# A theoretical framework for algorithm-architecture co-design

Kenneth Czechowski, Richard Vuduc
*School of Computational Science and Engineering*
*Georgia Institute of Technology, Atlanta, Georgia*
{*kentcz,richie*}*@gatech.edu*

***Abstract*—We consider the problem of how to enable computer architects and algorithm designers to reason directly and analytically about the relationship between high-level architectural features and algorithm characteristics. We propose a modeling framework designed to help understand the long-term and high-level impacts of algorithmic and technology trends. This model connects abstract communication complexity analysis—with respect to both the inter-core and inter-processor networks and the memory hierarchy—with current technology proposals and projections. We illustrate how one might use the framework by instantiating a particular model for a class of architectures and sample algorithms (three-dimensional fast Fourier transforms, matrix multiply, and three-dimensional stencil). Then, as a suggestive demonstration, we analyze a number of what-if scenarios within the model in light of these trends to suggest broader statements and alternative futures for power-constrained architectures and algorithms.**

## I. Introduction

We seek a formal framework that explicitly relates characteristics of an algorithm, such as its inherent parallelism or memory behavior, with parameters of an architecture, such as the number of cores, structure of the memory hierarchy, or network topology. Our ultimate goal is to say precisely and analytically how high-level changes to the architecture might affect the execution time, scalability, accuracy, and power-efficiency of a computation; and, conversely, identify what classes of computation might best match a given architecture. Our approach marries abstract algorithmic complexity analysis with key physical constraints, such as caps on power and die area, that will be critical in the extreme scale systems of 2018 and beyond [1, 35]. We refer to our approach as one of *algorithm-architecture co-design*.

We say "algorithm-architecture" rather than "hardware-software," so as to evoke a high-level mathematical process that precedes and complements traditional methods based on detailed architecture simulation of concrete benchmark code artifacts and traces [9, 23, 27, 30, 48, 51]. Our approach takes inspiration from prior work on high-level performance analysis and modeling [3, 26–28, 44], as well as the classical theory of circuit models and the area-time trade-offs studied in models based on very large-scale integration (VLSI) [37, 49]. Our analysis is in many ways most similar to several recent theoretical exascale modeling studies [22, 47], combined with trends analysis [34]. However, our specific methods return to higher-level I/O-centric complexity analysis [4, 5, 10, 13, 21, 54], pushing it further by trying to resolve analytical constants, which is necessary to connect abstract complexity measures with the physical constraints imposed by power and die area. This approach necessarily will *not* yield cycle-accurate performance estimates, and that is not our aim. Rather, our hope is that a principled algorithmic analysis that accounts for major architectural parameters will still yield interesting insights and suggest new directions for improving performance and scalability in the long run.

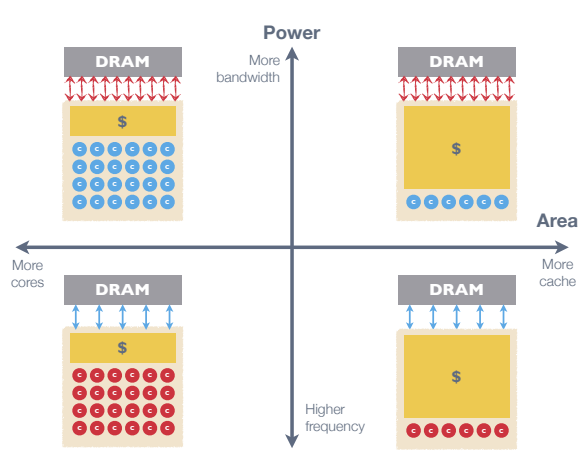
**A formal framework.** We pose the formal co-design problem as follows. Let $a$ be an algorithm from a set $A$ of algorithms that all perform the same computation within the same desired level of accuracy. The set $A$ might contain different algorithms, such as "$A = \{$fast Fourier transform, F-cycle multigrid$\}$," for the Poisson model problem [17, 41]. Or, $A$ may be a set of tuning parameters for one algorithm, such as the set of tile sizes for matrix multiply. Next, let $\mu$ be a machine architecture from a set $M$, and suppose that each processor of $\mu$ has an area of $\chi(\mu)$. Lastly, let $T(n; a, \mu)$ be the time to execute $a$ on $\mu$ for a problem of size $n$, while using a maximum instantaneous power of $\Phi(\mu)$. Then, our goal is to determine the algorithm $a$ and architecture $\mu$ that minimize time subject to constraints on total power and processor die area, e.g.,

$$(a^*, \mu^*) = \operatorname*{argmin}_{(a \in A,\ \mu \in M)} T(n; a, \mu) \quad (1)$$

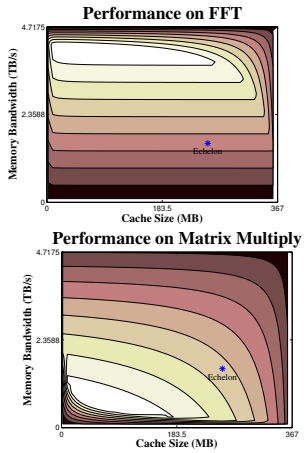
subject to

$$\Phi(\mu^*) = \Phi_{\max} \quad (2)$$

$$\chi(\mu^*) = \chi_{\max}, \quad (3)$$




(a) A notional power/transistor allocation problem




(b) Projected performance for a 3D FFT and matrix multiply at some problem size (lighter is better)

Figure 1: (a) In our framework, a fixed die area (allocated between cores and cache) and a fixed power budget (allocated between core frequency and bandwidth), define a space of possible machines. (b) Different algorithms may perform differently on these machines. The marker is the approximate "location" within this space of the NVIDIA Echelon GPU-like architecture proposed for the year 2017 [32]. In the 3D FFT example, the optimal configuration is 2.6 times faster than Echelon.

where $\Phi_{\max}$ and $\chi_{\max}$ are caps on power and die area, respectively. The central research problem is to determine the form of $T(n; a, \mu)$, $\Phi(\mu)$, and $\chi(\mu)$. The significance and novelty of this analysis framework is that it explicitly binds characteristics of algorithms and architectures, Equation (1), with physical hardware constraints, Equations (2)–(3).

**A demonstration.** Suppose we wish to design a manycore processor $\mu$, which we represent by the four-tuple $(\beta_{\text{mem}}, q, f, Z)$: $\beta_{\text{mem}}$ is the processor-memory bandwidth (words per unit time), $q$ is the number of cores per processor, $f$ is the clock frequency of each core (cycles per unit time), and $Z$ is the total size of the aggregate on-chip cache (in words), assuming just a two-level hierarchy (cache and main memory). Further suppose that the $\chi_{\max} = 141$ mm$^2$ of die area can be divided between on-chip cache ($Z$) and cores ($q$). Lastly, suppose the node power budget is constrained to $\Phi_{\max} = 173$ Watts, which can be used to increase cycle-frequency ($f$) or boost off-die memory bandwidth ($\beta_{\text{mem}}$). Figure 1a is a cartoon that suggests how these parameters and constraints imply a space of possible designs. Figure 1b shows how, given a specific model of different algorithms on this space of machines, we might then solve the optimization problem of Equations 1–3 to identify optimal systems. Unsurprisingly, a processor tuned for a communication-intensive 3D fast Fourier transform will devote more of a fixed power budget (y-axis) to memory bandwidth, compared to matrix multiply.

However, Figure 1b also suggests an intriguing possibility. Observe that a die area configuration (x-axis) that is good for matrix multiply will also be good for an FFT; to make a system that can perform "optimally" on both workloads, we would need the ability to dynamically *shift* power from the processor to memory bandwidth, by a large factor of roughly 7×. That is, reconfigurability of processor transistors may be relatively less important than extreme power reconfigurability with respect to bandwidth. Whether one can build such a system is a separate question; this demonstration suggests and attempts to quantify the possibility.

The remainder of this paper formalizes this analysis. As a demonstration, we develop an analytical model of these constraints, $\Phi(\mu)$ and $\chi(\mu)$, as well as performance models, $T(n; a, \mu)$. To suggest the possibilities of the framework, we develop models for a full-system configuration, consisting of a distributed memory machine comprising any number of manycore processors connected by a network, in the case of distributed matrix multiply, distributed 3D FFTs, and distributed stencil algorithms. We do not view any specific models and projections as the main contribution of our work. Rather, we wish to emphasize the basic framework, with the large variety of potential detailed modeling strategies, analyses, and projections as possibilities based on

it.

## II. Background and Related Work

The principal challenge is how to connect power (and implicitly, energy) and area constraints with a complexity analysis. There are numerous approaches. The most widely-cited come from the computer architecture community [11, 19, 25, 26, 40, 57]. In such approaches, the application or algorithm is typically abstracted away through an Amdahl's Law style analysis, which means it can be difficult to relate high-level algorithmic characteristics to architectures precisely. Theorists have also considered a variety of new complexity models that incorporate energy as an explicit cost [36, 42]. However, this body of work is very abstract and focused purely on asymptotics, making these models difficult to operationalize, in our view. Lastly, there is a considerable body of work from the embedded hardware/software community, emphasizing analysis suitable for compiler- and run-time systems [15, 33, 53]. However, this work necessarily focuses on specific concrete code and architecture implementations, and therefore do not explicitly illuminate constraints due to fundamental algorithmic and physical limits. It is these limits that we seek to understand to make forecasts about future algorithm and system behavior.

Regarding area constraints, note that our framework treats die area, $\chi(\mu)$, as a function of architecture $\mu$ only. Thus, to construct this function we need to consider what architectural components we will put on chip (e.g., cores, caches, on-chip networks) and derive cost estimates for the size of each component. Effectively, this choice implies that we are most interested in still building architectures that can execute general-purpose computations; however, the power and area allocations are tuned to specific workloads. Thus, our approach stands in contrast to the classical work on models of computation under very large-scale integration (VLSI) [49, Chap. 12]. That body of work also considers physical area-time tradeoffs [37, 45], with connections to energy [2, 55], but does so for specific circuit structures that correspond to specific computations. We imagine a bridging between these two approaches but are for the moment agnostic on the specific analytical form of that bridge.

To develop cost models for both power and area, we are mining the vast literature on models and technology trends for everything from low-level processor device physics, functional units, cores, caches and memory systems, and on-chip and off-chip networks [6, 7, 16, 24, 31, 35, 38, 50]. Since our ultimate goal is to consider potential long-term outcomes, we focus on recent projections of scaling trends [6, 32, 34, 35, 50].

## III. An Example of Instantiating a Model within the Framework

This section explains how one might go about constructing meaningful cost and constraint models within the framework. In particular, we instantiate specific forms for $T(n; a, \mu)$, $\Phi(\mu)$, and $\chi(\mu)$. These forms are meant to be illustrative, not necessarily definitive. Armed with such a model, Section IV then considers a variety of what-if scenarios at exascale (roughly 1 exaflop/s capable systems) expected in the 2017–2020 timeframe, to illustrate the kinds of insights possible within the framework.

### A. Technological and architectural parameters

To guide parameter selection and model calibration, we start with the basic technology trend assumptions laid out in a recent description of the proposed NVIDIA Echelon processor, scheduled for release in approximately the year 2017 [32].

For our subsequent analysis and projections, we will assume the technology constants that appear in Table I. These values depend on specific assumptions about technological capabilities in the 2017–2020 time frame, for which we borrow the expectations used to design Echelon. We take those projections as-is; debates about the accuracy of these values are beyond the scope and intention of this paper.

Our target system is a supercomputer. We parameterize the high-level architecture $\mu$ by the following: the number of cores per processor ($q$), cycle-frequency of each core ($f$), the aggregate on-chip cache capacity ($Z$), memory bandwidth ($\beta_{\text{mem}}$), on-chip network bandwidth ($\beta_{\text{noc}}$), off-chip network link bandwidth ($\beta_{\text{net}}$), and total number of nodes ($p$). By "node" we really mean a single unit of distributed memory processing consisting of a (manycore) processor, local memory, persistent storage (disk). This definition constitutes a simplification of how a real "node" might look in a future system; however, we do model the power required by such a node (see $\epsilon_{\text{node}}$ in Table I). We use the term "core" to represent the basic unit of processing in the system. We endow a core with general-purpose processing capabilities, e.g., ALU, address generation, branch unit, register file; however, because the sample algorithms we analyze

are floating-point intensive, we characterize the core essentially by its peak rate of floating-point instructions completed per cycle. The caches are core-private but the value $Z$ is aggregate over all cores on a chip. We connect cores via a $\sqrt{q} \times \sqrt{q}$ 2D Mesh with an on-chip link bandwidth of $\beta_{\text{noc}}$. The on-chip cache is evenly distributed to all cores so that each cores has a private cache of size $\frac{Z}{q}$. We connect nodes by a $p^{\frac{1}{3}} \times p^{\frac{1}{3}} \times p^{\frac{1}{3}}$ 3D torus with a link bandwidth of $\beta_{\text{net}}$. As a reference point, the values of these parameters as proposed for the Echelon design appear in column 2 of Table II. (Columns 3 and 4 will be discussed in Section IV.)

Our specific algorithmic analyses will also assume an abundance of parallelism. Though this assumption seems strong, it is also a necessary condition for any application that can hope to scale to very large numbers of nodes relative to today's standards.

| Parameters | | 2018 (10 nm) Value |
|---|---|---|
| System Power Cap: | $\Phi_{\text{max}}$ | 20 MW |
| Chip Die Area: | $\chi_{\text{max}}$ | 141.7 mm$^2$ |
| Cache Density: | $\sigma_{\text{cache}}$ | .386 mm$^2$/MB |
| Core Density: | $\sigma_{\text{core}}$ | .0105 mm$^2$/MB |
| Memory BW Power: | $\lambda_{\text{mem}}$ | 36 mW/$\frac{\text{GB}}{\text{s}}$ |
| Network BW Power: | $\lambda_{\text{net}}$ | 36 mW/$\frac{\text{GB}}{\text{s}}$ |
| On-Chip Network BW Energy: | $\lambda_{\text{noc}}$ | .75 pJ/mm Byte |
| Dynamic Power Coefficient: | $c_{\text{dynamic}}$ | 0.00129704 |
| Short-Circuit Coefficient: | $c_{\text{short}}$ | 0.0032426 |
| Leakage Coefficient: | $c_{\text{leakage}}$ | 0.002026625 |
| Node Overhead: | $\epsilon_{\text{node}}$ | 2 W |

Table I: Technology Constants: Projected values for 2018.

### B. A model of physical constraints

We use the following models of power and area, based on the parameters shown in Tables I and II.

The total system power comprises the power of the cores ($P_{\text{comp}}$), memory ($P_{\text{mem}}$), on-chip interconnect ($P_{\text{noc}}$), and the system network ($P_{\text{net}}$). There is also a nominal per node power cost ($\epsilon_{\text{node}}$) that represents the inherent overhead cost of maintaining a node, e.g., power supply, chipset, disk.

$$\Phi(\mu) \quad = \quad p\left(P_{\text{comp}} + P_{\text{mem}} + P_{\text{noc}} + \epsilon_{\text{node}}\right) + P_{\text{net}} \quad (4)$$

Power consumption of CMOS circuits are frequently modeled with a three term equation of the form, $P = ACV^2f + \tau AVIf + VI_{\text{leakage}}$, which accounts for the dynamic power consumption, short-circuit current, and leakage current [43]. The key variables from our perspective are voltage $V$ and frequency $f$. Since the maximum operating frequency $f$ is roughly linearly related to the voltage $V$, we can simplify the expression for the power consumption of each core to be a function that is cubic in $f$:

$$P_{\text{comp}} \quad = \quad q\left(c_{\text{dynamic}}f^3 + c_{\text{short}}f^2 + c_{\text{leakage}}f\right). \quad (5)$$

We can obtain suitable coefficients by fitting this equation to the different voltage, frequency, and energy settings provided in the Echelon paper [32].

Bandwidth power (Joules/sec or Watts) is bandwidth (Bytes / sec) times energy cost per byte (Joules/Byte):

$$P_{\text{mem}} \quad = \quad \beta_{\text{mem}} \cdot \lambda_{\text{mem}} \quad (6)$$

$$P_{\text{net}} \quad = \quad \beta_{\text{net}} \cdot \text{Links}(p) \cdot \lambda_{\text{net}} \quad (7)$$

The die area dedicated to each core is $\frac{Z}{q}\sigma_{\text{cache}} + \sigma_{\text{core}}$. Assuming each core is square in shape, the distance between the center of two neighboring cores is $\sqrt{\frac{Z}{q}\sigma_{\text{cache}} + \sigma_{\text{core}}}$, which we will use to approximate the length of the on-chip interconnect links. Assuming a 2D mesh topology, which is the most natural considering the current planar manufacturing process of modern processors—there are a total of $4q - 4\sqrt{q}$ links. The power of the on-chip network is therefore

$$P_{\text{noc}} = (4q - 4\sqrt{q})\left(\sqrt{\frac{Z}{q}\sigma_{\text{cache}} + \sigma_{\text{core}}}\right)\beta_{\text{noc}}\lambda_{\text{noc}}. \quad (8)$$

The limited processor die area constrains the number of cores and cache that can be placed on a single node. A larger cache capacity means there is less space for cores and vice-versa. Thus, given a total die area of $\Omega_{\text{die}}$, we constrain total core and cache capacity by requiring that

$$\Omega_{\text{die}} \quad = \quad (q \cdot \sigma_{\text{core}}) + (Z \cdot \sigma_{\text{cache}}). \quad (9)$$

### C. Algorithmic cost models

Given the basic architectural model and parameters, the next step is to analyze an algorithm or class of algorithms, so that we can compute $T(n; a, \mu)$. We specifically consider the total execution time to be the maximum of four component times:

$$T \quad = \quad \max\left\{T_{\text{comp}}, T_{\text{net}}, T_{\text{mem}}, T_{\text{noc}}\right\} \quad (10)$$

where $T_{\text{comp}}$ is the time performing compute (flops), $T_{\text{net}}$ is the time spent in network communication, $T_{\text{mem}}$ is the time spent performing processor-memory communication, and $T_{\text{noc}}$ is the time spent in on-chip network communication. Below, we give sample analyses for 2.5D matrix multiply, a three-dimensional FFT, and a stencil computation.

| Parameters | | Echelon Value | Ideal Matrix Multiply Value | Ideal FFT Value | Ideal Stencil Value |
|---|---|---|---|---|---|
| Cores: | $q$ | 4,096 | 11,491 | 11,295 | 9,494 |
| Frequency: | $f$ | 2.0 GHz | .275 GHz | 3.0 GHz | .280 |
| Aggregate On-chip Cache: | $Z$ | 256 MB | 54.5 MB | 59.8 MB | 109 MB |
| Memory Bandwidth: | $\beta_{\text{mem}}$ | 1.6 TB/s | .034 TB/s | 29.5 TB/s | .787 TB/s |
| On-chip Network Bandwidth: | $\beta_{\text{noc}}$ | 4.0 GB/s | .11 GB/s | 38.2 GB/s | .19 GB/s |
| Network Bandwidth: | $\beta_{\text{net}}$ | 67 GB/s | 11.4 GB/s | 18,100 GB/s | 12.5 GB/s |
| Number of nodes: | $p$ | 102,500 | 1,280,000 | 3,400 | 480,000 |
| Peak: | $2qfp$ | 1.7 EF/s | 8.1 EF/s | 230 PF/s | 2.5 EF/s |

Table II: Hardware Characteristics ($\mu$).

*1) Example: Distributed 2.5D matrix multiply:* The 2.5D matrix multiply algorithm of Solominik and Demmel [52] is a particularly interesting case for our framework. In particular, it contains a tuning parameter that can be used to reduce communication at the cost of increasing memory capacity, a trade-off that we subsequently analyze.

The 2.5D matrix multiplication algorithm decomposes a $n \times n$ matrix multiply, distributed across $p$ nodes, into a sequence of $(p/C)^{3/2}$ matrix multiplies, each of size $(n\sqrt{C/p}) \times (n\sqrt{C/p})$. The value $C$ is the tuning parameter that, when increased, decreases communication at the cost of increased (replicated) storage.

We take computation time to be that of the conventional (non-Strassen) algorithm:

$$T_{\text{comp}} = \frac{2n^3}{pqf}. \tag{11}$$

Network communication costs are based on the asymptotically optimal bandwidth costs of the 2.5D algorithm [52]:

$$T_{\text{net}} = \frac{2n^2}{\sqrt{Cp}\beta_{\text{net}}}. \tag{12}$$

Each node will compute $(p^{1/2}/C^{3/2})$ local matrix multiplies of size $\left(n\sqrt{\frac{C}{p}}\right) \times \left(n\sqrt{\frac{C}{p}}\right)$ during the computation. From this fact, we can calculate the time spent locally transferring data between the processor and memory, given the aggregate cache of size $Z$. Assuming an asymptotically I/O-optimal blocked algorithm with block size $b = \sqrt{\frac{Z}{3}}$, we obtain

$$T_{\text{mem}} = \left(\frac{\sqrt{p}}{C^{\frac{3}{2}}}\right)\left(\frac{n\sqrt{C}}{\sqrt{p}}\right)^3\left(\frac{2\sqrt{3}}{\sqrt{Z}\beta_{\text{mem}}}\right) \tag{13}$$

$$= \frac{2\sqrt{3}n^3}{p\sqrt{Z}\beta_{\text{mem}}}. \tag{14}$$

Since we assume private caches and a 2D mesh network, we can treat the local matrix multiply as a distributed computation across the cores. We estimate the on-chip network communication time assuming the communication-optimal Cannon algorithm [8],

$$T_{\text{noc}} = \left(\frac{\sqrt{p}}{C^{\frac{3}{2}}}\right)\left(\frac{2n^2C}{p\sqrt{q}\beta_{\text{noc}}}\right) = \frac{2n^2}{\sqrt{Cpq}\beta_{\text{noc}}}, \tag{15}$$

where the constants reflect the additional assumption of the matrix operands being distributed in "skew" order across the private caches.

*2) Example: Distributed 3D FFTs:* Our second example is the 3D FFT. We assume a problem of size $N = n^3$ using the pencil decomposition [39]. The algorithm consists of three computation phases separated by two communication phases. Each computation phase computes $n^2$ 1D FFTs of size $n$ in parallel. Each communication phase involves $\sqrt{P}$ independent $P$-node personalized all-to-all exchanges.

Each 1D FFT of size $n$ is computed locally on a node using $\Theta(n \log n)$ floating point operations. We assume the classic Cooley-Tukey radix-2 algorithm, which requires approximately $5n \log_2 n$ flops; thus,

$$T_{\text{comp}} = 3 \times \frac{5n^3 \log_2 n}{fpq}. \tag{16}$$

During each of the two communication phases, approximately $n^3$ data points are transferred across the network. Assuming a 3D torus network with a bisection bandwidth of $\mathcal{O}\left(p^{\frac{2}{3}}\beta_{\text{net}}\right)$, the communication cost is approximately

$$T_{\text{net}} = 2 \times \frac{n^3}{p^{\frac{2}{3}}\beta_{\text{net}}}. \tag{17}$$

During each of the three computation phases, each node must compute $\frac{n^2}{p}$ 1D FFTs. The number of processor-memory transactions necessary to compute each local 1D FFT depends on the total cache capacity $Z$ of the node. If the entire 1D FFT can fit within the on-chip caches ($n \leq Z$), then memory transfers are limited to just $\mathcal{O}(n)$

compulsory cache misses. Otherwise, the computation will incur an additional $\Theta(n \log_Z n)$ capacity misses [29]. We have previously estimated a lower-bound on this constant to be 2.5 [14], using hardware counters measurements of last-level cache misses incurred by the highly-tuned FFTW [20]. Thus,

$$T_{\text{mem}} = 3 \times \frac{n^2}{p} \times \frac{2.5n \max(\log_Z n, 1)}{\beta_{\text{mem}}} \quad (18)$$

$$= \frac{7.5n^3 \max(\log_Z n, 1)}{p\beta_{\text{mem}}}, \quad (19)$$

where the max function ensures that the transfers include at least the compulsory misses.

If an entire 1D FFT can fit in the private cache of a single core ($n < \frac{Z}{q}$), then no on-chip communication is necessary beyond the compulsory cache misses to DRAM. Otherwise, the 1D FFT must be distributed across $\hat{q} = \frac{nq}{Z}$ cores, which requires $\mathcal{O}\left(\frac{n}{\sqrt{\hat{q}}\beta_{\text{noc}}}\right)$ time assuming a 2D mesh topology. In total, each group must compute at least $\frac{n^3}{pZ}$ of these distributed FFTs. Additionally, we only consider large problems sizes ($q \ll n$) in this paper, so that load balance should not be a significant factor. Thus,

$$T_{\text{noc}} = 3 \times \left(\frac{n^3}{pZ}\right)\left(\frac{n\sqrt{Z}}{\beta_{\text{noc}}\sqrt{q}\sqrt{n}}\right) \quad (20)$$

$$= \frac{3n^3\sqrt{n}}{p\sqrt{q}\sqrt{Z}\beta_{\text{noc}}}. \quad (21)$$

*3) Example: Distributed 3D Stencil:* Our final example is a 3D stencil. For simplicity of demonstration, we will only consider a 3D cross-shaped stencil of width $w$ and total of $6w + 1$ points, and we will ignore the possibility of algorithms that tile in time. We assume a problem of size $N = n^3$.

The most direct method consists of $12w + 1$ floating point operations per element; thus,

$$T_{\text{comp}} = \frac{n^3(12w+1)}{fpq}. \quad (22)$$

Assuming each node owns a $\frac{n}{\sqrt[3]{p}} \times \frac{n}{\sqrt[3]{p}} \times \frac{n}{\sqrt[3]{p}}$ block of the dataset, each node will need a $\frac{n}{\sqrt[3]{p}} \times \frac{n}{\sqrt[3]{p}} \times w$ sub-block from each of the six adjacent nodes on the 3D torus network. Therefore, the communication cost is approximately

$$T_{\text{net}} = \frac{6wn^2}{p^{\frac{2}{3}}\beta_{\text{net}}}. \quad (23)$$

Without reuse, the number of processor-memory transactions necessary to compute each element is $6w + 2$. A cache of size $Z$ can be used to reduce the total number of reads by a factor of $\mathcal{O}\left(Z^{\frac{1}{3}}\right)$. Thus,

$$T_{\text{mem}} = \left(\frac{n^3}{p}\right)\left(\frac{6w}{Z^{\frac{1}{3}}} + 2\right)\left(\frac{1}{\beta_{\text{mem}}}\right), \quad (24)$$

where $\frac{n^3}{p}$ is the number of elements processed on each node.

We can approximate the amount of on-chip communication by comparing the cache misses incurred by a core with a private cache of size $\frac{Z}{q}$ with the number of cache misses incurred by a processor with an aggregate cache of size $Z$. Thus,

$$T_{\text{noc}} = \frac{\left(\frac{n^3}{p}\right)\left[\left(\frac{6w}{\left(\frac{Z}{q}\right)^{\frac{1}{3}}} + 2\right) - \left(\frac{6w}{Z^{\frac{1}{3}}} + 2\right)\right]}{q\beta_{\text{noc}}} \quad (25)$$

$$= \frac{6wn^3\left(q^{\frac{1}{3}} - 1\right)}{Z^{\frac{1}{3}}qp\beta_{\text{noc}}}. \quad (26)$$

## IV. Analysis

Given the models and architectural parameters of Section III, we can now carry out a series of what-if analyses to gain some insight into possible futures and high-level architectural designs, and even compute solutions to Equation 1.

### A. Ideal architectures

We solved the optimization problem of Equation 1 for the 3D FFT, matrix multiply, and stencil algorithms. For this first analysis, we fixed the matrix multiply algorithm to be the Cannon (2D) algorithm, rather than the 2.5D algorithm considered in the next section, and fix the stencil width ($w = 10$). The ideal configuration for each appears in columns 3-5 of Table II. Think of these configurations as being the ones optimally tuned for the corresponding algorithm, though recall that the model is for a general-purpose system. Figure 2 shows how resources are allocated in each of these tuned configurations, as well as in the proposed Echelon configuration. Figure 3 shows execution times for each of the hypothetical machines on the 3D FFT, stencil, and matrix multiply workloads. We can make a number of observations about these results.

Even under optimistic assumptions and an optimal machine configuration, the ideal FFT machine has a peak of only 230 petaflop/s (PF/s) with 20 MW of power, which is just 1/36 of the 8 exaflop/s (EF/s) possible on the ideal matrix multiply system. This means that relative to a matrix multiplication, the FFT computation requires 36×

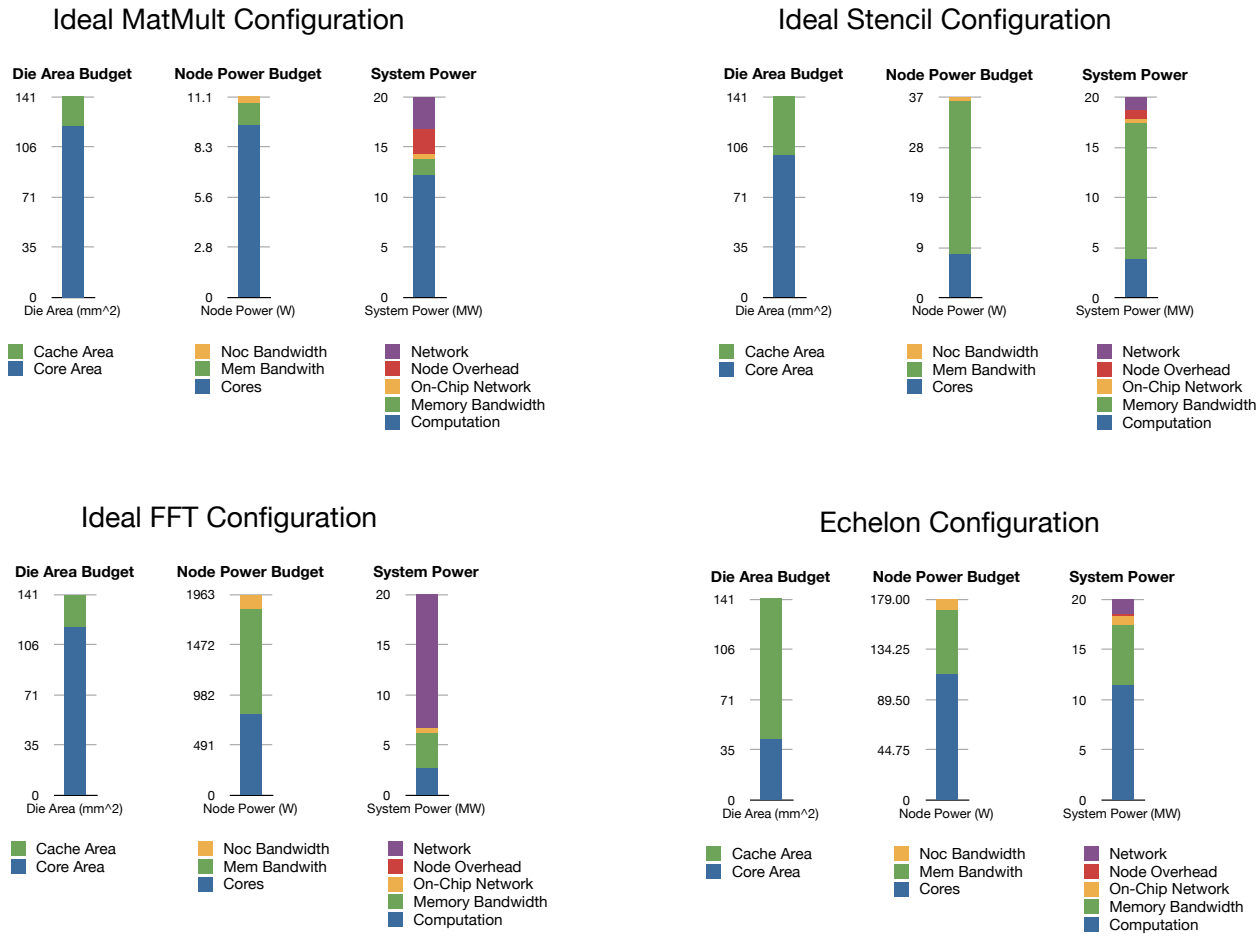



Figure 2: Hardware configurations for the hypothetical machines. The subplots break down the power and die area resource allocations.

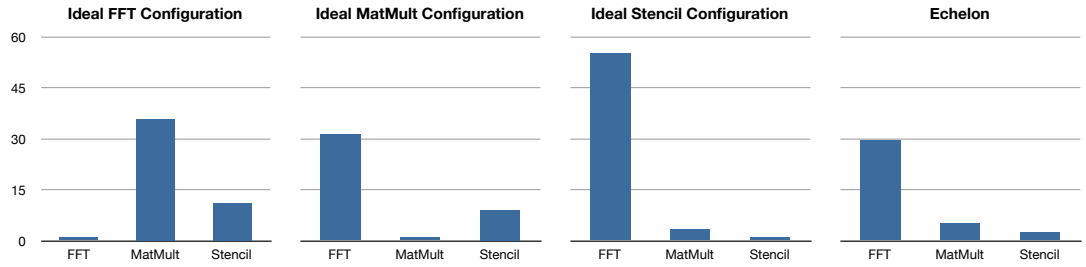



Figure 3: Relative execution times for the hypothetical machines. The subplots show execution time relative to the ieal FFT, Stencil, and MatMult configurations.

more energy per floating-point operation. However, tuning for a 3D FFT means we will necessarily divert power resources (and, therefore, energy efficiency) elsewhere in the system.

The Echelon design calls for 256 MB of on-chip cache which is over four times more than the ideal FFT and ideal MatMult. This is interesting because relative to NVIDIA's current GPU, the core count increased by a factor of 16 (the scaling factor from a 40nm to 10nm process technology) but the cache capacity by 64 (4x more than the scaling factor).

It is interesting to further consider these configurations in light of our motivating demonstration of Section I. There, we observed that a single-processor system with extreme reconfigurability of *power*—rather than die area—*might* lead to designs capable of performing both a compute-intensive matrix multiply and a communication-intensive

3D FFT. The three ideal configurations, depicted in Figure 2 and enumerated in Table II, are similarly suggestive. From Figure 2, observe that the processor configurations for the three ideal systems are not too dissimilar. Rather, the dramatic differences come from shifting power allocations to processors (Ideal MatMult), memory bandwidth (Ideal Stencil), or network (Ideal FFT). The node counts also different significantly (Table II). Thus, an intriguing question is whether there is any way to engineer a single system having the same processors but a mechanism to perform dramatic power reconfigurations (drawing down or shutting off nodes as needed, and diverting power to bandwidth).

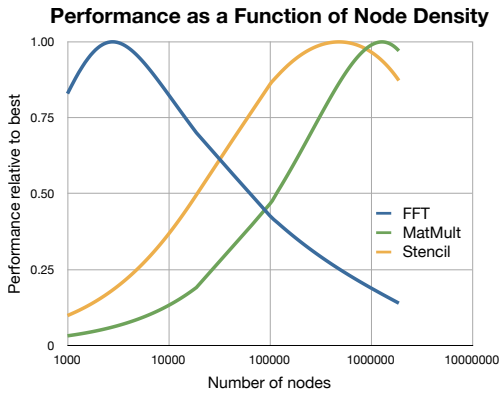



Figure 4: Node Density Plots.

### B. Architecture trade-offs: lightweight vs. heavyweight designs

Recent discussions surrounding the direction of high-end systems often characterize design strategies as either "lightweight" or "heavyweight." The key distinction between these two strategies is node density. Lightweight designs, exemplified by the Blue Gene-style processors, consist of many lower power processors. Alternatively, heavyweight designs, exemplified by Jaguar-class machines, consists of fewer but more powerful processors, each operating at high clock frequencies.

Interestingly, these characterizations apply to the ideal machine configurations in Figure 2. The ideal matrix multiply configuration, "Ideal MatMult," reflects a lightweight strategy, whereas the ideal 3D FFT configuration, "Ideal FFT," resembles a heavyweight strategy. The difference is extreme: Ideal FFT has only 3,400 nodes, which is 376× fewer nodes than Ideal MatMult. Essentially, an FFT is communication-bound and is therefore highly sensitive to the decreased energy-efficiency of a large network — on a 3D torus, the energy-efficiency will decrease at a rate of $\mathcal{O}\left((p^{2/3})/p\right)$ as $p$ increases. Unlike the FFT, matrix multiply benefits from large node counts because it can exploit the increased core count to make the computation more efficient with a lower clock frequency. Figure 4 shows the stark contrast in performance between the two computations as a function of system density.

### C. Algorithm trade-offs: computation v. communication

A convolution is an algorithmic pattern that can capture the characteristics of many scientific computing problems. We may interpret a convolution as the application of a "filter" to a "signal." Computationally, a convolution may be implemented as a stencil computation, when the filter is compact, or alternatively by an FFT. A small value of the stencil width $w$ in our model (Section III-C3) corresponds to our measure of compactness. In a stencil-based approach, the computational complexity of convolution will be $\mathcal{O}(wn)$. When the filter is not compact, an FFT-based method may be more suitable as it reduces computational complexity to $\mathcal{O}(n \log n)$. However, the FFT-based methods have a higher communication cost. For moderate sized stencils this results in a computation versus communication trade-off: the FFT-based method requires fewer floating point operations but more data movement than the stencil method [12, 18].

In the case of a large 3D convolution ($n = 2^{17}$), the FFT-based method will in our model become more efficient when $w \geq 22$. Figure 5 compares the execute time of the stencil and FFT-based methods over various stencil sizes. The results show that an ideal stencil machine is actually much faster than the ideal FFT machine until the stencil size is significantly larger ($w \approx 600$) than expected. The reason for the discrepancy is the relative cost of a floating-point operation versus data movement. As expected trading communication for computation is beneficial until the trade-offs become extreme.

### D. Algorithm trade-offs: space vs communication

In Section IV-A, we found the ideal architecture for Cannon's matrix multiply algorithm (the "2D" approach). While this algorithm is asymptotically optimal when all of the available system memory is utilized, the class of "2.5D" algorithms can further reduce network communication by a factor of $\sqrt{C}$ by making an additional $C$ copies of the data. Based on historical trends, we estimate that in 2018 increasing the system memory capacity

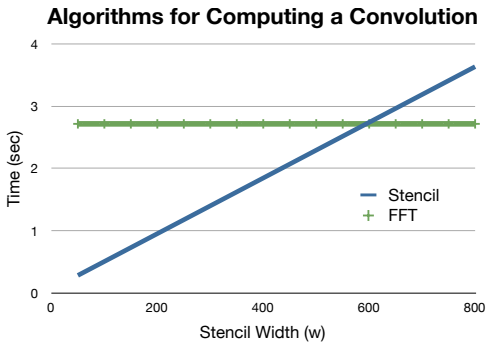



Figure 5: Plot of the time to compute a convolution using different stencil sizes. The figure compares the Stencil method with an FFT based method in the 3D case.

could require an extra watt of power for every eight GB of additional DRAM [46, 56]. Thus, the power consumed by the requisite memory capacity, $3Cn^2$ nanowatts, increases as $C$ increases. This introduces a trade-off between memory utilization and network communication. An interesting question is to what extent replication can be used to improve performance without increasing the total power and energy costs.

To find the optimal balance, we solve Equation 1 for the optimal algorithm, $a^*$, and the corresponding architecture, $\mu^*$, considering the set of all 2.5D implementations (i.e., values of the replication factor, $C$). Figure 6 shows the performance and resource allocation of these algorithms. The results show that indeed, replication can slightly improve upon Cannon's algorithm under these conditions. However, the optimal balance is not at one of the extremes, but rather somewhere in-between.

### E. Increasing the power budget

The U.S. Department of Energy, one of the primary customers of top-tier supercomputers, has instituted a strict 20 MW power cap for future supercomputers. This power constraint is one of the dominant challenges to reaching exascale.

We can consider how much performance improves if the power cap is relaxed a little by changing the power constraint, $\Phi_{\text{max}}$, in Equation 1. Figure 7 compares the performance of an 'ideal' machine, designed for a 20 MW power budget, with a similar machine that is designed for a larger power budget. As the figure shows, communication-heavy applications like the FFT will improve at a slower rate than applications that are less dependent on communication. This is because the most efficient way to scale the machine is to increase the number of nodes, which in turn increases the communication overheads.

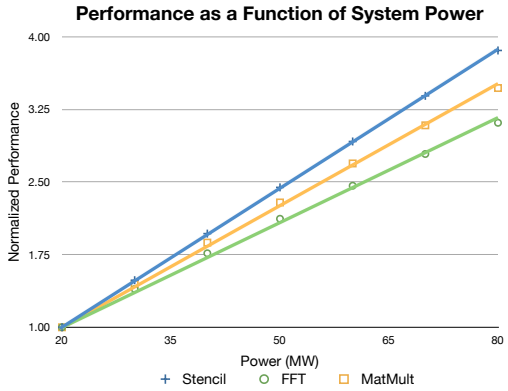



Figure 7: Plot of performance as a function of the power budget. Performance values for the algorithms are scaled to their performance at a 20 MW power budget.

## V. Conclusion

The ultimate goal of this work is to determine whether alternative ways to allocate die area and power might lead to better designs for specific applications. Our proposed framework suggests how to achieve this goal in a way that retains analytical rigor while remaining sufficiently high-level to yield insight. The suggestion of extreme power reconfigurability in the architecture as a mechanism that might support diverse applications is one forward-looking example.

Having said that, we emphasize the methodological contribution of our work over any specific quantitative predictions. The intent of this paper is to suggest a different way of thinking about the high-level directions for algorithms and architectures to drive the subsequent detailed co-design process, rather than to provide a comprehensive critique of exascale designs. We hope that a much deeper exploration of our basic setup—including more architectural details and consideration of richer classes of algorithms and mixed workloads—could produce a number of alternative futures, for both architecture and algorithm design.

## VI. Acknowledgements

We thank Mark Adams for comments. This work was supported in part by the National Science Foundation (NSF) under NSF CAREER award number 0953100; the U.S. Dept. of Energy (DOE) under award DE-FC02-10ER26006/DE-SC0004915; and grants from the Defense Advanced Research

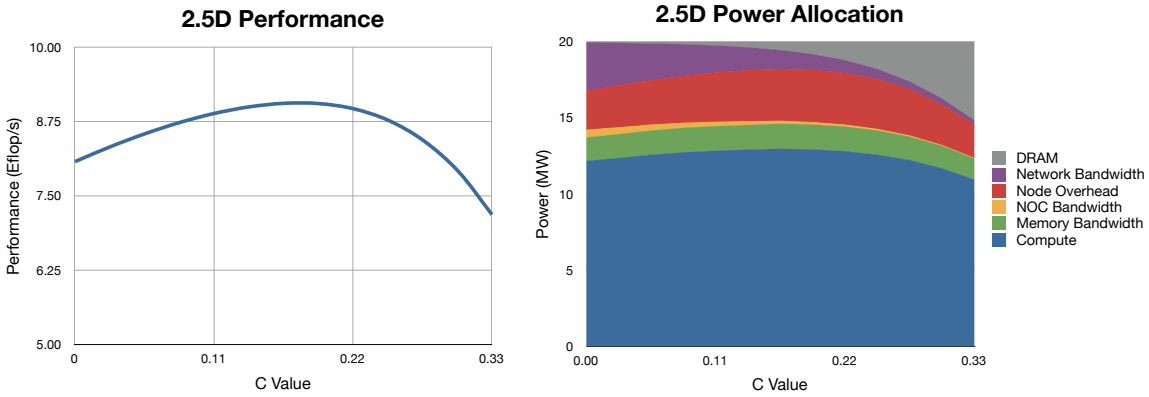



Figure 6: Performance of 2.5D Matrix Multiply variants. The 2.5D algorithm is parameterized by a value $C = p^{\alpha}$ where $0 \leq \alpha \leq \frac{1}{3}$. 'DRAM' represents the power consumed by the requisite memory capacity.

Projects Agency (DARPA) Computer Science Study Group program. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of NSF, DOE, or DARPA.

## References

[1] *The Potential Impact of High-End Capability Computing on Four Illustrative Fields of Science and Engineering*. The National Academies Press, Washington, DC, USA, 2008.

[2] A. Aggarwal, A. Chandra, and P. Raghavan. Energy consumption in VLSI circuits. In *Proceedings of the twentieth annual ACM symposium on Theory of computing - STOC '88*, pages 205–216, New York, New York, USA, 1988. ACM Press.

[3] K. J. Barker, A. Hoisie, and D. J. Kerbyson. An early performance analysis of POWER7-IH HPC systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11*, page 1, New York, New York, USA, 2011. ACM Press.

[4] G. E. Blelloch. Programming parallel algorithms. *Communications of the ACM*, 39(3):85–97, March 1996.

[5] G. E. Blelloch, P. B. Gibbons, and H. V. Simhadri. Low depth cache-oblivious algorithms. In *Proc. ACM Symp. Parallel Algorithms and Architectures (SPAA)*, Santorini, Greece, June 2010.

[6] S. Borkar and A. A. Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67, May 2011.

[7] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. Int'l. Symp. Computer Architecture (ISCA)*, pages 83–94, Vancouver, British Columbia, Canada, June 2000.

[8] L. E. Cannon. *A cellular computer to implement the Kalman filter algorithm*. PhD thesis, Montana State University, 1969.

[9] M. Casas, R. M. Badia, and J. Labarta. Prediction of behavior of MPI applications. In *2008 IEEE International Conference on Cluster Computing*, pages 242–251. IEEE, Sept. 2008.

[10] R. A. Chowdhury, F. Silvestri, B. Blakeley, and V. Ramachandran. Oblivious algorithms for multicores and network of processors. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–12. IEEE, 2010.

[11] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai. Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs ? In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 225–236, Atlanta, GA, USA, 2010.

[12] R. Clapp, H. Fu, and O. Lindtjorn. Selecting the right hardware for reverse time migration. *The leading edge*, 29(1):48–58, 2010.

[13] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. *ACM SIGPLAN Notices*, 28(7):1–12, July 1993.

[14] K. Czechowski, C. McClanahan, C. Battaglino, K. Iyer, P.-K. Yeung, and R. Vuduc. On the

communication complexity of 3D FFTs and its implications for exascale. In *Proc. ACM Int'l. Conf. Supercomputing (ICS)*, San Servolo Island, Venice, Italy, June 2012. (*to appear*).

[15] P. D'Alberto, M. Püschel, and F. Franchetti. Performance/energy optimization of DSP transforms on the XScale processor. In *Proc. High Performance Embedded Architectures and Compilers (HiPEAC)*, volume LNCS 4367, pages 201–214, Ghent, Belgium, January 2007.

[16] W. J. Dally, J. Balfour, D. Black-Shaffer, J. Chen, R. C. Harting, V. Parikh, J. Park, and D. Sheffield. Efficient Embedded Computing. *Computer*, 41(7):27–32, 2008.

[17] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

[18] M. Ernst. Serializing parallel programs by removing redundant computation. Master's thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1992.

[19] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the End of Multicore Scaling. In *Proceedings of the 28th International Symposiumn Computer Architecture (ISCA)*, San Jose, CA, USA, 2011.

[20] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proc. IEEE: Special issue on "Program Generation, Optimization, and Platform Adaptation"*, 93(2):216–231, 2005.

[21] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. Symp. Foundations of Computer Science (FOCS)*, pages 285–297, New York, NY, USA, October 1999.

[22] H. Gahvari, A. H. Baker, M. Schulz, U. M. Yang, K. E. Jordan, and W. Gropp. Modeling the performance of an algebraic multigrid cycle on HPC platforms. In *Proceedings of the international conference on Supercomputing - ICS '11*, page 172, New York, New York, USA, 2011. ACM Press.

[23] J. Gonzalez, J. Gimenez, M. Casas, M. Moreto, A. Ramirez, J. Labarta, and M. Valero. Simulating Whole Supercomputer Applications. *IEEE Micro*, 31(3):32–45, May 2011.

[24] N. Hardavellas, M. Ferdman, A. Ailamaki, and B. Falsafi. Power Scaling: the Ultimate Obstacle to 1K-Core Chips Power Scaling : the Ultimate Obstacle to 1K-Core Chips. *Northwestern University, Technical Report NWU-EECS-10-05*, pages 1–23, 2010.

[25] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward Dark Silicon in Servers. *IEEE Micro*, 31(4):6–15, July 2011.

[26] M. D. Hill and M. R. Marty. Amdahl's Law in the multicore era. *IEEE Computer*, 41(7):33–38, July 2008.

[27] T. Hoefler, T. Schneider, and A. Lumsdaine. LogGOPSim: Simulating large-scale applications in the LoGOPS model. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC '10*, page 597, New York, New York, USA, 2010. ACM Press.

[28] A. Hoisie, G. Johnson, D. J. Kerbyson, M. Lang, and S. Pakin. A performance comparison through benchmarking and modeling of three leading supercomputers: Blue Gene/L, Red Storm, and Purple. In *Proc. ACM/IEEE Conf. Supercomputing (SC)*, number 74, Tampa, FL, USA, November 2006.

[29] J.-W. Hong and H. Kung. I/O complexity: The red-blue pebble game. In *Proc. ACM Symp. Theory of Computing (STOC)*, pages 326–333, Milwaukee, WI, USA, May 1981.

[30] H. Jagode, A. Knupfer, J. Dongarra, M. Jurenz, M. S. Mueller, and W. E. Nagel. Trace-based performance analysis for the petascale simulation code FLASH. *International Journal of High Performance Computing Applications*, Dec. 2010.

[31] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *Proceedings of the Design, Automation, and Test in Europe (DATE) Conference & Exhibition*, volume 29, pages 1–6. IEEE, 2009.

[32] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco. GPUs and the Future of Parallel Computing. *IEEE Micro*, 31(5):7–17, Sept. 2011.

[33] W. Kirschenmann and L. Plagne. Optimizing computing and energy performances on GPU clusters: Experimentation on a PDE solver. In M. Pierson and H. Hlavacs, editors, *Proceedings of the COST Action IC0804 on Large Scale Distributed Systems*, pages 1–5. IRIT, 2010.

[34] P. Kogge and T. Dysart. Using the top500 to trace and project technology and architecture trends. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 28. ACM, 2011.

[35] P. Kogge et al. Exascale Computing Study: Technology challenges in acheiving exascale systems, September 2008.

[36] V. A. R. Korthikanti and G. Agha. Analysis of

parallel algorithms for energy conservation in scalable multicore architectures. In *Proc. Int'. Conf. Parallel Processing (ICPP)*, Vienna, Austria, September 2009.

[37] H. Kung. Let's design algorithms for VLSI systems. In *Proceedings of the Caltech Conference on VLSI: Architecture, Design, and Fabrication*, pages 65–90, 1979.

[38] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In D. H. Albonesi, M. Martonosi, D. I. August, and J. F. Mart'inez, editors, *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42*, number c in MICRO 42, page 469, New York, New York, USA, 2009. HP Laboratories, ACM Press.

[39] C. V. Loan. *Computational frameworks for the Fast Fourier Transform*. SIAM, 1992.

[40] G. Loh. The cost of uncore in throughput-oriented many-core processors. In *Workshop on Architectures and Languages for Throughput Applications*, number June, pages 1–9. Citeseer, 2008.

[41] J. Mandel and S. V. Parter. On the multigrid F-cycle. *Applied Mathematics and Computation*, 37(1):19–36, May 1990.

[42] A. J. Martin. Towards an energy complexity of computation. *Information Processing Letters*, 77(2–4):181–187, February 2001.

[43] T. Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.

[44] R. W. Numrich and M. a. Heroux. Self-similarity of parallel machines. *Parallel Computing*, 37(2):69–84, Feb. 2011.

[45] E. T. L. Omtzigt. *Domain flow and streaming architectures: A paradigm for efficient parallel computation*. Phd dissertation, Yale University, 1993.

[46] D. Patterson. Technology trends: The datacenter is the computer, the cellphone/laptop is the computer, October 2007. www.hpts.ws/papers/2007/TechTrendsHPTSPatterson2007.ppt.

[47] S. J. Pennycook, S. D. Hammond, S. A. Jarvis, and G. R. Mudalige. Performance analysis of a hybrid MPI/CUDA implementation of the NAS-LU benchmark. In *Proceedings of the International Workshop on Performance Modeling, Benchmarking and Simulation*, New Orleans, LA, USA, Nov. 2010.

[48] A. F. Rodrigues et al. The structural simulation toolkit. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):37, Mar. 2011.

[49] J. E. Savage. *Models of Computation: Exploring the power of computing*. CC-3.0, BY-NC-ND, electronic edition, 2008.

[50] J. Shalf, S. Dosanjh, and J. Morrison. Exascale computing technology challenges. *High Performance Computing for Computational ScienceVECPAR 2010*, pages 1–25, 2011.

[51] A. Snavely, N. Wolter, and L. Carrington. Modeling application performance by convolving machine signatures with application profiles. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, pages 149–156. IEEE.

[52] E. Solomonik and J. Demmel. Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. Technical report, University of California, Berkeley, CA, USA, 2011.

[53] H. Takizawa, K. Sato, and H. Kobayashi. SPRAT: Runtime processor selection for energy-aware computing. In *Proc. IEEE Int'l. Conf. Cluster Computing (CLUSTER)*, pages 386–393, Tsukuba, Japan, October 2008.

[54] S. Toledo. Locality of reference in LU decomposition with partial pivoting. *SIAM J. Matrix Anal. Appl.*, 18(4):1065–1081, October 1997.

[55] A. Tyagi. Energy-Time Trade-offs in VLSI Computations. In *Foundations of Software Technology and Theoretical Computer Science*, volume LNCS 405, pages 301–311, 1989.

[56] T. Vogelsang. Understanding the energy consumption of dynamic random access memories. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 363–374. IEEE Computer Society, 2010.

[57] D. H. Woo and H.-H. S. Lee. Extending Amdahl's Law for energy-efficient computing in the many-core era. *IEEE Computer*, 41(12):24–31, December 2008.