Green Computing Algorithmics

Kirk Pruhs(✉)

University of Pittsburgh, Pittsburgh, PA 15260, USA
kirk@cs.pitt.edu
https://people.cs.pitt.edu/~kirk/

Abstract. We discuss what green computing algorithmics is, and what a theory of energy as a computational resource isn't. We then present some open problems in this area, with enough background from the literature to put the open problems in context. This background should also be a reasonably representative sample of the green computing algorithmics literature.

1 Introduction

Time arguably ascended over space as the dominant computational resource circa 1970, when semiconductor memory replaced magnetic core memory, and energy arguably ascended over time as the dominant computational resource circa 2000, when information technologies could no longer handle the exponentiality of Moore's law. Three examples illustrating the ascendancy of energy in the areas of computer chip design, data center management, and high performance computing are:

- In May 2004, Reuters reported that: **Intel Hits Thermal Wall:** "Intel Corp. said on Friday it has scrapped the development of two new computer chips (code-named Tejas and Jayhawk) for desktop/server systems in order to rush to the marketplace a more efficient chip technology more than a year ahead of schedule. Analysts said the move showed how eager the world's largest chip maker was to cut back on the heat its chips generate. Intel's method of cranking up chip speed was beginning to require expensive and noisy cooling systems for computers."
- In September 2002, the New York Times quoted then Google CEO Eric Schmidt as saying, "What matters most to the computer designers at Google is not speed, but power, low power, because data centers can consume as much electricity as a city."
- In 2010 the Advanced Scientific Computing Advisory Committee in their report *The Opportunities and Challenges of Exascale Computing* stated that one of the three main challenges is reducing power requirements. In particular, it said, "Based on current technology, scaling today's systems to an exaflop level would consume more than a gigawatt of power, roughly the output of Hoover Dam. Reducing the power requirement by a factor of at least 100 is a challenge for future hardware and software technologies."

B. Steffen and G. Woeginger (Eds.): Computing and Software Science, LNCS 10000, pp. 161–183, 2019.
https://doi.org/10.1007/978-3-319-91908-9_10

Thus we are about a decade into a green computing revolution in which a wide array of information technologies are being redesigned with energy as a first class design constraint [54]. Inevitably these new technologies have spawned new theoretical/algorithmic problems. The most obvious type of algorithmic problem arising from this green computing revolution involves directly managing power, energy or temperature as a computational resource. Additionally, these new, more green, technologies also often have different non-energy-related physical properties than previous technologies. For example, in many new memory technologies, writes are significantly more time consuming than reads. Thus the management of these new informational technologies, that will likely be adopted for reasons related to energy efficiency, often give rise to new algorithmic problems that do not directly involve managing energy. Green computing algorithmics is just the study of the algorithmic problems spawned by the green computing revolution.

When I (as one of the very few researchers working extensively on green computing algorithms) was asked to write this paper, I had first had to decide who my intended audience to be. I decided that my primary intended audience are researchers, particularly young researchers, who are potentially interested in initiating a research program in green computing algorithms. Another plausible audience for this paper are researchers who want to know what has happened in green computing algorithmics, and where the field is going.

Firstly we should discuss what research in green computing algorithms is like. Some of the problems spawned by the green computing revolution are algorithmically interesting, and many/most are not. For our purposes, let us say that a problem is algorithmically interesting if it is a problem for which a trained/expert algorithms researcher can obtain interesting results/insights that a generic application researcher likely wouldn't be able to attain. However, before algorithmic researchers can contribute, they have to become aware of these algorithmically interesting problems. From my experience, the bottleneck for algorithmic research community making a contribution to green computing is not a lack of researchers with the skills to make headway on algorithmically interesting problems, but a lack of researchers who make the effort to mine/identify algorithmically interesting management issues in new greener information technologies, formalize these problems, and then expose these problems to the algorithmic research community. And with all due respect to the experimental computer systems research communities, few researchers in those communities are able to identify algorithmically interesting problems, or are able to formalize the problems so that algorithmic research will provide maximum insight. There is a bit of an art to constructing a formal model/problem that accurately captures the complicated interactions and goals in the real computing technology, but that is still amenable to mathematical analysis that yields useful insight. Often the most obvious ways to formally model technological problems lead to uninteresting formal problems, for example because one gets a trivial impossibility result.

Thus much of green computing algorithmics research involves problem mining and problem formulation, not solving well-known formal problems. But as

effective problem mining and formulation is usually difficult for researchers who are new to an application area, particularly if they are also junior researchers, I've decided structure this survey around a collection of formal algorithmic problems related to green computing algorithmics that I find interesting, instead of trying to do some sort of comprehensive survey, like the previous two surveys on green computing algorithmics [2,39]. The hope is these problems might provide a starting point for a researcher looking to initiate a research program on green computing algorithmics. This survey is certainly biased towards my own research. But still I think that these open problems, and the associated background that I give, should form a representative sample of the research in green computing algorithmics.

The outline for the rest of the survey is:

- Section 2 discusses why a theory of energy as a computational resource will look different than the established theory of time or space as computational resources.
- Section 3 discusses common modeling assumptions.
- Section 4 discusses online convex optimization problems that naturally arise from different data centers trying to handle a time-varying workload in the most energy efficient manner.
- Section 5 discusses energy efficient circuit routing in a network.
- Section 6 discusses near-threshold computing, and designing combinatorial circuits that optimally tradeoff energy efficiency and reliability.
- Section 7 discusses online scheduling on heterogeneous processors.
- Section 8 discusses the complexity of finding schedules that optimally tradeoff energy and performance.
- Section 9 provides a brief conclusion that compares the current state of the theory energy as a computational resource to the theories of time and space as computational resources.

2 The Theory of Energy as Computational Resource

At the start of this green computing revolution, there were several National Science Foundation sponsored visioning workshops, many of which I attended, and one of which, on the *The Science of Power Management*, I organized. A ubiquitous first demand of all manner of computing researchers at all of these visioning workshops was:

"We need a big Oh theory for energy!"

The universal demand for a such theory is a testament to the tremendous success of the current theory time (and space) as a computational resource within the broad computing community. Of course what they are talking about is only tangentially related to big Oh. They are referring to being able to label algorithms, and problems, with time and space complexities. So the algorithm MergeSort

takes time $O(n \log n)$ and the problem of deciding whether two regular expressions requires exponential space. This is possible because of the time hierarchy and space hierarchy theorems, which are the bedrock of the theory of time and space as computational resources.

But the laws of physics make it difficult to assign algorithms, and thus problems, energy complexities. There are two natural possible approaches. The first approach is to assume that all standard computer operations use unit energy, analogous to the standard assumption that all operations take unit time. At first glance this assumption for energy seems not significantly more unreasonable than the assumption for time. But a consequence this assumption is that the energy used by an algorithm is equal to the time used by that algorithm. This is unsatisfying theoretically as it produces no new mathematical/theoretical questions, and is unsatisfying practically as the green computing revolution is being caused by the fact that optimizing for energy instead of time has lead to practitioners developing significantly different technologies/solutions.

The second natural approach would be to assume that different operations require different amounts of energy, where the amount of energy required by an operation would be the minimum energy allowed by the laws of physics. The most natural candidate for a physical law to lower bound the energy for computation is Landauer's principle, which is intuitively related to the second law of thermodynamics. Landauer's principle states that the amount of energy that must be expended each time a bit of information is lost is linear in the temperature. Thus irreversible operations, where information is lost, like overwriting a bit, or taking the AND of two bits, require energy. But Landauer's principle does not give a lower bound on energy for reversible operations, like swapping the contents of two memory locations. Thus Landauer's principle can give a lower bound on energy for a combinatorial circuit, or a particular implementation of an algorithm. But unfortunately for this approach, all computation can be made reversible. In the context of combinatorial circuits, there are reversible logically-complete gates, such as the controlled swap or Fredkin gate. In the context of Turing machine, all computation can be made reversible at a cost of a linear slowdown in time. Thus every algorithm can be implemented using only reversible operations. Thus its at best a bit tricky giving an algorithm an energy complexity as it depends on the implementation. And there is no way to give an energy complexity to a problem as every computable problem can be solved with arbitrarily little energy.

I know from experience that many researchers are reticent to accept that a theory of energy as a computational resource will not be based on complexity classes like ENERGY(n^2), that are analogous to the TIME(n^2) and SPACE(n^2) complexity classes that are the foundation of the current theory of time/space as a computational resource. But the lack of energy complexity classes seems to be dictated by physics, and their absence from the current theory of energy as a computational resource isn't an oversight.

But there should be ways in which a theory of energy as a computational resource will be similar to the established theory of time/space as computational

resource. It should be based on simple models that balance the competing needs of accurately reflecting reality, and being mathematically tractable. It should serve engineers and computing researchers, when confronted with problems in which power/energy/temperature is the key scarce, as the current theory of time as a computational resources serves them when confronted with problems where time is the scarce resource. That is, it should give them appropriate simple models for commonly arising situations that will allow then to think heuristically about the preeminent issues.

For example, a common mechanism for achieving energy efficiency is building a system with heterogeneous devices, each with different energy and performance characteristics. For a given area and power budget, heterogeneous designs often give significantly better performance, for a given energy/hardware budget, for standard workloads than homogeneous designs [24,42,52,53]. One common scenario is that there might be some devices that are slow but energy efficient, and other devices that are fast but energy inefficient. The resulting management problem is how to choose the appropriate collection of devices for a workload to properly balance energy efficiency and performance. So a theory of energy as a computational resource should include appropriate models for such situations, and general algorithmic design and analysis techniques for dealing with such problems.

3 Common Modeling Assumptions

We quickly summarize common modeling assumptions. As much of the green computing algorithmic literature focuses on saving energy in processors, this discussion is biased toward models related to processors.

Energy is power integrated over time. Power has two components, dynamic power and static power. Dynamic power is the power used by the process of computing, and the static power is the energy used by the device just from being on. Thus the only real way to not spend energy on static power is to turn the device off. Before say the year 2000, the static power in common processors was generally negligible compared to the static power, but now the static power is often comparable to dynamic power. Generally there is a strictly convex relationship between speed and power. So faster processors have a higher ratio of power to speed than lower speed processors. Thus faster processors are less efficient than slower processors in terms of the energy that they expend per operation. Which seems to be a general consequence of the laws of physics, as for example, high performance cars invariably less energy efficient than lower performance cars. The most common model is that the dynamic power $P(s)$ is equal to s^{α}, where s is the speed and α is some constant that is strictly bigger than 1. For example, the well known cube-root rule for CMOS based devices states that the speed s is roughly proportional to the cube-root of the dynamic power P, or equivalently, the dynamic power is proportional to the speed cubed [25]. So the most common model for power in the literature is $P(s) = s^{\alpha} + \beta$, where β is constant representing the static power, and α is a constant that one thinks as being about 2 or 3. For more detail on modeling processor power see [26].

We now turn our attention to temperature. Cooling, and hence temperature, is a complex phenomenon that can not be modeled completely accurately by any simple model [60]. All the green computing literature that I have seen assumes that cooling is governed by Newton's law of cooling, and implicitly assumes that environmental temperature is constant (which is what a fan is in principle trying to achieve). Newton's law cooling states that the rate of cooling is proportional to the difference in temperature between the object and the environment. Without loss of generality one can scale temperature so that the environmental temperature is zero. A first order approximation for the rate of change T' of the temperature T is then $T' = aP - bT$, where P is the supplied power, and a and b are constants. The maximum temperature is within a factor of 4 of a times the maximum energy used over any interval of length $\frac{1}{b}$ [14]. This observation also shows that there is a relationship between total energy and maximum temperature optimization and simplifies the task of reasoning about temperature. If the cooling parameter b is 0 then maximum temperature minimization becomes equivalent (within a constant factor) to energy minimization. This also explains why some algorithms in the literature for energy management are poor for temperature management, that is, these algorithms critically use the fact that the parameter $b = 0$. If the cooling parameter b is ∞ then maximum temperature minimization becomes equivalent to minimizing the maximum power, or equivalently minimizing the maximum speed. [14] uses the term *cooling oblivious* to refer to an algorithm that doesn't rely on the particular values of a and b. Thus if a cooling oblivious algorithm performs well for the objective of minimizing the maximum temperature, it should perform reasonably well for the objective of minimizing the total energy and the objective of minimizing the maximum speed/power.

Moore's law states that the switch/transistor density in processors doubles about every other year. While at some point Moore's law has stop holding, it looks like it will continue to hold a bit longer. Up until around 2004, this also meant that processor speeds also doubled about every other year. But since 2004 the rate of improvement of processor performance (speed) lags the rate of transistor density. This is called Moore's gap. So let me now summarize some of the more energy efficient processor architecture designs that have arisen to cope with the exponentiality of Moore's law, and how they are generally modeled.

Speed Scalable Processors: A speed scalable processor has a collection of operational modes, each with a different speed and power. When performance is important the processor can be run at a high-speed (but with low energy efficiency) and when performance is less important, the processor can be run at a lower speed that has better energy efficiency. One gets a wide variety of models depending on whether the speed is assume to be discrete or continuous, whether there is an upper bound on speed, and the speed to power function.

Multiprocessor Chips: Another alternative design is multiprocessor chips. Roughly speaking, k speed s/k processors would use only about $1/k^2$ fraction of the power of a single speed s processor, but potentially would have the same

processing capability. But the fact that such efficient parallelization of computation is not so easy to pull off in practice is one of the reasons for Moore's Gap. As best as I can tell there are three different visions of the future of multiprocessor chips:

- *Identical Processors:* The first is expressed by the following quote from Anant Agarwal, CEO of Tilera: "I would like to call it a corollary of Moore's Law that the number of cores will double every 18 months." [52].
- *Related Processors:* Others [24,42,43,52,53] predict that the future dominant multiprocessor architecture will be heterogeneous processors/cores. It is envisioned that these heterogeneous architectures will consist of a small number of high-performance processors (with low energy efficiency) for critical jobs, and a larger number of lower-performance (but more energy efficient) processors for less critical jobs (and presumably eventually processors of intermediate performance for jobs of intermediate importance). For a given area and power budget, heterogeneous designs can give significantly better performance for standard workloads [24,42,52,53]. Such technologies are probably best modeled by what is called "related" processors in the scheduling nomenclature: That is, each processor i has an associated speed s_i and power P_i.
- *Unrelated Processors:* Some architects claim that, again due to Moore's law, chip makers soon will hit another thermal wall in that the density of switches will become so great that it will be prohibitively expensive to cool the chip if all switches were active at the same time [29]. Thus it will be necessary that all times, some switches must be turned off. This commonly goes by the moniker "dark silicon". Thus it is envisioned that there will be processors specialized for particular types of jobs, and for jobs to be assigned to a processor best suited for that job; and the processors not best suited for the current jobs would be turned off. In such a setting the processors might naturally be modeled by what in scheduling parlance is called "unrelated machines", where the execution time of a job is dependent on the processor to which is it assigned (and there is no particular consistency between which processors are fastest between jobs). We should point out that even multiprocessors that were designed to be homogeneous, are increasingly likely to be heterogeneous at run time [24]: the dominant underlying cause is the increasing variability in the fabrication process as the feature size is scaled down.

4 Online Convex Optimization

The Definition of the Online Convex Optimization Problem (OCO): The input is an online sequence $F_1, F_2, \ldots, F_n$ of convex functions from $\mathbb{R}^k$ to $\mathbb{R}^+$. In response to the convex function F_t that arrives at time t, the online algorithm can move to any destination/point p_t in the metric space $\mathbb{R}^k$. The cost of such a feasible solution is $\sum_{t=1}^{n} (d(p_{t-1}, p_t) + F_t(p_t))$, where $d(p_{t-1}, p_t)$ is the distance between points p_{t-1} and p_t, that is, the distance traveled plus the value of the convex functions at the destination points. The objective is to minimize the cost.

The Motivating Data Center Application: The initial motivation for introducing and studying the OCO problem was due to its applications in rightsizing power-proportional data centers, see for example [4,46–49,51,65]. To explain this application, let us for simplicity initially assume that we have one data center consists of a homogeneous collection of servers/processors that are speed scalable and that may be powered down. The load on the data center varies with time, and at each time the data center operator has to determine the number of servers that will be operational. The standard assumption is that there is some fixed cost for powering a server on, or powering the server off. Most naturally this cost incorporates the energy used for powering up or down, but this cost may incorporate ancillary terms such as the cost of the additional wear and tear on the servers. In response to a load L_t at time t, the data center operator decides on a number of servers x_t to use to handle this load. In a data center, there are typically sufficiently many servers so that this discrete variable can be reasonably be modeled as a continuous one. The data center operator pays a convex cost of $|x_{t-1} - x_t|$ for either powering-up or powering-down servers, and a cost of $F_t(x_t) = x_t((L_t/x_t)^\alpha + \beta)$ for handling the load, which is the most energy efficient way to service the load L_t using x_t processors, assuming the standard model for processor power. So this corresponds to an instance of OCO on a line, that is, where $k = 1$.

A general instance of OCO would arise from a setting where there are k different types of servers, and where each type of server would generally have different power characteristics. Then $x_{i,t}$ would represent the number of servers of type i that are powered on, and $F_t(x_{1,t}, \ldots, x_{k,t})$ would represent the minimum cost way to handle the load L_t using the number of specified numbers of each type (which is convex for all standard models of power).

4.1 Looking Backward

Theoretical work on OCO deals primarily with the case that $k = 1$. [48] gave a 3-competitive deterministic full-history algorithm. [4] showed that there is an algorithm with sublinear regret, but that $O(1)$-competitiveness and sublinear regret cannot be simultaneously achieved. [13] gave a 2-competitive algorithm, that is most easily understood as a randomized algorithm that maintains a probability distribution p over destinations. [13] also observed that any randomized algorithm for OCO can be converted to a deterministic algorithm without any loss of approximation. [13] also give a simple 3-competitive memoryless algorithm, which in the context of OCO means that the algorithm determines p_t based solely on p_{t-1} and F_t, and showed that this competitiveness is optimal for deterministic memoryless algorithms. Finally [13] gave a general lower bound of 1.86 on the competitiveness of any algorithm, which shows that in some sense this problem is strictly harder than classic online ski rental problem [22].

The OCO problem is a special case of the classic *metrical task system* problem, where both the metric space and the cost functions can be arbitrary. The optimal deterministic competitive ratio for a general metrical task system is $2n - 1$, where n is the number of points in the metric [23], and

the optimal randomized competitive ratio is $\Omega(\log n / \log\log n)$ [20,21], and $O(\log^2 n \log\log n)$ [30].

The Convex Body Chasing problem is a special case of OCO where the convex functions are restricted to be those that are zero within some convex region/body, and infinite outside that region. So in the Convex Body Chasing problem the algorithm sees a sequence of convex bodies, and must in response move to a destination within the last convex body. The objective is to minimize the total distance traveled. [31] observed that there is a lower bound of $\Omega(\sqrt{k})$ on the competitive ratio for Convex Body Chasing. Most of the upper bounds in the literature for Convex Body Chasing are for chasing certain special types of convex bodies. [31] gave a somewhat complicated algorithm and $O(1)$-competitiveness analysis for chasing lines in two dimensions, and observe that any $O(1)$-competitive line chasing algorithm for two dimensions can be extended to an $O(1)$-competitive line chasing algorithm for an arbitrary number of dimensions. [31] gave an even more complicated algorithm and $O(1)$-competitiveness analysis for chasing arbitrary convex bodies in two dimensions. [62] showed in a very complicated analysis that the work function algorithm is $O(1)$-competitive for chasing lines and line segments in any dimension. The (Generalized) Work Function Algorithm moves to the location p_t that minimizes a linear combination of $d(p_{t-1}, p_t)$ and $w_t(p_t)$, where $w_t(x)$ is is the cheapest cost to handle the first t requests and end in location x. [32] showed that the greedy algorithm is $O(1)$-competitive if $k = 2$ and the convex bodies are regular polygons with a constant number of sides.

[8] considered the relationship between OCO and Convex Body Chasing. [8] showed that an $O(1)$-competitive algorithm for Convex Body Chasing can be used to obtain an $O(1)$-competitive algorithm for OCO instances in which the convex functions have radial symmetry. This reduction is through an intermediate Lazy Convex Body Chasing problem, which is a special case of OCO in which the convex functions are zero within some convex region, and increase linearly as one moves away from this convex region. [8] also gave an online algorithm for Convex Body Chasing when the convex bodies are subspaces, in any dimension, and an $O(1)$-competitiveness analysis when $k = O(1)$. Finally [8] gave an online algorithm for chasing lines and line segments in any dimension, and show that it is $O(1)$-competitive. The underlying insight of this online algorithm is the same as in [31], to be greedy with occasional adjustments toward the area where the adversary might have cheaply handled recent requests. However, the algorithm and its analysis is cleaner/simpler than the algorithm in [31], as well as being essentially memoryless.

4.2 Open Problems

The two clear top open problems are:

Open Problem: Find an online algorithm for OCO that one can prove is $O(1)$-competitive when the number of dimensions is constant, which is a reasonable assumption in data center applications.

Open Problem: Find a provably $O(1)$-competitive algorithm for the special case of Convex Body Chasing. Convex Body Chasing is easier to think about, and one take away from our paper [8] is that probably OCO isn't too much harder than Convex Body Chasing.

Intuitively the main difficulty of obtaining a provably $O(1)$-competitive algorithm for Convex Body Chasing is balancing the competing needs of the algorithm remembering enough history to be able to be competitive, but keeping its consideration of history sufficiently simple to be analyzable. In some sense, most of the algorithms in the literature keep the consideration of history simple. These algorithms break the input into "phases", and then determine their new destination based only on the history within the current phase. And their analyses all have essentially the same form: after each "phase" in the input, either the location of the online algorithm has either moved closer to the adversary's location by an amount proportional to its cost in the phase, or the adversary is in some position where its costs are proportional to the online algorithm's costs. On the other one extreme is the Work Function algorithm, which uses all of the history in the most complete way imaginable. This makes it quite likely that the algorithm is $O(1)$-competitive, but also means that even for simple instances like line chasing, the analysis is quite involved, and we don't currently know how to analyze the Work Function algorithm for chasing more complex convex bodies.

To get some feel for the issues involved, let us consider two instances for halfspace chasing in three dimensions, which is the simplest setting where we don't know how to achieve $O(1)$-competitiveness. In the first instance, the halfspaces are rotating around a line L, say L is the z axis for the moment. So the halfspace arriving at time t (think of time as being continuous for the moment) would be $ax + by \geq 0$, where $a = \cos\ t$ and $b = \sin\ t$. Any $O(1)$-competitive algorithm would have to move quickly to the line L. From an algorithmic design viewpoint, this isn't a problem at all, as natural generalizations of the algorithms in the literature would move to L. But this instance is a bit of a problem from the standard algorithm analysis point of view. Observe that if the adversary's location is on the line L, but far from the online algorithm's location, then both the adversary's cost is low, and the algorithm isn't get closer to the adversary fast enough to offset its costs.

The second instance is a bit trickier. In this instance, as the halfspaces are again continuously rotating around the line L, but now the line L is simultaneously continuously spinning around the origin in such a way that the loci of points that L sweeps out is a cone. So L at time t could be the line where $0 = ax + by$ and $z^2 = x^2 + y^2$, where $a = \cos\ t$ and $b = \sin\ t$. Any $O(1)$-competitive algorithm would have to move quickly to the origin. But its a bit unclear how one could algorithmically recognize that the algorithm needs to move to the origin in a way that is both principled, and that uses the history in some sort of limited way.

5 Energy Efficient Routing

According to the US Department of Energy [1], data networks consume more than 50 billion kWH of energy per year, and a 40% reduction in wide-area network energy is plausibly achievable if network components were energy proportional.

5.1 Looking Back

Circuit routing, in which each connection is assigned a reserved route in the network with a guaranteed bandwidth, is used by several network protocols to achieve reliable communication [44]. [6] introduced the problem of routing circuits to minimize energy in a network of components that are speed scalable, and that may be shutdown when idle, using the standard models for circuit routing and component energy. The input consists of an undirected graph G, and a collection of source/sink vertex pairs. The output is a path, representing the circuit for a unit bandwidth demand, between each source/sink pair. The power used by an component with positive flow f is $\sigma + f^{\alpha}$, and the component is shutdown and consumes no power if it supports no flow. The objective is to minimize the aggregate power used over all the components. Primarily for reasons of mathematical tractability, the initial research assume that the speed scalable components are the edges.

The difficulty of these problems comes from the competing goals of minimizing static power, where it's best that flows are concentrated, and minimizing dynamic power, where it's best that the flows are spread out. A critical parameter is $q = \sigma^{1/\alpha}$. If the flow on an edge is at least q, then one knows that the dynamic power on that edge is at least the static power. [6] show that there is a limit to how well these competing demands can be balanced by showing that there is no polynomial-time algorithm with approximation ratio $o(\log^{1/4} n)$, under standard complexity theoretic assumptions. In contrast, [5] shows that these competing forces can be "poly-log-balanced" by giving a polynomial-time poly-log-approximation algorithm that uses several "big hammers" [27,40,59].

We have a line of papers [11,12,41] that significantly extend the results on this line of research. We started with [12], which considered the case of a common source vertex s for all request-pairs, that is all $s_i = s$. Applications for a common source vertex include data collection by base stations in a sensor network, and supporting a multicast communication using unicast routing. [12] gives a polynomial-time $O(1)$-approximation algorithm. The algorithm design and analysis is considerably easier than [5] because, after aggregation into groups, all the flow is going to the same place. [12] also gives an $O(\log^{2\alpha+1} k)$-competitive randomized online algorithm, by giving a procedure for forming the groups in an online fashion.

We then extended these ideas in [11], which contained two main results. The first main result was a polynomial-time $O(\log^{\alpha} k)$-approximation algorithm for the general problem. This algorithm consisted of 3 stages, each of which was essentially a combinatorial greedy algorithm:

Activating a Steiner Forest: The algorithm first activates a Steiner forest, to ensure minimal connectivity, using the standard $O(1)$-approximation algorithm for Steiner forest [66].

Activating the Hallucination Backbone: Then each request-pair, with probability $\Theta(\frac{\log k}{q})$ *hallucinates* that it wants to route q units of flow unsplittably on a path between its end points. This *hallucinated flow* is then routed using the natural greedy algorithm, all edges on which hallucinated flow is routed are then activated. (Note that no actual flow is routed here, this hallucinated flow is just used to determine which additional edges to activate).

Routing: The algorithm routes the flow on the activated edges using the natural greedy algorithm. The heart of the algorithm analysis is to appeal to the flow-cut gap for multicommodity flows [45,50] to show that there must be a low-congestion routing.

The second main result in [11] is a randomized $\tilde{O}(\log^{3\alpha+1} k)$-competitive online algorithm. The offline algorithm rather naturally extends to an online algorithm. The analysis however is considerably more involved than in the offline case. Since the edges are bought online, the analysis in [35] only shows that the dynamic power for the greedy algorithm is competitive against the power used in an optimal "*priority routing*", where a request-pair can only route over edges bought by the online algorithm up until the arrival time of the request-pair. Thus to mimic the analysis in the offline case, we need to show that there is a low-congestion *priority* multicommodity flow on the bought edges. This is accomplished by characterizing the notion of *sparsest priority-cuts*, and then bounding the priority flow-cut gap for multicommodity flows.

Finally in [41] we made a start on extending the results in previous papers to the case that the speed scalable components of the graph are the vertices, and not the edges. The main difficulty in emulating the approaches in the previous (edge based) papers is that they all relied on the fact that it is possible to aggregate flows in a (Steiner) tree in such a way that there is low edge congestion, but we would need that there is low node congestion, which is not possible in some trees, e.g. a star. To surmount this difficultly we showed how to efficiently find a low-cost collection of nearly node-disjoint trees that span all terminals, which can then be used to obtain an aggregation of flows with low vertex congestion.

5.2 The Open Problems

Open Problem: Find a poly-log approximation algorithm, when the speed scalable components are the edges, where the polynomial doesn't depend on α.

All of the analysis of the energy used by circuit routing protocols in [5,6,11,12,41] goes via congestion. To understand this consider the situation of randomly throwing n balls into n bins, where the resulting energy is the sum of the αth power of the bin sizes. An analysis in the spirit of the ones in [5,6,11,12,41] would argue that the resulting energy is within a $\tilde{O}(\log^{\alpha} n)$ factor of optimal because with high probability no bin has $\Omega(\log n)$ balls. As some bin likely has $\tilde{\Omega}(\log n)$ balls, this is the best bound one can obtain by only analyzing the fullest bin.

But in actuality, the energy used is $O(1)$ approximate to optimal because very few bins have loads near $\log n$. A competitive analysis of a poly-log competitive algorithm, where the polynomial doesn't depend on α, can not go via congestion, and would require reasoning about energy more directly. It is instructive to first see why the Hallucination algorithm won't work. The oversampling in the formation of the Hallucination backbone in [11] was required if the analysis was via congestion; otherwise there would likely be a cut without sufficient capacity to route all the flow across the cut with low congestion. But this oversampling meant that the hallucinated flow was a $\log n$ factor more than the actual flow, and thus increasing costs by a $\log^{\alpha} n$ factor. Thus one needs to be a bit more careful about how one oversamples. It seems that a new Hallucination algorithm, that is a modest tweak of the original Hallucination algorithm in [11] is a reasonable candidate algorithm. The hallucination backbone would be the union of $\log n$ sub-backbones. For each sub-backbone each source/sink hallucinates a flow of q with probability $\frac{1}{q}$, so there is no oversampling in a sub-backbone. Intuitively if the sub-backbones are (nearly) disjoint, then the static power would only be a $O(\log n)$ more than optimal, and there would still be sufficient capacity to route all flow with low dynamic power. The worry is that if all the sub-backbones had high overlap, then there would be insufficient capacity for a low energy routing. But for all the graphs that we can think of where the sub-backbones might overlap with some reasonable probability, it is the case that for these graphs oversampling is not required to obtain sufficient capacity for a low energy routing.

Open Problem: Find a poly-log competitive online algorithm when the speed scalable components are the vertices.

The obvious starting point is try to find a way to build online a low-cost collection of nearly node-disjoint trees that span all terminals.

6 Energy Efficient Circuit Design

The threshold voltage of a transistor is the minimum supply voltage at which the transistor starts to conduct current. However, if the designed supply voltage was exactly the ideal threshold voltage, some transistors would likely fail to operate as designed due to manufacturing and environmental variations. In the traditional approach to circuit design the supply voltages for each transistor/gate are set sufficiently high so that with sufficiently high probability no transistor fails, and thus the designed circuits need not be fault-tolerant. One potential method to attain more energy-efficient circuits is *Near-Threshold Computing*, which simply means that the supply voltages are designed to be closer to the threshold voltage. As the power used by a transistor/gate is roughly proportional to the square of the supply voltage [26], Near-Threshold Computing can potentially significantly decrease the energy used per gate. However, this energy savings comes at a cost of a greater probability of functional failure, which necessitates that the circuits must be more fault-tolerant, and thus contain more gates. As the

total energy used by a circuit is approximately the energy used per gate times the number of gates, achieving energy savings with Near-Threshold Computing involves properly balancing the energy used per gate with the number of gates used.

6.1 Looking Back

In [10] we initiated the theoretical study of the design of energy-efficient circuits. We assumed that the design of the circuit specifies both the circuit layout as well as the supply voltages for the gates. We assume a failure-to-energy function $P(\epsilon)$ that specifies the power required to insure the probability that a gate fails is at most ϵ. For current CMOS technologies, it seems that the "right" model for failure-to-energy function is $P(\epsilon) = \Theta(\log^2(1/\epsilon))$ [28]. For simplicity we assume this failure-to-energy function, but the theoretical results are not particularly sensitive to the exact nature of this function. We subsequently had three follow-up papers [9,18,19] on theoretical issues related to near-threshold computing.

[10] showed how to use techniques from the literature on fault-tolerant circuits to obtain bounds on circuit energy. [10] show that $\Omega\left(s \log\left(s(1-2\sqrt{\delta})/\delta\right)\right)$ energy is required by any circuit that computes a relation with sensitivity s correctly with probability at least $(1-\delta)$. [10] also showed, using techniques from [64] and [33,55], that a relation h that is computable by a circuit of size c can, with probability at least $(1-\delta)$, be computed by a circuit of faulty gates using $O(c \log(c/\delta))$ energy.

In [9] we considered the problem of: given a circuit C, an input I to C, and a desired circuit error bound δ, compute a supply voltage s that minimizes energy subject to the constraint that the error probability for the circuit is less than δ. The traditional approach/algorithm cranks up the supply voltage s until the error probability at each gate is δ/n, so that by the union bound the probability that the circuit is incorrect is at most δ. In [9] we observed that the traditional algorithm produces an $O(\log^2 n)$ approximation to the minimum energy. The main result in [9] is that is NP-hard to obtain an approximation ratio of $O(\log^{2-\epsilon} n)$. This shows that there are complexity theoretic barriers to systematically beating the traditional approach.

In [18] we showed that almost all Boolean functions require circuits that use exponential energy (foreshadowing slightly, this holds even if circuits can have heterogeneous supply voltages). This is not an immediate consequence of Shannon's classic result [61] that almost all functions require exponential sized circuits of faultless gates because (as we showed in [18]) the same circuit layout can compute many different functions, depending on the value of the supply voltage. The key step in the proof is to upper bound the number of different functions that one circuit layout can compute as the supply voltage changes.

While it may not currently be practical, in principle the supply voltages need not be homogeneous over all gates, that is, different gates could be supplied with different voltages. This naturally leads to the question of whether allowing heterogeneous supply voltages might yield lower-energy circuits than are possible

if the supply voltages are required to be homogeneous. While each of [9,10,18] touched on this question, and [19] squarely addressed this question. Intuitively, heterogeneous voltages should benefit a circuit where certain parts of the computation are more sensitive to failure than others. For example, in order for a circuit to be highly reliable, gates near the output need to be highly reliable. However, it may be acceptable for gates that are far from the output to be less reliable if there is sufficient redundancy in the circuit.

We considered four variations on the question, depending on

- whether what one is trying to compute, $f : \{0,1\}^n \to \{0,1\}$, is a function, or an injective relation (meaning one doesn't care what the output is on some inputs), and
- whether one wants the circuit to be correct with a fixed/constant probability, or with high probability (so the error decreases inverse polynomially as the input size increases).

For each of these four variations, we wanted to determine whether $\omega(1)$ energy savings is possible for all, none, or some of the f. It is relatively straight-forward to observe that the maximum possible energy savings due to allowing heterogeneity is $O(\log^2 n)$ in all cases. Our answers to date can be found in Table 1.

Table 1. Possible energy savings from heterogeneous supply voltages

	Circuit error $\delta = \Theta(1)$	Circuit error $\delta = \Theta\left(\frac{1}{\text{poly}(n)}\right)$
Functions	$\Theta(1)$ for some	$\Theta(\log n)$ for all with linear size circuits
Injective relations	$\Theta(1)$ for some	$\Omega(\log^2 n)$ for some
		$\Omega(\log n)$ for all with linear sized circuits

So one can see from the table that we have a reasonable understanding of when heterogeneity can save energy when computing with high probability, as long as the functions and relations that have linear sized circuits. Functions/relations with (near) linear sized circuits presumably include those that one is most likely to want to implement in hardware. When computing with high probability, heterogeneous circuits can save a $\Theta(\log n)$ factor of energy for all functions with linear sized circuits, and an additional $\log n$ factor can be saved for some relations with linear sized circuits.

6.2 Open Problems

There are myriad open problems in this area, but if I had to pick one as the best open problem, it would be the following:

Open Problem: Does there exist a function (or injective relation) $f : \{0,1\}^n \rightarrow \{0,1\}$ where there is a heterogeneous circuit that computes f with constant error using an $o(1)$ factor less energy than any homogeneous circuit that computes f with constant error?

We know that heterogeneity can give at most constant energy savings for computing some functions with constant error. Not surprisingly one such function is the parity function as intuitively every part of any reasonable circuit for parity is equally highly sensitive to error. And there are examples of relations where heterogeneity helps the obvious circuit. In [10] we showed that the most obvious circuit, a tree of majority gates, to compute the super-majority relation can be made $o(1)$ more energy efficient by turning down the supply voltage near the input gates. But there is a less obvious circuit to compute super-majority [63], that seemingly can not be made more energy efficient with heterogeneous supply voltages.

This seems to be asking a fundamental question about computation. Is it true for it is always the case in the most energy efficient circuit to compute some function with constant error probability that essentially every part of the circuit is equally sensitive to error? (Recall that we know that this answer is no when computing with high probability.) If the answer is always yes for computing with constant probability then we known that heterogeneous circuits will not yield any energy savings for computing with constant error probability.

7 Online Scheduling of Power Heterogeneous Processors

The new architectural designs lead to a plethora of natural algorithmic scheduling problems related to balancing the competing demands of performance and energy efficiency. The natural resulting algorithmic management problems involve determining which task to run on each processor, and for speed scalable processors at what speed to run that task, so as to obtain a (near) optimal tradeoff between the conflicting objectives of quality of service and energy efficiency.

There enough literature in this area to justify an independent survey. So in the interest of space, I will just discuss one result, which at least arguably is the culmination of this research line for the common processor models. We showed in [37] that there is a scalable algorithm, which we called SelfishMigrate, for scheduling unrelated machines to minimize a (user/application specified) linear combination of energy and weighted job delay (here some jobs are more important and contribute more to this total). The result holds even when each machine has an arbitrary convex speed-to-power function, and processors may be shutdown (and thus consume no energy). An algorithm is scalable if it guarantees that the objective value is within a constant factor of optimal for processors that run slightly slower at each power level. An algorithm is nonclairvoyant if does not know the size of a job. The algorithm does however need to know the suitability of each processor for each job, or more formally, the rate that each processor can process each job. Its easy to see that such knowledge is necessary to achieve any reasonable approximation. We had previously shown in [34] that the

standard priority algorithms, like Highest Density First (HDF), that one finds in standard operating systems textbooks, can perform quite badly on heterogeneous processors when the quality of service objective is weighted delay [34]. So we knew we were not going to be able to obtain scalability with a known standard algorithm.

The SelfishMigrate algorithm can be best viewed in a game theoretic setting where jobs are selfish agents, and machines declare their scheduling policies in advance. Each machine maintains a virtual queue on the current set of jobs assigned to it; newly arriving jobs are appended to the tail of this queue. Each machine treats a migration of a job to it as an arrival, and a migration out of it as a departure. This means a job migrating to a machine is placed at the tail of the virtual queue. Each job j has a virtual utility function, which roughly corresponds to the inverse of the instantaneous weighted delay introduced by j to jobs ahead of it in its virtual queue, and their contribution to j's weighted delay. Using these virtual utilities, jobs perform sequential best response dynamics, migrating to machines (and get placed in the tail of their virtual queue) if doing so leads to larger virtual utility. Therefore, at each time instant, the job migration achieves a Nash equilibrium of the sequential best response dynamics on the virtual utilities. The analysis is via dual fitting, and involves showing that the Nash dynamics on virtual utilities directly corresponds to our setting of dual variables being feasible. In hindsight, we believe this framework is the right way to generalize the greedy dispatch rules and dual fitting analysis from previous works [3,36].

This result suggests that perhaps that heterogeneous multiprocessors should be scheduled very differently than the way that uniprocessors and homogeneous multiprocessors have been scheduled.

7.1 Open Problem

Open Problem: Show that the standard priority scheduling algorithms are scalable for the objective of total flow time when scheduling on related processors.

The standard priority scheduling algorithms that one finds in introductory operating systems texts, e.g. Shortest Remaining Processing Time (SRPT), Shortest Job First (SJF), Shortest Elapsed Time First (SETF), Multi-Level Feedback (MLF), are all known to be scalable for the objective of total delay on one processor and on identical processors [38,56,57]. Given that it is often difficult to get new policies/protocols adopted, it would be good to know how bad things can get for these standard scheduling policies on heterogeneous processors. That is, are these algorithms scalable for total delay on related machines. Intuitively I see no reason to think that they are not scalable. The intuition why the standard algorithms are not scalable for weighted delay is that if one has multiprocessor with many slow processors and few fast processors then it can be difficult to harness the aggregate speed of the slow processors. Somehow it seems that, if jobs are of equal importance, this is not such an issue. The standard potential function and dual fitting approaches don't seem immediately applicable as there doesn't seem to be a simple algebraic expression for

the contribution of a particular job towards the objective. So it seems that some innovation in algorithm analysis will be required.

8 Understanding Optimal Energy Tradeoff Schedules

8.1 Looking Back

Another line of my research related to speed scalable processors is to understand optimal energy-performance tradeoff schedules, primarily by finding efficient algorithms to compute them. We initiated this line of research in [58] by giving a polynomial time algorithm for scheduling jobs that have a common size with an objective of minimizing a linear combination of total delay and energy. In [15] we considered the problem of scheduling arbitrary sized jobs with the objective of minimizing a linear combination of fractional weighted delay and energy, and showed how to recognize an optimal schedule. In [7] we gave a polynomial time algorithm for scheduling arbitrary sized jobs with the objective of minimizing a linear combination of fractional weighted delay and energy. The algorithm in [7] can be viewed as a primal-dual algorithm that raises the dual variables in an organized way. In [16] we considered the setting of a sensor that consists of a speed-scalable processor, a battery, and a solar cell that harvests energy from its environment at a time-invariant recharge rate. The processor must process a collection of jobs of various sizes. Jobs arrive at different times and have different deadlines. The objective is to minimize the *recharge rate*, which is the rate at which the device has to harvest energy in order to feasibly schedule all jobs. The main result was a polynomial-time combinatorial algorithm for processors with a natural set of discrete speed/power pairs. The main takeaway form this paper was that it is much harder to reason about energy when it is supplied over time instead of all being initially available.

One can formulate many different optimization problems depending on how one models the processor (e.g., whether allowed speeds are discrete or continuous, and the nature of relationship between speed and power), the performance objective (e.g., whether jobs are of equal or unequal importance, and whether one is interested in minimizing waiting times of jobs or of work), and how one handles the dual objective (e.g., whether they are combined in a single objective, or whether one objective is transformed into a constraint). In [17] we finally bit the bullet, and determined the complexity of a reasonably full landscape of all the possible formulations.

One commonality of the algorithms that we developed in [7,16,58] is that they all can be viewed as continuous homotopic optimization algorithms. These homotopic algorithms trace the evolution of the optimal solution as either the objective function evolves from one, where it is simple to compute the optimal, to the desired objective, or the constraints evolve from ones, that are easy to satisfy, to the desired constraints. So one take away point from these results is that homotopic optimization can be a fruitful approach for computing and understanding optimal tradeoff schedules.

8.2 Open Problem

The clear top open problem in this area, which was left open in [17] is:

Open Problem: Determine the complexity of finding optimal schedule for the objective of total delay plus energy on a speed scalable processor.

In the optimal schedule it must be the case that at all times the job being processed is the one that has the least amount of work left to be processed. But this is of less help than it might first appear, as this doesn't help decide what speed the processor should run, and thus how much work should be left on this job at the next time step. The main difficulty of extending the algorithms in [7,58] is that the optimal total delay plus energy schedule for arbitrary work jobs is more fragile than if the jobs had unit size or the quality of service objective was fractional total delay. Still, many of the insights in [7,58] carry over. In particular, the power of a job should generally be proportional to the number of jobs that depend on that job. As a consequence of this, if one knew the ordering of the release times and completion times, obtaining an optimal schedule subject to such an ordering constraint is straightforward. So my intuition at this point says that this problem should be solvable in polynomial time. But I don't have any real idea how to touch this problem. I find it a bit surprising that even computing an optimal schedule when there are only 2 speeds seems hard. I think it would be fine to get an algorithm whose running time is polynomial in the number of jobs, but possibly exponential in the number of speeds, as usually the number of speeds is usually on the order of 5 to 10. However, it is certainly not inconceivable that there is no polynomial-time algorithm for this problem, say because the problem is NP-hard. There is no proof in the literature showing the hardness a speed scaling problem where the hardness somehow arose nontrivally from the speed scaling aspect of the problem. So a hardness proof would be interesting, as SRPT is optimal for a fixed speed processor, and its not clear where the hardness would come for a speed scalable processor.

9 Conclusion

Some characteristics of the theory of energy as a computational resource that has developed over the last decade are:

- Most problems arise at a lower layer of the information technology stack that is not aware of the exact nature of the computation taking place on the high layer.
- Most problems involve managing some mechanism/technology created/installed to achieve greater energy efficiency.
- Most problems involve balancing dual objectives, one related to energy, and one related to performance. Often of these objectives is implicit as it has been turned into a constraint.

As a consequence, the current theory energy as a computational resource is less concentrated, and less distinct from other research areas, than the theory of time/space as a computational resource were in their first decade.

References

1. Vision and roadmap: routing telecom and data centers toward efficient energy use. In: Proceedings of Vision and Roadmap Workshop on Routing Telecom and Data Centers Toward Efficient Energy Use, May 2009
2. Albers, S.: Energy-efficient algorithms. Commun. ACM **53**(5), 86–96 (2010)
3. Anand, S., Garg, N., Kumar, A.: Resource augmentation for weighted flow-time explained by dual fitting. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 1228–1241 (2012)
4. Andrew, L.L.H., Barman, S., Ligett, K., Lin, M., Meyerson, A., Roytman, A., Wierman, A.: A tale of two metrics: simultaneous bounds on competitiveness and regret. In: Conference on Learning Theory, pp. 741–763 (2013)
5. Andrews, M., Antonakopoulos, S., Zhang, L.: Minimum-cost network design with (dis)economies of scale. In: IEEE Symposium on Foundations of Computer Science, pp. 585–592 (2010)
6. Andrews, M., Fernández, A., Zhang, L., Zhao, W.: Routing for energy minimization in the speed scaling model. In: INFOCOM, pp. 2435–2443 (2010)
7. Antoniadis, A., Barcelo, N., Consuegra, M., Kling, P., Nugent, M., Pruhs, K., Scquizzato, M.: Efficient computation of optimal energy and fractional weighted flow trade-off schedules. In: Symposium on Theoretical Aspects of Computer Science (2014)
8. Antoniadis, A., Barcelo, N., Nugent, M., Pruhs, K., Schewior, K., Scquizzato, M.: Chasing convex bodies and functions. In: Kranakis, E., Navarro, G., Chávez, E. (eds.) LATIN 2016. LNCS, vol. 9644, pp. 68–81. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49529-2_6
9. Antoniadis, A., Barcelo, N., Nugent, M., Pruhs, K., Scquizzato, M.: Complexity-theoretic obstacles to achieving energy savings with near-threshold computing. In: International Green Computing Conference, pp. 1–8 (2014)
10. Antoniadis, A., Barcelo, N., Nugent, M., Pruhs, K., Scquizzato, M.: Energy-efficient circuit design. In: Innovations in Theoretical Computer Science, pp. 303–312 (2014)
11. Antoniadis, A., Im, S., Krishnaswamy, R., Moseley, B., Nagarajan, V., Pruhs, K., Stein, C.: Energy efficient virtual circuit routing. In: ACM-SIAM Symposium on Discrete Algorithms (2014)
12. Bansal, N., Gupta, A., Krishnaswamy, R., Nagarajan, V., Pruhs, K., Stein, C.: Multicast routing for energy minimization using speed scaling. In: Even, G., Rawitz, D. (eds.) MedAlg 2012. LNCS, vol. 7659, pp. 37–51. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34862-4_3
13. Bansal, N., Gupta, A., Krishnaswamy, R., Pruhs, K., Schewior, K., Stein, C.: A 2-competitive algorithm for online convex optimization with switching costs. In: Workshop on Approximation Algorithms for Combinatorial Optimization Problems, pp. 96–109 (2015)
14. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. J. ACM **54**(1), 3 (2007)
15. Barcelo, N., Cole, D., Letsios, D., Nugent, M., Pruhs, K.: Optimal energy trade-off schedules. Sustain. Comput.: Inf. Syst. **3**, 207–217 (2013)
16. Barcelo, N., Kling, P., Nugent, M., Pruhs, K.: Optimal speed scaling with a solar cell. In: Chan, T.-H.H., Li, M., Wang, L. (eds.) COCOA 2016. LNCS, vol. 10043, pp. 521–535. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48749-6_38

17. Barcelo, N., Kling, P., Nugent, M., Pruhs, K., Scquizzato, M.: On the complexity of speed scaling. In: Italiano, G.F., Pighizzini, G., Sannella, D.T. (eds.) MFCS 2015. LNCS, vol. 9235, pp. 75–89. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48054-0_7
18. Barcelo, N., Nugent, M., Pruhs, K., Scquizzato, M.: Almost all functions require exponential energy. In: Italiano, G.F., Pighizzini, G., Sannella, D.T. (eds.) MFCS 2015. LNCS, vol. 9235, pp. 90–101. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48054-0_8
19. Barcelo, N., Nugent, M., Pruhs, K., Scquizzato, M.: The power of heterogeneity in near-threshold computing. In: International Green and Sustainable Computing Conference, pp. 1–4 (2015)
20. Bartal, Y., Bollobás, B., Mendel, M.: Ramsey-type theorems for metric spaces with applications to online problems. J. Comput. Syst. Sci. **72**(5), 890–921 (2006)
21. Bartal, Y., Linial, N., Mendel, M., Naor, A.: On metric Ramsey-type phenomena. In: ACM Symposium on Theory of Computing, pp. 463–472 (2003)
22. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
23. Borodin, A., Linial, N., Saks, M.E.: An optimal on-line algorithm for metrical task system. J. ACM **39**(4), 745–763 (1992)
24. Bower, F.A., Sorin, D.J., Cox, L.P.: The impact of dynamically heterogeneous multicore processors on thread scheduling. IEEE Micro **28**(3), 17–25 (2008)
25. Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J.-D., Zyuban, V., Gupta, M., Cook, P.W.: Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors. IEEE Micro **20**(6), 26–44 (2000)
26. Butts, J.A., Sohi, G.S.: A static power model for architects. In: ACM/IEEE International Symposium on Microarchitecture, pp. 191–201 (2000)
27. Chekuri, C., Khanna, S., Shepherd, F.B.: Multicommodity flow, well-linked terminals, and routing problems. In: ACM Symposium on Theory of Computing, pp. 183–192 (2005)
28. Dreslinski, R.G., Wieckowski, M., Blaauw, D., Sylvester, D., Mudge, T.N.: Near-threshold computing: reclaiming Moore's law through energy efficient integrated circuits. Proc. IEEE **98**(2), 253–266 (2010)
29. Esmaeilzadeh, H., Blem, E.R., Amant, R.S., Sankaralingam, K., Burger, D.: Dark silicon and the end of multicore scaling. IEEE Micro **32**(3), 122–134 (2012)
30. Fiat, A., Mendel, M.: Better algorithms for unfair metrical task systems and applications. SIAM J. Comput. **32**(6), 1403–1422 (2003)
31. Friedman, J., Linial, N.: On convex body chasing. Discret. Comput. Geom. **9**, 293–321 (1993)
32. Fujiwara, H., Iwama, K., Yonezawa, K.: Online chasing problems for regular polygons. Inf. Process. Lett. **108**(3), 155–159 (2008)
33. Gács, P.: Reliable computation. In: Algorithms in Informatics, vol. 2. ELTE Eötvös Kiadó, Budapest (2005)
34. Gupta, A., Im, S., Krishnaswamy, R., Moseley, B., Pruhs, K.: Scheduling heterogeneous processors isn't as easy as you think. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 1242–1253 (2012)
35. Gupta, A., Krishnaswamy, R., Pruhs, K.: Online primal-dual for non-linear optimization with applications to speed scaling. In: Erlebach, T., Persiano, G. (eds.) WAOA 2012. LNCS, vol. 7846, pp. 173–186. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38016-7_15

36. Im, S., Kulkarni, J., Munagala, K.: Competitive algorithms from competitive equilibria: non-clairvoyant scheduling under polyhedral constraints. In: Symposium on Theory of Computing, pp. 313–322 (2014)
37. Im, S., Kulkarni, J., Munagala, K., Pruhs, K.: Selfishmigrate: a scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In: Symposium on Foundations of Computer Science, pp. 531–540 (2014)
38. Im, S., Moseley, B., Pruhs, K.: A tutorial on amortized local competitiveness in online scheduling. SIGACT News **42**(2), 83–97 (2011)
39. Irani, S., Pruhs, K.: Algorithmic problems in power management. SIGACT News **36**(2), 63–76 (2005)
40. Khandekar, R., Rao, S., Vazirani, U.V.: Graph partitioning using single commodity flows. J. ACM **56**(4), 19 (2009)
41. Krishnaswamy, R., Nagarajan, V., Pruhs, K., Stein, C.: Cluster before you hallucinate: approximating node-capacitated network design and energy efficient routing (2014)
42. Kumar, R., Tullsen, D.M., Jouppi, N.P.: Core architecture optimization for heterogeneous chip multiprocessors. In: International Conference on Parallel Architectures and Compilation Techniques, pp. 23–32. ACM (2006)
43. Kumar, R., Tullsen, D.M., Ranganathan, P., Jouppi, N.P., Farkas, K.I.: Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. SIGARCH Comput. Archit. News **32**(2), 64 (2004)
44. Kurose, J.F., Ross, K.W.: Computer Networking: A Top-Down Approach. Addison-Wesley Publishing Company, Boston (2009)
45. Leighton, F.T., Rao, S.: Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. J. ACM **46**(6), 787–832 (1999)
46. Lin, M., Liu, Z., Wierman, A., Andrew, L.L.H.: Online algorithms for geographical load balancing. In: International Green Computing Conference, pp. 1–10 (2012)
47. Lin, M., Wierman, A., Andrew, L.L.H., Thereska, E.: Online dynamic capacity provisioning in data centers. In: Allerton Conference on Communication, Control, and Computing, pp. 1159–1163 (2011)
48. Lin, M., Wierman, A., Andrew, L.L.H., Thereska, E.: Dynamic right-sizing for power-proportional data centers. IEEE/ACM Trans. Netw. **21**(5), 1378–1391 (2013)
49. Lin, M., Wierman, A., Roytman, A., Meyerson, A., Andrew, L.L.H.: Online optimization with switching cost. SIGMETRICS Perform. Eval. Rev. **40**(3), 98–100 (2012)
50. Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some of its algorithmic applications. Combinatorica **15**(2), 215–245 (1995)
51. Liu, Z., Lin, M., Wierman, A., Low, S.H., Andrew, L.L.H.: Greening geographical load balancing. In: ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 233–244 (2011)
52. Merritt, R.: CPU designers debate multi-core future. EE Times, February 2008
53. Morad, T.Y., Weiser, U.C., Kolodny, A., Valero, M., Ayguade, E.: Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors. IEEE Comput. Archit. Lett. **5**(1), 4 (2006)
54. Mudge, T.: Power: a first-class architectural design constraint. Computer **34**(4), 52–58 (2001)
55. Pippenger, N.: On networks of noisy gates. In: Symposium on Foundations of Computer Science, pp. 30–38 (1985)

56. Pruhs, K.: Competitive online scheduling for server systems. In: Special Issue of SIGMETRICS Performance Evaluation Review on New Perspectives in Scheduling, no. 4 (2007)
57. Pruhs, K., Sgall, J., Torng, E.: Online scheduling. In: Handbook of Scheduling: Algorithms, Models, and Performance Analysis (2004)
58. Pruhs, K., Uthaisombut, P., Woeginger, G.J.: Getting the best response for your erg. ACM Trans. Algorithms **4**(3), 38:1–38:17 (2008)
59. Rao, S., Zhou, S.: Edge disjoint paths in moderately connected graphs. SIAM J. Comput. **39**(5), 1856–1887 (2010)
60. Sergent, J.E., Krum, A.: Thermal Management Handbook. McGraw-Hill, New York (1998)
61. Shannon, C.E.: The synthesis of two-terminal switching circuits. Bell Syst. Tech. J. **28**, 59–98 (1949)
62. Sitters, R.: The generalized work function algorithm is competitive for the generalized 2-server problem. SIAM J. Comput. **43**(1), 96–125 (2014)
63. Valiant, L.G.: Short monotone formulae for the majority function. J. Algorithms **5**(3), 363–366 (1984)
64. von Neumann, J.: Probabilistic logics and the synthesis of reliable organisms from unreliable components. In: Shannon, C.E., McCarthy, J. (eds.) Automata Studies, pp. 329–378. Princeton University Press, Princeton (1956)
65. Wang, K., Lin, M., Ciucu, F., Wierman, A., Lin, C.: Characterizing the impact of the workload on the value of dynamic resizing in data centers. In: IEEE INFOCOM, pp. 515–519 (2013)
66. Williamson, D.P., Shmoys, D.B.: The Design of Approximation Algorithms. Cambridge University Press, Cambridge (2011)