guard Bu through the term Ku. In general, the delay grows with the square of the number of transistors in the longest transistor chain, i.e., the square of the number of conjuncts in the largest term of Bu in disjunctive normal form.

We can already appreciate the trade-offs between energy and delay that the programmer has to deal with. If Bu is complicated, decomposing the production rule in a collection of production rules with smaller guards will reduce the total delay but increase the total energy. But it is clear from equations 1 and 2 that there is also a trade-off between energy and delay through voltage adjustment, and that is a trade-off the programmer cannot control since the voltage is not a programming concept. How, then, can programmers compare two algorithms for energy and delay if the two quantities vary in opposite directions with the voltage?

3 Comparing Algorithms for Energy and Delay

Given two algorithms A and B, with energy and delay (E_A, t_A) and (E_B, t_B), respectively, how do we compare them for energy and delay?

3.1 Why $E \times t$ Is Not the Right Metric

The energy-delay product $E \times t$ is often used to compare designs but is unfortunately not an acceptable metric.

Assume $E_A = 2 \times E_B$ and $t_A = \frac{t_B}{2}$. Then, according to the $E \times t$ metric, A and B are equivalent. But let's reduce the voltage of A by half. Let (E'_A, t'_A) be the new values of energy and delay for A. Given the dependence of energy and delay on voltage as specified by equations 1 and 2, we have:

$$E'_A = \frac{E_A}{4}$$

$$t'_A = 2 \times t_A,$$

which gives:

$$E'_A = \frac{E_B}{2}$$

$$t'_A = t_B.$$

Hence, A has the same delay as B but only half the energy, and therefore A is a better computation than B contrary to what the $E \times t$ metric indicates.

3.2 The Θ Metric

We ignore the lower and upper bounds imposed on the voltage by the technology and we assume that we can always trade E and t against each other through voltage adjustment.

Suppose there exists a function $\Theta(E, t)$ with the properties:

(1) Θ is monotonically increasing in E and t,
(2) Θ is independent of V.

Theorem 1 *Given two computations A and B with Θ's, Θ_A and Θ_B, respectively.*

- *If $\Theta_A < \Theta_B$ then A is more efficient than B with respect to energy or delay.*
- *If $\Theta_A = \Theta_B$ then A is equivalent to B with respect to energy or delay.*

Proof: Because of Property (2), we can equalize either the energy or the delay of the two computations under invariance of the Θ's. We arbitrarily choose to equalize the delays: $t_A = t_B$.

We can now compare the two computations, thanks to Property (1):

- $(\Theta_A < \Theta_B) \Rightarrow (E_A < E_B)$ i.e. A is better than B,
- $(\Theta_A = \Theta_B) \Rightarrow (E_A = E_B)$ i.e. A is equivalent to B.

Hence, for any chosen delay t, A is more energy-efficient than B. For any chosen energy E, A is more time-efficient than B. □

3.3 The $E \times t^2$ Metric

Any expression in E and t that is monotonically increasing in E and t and that is invariant of V can be used as complexity metric Θ. In CMOS, given the formulas of equations 1 and 2 for energy and delay, we choose:

$$\Theta \stackrel{\text{def}}{=} E \times t^2.$$

(This complexity measure was introduced for the design of an asynchronous MIPS R3000 microprocessor [2].) If we now return to the example we used at the beginning of this section, we compare the two computations A and B by comparing their Θ's:

$$\begin{aligned}
\Theta_A &= E_A \times t_A^2 \\
\Theta_A &= (2 \times E_B) \times \left(\frac{t_B}{2}\right)^2 \\
\Theta_A &= \frac{\Theta_B}{2}.
\end{aligned}$$

Hence, without reference to voltage, we can conclude that A is twice as efficient as B in terms of $E \times t^2$. For equal delays, $E_A = \frac{E_B}{2}$. For equal energies, $t_A = \frac{t_B}{\sqrt{2}}$.

3.4 $E \times t^2$ Measurements

How constant is $E \times t^2$ in reality? As we already mentioned, there are several operating modes for the CMOS transistor, each with a very different relation between current and voltage. In particular, at high electric field, the electron velocity saturates and becomes constant; the delay becomes independent of the voltage, and $E \times t^2$ becomes quadratic in the voltage. The $E \times t^2$ measure will be criticized on the ground that velocity saturation invalidates it.

But we have strong experimental evidence to the contrary. The graph of Figure 1 shows the measured Et^2 for a 2-million-transistor asynchronous MIPS R3000 microprocessor designed at Caltech between 1996 and 1998. It was fabricated in CMOS 0.6μm technology and was found to be entirely functional on first silicon[2]. (All measurements on other fabricated chips give similar results.)

The exponential behavior below 1.5 V is typical of subthreshold voltage behavior which is outside the range of operating voltages for digital designs. The positive slope from 3 V upward shows the effect of velocity saturation.

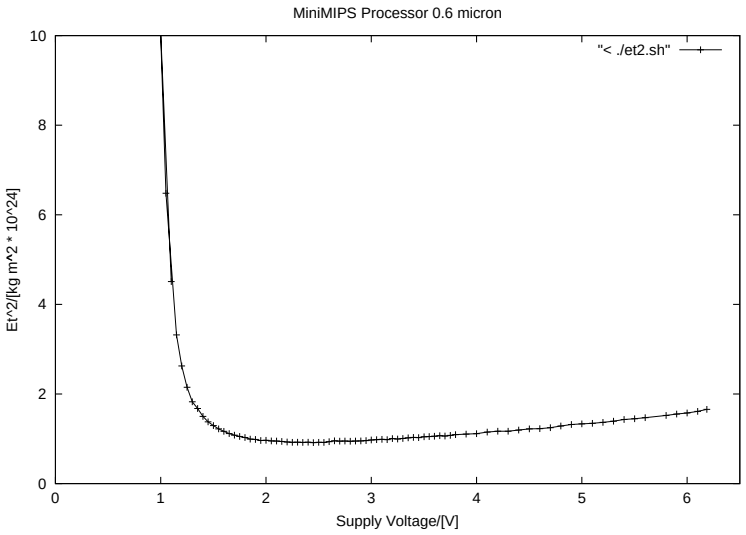

Fig. 1. Measured Et^2 for a 2-million-transistor asynchronous microprocessor

4 Example: The Θ-Complexity of Pipelining

Consider a sequential process S computing the value of a function $F(x)$. Process S communicates with its environment through an input channel L and an output channel R: For each value x received on L, S outputs the value of

$F(x)$ on R. In CHP notation, the program for S is:

$$S : *[L?x \ ; \ R!F(x)].$$

(Notation: $*[...]$ denotes the infinite repetition of the program part enclosed within the square brackets. $L?x$ denotes an input (receive) command on channel L: the value received on L is stored in x. $R!F(x)$ denotes an output (send) command on channel R: the value sent on R is the value of $F(x)$ for the current value of x. The semicolon is the sequential composition.)

Pipelining is possible when F is the composition of m functions f_i:

$$F = f_{m-1} \circ \ldots \circ f_1 \circ f_0.$$

Pipelining is the program transformation that replaces the single process S with a chain P of n processes P_0, P_1,.., P_{n-1} called the "stages" of the pipeline. The chain is denoted as the parallel composition of the n processes P_j:

$$P : (\|j : 0..n-1 : P_j)$$

where:

$$P_j : *[C_j?x; \ C_{j+1}!g_j(x)].$$

Two adjacent processes P_j and P_{j+1} in the chain are connected by channel C_{j+1}, which serves as the output channel of P_j and the input channel of P_{j+1}. The leftmost input channel C_0 is the input channel of the whole chain and is therefore identified with L. The rightmost output channel C_n is the output channel of the whole chain and is therefore identified with R.

The g's are chosen such that $F = g_{n-1} \circ \ldots \circ g_1 \circ g_0$, with $0 < n \leq m$. Observe that the functions g_j are not necessarily identical to the functions f_i, since one issue in optimizing the pipeline is the choice of the granularity of the functions the processes P_j compute. The designer may choose to agregate several consecutive f's into one g. Assigning the computation of each f_j to a stage P_j gives the maximal pipelining and the lowest cycle time, but not necessarily the most efficient pipeline in terms of $E \times t^2$.

4.1 The Θ-Complexity of S

We first determine the energy $E0$ and cycle time $t0$ for the computation of one $F(x)$ by process S. In evaluating the time efficiency of a pipelined computa-

tion, we may be interested in two parameters: the latency and the cycle time. The latency is the delay between the input of the ith value x_i of parameter x by process S and the output of $F(x_i)$ by S, averaged over all values of i. The cycle time is the delay between the inputs of two consecutive values x_i and x_{i+1} of parameter x by process S, again averaged over all values of i. Here, we are interested in the cycle time and the energy consumed by one cycle.

There are two parts to the activity of a cycle: the computation of F proper and the communication overhead. Let E be the energy and t the time to compute F. For the communication overhead, we make the worst-case assumption that it depends on the size of the parameter x and is therefore proportional to the cost of computing F. For the sake of simplicity, we assume that the constant of proportionality k is the same for energy and delay. Hence, the communication overhead energy is $k \times E$ and the communication overhead delay is $k \times t$. Putting the pieces together, we get:

$$E0 = E + k \times E,$$

$$t0 = t + k \times t,$$

$$\Theta 0 = E \times t^2 \times (1 + k)^3.$$

4.2 Energy and Cycle Time of the Pipeline

We choose the "depth" of the pipeline, n, and the functions g so as to minimize the Θ of the pipeline in terms of the energy and cycle time for computing one $F(x)$.

The cycle time of the pipeline is the cycle time of the slowest stage. Hence, the functions g are chosen such that all stages have the same cycle time. For all j:

$$t_j = \frac{t}{n} + k \times t.$$

But the stages don't need to have the same energy:

$$E_j = F_j + k \times E,$$

with $(\sum j : 0..n-1 : F_j) = E$. Observe that the communication overhead is the same for the pipelined and non-pipelined cases since the communication actions are the same.

We can now compute the energy Ep and the cycle time tp for the computation of one $F(x)$:

$$Ep = (\sum j : 0..n-1 : E_j)$$

$$tp = t_j,$$

which gives:

$$Ep = E + k \times n \times E,$$

$$tp = \frac{t}{n} + k \times t.$$

4.3 Optimal Θ for the Pipeline

Let Θp be the Θ of the pipeline. By definition, $\Theta p = Ep \times tp^2$, i.e.:

$$\Theta p = E \times (1 + k \times n) \times t^2 \times (\frac{1}{n} + k)^2,$$

$$\Theta p = E \times t^2 \times \frac{(1 + k \times n)^3}{n^2}.$$

We can express the improvement of the pipeline compared to the non-pipelined case as the ratio of the two Θ's:

$$\frac{\Theta p}{\Theta 0} = \frac{1}{n^2} \times \frac{(1 + k \times n)^3}{(1 + k)^3}.$$

The ideal case $k = 0$ (no overhead) gives an improvement:

$$\frac{\Theta p}{\Theta 0} = \frac{1}{n^2}$$

with $Ep = E$ and $tp = \frac{t}{n}$. Although it looks like we have gained nothing in energy, in fact we can save up to a factor n^2 in energy if we equalize the cycle time to that of the non-pipelined case through voltage scaling.

For $k > 0$, the optimal improvement is achieved for

$$\frac{d}{dn}\left(\frac{\Theta p}{\Theta 0}\right) = 0$$

i.e. for $n \times k = 2$.

In the optimal case:

$$Ep = 3 \times E$$

$$tp = \frac{3}{2} \times k \times t$$

from which we derive the following theorem on the optimal depth of a pipeline.

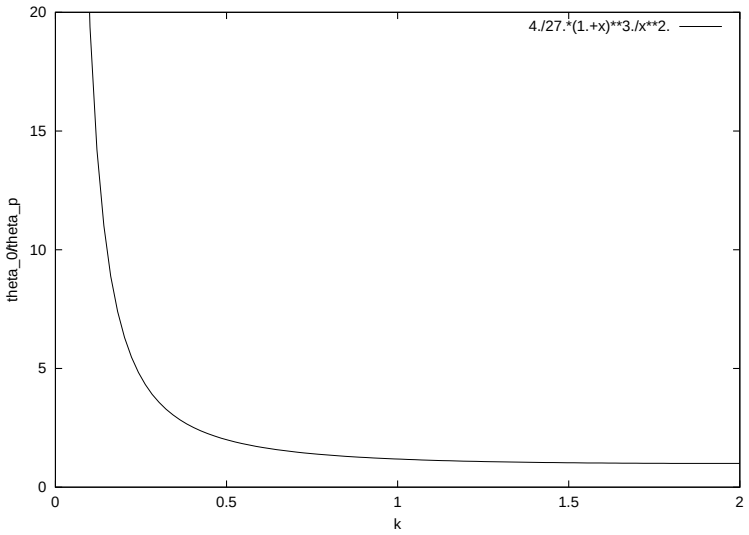

Fig. 2. Θ improvement as a function of the overhead ratio k

Theorem 2 *Given a pipelined computation of a function F in which the overhead energy and overhead delay due to communication are proportional to the computation energy and delay, respectively, the Θ-optimal pipeline requires an energy that is 3 times the energy required for computing F. It has a cycle time that is $\frac{3}{2}$ the overhead cycle-time.*

4.4 Sensitivity of Θ to Communication Overhead

Let us compute the optimal pipeline improvement as a function of the communication overhead ratio k, $(n = \frac{2}{k})$. We get:

$$\frac{\Theta 0}{\Theta p} = \frac{4}{27} \times \frac{(1+k)^3}{k^2}.$$

The results for various values of k between 0.1 and 2 are shown in Figure 2.

The figure shows that the pipeline is very sensitive to the communication overhead. The results are surprising: For an overhead ratio of 1, which is not at all unusual, the pipeline offers practically no gain.

5 Conclusion

Future theories of algorithmic complexity will have to include both energy and time in their efficiency metric. This paper suggests a method for reasoning about energy and time complexity and for formalizing the trade-off between energy and time in algorithm design. Although we have strong theoretical and experimental evidence that the $E \times t^2$ measure is the most appropriate in today's technology, it may have to be adjusted to reflect future technological changes. It will always be possible to find a Θ that satisfies the requirements proposed in this paper.

The result concerning the optimal pipeline is new and surprising. Of course, the model for the communication overhead is probably oversimplified and the calculations may have to be refined. But at this point, the general method is more important than the results of this particular calculation.

As evidenced by the pipeline example, one challenge is that of reasoning about energy at the algorithmic level and yet retaining enough relevant information about the physical implementation. The best work in this area so far is José Tierno's energy complexity model, which relates the energy of CHP constructs to information-theoretic entropy[5], [4].

Acknowledgment. I am indebted to Mika Nyström for comments and suggestions in the preparation of this paper. The idea to use $E \times t^2$ as a complexity measure was born out of many animated discussions in our research group, starting in 1995 when we embarked on the design of the MIPS. The most vocal participants in the discussions were José Tierno, Peter Hofstee, Andrew Lines, Mika Nijström, Rajit Manohar, and Paul Penzes.

The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, and monitored by the Air Force under contract F29601-00-K-0184.

References

[1] Carver A. Mead and Lynn Conway. *Introduction to VLSI Systems*, Addison-Wesley, Reading MA, 1980.

[2] Alain J. Martin, Andrew Lines, Rajit Manohar, Mika Nyström, Paul Penzes, Robert Southworth, Uri Cummings and Tak Kwan Lee. The Design of an Asynchronous MIPS R3000 Microprocessor. *Proceedings of the 17th Conference on Advanced Research in VLSI*, IEEE Computer Society Press, 164-181, 1997.

[3] Alain J. Martin. Synthesis of Asynchronous VLSI Circuits. *Formal Methods for VLSI Design*, ed. J. Staunstrup, North-Holland, 1990.

[4] José A. Tierno and Alain J. Martin. Low-Energy Asynchronous Memory Design. *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, IEEE Computer Society Press, 176-185, 1994.

[5] José A. Tierno. *An Energy-Complexity Model for VLSI Computations.* PhD Thesis. California Institute of Technology, 1995.