Computation with Energy-Time Trade-Offs: Models, Algorithms and Lower-Bounds

Brad D. Bingham Mark R. Greenstreet
University of British Columbia, Computer Science Technical Report TR-2008-03

Abstract –**Power consumption has become one of the most critical concerns for processor design. This motivates designing algorithms for minimum execution time subject to energy constraints. We propose simple models for analysing algorithms that reflect the energy-time trade-offs of CMOS circuits. Using these models, we derive lower bounds for the energy-constrained execution time of sorting, addition and multiplication, and we present algorithms that meet these bounds. We show that minimizing time under energy constraints is not the same as minimizing operation count or computation depth.**

1. Introduction

Power consumption is now widely recognized as the most critical issue for computer design. It affects portable applications where battery life is critical as well as desktop and server computers where performance is determined largely by the the limits of heat sinks, fans, and air conditioning [1], [2]. Fortunately, energy and time are fungible for computation: if more time is allotted for an operation, the operation can be performed using less energy. For example with CMOS logic circuits [3], [4], the delay of a logic gate, t, is roughly proportional to the inverse of the operating voltage, whereas the energy per transition, e, is proportional to the square of the voltage. This leads to a trade-off where et^2 is invariant under changes in the operating voltage [5].

The energy-time trade-offs afforded by CMOS technology have been studied intensively by the real-time systems community since the seminal paper by Yao *et al.* [6]. This has led to many papers that examine energy trade-offs for scheduling problems for uniprocessors, multiprocessors, with or without precedence constraints, etc. (e.g. [7]–[13]). In all of these papers, the set of tasks to be performed is taken as a given. We are not aware of any prior work that considers algorithm design with energy-time trade-offs.

The opportunity to exchange time for energy creates a compelling incentive to exploit parallelism: the individual operations in a parallel algorithm may be performed more slowly, thus saving energy, while completing the entire task in the same or less time than a sequential version. In fact, a parallel algorithm may perform more operations than its sequential counterpart while using less time *and* less energy. This report makes three principle contributions:

1) We present a simple computation model that accounts for energy, time and communication.
2) We use our model to derive lower bounds for sorting, binary addition and multiplication.
3) We present algorithms that achieve these lower bounds to within constant factors.

We believe that the problems we have chosen, sorting addition and multiplication, are ideal for initiating a study of energy-time trade-offs for two reasons. First, they are well-understood problems that represent common computational patterns common to many algorithms. For example, addition embodies the *map-reduce* communication pattern described in [14]; multiplication is a canonical example of convolution; sorting is an example of performing a general permutation of data values. Second, it is possible to formulate a model at the bit-level and establish upper and lower bounds for each of these problems. While we expect that a higher-level model may be useful for studying further problems, our bit-level formulation ensures that were not neglecting any critical details. Conversely, establishing the complexity of basic operations in our detailed model provides a robust foundation for formulating a more abstract model, for example, one with word-level operations as primitives.

We focus on asymptotic (i.e. big-O) results to avoid introducing constants that would make our analysis specific to a particular fabrication technology. We present some "surprising" results for energy-constrained, minimal time computation. For example, if the inputs of a sorting network are required to lie along a line, then "slow" algorithms such as bubble-sort and "fast" algorithms such as odd-even merge-sort have the same asymptotic energy-time complexity. Our construction for an optimal adder shows that broadcasting a bit to all $O(d^2)$ mesh locations within distance d of a source uses the same energy and time (to within a constant factor) as sending the bit to a single location distance d away.

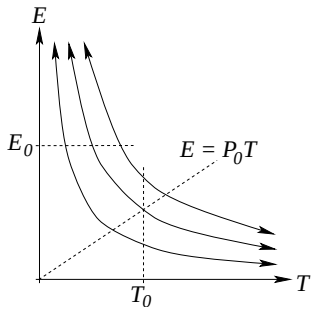

Fig. 1. Energy-Time Trade-Offs

The algorithms that we present for sorting and multiplication are energy-time optimal, yet they minimize neither operation count nor computation depth.

2. Modelling Energy-Time Trade-Offs

Processors with more than one-billion transistors are now in production [15], and even denser designs are expected in future fabrication generations. We model this large device count by assuming the availability of unlimited parallelism. However, communication overhead limits the exploitation of parallelism, and integrated circuits are essentially planar. Thus, our models include costs for communication that increase with distance assuming a planar embedding of the computation. Finally, computation or communication operations can be performed faster by using more energy, or slower by using less.

We note that there are several closely related formulations of optimal algorithm design given energy constraints including: minimizing execution time subject to an energy constraint, minimizing execution time subject to a power constraint, and minimizing energy subject to an execution time constraint. In our model, an algorithm gives rise to an energy-time trade-off of the form $ET^\alpha = f(N)$ where E is the energy for the computation, T is the time, N is the input size. Figure 1 illustrates this trade-off with the three curves representing three different algorithms. Minimizing time subject to an energy constraint corresponds to finding the curve with the leftmost intersection with the $E = E_0$ line. Likewise, the $E = P_0T$ lines represents a power constraint and the $T = T_0$ line represents a time constraint. In all three cases, the optimal algorithm minimizes $f(N)$.

The remainder of this section presents a simplified model for deriving lower bounds, and a detailed model for analysing specific algorithms to establish upper bounds. These differ in how they reflect the two-dimensional nature of integrated circuits. The assumptions and definitions that are common to both models are presented first. As this common model does not include a metric for communication distance, we call it the "metricless" model.

2.1. The Metricless Model

Computations are modelled as being performed by an ensemble of communicating *processing elements* (PEs). A PE has $O(1)$ input bits, output bits and internal state bits. PEs can represent common hardware components such as flip-flops, constant size combinational circuits and unit-length wires. We distinguish between PEs that have state bits and/or perform some meaningful computation, called *computation* PEs, and those which simply pass input values to outputs in a fixed manner, called *wire* PEs.

We assume that the computation being performed can be represented by an acyclic *task graph* [16, p. 50], $(\mathcal{V}, \mathcal{A})$ where vertices represent operations and arcs represent communication. Vertices $v \in \mathcal{V}$ with $\deg^+(v) = 0$ are called *inputs*; those with $\deg^-(v) = 0$ are called *outputs*, where $\deg^+(v)$ and $\deg^-(v)$ are the in-degree and out-degree of v, respectively. If v_i and v_j are vertices of $(\mathcal{V}, \mathcal{A})$, we write $v_i \prec v_j$ iff there is a directed path from v_i to v_j. Let function P map each vertex $v \in \mathcal{V}$ to a PE, $P(v)$, with the interpretation that operation v is performed by PE $P(v)$. The same PE can perform many different tasks as long as these operations are performed at different times. In particular, if v_i and v_j map to the same PE (i.e. $P(v_i) = P(v_j)$), then we require that either $v_i \prec v_j$ or $v_j \prec v_i$ must hold. Thus, P can be many-to-one. However, each input vertex must map to a different PE and likewise for output vertices. This restriction prevents implementations that hide the cost of data storage in their environments.

Let $\tau : \mathcal{V} \to \mathbb{R}^+$ map vertices v to their *duration* $\tau(v)$, the amount of time that elapses from when the last predecessor of v completes its operation until v completes. A particular PE, p, may have different durations for distinct operations in $P^{-1}(p)$. We refer to the function τ as a *schedule* for the computation.

We extend τ to apply to paths: if $w = v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ is a path in $(\mathcal{V}, \mathcal{A})$, then we define

$$\tau(w) \quad = \quad \sum_{j=1}^{k} \tau(v_{i_j}) \; .$$

We can now define the total time for a computation:

$$\mathcal{T}(\mathcal{V}, \mathcal{A}, \tau) \quad = \quad \max_{w \in \text{paths of } (\mathcal{V}, \mathcal{A})} \tau(w) \; . \tag{1}$$

In other words, $\mathcal{T}(\mathcal{V}, \mathcal{A}, \tau)$ is the last time at which an operation completes under schedule τ.

Energy-time trade-offs are incorporated by assuming a constant $\alpha > 0$, such that the energy to perform operation v is given by $e(v) = \tau(v)^{-\alpha}$, i.e., $e(v)\tau(v)^\alpha = 1$. For

simplicity, α is the same for all operations. The total energy for a computation is

$$\mathcal{E}(\mathcal{V},\tau) \quad = \quad \sum_{v\in\mathcal{V}} \tau(v)^{-\alpha} \ . \tag{2}$$

Given a task graph, $(\mathcal{V},\mathcal{A})$, we find a schedule τ that minimizes $\mathcal{E}(\mathcal{V},\tau)\mathcal{T}(\mathcal{V},\mathcal{A},\tau)^\alpha$ (or simply ET^α). The energy cost, time cost, or simply "cost" refers to the total energy E, total time T, and ET^α, respectively, for PE(s) with a schedule clear from context.

The minimum ET^α for any given task graph can be achieved for any time T allotted to complete the computation, as long as $T > 0$.

Lemma 2.1: [Scaling] Let τ_0 be a schedule for a task graph, $(\mathcal{V},\mathcal{A})$, and let $T_0 = \mathcal{T}(\mathcal{V},\mathcal{A},\tau_0)$. Let $T_1 > 0$ and let $\tau_1(v) = \frac{T_1}{T_0}\tau_0(v), \forall v \in \mathcal{V}$. Then,

$$\mathcal{E}(\mathcal{V},\tau_1)\mathcal{T}(\mathcal{V},\mathcal{A},\tau_1)^\alpha \quad = \quad \mathcal{E}(\mathcal{V},\tau_0)\mathcal{T}(\mathcal{V},\mathcal{A},\tau_0)^\alpha,$$

and thus τ_0 is optimal iff τ_1 is optimal.

Proof: The claims follow directly from the definitions of $\mathcal{E}$ and $\mathcal{T}$. ∎

Later in this report, we will often use Lemma 2.1 to obtain upper bounds for the ET^α cost of a computation by generalizing from a schedule for which the ET^α cost is straightforward to derive.

2.2. The Grid Model

The grid model refines the metricless model by requiring specific placements for PEs. This model is more restrictive than the gridless model presented in Section 2.3. We use the grid model to establish upper bounds for the complexities of problems and algorithms.

In the grid model, each PE occupies a unit square in the plane. No two PEs can occupy the same square. Thus, a PE can be identified by the coordinates of its lower-left corner, $(i,j) \in \mathbb{Z}^2$. If p_1 is a PE at location (i_1,j_1), and p_2 is a PE at (i_2,j_2), then p_1 and p_2 can connect inputs to outputs of each other iff $\|(i_1,j_1)-(i_2,j_2)\|_1 = 1$. Communication over longer distances then can be achieved with "wires" which we model with chains of wire PEs. The specification of PEs in the grid model corresponding to a task graph is called a *grid model implementation*, or simply *implementation*. Each operation of each PE is performed using energy e and time t with $et^\alpha = 1$.

Define *transmission* of $O(1)$ bits of information from PE p_1 to PE p_2 as the change of some input(s) to p_2 as a consequence of an earlier change on some output(s) of p_1. Let p_1 and p_2 be distinct PEs at locations (i_1,j_1) and (i_2,j_2). Let $d = \|(i_1,j_1)-(i_2,j_2)\|_1$. If p_1 transmits information to p_2, then this information traverses *at least* $d-1$ intermediate PEs. If p_1 and each of these intermediate PEs takes unit time to transmit the information to the next PE, then the transmission uses d units of time and d units of energy, for a total[1] ET^α of $d^{\alpha+1}$. This is optimal, as shown by the following lemma:

Lemma 2.2: [Wires] Let p_1 and p_2 be two PEs in an implementation where p_1 and p_2 are separated by Manhattan distance d_{12}. Let E_{12} and T_{12} be the total energy and time to transmit $O(1)$ bits of information from p_1 to p_2. Then $E_{12}T_{12}^\alpha \geq d_{12}^{\alpha+1}$.

Proof: (Sketch) Consider the chain of PEs through which the information is transferred and the amount of time for each transfer. ET^α is convex with respect to this vector of transfer times, and is minimized when all of these times are the same. The conclusion follows. ∎

2.3. The Gridless Model

While the geometric detail of the grid model ensures the existence of a planar implementation, it is overly restrictive for lower-bound arguments. In particular, it requires PEs to be square and for PEs to be arranged on a rectilinear grid. The gridless model removes these restrictions.

Computation PEs and wire PEs are distinguished in the gridless model. The task graphs for algorithms in the gridless model are bipartite where all arcs are between $\mathcal{X}$, the set of computation PEs, and $\mathcal{W}$ the set of wire PEs.

Computation PEs may assume arbitrary, convex shapes, with the restriction of a constant area circumcircle. We assume a distance metric (e.g. Euclidean distance or Manhattan distance) for computation PEs: let $\mu(p_1,p_2)$ be the Euclidean distance between the center points of some fixed inscribed circle of p_1 and of p_2. Because μ is a distance metric, the triangle inequality applies. We assume that computations are implemented in the plane, and model this assumption by requiring that any PE has at most d^2 other PEs within distance d of itself.

A wire PE has one input and one output. Let $P(w)$ be a wire PE and let $P(v_1)$ and $P(v_2)$ be computation PEs such that (v_1,w) and (w,v_2) are arcs of the task graph. Let $d = \mu(P(v_1),P(v_2))$. PE $P(w)$ may be thought of as conceptually equivalent to a chain of d wire PEs of length 1. By analogy with Lemma 2.2, each operation of w requires time t and energy e with $et^\alpha = d^{\alpha+1}$. Formalizing these observations yields the axioms of our gridless model:

1) The computation is expressed by a bipartite task graph, $((\mathcal{X}\cup\mathcal{W}),\mathcal{A})$, where $\mathcal{X}\cap\mathcal{W} = \emptyset$. Operations in $\mathcal{X}$ are implemented by computation PEs, and operations in $\mathcal{W}$ are implemented by wire PEs.
2) Each operation of a computation PE takes energy e and time t with $et^\alpha = 1$.

[1] In this context, E and T respectively denote the total energy and time for only this transmission.

3) Let $P(\mathcal{X})$ be the set of all computation PEs. There is a distance metric, $\mu : P(\mathcal{X}) \times P(\mathcal{X}) \to \mathbb{R}$, such that for all computation PEs, $p \in P(\mathcal{X})$,

$$\left|\{q \in P(\mathcal{X}) : (q \neq p) \wedge \mu(q,p) \leq d\}\right| \leq d^2.$$

4) Each wire PE has one input and one output.
5) If $v_1, v_2 \in \mathcal{X}$, and $w \in \mathcal{W}$ with $(v_1, w), (w, v_2) \in \mathcal{A}$, and $\mu(P(v_1), P(v_2)) = d$, then an operation of w takes energy e and time t with $et^\alpha = d^{\alpha+1}$.

Because the gridless model ignores issues of wiring congestion, it admits descriptions of computations that cannot be implemented in the plane. It is straightforward to show that the axioms for our gridless model are less restrictive than the constraints of the grid model. Thus, lower-bounds from the gridless model hold (to within constant factors) in the grid model, and upper-bounds from the grid model hold (to within constant factors) in the gridless model as well.

2.4. Proof Framework and Definitions

In this report, we consider solving specific *problems*, such as sorting. A problem is simply defined by its input to output mapping, and the type of the inputs and outputs, such as bits or words. We identify an *algorithm* that solves a problem by its associated equivalence class of task graphs. The equivalence class arises from task graphs with identical computation vertices, but flexibility of the length of all chains of communication vertices. This can be viewed as a many-to-one relation from the input size N to some task graph with N input vertices that produces the appropriate partial ordering of operations associated with the algorithm. We use the function P to define PEs in the grid model to implement the task graph. Finally, the implementation is scheduled by defining the previously mentioned τ function. Note that some task graphs containing high degree vertices may not be realizable under the mapping P to a valid grid model implementation. Thus we see a hierarchy of entities, where problems are the most general followed by algorithms, task graphs, grid model implementations and then schedules, as shown in Figure 2.

Establishing an upper bound at one level of the hierarchy establishes the bound at all levels above it as well. Likewise, a lower bound at one level applies to all levels below it. Thus, we typically prove upper bounds by giving a schedule for a grid model implementation. This establishes an upper bound from the problem. Conversely, we obtain lower bounds at the problem level, assuming a gridless model implementation. Because the gridless model is less restrictive than the grid model, this establishes a lower bound for grid model implementations as well.

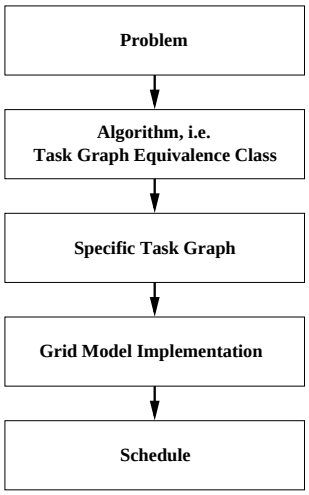

Fig. 2. Hierarchy of entities

3. Related Work

The trade-offs between voltage, energy and delay have been understood at least since Hoeneisen and Mead's paper in 1972 [3]. More recently, various researchers have promoted the use of the ET (e.g. [17]) and ET^2 (e.g. [1], [5]) metrics for modeling energy-time trade-offs in CMOS logic circuits. Dynamic voltage and frequency scaling is standard practice in the design of CPUs for portable [18] and embedded [19] computers and is now being applied to processors for desktop and server processors as well [20], [21]. Processor chips with hundreds of heterogeneous CPUs running at multiple and dynamically changing clock frequencies appear likely in the next few years [14]. Even current CPUs include *thermal throttling* to shut-off or slow-down all or parts of the chip if the die gets to hot. In principle, one can write algorithms that run slower than expected by triggering these mechanisms. Recently, Williams *et al.* [22] examined energy-performance trade-offs by comparing the IBM Cell processor [23] with several other processors. They reported that the architectural features of the Cell, especially its support for on-chip, parallel computation, provided advantages of factors of 20-50 by a measure of GFlops/Joule.

To the best of our knowledge, no one has examined the implications of energy-time trade-offs for algorithm design. Martin's work [5] comes closest when he examines an "energy complexity of computation." His treatment considers the trade-offs for various forms of parallel and sequential decompositions of computations. While Martin's work provides a clear motivation for considering the energy optimal implementations, he does not examine the fundamental energy-time limits of particular computational tasks (such as sorting or addition). These limits are the topic of this report.

Although we are unaware of work for energy-time tradeoffs for algorithms, the opportunity to exploit such tradeoffs for scheduling has attracted significant attention starting with Weiser's 1994 paper [24]. A formal model of this problem soon followed by Yao *et al.* [6] who gave an optimal poly-time offline algorithm and a constant factor competitive online algorithm. The model assumes $power = speed^{\beta}$ for some $\beta \geq 1$ which is equivalent to assuming that ET^{α} is constant with $\alpha = \beta - 1 \geq 0$ as in our model. Most of the subsequent research has focused on the single processor case. For example, Bansal *et al.* [9] give a new online algorithm for the model of Yao *et al.* that reduces competitive ratio for some values of β; they also adapt the model to account for temperature constraints. Chen *et al.* [13] offer a different model where jobs are precedence-constrained and the execution speed is chosen from a discrete set of values. Recently, some papers have turned attention to multiprocessor scheduling. These include Albers *et al.* [8] which extends the model of Yao *et al.* to a multiprocessor setting where job migration is forbidden, and Chen *et al.* [12] who study scheduling of jobs with a common deadline and where migration between processors is allowed. While the scheduling problem for multiprocessors seems suited for modern application, work in energy-aware scheduling has generally ignored the energy costs of communication. In contrast, our models account for the energy required for communication, and it is these costs that provide a limit on the optimal degree of parallelism for a given problem.

The techniques that we use to analyse algorithms with an ET^{α} cost metric are reminiscent of earlier work for deriving AT^2 bounds for algorithms where A denotes the area required for a VLSI implementation that completes the computation using time T (e.g. [25]–[29]). While some researchers included energy costs within the AT^2 model, they assumed fixed operating voltages and thus equated energy with capacitance (wire length). The AT^2 model did not include the possibility of trading energy and time through voltage scaling [3], [5], transistor sizing [30] and other techniques (see [1], [17]) that are subsumed by our model. Like the AT^2 work, our arguments are often based on communication. Whereas AT^2 results are typically based on arguments of cross-sectional bandwidth, our arguments tend to be based on the total *distance* that data must be transmitted.

In some special but useful cases, it is possible to convert an $AT^{\alpha+1}$ upper bound to an upper bound for our ET^{α} model. An algorithm uses *nearest neighbor communication* if its organization can clearly be abstracted into PEs that satisfy the requirement of our grid model. In particular, PEs p_1 and p_2 can only communicate if $\|(i_1, j_1) - (i_2, j_2)\|_1 = 1$, where (i_1, i_2) are the coordinates in a mesh for the lower left corner of p_1 and likewise for p_2.

Lemma 3.1: [Area-Time Complexity] If an algorithm uses only nearest neighbour communication and has an area-time complexity $AT^{\alpha+1} \in O(f(N))$ then $ET^{\alpha} \in O(f(N))$.

Proof: Assign unit time per operation. Details omitted to save space. ■

Lemma 3.1 shows that we can use known AT^3 results to establish ET^2 bounds. While there is an absence of explicit AT^3 results in the literature, many AT^2 arguments are robust to changes in the time exponent and establish the AT^3 bounds that we need.

4. Sorting

A sorting problem consists of N input values with an associated ordering relation. These values are initially placed in an arbitrary permutation in N predetermined input PEs and output PEs. At completion, the input values must be stored in the output locations in ascending order of the values for some pre-determined ordering of the output locations.

In Section 4.1, we consider algorithms based on sorting networks [31, chap. 5.3.4]. Our direct implementations place PEs according to the locations of the comparators in typical drawings of these sorting networks and the input and output location are each arranged along linear arrays of PEs at the left and right edges. Section 4.2 examines 2D implementations where the inputs and outputs may be placed at arbitrary locations. We show 2D implementations exist that are asymptotically better than the ET^{α} optimal 1D implementations. For simplicity, we assume that PEs operate on words in these sections, and that a single operation on a word can be performed with energy e and time t with $et^{\alpha} = 1$. Section 4.3 considers the impact of word size on the energy-time complexity of sorting by examining implementations where the data to be sorted consists of binary words of w bits and only PEs that operate on $O(1)$ bits are used.

4.1. 1D Sorting

A *sorting network* has N inputs of some fixed bit width that traverse a series of comparators leading to N outputs such that any permutation of input values will be output in sorted order. As an example, Figure 3 shows a sorting network for bubble sort from [31, Fig. 46(a)]. Each vertical line segment corresponds to a *compare-and-swap* unit (comparator, for short), and each horizontal segment corresponds to a connection between comparators. Data is input at the left, flows from left-to-right, and is output at the right. A comparator receives a data value on each of its inputs and outputs the larger of the two values on its top output and the smaller value on its bottom output. Figure 4 shows a direct implementation of the sorting network for bubble sort. Dark

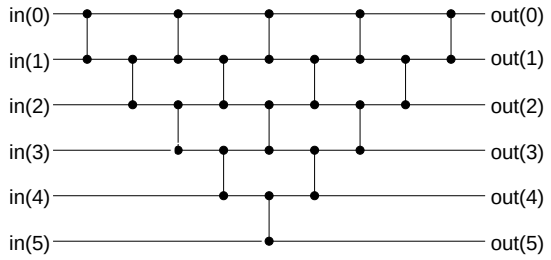

Fig. 3. A Sorting Network for Bubble Sort

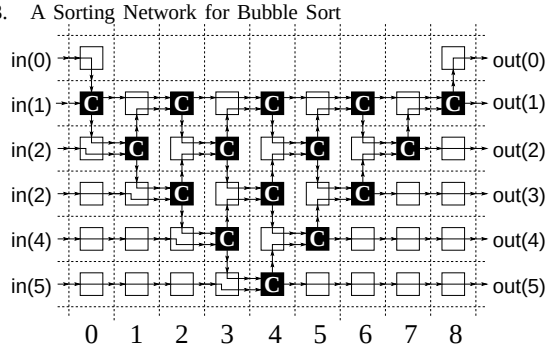

Fig. 4. The Direct Implementation of the Bubble Sort Network into a Grid of Processing Elements

boxes labeled **C** are comparators, and the other boxes are "wiring" units.

A direct implementation of a sorting network such as the one shown in Figure 4 is an example of what we call a "1D" implementation of an algorithm. In a 1D implementation, the input PEs must all lie along a line, and the output PEs must also lie on a (possibly different) line. In both the grid and gridless models, the line can be viewed as one which passes through the centerpoints of a fixed inscribed circle of the region each PE occupies. This is a natural arrangement for input/output ports that lie on the periphery of chips or functional blocks, and it also lends itself to simple interconnection and pipelining. The following lemma provides is useful for proving lower bounds in for 1D implementations.

Lemma 4.1: Let Q be a non-empty set of input (resp. output) PEs in a 1D implementation of an algorithm, and let p be an arbitrary PE. There exists $q \in Q$ such that $\mu(p,q) \geq (|Q|-1)/2$.

Proof: By induction on $|Q|$ using Axiom 3, it can be shown that there exist $q_1, q_2 \in Q$ such that $\mu(q_1,q_2) \geq |Q|-1$. From this, we conclude that $\mu(p,q_1)+\mu(p,q_2) \geq |Q|-1$ by the triangle inequality; thus $\max(\mu(p,q_1),\mu(p,q_2)) \geq (|Q|-1)/2$ establishing the claim. ■

The next theorem shows that the the energy-time complexity of 1D sorting is in $\Theta(N^{\alpha+1})$. The bubble sort implementation depicted in Figure 4 achieves this bound.

Theorem 4.2: [1D Sorting (word-wise)] Any optimal 1D implementation for the sorting problem on N inputs has $ET^\alpha = \Theta(N^{\alpha+2})$.

Proof: (Upper bound) The direct implementation as in Figure 4 has N rows with $2N-3$ PEs per row. It is straightforward to show that allotting unit time for all PEs results in linear total time and quadratic total energy. The upper bound follows immediately.

(Lower bound) Let $A = \{a_1, a_2, \ldots a_N\}$ be the set of input PEs and $B = \{b_1, b_2, \ldots b_N\}$ be the set of output PEs of any 1D implementation of sorting. Let

$$\begin{aligned} M_1 &= [1 \ldots N] \\ g(k) &= \arg\max_{i \in M_k} \mu(a_k, b_i) \\ M_{k+1} &= M_k - \{g(k)\}, k = 1, 2, ..., N-1 \end{aligned}$$

We choose a permutation of the input values that maps the value held in input a_i to output $b_{g(i)}$ for all $i \in [1 \ldots N]$. Clearly, $|M_i| = N+1-i$; thus $\mu(a_i, b_{g(i)}) \geq (N-i)/2$ by Lemma 4.1. Let e_i be the energy expended to send the input value of a_i to $b_{g(i)}$ if the computation is completed in T time units. By Axiom 5, $e_i \geq ((N-i)/2)^{\alpha+1} T^{-\alpha}$. Summing over all i yields $E \geq 2^{-(\alpha+1)} T^{-\alpha} \sum_{i=1}^{N} (N-i)^{\alpha+1} \in \Omega\left(N^{\alpha+2} T^{-\alpha}\right)$ which yields $ET^\alpha \in \Omega(N^{\alpha+2})$ as claimed. ■

It may seem surprising that bubble sort is an optimal algorithm for the problem of 1D sorting. For the sake of comparison, we have also analysed odd-even merge and shown that it also achieves $ET^\alpha \in O(N^{\alpha+2})$. While odd-even merge performs fewer comparisons than bubble-sort, odd-even merge has long-wires in stages where the comparators span many words. The total wire length in odd-even merge is a constant factor larger than that of bubble sort, and the energy and time for driving these long wires determines the cost of odd-even merge. More generally, fast sorting networks, such as odd-even merge, are "fast" because the traditional models for analysing such networks do not take into account the cost of communication. These algorithms exploit this "free" communication. In real chips, communication is not free, and the advantage of the fast algorithms is reduced to a constant factor.

4.2. 2D Sorting

We now consider implementations of sorting algorithms where the inputs and outputs may be placed at arbitrary locations. In this case, the largest distance between any input and output location can be reduced to $O(\sqrt{N})$. We first prove a lower bound for the ET^α complexity of 2D sorting implementations. We then show that the sorting algorithm of Schnorr and Shamir [29] achieves this bound, thus showing that this bound is tight. As in Section 4.1, we assume that PEs can operate on words.

Theorem 4.3: [2D Sorting (word-wise)] Any optimal implementation for the sorting problem on N inputs has $ET^\alpha \in \Theta(N^{\frac{\alpha+3}{2}})$.

Proof: (Upper bound) Schnorr and Shamir's algorithm performs sorts N values using a $\sqrt{N} \times \sqrt{N}$ mesh of cells, where each cell stores an input value and may compare and swap values with its vertical and horizontal neighbors [29] – all communication is between adjacent cells. The algorithm completes sorting in $O(\sqrt{N})$ time steps. Lemma 3.1 yields that the ET^α complexity of this algorithm is $O(N^{\frac{\alpha+3}{2}})$.

(Lower bound) Define A, B, M_i and $g(i)$ is in the proof for Theorem 4.2. We again choose a permutation of the input values that maps the value held in input a_i to output $b_{g(i)}$ for all $i \in [1 \ldots N]$. Clearly, $|M_i| = N + 1 - i$; thus $\mu(a_i, b_{g(i)}) \geq \sqrt{(N-i)}$ by Axiom 5. Let e_i be the energy expended to send the input value of a_i to $b_{g(i)}$ if the computation is completed in T time units. By Axiom 5, $e_i \geq (N-i)^{\frac{\alpha+1}{2}} T^{-\alpha}$. Summing over all i yields $E \geq T^{-\alpha} \sum_{i=1}^{N} (N-i)^{\frac{\alpha+1}{2}} \in \Omega\left(N^{\frac{\alpha+3}{2}} T^{-\alpha}\right)$ which yields $ET^\alpha \in \Omega(N^{\frac{\alpha+3}{2}})$ as claimed. ■

Note that the lower bound proofs for Theorems 4.2 and 4.3 are based on the communication requirements of sorting and the costs for performing comparisons has been disregarded. Intuitively, these proofs consider a "magical" sorting network that sends each value from its input PE to its output PE so that all values arrive at their destinations at the same time. The fact that real sorting networks can come to within a constant factor of these lower bounds, shows that the cost of sorting is bounded by its communication costs.

4.3. The Impact of Word Size

For simplicity, we admitted PEs that operate on words in Sections 4.1 and 4.2. However, the grid model requires PEs to have $O(1)$ inputs and outputs. We now consider the case where each word consists of w bits that are the binary representation of a positive integer value. We assume that the w bits for each input word of a sorting problem are stored in contiguous locations and likewise for the output words. We note that whether or not such scattering of the bits of a word can help in a model that charges for ET^α remains an open question. The lower bound arguments from Theorems 4.2 and 4.3 can be extended directly to establish lower bounds for the ET^α costs for 1D and 2D implementations for sorting of w-bit words. Matching upper bounds may be reached by adapting the bubble sort implementation for the 1D case and by adapting Schnorr and Shamir's algorithm for the 2D case. Due to space considerations, we state these bounds without proof, but provide the key intuition for the 2D upper bound below. Detailed proofs are found in [32].

Theorem 4.4: [Sorting (bits)] Any optimal 1D implementation for the sorting problem on N inputs of w bits when the bits are stored contiguously, likewise with the bit locations of the output bits and $w \geq \log N$ has $ET^\alpha \in \Theta((Nw)^{\alpha+2})$. Under the same conditions and the added restriction that $w \leq N^{1-\epsilon}$ for any $\epsilon > 0$, any 2D implementation for the sorting problem has $ET^\alpha \in \Theta((Nw)^{(\alpha+3)/2})$.

The condition that $w \geq \log N$ ensures that there are enough distinct values for words to ensure the existence of the permutations used in our lower bound arguments.

Schnorr and Shamir's sorting algorithm can be modified to meet the bounds described above. Our implementation uses a $\sqrt{Nw} \times \sqrt{Nw}$ array of PEs that are organized as tiles of $w \times w$ PEs. For phases where Schnorr and Shamir's algorithm performs comparisons between horizontally adjacent words, each column of a tile holds the bits of a word, and comparisons are done in a pipelined fashion starting from the most-significant bit. Likewise, for phases where Schnorr and Shamir's algorithm performs comparisons between vertically adjacent words, each row of a tile holds the bits of a word. Special tiles handle the bends in "snakes" arrangements.

When a horizontal phase is followed by a vertical phase (or vice-versa), each tile must perform a transposition of its bits between the phases. The requirement that $w \leq N^{1-\epsilon}$ ensures that the energy-time costs for these transposition operations are dominated by the other costs of the algorithm. Transposition introduces another issue as well. In Schnorr and Shamir's formulation, each column (resp. row) after the transposition has exactly one value from each row (resp. column) from before the transposition. With our formulation, each column (resp. row) receive w consecutive words from each row (resp. column). Thus the counting arguments in Schnorr and Shamir's proof of correctness must be modified to handle these changes. As noted above, details are given in [32].

Finally, we note that Schnorr and Shamir formulated their algorithm to achieve the lower bound for the AT^2 cost of sorting. Each of their comparators can exchange a complete word with its left-and-right neighbours in a single cycle, or it can exchange words with its upper-and-lower neighbours. If a word consists of w bits, then a Schnorr and Shamir cell must have height and width both in $\Omega(w)$ to have enough communication bandwidth for these operations. Thus, each comparator cell has area in $\Omega(w^2)$ and the entire sorting array has an area in $\Omega(Nw^2)$. Our sorter achieves an area in $\Theta(Nw)$, improving on the area-time cost of Schnorr and Shamir's algorithm when the fact that words must be implemented with multiple bits is taken into account.

5. Binary Addition

The binary addition problem consists of two N-bit input values, x and y, and one $(N+1)$-bit output value s. We write x_i to denote that i^{th} bit of x, and likewise for y and s.

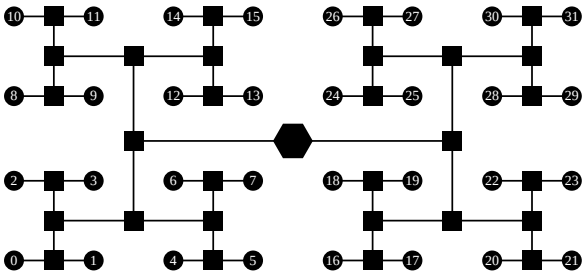

Fig. 5. An H-Tree Implementation of an Adder

There are predetermined input and output PEs for the bits of x, y, and s. At the completion of the algorithm, output PEs holding the bits of s are set to represent the sum of the values initially stored in x and y. As with sorting, we first consider 1D implementations where the bits of the words x, y, and s are each arranged as one-dimensional arrays followed by 2D implementations where the placement of the input and output bits are unconstrained.

We obtain lower bounds for ET^α complexity of addition by noting that the carry produced by the least significant bits x_0 and y_0 can affect all bits of s.

Lemma 5.1: [Addition, Lower Bounds] Any 1D (resp. 2D) implementation for the problem of binary addition on N-bit inputs has $ET^\alpha \in \Omega(N^{\alpha+1})$ (resp. $ET^\alpha \in \Omega(N^{(\alpha+1)/2})$.

Proof: Let $p_{x,i}$ denote the PE initially storing x_i; let $p_{s,i}$ denote the PE initially storing s_i. In any 1D implementation, there must be some $i \in \{0, \ldots, N-1\}$ such that $\mu(p_{x,0}, p_{s,i}) \geq N/2 \in \Omega(N)$ (Lemma 4.1). In any 2D implementation, there must be some $i \in \{0, \ldots, N-1\}$ such that $\mu(p_{x,0}, p_{s,i}) \in \Omega(\sqrt{N})$ (Axiom 3). The results follow from Axiom 5 as the value of x_0 can determine the value of s_i. ∎

The lower-bound for 1D implementations is achieved by a carry-ripple adder implemented by full-adder PEs in the obvious direct implementation. We note that a direct implementation of a Brent-Kung carry-lookahead adder [27] achieves the same bound, but only when the time and energy varies per PE. A straightforward 1D direct implementation of the Kogge-Stone [33] adder has optimal ET^α in $O(N^{\alpha+2})$. In computational models where communication is free, the Kogge-Stone adder achieves a constant factor improvement in time over the simpler Brent-Kung algorithm. It does this by duplicating the carry-lookahead calculation and using many long wires. When the cost of communication is accounted for, the Kogge-Stone adder is decidedly worse than simpler approaches.

The lower-bound for 2D implementations is achieved by a carry-lookahead adder embedded in an H-tree structure as shown in Figure 5. For simplicity, we assume that N is a power of two, i.e. $N = 2^m$. The addition proceeds in two phases. In the first phase, the leaf PEs which serve as both inputs and outputs (circles) compute propagate-generate (PG) values that are merged by the internal PEs (squares), eventually reaching the root (hexagon). In the second phase, the carry decisions are propagated down from the root, with each internal node computing the carries for its two subtrees based on the carry from its parent and the PG values from its subtrees. We note that a Brent-Kung adder could be used as well, but use this variant as it simplifies the following analysis.

Let the root of the tree be at level 0, and the leaves at level m. For $k > 0$, an edge between levels k and $k-1$ is implemented by a chain of length $2^{\lfloor (m-k)/2 \rfloor} - 1$ PEs. If we set the time for PEs at level k and the PEs in the chain from level k to level $k+1$ to $2^{k/(\alpha+1)}$, then the total time and energy to compute the addition are both given by

$$\begin{aligned} E, T &\leq \sum_{k=0}^{m-1} 2^{\frac{m}{2} + \frac{i(1-\alpha)}{2(\alpha+1)}} \\ &\in \begin{cases} O\left(N^{1/(\alpha+1)}\right), & 0 \leq \alpha < 1; \\ O(\sqrt{N} \log N), & \alpha = 1; \\ O(\sqrt{N}), & \alpha > 1. \end{cases} \end{aligned} \tag{3}$$

Lemma 2.1 yields that the H-tree adder can achieve $ET^\alpha \in O\left(N^{(\alpha+1)/2}\right)$ for any $\alpha > 1$.

Theorem 5.2: [Bounds for Addition] Any optimal implementation for the problem of 1D addition on N-bit inputs has $ET^\alpha \in \Theta(N^{\alpha+1})$. Any optimal implementation for the problem of 2D addition on N-bit inputs has

$$ET^\alpha \in \begin{cases} \Theta\left(N^{(\alpha+1)/2}\right), & \alpha > 1; \\ \Omega\left(N^{(\alpha+1)/2}\right), & \alpha > 0; \\ O\left(N(\log N)^2\right), & \alpha = 1; \\ O(N), & 0 < \alpha < 1. \end{cases}$$

Proof: The lower bounds were established in Lemma 5.1. The upper bound for the 1D case is achieved by a carry-ripple adder direct implementation, and upper bounds for the 2D case are achieved by an H-tree adder implementation. ∎

We conclude this treatment of addition with three observations. First, the lower bounds for addition are also lower bounds for broadcast. The surprising result is that an H-tree allows a bit to be transmitted to *all* PEs within distance d of a source for a constant factor times the cost of transmitting to a *single* destination at distance d. Second, we note that unlike sorting, the lower bound for addition requires operating PEs at different speeds. In particular, those close to the root must operate faster than those at the leaves. It is easy to show that broadcasting a bit to N destinations requires $ET^\alpha \in \Omega(N^{(\alpha/2)+1})$ if all PEs operate at the same speed. Finally, we note that the ability to perform the small number of operations near the root of the tree in less time (but at higher energy per operation) than those at the leaves allows

the energy-scaled algorithm to overcome the limitations of the sequential bottleneck in this reduction computation.

6. Multiplication

We define the binary multiplication problem in a manner analogous with that for binary addition. The input words are x and y and the output is word is p. At the completion of the algorithm, the output PEs holding the bits of p are set to represent the product of the values initially stored in the input PEs for x and y. We consider both 1D and 2D implementations.

6.1. 1D Multiplication

Following the classical arguments for the AT^2 complexity of multiplication [26], we observe that shifting can be reduced to multiplication. For any 1D arrangement of x and p, we can find a shift amount such that at least $N/2$ of the bits of x must move a distance of at least $N/2$ to reach their output PEs. Applying Axiom 5 yields a lower bound of $\Omega\left(N^{\alpha+2}\right)$ for 1D multiplication. A direct implementation of an add-pass multiplier using carry-save arithmetic [34, Chap. 10.3] achieves this bound. This yields:

Theorem 6.1: [1D Multiplication] Any optimal implementation for the problem of 1D multiplication on N-bit inputs has $ET^\alpha \in \Theta(N^{\alpha+2})$.

Proof: Omitted due to space constraints. See [32]. ■

6.2. 2D Multiplication

The reduction from shifting can be applied to 2D implementations as well. In this case, using Axiom 3 we conclude that for each input bit of x there are at least $3N/4$ output bits of p that are at least distance $\sqrt{N}/2$ away. A pigeon-hole argument shows that there must be a shift for which at least $3N/4$ input bits must be sent a distance of at least $\sqrt{N}/2$. Applying Axiom 5 yields a lower bound of $\Omega\left(N^{(\alpha+3)/2}\right)$.

Preparata presented an AT^2 optimal implementation of a multiplier using discrete Fourier transforms [28]. This implementation has $A \in O(N)$, $T \in O(\sqrt{N})$ and only uses nearest neighbor communication when decomposed into PEs. We apply Lemma 3.1 and conclude that Preparata's multiplier has $ET^\alpha \in O\left(N^{(\alpha+3)/2}\right)$.

Theorem 6.2: [Multiplication Tight Bound] Any optimal implementation for the problem of multiplication on N-bit inputs has $ET^\alpha \in \Theta(N^{\alpha+2})$.

Proof: The lower bound is sketched above based on reduction from shifting, and the upper bound is achieved by Preparata's multiplier. Details are omitted due to space constraints but can be found in [32]. ■

Problem	Algorithm/Implementation	ET^α
sorting	$1D$ bubble-sort	$O(N^{\alpha+2})$
	$1D$ odd-even merge-sort	$O(N^{\alpha+2})$
	Schnorr and Shamir's	$O(N^{(\alpha+3)/2})$
addition	$1D$ carry-ripple	$O(N^{\alpha+1})$
	$1D$ Brent-Kung	$O(N^{\alpha+1})$
	$1D$ Kogge-Stone	$O(N^{\alpha+2})$
	H-Tree	$O(N^{(\alpha+1)/2})$
multiplication	$1D$ carry-save	$O(N^{\alpha+2})$
	Preparata's	$O(N^{(\alpha+3)/2})$

TABLE I
Summary of ET^α complexities for the presented implementations.

7. Conclusions

Power consumption has become the most critical bottleneck to further increases in computer performance. The underlying physics of VLSI circuits allow operations to be performed using less energy when they are performed using more time. This creates a large incentive for exploiting parallel computation: a parallel algorithm can be faster and use less energy than its sequential counterpart even in cases where the parallel algorithm performs more operations. To exploit these opportunities, programmers need a computation model that reflects the energy-time trade-offs afforded by parallelism. However, we are aware of no prior work that analyses the complexity of computation in a model that takes energy-time trade-offs into account. This report presented a simple model for algorithm design that corresponds to the $power = speed^\alpha$ abstraction that has been used successfully for energy-optimal scheduling. In particular, each primitive operation uses energy, e, and time, t such that et^α is constant for some $\alpha > 0$.

We applied this model to the problems of sorting, binary addition and binary multiplication. For all three problems we derived asymptotic lower bounds and presented algorithms that achieve these bounds to within constant factors. These results are summarized in Table 1. For $\alpha = 2$, the optimal implementations for all three of these problems have the remarkable property that both time and energy can scale slower than N. For example, multiplication and sorting can have E and T both scale as $N^{5/6}$, and addition can have E and T scale as $\sqrt{N}$.

Our algorithms for multiplication and sorting are direct adaptations of existing systolic algorithms that were derived to achieve AT^2 optimal computation. For multiplication and sorting, all operations can be performed at the same et^α point — in other words, no voltage scaling is needed to achieve optimality for these problems. Our algorithm for addition is a simple, carry-lookahead adder embedded in a 2D mesh. Here, we use a different energy-time trade-off for

each level of the tree. This scaling achieves the somewhat surprising result of showing that a bit can be broadcast to all d^2 processing elements within distance d of a source for a constant multiple of the cost to send the bit to a single destination at distance d. It is straightforward to show that any implementation of addition where the same amount of time is used for all operations performed by all PEs has an $ET^\alpha \in \Omega(N^{(\alpha+1)/2})$.

The results presented in this report are just a beginning of an exploration of what can be done with energy-time trade-offs. The arguments we used to derive lower bounds and analyse algorithms are mathematically straightforward. We regard this as a feature of our model: fundamental algorithms can be analysed using basic techniques; thus our approach should be accessible to practicing programmers and computer designers. Likewise, we used our model to compare existing algorithms for the problems we considered and found ones that met the lower bounds we derived. Our approach does not require programmers to start over from scratch, but gives them a basis to evaluate existing algorithms and design new ones where needed.

We also believe that understanding energy-time trade-offs for algorithms can inform evaluations of architectural trade-offs when designing parallel computers. For example, current multi-core processors are implemented using cross-bar interconnects. While cross-bars are simple, they scale poorly to designs with large numbers of cores. For the algorithms that we considered, simple mesh and tree topologies were sufficient to achieve optimal performance. Extending our analysis to a broader class of algorithms could help architects select the interconnect topologies for processors with large numbers of cores. Our models allow each operation to be performed at a different energy-time trade-off point. Energy-time optimal addition requires this flexibility, but sorting and multiplication do not. A study of the energy-time trade-offs for a wider range of algorithms could help inform architectural decisions such as how many independent voltage and timing domains should be used in a processor.

We plan to extend this research by examining other models and extending the model to reflect additional physical phenomena such as transistor threshold voltages and leakage currents. Likewise, we would like to consider additional mechanisms for trading time and energy such as processes with multiple threshold voltages, transistor sizing [30], and microarchitectural trade-offs [1], [17]. Taken together, these offer several orders of magnitude of energy per operation, yet they need the abstraction of a simple model such as the one in this report to make the analysis accessible and useful to designers and programmers.

In addition to ET^α bounds, we are also interested in EP^α where P is the *period* (inverse throughput) for the computation. For example, a pipelined, add-pass multiplier achieves an EP^α of N^2 which, for any $\alpha > 1$, is better than Preparata's FFT-based multiplier that we described in Section 6.2. Observations such as these motivate studying the implications of latency and throughput trade-offs for the energy costs of computation.

Power consumption is the most critical bottleneck to computer performance now and for the foreseeable future. By including the energy-time trade-offs of CMOS VLSI in a simple model of computation, algorithms can be designed in a way that takes their energy consumption into account. We have shown that this approach can be used to identify energy-optimal algorithms for sorting, addition and multiplication. We believe that further work with models like the one presented in the report will provide programmers and architects with the analytical tools that they need to maximize performance in power-constrained technologies.

REFERENCES

[1] T. Mudge, "Power: a first class architectural design constraint," *IEEE Computer*, vol. 34, no. 4, pp. 52–58, Apr. 2001.

[2] C. Isci, A. Buyuktosunoglu, *et al.*, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *Proc. 39^{th} Annual IEEE/ACM Workshop on Microarchitecture (MICRO39)*, Dec. 2006, pp. 347–358.

[3] B. Hoeneisen and C. A. Mead, "Fundamental limits in microelectronics I: MOS technology," *Solid-State Electronics*, vol. 15, pp. 819–829, 1972.

[4] R. Dennard, F. Gaensslen, *et al.*, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE J. Solid-State Circuts*, vol. 9, no. 5, Oct. 1974.

[5] A. J. Martin, "Towards an energy complexity of computation," *Information Processing Letters*, vol. 77, pp. 181–187, 2001.

[6] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. of the 36^{th} Symp. on the Foundations of Computer Science*, Oct. 1995, pp. 374–382.

[7] S. Albers and H. Fujiwara, "Energy-efficient algorithms for flow time minimization," in *Proc. 23^{rd} Int'l. Symp. Theoretical Aspects of Computer Science*, 2006, pp. 621–633.

[8] S. Albers, F. Müller, and S. Schmelzer, "Speed scaling on parallel processors," in *SPAA '07: Proc. 19^{th} ACM Symp. Parallel Algorithms and Architectures*. New York, NY, USA: ACM, 2007, pp. 289–298.

[9] N. Bansal, T. Kimbrel, and K. Pruhs, "Dynamic speed scaling to manage energy and temperature," in *Proc. 45^{th} IEEE Symp. Foundatations of Computer Science (FOCS'04)*, 2004, pp. 520–529.

[10] P. Baptiste, "Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management," in *SODA '06: Proc. of the 17^{th} ACM-SIAM Symp. Discrete Algorithms*. 2006, pp. 364–367.

[11] D. P. Bunde, "Power-aware scheduling for makespan and flow," in *Proc. of the 18^{th} ACM Symp. Parallel Algorithms and Architectures (SPAA'06)*, 2006, pp. 190–196.

[12] J.-J. Chen, H.-R. Hsu, K.-H. Chuang, C.-L. Yang, A.-C. Pang, and T.-W. Kuo, "Multiprocessor energy-efficient scheduling with task migration considerations," in *ECRTS '04: Proc. 16^{th} Euromicro Conference on Real-Time Systems (ECRTS'04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 101–108.

[13] J.-J. Chen, T.-W. Kuo, and H.-I. Lu, "Power-saving scheduling for weakly dynamic voltage scaling devices." in *Proc. 9^{th} Workshop on Algorithms and Data Structures* (WADS), 2005, pp. 338–349.

[14] K. Asanovic *et al.*, "The landscape of parallel computing research: A view from berkeley," Dept. of Elec. Eng. and Comp. Sci., University of California, Berkeley, Tech. Rep. EECS-2006-183, Dec. 2006.

[15] S. Naffziger, B. Stackhouse, *et al.*, "The implementation of a 2-core, multi-threaded Itanium family processor," *IEEE J. Solid State Circuits*, vol. 41, no. 1, pp. 197–209, Jan. 2006.
[16] E. Gelenbe, *Multiprocessor Performance*. John Wiley and Sons, 1989.
[17] R. Gonzalez and M. A. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE J. Solid-State Circuits*, vol. 31, pp. 1277–1284, Sept. 1995.
[18] S. Gochman, R. Ronen, *et al.*, "The Intel Pentium M processor: Microarchitecture and performance," *Intel Technology Journal*, vol. 7, no. 2, pp. 21–36, May 2003.
[19] L. T. Clark, E. J. Hoffman, *et al.*, "An embedded 32-b microprocessor core for low-power and high-performance applications," *IEEE J. Solid-State Circuits*, vol. 36, no. 11, pp. 1599–1608, Nov. 2001.
[20] G. Magklis, M. L. Scott, *et al.*, "Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor," in *Proc.* 30^{th} *Int'l. Symp. Computer Architecture*, 2003, pp. 14–25.
[21] A. Iyer and D. Marculescu, "Power-performance evaluation of globally asynchronous, locally synchronous processors," in *Proc.* 29^{th} *Int'l. Symp. Computer Architecture*, June 2002, pp. 158–168.
[22] S. Williams, J. Shalf, *et al.*, "The potential of the Cell processor for scientific computing," in *Proc.* 3^{rd} *Conf. Computing Frontiers (CF'2006)*. New York, NY, USA: ACM Press, 2006, pp. 9–20.
[23] M. Gschwind, H. P. Hofstee, *et al.*, "Synergistic processing in Cell's multicore architecture," *IEEE Micro*, vol. 26, no. 2, pp. 10–24, 2006.
[24] M. Weiser, B. Welch, *et al.*, "Scheduling for reduced CPU energy," in *Proc.* 1^{st} *Symp. Operating Systems Design and Implementation (OSDI'94)*, 1994, pp. 13–23.
[25] C. D. Thompson, "A complexity theory for VLSI," Ph.D. dissertation, Carnegie Mellon University, 1980.
[26] R. P. Brent and H. T. Kung, "The area-time complexity of binary multiplication," *J. ACM*, vol. 28, no. 3, pp. 521–534, 1981.
[27] R. Brent and H. Kung, "A regular layout for parallel adders," *IEEE Trans. Computers*, vol. C-31, no. 3, pp. 260–264, Mar. 1982.
[28] F. P. Preparata, "A mesh-connected area-time optimal VLSI multiplier of large integers." *IEEE Trans. Computers*, vol. 32, no. 2, pp. 194–198, 1983.
[29] C. Schnorr and A. Shamir, "An optimal sorting algorithm for mesh connected computers," in *Proc.* 18^{th} *ACM Symp. Theory of Computing (STOC18)*, May 1986, pp. 255–263.
[30] J. C. Ebergen, J. Gainsley, and P. Cunningham, "Transistor sizing: How to control the speed and energy consumption of a circuit," in *Proc.* 10^{th} *Int'l. Symp. Asynchronous Circuits and Systems (ASYNC'04)*, Apr. 2004, pp. 7–16.
[31] D. E. Knuth, *The Art of Computer Programming*. Addison-Wesley, 1972, vol. 3: Sorting and Searching.
[32] Thesis written by one of the authors of this paper. Detailed proofs that do not fit in the space constraints of this paper are available here.
[33] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Computers*, vol. 22, pp. 786–793, 1973, uRL: `http://cr.yp.to/bib/entries.html#1973/kogge`.
[34] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, 2000.