ClearSpeed™

Strategies for Solving the Heat/Power Problem

ZETTA FLOPS

John L. Gustafson, Ph.D.
Chief Technology Officer, HPC
ClearSpeed Technology, Inc.

Even consumers are seeing the HPC heat issue

76-inch HDTV: 200 watts

Video game with IBM Cell BE processor: **380 watts**, twice what the TV uses!

Part 1: Near-term ideas

- ***Switch technology*** **is only a fraction of the problem… look at memory, communication**
- **Slow down the clock!**
- **Use multiple cores per control thread to minimize programmer pain**
- **Design active components to 70 watts/liter, no more, no less**
- **Embrace new tradeoff between sparse and dense methods**
- **Use higher-order stencils, implicit methods to greatly increase useful flops per data point**
- **Move away from scout threads, speculative parallelism, unless legacy software is your ball-and-chain**

Some 2005-era data (from Bill Dally)

Operation	Energy (130 nm, 1.2 V)
32-bit ALU operation	5 pJ
32-bit register read	10 pJ
Read 32 bits from 8K RAM	50 pJ
Move 32 bits across 10 mm chip	100 pJ
Move 32 bits off chip	1300 to 1900 pJ

- Moving data chip-to-chip is **much** more expensive than the gate activity on chip.
- Some of the burden will be on the **programmer**, not just the hardware engineer, to control this energy use.

The power cost of the clock itself

"It's 25% of the chip power"

—*Sun Microsystems, 2003*

"It's more like 50 to 60% of the chip power."

—*Multigig, 2006*

"Everything will be multi-cycle latency."

—*Thomas Sterling, two days ago*

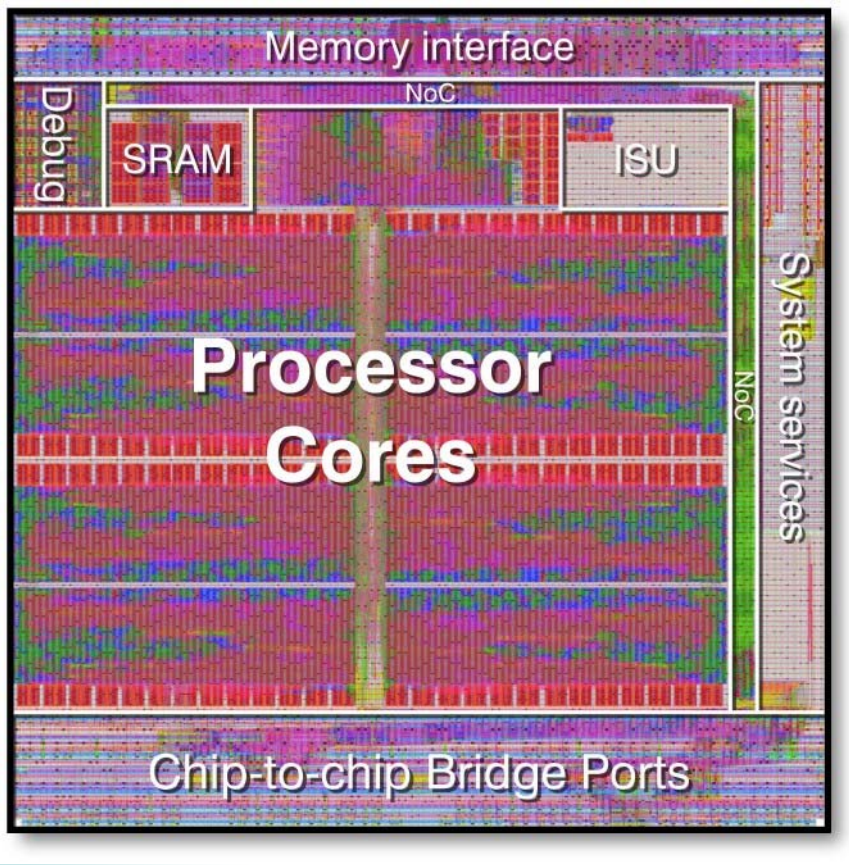
Reduce clock to the linear region or even more drastically. Do more per cycle. Pushing the clock is like revving your engine at a red light.

Take a cue from the human brain… 25 watts, ~1 Hz!

Increase ratio of cores to threads… *carefully*

- **Instruction control takes ~30 times the wattage of a 64-bit multiply-add unit, suggesting that we *overprovision* the multiply-add units**
- **Thinking Machines, MasPar, etc. proved that a ratio of thousands is too much for most things**
- **Current SSE ratio (2:1, 4:1) is OK for general computing, not aggressive enough for HPC**
- **A SIMD ratio of 32 to 256 seems about right for HPC without creating pain for programmers**
- **The C* idea (poly variable type) is about as easy as parallel programming gets, keeps resurfacing (for a good reason)**

Example: 210 MHz, 96 cores, 1 control thread



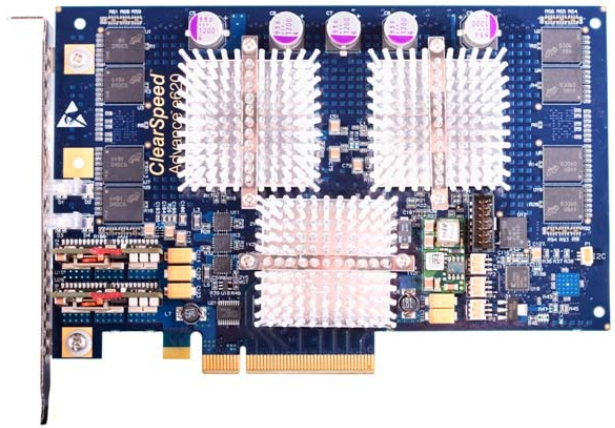
ClearSpeed CSX600

- **Array of 96 Processor Elements; 64-bit and 32-bit floating point**
- **210 MHz… key to low power**
- **47% logic, 53% memory**
  - About 50% of the logic is FPUs
  - Hence around one quarter of the chip is floating point hardware
- **About 1 TB/sec internal bandwidth**
- **Only 128 million transistors, but faster at 64-bit FLOPS than x86 multicore two generations later**
- **Approximately 10 Watts**

Heat leads to *bulk*

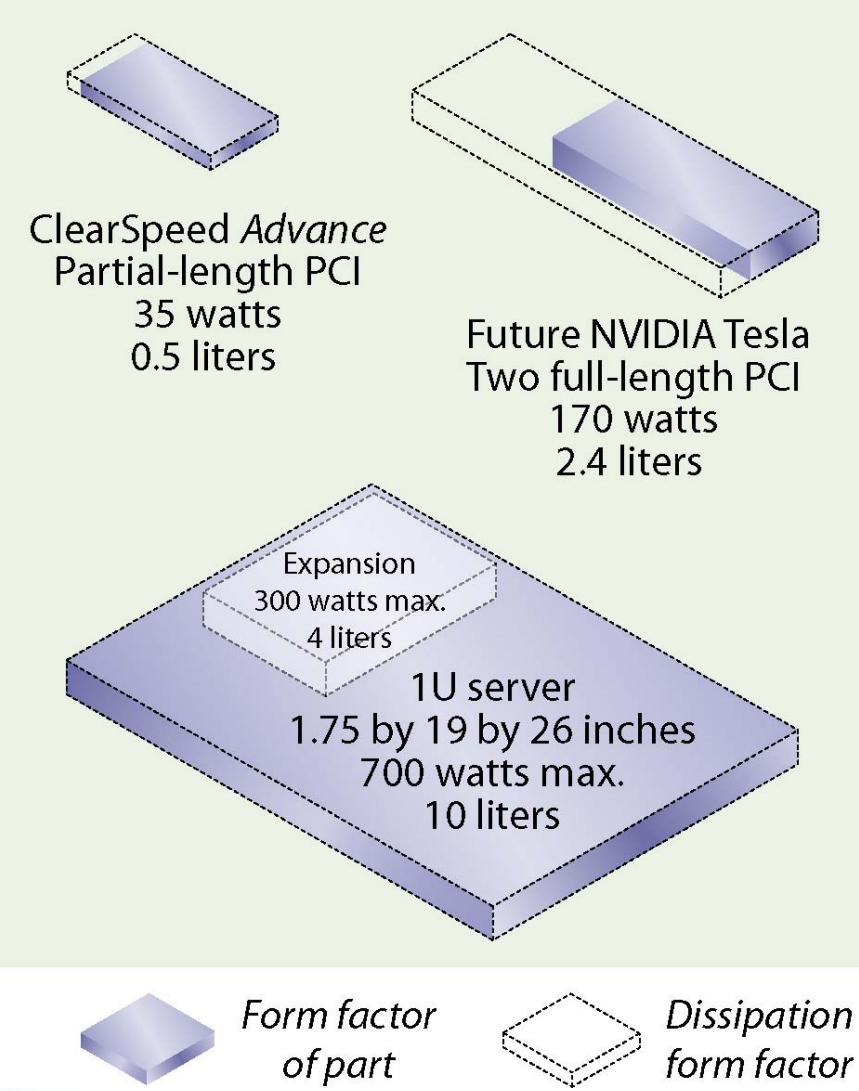
- **Air cooling hits limits at about 70 watts/liter**
  - PCI standard of 25 watts, size is ***0.3 liters***
  - A 1U server might use 1000 watts, volume is ***14 liters***
  - A 42U standard rack might use 40 kilowatts, ***3000 liters***
- **Exceed 70 watts/liter, and temperatures rise above operational limits**

4 inches by 6 inches
0.5 liter in system
35 watts
9 ounces



Current e620 ClearSpeed accelerator

Dissipation volume can exceed actual volume

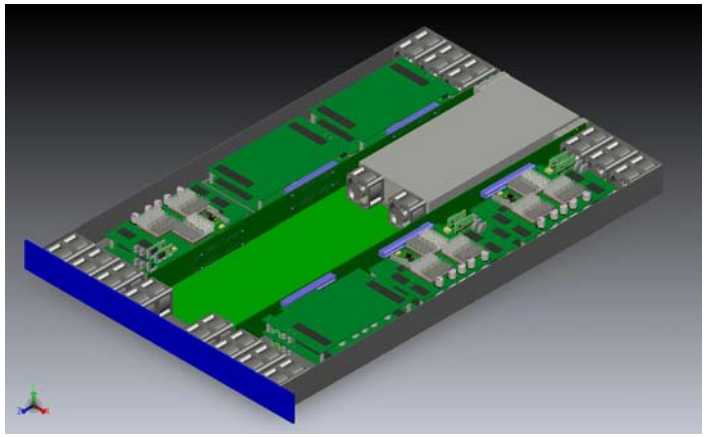


- **To find the *real* volume occupied by a component in liters, divide its wattage by 70**
- **What may seem like a dense, powerful solution might actually *dilute* the GFLOPS per liter because of heat generation.**

New Design Approach Delivers 1 TFLOP in 1U

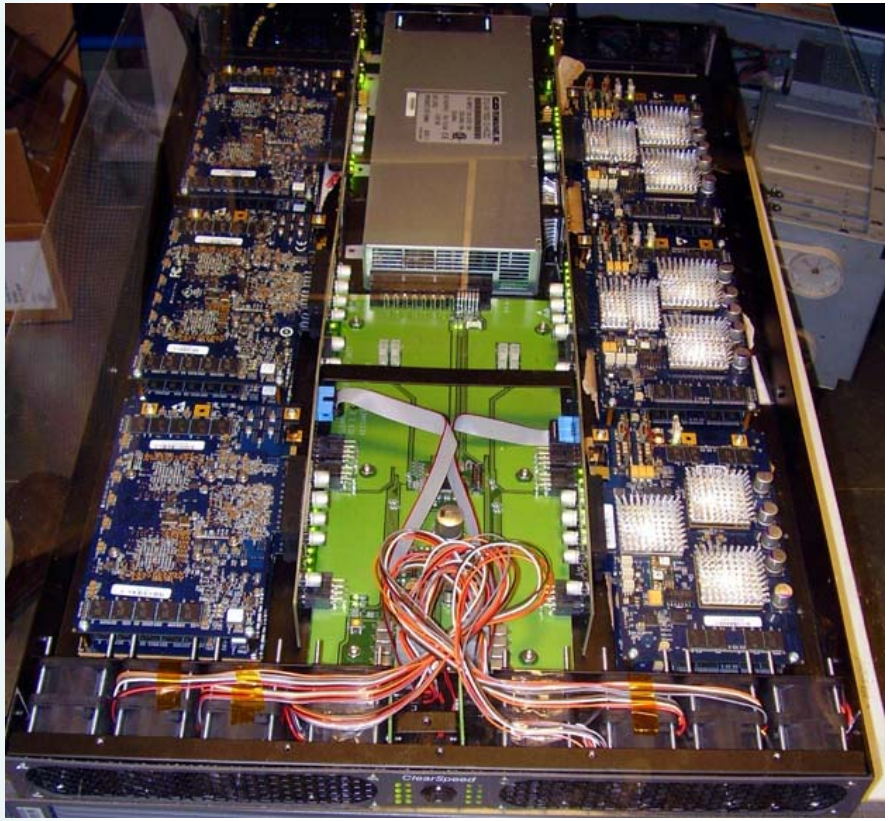
- **1U standard server**
- **Intel 5365 3.0 GHz**
  - 2-socket, quad core
  - 0.096 DP TFLOPS peak
  - Approx. 650 watts
  - Approx. 3.5 TFLOPS peak in a 25 kW rack

- **ClearSpeed Acceleration Server Concept**
  - 24 CSX600 hectacore processors
  - ~1 DP TFLOPS peak
  - Approx. 500 watts
  - Approx. 19 TFLOPS peak in a 25 kW rack
  - 18 standard servers & 18 acceleration servers



ClearSpeed Accelerated TeraScale Server (CATS)

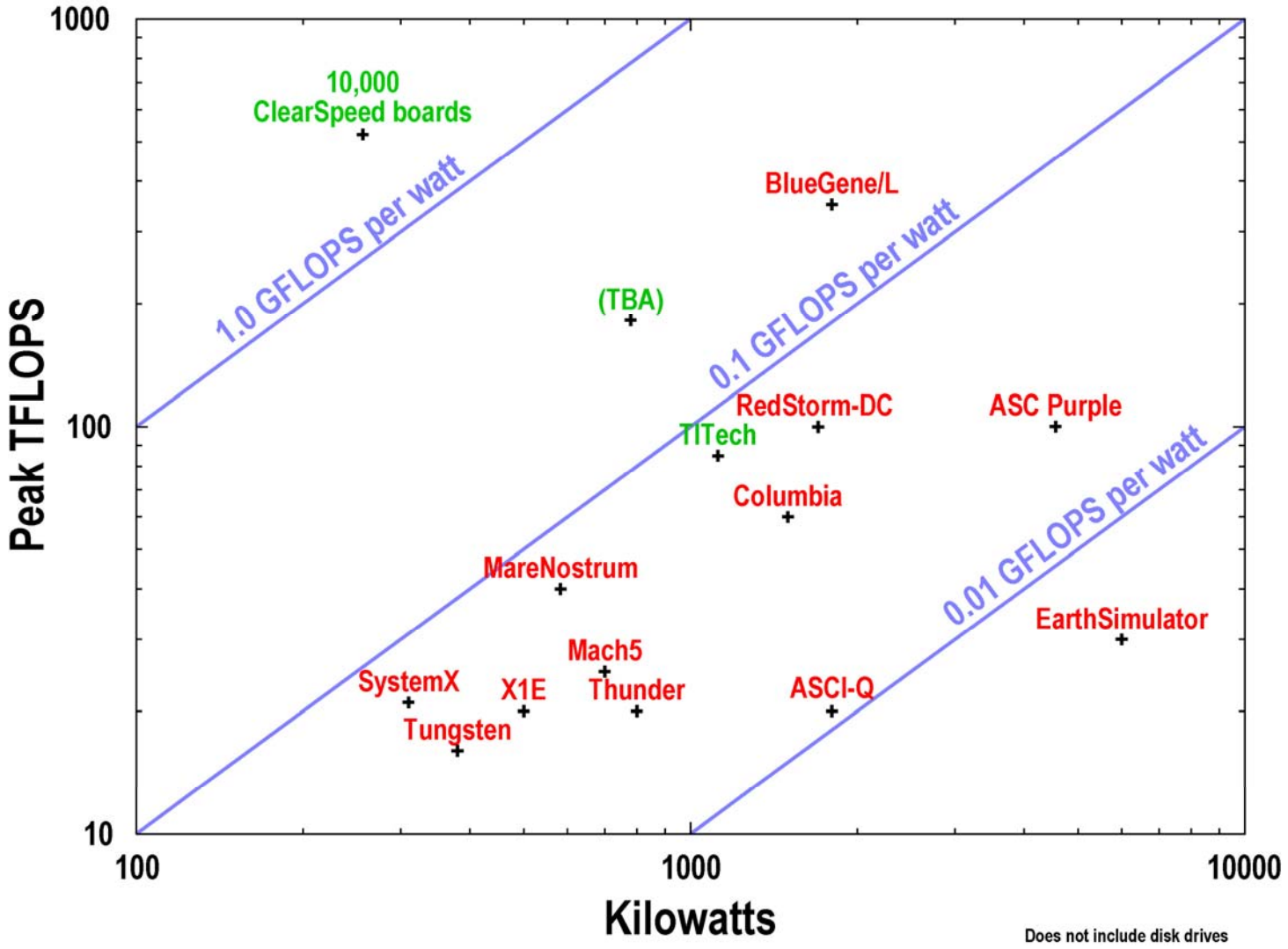
A sneak preview:



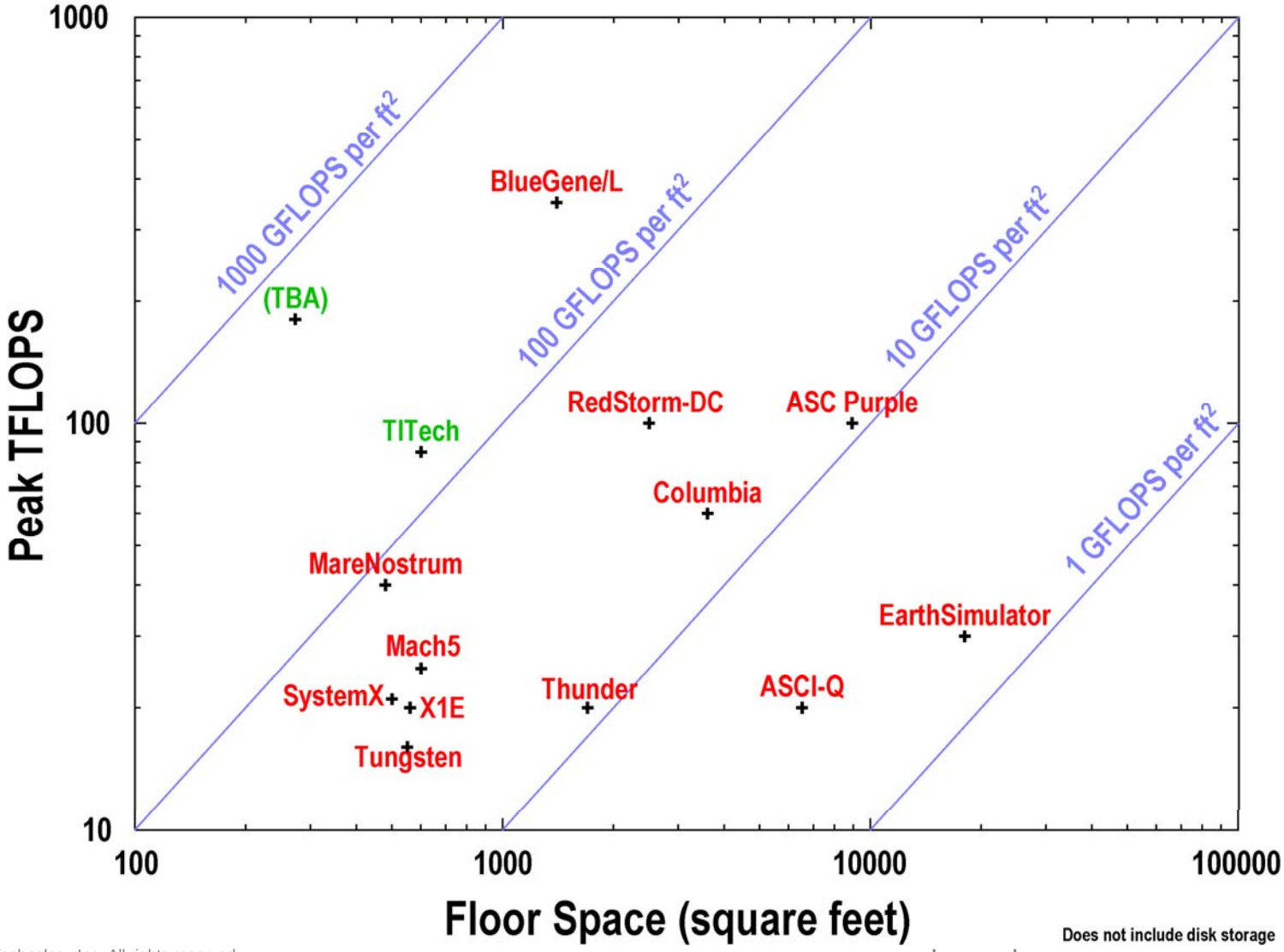
- **64-bit or 32-bit native IEEE FLOPS**
- **Prototype is 0.97 TFLOPS**
- **1U high**
- **Under 1 kW max**
- **Paired with x86 host (separate 1U unit)**
- **Allows 19 TFLOPS peak (64-bit) in a single air-cooled standard rack**

ClearSpeed™

GFLOPS per watt for some capability systems



GFLOPS per ft² for some capability systems

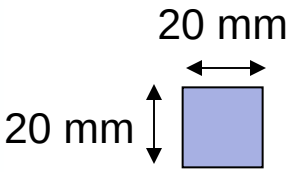


Does not include disk storage

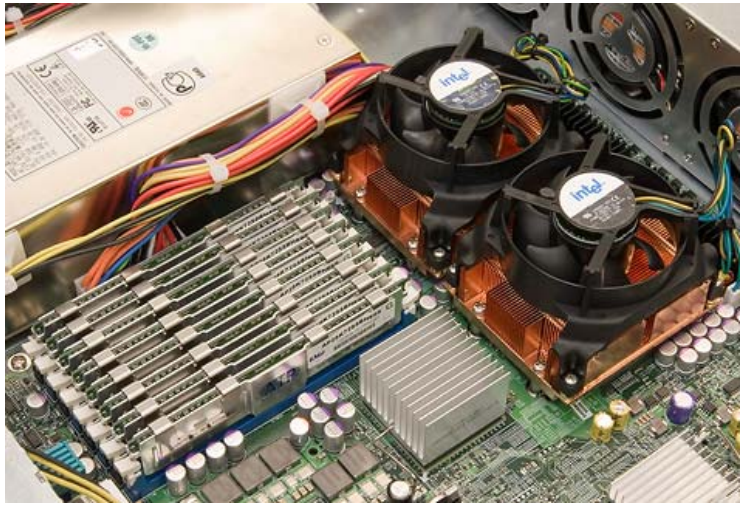
How big are chips, *including the cooling volume?*

- **Itanium and Tigerton: 130 watts, 1.9 liters**
- **Clovertown, 3 GHz: 120 watts, 1.8 liters**

A typical CPU chip…



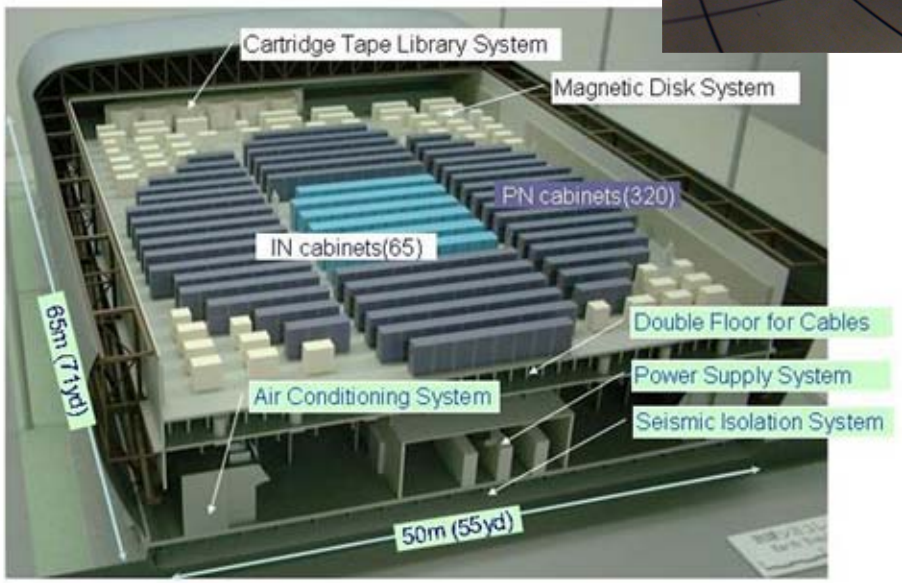
Actual space in system



- **8 Gigabytes of DDR-3 DRAM: 80 watts, 1.1 liters**
- **ClearSpeed CSX600 chip: 10 watts, 0.14 liters**

Example of bulk-limited computing

The CPUs can perform 10 million operations in the time it takes a photon to traverse the Earth Simulator facility.



At 6 megawatts, it doesn't just simulate global warming. It *causes* global warming.

Three “shackles” from the 20th Century

1. **Floating-point arithmetic is hard, especially 64-bit precision, so you must use the algorithm that does the fewest possible operations.**
2. **Memory is expensive, dominating system cost, so you must make sure your program and data fit in the fewest possible bytes.**
3. **Parallel programming is hard because you have to coordinate many events, so you must express your algorithm sequentially.**

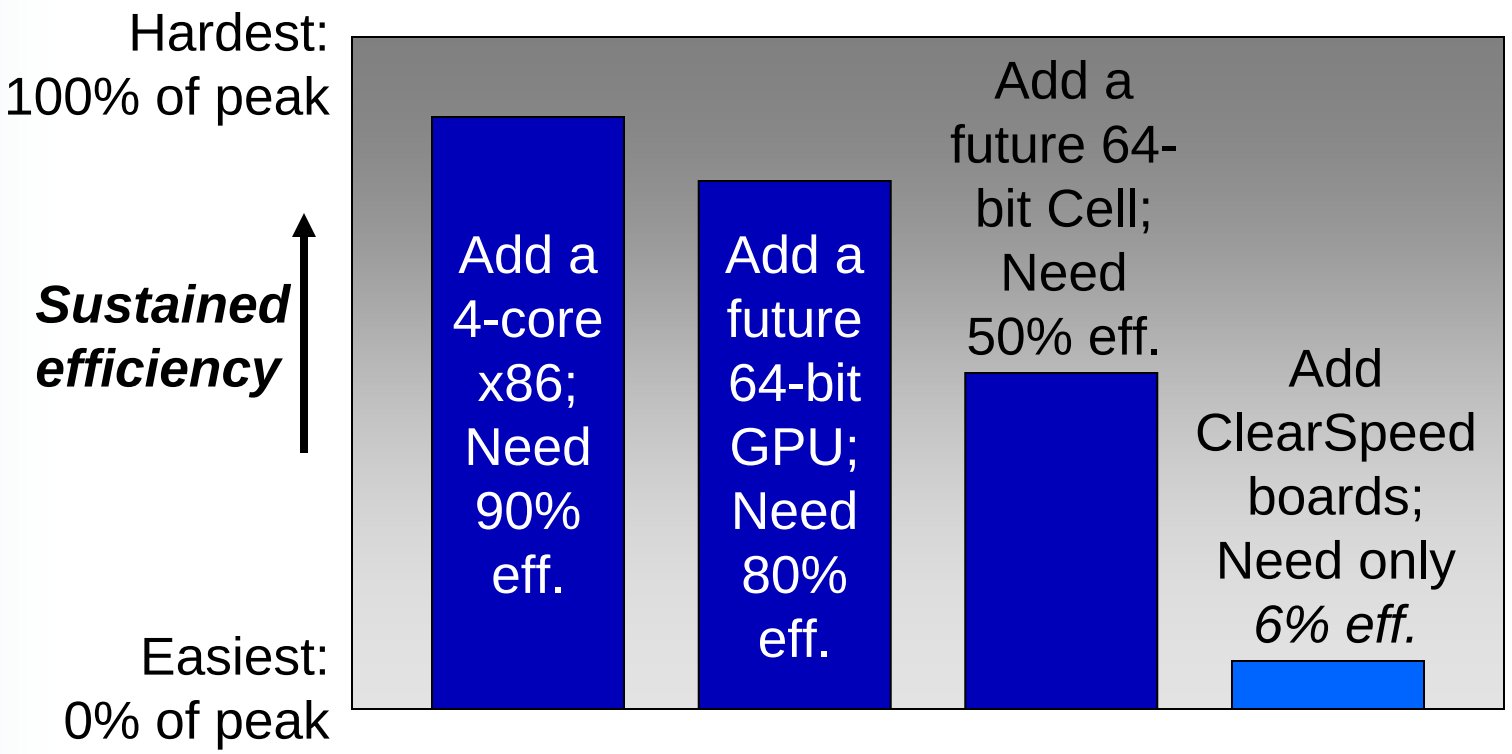

```
LOAD A(I)
ADD B(I)
STORE C(I)
INCREMENT I
IF I < N GO TO
```

The shackles still influence the way we use systems, but we must consciously move away from this mind set.

Letting go of old algorithms: Linear solvers

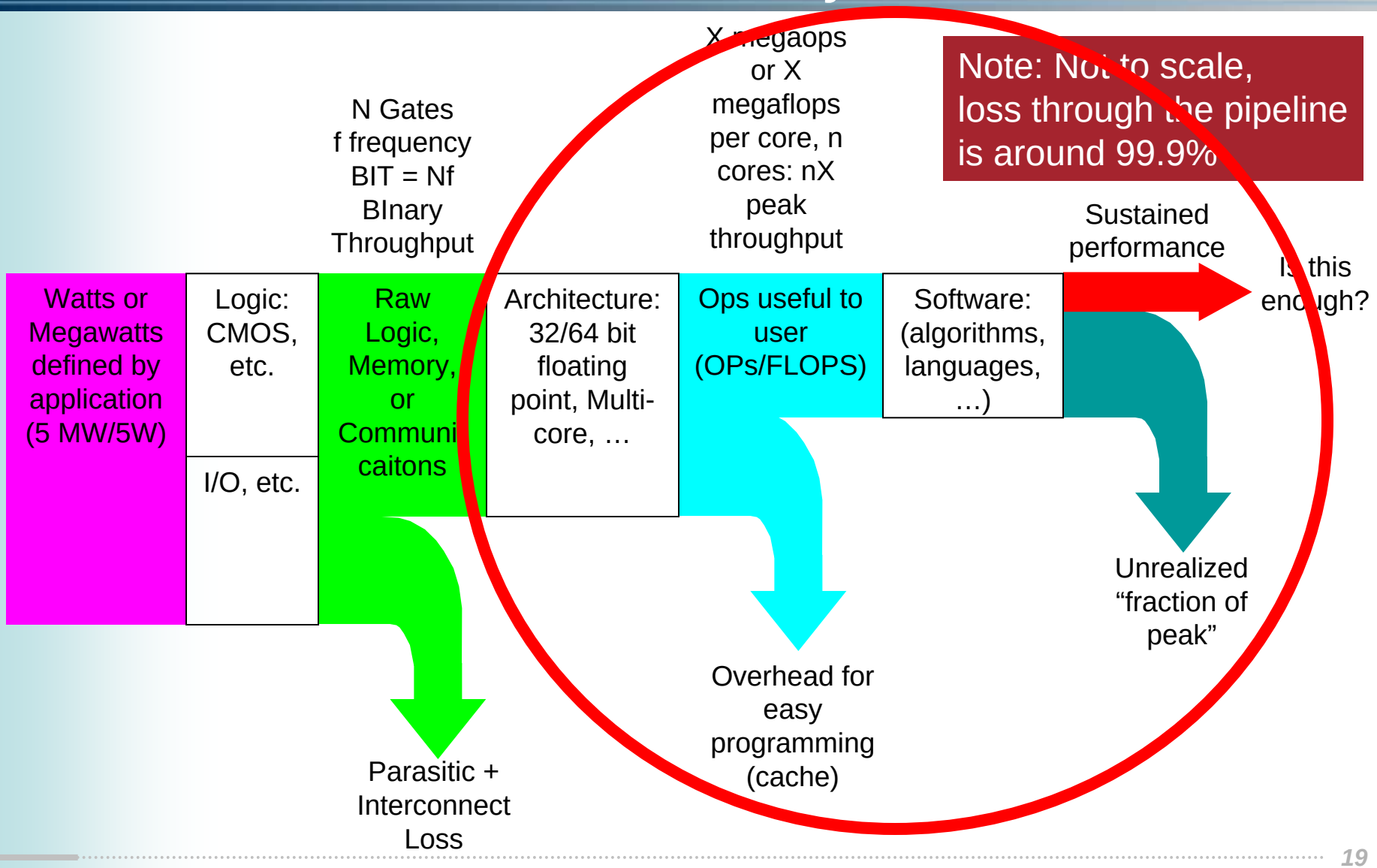
- **Dense methods applied to sparse matrices do more ops but take less time, if sparsity > *x*%**
  - Recent paper in *IEEE Computer* used FPGAs to reduce sparse solver (4000 by 4000, 2% nonzero) from 67.3 sec to 33.8 sec
  - Using a dense method takes less than 3 sec, uses about 128 MB (which costs about $10 lately)
  - Hundreds of hours of effort to create new version of 1970s algorithm leads to 10x slowdown. Economizes all the wrong things!
- **Even coding for symmetric matrices only saves 2x on arithmetic, memory, but increases execution time and programmer effort.**

50 GFLOPS from 250 W is easier at *low efficiency*



Power efficiency translates into ease-of-use by reducing optimization pressure on programmers

Remember Erik's Introductory slide



Part 2: Longer-term ideas

- You optimize what you measure, so measure and report power use to the *library programmers* at a fine grain
- Make memory hierarchy highly visible to library programmer (through tools or modifications to existing languages)
- Need much more care in the use of *numerical precision* (and tools to do that for us)
- Use *asynchronous* design wherever possible

Make memory hierarchy highly visible?

```
Register    :
      width     64 bits
      instances 28
      bandwidth 2.5e9 per sec
      latency   4e-10 sec
Tier1memory =
      width     256 bits
      instances 1530000
      bandwidth 1.25e9 per sec
      latency   3.2e-9 sec
Tier2memory ==  etc.
MassStorage === etc.
. . .
```

Ways to reveal the data motion cost

Show every move, with arithmetic only allowed from registers?

```
A[0]:B[2]+1.4
A[1]:B[3]
C=A[2 instances]
```

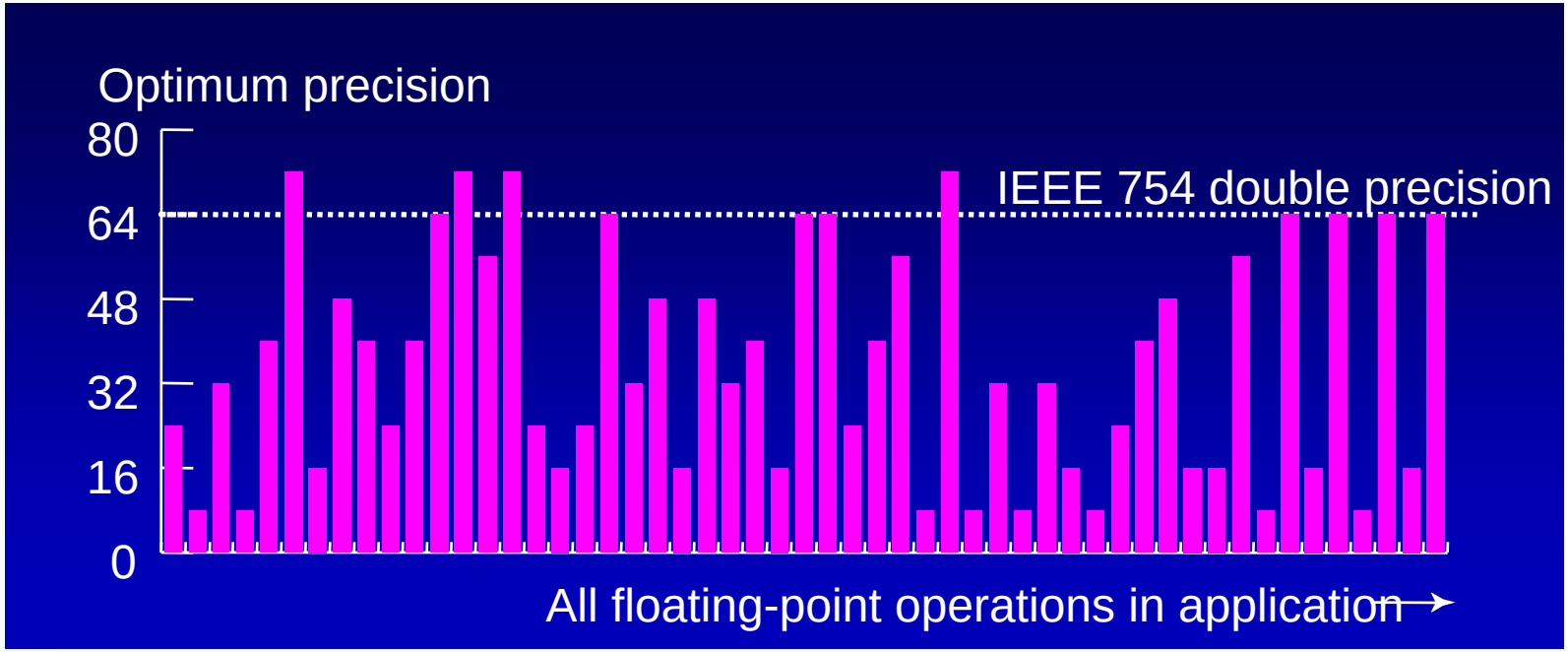
Or, flank the arithmetic with the notation for each tier?

```
YYY[0 to 255] === X[0 to 255] :*= ZZ[0 to 255]
```

Hard work you do *once*… but then it ports through decades by altering the values. You optimize what you can see, so we need to see the real costs now that operations like + – * / are cheap.

We may need to rethink 64-bit flops…

- **Every operation has an optimum number of bits of accuracy**
  - Using too few gives unacceptable errors
  - Using too many wastes memory, bandwidth, **joules**, dollars.
- **It is unlikely that a code uses *just the right amount* of precision needed.**



How do HPC programmers pick FP precision?

- Assume 64-bit is plenty, and use it everywhere.
- Use what is imposed by hardware (word size).
- Try two precisions; if answers agree, use the less precise one, otherwise use the more precise one.
- Compare computed answer for special cases where an analytic answer is known.
- Compare computed answer with physical experiment (rare).
- Perform careful analysis (very rare).

Can a tool estimate joules, W, $, min. precision?

64-bit op
42 bits needed
10 pJ
$1x10^{-15}$

64-bit op
39 bits needed
12 pJ
$1x10^{-15}$

```
a(i,j) -= a(i,k) * a(k,j)
```

Tier 0 read/write
28 pJ
$2x10^{-15}$

Tier 1 read
50 pJ
$8x10^{-14}$

Tier 2 read
1900 pJ
$5x10^{-13}$

Cost and electrical power and precision are almost as important as timing… why not develop analysis tools for them? You can only optimize what you can measure.

Finer-grained use of asynchronous design

- **Even better than slowing down the clock: *get rid of it.***
- **Async operation allows each part of a system to run as fast as it can.**
- **We have always had async at some global level (think of disk drives, network), and trend is to go finer and finer**
- **Main drawbacks**
  - Difficult to design and debug
  - Lack of good EDA tools
- **Advantages**
  - Higher performance
  - Lower active power consumption

Summary

- **The biggest barrier to exaflops and zettaflops is the heat/power problem. Transistors may be cheap, but the energy they dissipate is not.**
- **Heat/power is not all in switching hardware; most of it is wattage for communication and memory. And clock switching is increasingly wasteful.**
- **In the long term, application programmers can help just as much as hardware engineers, by being less sloppy with memory use and precision demands.**