SAND2004-0959
Unlimited Release
March 2004

Taking ASCI Supercomputing to the End Game

Erik P. DeBenedictis
Scalable Computing Systems
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-1110

Abstract

The ASCI supercomputing program is broadly defined as running physics simulations on progressively more powerful digital computers. What happens if we extrapolate the computer technology to its end?

We have developed a model for key ASCI computations running on a hypothetical computer whose technology is parameterized in ways that account for advancing technology. This model includes technology information such as Moore's Law for transistor scaling and developments in cooling technology. The model also includes limits imposed by laws of physics, such as thermodynamic limits on power dissipation, limits on cooling, and the limitation of signal propagation velocity to the speed of light.

We apply this model and show that ASCI computations will advance smoothly for another 10-20 years to an "end game" defined by thermodynamic limits and the speed of light. Performance levels at the end game will vary greatly by specific problem, but will be in the Exaflops to Zettaflops range for currently anticipated problems.

We have also found an architecture that would be within a constant factor of giving optimal performance at the end game. This architecture is an evolutionary derivative of the mesh-connected microprocessor (such as ASCI Red Storm or IBM Blue Gene/L). We provide designs for the necessary enhancement to microprocessor functionality and the power-efficiency of both the processor and memory system.

The technology we develop in the foregoing provides a "perfect" computer model with which we can rate the quality of realizable computer designs, both in this writing and as a way of designing future computers.

We found the end game based on certain assumptions about computers and the way we use them. This report focuses on classical computers based on irreversible digital logic, and more specifically on algorithms that simulate space over time with floating point. There are many opportunities in quantum computing, irreversible logic, analog computers, and other ways to address stockpile stewardship that are outside the scope of this report.

Contents

Figures

Tables

Nomenclature

3D…….....Three Dimensional
ASCI……..... Advanced Simulation and Computing
CMOS……. ..... Complementary Metal-Oxide Semiconductor
db…….. .....Decibel
DRAM……..... Dynamic Random Access Memory
ExaFLOPS……. ..... 10^{18} FLoating Operations Per Second
fJ……. .....femto Joule, or units of 10^{-15} Joules
FLOP…..... FLoating point OPeration
FLOPS….. .....FLoating Operations Per Second
FPU……. .....Floating Point Unit
IC…….....Integrated Circuit
ITRS……. ..... International Technology Roadmap for Semiconductors
MPI……..... Message Passing Interface
MPP……. ..... Massively Parallel Processor
NaN……. ..... Not a Number
nm……. ..... Nanometer, or units of 10^{-9} meter
PDE……..... Partial Differential Equation
PIM……..... Processor-In-Memory
SIA……. ..... Semiconductor Industries Association
SOR……. ..... Successive Over Relaxation
YottaFLOPS……. ..... 10^{24} FLoating Operations Per Second
ZettafLOPS……..... 10^{21} FLoating Operations Per Second

Intentionally Left Blank

Introduction

Conventional complexity theory counts floating point operations (FLOPs) required to execute an algorithm as a function of the problem's size, implying that we should minimize FLOPs to reduce complexity and get the best algorithm. However, the laws of physics limit the operation in a computer in a somewhat different way – predominately by limiting the speed of signal propagation to the speed of light and requiring the removal of some amount of heat for each logical operation. Our first objective is to seek a new measure of algorithmic "complexity" more directly related to how good the algorithm is when run on a real computer. This measure will correspond to the amount of computer hardware, time, and energy that must be consumed by an algorithm according to physical law. The algorithm that has the lowest complexity according to this new measure will run the best on any computer we can build in the physical universe.

The second objective in this report is to find a way of building computers that are within a constant factor of the highest possible speed, lowest possible power consumption, and lowest possible cost as permitted by the paragraph above. Of course, there may be multiple solutions given different algorithms and technology available to implement the computer. In this report, we consider primarily algorithms for physical simulation over time running on computers that are built of integrated circuits containing "irreversible logic" transistors. Within these constraints, we have found a universal design for the contents of a chip and a way of packaging this one chip into a large 3D mesh that meets the second objective.

Physical Simulations

Many of the key problems in science, engineering, and the ASCI program involve simulating a region of space over a period of time [Feynman 82]. The region could be an automobile, atomic bomb, or residential living room. The region could undergo a crash, explosion, or fire during the simulated period. As shown in figure 1, these simulations involve distributing the region across the processors of a parallel computer and simulating the time evolution of the region sequentially on the computer.

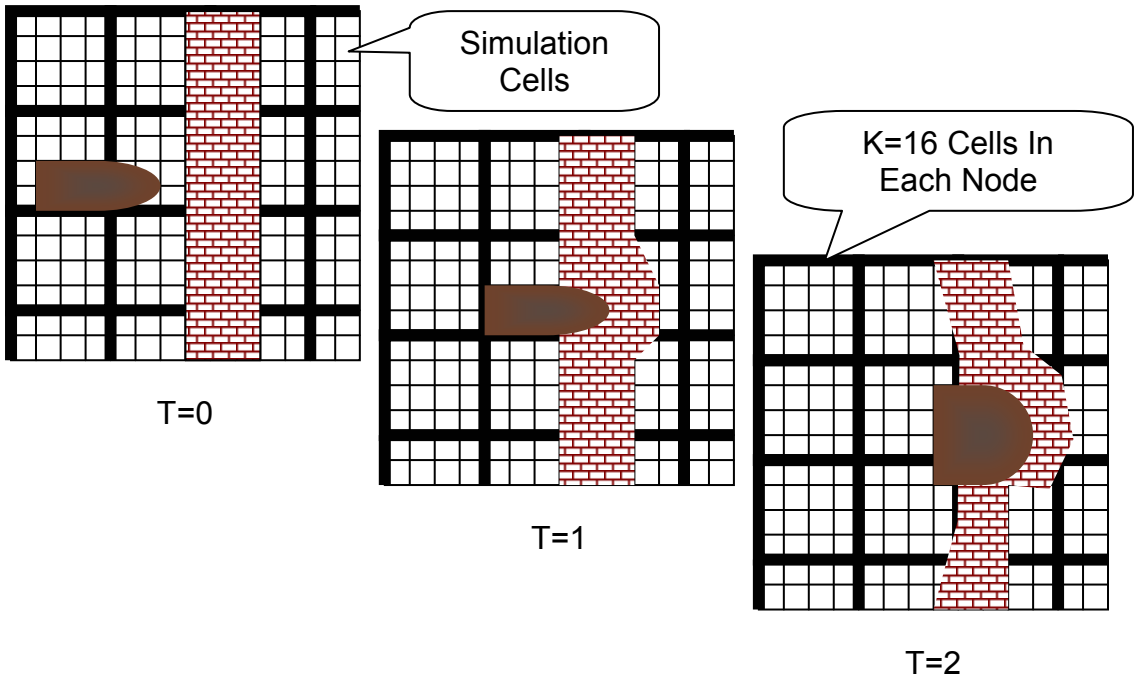

Figure 1: Spatial Simulation Over Time

As shown in figure 2, these simulations consist of a series of activities we define as singly coupled calculations. Each such calculation involves a series of arithmetic operations that are either completely independent or share information locally in the sense of the geometry of the region simulated, followed by a single calculation that involves the sharing of information across the entire problem.

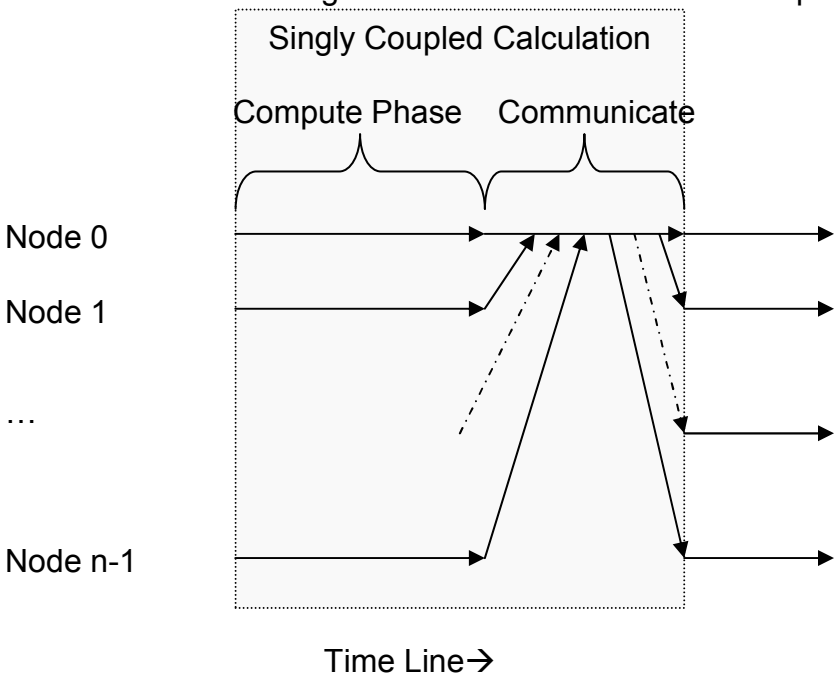

Figure 2: Singly Coupled Calculation

Typically, such an algorithm consists of a repetition of the two steps:

1. A compute phase where the state of the simulated space is updated to reflect the passage of time Δt based on current trajectories, temperatures, etc. This phase's calculations only use information representing nearby points in space and as a result involves no long distance communications among nodes.

2. A global communications phase where the accuracy of the previous step is evaluated based on calculations across the entire simulation. Based on the computed accuracy, time interval Δt is reevaluated. This step is often called an "allreduce" or global synchronization.

Performance Estimation

We can estimate the performance of algorithms on hypothetical computers. With considerable effort, we can project the performance of transistors, gates, and cooling systems years into the future. We can develop algebraic expressions to estimate all the necessary attributes of a computer developed with today's or a future technology, such as:

$$T_{Step} = \frac{K \times F_{cell}}{floprate} + T_{Global},$$

where T_{Step} is the time for a singly coupled calculation, K is the memory size in number of cells per node, F_{cell} is the number of floating point operations required to update one cell, floprate is the effective floating point performance rate of a node, and T_{Global} is the time for the global communications phase. T_{Step}, F_{cell}, floprate, and T_{Global} will be further developed later in this report.

Limits on Computer Performance

There are two opposing limits for physical simulations running on the type of computers covered by this report:

- Relativistic. The time to do the information sharing part of a singly coupled calculation is bounded from below by the diameter of the computer divided by the speed of light. To mitigate the effect of this limit, the computer should be made as small as possible. Even though a sphere has the best geometry for a computer by the criteria of this report, we will assume a computer is a cube with edge length L_{Edge}.

$$T_{Global} \geq \frac{2\sqrt{3} \times L_{Edge}}{c}$$

- Power and cooling. Thermodynamics requires that a minimum amount of heat be generated for each "irreversible" logical operation. While a computer can be allowed to heat up for awhile, at some point it must be

cooled. For every cooling method, the amount of heat that can be removed from an object is proportional to its surface area. This applies to radiative cooling where photons are emitted from the surface to a fluid cooled system where pipes move coolant perpendicular to the surface. To mitigate the effect of this limit, the computer needs to be as large as possible and have as much of its surface available for cooling as possible. For a cubical computer:

$$6 \times L_{Edge}^{2} \times C_x \leq \text{Power},$$

where C_x is the performance of the cooling system in watts that can be removed per unit surface area of the computer.

When both these effects are taken into account, the computer's size will be exactly at the threshold of what can be cooled with the available technology.

Scalability

The new complexity measure theory has somewhat different variables that we are used to. Conventional complexity theory generally looks for the asymptotic dependence of FLOPs (or operations) and memory size on the size of the problem. In our new complexity measure theory we have run time, power consumption, and physical size of the computer all as a function of the problem size.

We find that for a singly coupled calculation of size n:

- Running time $\propto n^{1/3}$
- Power $\propto n^{2/3}$
- Physical size $\propto n^{1/3}$ in linear dimension; $\propto$ n volume

These are different results than are popularly understood to apply to parallel computers: a traditional parallel computer can execute a singly coupled calculation in log n time whereas our computer requires $n^{1/3}$. The discrepancy is that a traditional parallel computer achieves its theoretical performance advantage through a design principle that slightly violates the laws of physics and would describe unbuildable machines if extended to large enough sizes.

Architecture

While the discussion above only established an upper bound on the performance of a computer doing a physical simulation, we have found a computer architecture that comes within a constant factor of meeting this bound. This architecture is comprised of the following parts:

- Integrated circuits of the prevailing technology, per projections of the Semiconductor Industry Association's (SIA's) International Technology Roadmap for Semiconductors [ITRS 02].

- A chip layout per figure 3. The chip will contain several processors of differing architectures that can be switched on and off by a power control system to limit power consumption to that which can be efficiently cooled. We recommend that the processors include a conventional microprocessor for compatibility with existing code plus new processor designs as described later in this report. The chip will also contain a new type of power efficient streaming memory as described later (although conventional DRAM would do almost as well) and an interconnect described in [DeBenedictis 03].

Power Control System:
$\alpha_1 P_1 + \alpha_2 P_2 \ldots \alpha_n P_n + P_{interconnect} + P_{memory} \leq P_{chip}$,
given duty cycle α_n for architecture unit n

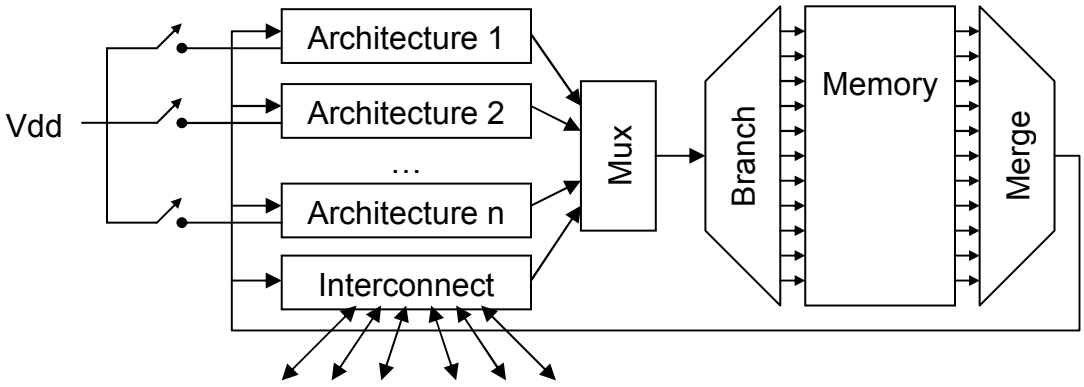

Figure 3: Universal Computing Element

- An Interconnect network capable of implementing a 3D mesh with signal propagation speed within a constant factor of the speed of light, such as shown in figure 4 [DeBenedictis 03].

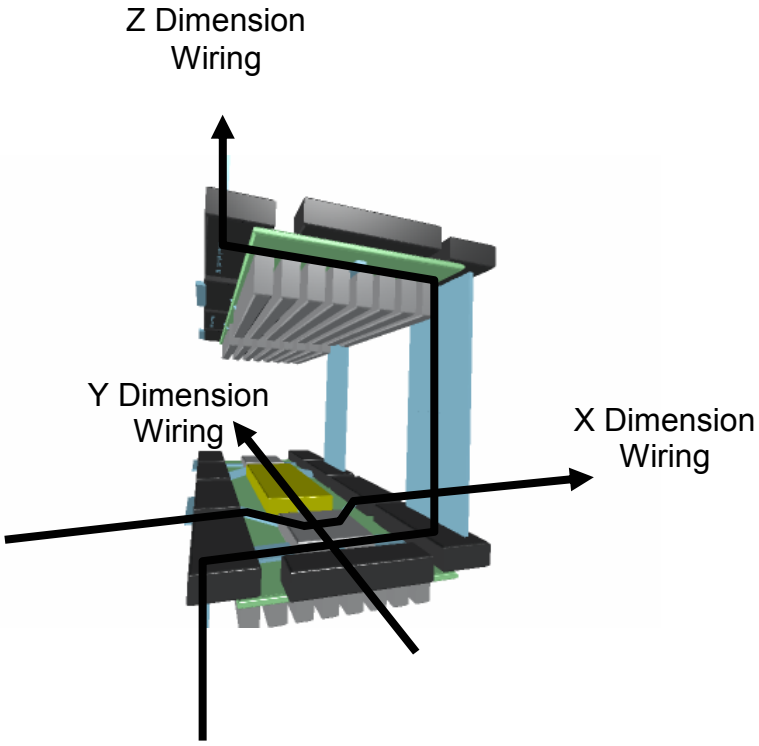

Figure 4: Mapping of 3D Mesh to Physical Structure

- Packaging of the 3D mesh in a way that can be cooled efficiently. Figure 5 [DeBenedictis 03] illustrates an air-cooled structure that is efficient enough at heat removal while preserving locality in the 3D interconnect.

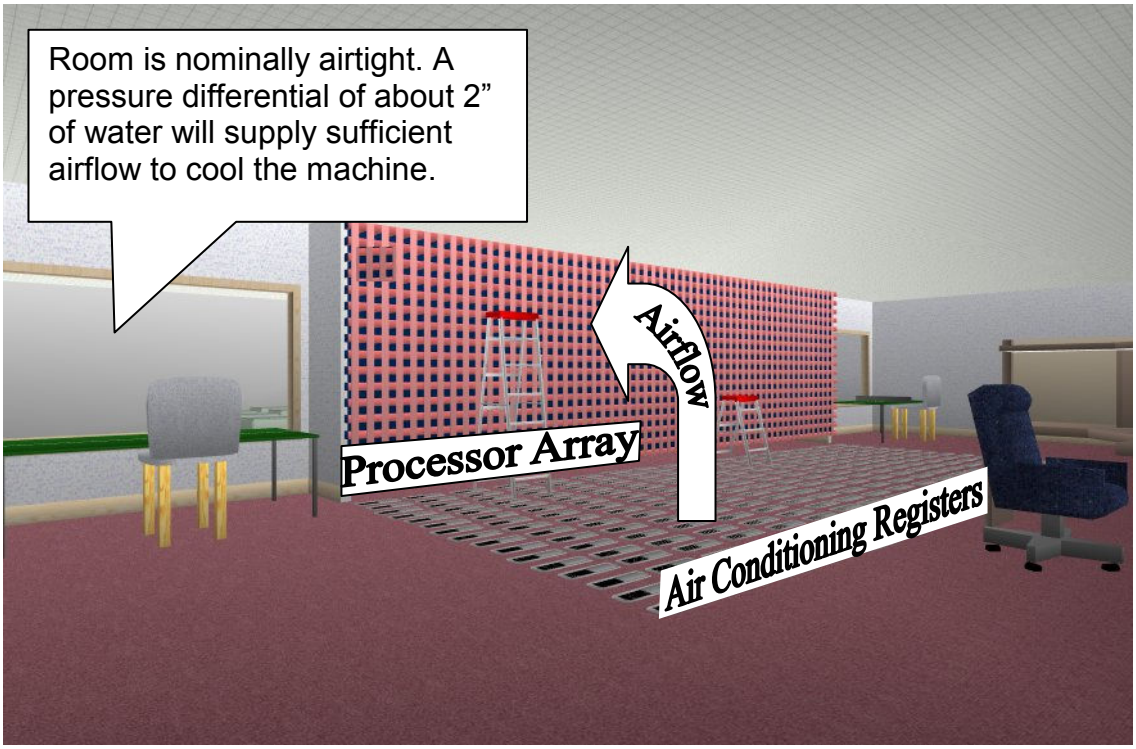

Figure 5: Air-Cooled Configuration

A Review of the Expected Progress in Semiconductors

Minimum Device Size

According to the International Technology Roadmap for Semiconductors [ITRS 02], the smallest CMOS transistor will have a half pitch of 22 nm in 2016 (see figure 6 for a definition of terms). Projections also suggest that a DRAM cell could be constructed in a 2D area 44 nm on a side.

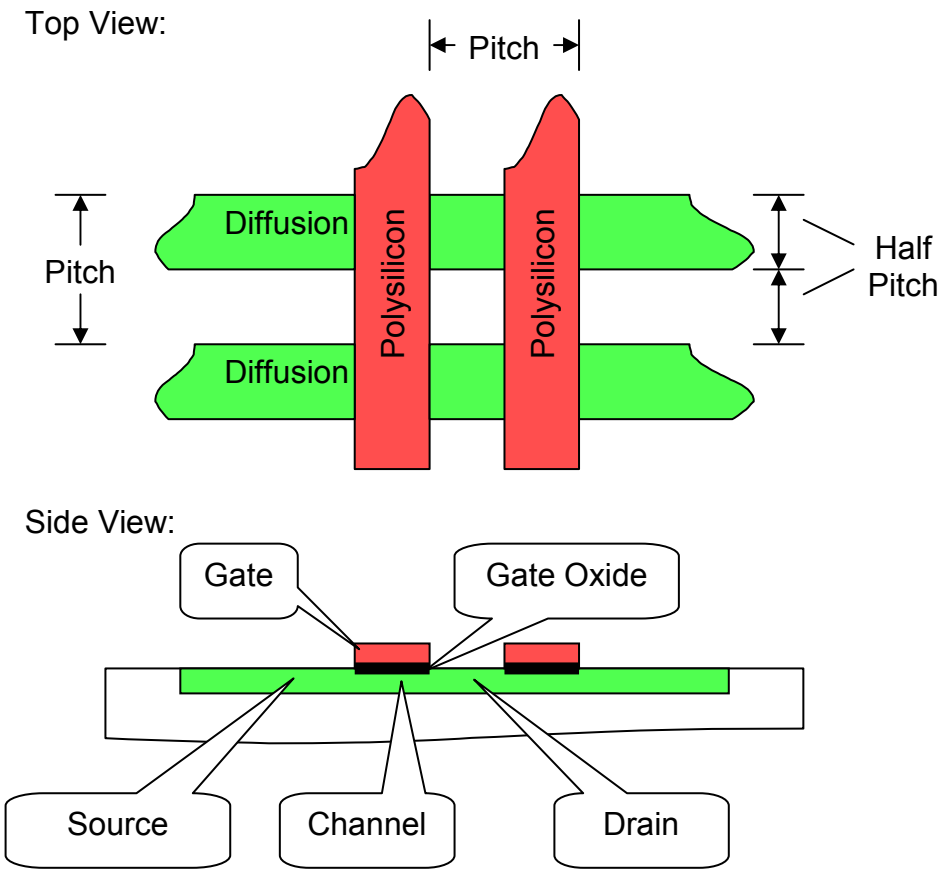

Figure 6: Definitions of Semiconductor Dimensions

The ITRS projects the way Moore's Law will drive the evolution of several hundred semiconductor parameters up to about a dozen years into the future. Table I is an extraction of the parameters from the ITRS used later in this report.

Year of Production	2010	2013	2016	Ref.
DRAM ½ Pitch (nm)	45	32	22	
MPU/ASIC ½ Pitch (nm)	50	35	25	
Physical gate length high-performance (HP) (nm)	18	13	9	
Power-delay product for (W/L_{gate}=3) device [C_{gate} * (3*L_{gate}) *V_{dd}^2] (fJ/device)	0.015	0.007	0.002	35b (HP)
Static power dissipation per (W/L_{gate} = 3) device (Watts/device)	9.70E-8	1.40E-7	1.10E-7	35b (HP)
High-performance NMOS device τ (C_{gate} *V_{dd} / I_{dd}-NMOS) (ps)	0.39	0.22	0.15	35b (HP)
Power-delay product for (W/L_{gate}=3) device [C_{gate} * (3*L_{gate}) *V_{dd}^2] (fJ/device)	0.032	0.016	0.006	36b (LOP)
Static power dissipation per (W/L_{gate} = 3) device (Watts/device)	5.30E-11	1.00E-10	2.00E-10	36b (LOP)
LOP NMOS device τ (C_{gate} *V_{dd} / I_{dd}-NMOS) (ps)	0.85	0.56	0.35	36b (LOP)
Power-delay product for (W/L_{gate}=3) device [C_{gate} * (3*L_{gate}) *V_{dd}^2] (fJ/device)	0.071	0.034	0.025	36d (LSTP)
Static power dissipation per (W/L_{gate} = 3) device (Watts/device)	2.53E-13	3.78E-13	4.32E-13	36d (LSTP)
LSTP NMOS device τ (C_{gate} *V_{dd} / I_{dd}-NMOS) (ps)	1.69	1.05	0.82	36d (LSTP)

Table I: Projections of Selected Semiconductor Properties

The ITRS is often called a self-fulfilling prophecy: Moore's Law has been generally accurate for over a decade but has no basis in physical law. However, much of the hi-tech economy is economically dependent on the continuation of Moore's Law and has an economic incentive to assure its continuation. The starting point for the ITRS report is a simple exponential projection into the future. The ITRS then assigns teams of experts to check projections against manufacturable technologies, research results, etc. The report then color-codes its projections based on the need for additional research and development to make Moore's Law continue as expected. Given the tremendous vested interest in assuring the continuation of Moore's Law, considerable money resources are available to close technology gaps.

The ITRS color code is shown below. All but one of the parameters on which this report is based are colored red. This means the report is based on a technology that a distinguished panel of experts considers to be a valid goal but for which a manufacturable solution is not known at this time.

White – Manufacturable solutions exist and are being optimized	
Yellow – Manufacturable solutions are known	
Red – Manufacturable solutions are not known	

A Review of the Limits of Computer Technology

Thermodynamic Heat Production from Logic Gates

In a now-famous paper, Landauer [Landauer 61] identified that thermodynamics sets a lower limit on the power dissipation of an "irreversible" logic gate at $k_B T \log_e 2$ per switching event ($k_B = 1.38 \times 10^{-23}$ watts/°K is Boltzman's constant and T is the temperature in Kelvins).

While Landauer justifies the $k_B T \log_e 2$ through several arguments, we will repeat just one here for the reader's benefit: Consider a flip flop and its surrounding semiconductor material to be a statistical mechanical system. Statistical mechanics defines the entropy S of a mechanical system as $S = k_B \log_e W$, where W represents the number of quantum states in the system. Let us define the number of states in the system to be W' if we ignore the information in the flip flop. If the flip flop is in an unknown state, the entropy of the system will be $S_1 = k_B \log_e (2\ W')$, corresponding to W' states with the flip flop in a "0" state and another W' states with the flip flop in a "1" state. If the flip flop is forcibly set to a known state ("0" or "1"), the entropy will be just $S_2 = k_B \log_e (W')$. The change in entropy due to destroying information in the flip flop will be $S_1 - S_2 = k_B \log_e 2$. The total entropy of the system cannot change, so the entropy must appear elsewhere as a heating effect, supplying $k_B T \log_e 2$ heat to the surrounding semiconductor material.

A gate (such as a 2 input AND, OR, NAND, or NOR) destroys information when differing inputs (as in a "1" and a "0" or a "0" and a "1") produce a single bit of output from which it is impossible to determine the input combination. Landauer explains more fully in his paper how the destruction of information in a gate is similar to the setting of a flip flop.

We believe that Landauer's lower bound is about a factor of 100 too low given the way we use digital computers. Landauer would acknowledge that gates approaching the minimum energy of $k_B T \log_e 2$ energy would become increasingly susceptible to glitches due to thermal noise. However, we expect computer logic to be immune from glitches. Furthermore, the consequence we impose for a glitch is that we replace the computer (this is a higher standard than is applied to memory devices where we would add an ECC circuit or a heart-lung machine where we would not build the machine in the first place if it were subject to glitches). Since computers have a finite life expectancy, this suggests that the probability of a glitch be less than one in the total number of logic operations the computer will perform in its lifetime. For a future supercomputer running in the Exaflops range, this would be less than one glitch in 10^{30}-10^{40} operations (a 100 Exaflops supercomputer expected to run ten years without error and which uses with 20,000 gate operations per FLOP would require a reliability of about 1 in 7×10^{32} operations).

The experience we will have with semiconductor reliability over the next dozen years is similar in many ways to driving out of town in a car while listening to FM radio: as we drive further away from the radio station, the initially clear signal acquires a “hiss” which grows over time until it obscures the signal and we turn the radio off.

This noise comes from the first amplifier stage in the FM radio: this transistor is exposed to both the radio signal from the antenna and the thermally induced noise signal from electrons in its own structure vibrating. The noise signal is constant throughout the drive out of town, but increases relative to the weakening radio signal.

The transistors in a logic gate are similarly exposed to the signal from the preceding gate and thermal noise from their own electrons. While the magnitude of noise in logic gates is exactly the same as the noise in FM radios (its magnitude is k_BT, dependent only on temperature), Moore’s Law is causing the signal energy to decline exponentially with time (through subsequent generations of electronic technology).

Logic gates are constantly comparing their input voltages against a threshold to determine whether they are receiving a “0” or “1.” The effect of noise is nil unless the noise signal makes an excursion in the opposite direction of the logic signal sufficient to exceed the threshold. The probability of this occurring grows exponentially with the power of the noise signal. We should expect the following:

- In today’s integrated circuits, the signal has about 100,000 times as much energy as the thermal noise (corresponding to a 50 db signal-to-noise ratio). As seen in Table II, the probability of a glitch at this signal to noise ratio is about $10^{-43,000}$, which is too small to worry about.

- The ITRS projects .002 fJ (femto Joules, or units of 10^{-15} Joules) switching energy for 22 nm transistors in 2016. This is about 1000 times the thermal noise, or 30 db. The probability of a glitch is about 10^{-437}, which is too small to worry about.

- Cutting the switching energy by another factor of 10 brings us to the limit. With a signal 100 times more powerful than noise (20 db), the probability of error is 10^{-45}, which is very close to the tolerable error limit for logic meeting our reliability requirements as described above.

It should be noted that the analysis above also represents a “best case.” There are many practical effects that can cut the signal to noise ratio at an input transistor: signal loss in long lines, manufacturing tolerances in transistor size or thresholds, noise from other sources, transistors running hot, etc. Therefore, it

appears that the semiconductor roadmap takes us into the safety margin at the end of the road.

SNR (db)	Power ratio	P_{error}
10	10	3.9E-6
12	16	9.0E-9
14	25	6.8E-13
16	40	2.3E-19
18	63	1.4E-29
20	100	1.0E-45 – Digital Limit
22	160	3.3E-71
24	250	1.4E-111
26	400	1.8E-175
28	630	1.1E-276
30	1000 – 2016 CMOS	4.5E-437
32	1600	3.5E-691
34	2500	7.1E-1094
36	4000	4.9E-1732
38	6300	2.2E-2743
40	10000	3.2E-4346
42	16000	1.8E-6886
44	25000	1.8E-10912
46	40000	3.8E-17293
48	63000	8.3E-27406
50	100000 – Current CMOS	3.2E-43433

Table II: Error Probability as a Function of Signal Power

Digital Computing with Floating Point

Our tradition of using floating point for calculations further defines the minimum power of a computer.

We will stipulate that a 64 bit floating point unit has 100,000 gates based on the following: A 64-bit floating-point multiplier includes a 53×53 multiplier array, each unit of which is about a dozen gates. This results in about 25,000 gates just for the multiplier array. While addition is O(N) instead of $O(N^2)$, where N is the number of bits, floating adders have complex shifters and are of similar complexity to a multiplier in practice. This takes us to 50,000 gates.

However, IEEE compliant floating point has with “Not a Numbers” (NaNs) and denormals. These last features often double the gate count without improving numerical performance. It is unclear whether IEEE compliance belongs in a discussion of theoretical limits of computing. Given these sources of imprecision, we offer 100,000 gates as a plausible FPU complexity.

Furthermore, let us assume a multiplier/adder can accomplish its task in 200τ and with energy corresponding to 20,000 gates switching (for both add and multiply).

These considerations suggest a minimum energy per FLOP of 2×10^6 k_BT per FLOP (20,000 gates $\times$ $100k_BT$ energy per gate operation).

Other Ways to Compute

While the authors feel comfortable presenting 2×10^6 k_BT as the minimum energy per FLOP, the overall approach is not beyond challenge.

The arguments in the previous several pages provide a basis for comparing digital and analog computers. For an analog computer to do an add or multiply will require an expenditure of energy of $k_BT \log_e W$, where W is the number of distinguishable states. While a floating-point number has two parts (mantissa and exponent) compared to just one part for an analog voltage, a double precision floating-point number has 2^{64} distinguishable states. This suggests that if an analog circuit could do an add or multiply with 2^{64} bits of precision, the required energy would be $k_BT \log_e(2^{64}) = 64\ k_BT \log_e 2$. This is 1/30,000 the power of a digital floating-point operation and suggests considerable upside potential for analog computers.

However, it would be remarkably difficult to build an analog circuit where the signals were stable to one part in 2^{64}. The Heisenberg Uncertainty Principle provides some insight: one version of this principle states that the uncertainty in time multiplied by the uncertainty in energy must be greater than $h/2\pi$ ($\Delta T\ \Delta E > h/2\pi$, where $h = 6.63\times10^{-34}$). This implies the time required for a hypothetical analog gate to measure a signal with total energy of about $64\ k_BT \log_e 2$ to a precision of one part in 2^{64} will be $T > h/(2\pi \times 2^{-64} \times 64\ k_BT \log_e 2)$, or about 3 hours. Thus, the analog gate would be impossibly slow.

Reversible logic may provide a solution, but has difficulties as well. Some researchers [Kim 01] made a half step toward a reversible logic computer by constructing a reversible 8×8 multiplier. In some domains, floating-point operations comprise a big part of the activity in scientific computation and consequently a big part of the "unavoidable" power dissipation. Cutting the power dissipation of just the floating-point unit through reversible logic would make a big difference. The idea is to construct a network of reversible logic gates that can perform a floating-point operation and subsequently operate in reverse to recover the energy so it doesn't have to be dissipated as heat. The desired energy flow would be this: A reversible logic network takes two 64 bit floating point numbers as input and operates on them to produce a 64 bit result. This operation will involve 20,000 units of energy equal to the gate switching energy, but it will just move the energy around and not convert it to heat. The 64-bit result will then be saved with irreversible logic, using 64 gate switching units of energy that will

eventually be dissipated as heat. The reversible gates will then be run backwards to restore the 20,000 units of energy to a state where they can perform another floating-point operation. In theory, this will accomplish a floating-point operation with just 64 gate-switching units of energy.

The 8×8 multiplier demonstrates the concept, but shows a “friction” that makes the concept substantially less interesting: this 8×8 multiplier was only about 75% efficient at recovering energy. Thus, if 20,000 units of energy were put in, only 15,000 could be recovered and 5,000 went to heat. Extrapolating these results from an 8×8 multiplier to a full floating point unit, a floating point operation would be 5064 gate-switching units of energy – considerably above the 64 units expected from theory.

Other researchers [Vieri 99] have taken a full step towards reversible logic by constructing a complete microprocessor using reversible logic.

The analysis in this section does not take errors due to Cosmic Rays into account. Cosmic Rays will cause glitches in logic that are indistinguishable from those caused by thermal noise. If we are to be consistent in our expectation that a computer does not produce logic glitches, we must find a solution for Cosmic Ray-induced logic glitches as well.

The idea that one would build a computer from devices that glitch occasionally has been explored intermittently from the early days of computing. Von Neumann [von Neumann 56] considered this topic extensively in the 1950s due to the inherent unreliability of vacuum tubes and the interest in biological, neural computing systems of that day. Von Neumann proposed and analyzed the idea of replacing wires and gates with bundles of wires and arrays of gates. Failure of one or a few gates or wires would not change the ultimate output of the computer due to the action of the redundant copies. This line of reasoning seems to have had a period of inactivity from the 60s to 90s due to the ascendancy of transistorized microelectronics, but has been resurrected recently in the context of nanotechnology. Von Neumann’s ideas seem to apply acceptably to unreliability caused by thermal noise and Cosmic Rays even through they were developed for other effects. However, recent researchers [Han 02] have found better solutions.

However, it does not appear that the work derived from von Neumann changes the conclusions of this report. This report finds power consumption to be the principal limiting factor in the performance of a computer. While emergence of a lower-power computing technology would change the result of this report, von Neumann’s work seems all headed towards higher power dissipation.

Static Power Dissipation

Figure 7 is a simplified view of some of the issues involved in the leakage current that causes undesirably high static power dissipation. Current semiconductor processes permit the designers to create transistors by drawing shapes on masks that correspond to the source, gate, and drain of transistors. If the designer varies the proportions of the shapes, they get transistors with varying properties. Depending on how they vary the proportions, they can optimize the transistors for efficient logic or low standby power:

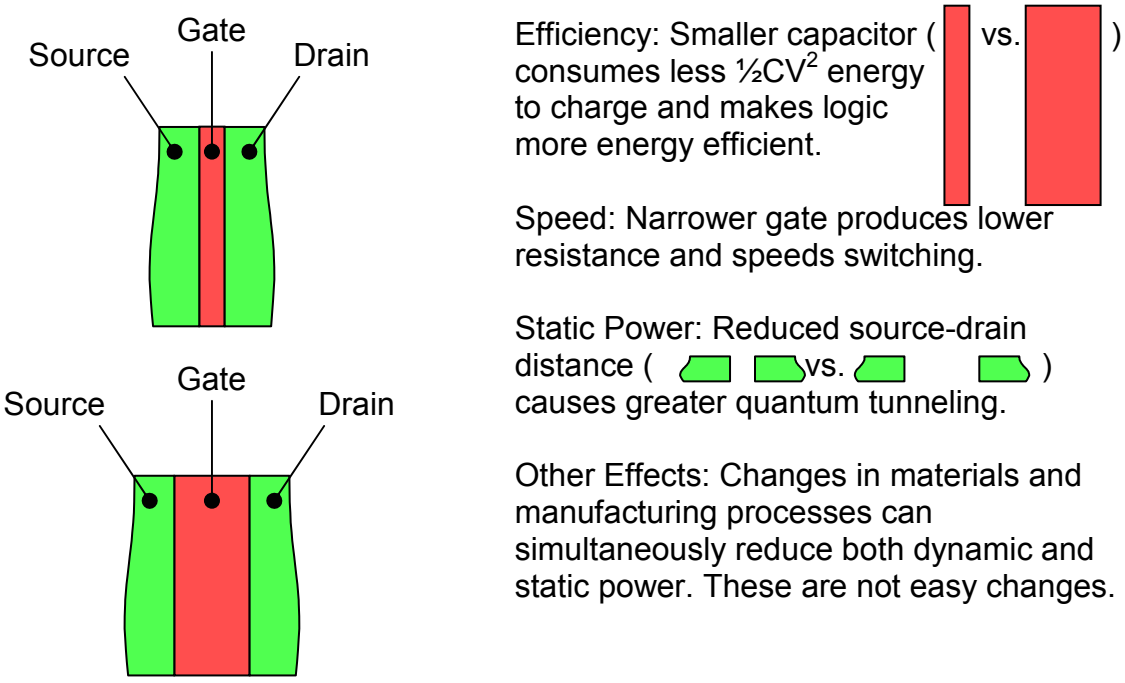

Figure 7: Leakage Current and Mitigations

1. If one ignores the effects of quantum tunneling, the designer would like to have the narrowest possible gate. The narrower the gate, the less its area. Since the gate capacitance is proportional to the gate area, this means less gate capacitance. Overall switching power is $\frac{1}{2}CV^2$, so lower gate capacitance reduces the switching energy. Furthermore, the region under the gate is resistive when the gate is "on." The narrower a gate, the lower its "on" resistance. This means the gate will charge its load faster. Narrower gates are therefore more "desirable" for performing logic by two "linear" factors combined.

2. However, quantum tunneling increases exponentially as the gate region becomes narrower. As the gate becomes very narrow, there becomes a significant probability that an electron will jump over the energy barrier created by a gate in the "off" condition. The sum of all the spurious electrons jumping over an entire chip corresponds to a leakage current

and corresponding power dissipation. Notably, this leakage occurs whether the gates are actively performing logic or just waiting. Quantum tunneling therefore makes narrow gates “undesirable” by an exponential factor.

The two opposing effects above give the designer the opportunity to optimize each transistor for its particular purpose. As summarized in Table III, A “logic” transistor that switches frequently and where speed is needed for overall system performance can be constructed with a narrow gate and the quantum leakage can be accepted as a cost of business. On the other hand, a transistor that switches infrequently (such as many in the memory subsystem) can get a fat gate to reduce static power dissipation. As illustrated in table I, the ITRS describes three classes of transistors (HP, LOP, and LSTP), although the classes are not determined by any authority and the engineer is free to develop others.

The discussion above was intended to illustrate just one way in which leakage current can be managed in conjunction with system architecture. There are other ways: Altering the power supply voltage or the design of transistors (threshold voltage) can vary the size of the energy barrier discussed above. These mitigations could be implemented on a chip-wide basis or by having multiple power supply voltages or transistor types on the same chip.

Duty Cycle	Critical Path	Mitigation
High	Yes	Use a High Performance (HP) transistor and don’t worry about leakage current.
Low	No	Use a Low Standby Power (LSTP) transistor that will minimize leakage current.
Low	Yes	Attempt to minimize the use of these transistors. If they occur in large groups, it may be possible to “power down” parts of the chip to mitigate their leakage current.

Table III: Leakage Current Mitigations

Dimensionality of Space

The fact that the universe has three spatial dimensions sets limits for both signal propagation and cooling. The maximum speed of signal propagation is limited to the speed of light applied to the distance between points in three-dimensional space. Furthermore, cooling is limited by the amount of two-dimensional surface area on a three-dimensional structure.

It is broadly understood that it is best to exploit the full-three dimensional structure of space through a three-dimensional computer packaging [Vitanyi 88].

Signal Propagation Velocity

It should be possible to move information at the speed of light, yet most real technologies move signals at between .1c and .95c. For example:

- Free space optics transmits signals at 95% of the speed of light or higher.
- Optical fibers and electrical transmission lines transmit signals at about 70% of the speed of light.
- Transmission in with wiring layers of an integrated circuit is via a diffusive process. A fixed electrical driver driving a wire with parasitic capacitance to ground will have quadratic delay as a function of length due to the time to charge the parasitic capacitance. Maximum propagation speed occurs when the signal is regenerated periodically with repeaters (inverters). These repeaters should be spaced approximately at intervals where the added delay due to wire equals the propagation delay τ of the inverter. For 22 nm technology in 2016, the this wire length will be 9-19 μm (ITRS table 62b) depending on which interconnect layer is used. For the same technology, τ is .15 ps (ITRS table 35b). These correspond to propagation velocities of .1c - .2c.

Thus, a real implementation may fall short of peak propagation velocity by up to a factor of 10 less than c.

Cooling

As illustrated in figure 8, cooling involves moving a coolant past a heat-producing device, absorbing heat, and removing it with the coolant. Since it makes no sense to consider a coolant pipe bigger than the object being cooled (the coolant would miss the object), the cooling capacity will depend on the area A of the coolant pipe and the matching area A of the device being cooled. Unless the coolant undergoes a phase transition, the heat removed by the cooling system will be A $\times$ velocity $\times$ ΔT $\times$ C, where C is the heat capacity of material. In the case of a phase transition ΔT $\times$ C is replaced by the energy of the phase transition. Practical and theoretical cooling systems differ only in these factors.

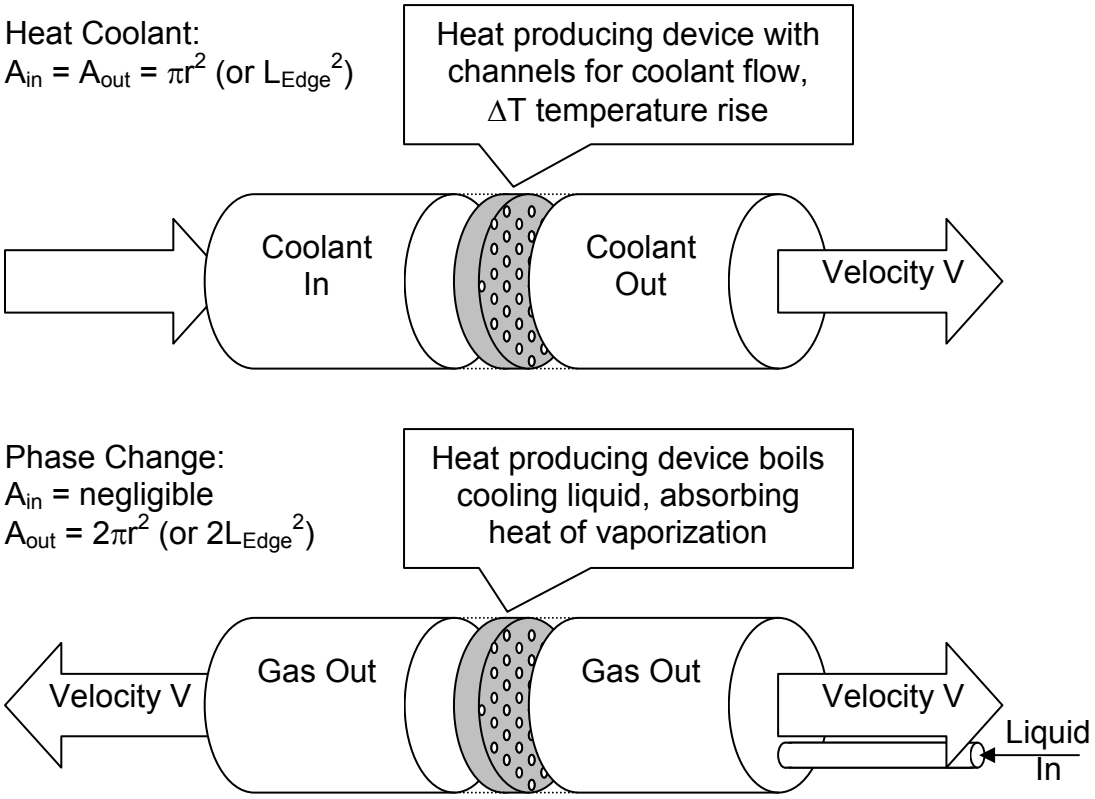

Figure 8: Geometries of High Performance Cooling Systems

For our purposes, the cooling problem has two parts:

1. Getting the heat from the device into the coolant. This task is controlled by a heat exchanger near the device. These heat exchangers will have some efficiency and performance, but their operation will be independent of the other parts of the computer.

2. Getting the coolant out of the computer. As the computer gets bigger, the amount of coolant that can go through the heat exchangers in item 1 grows with the volume of the computer but the surface area with which to run the coolant pipes grows only with its surface area. For some computer size, the ability to route the plumbing to the computer will become the limiting factor.

Table IV illustrates the known and proposed solutions for removing heat from an integrated circuit.

Method	Performance
Heat Sink	100 Watts/cm^2
Evaporative Spray Cooling	100 Watts/cm^2 (unverified figure)
LLNL Microchannel Cooling [LLNL 99]	100 Watts/cm^2
Drexler's fractal plumbing [Drexler 92]	100,000 Watts/cm^2

Table IV: Heat Removal from Chips

Fractal plumbing is illustrated in figure 9. Fractal plumbing refers to a coolant distribution and collection system where the coolant is routed through a series of stages each with more but smaller pathways. Figure 9 illustrates a three-stage distribution system, each stage splitting the pipe of diameter d into two pipes of diameter d/√2. The circulatory system of an animal is an example of fractal plumbing. The challenge in fractal plumbing is to find a design where the coolant can be pumped pretty fast without causing the small pipes to break. Drexler [Drexler 92] reports on an analysis of fractal plumbing that concludes that 10 KW could be removed from a 1 cm cube.

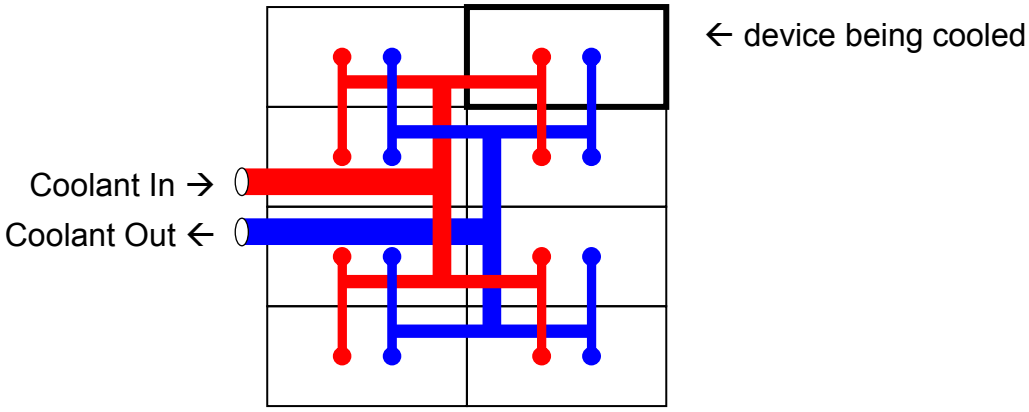

Figure 9: Fractal Plumbing

Removing heat from the overall computer involves efficiently using the surface area of the computer for routing plumbing for cooling. We propose to use the cubical structure in figure 10 for this purpose (again ignoring the fact that a spherical computer would be better by a small factor). The ideal cooling system would implement fractal plumbing in the pyramids.

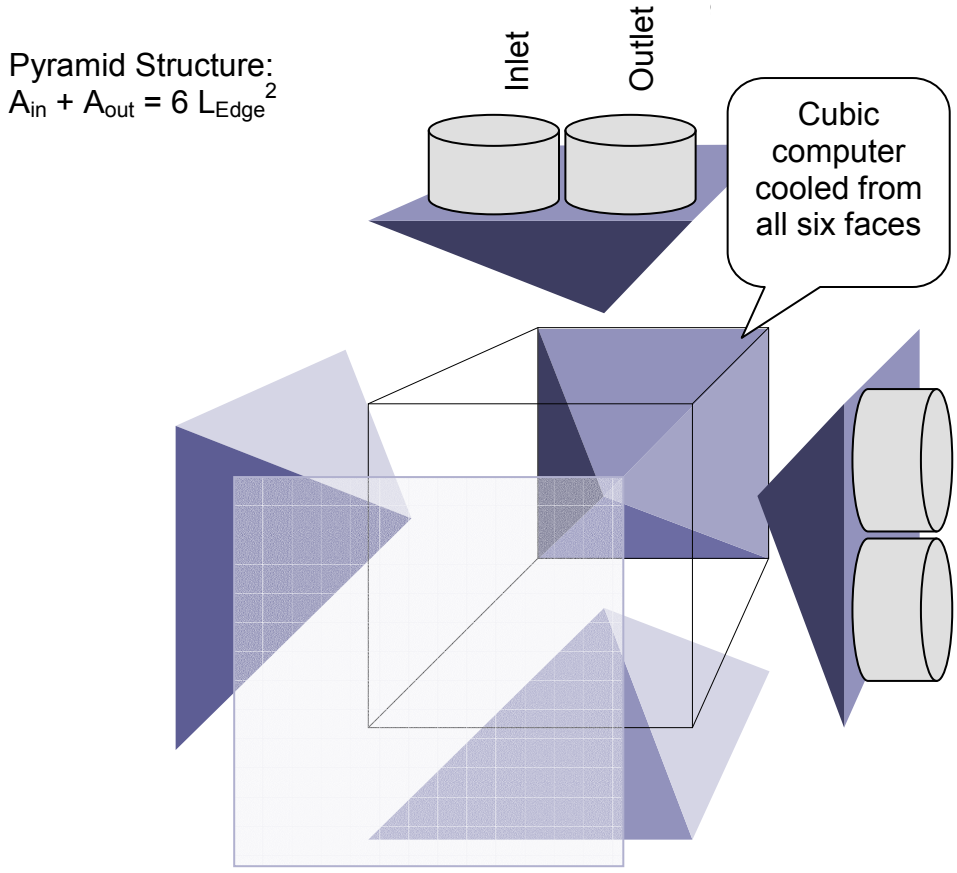

Figure 10: Peak Cooling for a Cube

Table V summarizes the performance of key cooling technologies as they will be used in the remainder of this report.

	V – Velocity of coolant	ΔT – temperature change of coolant	C – Heat capacity of coolant	ρ – density	Perform-ance C_x
Units	m/s	Degrees K	J/g/degree K	g/m^3	kW/m^2
Air	3	15	1.004	1000	45
Water	1	15	4.18	10^6	62,700
Boiling water	3 (steam)	5000 (heat of evaporation, times 2 because steam flows out both directions)		1000 (steam)	15,000

The theoretical limits of cooling are way above what is currently achievable. Table VI [Frank 97] illustrates the theoretical limits of various cooling.

Cooling Technology	Max entropy flux F Watts/cm2
Digital optic fiber	2.63E-08
Current passive emission	9.21E+01
Drexler's fractal plumbing	1.00E+05
Slow atomic ballistic	2.63E+05
Fast atomic ballistic	2.63E+12
Quantum theoretic maximum	1.32E+20

Table VI: Theoretical Limits of Cooling

Since we don't have good theoretical limits on cooling performance, the remainder of this report will leave the cooling system unspecified but parameterized. In some places we will presume a future computer will be built with an unspecified cooling technology characterized by the following:

- Designating the volume of a transistor to include a pro-rated share of the heat exchanger that would be needed to transfer its heat to the coolant.

- Limiting the power consumption of a computer to that which can be cooled through its surface area. If a computer would exceed this limit, it will be "inflated" by moving its parts away from each other uniformly until its surface area is sufficient to cool its contents.

We will also develop specific examples of (1) air cooling, (2) water cooling, (3) fractal plumbing, and (4) ignoring cooling – which is equivalent to a infinite performance coolant or machine operation in a pulsed mode.

The Successive Over Relaxation (SOR) Method as an Exemplary Problem

The SOR method is the simplest computational kernel that uses repetition of the singly coupled calculation. SOR is an enhancement to a simple finite difference approximation for a Partial Differential Equation (PDE). We give a brief tutorial below. While SOR is so simple that it is no longer in use, the methods generalize.

A straightforward PDE might be the solution of Laplace's equation in a 3D region with a uniform mesh. To illustrate the problem domain: the temperature in a 3D metal region would obey Laplace's equation. Since temperatures vary smoothly, the temperature at any point in the interior would be approximately the "average" of the surrounding areas. This might be approximated by a uniform mesh of points X(i, j, k), each value representing the temperature at a point. The most straightforward numerical solution method is to repeatedly update the value of each point with the average of neighboring points:

$$X(i, j, k)' = 1/6 \times [X(i-1, j, k) + X(i+1, j, k) + X(i, j-1, k) + X(i, j+1, k) + X(i, j, k-1) + X(i, j, k+1)] \quad (1)$$

As this assignment is repeatedly evaluated during iteration, the value of X at a given point is nudged slowly from some initial value to the correct answer. Mathematicians have devised an improved algorithm called SOR that essentially "nudges harder." To be slightly more precise, the improved algorithm figures out how much each value of X will change per equation 1. It then increases the amount of change by an over relaxation factor. A typical over relaxation factor might be 20, but is problem dependent. Equation 1 can be written as more loosely as

$$X' = f(\text{other points}), \quad (2)$$

with f defined in equation 1. SOR rewrites this as

$$X' = X + c_{SOR} \times [f(\text{other points})-X], \quad (3)$$

nudging harder by a factor of c_{SOR}. The precise equation used in this example is:

$$X(i, j, k)' = (1-c_{SOR}) \times X(i, j, k) + c_{SOR}/6 \times [X(i-1, j, k) + X(i+1, j, k) + X(i, j-1, k) + X(i, j+1, k) + X(i, j, k-1) + X(i, j, k+1)]. \quad (4)$$

To test for convergence and to control the over relaxation factor, the program will want to know the maximum amount of change in any X value during an iteration.

$$Y = \max \forall\, i, j, k \;\; |x(i, j, k) - x(i, j, k)'| \quad (5)$$

The over relaxation factor must be adjusted to assure that Y decreases smoothly. When Y drops below a threshold, the algorithm terminates.

Implementing the Calculation

Time-Space Tradeoff

To solve the problem as we intend requires addressing a time-space tradeoff. As illustrated in figure 11, this problem can be solved quickly with a lot of hardware or more slowly with less hardware. A computer will contain some amount of storage for input, output, and intermediate results as well as logic to transform input to output. The amount of storage is determined by the algorithm and data set. Storage consumes little if any power and makes no contribution to the computational power of the system. However, the amount of logic can almost always be ramped up and down to control the power consumption and/or computational power of the computer.

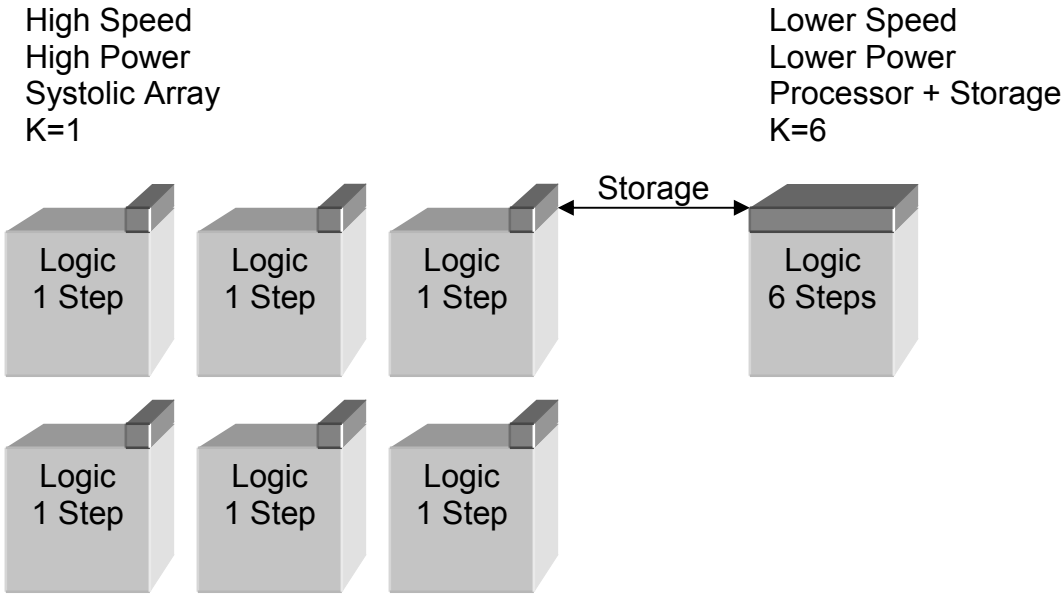

Figure 11: Power Control for a Problem with 6 Variables

We introduce a “power-savings knob” K, representing the degree of multiplexing in the use of the logic. K=1 represents an algorithm running on the computer in figure 1 containing the same number of storage cells as logic cells. This is illustrated on the left side of figure 11. As long as the system is regular, it will be possible to group the storage into groups of K words (i. e. memories of depth K) where the logic acts sequentially on the contents of the memories through K steps. This is illustrated on the right side of figure 11 for K=6.

It should be noted that K=1 corresponds to a Systolic Array[Kung 82]. Somewhat larger values correspond to Processor-In-Memory (PIM) systems[Sunaga 96]. As a broad generalization, typical ASCI clusters and MPPs have K values in the range of 125,000,000 (one microprocessor acting on 1 Gigabyte of memory – or 125,000,000 64 bit floats).

Magic Wiring and the Aerogel Computer Model

A stated objective for this report is to find an architecture approaching an optimal implementation for a particular class of computer. To meet this objective we will first develop an algorithm for a hypothetical "perfect" computer of the desired class (i. e. classical irreversible logic). If we can then develop a real computer that comes within a constant factor of the perfect computer, we can claim that the real computer is within a constant factor of being ideal.

Figure 12 illustrates the "perfect" computer. It has an idealized interconnect where information may flow between any two points at the speed of light. Each cell can be imagined to have a directional laser and a photo detector in a telescope that can be aimed at any other cell in the system. We further stipulate that all cells can "see" each other without the line of sight being blocked by other cells and that the telescopes will be pointed in the proper direction at all times. We call this "magic wiring."

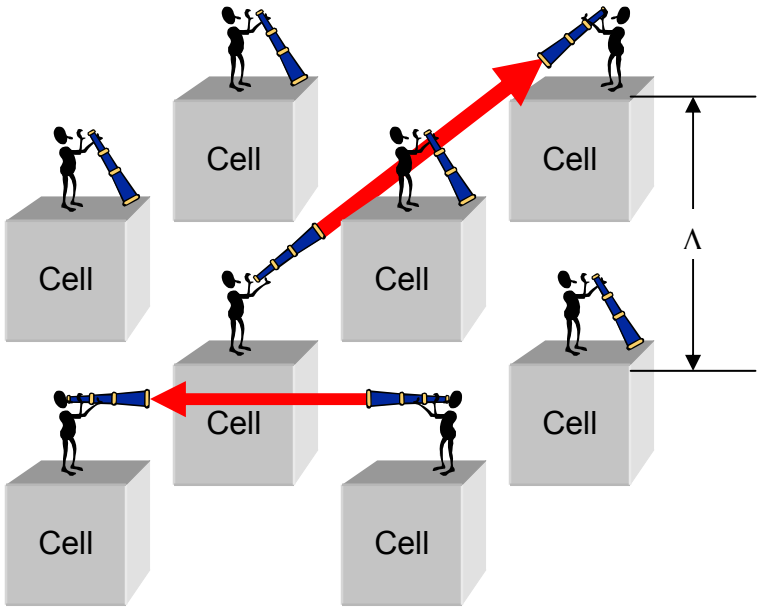

Figure 12: Aerogel Model With Magic Wiring

The physical structure of the "perfect" computer contains cells. The cells are modeled as packing at pitch Λ ("big Lambda") in 3D, thereby each occupying

volume Λ^3. Cells may either hold one bit (storage cells) or may contain a logic gate (AND, OR, NOT). All cells are initially undifferentiated, with the programmer specifying the type for each cell as part of the development of the program.

Non-memory cells perform elementary logical computations. Each such cell receives input information from other cells, performs the designated operation, and sends output information to other cells. This process requires time τ (a propagation delay added to the communications time for the inputs and outputs) and consumes energy E.

Λ is a parameter that can be used to model the active devices, cooling, or both.

- Λ can be used to model minimum device size by setting it to the edge dimension of a basic electronic device. For example, the projected semiconductor technology for 2016 has a basic transistor size of about 10^6 atoms occupying a volume 30-40 nm^2. With Λ set to such a value, one would be modeling an algorithm on a packed 3D array of transistors. This may be a useful model for exploring the limits of computation and may even be practical for a computer operated intermittently (as in a μs at a time). However, a real device constructed at such a density would quickly overheat.

- We can use Λ to account for heat production and cooling through our aerogel computer model. In this model, we use the structure of figure 12 but "inflate" the computer's structure to account for space in which to run a coolant and provide sufficient surface area for pipes to carry the coolant to an external refrigeration system. This hypothetical model would have a composition similar to an aerogel: the cells of figure 12 would be transistors as shown but there would be a lot of space between cells and the entire structure would be mostly empty space. We do not suggest anybody construct such a computer, but merely that it is a convenient approximation to an ideal computer (of the classical irreversible logic variety).

Logic

Assignment statements 4 & 5 above contain 9 floating-point operators and a maximum function. To avoid complicating this report, let us stipulate the mesh point calculation is done with 9 sequential operations of one Floating-Point Unit (FPU).

Figure 13 illustrates the logic of a single node. The cell has three parts, an FPU, a maximum unit, and a memory for holding K 64 bit numbers. The FPU will evaluate equation 4 in 9 steps, using values from the memories of its neighbors. If K>1, the node will evaluate equation 4 K times.

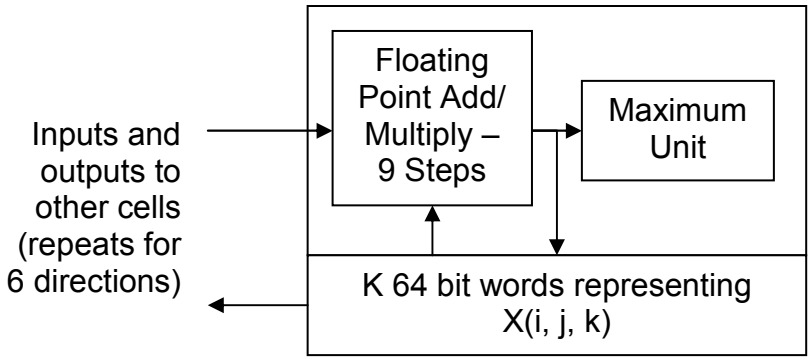

Figure 13: Layout for SOR Application

Calculating the Maximum

The maximum unit in figure 13 works with corresponding units on other such nodes as illustrated in figure 14 to calculate the over relaxation parameter c_{SOR} and control application termination. Calculation of the maximum function in equation 5 is done in two phases: Calculation of the maximum over the K steps associated with each FPU and combination of these local maxima into a single global maximum.

We suggest that the maximum unit in figure 13 can be implemented so simply that it need not be further considered. In the right floating point format, it is possible to compare floating point numbers by doing an integer comparison on the same bits. We assume this will be done (noting that this works just fine for numbers but not NaNs). Integer comparison can also be done bit serially, most significant bit first. With this representation, the maximum unit can be implemented as a bit serial shift register an a one-bit serial integer comparison unit. Such a circuit is of negligible complexity compared to a FPU

At the end of an iteration, the nodes compute Y from equation 5 through a global maximum function by passing the values to one corner of the entire machine. An external “host” processor uses Y to calculate a new c_{SOR}, which is broadcast by the reverse path.

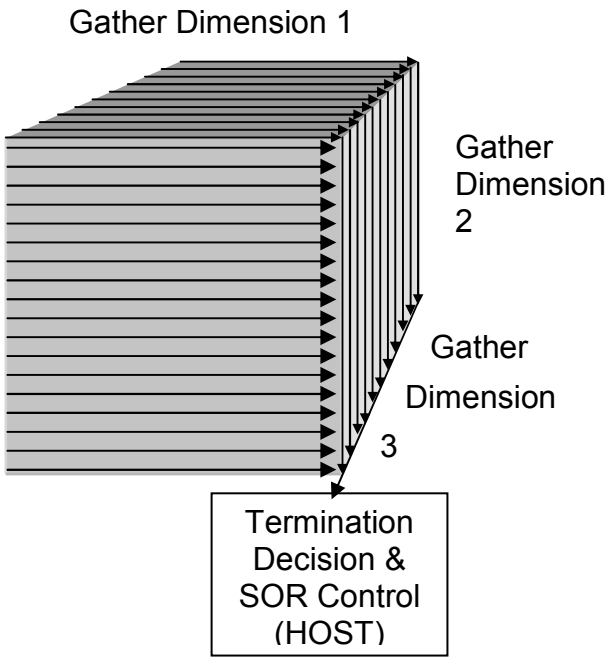

Figure 14: Global Synchronization

Energy Efficient Streaming Memory

Since this particular problem accesses the memory in figure 13 in a very specific way, we can simplify the memory enormously over a conventional memory hierarchy:

First, the access pattern is completely deterministic. Thus, it is possible to create an address generator unit in a thousand gates or so that generates the entire read-write address pattern for the memory. By running the address generator ahead of the addresses that the FPU needs, the memory system can be pipelined and made almost completely independent of read latency.

Second, the write-through latency doesn't matter. The mathematics of this finite difference problem has been formulated where written values are used immediately and where they are they are delayed substantially. The mathematics works both ways.

To meet the objectives of this report, the authors merely need to describe a logic design that comes close to the performance possible with the system in figure 13 (say within a factor of 2). In other words, we are better off referencing a non-optimal design that has been published than inventing an optimal one and writing a treatise on it.

To this end, we have found technology in [Coates 00] that meets our criteria if applied properly.

Figure 15 illustrates the overall strategy. Address generator logic will generate all addresses, but with “write” addresses delayed by perhaps a few dozen positions.

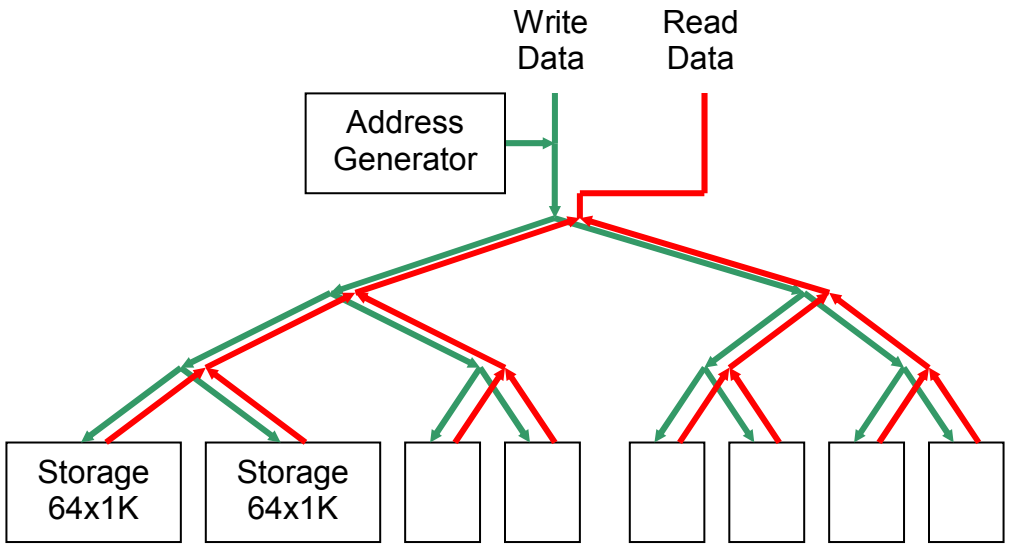

Figure 15: Hierarchical Memory

The memory will be organized as a tree. The “input” for each access (address and write data, if the access is a write) will be sent down a tree branching according to the addressing bits. To keep power consumption low, there will be no activity in the unused branches. The downward path is illustrated in green.

The leaves of the tree are comprised of small memories (say 1024 words of 64 bits), which perform the operation. The read data (or a data-free acknowledgement of write completion) are sent up the tree along the read path.

Figure 16 illustrates some additional technology for implementing the memory. The Sun FLEETZero[Coates 00] project has disclosed an asynchronous logic design for implementing the trees in figure 15 that preserves the order of access as the data emerges from the tree. The key to Sun’s approach is a branch-merge circuit that produces a third “order” stream reporting the branch direction. The sequence of order bits is stored in a 1 bit queue and fed to the merge unit. The merge unit accepts data only from the direction specified by the order bit.

Asynchronous logic is a further advantage of the Sun design: The system specified in [Coates 00] generates all the necessary clocking signals, with the property that unused branches of the tree have zero activity and hence no dynamic power dissipation.

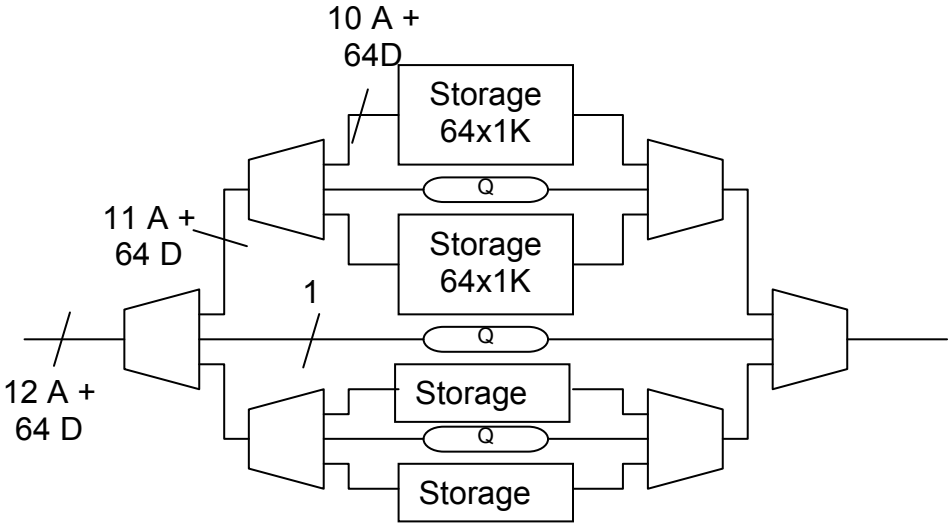

Figure 16: Sun FLEETZero

The additional power consumption due to this design can be estimated as follows: The branch-merge tree will require approximately $2\log_2 K$-10 levels. The logic at each switch point will be dominated by a 64 bit demultiplexer-multiplexer. If each bit requires 5 gates, this would create $(2\log_2 K\text{-}10) \times 5 \times 64 \times E$ additional switching power per FPU cycle.

The circuit in figure 16 will require effective assignment of “high performance” versus “low standby power” transistors. From figure 16, we have proposed a memory system where the data path is “fanned out” as a binary tree originating at the FPU and going to the memory. As a consequence of the binary tree, the “duty cycle” of the demultiplexer units will decline exponentially as units are positioned further from the FPU. In accordance with Table III, the transistors should be switched from HP to LSTP types. Figure 17 proposes making this transition after 4 stages.

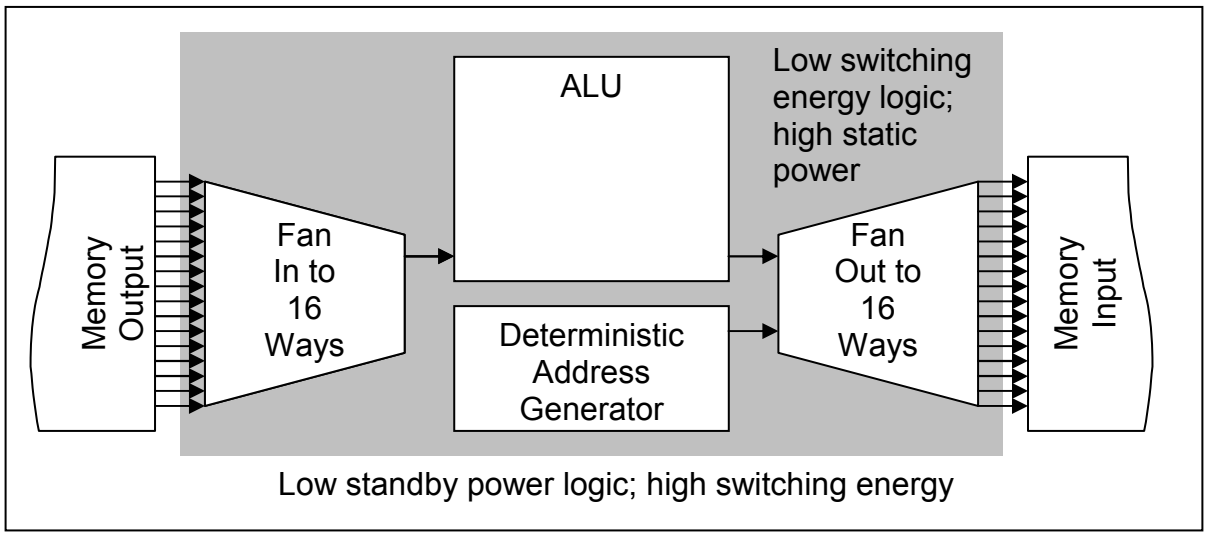

Figure 17: Assignment of Transistor Types

An Architecture Approaching The Physical Limits

While the progress of Moore's Law is normally seen to offer substantial but incremental improvements in computer performance, very large quantitative changes sometimes cause qualitative changes. In this case, transistor count is giving way to power dissipation as the limiting factor in chip performance.

Figure 18 illustrates the essential concept. We are seeking an architecture that approaches the physical limits, so let us stipulate we seek an architecture that reaches 75% of the physical limits.

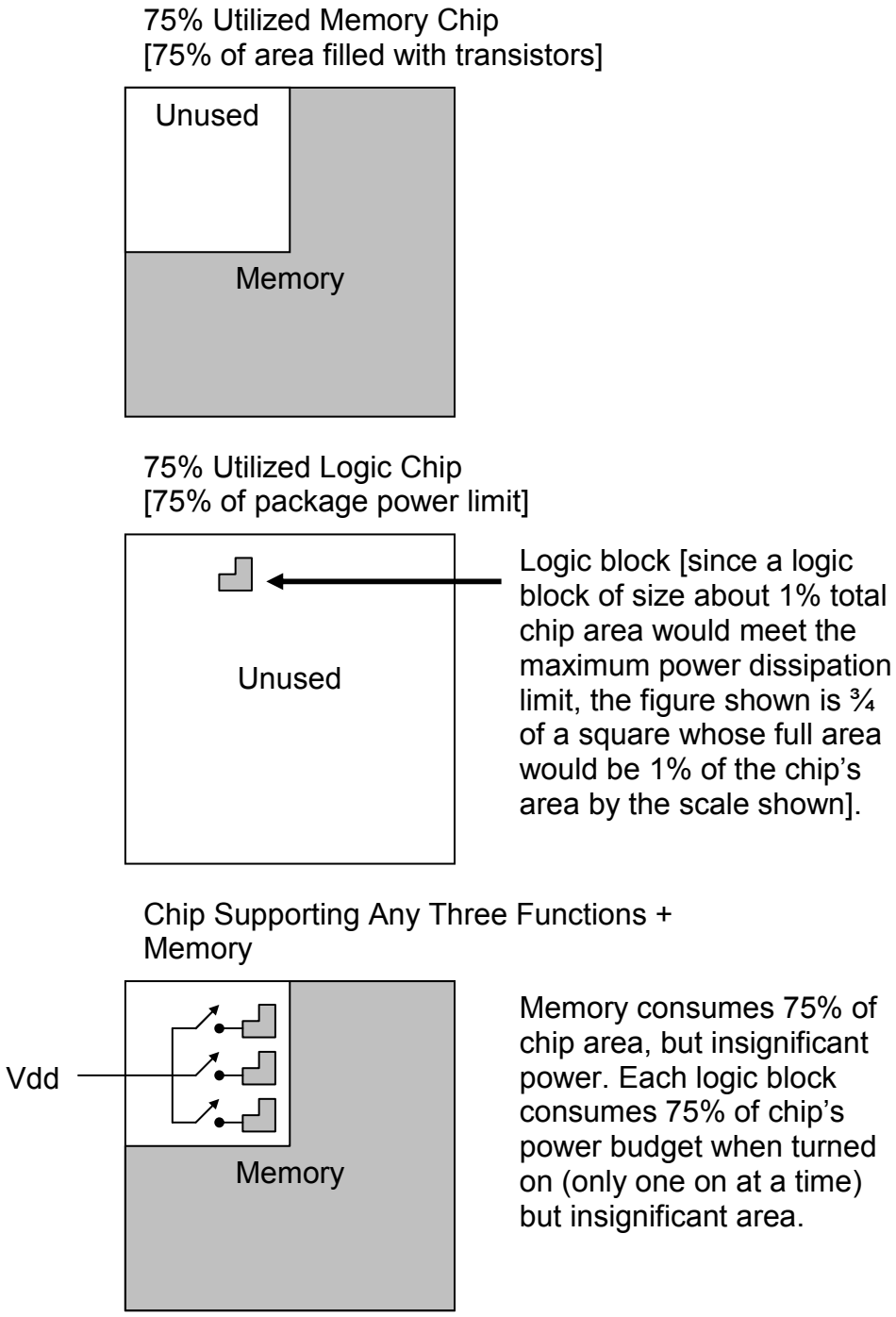

Figure 18: Future Limiting Factors for Chip Design

The top of figure 18 illustrates a 75% efficient memory chip: Since memory will remain limited by transistor count, the chip's surface area is 75% covered by transistors devoted to memory.

The middle of figure 18 illustrates a 75% efficient logic chip: In the 2016 time frame, a chip covered entirely by logic will have power densities that are well above the ability to cool the chip. While we may not know the details of a 2016 logic chip, we can say with some certainty that only a fraction of its surface will be covered by logic with power "on" at any given time.

Memory and logic stress different limits and are therefore compatible, as shown in the lower part of figure 18. A chip could simultaneously contain 75% of the maximum amount of memory while simultaneously containing logic to consume 75% of the maximum amount of power. Furthermore, by switching the power supply on and off, a chip could contain multiple logic blocks – as long as they were not all turned on at once.

Figure 19 illustrates an architecture that can meet the physical limits within a constant factor while being practical as well. The block labeled "architecture 1" would contain the logic in figure 13 and the memory would be as illustrated in figures 15-17. With the power to "architecture 1" turned on, the chip would perform the calculation described in this report with which we will later show has an efficiency approaching the physical limits. With some other architecture block turned on, the chip would be free to perform some other function.

Power Control System:
$\alpha_1 P_1 + \alpha_2 P_2 \ldots \alpha_n P_n + P_{interconnect} + P_{memory} \leq P_{chip}$,
given duty cycle α_n for architecture unit n

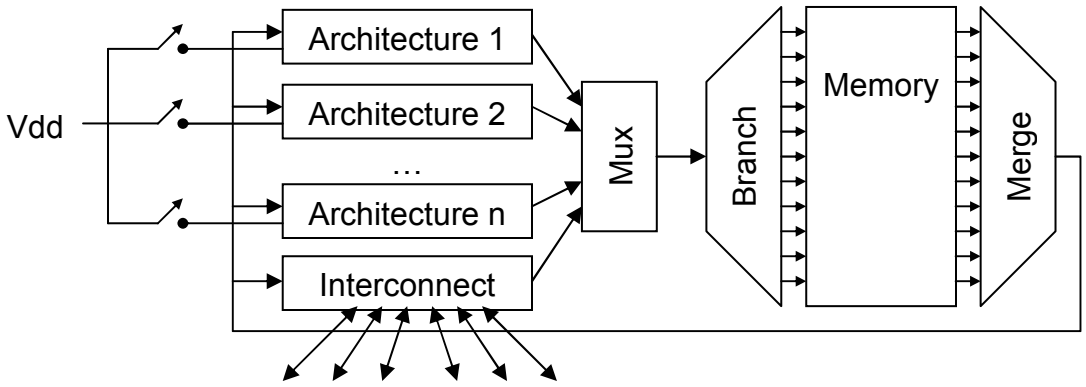

Figure 19: Universal Computing Element

The Universal Computing Element would need a power control circuit that would assure that the logic blocks would be switched on and off in such a way that the chip would not overheat.

We recommend that the block labeled "interconnect" in figure 19 be a wormhole-router based logic unit designed in accordance with other work by the author

[DeBenedictis 03] and integrated into a 3D mesh as illustrated in figures 4, 5, and in other work by the author [DeBenedictis 03]. The interconnect block would carry out the maximum function as illustrated in figure 14.

We would further recommend that the architecture blocks in figure 19 include a:

- Microprocessor. Most codes today are quite large, but with the computational activity concentrated on just a small percentage of the lines. We anticipate the microprocessor would be given the primary responsibility for executing codes, but would offload the main computational activity to other units (and then power itself down). The microprocessor will be useful for its flexibility and as such need be neither fast nor power efficient.

- Vector floating point. The main activity in many scientific codes can be formulated as vector operations on floating point numbers. The advantage being that a vector floating-point unit can be much more power efficient than a microprocessor. To be specific: a well-designed microprocessor can perform vector operations in much the same way as a vector unit. However, the microprocessor performs other (typically scalar) functions that consume power as well. These other functions include speculative instruction execution, storing values in a cache in the event they will be reused, etc.

- Reconfigurable processor. The academic community has considerable interest in processors whose internal components can be rearranged after manufacture to suit specific purposes. Once the components have been rearranged, the processor can be much faster and of lower power consumption than a microprocessor.

- Other special function units. Many applications communities have a small number of computations that form the main activity of many applications. These computations are typically compute-intensive but generic functions that can be transformed with modest compute power to specific tasks. For example: dense and sparse matrix operations in physics, searching in bioinformatics, etc. The advantage of special function units is that their performance and power efficiency can be much greater than any of the other options. Given the chip sizes projected to be available, it should be possible to include a handful of special function units where the benefit to even small constituencies is nonetheless greater than the incremental cost of adding the unit.

Performance Estimation

We will now estimate the performance of both the perfect computer constructed with magic wiring and our most realistic assessment of what is possible. A comparison of the results will not only show whether can approach the theoretical bound, but will also give an assessment of future computing capabilities and costs.

While the preceding parts of this report outlined many of the equations that would be necessary, table VII includes additional data on the packaging model used for realistic computers. The C++ source code for evaluating the model is attached as an appendix to this report.

	Aerogel	Realistic
First level packaging	N/a: only one level of packaging.	First level of packaging is the chip. A chip has a maximum transistor count, a maximum power dissipation, and a maximum bandwidth to other chips.
Nodes	N/a: only one level of packaging.	Each node is a FPU plus K words of memory. Each chip is deemed to hold an integer number of nodes.
Second level packaging	Machine is deemed to be cubical of a size determined by the basic devices packed into 3D but inflates to avoid exceeding coolant capacity limit.	Each chip has a designated minimum volume for the chip plus cooling apparatus. The cubical machine may inflate over the volume of the chips to avoid exceeding coolant capacity limit.

Table VII: Computer Packaging

Figure 20 is a composite graph showing key performance parameters as a function of K, the memory size on each node. The graph is divided into three regions, each with a different behavior.

The lines on the graph are in pairs with a thin line corresponding to an aerogel computer and a thick line for our most realistic estimate of computer performance.

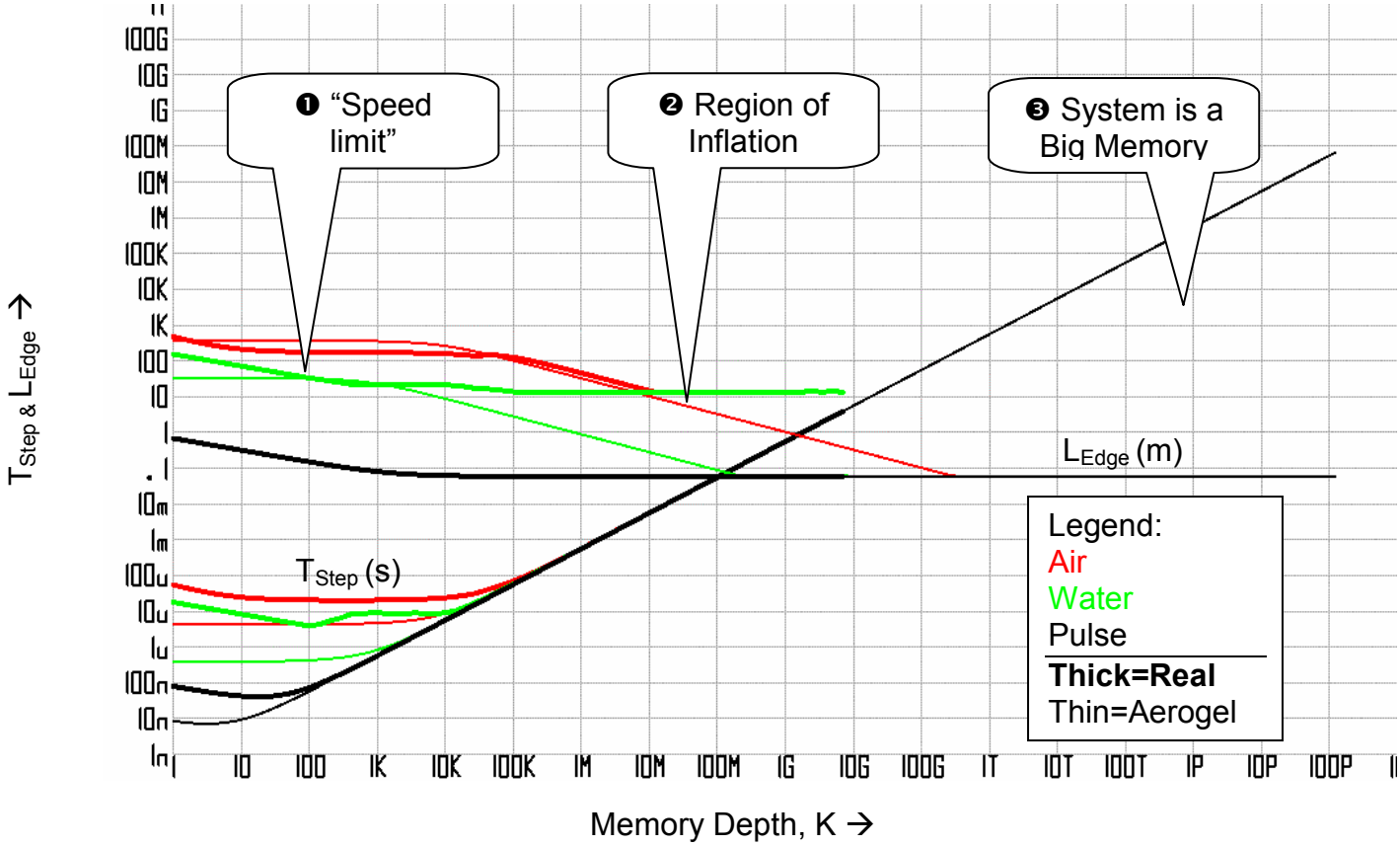

Figure 20: T_{Step} and L_{Edge} vs. Problem Size

- Power consumption is not a problem in the rightmost region❸. In this region, the computer is a cube about .1 m on a side. The cube is comprised of about 10 Exabytes of memory and a "few" processors (1 on the right margin to a billion at the left of the region). Given the large amount of memory and the small number of processors, the size of the cube doesn't change over the region. However, the speed of solution (T_{Step} graph) and the power consumption vary in inverse proportion. The region is characterized by low enough power generation that it can be cooled through the faces of the cube using air or water cooling.

- Cooling is the predominate issue in the center region ❷. In this region, the power generated is too much to be cooled through the faces of the original .1 m cube using the designated cooling method. The remedy is to inflate the machine to increase its surface area. The inflation may be substantial: the original .1×.1×.1 m cube expands to 300×300×300 m in the case of air cooling! While speed of light delays increase as the computer expands, the speed of light effect does not dominate until the next region.

- The speed of light is a controlling factor in the left region ❸. This region begins when the time to compute an ”iteration” approximately equals the speed of light delay in computing the SOR coefficient. As the graph continues leftward, the amount of ALU hardware increases dramatically, yet overall speed doesn’t increase because it is controlled by the speed of light. Overall power consumption doesn’t increase either because the ALU hardware becomes largely idle waiting for the global communications.

Figure 21 plots the timestep T_{Step} as a function of the number of cells n along each side of the solution region. Note that the problem size is n^3.

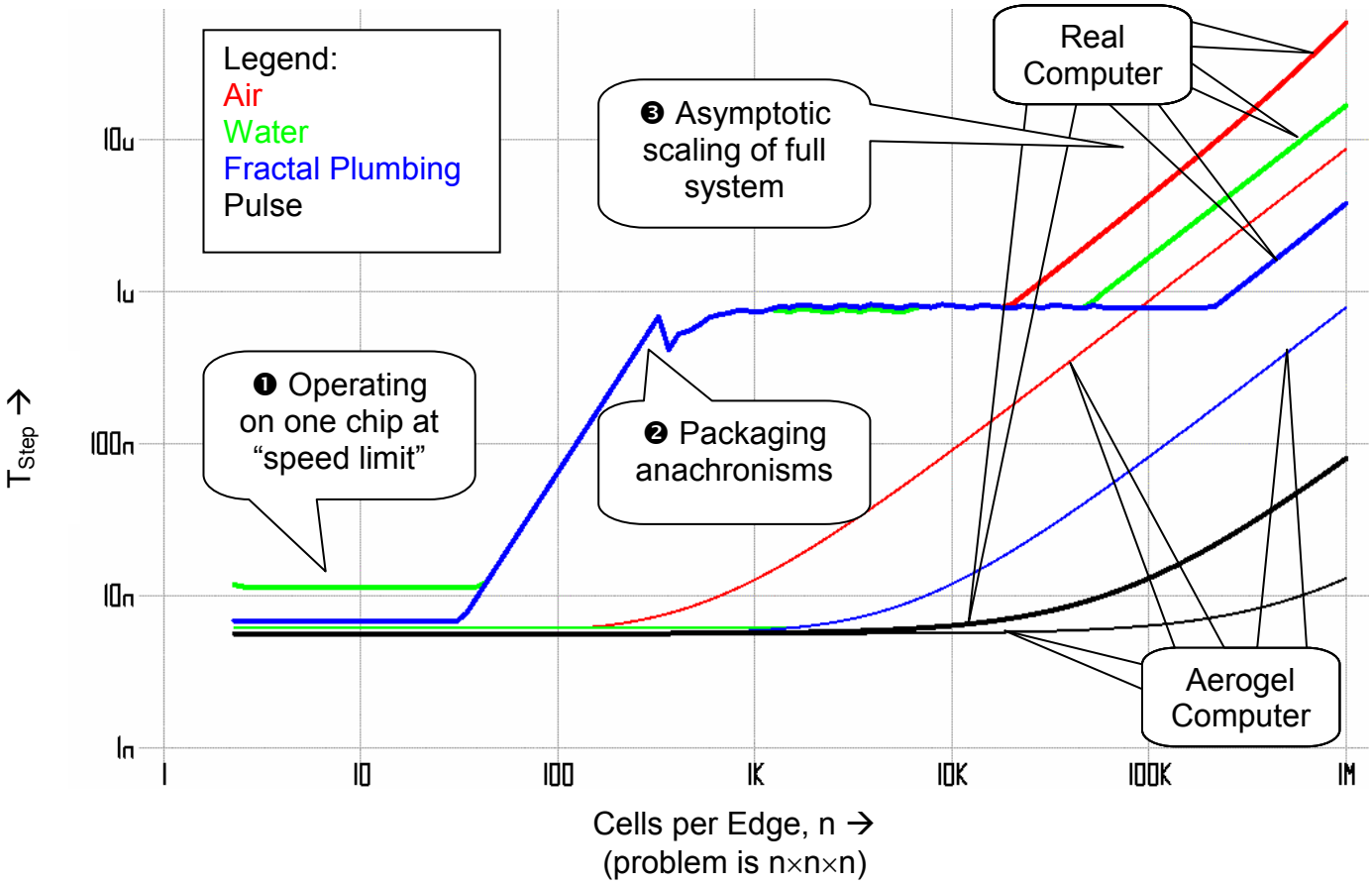

Figure 21: T_{Step} vs. Problem Size

Consider the thinner group of lines first. These represent the behavior of aerogel computers implemented with various cooling systems.

- In region ❶, all the aerogel graphs coincide. In this region, the problem being solved is small and the limiting factor is the speed of the floating-point unit.

- In region ❷, the time steps increase and the aerogel graphs diverge. This region is characterized by power dissipation reaching the limit for the cooling technology and requiring "inflation."

- In region ❸, the aerogel lines become straight as they head to infinity. In this region, the computer is inflated to provide enough surface area to be cooled, which determines the synchronization time in accordance with speed of light delays.

The thicker lines correspond to realistic computers.

- Region ❷ shows a big “bump” in the time step compared to the theoretically perfect “aerogel” computer. This is caused by the two-step packaging system: transistors on chips and chips in a system. Calculations take place predominately in one chip on the left of the region, which is quite efficient. Toward the right of the region, signals must propagate between chips.

- In region ❸, the real systems take on the scaling properties of the overall system. One will note that the thin and thick lines have the same slope on the right, indicating that the performance of the real systems lags the “aerogel” computers by a constant factor.

One notes that the better cooling technology has a significant effect on performance. On the right of the graph, air cooling, water cooling, fractal plumbing, and infinite cooling spread themselves out of 2 ½ orders of magnitude in performance.

Figure 22 shows that there is a lot of performance upside available by plotting peak FLOP rates as a function of problem size. It is widely believed that larger problems have more parallelism available for exploitation than do smaller ones. This effect controls the overall upward slope of the graph.

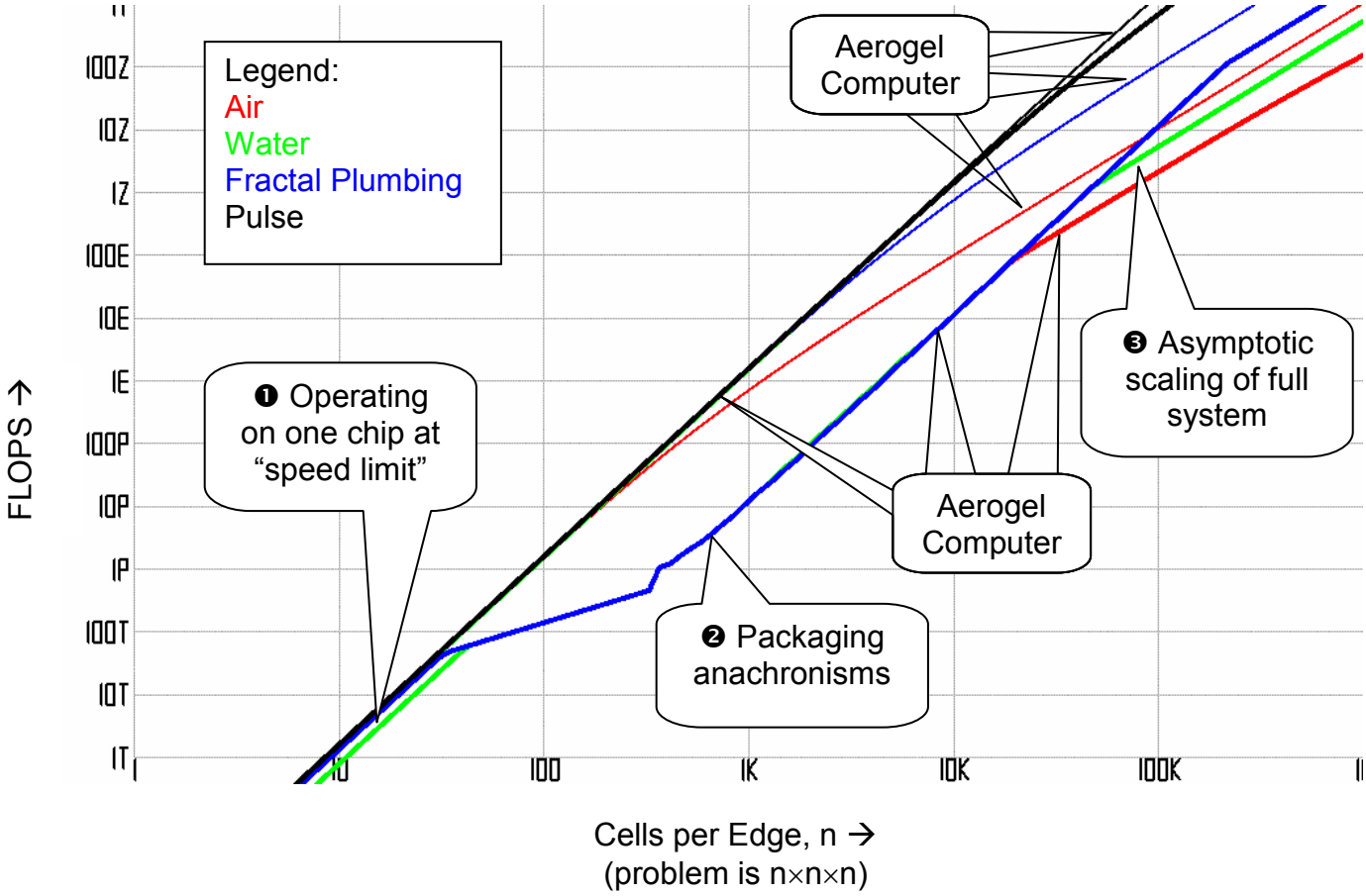

Figure 22: FLOPS vs. Problem Size

When the ASCI program was originally conceived, it was believed that a stockpile simulation could be accomplished with a 1000^3 grid. For all real computers, this problem is on the graph at 10 PFLOPS. This is 100 times larger than was originally proposed by the ASCI program.

The fact that the performance lines go off the graph at 1 YFLOP on the upper right shows that scaled speedup continues to work. However, the power consumption of a computer at the YFLOPS level would be prohibitive. Since the graph covers computers at or close to the thermodynamic limit for classical irreversible logic, the only way to avoid this limit would be to exploit a computing technology based on a different type of physics.

Figure 22 elaborates on figure 20. In this figure, we have set the problem size to 500,000×500,000×500,000 mesh points vary the degree of parallelism in the computer through variable K.

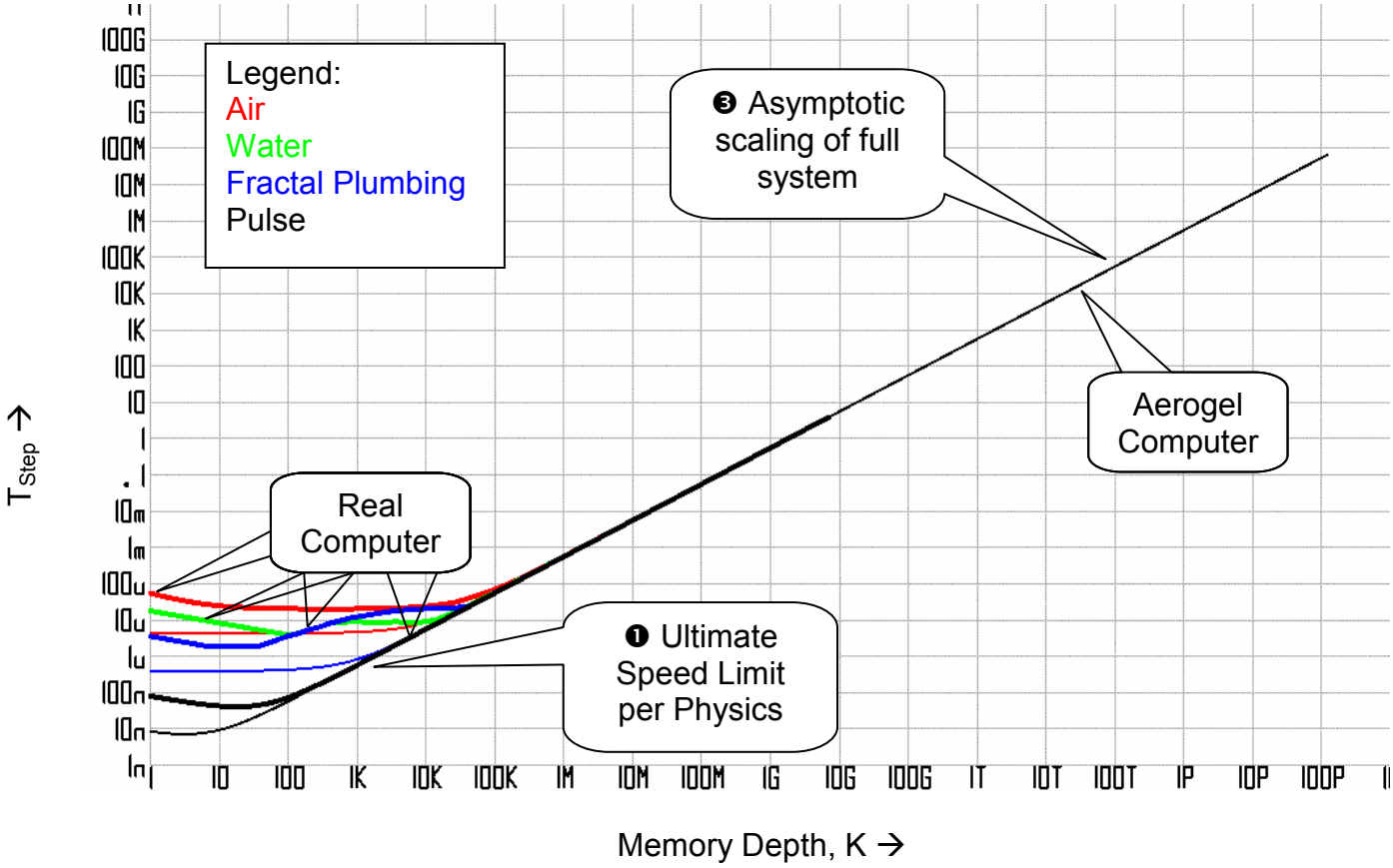

Figure 23: T_{Step} vs. K

- The speed of light is the predominate effect in region ❶. This region begins when the time to compute an "iteration" approximately equals the speed of light delay in computing the SOR coefficient. As the graph continues leftward, the amount of FPU hardware increases dramatically, yet overall speed doesn't increase because it is controlled by the speed of light.

- Power consumption is not a problem in the region on the right ❸. In this region, the computer is a cube about .1 m on a side. The cube is comprised of about 10 Exabytes of memory and a "few" processors (1 on the right margin to a billion at the left of the region). Given the large amount of memory and the small number of processors, the size of the cube doesn't change over the region. However, the speed of solution varies in proportion to the number of FPUs.

Figure 24 diagrams the physical size of the computer of figure 23. The computer is defined to be a cube with edge dimension L_{Edge}.

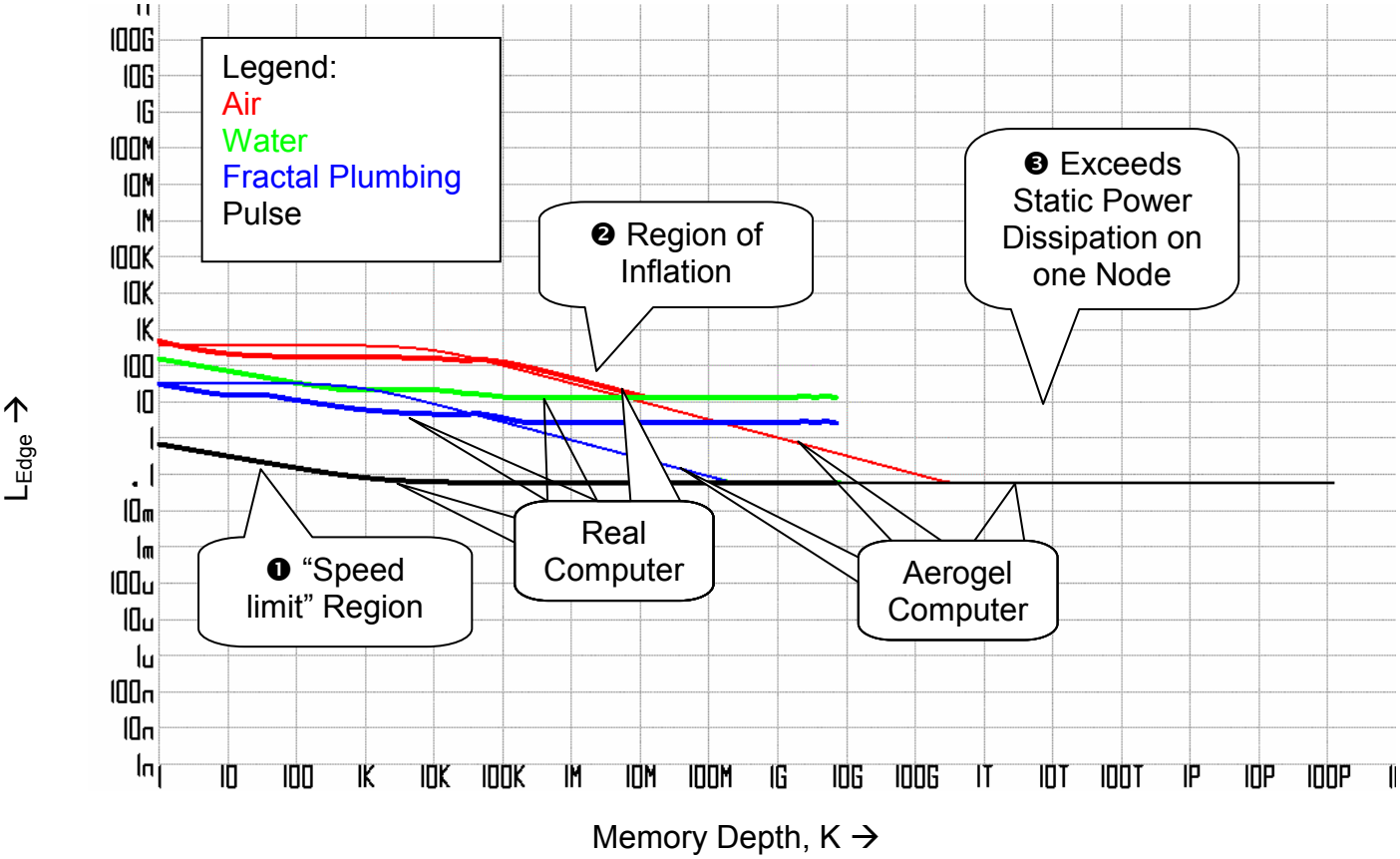

Figure 24: L_{Edge} vs. Memory Depth, K

The interesting region is❷. In this region, the computer is producing so much power that it cannot be cooled just by having heat escape from its surface. To provide enough surface area, the computer is "inflated." While the computer is expanding in this region, the expansion is not so much as to reduce its performance though speed of light effects. The inflation concept applies to several of the lines, which slope downwards at 30° or so. However, the thick lines corresponding to "real" computers are horizontal in some parts of this region. This is caused by static power dissipation creating a lower bound on power consumption and thereby computer size.

In region ❶, the computer has reached the "speed limit." In this region, the calculation cannot be sped up by adding hardware because the added heat the hardware would produce increases the size of the computer and associated speed of light delays.

In region ❸, the computer is a big memory. The thick lines representing the real computers cease to exist beyond K=8G because one chip is not big enough to hold a node.

Figure 25 represents the overall cost effectiveness of the computers shown in figures 23 and 24. The range in figure 25 is a comprehensive estimate of the quality of an financial investment in a specified computer. The range is given in TFLOPS of computer performance for each dollar spent per year, where the expenses include a pro-rated portion of the purchase price of the computer, electric power, and real estate rental cost.

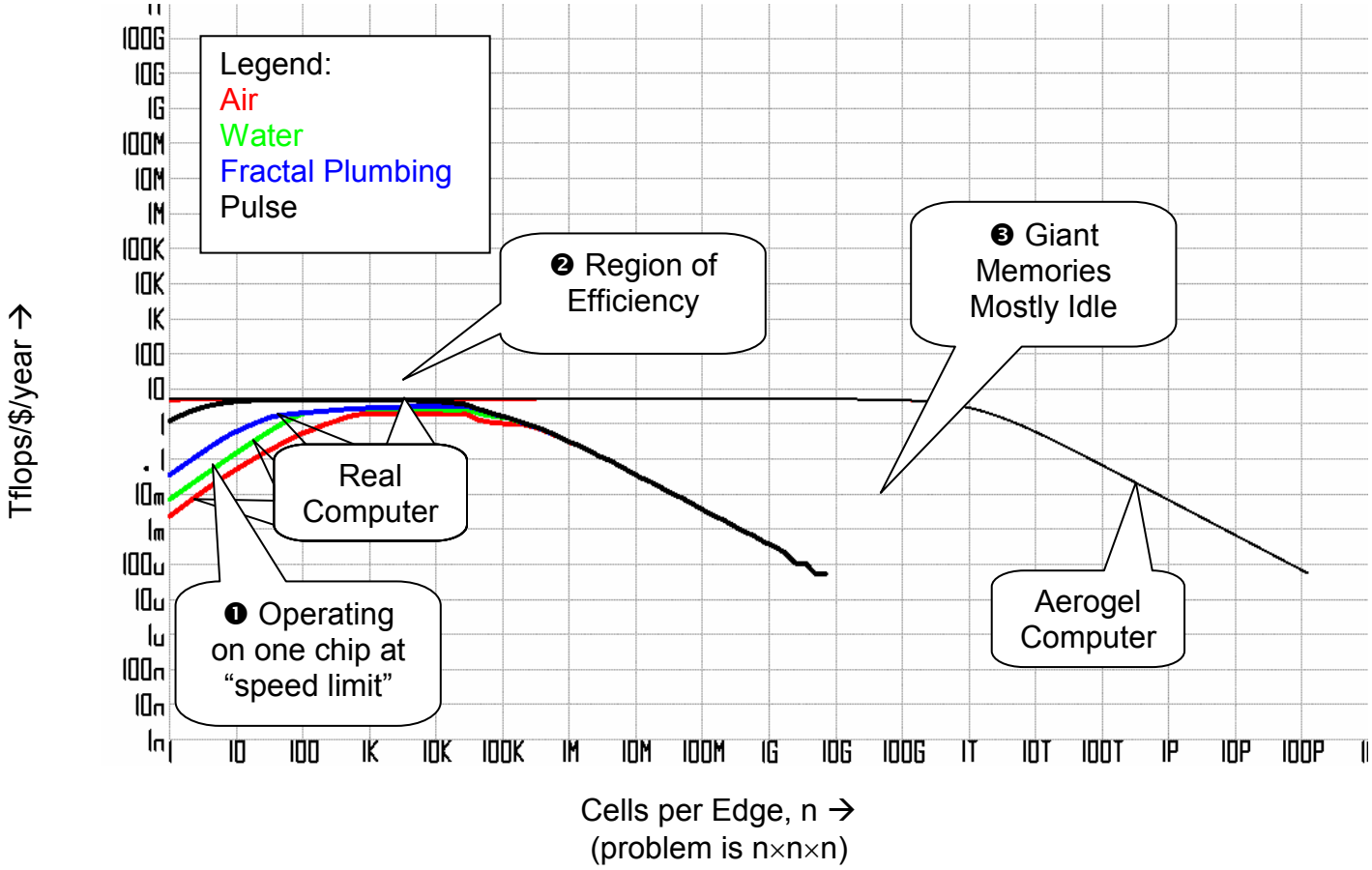

Figure 25: Quality vs. Memory Depth, K

One notes that all the graphs have a peak in the range of 5-8 TFLOPS/$. For all real computers, this peak is reached for 100<K<50,000. These values correspond to present-day Processor-In-Memory designs.

It is also notable that the cooling technology makes little difference in this graph. It appears that high performance cooling lets one get closer to the ultimate limits, yet machines near the ultimate performance limit may not be practical.

Runtime of a Singly Coupled Calculation

The singly coupled calculation that we have defined is a natural basis for the computing model we have introduced with this report. More specifically, we have analyzed a coupling between then fully-parallelizeable (embarrassingly parallel) parts of a calculation and global synchronization. The singly coupled calculation is an "eigenvector" of this system, combining FLOPs and synchronization in a proportion that will scale without changing the proportion. The rate of this scaling will help project the performance of a broad range of problems. For example, each pivot in a matrix inversion is a singly coupled calculation.

At the physical limit (for a computer that can be cooled), the time to execute a singly-coupled calculation will be the sum of two parts:

- The time to share the information – controlled by the speed of light delay for a signal to propagate across the diameter of the machine twice.

- The time to do the computation – controlled by the compute speed – which is controlled by the ability to cool the machine – which is controlled by the surface area of the machine and the performance of the coolant.

These two terms will be equal at the "knee in the curve," at which point the following relation holds:

$$\frac{2\sqrt{3}L_{Edge}}{c} = \frac{9n^3}{6x^2 \times \{\text{ performance of coolant in PetaFLOPS/cm}^2\}},$$

where L_{Edge} is the edge dimension of the cubical machine.

Algebraic rearrangement leads to:

$$L_{Edge}{}^3 = \frac{9cn^3}{2\sqrt{3} \times 6\ L_{Edge}{}^2 \times \{\text{ performance of coolant in PetaFLOPS/cm}^2\}}$$

Which implies $L_{Edge} \propto n$. Thus, we should expect the edge size of a supercomputer optimized for singly-coupled calculations to increase in proportion to the number of cells along each edge in the decomposition. The time per iteration will vary similarly, as in $t \propto n$.

Let us generalize this result for a supercomputer optimized for running a singly-coupled calculation comprised of f floating point operations (in the previous discussion, $f = 9n^3$):

- $L_{Edge} \propto \sqrt[3]{f}$, or the linear dimension of a supercomputer optimized for singly-coupled calculations comprising f floating point operations will vary as the cube root of f.

- $t \propto \sqrt[3]{f}$, or the time to compute a singly-coupled calculation will scale with the cube root of the size of the calculation.

- FLOPS $\propto f^{2/3}$, or the FLOPS rate of a supercomputer as powerful as physical limits will permit (given that the supercomputer can be cooled continuously) will grow sublinearly with problem size (⅔ power).

ASCI Plan

The ASCI program has a semi-formal succession plan for projecting the advance of machines and applications from one generation to the next [Tomkins 01]. Based on the material in this report, we believe this plan must change.

The plan assumes ASCI needs to solve the same simulation problem at all times, but that increasing computer speed is used to increase the resolution of the simulation. As summarized in the first three columns of Table VIII, each "generation" drives a halving in the mesh size in each of three dimensions and in the time step. This results in an 8× increase in memory size and a 16× increase in processor speed requirements. It is assumed that the overall time to solution stays constant due to speedup driven by Moore's Law.

However, this report shows a different result: For supercomputers at the physical limit, each generation will result in a 4× slowdown of simulation on a machine of volume 8× and consuming 4× as much power.

	Old ASCI Rule Generation n	**Old ASCI Rule Generation n+1**	**New ASCI Rule Generation n**	**New ASCI Rule Generation n+1**
Mesh	n×n×n	2n×2n×2n	n×n×n	2n×2n×2n
Time step	T	T/2	T	T/2
Compute capacity	1	16	1	4
Linear dimension			1	2
Memory capacity	1	8	N/a	N/a
Time to solution	1	1	1	4
Volume of Computer	1	1	1	8

Table VIII: Scaling of Resolution in Simulations

Conclusions

In this report, we devised and demonstrated a process for understanding the quality of a parallel computer in absolute terms. This process involves applying an algorithm to a hypothetical computer model (called “aerogel”) that represents the best computer that can be constructed given the laws of physics. In conjunction, the same algorithm is applied to an actual computer that we may be designing. The amount by which the actual computer falls short of the “aerogel” model in performance, cost, or some other factor, tells us how good a design job we did. This process can be used to analyze existing designs or as part of a design iteration to create new designs.

In the process of describing the aerogel computer model, we identified the key limitations placed on ASCI-like computers by the laws of physics. These limits include fundamental heat generation in floating point and logic and the speed of light limit to signal propagation velocity.

To guide our analysis, we used a very computational algorithm called the Successive Over Relaxation (SOR) method. This algorithm is simple enough to analyze in a report yet incorporates the basic combination of spatially distributed computations and global synchronization common to many ASCI applications.

We found no bottleneck to prevent ASCI supercomputers from growing in performance to the Exaflops level and above. Assuming that problems continue to scale up in size (the principle of scaled speedup), the ultimate limit on ASCI supercomputer performance is likely be the minimum thermodynamic energy required for computation and our ability to pay the power bill.

By including the effects of heating and the speed of light on signal propagation, we may have defined a new model of computation. This model is similar to the parallel computer model of which we are familiar, but cuts the asymptotic performance somewhat to keep a computer from overheating. We presented a way to predict scalability on this type of computer by decomposing an algorithm into “singly coupled calculations,” each of which scales predictably.

We believe this approach is useful for the current class of digital computers, but it is not universal. To be precise, we believe this approach is valid for computers based on classical, irreversible logic and specifically for logic used as the basis of floating point calculations. However, this approach would not apply to analog computers such as neural networks and biological computers or to computers using quantum entanglement.

```
        void Compute() {
                // Physical Constants
                double kB = 1.3806503e-23;              // Boltzmann's constant J/K
                double T = 300;                                 // room temperature K
                double c = 299792458;                   // speed of light m/s
                double MetersPerFoot = 2.54*12/100;

                // Parameters that could be static
                double HSSGBits = 40e9;                 // HSS speed (bits/s)
                double ChipArea = .02 * .02;        // Nominal area of a chip = 2 cm x 2 cm = 400mm^2 (m^2)
                //double ChipArea = 140e-6;             // MPU High Volume per ITRS 1h 2002 (m^2)
                //double ChipArea = 572e-6;             // ASIC maximum chip size at production per ITRS 1j 2002 (m^2)
                double FloatBits = 64;                  // number of bits per floating point number (bits)
                double GrindFLOPS = 9;                  // number of flops per SOR update (floating ops)
                double RentalCostSquareFootPerYear = 12;     // rental cost of real estate ($ per square foot per year)
                double CostPerChip = 1000;              // purchase price per chip in a system ($)
                double KWHCost = .15;                   // price per kilowatt-hour of electricity ($/KWH)
                double DepreciationFactor = .3;    // fraction of HW cost to amortize per year
                double FracSpeedOfLight = .1;     // signal propagation velocity as fraction of c
                double WordsPerMemory = 1000;// number of words in primitive memory

                // Formulas
                double TotalNodes = n*n*n/K;
                double SystemMemoryBits = FloatBits*n*n*n;
                double SystemCPUGates = FloatCells*TotalNodes;
                double TotalCells = SystemMemoryBits + SystemCPUGates;
                double MeshUpdateTime = GrindFLOPS*K*FloatTau*LogicProcess.Tau;
                double PropagationVelocity = Magic ? c : FracSpeedOfLight*c;        // speed of signal propagation

                // FLEETZero branchmerge
                // properties for the branch-merge circuit down to WordsPerMemory word memories
                double BranchMergePerNode = ceil(K/WordsPerMemory)-1;
                double FastBranchMergePerNode = min(BranchMergePerNode, 31);
                double SystemFastBranchMergeGates = TotalNodes * 30*FastBranchMergePerNode*FloatBits;         // 30
gates per bit * 64 bits
```

```
		double SlowBranchMergePerNode = BranchMergePerNode - FastBranchMergePerNode;
		double SystemSlowBranchMergeGates = TotalNodes * 30*SlowBranchMergePerNode*FloatBits;	// 30
gates per bit * 64 bits
		double DecoderLevels = ceil(log(K/WordsPerMemory)/log(2));		// levels of decoders
		double BranchMergeEnergyPerAccess = 5*FloatBits*(max(DecoderLevels, 5)*LogicProcess.E +	// 5
gates switch per bit for each level
				min(DecoderLevels-5, 0)*MemoryProcess.E);

		double SystemStaticPower;
		if (Magic)
			SystemStaticPower = 0;
		else {
			double FastPower = (TotalNodes*FloatCells + SystemFastBranchMergeGates) *
LogicProcess.SPwr;
			double SlowPower = TotalNodes*SystemSlowBranchMergeGates * MemoryProcess.SPwr;
			SystemStaticPower = FastPower + SlowPower;
		}

		// transistor count per node
		double RamBitsPerNode = FloatBits*K;
		double FPUGatesPerNode = FloatCells;
		double MemoryGatesPerNode = 30*min(K/WordsPerMemory, 32)*FloatBits;
		double TransistorsPerNode = RamBitsPerNode + FPUGatesPerNode + MemoryGatesPerNode;

		double MaxTransistorsPerChip = ChipArea/(LogicProcess.Lambda*LogicProcess.Lambda);
		double Edge3D = pow(TotalCells, (1./3.))*LogicProcess.Lambda;	// Edge assuming 3D packaging

		double SystemFloatEnergy = n*n*n*GrindFLOPS*FloatPower*LogicProcess.E;
		double SystemMemEnergy = Magic ? 0 : n*n*n*BranchMergeEnergyPerAccess;
		double SystemEnergy = SystemFloatEnergy + SystemMemEnergy;

		Quality = -1;
		ComputerInstance Test = *this;
```

```
// Elaborate loop:
// Step through the range tNodesPerChip = 1..TotalNodes in tandem with
// Test.Chips = TotalNodes..1 such that
// TotalNodes = Test.Chips * tNodesPerChip
// However, have both tNodesPerChip and Test.Chips cover small ascending integers
for (double x = 1, ex = sqrt(TotalNodes); x < ex; x += (x < 10 ? 1 : x/10)) for (int y = 0; y < 2; y++) {

        double tNodesPerChip;
        if (y) {
                tNodesPerChip = x;
                Test.Chips = TotalNodes/x;
        }
        else {
                Test.Chips = x;
                tNodesPerChip = TotalNodes/x;
        }
        double tCubeRootChips = pow(Test.Chips,(1./3.));
        double tTransistorsPerChip = TransistorsPerNode * tNodesPerChip;

        // communications
        double tCellsPerEdge = n/tCubeRootChips;
        double tCommCells = 6.*tCellsPerEdge*tCellsPerEdge;
        double tCommBits = FloatBits*tCommCells;
        Test.CommunicationsTime = Magic ? 0 : tCommBits/HSSGBits/Pins;

        // exceeds capacity of chip -- not viable
        if (!Magic && tTransistorsPerChip > MaxTransistorsPerChip)
                continue;

        // Compute edge size from 3D volume or cooling calculations, whichever is larger
        double tEffectiveEdge = max(Edge3D, tCubeRootChips*CooledPkgEdge);

        // Fraction of chip area occupied, rest will be left empty
        Test.FractionChipOccupancy = tTransistorsPerChip/MaxTransistorsPerChip;
```

```
                double c1 = tEffectiveEdge, c2 = 2*sqrt(3.)/PropagationVelocity, c3 = MeshUpdateTime, c4 =
Test.CommunicationsTime, c5 = SystemEnergy, c6 = SystemStaticPower, k = CoolantPower;

                double tInflateLow = 1, tInflateHigh = 1e9;
                for (int j = 0; j < 20; j++) {

                        // Binary search for the best inflation factor by geometric interpolation
                        Test.Inflate = sqrt(tInflateLow * tInflateHigh);
                        double i = sqrt(tInflateLow * tInflateHigh);

                        Test.LEdge = tEffectiveEdge*Test.Inflate;

                        c1 = tEffectiveEdge;
                        double y1 = c1*i;

                        double itSynchronizeTime = 2*sqrt(3.)*Test.LEdge/PropagationVelocity;
                        Test.TimeStep = max(MeshUpdateTime + itSynchronizeTime, Test.CommunicationsTime);

                        c2 = 2*sqrt(3.)/PropagationVelocity;
                        c3 = MeshUpdateTime;
                        c4 = Test.CommunicationsTime;
                        double y2 = c2*y1;
                        double y3 = max(c3+y2, c4);

                        double itSystemDynamicPower = SystemEnergy/Test.TimeStep;

                        c5 = SystemEnergy;
                        double y4 = c5/y3;

                        Test.SystemPower = itSystemDynamicPower + SystemStaticPower;

                        c6 = SystemStaticPower;
                        double y5 = c6 + y4;

                        Test.FacePowerDensity = Test.SystemPower/6/Test.LEdge/Test.LEdge;
```

```
                double y6 = y5/y1/y1/6;


                FPASSERT((c6 + c5/max(c3+(c2*c1*i), c4))/(c1*c1*i*i*6), Test.FacePowerDensity);

                if (Test.FacePowerDensity < CoolantPower)
                        tInflateHigh = Test.Inflate;
                else
                        tInflateLow = Test.Inflate;
        }

        if (Test.Inflate > 1.01) {

                double i = Test.Inflate;
                double test = (c6 + c5/max(c3+(c2*c1*i), c4))/(c1*c1*i*i*6);
                FPASSERT((c6 + c5/max(c3+(c2*c1*i), c4))/(c1*c1*i*i*6), CoolantPower);
        }

        // overheats chip -- not viable
        if (!Magic && Test.SystemPower/Test.Chips > CooledMaxPower)    // parameter: max power per
chip
                continue;

        // yearly electricity cost at 15 cents/KWH
        double YearlyPowerCost = Test.SystemPower/WordsPerMemory*24*365*KWHCost;

        // cost of hardware at $1000/chip
        double HardwareCost = Test.Chips*CostPerChip;

        // square feet of building required assuming 8' high with 50% aisle
        double LEdgeFeet = LEdge/MetersPerFoot;
        double SystemVolumeCubicFeet = LEdgeFeet*LEdgeFeet*LEdgeFeet;
        double SquareFeetFloor = SystemVolumeCubicFeet/8*2;
```

```
                // yearly rental cost @$12/sq ft/year
                double YearlyRental = SquareFeetFloor*RentalCostSquareFootPerYear;

                // yearly cost -- assumes 30% of hardware cost per year -- like a 5 year lifespan with interest
                Test.YearlyCost = YearlyPowerCost + HardwareCost*DepreciationFactor + YearlyRental;

                Test.FLOPS = n*n*n/Test.TimeStep*GrindFLOPS;
                Test.Quality = Test.FLOPS/Test.YearlyCost/1e15*1000;

                // save best design
                if (Quality < Test.Quality)
                        *this = Test;
        }
}
```

References

- [Coates 00] W. S. Coates, J. K. Lexau, I. W. Jones, S. M. Fairbanks, I. E. Sutherland, “FLEETZero : An Asynchronous Switch Fabric Chip Experiment,” Proceedings of the Seventh International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2001), 11-14 March 2001, Salt Lake City, UT, USA, pp. 173-182.

- [DeBenedictis 03] Erik P. DeBenedictis, “A Network Architecture for Petaflops Supercomputers,” Sandia National Laboratories SAND report SAND2003-2954, August 2003

- [Drexler 92] Drexler, K. Eric., “Nanosystems: Molecular Machinery, Manufacturing, and Computation,” John Wiley & Sons, Inc., 1992.

- [Feynman 82] Richard P. Feynman, “Simulating Physics with Computers,” International Journal of Theoretical Physics, Vol. 21. Nos. 6/7, 1982.

- [Frank 97] Michael P. Frank, “Ultimate theoretical models of Nanocomputers,” Nanotechnology 9 (1998) 162-176.

- [Han 02] Jie Hand and Pieter Jonker, “A System Architecture Solution for Unreliable Nanoelectronic Devices,” IEEE Transactions on Nanotechnology Vol. 1, No. 4 (2002) 201-208.

- [ITRS 02] International Technology Roadmap for Semiconductors, http://public.itrs.net. All figures used in this report refer to the ITRS 2002 update.

- [Kim 01] Kim, S., Zeisler, C., Papaefthymiou, M., "A True Single-Phase 8-bit Adiabatic Multiplier," in proceedings of the 2001 Design Automation Conference, pp. 758-763.

- [Kung 82] Kung, H. T. "Why Systolic Architectures?," Computer, vol. 15, no. 1, pp. 37-46, 1982.

- [Laundauer 61] Landauer, R., “Irreversibility and heat generation in the computing process,” IBM J. Res. Dev. 5, 183-191, 1961.

- [LLNL 99] Lawrence Livermore National Laboratory, “MICROCHANNEL COOLING,” http://www.llnl.gov/IPandC/technology/profile/lasers/MicrochannelCooling/index.php.

- [Sunaga 96] Sunaga, T., Peter M. Kogge, et al, "A Processor In Memory Chip for Massively Parallel Embedded Applicatiions," IEEE J. of Solid State Circuits, Oct. 1996, pp. 1556-1559.

- [Tomkins 01] James Tomkins, private communications, 2001.

- [Vieri 99] Vieri, Carlin, “Reversible Computer Engineering and Architecture,” Ph. D. Thesis, Massachusetts Institute of Technology 1999.

- [Vitanyi 88] Vitanyi, P. M. B., “Locality, communications, and interconnect length in multicomputers,” SIAM J. on Computing, 17, 4 (1988), 659-672.

- [von Neumann 56] von Neumann, J., “Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components,” in C. E. Shannon and J. McCarthy, Eds. Automata Studies. Princeton: Princeton University Press, pp. 43-98, 1956.

Distribution:

1 MS	9037	J. C. Berry, 8945	
1	9019	S. C. Carpenter, 8945	
1	9012	J. A. Friesen, 8963	
1	9012	S. C. Gray, 8949	
1	9011	B. V. Hess, 8941	
1	9915	M. L. Koszykowski, 8961	
1	9019	B. A. Maxwell, 8945	
1	9012	P. E. Nielan, 8964	
1	9217	S. W. Thomas, 8962	
1	0824	A. C. Ratzel, 9110	
1	0847	H. S. Morgan, 9120	
1	0824	J. L. Moya, 9130	
1	0835	J. M. McGlaun, 9140	
1	0833	B. J. Hunter, 9103	
1	0834	M. R. Prarie, 9112	
1	0555	M. S. Garrett, 9122	
1	0821	L. A. Gritzo, 9132	
1	0835	E. A. Boucheron, 9141	
1	0826	S. N. Kempka, 9113	
1	0893	J. Pott, 9123	
1	0835	K. F. Alvin, 9142	
1	0834	J. E. Johannes, 9114	
1	0847	J. M. Redmond, 9124	
1	1135	S. R. Heffelfinger, 9134	
1	0826	J. D. Zepper, 9143	
1	0825	B. Hassan, 9115	
1	0557	T. J. Baca, 9125	
1	0836	E. S. Hertel, Jr., 9116	
1	0847	R. A. May, 9126	
1	0836	R. O. Griffith, 9117	
1	0847	J. Jung, 9127	
1	0321	P. R. Graham, 9208	
1	0318	J. E. Nelson, 9209	
1	0847	S. A. Mitchell, 9211	
1	0310	M. D. Rintoul, 9212	
1	1110	D. E. Womble, 9214	
1	1111	B. A. Hendrickson, 9215	
1	0310	R. W. Leland, 9220	
1	1110	N. D. Pundit, 9223	
1	1110	D. W. Doerfler, 9224	
1	0847	T. D. Blacker, 9226	
1	0822	P. Heermann, 9227	
1 MS	0818	P. Yarrington, 9230	
1	0819	R. M. Summers, 9231	
1	0820	P. F. Chavez, 9232	
1	0316	S. S. Dosanjh, 9233	
1	0316	J. B. Aidun, 9235	
1	0813	R. M. Cahoon, 9311	
1	0801	F. W. Mason, 9320	
1	0806	C. Jones, 9322	
1	0822	C. Pavlakos, 9326	
1	0807	J. P. Noe, 9328	
1	0805	W.D. Swartz, 9329	
1	0812	M. R. Sjulin, 9330	
1	0813	A. Maese, 9333	
1	0812	M. J. Benson, 9334	
1	0809	G. E. Connor, 9335	
1	0806	L. Stans, 9336	
1	1110	R. B. Brightwell, 9224	
1	1110	R. E. Riesen, 9223	
1	1110	K. D. Underwood, 9223	
10	1110	E. P. DeBenedictis, 9223	
1	0321	W. Camp, 9200	
1	0841	T. Bickel, 9100	
1	9003	K. Washington, 8900	
1	0801	A. Hale, 9300	
1	0139	M. Vahle, 9900	
1	9018	Central Technical Files, 8945-1	
2	0899	Technical Library, 9616	